

UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology

**Advanced Techniques of Cross Domain
Translation Learning**

by

Wanming Huang

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

Sydney, Australia

2020

Certificate of Original Authorship

I, Wanming Huang, declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and IT at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Production Note:

Signature: Signature removed prior to publication.

Date: 17/09/2020

ABSTRACT

Advanced Techniques of Cross Domain Translation Learning

by

Wanming Huang

Cross domain translation, such as image captioning, fashion synthesis from text descriptions, music composition in a particular style, has attracted considerable interest in the deep learning community lately. Despite significant progress in this field, certain drawbacks in previous methods have been identified. First, although the attention mechanism has been widely applied to domain transfer and achieved remarkable outcomes, cross-domain translation remains an open research question on cross domain transfer learning because of the different data structures. Second, most domain translation algorithms address only a pair of domains, and there is a need for $2 \times \binom{N}{2}$ transfer functions given N image domains. This makes training prohibitively unmanageable. We have proposed a set of solutions to solve these two problems, as described in detail in Chapter 3. Third, most generative model based domain-transfer algorithms uses single-mode distribution to model the latent space. This does not work well on datasets that contain diversified samples that form multiple clusters. Our study applies mixture models to cross-domain generation, of which the effects and properties are illustrated in Chapter 4. Finally, cross-domain translation models usually suffer from long training time and are difficult to converge. Indeed, this applies to most deep neural network training that involves complex network designs and large datasets. Our work in Chapter 5 accelerates deep neural network training with a specially designed mini-batch sampling strategy.

Dissertation directed by Associate Professor Richard Yi Da Xu

School of Electrical and Data Engineering

Acknowledgements

First and foremost, I would like to express deep feelings of gratitude to my supervisor, Associate Professor Richard Yi Da Xu, who has guided me through my PhD journey. He has taught me how to conduct research, and how interesting and amazing machine learning can be. He has spent time with me and my colleagues from day one, taught us calculus, statistics and all other mathematical theories required to conduct machine learning research. He has also shared his insights and the latest breakthrough in many areas of machine learning, which enlightens our own research. Further, he has provided guidance to each one of my paper and my thesis, not only on research ideas, but also on the structuring and writing. He helped me overcome each obstacle I met in my research. I appreciate all his time, effort, and funding that supported my PhD research. I am also thankful for his dedication and hardworking towards research, which exemplified moral and ethical research.

I would also like to thank my external supervisor Dr. Ian Oppermann. Ian and I have arranged meetings regularly to discuss my research progress and exchange ideas on the application of machine-learning algorithms to industry projects. I am deeply grateful for these meetings because they provided insights into my research. Ian also shared his own experience in research and discussed with me possible applications of my work in the industry, which shed light upon my future career path. Discussions with Ian further allowed me to examine problems in a more systematic way and realised what could be improved and what was left to be addressed. He also provided me with valuable advice on PhD study and thesis structuring.

The members of the research group have also contributed immensely to my research and personal life. Discussions with my colleagues, Shuai Jiang, Xuan Liang, Chen Deng, Yi Huang, Ying Li, Wei Huang and all other group members have allowed me to conquer numerous challenges. We have collaborated in multiple

projects, and I greatly appreciate the inspiration and suggestions they provided from multiple fields. Shuai Jiang helped me a great deal with optimisation and statistics; Xuan Liang has enlightened me with his experience in neural translation models. I would also like to thank Haodong Chang and Wei Huang for their help in finding a proofreader.

Regarding my research in latent space modelling of generative adversarial networks with applications in anomaly detection, I would like to thank Dr. Hongbo Xie from the University of Queensland. He not only provided me with his knowledge and experience from a statistical perspective, he has also provided me with valuable advice on paper writing.

Further, I would like to thank Elite Editing for their effort in proofreading this thesis, and editorial intervention was restricted to Standards D and E of the *Australian Standards for Editing Practice*.

Finally, I would like to thank my parents for always being supportive of my research and my life.

Wanming Huang
Sydney, Australia, 2020.

List of Publications

Conference Papers

- C-1. W. Huang, R. Y. D. Xu, and I. Oppermann, “Realistic image generation using region-phrase attention,” in Proceedings of The Eleventh Asian Conference on Machine Learning, ser. Proceedings of Machine Learning Research, W. S. Lee and T. Suzuki, Eds., vol. 101. Nagoya, Japan: PMLR, 17–19 Nov 2019, pp. 284–299.
- C-2. W. Huang, R. Y. D. Xu, and I. Oppermann, “Efficient diversified mini-batch selection using variable high-layer features,” in Proceedings of The Eleventh Asian Conference on Machine Learning, ser. Proceedings of Machine Learning Research, W. S. Lee and T. Suzuki, Eds., vol. 101. Nagoya, Japan: PMLR, 17–19 Nov 2019, pp. 300–315.
- C-3. “GAN-based Gaussian Mixture Model Responsibility Learning” accepted by ICPR 2020

Contents

Certificate	ii
Abstract	iii
Acknowledgments	iv
List of Publications	vi
List of Figures	xi
Abbreviation	xvi
Notation	xviii
1 Introduction	1
1.1 Background	1
1.2 Problems	3
1.3 Contributions	10
1.4 Research Objectives	14
1.5 Thesis Organisation	14
2 Literature Survey	16
2.1 Introduction	16
2.2 Cross Domain Translation	16
2.3 GAN and its Variants	18
2.3.1 GAN	18
2.3.2 GAN Variants	19

2.3.3	GAN Based Text-to-Image Generation	20
2.3.4	Attention Mechanism and its Applications in Text-to-Image Generation	23
2.3.5	Text-to-Image Alignment	25
2.3.6	Image Domain Transfers With GANs	26
2.3.7	Normalisation Methods and Applications in GANs	27
2.4	Latent Space Modelling of GANs	29
2.4.1	GMM	29
2.4.2	GMM in GANs	30
2.4.3	Dirichlet Process	30
2.4.4	DP Applications in Generative Models	32
2.4.5	Other Latent Space Modelling	33
2.5	Mini-Batch and SGD	34
2.5.1	DPP	35
2.5.2	K-DPP	36
2.5.3	Markov DPP	36
2.5.4	DPP for Mini-Batch Sampling	36
3	Applications of Cross Domain Transfer	39
3.1	Realistic Image Generation using Region-Phrase Attention	40
3.1.1	Text Encoder	42
3.1.2	Overall Text Embedding Loss	48
3.1.3	Attentional Text-to-Image Generation	48
3.1.4	Bounding Box Prediction	53
3.1.5	Experiments	54

3.2	Transfer of One Thousand Styles	62
3.2.1	Updating Chain Ordering	63
3.2.2	Alternative Mechanic using Scheduled Sampling	66
3.2.3	Experiments	67
3.3	Summary	73
4	Latent Space Modelling in Style Transfer	75
4.1	GAN-based GMM Responsibility Modelling	76
4.1.1	Posterior Consistency Module (PCM)	76
4.1.2	The Generator	79
4.1.3	The Discriminator	79
4.1.4	Training Parameters for the Prior Distribution	80
4.1.5	Experiments	80
4.2	Compressing GANs with Gaussian Mixture Prior	87
4.2.1	Method	87
4.2.2	Experiments	90
4.3	GMM in Text-to-image Generation	98
4.4	Dirichlet Process GAN	101
4.4.1	Generate z via Particle Filter	102
4.4.2	Updating DPGMM Parameters	106
4.4.3	Updating Other Parameters	106
4.4.4	Experiments	108
4.5	Summary	111
5	Training Acceleration in Style Transfer	117

5.1	Efficient Diversified Mini-Batch Selection Using Variable High-Layer Features	118
5.1.1	Gram-Matrix Construction from Variable Higher-Layer Features	119
5.1.2	Computation Complexity Analysis	126
5.1.3	Experiments	128
5.2	Diversified Mini-Batch Selection in Text-to-Image Generation	134
5.3	Summary	137
6	Conclusion	138
7	Appendix	143
7.1	Network Details for the Text-to-Image Generation	143
7.1.1	Hyperparameters	143
7.1.2	Basic Network Blocks	143
7.1.3	Network Architecture for the Generator	145
7.1.4	Network Architecture for the Discriminators	146
	Bibliography	148

