Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A Study of Dynamically Routed Capsule Networks in Generative Adversarial Modeling

Semester Thesis

Andrawes Al Bahou

Advisors: Dr. Zhiwu Huang, Dr. Paudel Danda Pani
Supervisor: Prof. Dr. Luc van Gool
Computer Vision Laboratory
Department of Information Technology and Electrical Engineering

February 23, 2018

Abstract

Traditional Convolutional Neural Networks (CNNs) have a few shortcomings. In particular, they suffer from a difficulty in generalizing a learned representation of an object when shown in a different viewpoint or in a new pose (given by a different rotation, scaling or translation). To learn this invariance, they must be trained on many differing viewpoints, which is however non-trivial to achieve in practice.

In order to address these pitfalls, Hinton et al. recently make an initial attempt to propose an alternative baptized Capsule Networks using dynamic routing. Essentially capsules are a group of neurons outputting an activity vector, whose length is to represent the probability that the entity exits and its orientation is to represent the instantiation parameters (such as location, scale, rotation, skewness, etc). In the context of their proposed dynamic routing system, the capsules for low-level features are designed to predict the outputs of higher level capsules corresponding to more abstract features. When low-level capsules agree on the predicted outcome of a particular higher-level capsule, then their outputs are routed to this higher level-capsule.

In this work we explore the potential usefulness of Capsule Networks with dynamic routing under different scenarios for generative modeling. Particularly we study their performance as discriminators in the domain of Generative adversarial networks (GANs) by designing multiple network architectures for standard datasets. From the study, we discover that while their use is indeed feasible, their performance is poor on datasets with rich image semantics. In addition, we devise a novel ”inverted” Capsule Network as a generative model for GANs, and uncover inherent instability when used to generate images. Finally, we explore the reasons behind this and suggest a possible, (but expensive) way of solving it.

Acknowledgements

I would like to express my gratitude for the help received from all those who took part in bringing this project into its current shape.

First, I would like to thank my advisors Zhiwu Huang and Paudel Danda Pani for their continuous support in all matters conceptual and technical. They have provided guidance for this project, and have supported my initiatives in shaping this project to fit a topic of interest to me.

Secondly, I would like to address my thanks to Prof. Dr. Luc Van Gool, who provided the academic environment for this project to take place.

I owe much gratitude to Prof. Dr. Geoffrey Hinton and Sara Sabour for their feedback and deep insights which have helped explain and solve many results observed over the course of this work.

Finally the encouragement of my family and close friends kept me motivated in the face of the many inevitable challenges. I thank them for their lifelong support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation of this Work

In recent tlimes, Convolutional Neural Networks (CNNs) have become the default go-to algorithm for a very diverse range of computer vision tasks, including classification, segmentation and feature detection on both image and video data. They are increasingly used as a bulding block in more complex tasks such as compression, style transfer, or even forging images and video. In some tasks, their accuracy has even been able to surpass humans [8].

Despite their enormous success, CNNs suffer from several drawbacks. To understand these, Convolutional Neural Networks must be seen as a hierarchy of feature detectors. These feature detectors are essentially replicated spatially by the sliding of the convolutional window. Thus the higher up in the hierarchy a feature detector is, the larger is its receptive field. With this said, an important problem arises when we introduce pooling layers which subsample adjacent features produced by a feature detector. This appears to be a source of loss of important information, especially about the relationship between "parts" and "wholes". This effect is more and more severe as we move higher up in the hierarchy of feature detectors.

When changing the pose of an object or the viewpoint from which we observe it, much of the variations of the pixels can be explained by simple linear transformations applied onto this object, such as rotation, scaling, or shearing. Traditional CNNs likely fail to make use of this fact, and therefore lack the generalization capability when extrapolating to new viewpoints. There is evidence that the human brain, unlike CNNs, does indeed perform these linear operations.

Consider the this example. We want to decide whether the following image depicts the letter "R" or "Я".

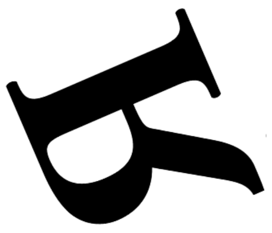

Figure 1.1: Rotated Letter

Making a decision takes a few moments, since the brain has to rotate the image until it becomes vertical, and then decide that it is letter “Я”.

In short CNNs suffer from several limitations, only few of which have been shown here. Recent research by Hinton et al. [13] attempts to solve these shortcomings with a new architecture for Neural Networks called Capsule Networks. They show superior accuracy with their novel architecture when the network is confronted with new viewpoints (which it has not been trained on). They argue the model they build carries deeper understanding of the effects of linear transformations on how objects are seen in pixel space.

With these benefits of Capsule Networks, this work sets out to understand if Capsule Networks can be used in Generative Adversarial modeling, where they could bring in their ability to learn additional insights in order to create better Generative Adversarial Networks (GAN).

In the context of GANs, our work consisted in modifying CapsNets to be used as both a discriminator or as a generator network. We therefore create generative and discriminative architectures from CapsNets and evaluate their behavior. In doing so, we have uncovered some very useful insights.

1.2 Thesis Organization

This report is structured in the following manner: First, we will lay down the theoretical foundations underpinning Capsule Networks and dynamic routing in chapter 2 . Next, we will explain and justify the models we have built in chapter 3. Following this, we will present the experiments we have carried out to test our hypotheses as well as the results of those experiments, in chapter 4. Finally, in chaper 5 we shall discuss the results and attempt to answer the main questions driving this work: ***Can Capsule Networks be applied to Generative Adversarial Networks?*** and ***Can generative adversarial networks benefit from capsule networks?***

Chapter 2

Related Work

As it had been explained in chapter 1, this work builds on top of the recent theory behind Capsule Networks as well as the significantly more established theory underpinning Generative Adversarial Networks. Therefore, this section presents the intuition necessary for understanding this work's contributions and results. First, the idea of Capsule Networks is explained. This is then followed by a simple overview of Generative Adversarial Networks.

2.1 Capsule Networks and Routing-by-Agreement

The groundwork for Capsule Networks has come from a series of research endeavors by Hinton et al. The first [2] introduces the concept of capsules as a group of neurons carrying out a specific task. The second [13], entitled "Dynamic Routing Between Capsules" demonstrates a method for interconnecting layers in a hierarchy of capsules through an agreement filtering mechanism. The third work [4], "Matrix Capsules with EM Routing" improves upon the algorithm presented in [13], and overcomes many of its pitfalls. In this chapter, we look at the intuition behind Capsule Networks and Dynamic Routing. In the next chapter 3, we will present a more mathematical description of Capsule Networks.

2.1.1 The Idea Behind Capsules

The authors of [2] introduce the concept of *capsules*: It is simply a grouping of neurons. The output of a *capsule* is a vector, called *activity vector*, unlike neurons whose output is a scalar value. The elements of this vector output are called *instantiation parameters*. Each *instantiation parameter* encodes information about the pose of the feature which the capsule specializes in detecting. The term "Pose" here should be taken in a more general sense, as it may include information about location, scale, rotation, skewness, shear, texture, etc. The norm of the activity vector (i.e the length of the vector) represents the probability that a feature is detected by its associated capsule. We will later see why this might be a problem.
When capsules are stacked made into a network, as is the case with CNNs, lower level capsules attempt to extract information about low-level features, or in other terms "parts" of a larger visual entity. Similarly, capsules higher up in the hierarchy attempt to extract information about entire entities, or "wholes". The advantage of vector outputs is that they are able to encode the relative *spatial* relationship between parts and wholes, or more simply their poses relative to one another. To understand this, the relative spatial relationship between a part and a whole can be expressed as:

$$\mathbf{T}_B = \mathbf{T}_{AB}\mathbf{T}_A \tag{2.1}$$

where $\mathbf{T}_A$ is the pose output of low-level capsule A, $\mathbf{T}_{AB}$ is the part-whole transformation matrix, and $\mathbf{T}_B$ is the pose output of high-level capsule B. Intuitively, $\mathbf{T}_{AB}$ represents the "canonical" relationship between the part detected by A, and the whole detected by B. What this means, is that if the transformation matrix $\mathbf{T}_{AB}$ and $\mathbf{T}_A$ are known, then we can predict the pose $\mathbf{T}_B$ of the high-level entity detected by capsule B. Note already that the part-whole matrix $\mathbf{T}_{AB}$ is not known in advance; it is learned during the training procedure.

2.1.2 An Example of Capsules in Action

The following illustrations present a contrived low-dimensional example into what is happening. Suppose the task is to classify images of boats and houses like the ones below.

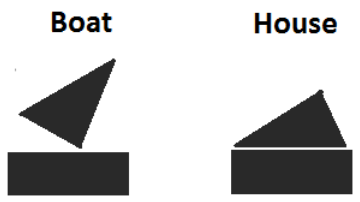

Figure 2.1: Images of house and sailboat (from Aurélien Géron [5])

Suppose additionally that we have capsules for the low-level features, in this case the triangle and rectangle. The output of the rectangle and triangle capsules as in Fig. 2.2 is a vector encoding simply one instantiation parameter, the rotation.

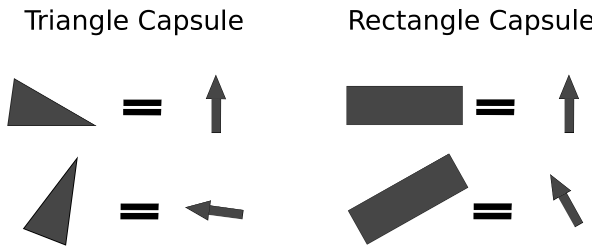

Figure 2.2: Outputs of the triangle and rectangle capsules with respect to input features varying in rotation

When faced with an input image as the following in Fig. 2.3, the low-level Capsules for Rectangles and triangles activate.

Each low-level capsule can now cast a prediction about what *every* high-level capsule's outcome could be given its current instantiation parameters. Fig. 2.4 illustrates the two possible outcomes.

The two capsules cast different predictions on the outcome of the house capsule, however, they agree on the outcome of the boat capsule. Therefore the boat capsule is activated. This essentially is *Routing-by-Agreement*.

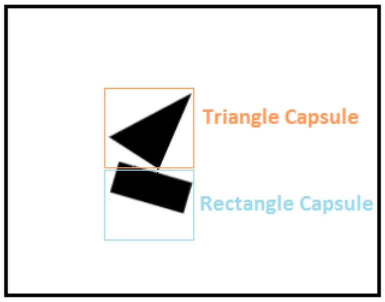

Figure 2.3: Activation of the Rectangle and Triangle Capsules

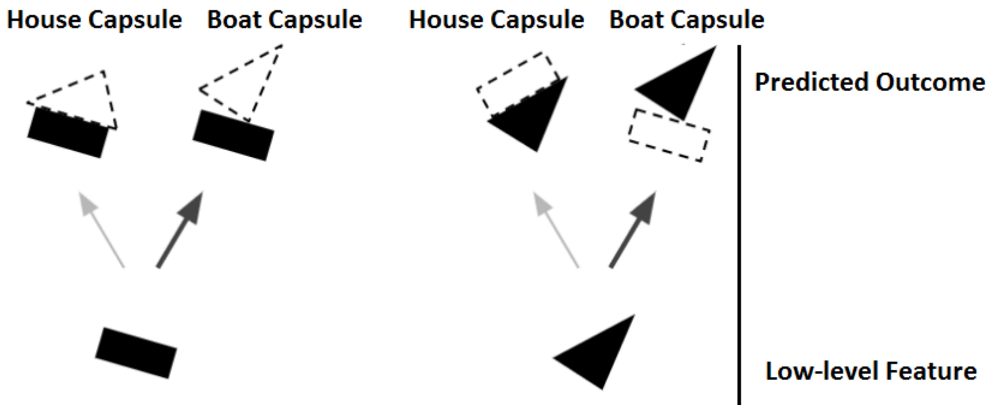

Figure 2.4: Prediction of the presence of *whole* object based on the present of object *part*. Image from Aurélien Géron [5]

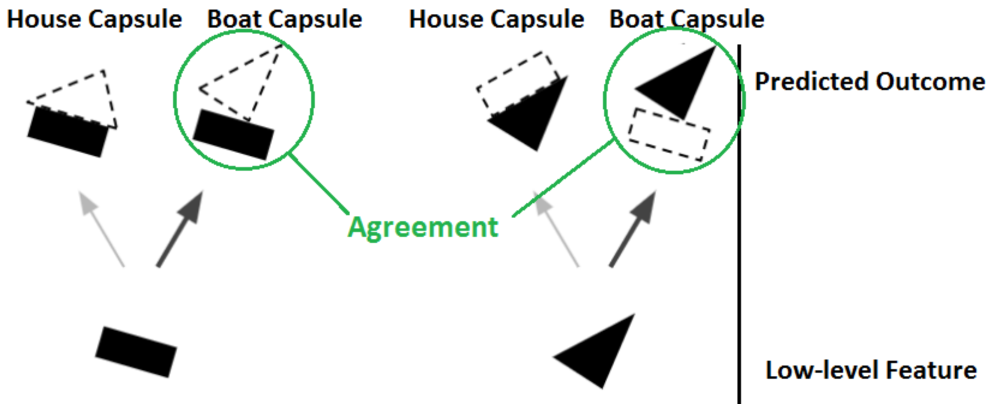

Figure 2.5: Agreement between 2 low-level capsules that the outcome based on the observed parts should be a boat, not a house. Image from Aurélien Géron [5]

In the next chapter, we will go into more detail regarding the underlying mathematics and the full description of the routing algorithm.

2.2 Generative Adversarial Networks

2.2.1 A Brief Overview of Generative Adversarial Networks

Generative Adversarial Nets [6] introduced in 2014 have been a very successful type of neural network architecture. Essentially, they consist of 2 networks: A Generator, and a Discriminator. The generator is a function $G(z)$ which attempts to approximate the distribution of an input dataset. It takes a random input $z \sim P_z(z)$, and maps to it a high-dimensional output which should appear to be sampled from the distribution of the input dataset. Its training is supervised by a second network called a Discriminator. A discriminator is a function $D(x)$ which maps an input $x \sim P_{data}(x)$ to a probability value. This probability indicated the likelihood that the input belongs to the dataset whose distribution is to be approximated.

In the case of image datasets, Generators attempt to create images which look similar to the images of a dataset. The output of the Generator is labeled as “Fake”. Then the Discriminator is fed one of 2 inputs: either a real image taken from the dataset and labeled as “Real”, or an image produced by the Generator. The output of the discriminator is used to compute the loss which the Generator incurs. The discriminator progressively learns to detect real images from false ones, whereas the Generator learns how to create more and more “Realistic” images to “trick the discriminator”. Eventually, the Generator becomes good, so the discriminator may be discarded.

Generator and Discriminator functions we have talked about can, and usually are Convolutional Neural Networks. More precisely the Generator is usually a partially-strided convolutional Neural Network. A partially strided convolution, also falsely referred to as “Deconvolution” is the operation which reverses the effect of convolution, and therefore takes from high-level features to low level features.

This can be seen from the perspective of Game Theory as a zero-sum game where the loss of one agent is the profit of the other. Thus the loss function for a GAN as seen in equation 2.2 is written as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim P_{data}(x)}[log(D(x))] + \mathbb{E}_{z \sim P_z(z)}[log(1 - D(G(z)))] \tag{2.2}$$

where the log terms come from the cross-entropy loss. $P_{data}(x)$ is the distribution of the dataset.

There are various uses to Generative Adversarial Networks. They have been used to generate visual Art or new musical pieces. They have also been useful in image super-resolution [11]. Recent works have used them to transform 2D images and sketches into 3D models [3].

2.2.2 Stability of Generative Adversarial Networks

One must know that GANs are particularly difficult to train, and suffer from instability. Instability here is taken in the context of control theory, where a system fails to converge. It may diverge, oscillate, or enter a so-called limit cycle. GANs are not guaranteed to converge unless some constraints are applied.

The authors of [12], present a framework, commonly referred to as “DCGAN” for Deep Convolutional Generative Adversarial Networks. In their work, they introduce a set of rules which help GANs converge under a wide spectrum of possible configurations.

One of the most prominent works in the development of GANs is the work by Bottou et al. [1], called "Wasserstein GAN". The essence of their work is the use of the Wasserstein distance in the discriminator loss function. This function is shown to have better gradient properties allowing for more stable convergence.

This work is improved upon by the authors of [7] in their work "Improved Training of Wasserstein GANs". They get rid of a weight clipping operation during training, and reintroduce it as a soft constraint in the discriminator loss. This leads, among other things, to improved stability. "Improved Training of Wasserstein GANs" could be considered as the state of the art in GANs.

Chapter 3

Materials and Methods

3.1 Material

This section presents the software and theoretical framework used to carry the experiments which will be detailed in the next chapter 4.