List of Figures

1.1	Illustrative example of the mapping between latent distribution and data distribution when the latent distribution is modelled with (a) a single-mode Gaussian distribution; and (b) a mixture of Gaussians.	6
1.2	Illustrative example of the mismatch between latent distribution and data distribution.	7
1.3	Illustration of clustering (a) low-dimensional data and (b) high-dimensional data.	8
1.4	Illustrative example of the mapping between latent distribution and data distribution when the number of modes in the latent distribution is (a) smaller than; and (b) larger than the actual number of data modes.	9
2.1	In both figures, magenta points are drawn from k-DPP. Cyan points are its subsequent draw. Green points are samples selected in both steps, where (a) is using an independent k-DPP, and (b) is drawn from a Markov k-DPP, conditioning the first.	37
3.1	Network structure for text embedding and GAN network.	41
3.2	Text embedding with a two-layer LSTM networks.	42
3.3	Text embedding with CNN layers. n represents the number of words in the sentence. “heads = 4” means four scaled dot-product attention run in parallel.	44

3.4	Examples of full image region, the regular-grid region and object-grid region	45
3.5	Thumbnail generator	49
3.6	Thumbnail bounding box conditioned discriminator	49
3.7	Example of attention being paid to a phrase when generating each object-grid region. White rectangles on the left figure highlight the object-grid regions in the image. The matched pair of phrase and object-grid image region is highlighted in the right image.	52
3.8	Validation losses of coordinates prediction and number prediction in terms of whether to use the sentences that include position related words for the training.	62
3.9	Overall architecture of the proposed framework in two epochs. The graph shows the structure when in t^{th} epoch, the order of styles is $3, 1, 2, \dots, N$, the order is updated to $1, 3, 2, \dots, N$ in the next epoch. The style ordering shown in the graph is for illustration only; the actual ordering is decided during training.	63
3.10	NOD=1 and NOD=2. Numbers are the index of each style, the two nodes being swapped in each step are highlighted in yellow.	65
3.11	Forward and backward generation results for the toy dataset. Real samples are in blue and the generated samples are in orange.	68
3.12	Forward and backward generation results on MNIST. The first row of both graphs is the real images; the following three rows are synthetic samples by taking 1, 2, and 3 steps from the real samples.	70
3.13	Forward and backward generation results on cars from multiple angles.	71
3.14	Forward and backward generation results on images with multiple brightness.	73

4.1	The overall architecture. The feed-forward logic of the classifier C , the generator G and the discriminator D are marked with different colours.	76
4.2	Samples generated by the proposed models trained on the MNIST (left column), Fashion-MNIST (middle column) and CIFAR-10 (right column) datasets. The top row contains images generated using random vectors sampled from a different Gaussian. The bottom row shows the linear interpolation result from one Gaussian to another. The index of Gaussian does not necessarily correspond to the actual digit, generated images are reordered for demonstration purpose.	83
4.3	Linear interpolation over three Gaussians on the MNIST, Fashion-MNIST and Oxford-102 datasets.	84
4.4	Inception score and FID scores over training epochs on the Oxford dataset.	84
4.5	Inception and FID scores over epochs for the proposed GMM-based GAN with various size of the network compared with the vanilla GAN.	86
4.6	Performance on highly imbalanced dataset.	87
4.7	The complete architecture of our model. K is the number of Gaussians in the GMM; z_i is the random vector sampled from the GMM. x_i and \hat{x}_i are the real and fake sample. G , C and D are the generator, discriminator and classifier.	88
4.8	Inception and FID scores over training epochs under different compression rates of GM-GAN and vanilla GAN on the CIFAR-10 and Oxford-102 dataset.	93
4.9	Performance of anomaly detection measured by area under curve. (a) AUC over anomalous classes on the MNIST dataset; (b) AUC over anomalous classes on the Fashion-MNIST dataset; (c) AUC over anomalous classes on the CIFAR-10 dataset.	97

4.10	Overall architecture of GMM based text-to-image generation.	99
4.11	Comparison between the Gaussian and GMM based GANs in terms of the inception and FID scores over epochs on the CUB and MSCOCO dataset.	102
4.12	The complete architecture of DP-GAN.	104
4.13	The overall flow of how the latent DPGMM distribution is updated. Notations on the left are the traditional flow of particles; notations on the right are how it is applied in the proposed architecture.	105
4.14	Samples from the toy dataset and generated from (a) GAN and (b) DP-GAN. Samples from the training set and its PDF are plotted in blue and synthetic samples and the PDF of their distribution are plotted in red. (c) Stick weights for the DPGMM when the model is fully trained.	109
4.15	(a) Samples generated by DP-GAN after trained on the MNIST dataset. Each row contains images sampled from a different Gaussian. Rows are sorted using the components' weights in the descending order. (b) Stick weights for the DPGMM when the model is fully trained.	110
4.16	(a) Samples generated by DP-GAN after training on the MNIST dataset. Each row contains images sampled from a different Gaussian. Rows are sorted using the components' weights in the descending order. (b) Stick weights for the DPGMM when the model is fully trained.	111
5.1	A visualization of three feature vectors being used to construct Gram-matrices. From left to right, each figure represents <i>raw</i> , <i>fixed higher-layer</i> and <i>variable higher-layer</i> features respectively. Each category is represented in a unique colour. Features are dimension reduced using t-SNE.	120

5.2	Duration over T on the Oxford 102 Flower dataset.	128
5.3	Calinski-Harabasz scores over iterations and the validation accuracy over the training duration on the Oxford 102 Flower dataset.	130
5.4	Validation accuracy over the training duration by the proposed sampling methods and the uniform sampling on the Stanford Dogs dataset.	132
5.5	Validation accuracy over the training duration by the proposed sampling methods and the uniform sampling on the Caltech 101 dataset.	133
5.6	Validation accuracy over training duration by the proposed sampling methods and the uniform sampling on the MNIST dataset.	135
5.7	Inception, FID and R-precision over epochs on the CUB dataset. “SGD” denotes the experiment performed using the conventional SG; “FULL-SINGLE” denotes the one performed using the FULL-SINGLE DPP sampling.	136

Abbreviation

3D - three-dimensional

AUC - area under the receiver operating characteristic curve

bi-LSTM - bidirectional long short-term memory

CUB - Caltech-USCD Bird

CNN - convolutional neural network

DCGAN - deep convolutional generative adversarial network

DM-SGD - Diversified Mini-Batch SGD

DNN - deep neural network

DP - Dirichlet process

DPGMM - Dirichlet process Gaussian mixture model

DPP - determinantal point process

FC - fully connected

FID - Fréchet Inception Distance

GAN - generative adversarial network

GAWWN - generative adversarial what-where network

GLU: gated linear unit

GM-GAN: Gaussian mixture generative adversarial network

GMM: Gaussian Mixture Model

GRU: gated recurrent unit

IS: inception score

LSTM - long short-term memory

MSCOCO - Microsoft COCO dataset

NF - normalizing flow

NMT - neural machine translation

NLP - natural language processing

PCM - posterior consistency module

RCM - responsibility consistency module

RCNN - region-based convolutional neural network

ReLU: rectified linear unit

RNN - recurrent neural network

ROC - receiver operating characteristic

RoI - region of interest

SGD - stochastic gradient descent

SMC - sequential Monte Carlo

SVRG - stochastic variance reduced gradient

VAE - variational autoencoder

Nomenclature and Notation

Capital letters denote matrices.

Lower-case alphabets denote column vectors.

$(\cdot)^T$ denotes the transpose operation.

I_n is the identity matrix of dimension $n \times n$.

0_n is the zero matrix of dimension $n \times n$.

\mathbb{R} , \mathbb{R}^+ denote the field of real numbers, and the set of positive reals, respectively.