3.1.1 Software Environment

All code is written in Python 2.7, using Tensorflow 1.4.1, CUDA 8.0. Experiments are run on the BIWI cluster computers.

3.1.2 Datasets

We work notably with three datasets.

- **MNIST**: MNIST [10], is a dataset of 70000 grayscale images of handwritten digits, contained in images of dimensions 28x28.
- **CIFAR-10**: CIFAR-10 [9], is a dataset of 60000 colored images of dimensions 32x32. These images shocase objects and animals of 10 different categories (cat, dog, airplane, automobile, etc.).
- **SG-CIFAR-10**: This is a version of CIFAR-10 which we have created where the images are cropped down to images of dimension 28x28. These images are also all converted to grayscale. SG stands for Small Grayscale.

3.1.3 External Code and Frameworks

Framework for Developing GANs

The authors of "Improved Training of Wasserstein GANs" [7] have published the code allowing for their reported results to be reproduced. In our work, we make use of this code as a Framework for developing GANs, where we essentially adapt the Generator or Discriminator architectures. Their code can be found here: `https://github.com/igul222/improved_wgan_training`.

Library functions for Dynamic Routing

We adapt some functions found at `https://github.com/naturomics/CapsNet-Tensorflow` in order to implement the Dynamic Routing by agreement procedure.

3.2 Understanding the Dynamic Routing Algorithm

This project takes inspiration from the principles of Capsule Networks and Dynamic Routing [[2], [13]]. The intuition behind them was presented in the previous chapter 2. A deeper look is now taken into the intricacies of Capsule Networks in order to explain the Dynamic Routing Algorithm.

In chapter 2, we showed that Capsules produce a vector output encoding some *instantiation parameters* of the feature they are trained to detect. We also explained that the magnitude of the vector outputs represent the activation probability of a capsule. Finally we showed that computing the output of a capsule j was a type of clustering of the prediction vectors cast from lower-level capsules. The cluster center, i.e the output of the higher-level capsule is iteratively computed using the dynamic routing algorithm. We will demonstrate how the Dynamic Routing algorithm is constructed.

Let $\mathbf{s}_i$ be the output vector of capsule i. The first step is to resize this vector, such that its magnitude is between 0 and 1. This is analogous to a non-linear activation function such as $tanh$ or the sigmoid function. Thus we define the *squash* function:

$$\mathbf{v}_i = \frac{||\mathbf{s}_i||^2}{1 + ||\mathbf{s}_i||^2} \frac{\mathbf{s}_i}{||\mathbf{s}_i||} \tag{3.1}$$

where $\mathbf{v}_i$ is the "squashed" version of $\mathbf{s}_i$. With the outputs of all capsules of a layer l ready, the next step is to determine their connectivity to the capsules in the layer $l + 1$. Let i be a capsule in layer l, and j a capsule in layer $l + 1$. A prediction on the output of capsule j can now be made based on the output $\mathbf{v}_i$ of capsule i.

$$\hat{\mathbf{v}}_{j|i} = \mathbf{W}_{ij}\mathbf{v}_i \tag{3.2}$$

where $\hat{\mathbf{v}}_{j|i}$ is the "prediction" vector, and $\mathbf{W}_{ij}$ is the part-whole transformation matrix. All the capsules i in layer l which are connected to any particular capsule j in layer $l + 1$ thus cast their prediction. The output $\mathbf{s}_j$ of capsule j is then given by the weighted sum of all predictions.

$$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{v}}_{j|i} \tag{3.3}$$

where c_{ij} is the weight of each prediction from the capsule i about a the capsule j. c_{ij} are referred to as the coupling coefficients (between two capsules). c_{ij} are determined by a routing "softmax", since we would like them to represent probabilities, thus making $\sum_i c_{ij} = 1$. They are obtained in the following way:

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{3.4}$$

where b_{ij} are logits updated during the Routing procedure. The adjective "Dynamic" in Dynamic Routing comes from the fact that the weights c_{ij} are computed at runtime, i.e they are neither constant nor learned. They are calculated in an iterative manner by the Dynamic Routing Algorithm.

We had said previously that determining the output of capsule j is a clustering problem. This is where we introduce the distance metric in question, which is the agreement factor a_{ij}.

$$a_{ij} = \hat{\mathbf{v}}_{j|i}.\mathbf{v}_j \tag{3.5}$$

The agreement is indeed measured by the dot product between the prediction of capsule j's output and its actual output (which is a weighted sum of predictions). To close the feedback loop, a_{ij} is added to

Figure 3.1: Dynamic Routing Algorithm taken from [13]

Procedure 1 Routing algorithm.

```
1: procedure ROUTING($\hat{\mathbf{v}}_{j|i}$, $r$, $l$)
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 $l$: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
5:         for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{v}}_{j|i}$
6:         for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{v}}_{j|i}.\mathbf{v}_j$
       return $\mathbf{v}_j$
```

the logits b_{ij}. This procedure is repeated for r routing iterations. Thus we obtain the Dynamic Routing Algorithm in 3.1.

3.3 GAN Architectures

In chapter 2, we presented Generative Adversarial Networks, and showed that they consist of two networks: A discriminator and a generator. In this section, we present all the GAN architectures we have built to carry out our experiments. The table 3.1 summarizes the various types of GAN architectures we attempt to build.

Table 3.1: Summary of GAN architectures

Architecture	Generator Type	Discriminator Type
Gan1 (*baseline*)	CNN	CNN
Gan2	CNN	Capsule Network
Gan3	Capsule Network	CNN
Gan4	Capsule Network	Capsule Network

3.3.1 Baseline CNN Architectures

In this work, we want to compare our Capsule Network-based GAN architectures to some satisfying baseline result achieved with traditional Convolutional Neural Networks.

To this end, we use the following (cf. Table 3.2) architecture in our baseline Generator (Gen1) network for MNIST-sized images.

This architecture is taken from the online github implementation by the authors of [7]. It is a simple architecture where we start with a random vector of dimension 128, fully connected to a hidden layer with 4096 neurons. This hidden layer is reshaped, then passed through three partially strided convolution (or “Deconvolution”) layers (with ReLU), until we obtain an image of the same dimensionality as an MNIST image.

The baseline CNN discriminator architecture (Disc1) also comes from the same source. As can be seen in Fig. 3.3, it does the opposite of the generator: It starts with an image, applies three successive convolution and leaky ReLU layers, before the tensor gets flattened, and then fully connected to an output neuron whose output represents the probability that the input image is legitimate (not generated, but sampled from the dataset).

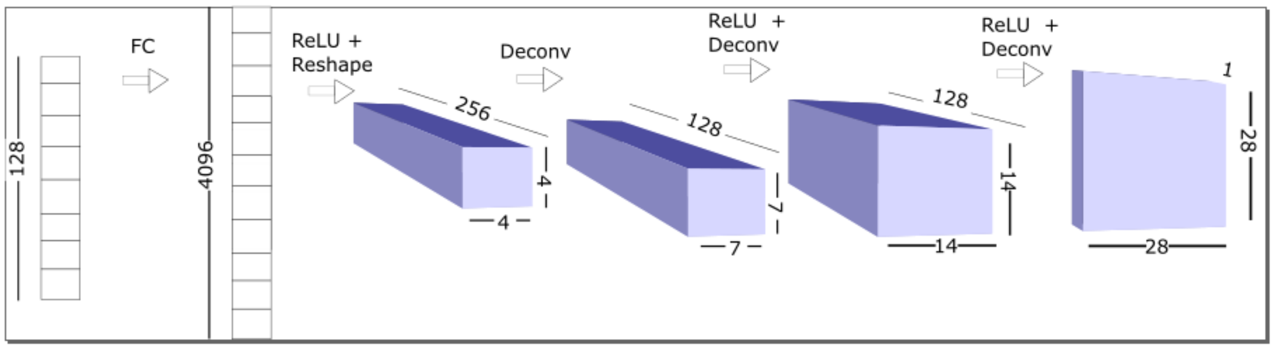

Figure 3.2: MNIST baseline CNN generator architecture

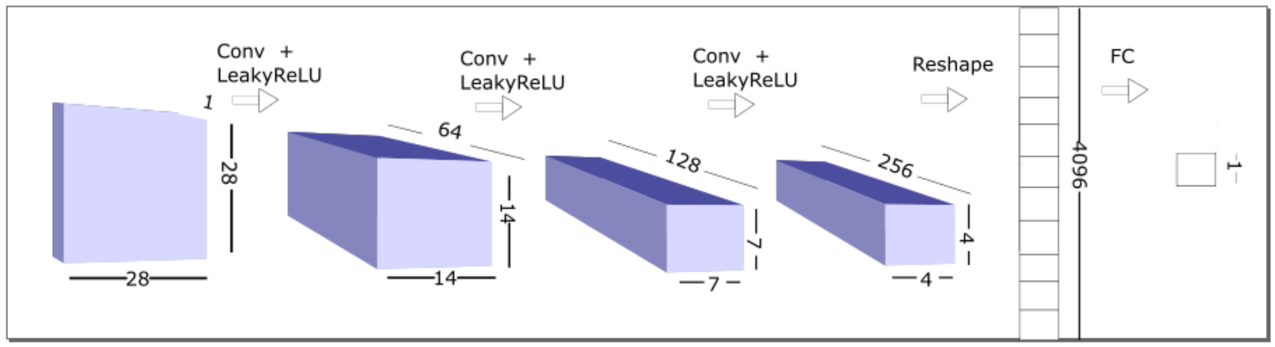

Figure 3.3: MNIST baseline CNN discriminator architecture

For the CIFAR-10 dataset, we used a modified version of the above architecture (of both generator (Gen2) and discriminator (Disc2)) to allow for 3 channels and 32x32 images. The results are shown in the figures 3.4 and 3.5.

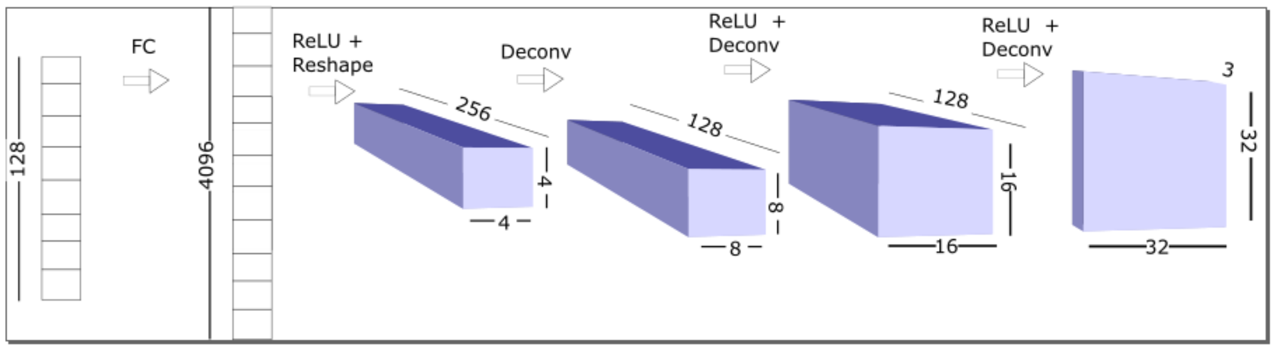

Figure 3.4: CIFAR baseline CNN generator architecture

As for SG-CIFAR-10 dataset, we use the same architectures as the ones used for MNIST to produce the baseline.

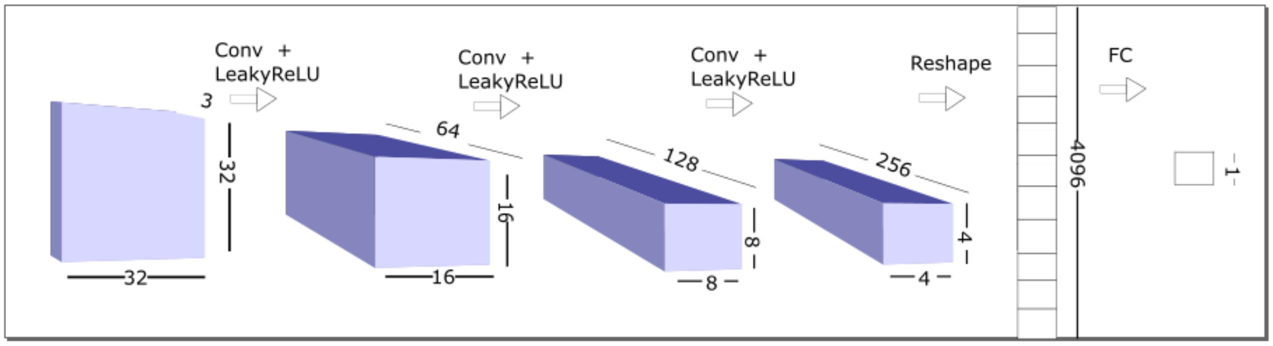

Figure 3.5: CIFAR-10 baseline CNN discriminator architecture

3.3.2 Discriminator Architectures Based on Capsule Networks

The discriminator architectures we build here are inspired by the classifier architecture presented in [13], and shown in Fig. 3.6

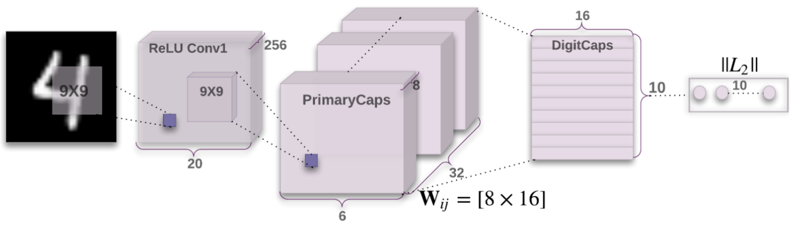

Figure 3.6: Original classifier for MNIST proposed in [13]

The network starts simply with a convolution then ReLU on the imput image. A second convolution is applied. This is reshaped into a tensor of 6x6x32. Each of the 32 channels does not contain single pixels, but rather activity vectors of dimensionality 8. Then, dynamic routing connects this so-called “Primary Capsules” layer into a second layer with ”Digit Capsules”. There are 10 digit capsules (10 digits in MNIST). These have an activity vector output of dimensionality 16.

We turn this architecture into a discriminator (CapsDisc1) (Fig. 3.7) by reducing the number of digit capsules to a single capsule so that the output of the network is simply the norm of this output activity vector. Remember that the norm of the vector here represents the probability that the input image is legitimate. We also make a few modifications such as turning ReLU layers into LeakyReLU activations as recommended by the authors of DCGAN [12]. In a modified version of this architecture, we create (CapsDisc2) which includes batch normalization layer after the second convolution. In yet another modified version (CapsDisc3), we remove the LeakyReLU layers.

3.3.3 Generator Architectures Based on Capsule Networks

The idea behind the Capsule Network-based generator (CapsGen1) is non-trivial, and is a new undocumented endeavor. By contrast, the generator case was simple in that it only consisted in a small re-adaptation of an existing CapsNet classifier.

To design a Capsule Network generator, the idea is to create an ”inverse” network to the CapsNet generator. Fig. 3.8.

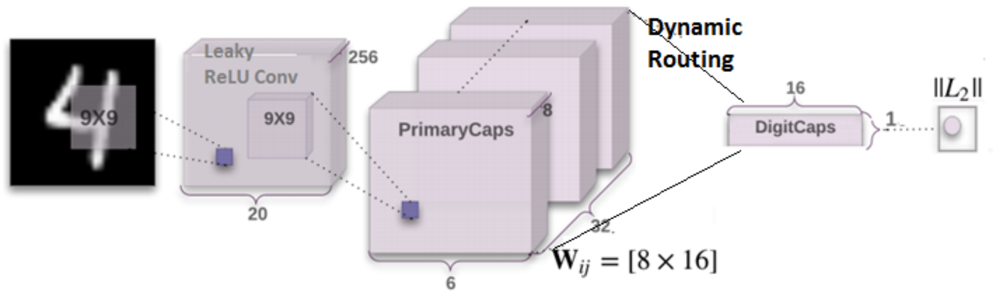

Figure 3.7: Our CapsNet based discriminator (CapsDisc1)