Chapter 1

Introduction

1.1 Background

Machine translation was originally applied to natural language processing (NLP) to investigate the use of software in the translation from one language to another. The idea of using computers for the translation can be traced back to the 1940s, when researchers proposed and designed systems to translate between languages. One of the most influential publications is the memorandum by Warren Weaver [40]. More work on machine translation emerged over the next 20 years, such as the Georgetown-IBM experiment which offered the first public demonstration of machine translation.

In the early 1970s, the concept of the rule-based system was brought into machine translation. Rule-based systems rely on linguistic information on sources and languages retrieved from dictionaries and grammar manuals. Several important rule-based systems were developed in this period, including Systran, Japanese MT systems, and EUROTRA. Such systems were replaced by statistical methods in the 1990s [12].

Deep learning based algorithms, since the day they were applied to machine translation, have enabled significant performance gains. The so-called neural machine translation (NMT) allows a single, large neural network to be trained directly on source and target sentences. Deep neural networks have proven to be a powerful tool in solving translation tasks. They have been applied to multiple areas of NLP, such as language translation [93, 65, 5, 19, 100] and text summarisation

[74, 60, 20, 18, 85], where some of the trained models can deliver even more accurate results than can humans. They have also been applied to translating between images, such as image super-resolution [58] and image style transfer [27].

The very first NMT model was proposed by Kalchbrenner and Blunsom [46] in 2013. In 2014, seq2seq was proposed by Sutskever, Vinyals and Le [93], who performed the translation using two networks: an encoder that understood the input sentence, and a decoder that outputs the translation outcome. In addition to the encoder-decoder structure, a crucial module was introduced by Bahdanau, Cho and Bengio [5]: the attention mechanism. The attention mechanism enables the decoder to focus on different parts of the input sequence when generating each word of the output sequence.

The technique of neural translation has later been applied to perform cross-domain translation tasks in recent years, mostly because of the development of deep neural networks (DNNs). Cross-domain translation, translates contents between different media, such as text to image, or from sound to video. Valuable research has been conducted in several areas: for instance, high-resolution images generation from a sentence [103], pose guided person image generation [66] and video generation from text [104, 78].

Generative models, such as generative adversarial network (GAN) [31], variational autoencoder [51], normalizing flow [84], are often used as the main framework in DNN based cross-domain translation algorithms. In particular, GAN, a family of generative models based on game theory, is one of the most widely used frameworks in cross domain translation, especially in image generation because images generated by GANs are sharp and can be indistinguishable from real data [48, 11]. While the attention mechanism was originally devised in the field of language translation, it has been applied to perform cross-domain translation, such as text to image gen-

eration in AttnGAN [103]. It was used to guide the generator to focus on certain words when generating specific image regions and has shown promising results.

1.2 Problems

Despite the valuable achievements made in the cross-domain translation, challenges still exist that stop these algorithms from being adopted in daily-life scenarios.

Problem 1: Attention mechanism on cross-domain

The first challenge comes from the limitations of the current network design. Consider text-to-image generation as an example, Algorithms based on the GANs, specifically deep convolutional GAN (DCGAN) [79] and conditional GAN [71] have achieved remarkable progress on various datasets. Works from [83] and [103] have shown promising results in synthesising images that contain a single object, such as on the Caltech-USCD Bird (CUB) [99] and Oxford-102 [76] datasets. However, synthesising an image that models human poses or involves multi-object interactions usually lacks sufficient detail and can easily be distinguished from real images. We believe that the in-depth connection between individual words and image subregions is not yet fully utilised in the current network design and the model performance could be improved upon. Current frameworks build the connection between individual words and equal-sized regular-grid regions. This does not work well when multiple objects are present and multiple words are used to describe each object. For example, consider this sentence: a man swinging a baseball bat. We would expect the two phrases: a man and a baseball bat, each defines an identifiable object in the generated image.

Problem 2: Transfer across N domains with $N > 2$

Another limitation of domain transfer algorithms is that once a network is fully trained, it is only applicable to a dataset that contains limited domains (usually a pair). Existing unsupervised style transfer methods such as CycleGAN [114] and DiscoGAN [50] are only able to capture a specific style through a pair of networks. For multiple styles, they can be extended to train multiple networks, often from scratch, to form a multistyle transfer system. This creates an unnecessary number of parameters by treating each pair of styles as completely independent.

Some recent work has made the multistyle transfer system slightly more elegant. For example Huwe *et al.* [39] proposed to use a global shared autoencoder and N domain-specific encoder/decoders to perform domain translation over N domains. Recently, [14, 96, 37, 15, 62, 17] explored multiple style transfer in a single network. Vincent, Dumoulin and Kudlur [96] proposed the conditional instance normalisation, which allowed the style transfer network to learn multiple styles. It shared all convolutional weights of a style transfer network across many styles, and only tuned parameters for an affine transformation after normalisation for each style. Huang and Belongie [37] designed the adaptive instance normalisation (AdaIN) layer, which performs style transfer in the feature space by adjusting the mean and variance of the content input to match those of the style input. Chen *et al.* [15] proposed an autoencoder network embedded with a StyleBank, consisting of multiple convolution filter banks and each filter bank represents one style; Lwe *et al.* [62] took a noise vector and a selection unit as input to generate diverse image styles. Finally, Chen *et al.* [17] designed a gated transformer module based on the encoder-decoder architecture to help to generate images in multiple styles in a single network.

However, all the above methods require conditional parameters, such as normalisation parameters or style banks, to indicate the specific style. Moreover, they need

to extract explicit style presentations from style images during training. Most importantly, they do not exploit the fact that many databases contain gradual changes of styles, which can help drastically reduce the parameter numbers. For example, a set of three-dimensional (3D) objects observed from increasing angles [26], photographs taken under increasing lighting conditions or drawings containing rough to fine details. In this case, instead of learning a different transfer function between every pair of styles, a pair of universal transfer functions can be applied when styles are properly “sorted”. In this way, the transform achieved by the transfer function is to correspond to a small style change between the domain pairs. For example, they may be a minor difference between their lighting, viewing angle, or drawing details in the applications described earlier.

Problem 3: Latent space modelling of generative models

Apart from the drawbacks of the current network design, the latent distributions of generative models are mostly modelled with the uniform distribution or the standard Gaussian distribution. While effective, this method also renders the latent space non-informative. The choice of distribution is arbitrary and only serves the purpose of stochasticity (i.e., it only enables one to sample data from it). While this works well for some datasets, the underlying assumption of the model does not generalise well for complex datasets, especially for datasets containing multiple modes.

Using an illustrative example, where $z \sim \mathcal{N}(0, 1)$ and the generator aims to generate a single-mode, Picasso paintings. Then, a value $z = 0$ roughly equates to generating what an “average” Picasso painting may resemble, or more generally: where $p(\text{data})$ receives the highest probability. This may be doable for single-mode data distributions; however, it cannot work well for datasets containing multiple modes. Further, along with the painting images examples, in Figure 1.1a, we denote

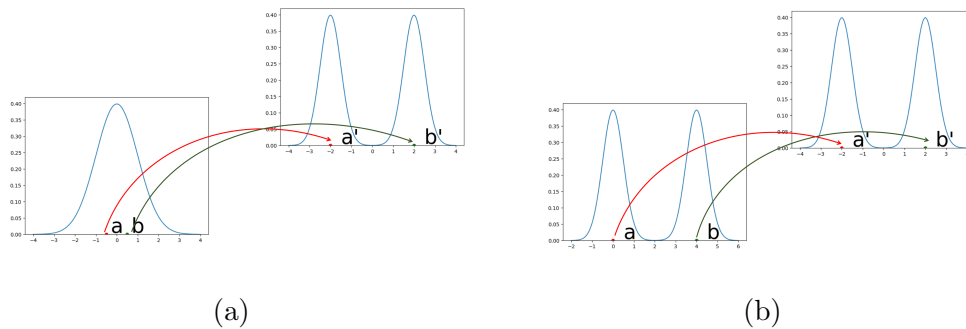


Figure 1.1 : Illustrative example of the mapping between latent distribution and data distribution when the latent distribution is modelled with (a) a single-mode Gaussian distribution; and (b) a mixture of Gaussians.

a' and b' to be the two most likely images of each of the two painters. They are supposed to be farther apart from each other in the data space. Since each receives a high probability, it is natural to assume their z counterparts (i.e., a and b) should also do so in $p(z)$. The only way this can happen is for both a and b to be close and to be zero. Together, it requires the generator to be able to generate images appearing drastically different from one other from a very similar vector in z . This burdens the generator, which may contribute to the difficulties in training GAN. In contrast, we model the latent distribution $p(z)$ using mixture densities, in a way that each mode in the latent space corresponds to the modes in the data space. This is shown in Figure 1.1b; a and b become further apart this time, even though they both receive high $P(z)$. It not only reduces the burden of the generator, but it also makes the $p(z)$ meaningful.

Recently, Ben-Yosef and Weinshall [7] presented a Gaussian mixture GAN (GM-GAN), in which the probability distribution over the latent space is a mixture of Gaussians. However, the properties of the distribution are not fully explored and utilised in this work. That is, GM-GAN expects to have one-to-one correspondence between each class of synthetic images and a component of the GMM without any

supervision. However, situations such as those depicted in Figure 1.2 often occur in training; latent vectors a_1 and a_2 that are close in the latent space are forced to generate samples from different classes because labels are not provided in training. This burdens the training and slows down the convergence.

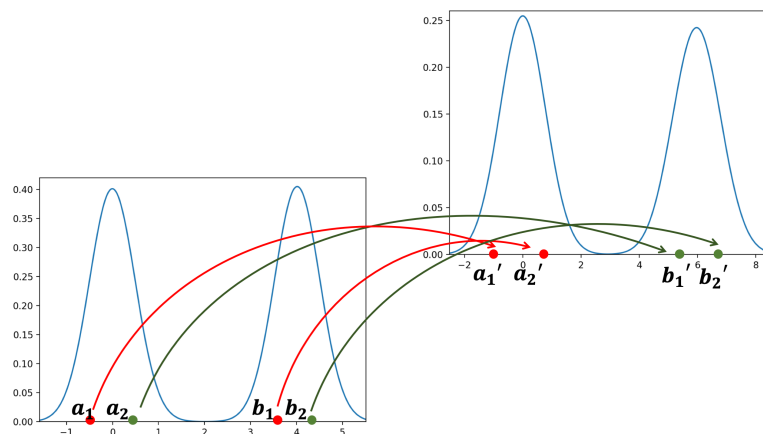


Figure 1.2 : Illustrative example of the mismatch between latent distribution and data distribution.

In addition, given x is the data and k is the (latent) index of the mixture density, GMM allows us to compute the conditional likelihood $p(x|k, \theta)$ and the responsibility probability $p(k|x, \theta)$. In the Bayesian paradigm, one may refer to $p(k|x, \theta)$ as the posterior density, where the prior is $p(k) \equiv (\alpha_1, \dots, \alpha_k)$. This will further allow us to retrieve many important statistics and properties about the dataset: for example, (1) the soft clustering membership of the new and existing data; and (2) the overall weight of each component in the dataset.

When Euclidean distance can be meaningfully defined over the data x in hand (typically when it has a low dimensionality), one can compute both $p(x|k, \theta)$ and $p(k|x, \theta)$ directly (see Figure 1.3a). However, it will be difficult to do so when managing complex and high data dimensional data, such as images, because x in its

raw form does not form mixtures naturally, as shown in Figure 1.3b.

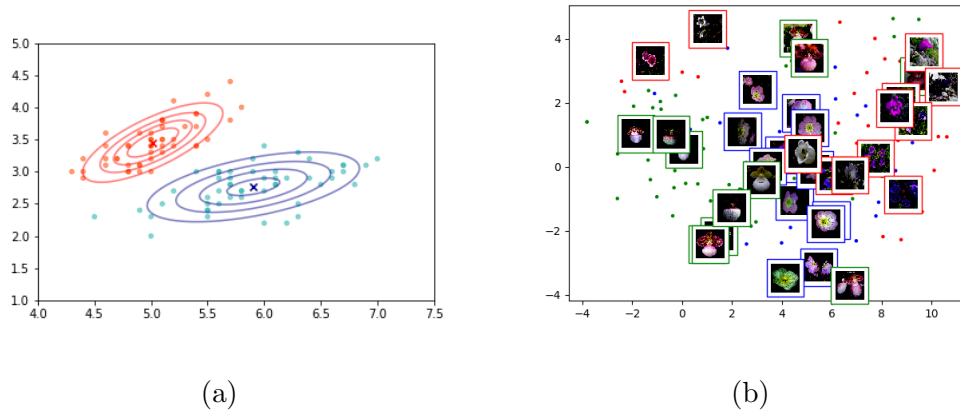


Figure 1.3 : Illustration of clustering (a) low-dimensional data and (b) high-dimensional data.

Combining GANs with the GMM provides a way to compute the latent representation z associated with the data x . However, unlike variational autoencoder (VAE), the generative process of GANs from the latent space to the data space is irreversible and the latent distribution does not necessarily reflect the nature of a dataset. In particular, densities of $p(x|k, \theta)$ and $p(k|x, \theta)$ cannot be obtained by simply modelling the latent space of a Gaussian mixture distribution because:

(1) The generation process is one-directional (i.e., $z \rightarrow x$), making it infeasible to compute $p(k|x, \theta)$ by finding its corresponding z . A GAN can be viewed somewhat like a VAE, with only the decoder $z \rightarrow x$, but without the encoder $x \rightarrow z$.

(2) Even if we were able to obtain $x \rightarrow z$ under GAN, because of its objective function, some training examples may come close to the generated images but might still have nearly zero probability under the generator, making them unable to achieve our goal (i.e., $p(x|k, \theta)$ is may not be a true representation of its likelihood).

Another drawback of simply applying a Gaussian mixture to GAN is that the number of mixtures K needs to be predetermined. The optimal value for K is the

number of modes in the training dataset. In an unlabelled dataset, it is not obvious to guess what this value may be from observing $p(\text{data})$ alone (e.g., a very large clothing dataset containing an unknown number of styles). If we choose K to be less than the actual number of data modes, it produces a similar scenario to that depicted in Figure 1.4a. Conversely, if K is chosen to be larger than the actual data modes, we may potentially produce appearance-wise similar data to have a vastly different value in their z . This is illustrated in Figure 1.4b.

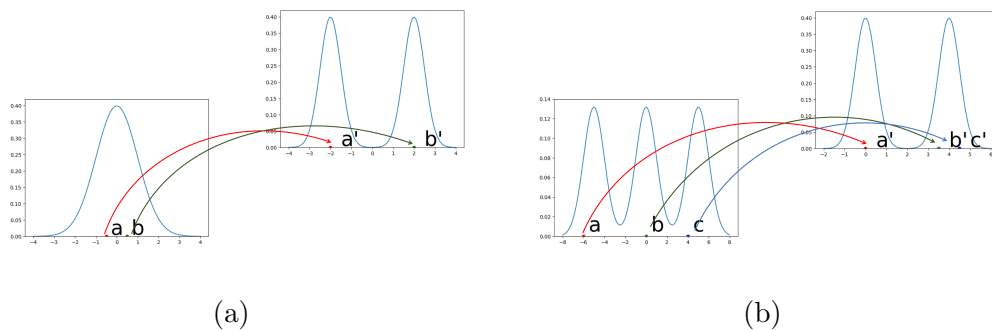


Figure 1.4 : Illustrative example of the mapping between latent distribution and data distribution when the number of modes in the latent distribution is (a) smaller than; and (b) larger than the actual number of data modes.