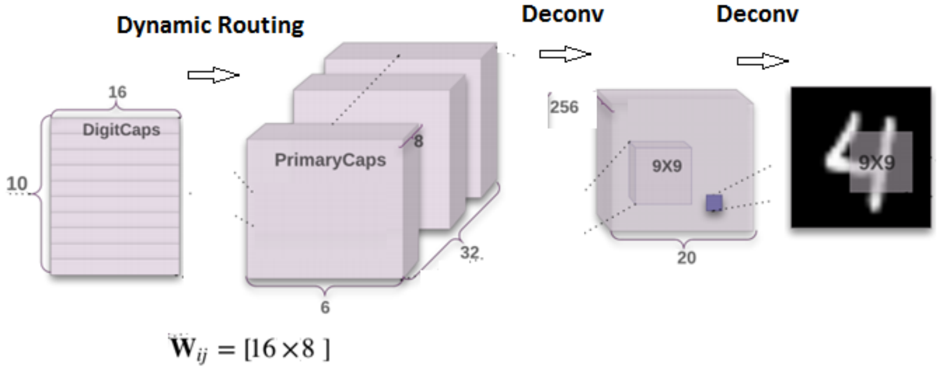

Figure 3.8: CapsNet Generator for MNIST

As with all random generators, we start by feeding it a set of random inputs. In this case, it's a 16x10 matrix. This structure comes from the digit capsules mentioned earlier. However, to generate this noise, with start with selecting a random number from 0 to 9. The selected number indicates the matrix row, or the digit which should be encoded. This row is populated with random scalars. The rest of the rows are filled with 0's. This noise "masking" originates from a similar idea in the Dynamic Routing by Agreement paper [13]. Next, we squash the noise so as to contain the norm of the unmasked noise vector between 0 and 1. Then, we route the DigitCaps to the next layer, using dynamic routing. The next layer is equivalent in dimensions to the Primary Caps layer seen in Fig. 3.6. This is subsequently followed by two partially strided convolution layers which produce an image sized 28x28. We create 3 modifications of this architecture (CapsGen2, CapsGen3 and CapsGen4), where we respectively change the deconvolution mask size from 9x9 to 3x3 and 5x5 (in CapsGen2, and CapsGen3), and in CapsGen4, we replace dynamic routing with a regular fully connected layer for debugging purposes.

Chapter 4

Experiments and Results

4.1 Experimental Setup

In our experimentation, we have tried many combinations of Generator and Discriminator Architectures combined to form a GAN. In this section, we present general unchanged parameters, and we detail the various experimental setups we have tried.

4.1.1 General Parameters

Some parameters have been shared by all experimental setups. In particular:

- **Training**: In GANs, training of Generator and Discriminator is separated. We use tensorflow's Adam Optimizer with its default parameters (unless otherwise specified.)
- **Critic Iterations**: For every training iteration of the generator, we perform 5 iterations of the discriminator (unless otherwise specified).
- **Batch Size**: 64 when training on CIFAR-10, 50 when training on all other datasets.

4.1.2 Summary of All Experimental Setups

The table 4.1 details all the experimental setups, and experiments which have been carried out. It includes details such as the Dataset used, the Generator and Discriminator architecture used, as well as some other hyper-parameters. When a hyper-parameter shows a list (e.g: DLR={1e-4, 1e-5}), this means a grid-search was done over all the parameters specified.

4.2 Results

In this section we will explain the representative results obtained for all the test setups mentioned in table 4.1.

The results we present here can be divided into two sets of experiments. The first set consists of the scenarios with Capsule-Network based Discriminator. The second set corresponds to the experiments with a Capsule-Network based Generator.

One prerequisite for developing these Capsule Network-based generators and discriminators is the verification of our Capsule Network model. We have been able to successfully reproduce the results reported in [13], thus attesting to the correctness of our implementation.

Table 4.1: Details of all experimental setups

Exp. Setup#	Dataset	Generator	Discriminator	Hyperparameters
0	MNIST	Gen1	Disc1	DLR[1]=1e-4 ;GLR[2]=1e-4
1	MNIST	Gen1	CapsDisc1	DLR={1e-3, 1e-4, 5e-4, 1e-5, 5e-5}; GLR=1e-4
2	MNIST	Gen1	CapsDisc2	DLR={1e-5, 5e-6}; GLR=1e-4
3	MNIST	Gen1	CapsDisc3	DLR=5e-6; GLR=1e-4
4	MNIST	CapsGen1	Disc1	DLR=1e-4; GLR={1e-4, 1e-5}; RI[2]={3, 5}
5	MNIST	CapsGen2	Disc1	DLR=1e-4; GLR={1e-4, 1e-5, 1e-6} , RI=3
6	MNIST	CapsGen3	Disc1	DLR=1e-4; GLR={1e-3, 1e-4, 1e-5}
7	MNIST	CapsGen4	Disc1	DLR=1e-4; GLR=1e-4
8	CIFAR-10	Gen2	Disc2	DLR=1e-4; GLR=1e-4
9	CIFAR-10	Gen2	CapsDisc4	DLR=; GLR=1e-4
10	CIFAR-10	Gen2	CapsDisc5	DLR=; GLR=1e-4
11	CIFAR-10	Gen2	CapsDisc6	DLR={1e-5,1e-6,5e-7}; GLR=1e-4; CI[4]={5,7,8}
12	SG-CIFAR-10	Gen1	CapsDisc1	DLR=5e-6; GLR=1e-4
13	SG-CIFAR-10	Gen2	CapsDisc1	DLR=5e-6; GLR=1e-4

[1] DLR: Discriminator Learning Rate
[2] GLR: Generator Learning Rate
[3] RI: Routing Iterations of the Dynamic Routing Algorithm. If not specified, RI=3
[4] CI: Critic Iterations. If not specified, CI=5.

4.2.1 Discriminative Capsule Networks

Experiments on MNIST

Experimental setups 0-3 (c.f. 4.1) correspond to our evaluation of Capsule Network based Discriminators (on MNIST data).

We begin with experiment 0, which simply establishes a baseline result to compare with. The GAN in experiment 0 is based solely on CNNs in its Generator and Discriminator. The baseline CNN GAN therefore is able to produce the images as seen in Fig. 4.1. What you see in 4.1 is a grid of 128 randomly generated digits (generated with a different random input), taken after 14k training iterations.

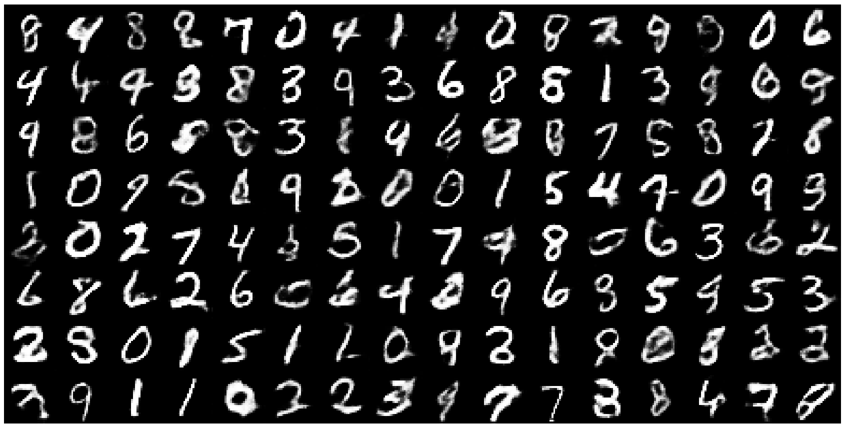

Figure 4.1: MNIST Baseline with CNN based G and D

Now, experiments 1-3 are attempts at making a GAN with CapsNet Discriminators. To make sure our implementation of CapsNet Discriminator is correct, we construct CapsNet classifier, and made sure it is able to reproduce the accuracy performance reported in [13].

Although we explore multiple variations of the architecture CapsDisc1, CapsDisc2 and CapsDisc3, the results they produce are similar in quality. There are only some minor differences in convergence rates between them. Fig 4.2 is taken from experiment 2 after 14k iterations, and Fig 4.3 after 28k iterations.