Problem 4: High training cost

Training a cross domain transfer algorithm is also complicated by high training costs and difficulty in convergence. This could apply to other DNNs trained on large datasets. Mini-batch and stochastic gradient descent (SGD) are common tools used in machine learning. Often, the size of training data becomes so large that it is impractical to render batch optimisation (i.e., using the entire dataset in every iteration). For example, Kulesza and Taskar [53] trained a CNN to classify 1.2 million images from ImageNet. Sutskever, Vinyals and Le [93] built a sequence-to-sequence neural translation model with 12 million sentences that contained 348

million French words and 304 million English words. Therefore, mini-batch training was employed to reduce communication costs and parallelise the learning process [61]. As standard SGD employs uniform sampling, the stochastic gradient is an unbiased estimate of the true gradient. Nonetheless, it introduces variance between iterations, which negatively affects the performance [113].

To improve the efficiency of SGD convergence without the assumption of any underlying data distributions, Zhang, Kjellström and Mandt [108] proposed a strategy called Diversified Mini-Batch SGD (DM-SGD) that applied k-DPP to mini-batch sampling.

Despite the improvement made on the convergence, DM-SGD uses raw input or feature vectors that are not fine-tuned to measure distances. This is prohibitive when DNNs are involved because data are non-linearly transformed from their original space into a layer before the output, such that the features now become much more expressive. Further, mini-batch selection using the determinantal point process (DPP) may become prohibitively slow when the entire set of data Ω is used to compute its Gram-matrix, even if this operation is only computed once. Therefore, the “fixed” feature approach is somewhat only theoretically plausible. Facing these four challenges, we have made the following contributions in this thesis.

1.3 Contributions

Our first contribution is the enhancement of domain transfer applications by improving network structures. A few novel strategies have been proposed to improve the current attention based mechanism. In particular, we uniquely incorporate object-grid image features into the learning of text phrase embedding in the encoder. This embedding is then used to compute a new set of attentions with the image in the generator. These strategies have delivered three unique outcomes:

(1) When generating pixels inside an object-grid region, the attention is paid to the phrases rather than individual words, which makes sense from a natural language perspective.

(2) Pixels within each region describe the same phrase by sharing the same attention score so that objects are more easily recognisable.

(3) Moreover, the spatial information of the generated objects is more likely to be correctly reflected.

In terms of transferring across multiple domains. When natural ordering exists, we propose a methodology that automatically studies the optimal ordering of styles while learning a single pair of universal transfer functions without extra conditional parameters. More concretely, we learn one forward and one backward generator that can transfer across multiple image domains. we separate discriminator functions for each domain. The generators, discriminator and chain sorting parameters are updated in turn. However, the chain sorting function is discrete in nature and even a single pair of node swaps may undo the learning of the other parameters. To this end, several learning strategies are proposed to address this issue, which is to be elaborated upon in section 3.2.

Third, we examine the effects of applying different distributions to model the latent space of GANs. The first distribution used is the Gaussian mixture model (GMM). In particular, we propose to use a posterior consistency module (PCM) as part of end-to-end GAN training. The introduction of PCM has allowed us to efficiently change $p(z)$ in GAN from a single to a mixture of Gaussian densities while still being able to compute the responsibility $p(k|x, \theta)$. This will further allow us to retrieve many important statistics and properties about the dataset in addition to generating high-quality images: for example, (1) the soft clustering membership of the new and existing data; and (2) the overall weight of each of the components in

the dataset.

We also notice that applying GMM to GANs can significantly save the computation cost while achieving a similar outcome. Experimental results suggest that:

1) The size of the GM-GAN network can be reduced while achieving a comparable image quality to vanilla GANs.

2) With the classifier in the network, the image generation quality can be further improved in terms of the inception score [86] and the Fréchet inception distance (FID) score [35]. So far, the application of GM-GANs is mostly limited to image generation. In this study, we further extend the GM-GAN to the anomaly detection problem. We find that our GM-GAN model with classifier results in the best anomaly detection performance on several datasets in terms of the area under the ROC Curve (AUC) value.

The second distribution we use is the Dirichlet process Gaussian mixture model (DPGMM). Compared with the GMM distribution, DP mixtures can be applied to high-dimensional datasets with an unknown number of mixtures; improvements are made over the vanilla GANs to achieve enhanced generation quality. Although this methodology is compelling, adapting it from training traditional DPGMM to DP-GAN requires some novel strategies. The main reason is that the real data x cannot be modelled using DPGMM directly because marginalizing z out is infeasible. Therefore, we model it through their latent counterparts (i.e., z). However, because vanilla GAN does not provide a backward transformation from $x \rightarrow z$, we devised two innovative approaches to this. Inspired by sequential Monte Carlo (SMC) or particle filter [22], using SMC terminology, we first “propose” a set of particles z from the proposal distribution $p(z)$ and then weigh each particle using data x . However, as opposed to traditional particle filters, in which weights can be achieved easily from $p(x|z)$ in the case of condensation filter, in our work, we compute them using the

normalised discriminator D score. In another words, we use normalised $D(G(z))$ to approximate the particle weights. To avoid “particle collapse”, a phenomenon that refers to the situation in which there are only a few particles with higher weights and the rest are close to zero, we also followed the unbiased particle resampling strategy. In addition, it is desirable to ensure that the appropriate G is obtained to ensure consistent posterior responsibility, regardless of using real data x or the pseudo data z (i.e., we want to ensure that $p(c|z)$ and $p(c|x)$ resemble with each other for any given pair (x, z) , c is the cluster assignment variable). Experiments have been conducted on both synthetic and real-world datasets. We demonstrate that the proposed approach can provide an appropriate representation of the latent space in which each component of the mixture corresponds to a specific image class. Although the training is performed in a completely unsupervised manner, each component in the mixture reflects a particular image class at the end. Simultaneously, the model can also generate high-quality images. This allows us to generate images of a desirable class by simply sampling the latent vector from a specific Gaussian.

The last contribution we made applies DPP to mini-batch sampling in DNN training. In particular, our work updates the features to compute the Gram matrix at each iteration. Given sampling DPP once is already far too much computation, sampling at each iteration with an updated Gram matrix would further increase the already-infeasible complexities. Therefore, it is natural to use an approximated DPP to dramatically improve its computation time. By observing this in the last or higher layer, we found that features within each class have a higher tenancy to remain closer and data between classes tend to separate. Therefore, we chose the last fully connected (FC) layer to construct the Gram-matrix in our algorithms. We also obtained the so-called class-dependent DPP sampling using hierarchical sampling to break down a single DPP sampling with large Gram-matrix into many DPP sampling of much smaller Gram-matrix. In the lower hierarchy, each sampling

is to be performed on data within its own class. To further improve the computational efficiency, we also used Markov k-DPP to encourage diversity across iterations. Accordingly, we propose five separate mini-batch selection algorithms, which are explained in section 5.1.2. We also show their empirical effectiveness, in which these mini-batch selection schemes are applied to classification problems on the Oxford 102 [76], Stanford Dogs [49], and Caltech 101 dataset [25].

1.4 Research Objectives

The aims of the project are to:

- i. enhance the performance of domain translation algorithms by improving the design of the current attention mechanism, particularly on the text-to-image generation.
- ii. propose to use a single pair of universal transfer functions to performance domain transfer over more than two image domains.
- iii. study the effects of applying different distributions to model the latent space of GANs other than the traditional standard Gaussian.
- iv. accelerate the convergence of cross domain transfer by applying DPP sampling to mini-batch SGD.

1.5 Thesis Organisation

This thesis is organised as follows:

- *Chapter 1:* This chapter has introduced the background and purpose of the thesis.
- *Chapter 2:* This chapter presents a survey of previous domain transfer algorithms and other literature upon which our work is based.

- *Chapter 3:* This chapter presents our proposed methods in cross domain applications. The first is the enhanced attention structure built between texts and images; the second is domain transfer over multiple image domains.
- *Chapter 4:* This chapter presents latent space modelling with several distributions and examines their performances.
- *Chapter 5:* This chapter presents training acceleration we performed on mini-batch sampling using DPP.
- *Chapter 6:* A summary of the thesis and its contributions are given in the final chapter, along with recommendation for future research.

Chapter 2

Literature Survey

2.1 Introduction

Our work encompasses multiple disciplines, including sentence and image embedding, GAN networks, DPP and computer vision related algorithms. The background knowledge and previous methods that inspire my work are introduced in this section.

2.2 Cross Domain Translation

Image captioning is one of the first areas in cross domain translation that has achieved great success with the assistance of DNNs. The sequence-to-sequence model, in which the input sequence is embedded to a feature vector and fed to the the second network to generate the output sequence, is one of the first architectures applied. An early work [97] designed an end-to-end convolutional network followed by a recurrent neural network. The authors suggested that the network design is inspired by the vanilla sequence-to-sequence structure except that the RNN text encoder is replaced by a CNN image encoder. In [68], another multi-model recurrent based network was used for image captioning. Compared with [97], in which the image feature is only fed to generate the first word, this time the image feature was fed to the decoder in every time frame. Later, the attention mechanism was brought into image captioning. Xu *et al.* [102] added the attention connection between word and regular-grid image region to the original framework [97], and has proved that this way drastically improved accuracy.

These methods are all classic sequence-to-sequence based; later, GANs and VAEs

were used for image captioning, enabling greater possibilities for the generated captions. Dai *et al.* [21] applied conditional GAN to perform the task in which the generator produced descriptions that were learnt jointly with an evaluator that measured how well the description fit the image. In [90], the training objective of the generator changed from reproducing the ground-truth captions to generating captions that are indistinguishable from human-written ones. Jain, Zhang and Schwing [43] designed a VAE based network to generate questions for images, in which the encoder and the decoder are jointly trained to maximise the likelihood of the questions given the images. Wang, Schwing and Lazebnik [98] proposed conditional VAE based methods that explicitly structure the latent z space around components that correspond to different types of image contents and are combined later to create priors for images.

Style specific music composition has achieved great progress before generative models were widely applied in this area. Eck and Schmidhuber [23] applied long short-term memory (LSTM) to generate a form of blue music. A graphical model, DeepBach, was introduced in [33], which generates highly convincing chorales in the style of Bach. However, these networks are limited to certain styles of generation; thus, they are not completely conditional generation. [67] allowed style specific music generation by constructing biaxial LSTM architecture and enforcing music style in the output. MidiNet ([105]) utilised a GAN structure and applied CNNs to both the generator and a discriminator to generate a MIDI score, which is a symbolic music representation, by following a chord sequence.

Research has also been conducted on texts and videos. [77] provided an end-to-end video generation from text through encoding sentences by Long-Short Term Memory (LSTM) and produced video using 3D convolutions. [69] introduced long-term and short-term dependencies between video frames.

2.3 GAN and its Variants

DNN based cross domain translation models are mostly built upon two types of generative models: GAN and VAE [51]. The training target of GANs is to find an equilibrium between the generator and the discriminator; VAEs aim to maximise the lower bound of the data log-likelihood. While several variants of VAEs, such as [92], are also widely employed to perform domain transfer, GANs generates sharper images compared to VAEs. Thus GAN is used as the core of our proposed frameworks. This section describes the theory behind the vanilla GAN and some of its variants.

2.3.1 GAN

GAN has attracted significant attention in the deep learning community for its wide application in many different areas, such as fashion and advertising [115, 66, 107], animation production [44] and music composition [106]. GAN was originally proposed in [31]. It involves a two-player non-cooperative game by the generator G and the discriminator D ; the training target is to find an equilibrium between the two. The generator produces samples from the random noise vector z , and the discriminator differentiates between true samples and fake samples. Concretely, G and D play a minimax game, and the objective function of the two networks is as follows:

$$\begin{aligned} \mathcal{L}^{\text{adversarial}} = \min_G \max_D V(D, G) = & \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] \\ & + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]. \end{aligned} \quad (2.1)$$

The author proved that a global optimum can be achieved when $p_g = p_{\text{data}}$, where p_g is the distribution of the generated samples $G(z)$.

2.3.2 GAN Variants

This section introduces several variants of GAN, from which our work is inspired from.

DCGAN Based on the first proposed GAN network, DCGAN [79] utilises several layers of convolutional neural networks to encode and decode images. Authors have proposed several guidelines to stabilise its training. These guidelines included first, replacing the pooling layer with strided and fractional-strided convolutions; second, applying batch normalisation in both the generator and discriminator were later widely employed by other image generation models.

Conditional GAN Conditional GAN ([71]) has opened up more possibilities for GAN. Unlike the original unconditional GAN, which has no control of the data generated, conditional GAN produces synthetic samples from a specific conditioning factor, such as a label or a sentence, rather than from a noise distribution. The objective function of the minimax game now becomes:

$$\begin{aligned} \min_G \max_D V(D, G) = & \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x|c)] \\ & + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z|c)))]. \end{aligned} \quad (2.2)$$

Here, c is the extra information or the conditioning factor, such as sentence embedding if the task is text-to-image generation.

The first conditional GAN model can generate MNIST digits conditioned on class labels. Researchers have extended conditional GAN to even broader applications, such as conditional face generation [28], future frame prediction [70], and text to image generation.

2.3.3 GAN Based Text-to-Image Generation

The GAN network has been employed in multiple cross domain translation cases, including text to image and video generation, 3D object induction from 2D views and many other applications. As most of our proposed algorithms were applied to text-to-image generation, we review previous achievements made in GAN based text-to-image generation research in Section 2.3.3. Performing text-to-image generation requires feature vector extraction from texts, thus Section 2.3.3 reviews previous work on sentence embedding in GANs.

Sentence Embedding in GANs

Conditional generative models can be obtained by feeding a conditional factor c to both the generator G and the discriminator D . In the case of text-to-image generation, c is a fixed-length vector of the input sentence.

c can be extracted from raw sentences with a sentence embedding technique. Previously used techniques included the pre-trained convolutional recurrent network from [82] and the bidirectional long short-term memory (bi-LSTM) trained end-to-end with the GAN network.

The convolutional recurrent network from [82] was designed to learn representations for image descriptions that have a high compatibility with the matching images. The sentence embedding network is a recurrent network stacked on top of a mid-level temporal convolutional hidden layer. Thus, it retain advantages of both temporal dependency from recurrent nets and scalability of convolution nets.

Bi-LSTM involves two hidden LSTM layers of opposite directions. Therefore, each word corresponds to two hidden states, which are concatenated to represent the word. The last hidden states are concatenated to be the sentence representation.

GAN Based Text-to-Image Generation

GAN-CLS On the basis of DCGAN, several text based GAN networks were proposed. The first GAN based text-to-image generation network was GAN-CLS [83]. It generates images based on the corresponding image caption t in addition to the noise vector z . z is sampled from a Gaussian distribution $z \in \mathbb{R}^Z \sim \mathcal{N}(0, 1)$, and the text description t is encoded with a pre-trained text encoder φ to be $\varphi(t)$. $\varphi(t)$ is then concatenated with z and processed through a series of transposed convolution layers to generate the fake image. In the discriminator D , a series of convolution-batch normalization-leaky ReLU are applied to discriminate between true and fake images.

In addition to the invention of sentence embedding as the conditioning factor, the second major contribution from GAN-CLS was the application of the so-called matching aware discriminator. The original design of the discriminator identified the sample as real or fake. The matching aware discriminator also indicated if the sample was falsely aligned with the given text description. This is vital for future conditional GAN. Later, an explicit metrics is designed to evaluate how well the generated image reflect its text description.

GAWWN The generative adversarial what-where network (GAWWN) [81] was the first architecture that is able to provide controllable text-to-image generation that adopted supplementary information, such as bounding boxes or part locations of the main object in the image. In the case of provided bounding boxes, each box is converted to a mask so that regions in convolution layers that are out of the bounding box are set to zero. It applies to both the generator and discriminator. Researchers have demonstrated that it allows objects to appear in the place of the provided bounding box.