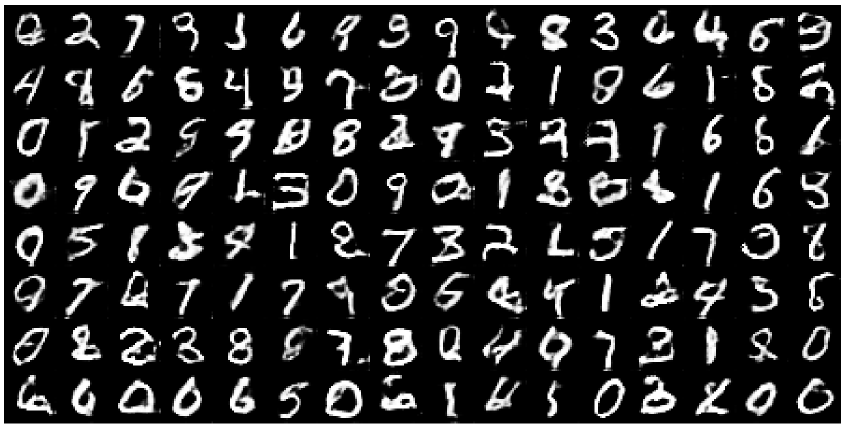

Figure 4.2: Randomly generated digits after 14k training iterations with CapsNet based Discriminator

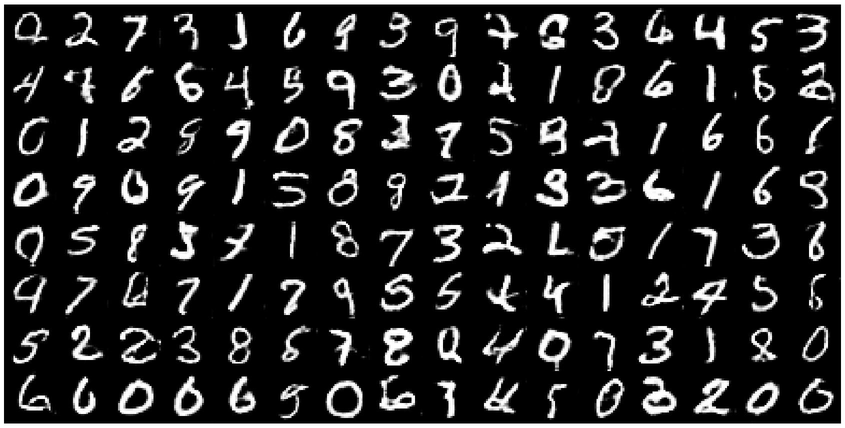

Figure 4.3: Randomly generated digits after 28k training iterations with CapsNet based Discriminator

The results look fairly competitive with the baseline. There does not seem to be any mode collapse (i.e there is variety in the generated samples). In addition, some "new numbers" are created, which is a clue that the network is not strongly overfitting.

Despite these promising results, there are two differences between the two. The first difference is the time taken per training iteration. In the CNN-only GAN, execution is about 4x faster than the one using Capsules. In addition, the convergence rate is much faster.

All in all, it seems that our architecture has passed the MNIST sanity check. We want to see next how it performs on a dataset with much richer image semantics, such as CIFAR-10 and SG-CIFAR-10.

Experiments on CIFAR

To begin with, we establish a CNN-based baseline GAN for generating CIFAR-10 images. This corresponds in the experiments summary table 4.1 to experiment 8. The baseline results we achieve are presented in Figs. 4.4, 4.5 and 4.6.

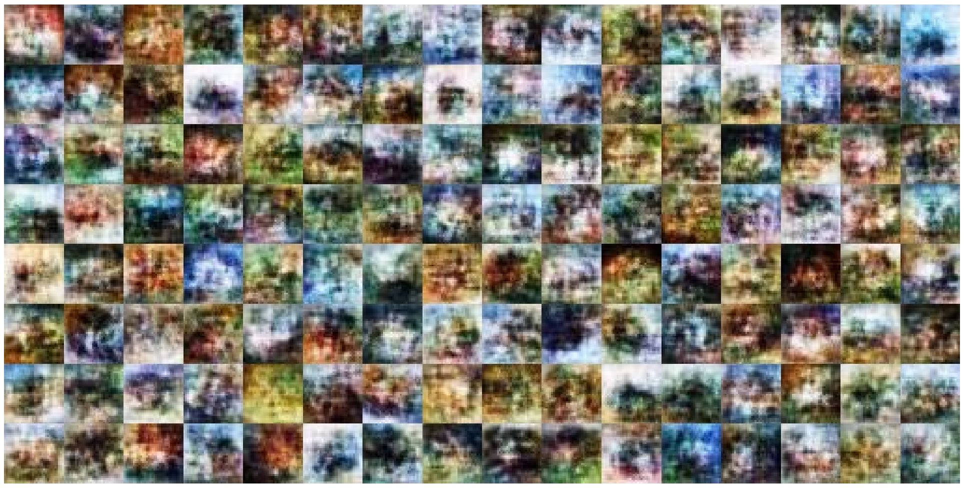

Figure 4.4: Randomly generated images after 500 training iterations with CNN-based Discriminator

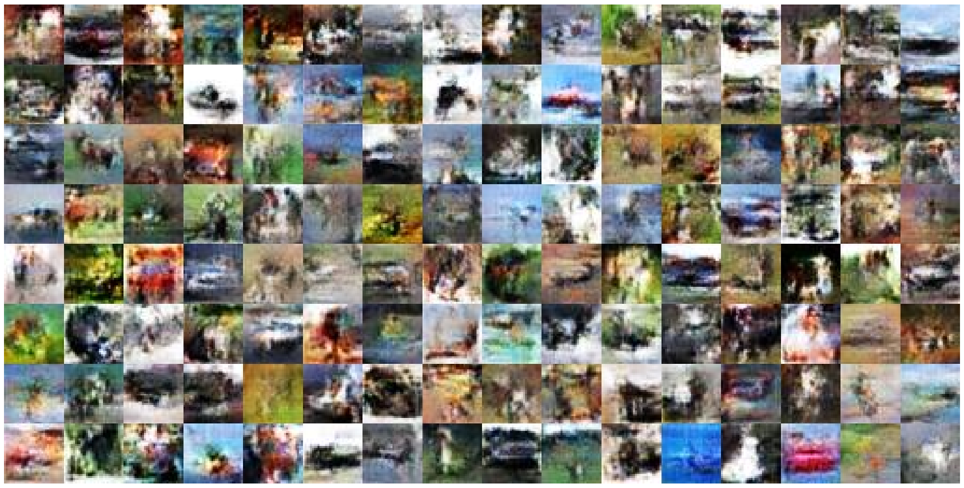

Figure 4.5: Randomly generated images after 5000 training iterations with CNN-based Discriminator

It seems the baseline CNN-based GAN is capable of learning how to generate realistic-looking CIFAR images. Next we look at the CIFAR GANs using CapsNet Discriminators.

In table 4.1, experiments 9-13 are run on CIFAR-10 or SG-CIFAR-10 datasets. We started with experiments on CIFAR-10. The "best-looking" results we obtained with CIFAR-10 came from experiment 11, and are presented in figure 4.7, and 4.8. The two figures are generated within 100 training iterations from each other.

There are a few problems here. First, we observe a mode collapse, where there is very little variation in the generated images regardless of the random input we feed the generator. The images are nearly identical up to a few pixel differences. Another problem is the lack of convergence. We have shown in Figs. 4.7, and

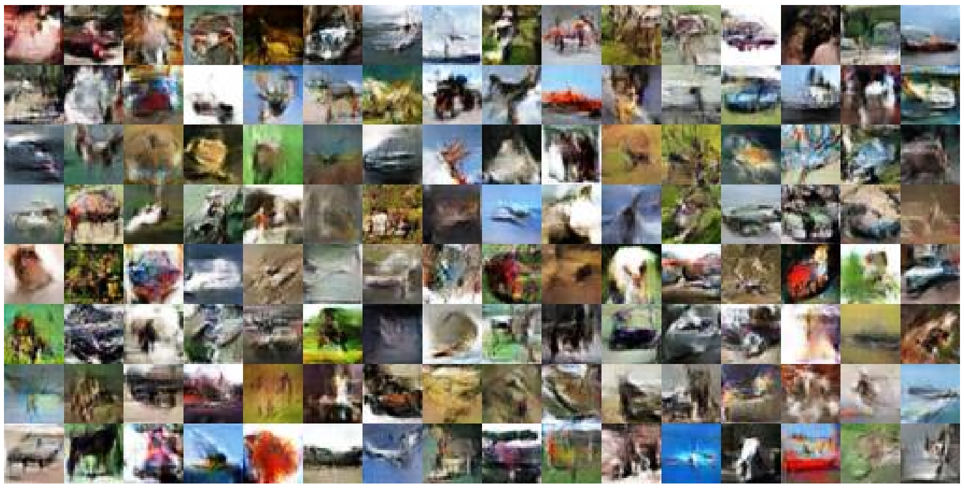

Figure 4.6: Randomly generated images after 20000 training iterations with CNN-based Discriminator