In the case of provided key points, the location key-points are encoded as a spatial feature tensor in which each channel corresponds to one key point. This tensor is fed to the text embedding, the generator and the discriminator. The generated image roughly follows the shape of the provided key points.

StackGAN As previous works failed to generate images with a resolution higher than 128×128 , StackGAN [110] employed a two-stage GAN network to generate photo-realistic 256×256 images from text descriptions. Its architecture consisted of two stages: Stage-I generated a low-resolution image (e.g., 64×64) based on texts and random noises; Stage-II generated a higher resolution image (e.g., 256×256) based on texts and the lower resolution images from Stage-I.

Apart from the two-stage architecture, one particular important idea from this work is the use of a Gaussian latent variable c_0 . It is generated from an independent Gaussian distribution $\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t))$ whose mean and diagonal covariance matrices are functions of the sentence embedding. In Stage-I, instead of applying text embedding to generate images directly, c_0 is processed through the generator in addition to the random noise. A regularisation term is then added to the objective of generator. This term calculates the Kullback-Leibler divergence between $\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t))$ and a standard Gaussian distribution.

This approach was later utilised in other text-to-image generation methods because it mitigated the problem of high dimensional text embedding causing discontinuity in the latent data manifold. This is not desirable in the generator training.

AttnGAN AttnGAN [103] is an extension of StackGAN. Although it also uses multistage GAN design, multiple stages of generator networks are trained simultaneously instead of separately, as in StackGAN. AttnGAN uses an attention mechanism to construct its network in addition to an image-text matching score that encourages

the close matching between the correct pairs of image and sentence. It can generate images of better quality and achieve a higher inception score.

The generation process of AttnGAN is based on a multistage attention mechanism. At each stage, the generated image receives information from attention weights. These attention weights are calculated between image features from the last stage and text features. The attention mechanism is also used to calculate a deep attentional multimodal similarity model (DAMSM) loss, which encourages correct matching between sentence-image pairs.

2.3.4 Attention Mechanism and its Applications in Text-to-Image Generation

Attention mechanism has been widely applied to both classic translation and cross domain translation tasks [3, 102, 103]. It has been proven to enhance the performance significantly, and is used in our proposed work. This section introduces the theory behind the attention mechanism.

Attention mechanism was first proposed in an NMT system [5]. The attention mechanism defines a probability α_{ij} which reflects the importance of an input hidden state j towards the output last hidden state $i - 1$ in terms of generating the next token. The probability is calculated as:

$$\alpha_{i,j} = \frac{\exp(e_{ij})}{\sum_{t=1}^{T_x} \exp(e_{it})}, \quad (2.3)$$

where e_{ij} is a defined score between input hidden state j and output hidden state $i - 1$. T_x is the number of elements in the input.

Therefore, for each output position i , there is a context vector c_i defined as the weighted sum of all input hidden states:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (2.4)$$

where h_j represents the feature vector for the hidden state with index j .

The attention mechanism was applied to text-to-image generation in AttnGAN [103]. It is used to guide the generator to focus on certain words when generating specific image regions.

Self-attention is a specific attention mechanism that computes a sequence representation by relating different positions of the sequence itself [95]. Therefore, no convolution or recurrent process is required to encode a sequence. [95] was the first work to build a sequence transduction models solely on attention mechanisms.

In [95], authors describe an attention function as a mapping query and a set of key-value pairs to the output. The output is calculated as a weighted sum of values, where each weight is computed from the query and the corresponding key. In the case of self-attention, query, keys and values are all transformed representations of the input sequence itself. In particular, [95] presented self-attention as a “scaled dot-product attention”. Given a query Q , keys K and values V , the output is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \quad (2.5)$$

where d_k is the dimension of the query and keys.

Self attention GAN is a recently proposed network that uses the self attention mechanism to improve the generation quality [109]. It adds an additional self attention layer in between regular convolution layers in the original GAN structure.

For a convolution feature map x , it is first fed through three separate 1×1 convolution filters to derive $f(x)$, $g(x)$ and $h(x)$, which are equivalent to Q , K and V

in the sequence self attention. Afterwards, attention probabilities are calculated between each individual regular-grids using $f(x)$ and $g(x)$. A dot product is performed between the resulting attention map and $h(x)$, which generates the output attention feature map. Authors claim that adding such self attention layer improves the generation quality. They also assert that such a self attention layer can be inserted between regular convolutional layers to improve GAN generation results. Later in Section 3.1.5, we compare the image generation performance with or without such layers in our proposed framework.

2.3.5 Text-to-Image Alignment

Alignment between text to image has been studied in multiple works to enhance the “precision” of the generation. That is, the synthetic image should reflect exactly what is described in the caption, or the caption should match exactly the content of an image. As our first contribution focuses on improving the attention mechanism built between texts and images, this section reviews several text-image alignment methods used previous literature.

Region of interest (RoI) pooling Measuring the alignment between texts and images requires extracting feature vectors from both. Text features can be extracted via multiple text embedding algorithms. Extracting regional image features is more difficult when regions have different shapes and sizes, which is why RoI pooling is applied in our work.

RoI pooling was first introduced in the object detection algorithm regional CNN [29]. For an image region with spatial size $h \times w$, it was first divided into $H \times W$ grids of subwindows. Each subwindow was then fed through a max-pooling layer, which derived a final pooling result with spatial size $H \times W$. The RoI pooling allows each image region with various sizes to be embedded into a fixed-length vector with

no additional parameters and training involved.

Text-image alignment Text-image alignment studies the compatibility between images or image subregions with sentences or words; [47] presented a model that generate descriptions of an image. This is one of the first works to design an alignment model that connects words and object-grid image regions.

Object-grid regions in each image are detected via the pre-trained region convolutional neural network (RCNN) [30]. RCNN also provides representations for each object-grid image region detected. Authors chose to use the top 19 detected locations in addition to the whole image and apply linear transformation to these feature vectors. Therefore, each image can now be represented as a series of feature vectors $\{v_i | i = 1, \dots, 20\}$. Word representations are derived from a bidirectional RNN network.

Given the regional and word representations, the measure of similarity between the i -th region and t -th word is given as $v_i^\top w_t$. Thus, the similarity score between an image k and a sentence l is given as:

$$S_{kl} = \sum_{t \in g_l} \max_{i \in g_k} v_i^\top w_t. \quad (2.6)$$

The final alignment objective is defined as a max-margin loss:

$$\mathcal{C}(\theta) = \sum_k \left[\sum_l \max(0, S_{kl} - S_{kk} + 1) + \sum_l \max(0, S_{lk} - S_{kk} + 1) \right]. \quad (2.7)$$

2.3.6 Image Domain Transfers With GANs

CycleGAN [114] performs translation between a pair of image domains using a cycle consistency loss, in the absence of any paired training examples. In addition

to the classic adversarial loss in Equation 2.1, CycleGAN defines a cycle consistency loss as:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{\text{data}(x)}} [\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}(y)}} [\|G(F(y)) - y\|_1], \quad (2.8)$$

where G and F are forward and backward generators between two image domains X and Y . DiscoGAN [50] functions similarly to CycleGAN because it translates between two image domains with unpaired data.

2.3.7 Normalisation Methods and Applications in GANs

GAN training often suffers from mode-collapse and slow convergence. Thus, normalisation methods, such as batch normalisation, are often employed in GAN to stabilise the training [79]. This section reviews several normalisation methods that are commonly applied in DNN training. In Section 3.1.5, the impacts of different normalization methods are examined on the proposed models.

Batch normalisation

Batch normalisation ([41]) was one of the first normalisation techniques applied to accelerate DNN training. The motivation for this is that the distribution of layers' input keeps changing, which forces each layer to continuously adapt to the new distribution. This phenomenon is termed "internal covariate shift". Therefore, batch normalisation is designed to reduce such internal covariate shift.

The implementation of batch normalisation is broken down into two parts. In training, for d -dimensional layer input $x = (x^{(1)} \dots x^{(d)})$, each dimension is normalised as:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}, \quad (2.9)$$

where the expectation and variance are estimated using mini-batches.