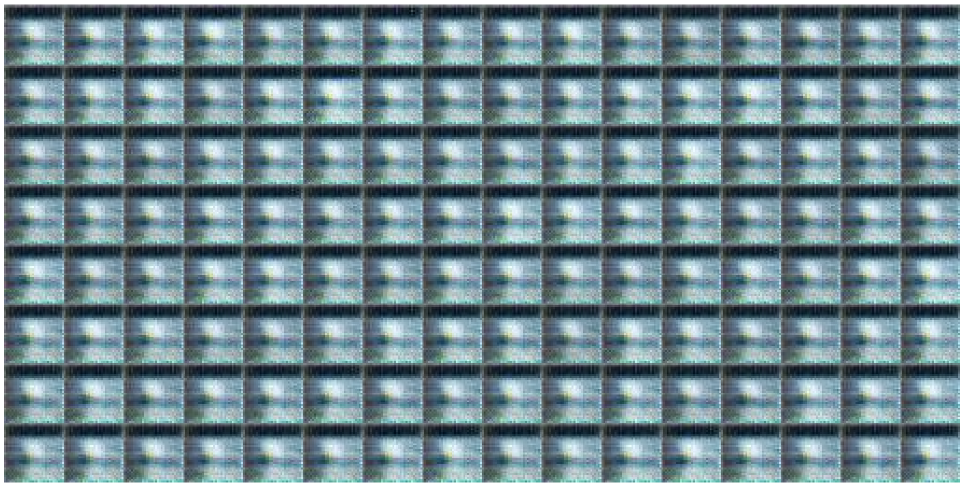

Figure 4.7: Randomly generated images after 7k training iterations with CapsNet based Discriminator

4.8 which are only a few training iterations apart, that the appearance of the output does not at all converge. This behavior continues.

We first thought that this could be due to an implementation problem, so to verify this, we created a second dataset SG-CIFAR-10 which converts the original CIFAR images into single-channel 28×28 images. We do this in order to use SG-CIFAR-10 dataset on the already working CapsNet-based architecture we designed for MNIST (and which we presented in the previous section 4.2.1). The new experiments with SG-CIFAR-10 corresponds in the experiments summary 4.1 to experiments 12 and 13. The best results from these come from experiment 12, in which we produced the figures 4.9 and 4.10 about 4k iterations apart one from another.

We notice here that the problem of mode collapse is gone, and we have some convergence, since there is some similarity between the figures we produced at rather different times during the training. However, the quality of the produced images stops evolving even when we let it train for many hours. Nevertheless, it seems our network is incapable of capturing the richness of the image semantics of the CIFAR dataset.

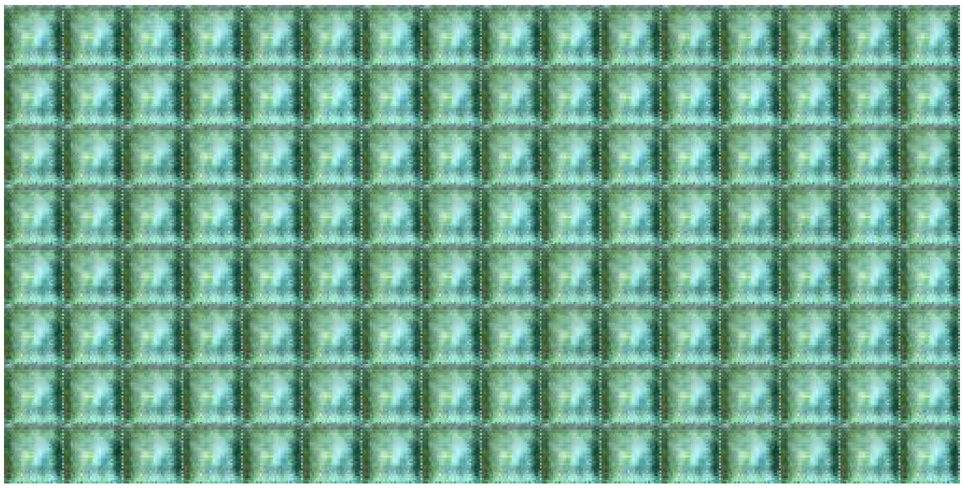

Figure 4.8: Randomly generated images after 7.1k training iterations with CapsNet based Discriminator

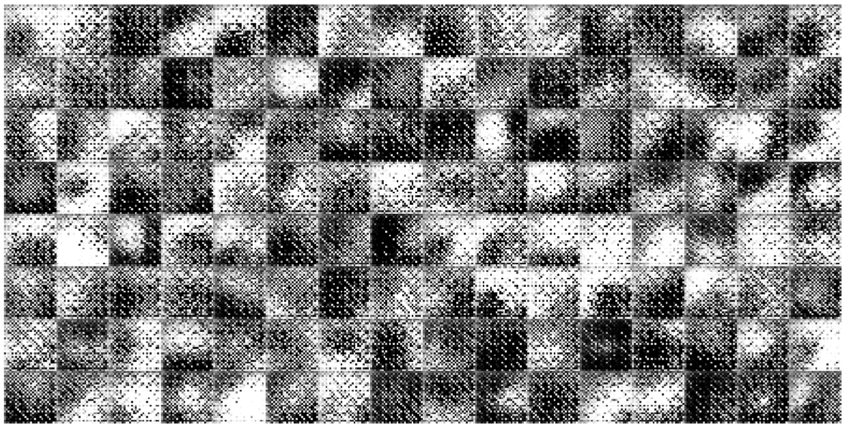

Figure 4.9: Randomly generated images after 4k training iterations with CapsNet based Discriminator

4.2.2 Generative Capsule Networks

In table 4.1, the experimental setups 4-7 correspond to the trials we had conducted with a Capsule Networks-based generator.

Invariably, the result was mostly black images with some minor repetitive grid-like structures in the pixels. Fig 4.11 shows an example. Upon inspection, it seems that the elements of the transformation matrices $\mathbf{W}_{ij}$ collapse to 0.

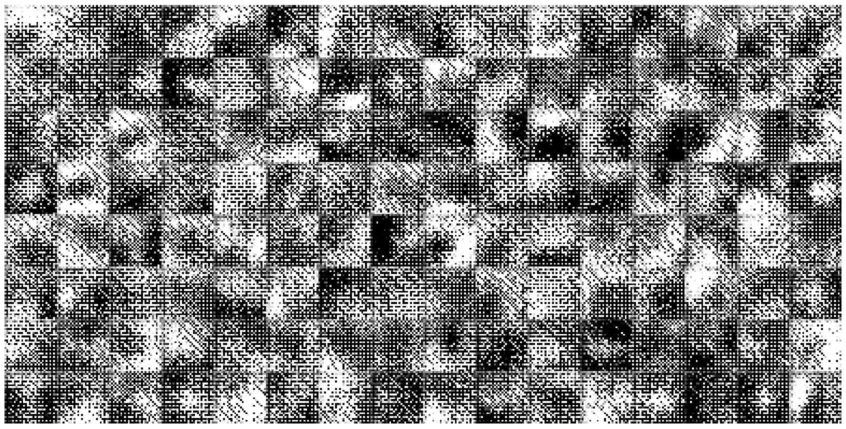

Figure 4.10: Randomly generated images after 8k training iterations with CapsNet based Discriminator

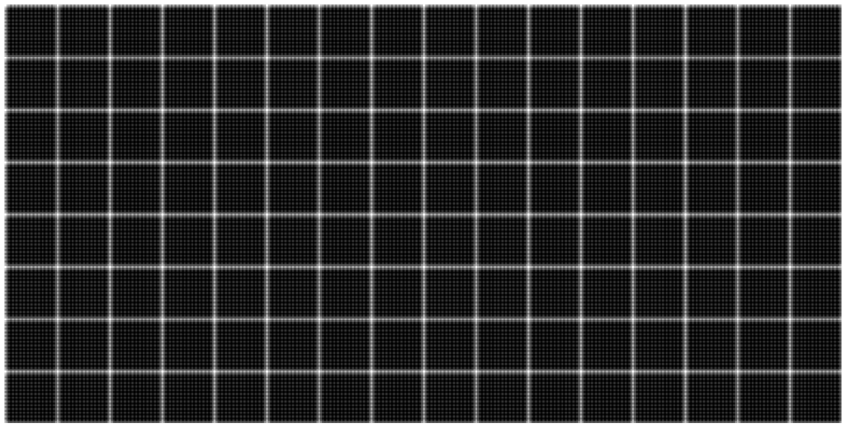

Figure 4.11: Output image from GAN with CapsNet Generator

Chapter 5

Discussion

5.1 A Fundamental Flaw in the Dynamic Routing Algorithm

5.1.1 Dissection of the Problem

One of the reasons we believe a CapsuleNetwork fails as a discriminator for datasets with rich image semantics has to do with a number of factors. The first among them is the distance metric we had talked about in chapter 3, in equation 3.5. This distance metric (which is the agreement factor) is the dot product of the prediction vector $\hat{\mathbf{v}}_{j|i}$ with the computed output $\mathbf{v}_j$. However, because it is a dot product, the larger the magnitude of the prediction vector $\hat{\mathbf{v}}_{j|i}$, the higher the probability of the capsule producing it getting activated. Essentially, the norm of an output activity vector of a capsule encodes the probability that the feature (which the capsule is sensitive to) exists. We believe this to be a serious flaw.

Here is a contrived example: suppose one of the elements of the vector encodes the rotation of the detected feature. Suppose a value of 0 encodes a rotation of 0 degrees and a value of 1 encodes 90 degrees. Thus, the magnitude of the vector encoding 90 degrees rotation is larger than the vector encoding 0 degrees rotation, therefore, the agreement factor a_{ij} is larger for the vector encoding 90 degrees. Therefore c_{ij} will also be larger, meaning that the capsule encoding 90 degrees will have much more impact! There is no reason for this to be the case. We strongly believe that this leads to many meaningless routing decisions by the routing algorithm.

One may wonder how it was successful case for MNIST data. Our hypothesis about why this happens is that image semantics in MNSIT are so simple that the Capsule Networks-based discriminator finds a way to learn something meaningful even with some meaningless routing connections.

5.1.2 Suggestions for Improvement

We suggest a modification of the routing algorithm such that the magnitude of the vector outputs no longer has an effect on the probability of activating the capsule. This could be achieved with a different distance metric. The probability of a capsule activating should be encoded by a separate logistic unit in each capsule. As a matter of fact, this is precisely what is done in a very recently published follow-up work on Dynamic routing [4] by its own authors.

5.2 Timing Issues and Convergence Rate of CapsuleNetwork based GAN

As we have seen in chapter 4, we succeed in training a GAN with a CapsNet based Discriminator, on the MNIST dataset. Despite the success of this method, we address here the issue of training time. Indeed, it seems that the CapsuleNetwork-based GAN takes many iterations to train, before it produces images similar in realism to the CNN based discriminator.

The reason behind this is simple: In the CapsuleNet discriminator, the network has to learn both the regular parameters of the convolution kernels *and* the part-whole transformation matrices $\mathbf{W}_{ij}$. This is why CapsNet requires a longer time to converge.

In addition, each training iteration requires more time on average, 1.8 seconds vs 0.36 seconds. This is in part due to the fact that routing is not static, and the Dynamic Routing Algorithm must run r iterations for each pass.

All in all, it seems that for datasets with very simple image semantics such as MNIST, CapsuleNetworks achieves similar performance in terms of how well it can generate realistic-looking images, however, its training time and running time overhead make it difficult to justify its use instead of a regular Convolutional Neural Network.

5.3 Capsule Networks as a Generator

5.3.1 Failure Mechanism

Our results in chapter 4 show complete failure of the inverted CapsNet as a generator. The output images were regular structures which did not seem to evolve much throughout the training.

It seems there is a numerical instability in the gradients. After much testing, we had come to the conclusion that it was likely a fundamental issue embedded into the math of the dynamic routing algorithm, rather than an implementation error.

We turned to the authors of [13] for more information. An excerpt of their explanation is reported below. Their full answer is in the appendix.

“The main problem will be that we used an efficiency hack for the routing that only works with discriminative training. [...].If we measure the error in this (incorrect) way and use unsupervised training, it will make all of the predictions be zero so they all fit perfectly but explain nothing. ” **Geoffrey Hinton**

“The gradients of capsule network with respect to input image are not very stable in my experience. Therefore, designing an “inverse” capsule network can be tricky.” **Sara Sabour**

Our explanation of the problem is the following. In chapters 2 and 3, we have framed the Dynamic Routing Algorithm as a method to solve a high-dimensional clustering problem, or co-incidence filtering problem. Looking closely at the algorithm, the datapoints of this clustering algorithm are the prediction vectors $\hat{\mathbf{v}}_{j|i}$, which are a linear transformation of the squashed vector outputs $\mathbf{v}_i$. This means that different high-level capsules see a different version of the prediction vector space, since each output vector of a low-level capsule is premultiplied by a *different* transformation matrix $\mathbf{W}_{ij}$. So two close predictions for one capsule might be far from each other for a different capsule. For reasons beyond the scope of this project, this ends up not being a problem in the discriminative training case. However, in the unsupervised, generative case, it turns out to be fatal.

It seems that inverting the Capsule Network to create a generator is a larger task than anticipated. The author does indeed suggest a feasible solution (see Appendix), however it is a computationally expensive one requiring many matrix inversions of the weight matrices $\mathbf{W}_{ij}$ and a modification of the dynamic routing algorithm. Therefore, creating a capsule Network based generator remains an open research question worth investigation. Time constraints place such an endeavor outside the scope of this semester project.

5.4 Results of CapsNet GAN on CIFAR-10 and SG-CIFAR-10

The results we achieve on CIFAR-10 AND SG-CIFAR-10 are not satisfactory. There are 2 hypotheses behind why this occurs. The first hypothesis was presented in 5.1.1. The second hypothesis is that the current discriminator model used is not sufficiently large or deep to be able to learn the rich semantics of CIFAR. This would require some additional effort exploring deeper Capsule Network architectures. The work presented in [4] is a step in that direction. We have not been able to carry out this extended investigation due to time constraints of the project.

Chapter 6

Conclusion

In this semester project, we have explored the applications of Capsule Networks with Dynamic Routing to the context of Generative Adversarial Networks.

We have attempted a Capsule Network-based GAN whose discriminator is a Capsule Network. The attempt was successful when compared to a baseline CNN GAN. However convergence time, and computational time were higher in order to achieve similar quality of results.

Despite their success on the MNIST dataset, we have not been able to show competitive results with datasets with rich image semantics (CIFAR). This is possibly due to two reasons. The first is a problem with the use of activity vector norm as a measure of probability. The second is the possibility that the network is not deep enough, and thus calls for further exploration of deeper CapsNet discriminator architectures.

On the other hand, the use of CapsNets for the generator of GANs has shown to be impossible using the current dynamic routing algorithm. This is due to a computational "efficiency" hack baked into the dynamic routing algorithm. Attempting to solve the ”inverse” or generative Capsule Network remains an open research problem.

In the future, Capsule Networks applied to GANs might see a few directions for evolution. The first is the implementation of more efficient dynamic routing algorithms, and fixing of the shortcomings of Capsule Networks which we have exposed in this work. Some recently published work had begun the efforts of refining dynamic routing by reformulating the underlying clustering problem [4] into a Gaussian Mixture Model solved with the EM algorithm.

The second would be to see Capsule Networks applied to other networks such as Variational Autoencoders which also use Classification and Generation. A final interesting application of CapsuleGANs is the the generation of 3D data, where the idea of linear transformations on 3D objects fits well to the idea of Capsule Networks and Dynamic Routing.

Appendix A

Appendix

A.1 Communication with the Authors of Dynamic Routing Between Capsules

"The main problem will be that we used an efficiency hack for the routing that only works with discriminative training. The correct way to measure how well a higher level capsule explains a lower level capsule is to reconstruct a gaussian predictive distribution for the mean of a lower level capsule from the gaussian distribution of the mean of a higher level capsule and then measure the probability density of the lower level mean under this predictive distribution. But this involves inverting the part-whole matrices. So instead, we measure we use discriminative training of the part-whole matrices. If we measure the error in this (incorrect) way and use unsupervised training, it will make all of the predictions be zero so they all fit perfectly but explain nothing.

Good Luck! Geoff" **Geoffrey Hinton**

"Hi Andrawes, The gradients of capsule network with respect to input image are not very stable in my experience. Therefore, designing an "inverse" capsule network can be tricky. We have not done any research in this area. In the context of our ICLR submission it should be feasible to sample from Gaussian of each capsule to inverse the network and seems a plausible and interesting idea. Cheers, Sara" **Sara Sabour**

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[3] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3d shape induction from 2d views of multiple objects. *arXiv preprint arXiv:1612.05872*, 2016.

[4] Nicholas Frosst Geoffrey E Hinton, Sara Sabour. Matrix capsules with EM routing. *International Conference on Learning Representations*, 2018.

[5] Aurélien Géron. Introducing capsule networks. 2018.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[9] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint*, 2016.

[12] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[13] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.