In addition, authors have suggested that simply normalising each input of a layer may change what the layer can represent. Therefore, a pair of trainable parameters γ and β is introduced for each activation $x^{(k)}$ to scale and shift the normalised value, so as to restore the representation power of the network:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}. \quad (2.10)$$

During testing, population statistics (which are computed from training mini-batches) are used instead:

$$E[x] = E_{\beta}[\mu_{\beta}] \quad (2.11)$$

$$Var[x] = \frac{m}{m-1} E_{\beta}[\sigma_{\beta}^2]. \quad (2.12)$$

Layer normalisation

Layer normalisation ([4]) is another normalisation technique that accelerates DNNs. It does not use batch statistics to perform the normalisation. Instead, it uses statistics from all units within each layer:

$$\mu^{(k)} = \frac{1}{H} \sum_{i=1}^H x_i^{(k)} \quad (2.13)$$

$$\sigma^{(k)} = \sqrt{\frac{1}{H} \sum_{i=1}^H (x_i^{(k)} - \mu^{(k)})^2}, \quad (2.14)$$

where H is the number of total hidden units in the layer.

In layer normalisation, different training samples can have different normalisation terms (i.e., no dependencies introduced between samples in a mini-batch).

Weight normalisation

Weight normalisation [87] performs a reparameterisation of the parameters in a neural network. Consider a weight vector w ; it is now expressed by a new parameter vector v and a scalar parameter g :

$$w = \frac{g}{\|v\|}v. \quad (2.15)$$

Such reparameterisation decouples the length of weight vectors from their direction. In addition, unlike batch normalisation, it does not introduce dependencies between samples in a mini-batch; thus, it can be applied in recurrent structures.

2.4 Latent Space Modelling of GANs

This section describes distributions other than the standard Gaussian and uniform distributions that have been applied to GANs in previous literature and in the proposed algorithms.

2.4.1 GMM

GMM is one of the most commonly used probabilistic framework for datasets with multiple modes. It assumes that all data points come from a mixture of a finite number of Gaussian distributions. The density function of the GMM is defined as:

$$p_{\mathcal{X}}(x) = \sum_{k=1}^K \alpha_k \mathcal{N}(x; \mu_k, \Sigma_k), \quad (2.16)$$

where K is the total number of Gaussians in the mixture and the k^{th} component is characterised by a Gaussian distribution with weight α_k , mean μ_k and covariance matrix Σ_k . The parameters of the GMM are commonly estimated by the EM algorithm or variational Bayesian inference [59].

2.4.2 GMM in GANs

This section discusses some recent research that is closely related to our work of applying GMM to GANs. In [24], authors proposed to integrate a GMM into the GAN framework. Both of the means and covariance matrices are trainable through the generator loss. Instead of applying the classic adversarial loss as in Equation 2.1, the authors proposed to use GMM likelihood for the minimax game. The discriminator encodes images to feature vectors and estimates the GMM parameters from the encodings of real images. The discriminator is updated in each iteration such that the likelihood of the encoded real images is close to one and the likelihood of encoded synthetic images is close to zero.

In [7], authors also proposed a method named GM-GAN that used Gaussian mixture to model the distribution over the latent space, in addition to its variant for the conditional generation. The supervised GM-GAN modifies the discriminator, so that instead of a single scalar, it returns a vector $o \in \mathbb{R}^N$. Each element of o represents the probability of the given sample being in each class. This ensures that images generated from the generator will be classified by the discriminator as a certain class.

2.4.3 Dirichlet Process

One of the drawbacks of modelling the latent space of GAN using standard GMM is that the number of Gaussians must be predetermined. Therefore, researchers also tried to use Dirichlet process (DP) in generative models.

The DP is a stochastic process which is specified by a base distribution H and a concentration parameter α :

$$G \sim DP(\alpha, H). \tag{2.17}$$

The expectation of the total distinct number of components K in the DPGMM is determined by α and the number of samples M .

$$\mathbb{E}(K) = \alpha(\varphi(\alpha + M) - \varphi(\alpha)). \quad (2.18)$$

The DPGMM can be written as follows:

$$(v_i | \tilde{\mu}_i) \sim N(\tilde{\mu}_i, \sigma^2) \quad (2.19)$$

$$\tilde{\mu}_i \sim G$$

$$G \sim DP(H(\lambda), \alpha),$$

where G is a random prior distribution and $G(\tilde{\mu}_i) = \sum_{k=1}^{\infty} \pi_k \delta_{\mu_k}(\tilde{\mu}_i)$. μ_k are independently distributed according to $H(\lambda)$.

There are multiple approaches to perform DP sampling, our work uses the stick-breaking process, which is introduced below.

Stick-Breaking Process

One way to draw from the DP is performing the stick-breaking process. The probability mass function is defined as:

$$f(\theta) = \sum_{k=1}^K \beta_k \cdot \delta_{\theta_k}(\theta), \quad (2.20)$$

where δ_{θ_k} is the indicator function. The probabilities β_k are calculated as:

$$\beta_k = \beta'_k \cdot \prod_{i=1}^{k-1} (1 - \beta'_i). \quad (2.21)$$

Distribution of the stick weights $\beta_{k=1}^K$ is modelled with the beta distribution. In practice, sampling from a DP mixture is usually performed with variational inference.

Variational Inference

Blei and Jordan [9] developed a mean-field variational algorithm for the DP mixture. The variational bound is written as:

$$\begin{aligned} \log p(x|\alpha, \lambda) &\geq E_q[\log p(V|\alpha)] + E_q[\log p(\eta^*|\lambda)] \\ &\quad + \sum_{n=1}^N (E_q[\log p(Z_n|V)] + E_q[\log p(x_n|Z_n)]) \\ &\quad - E_q[\log q(V, \eta^*, Z)]. \end{aligned} \tag{2.22}$$

The authors proposed the following factorised family of variational distributions to approximate the distribution of the infinite-dimensional random measure G :

$$q(V, \eta^*, Z) = \prod_{t=1}^{T-1} q_{\gamma_t}(v_t) \prod_{t=1}^T q_{\tau_t}(\eta_t^*) \prod_{n=1}^N q_{\phi_n}(z_n), \tag{2.23}$$

where $q_{\gamma_t}(v_t)$ are beta distributions, $q_{\tau_t}(\eta_t^*)$ are exponential family distributions and $q_{\phi_n}(z_n)$ are multinomial distributions.

2.4.4 DP Applications in Generative Models

Nalisnick and Smyth [73] proposed using stick-breaking prior for the variational autoencoder (SB-VAE) instead of the standard Gaussian distribution as in regular VAE. As introduced in Section 2.4.3, sampling using the stick-breaking process requires first sampling from a Beta distribution. However, the beta distribution does not have a non-centered parametrisation. Therefore, authors propose to use another distribution, Kumaraswamy distribution, which is a beta-like distribution with a closed-form inverse CDF:

$$\text{Kumaraswamy}(x; a, b) = abx^{a-1}(1-x)^{b-1}, \tag{2.24}$$

where $x \in (0, 1)$, and a, b are non-negative shape parameters. Samples can be drawn via the inverse transform:

$$x \sim (1 - u^{\frac{1}{b}})^{\frac{1}{a}}, \quad (2.25)$$

where $u \sim \text{Uniform}(0, 1)$. Thus, the hidden representation for VAE is made from the infinite sequence of stick-breaking weights instead of drawing from the standard Gaussian.

Another work [72] applied Dirichlet Process mixtures to model the latent space of VAE. The generative process becomes: $\pi_i \sim \text{Dir}(\alpha)$, $z_i \sim \sum_{k=1}^K \pi_{i,k} N(z, \theta_k)$, $x_i \sim p_\theta(x|z_i)$ where $p_\theta(x|z_i)$ is a density network.

Our work is by far the first to model the latent space of GANs using DPGMM. Different from VAEs, the architecture of vanilla GANs does not allow the latent vectors to be learnt directly from the data itself. Therefore, specific approaches are proposed to properly update the parameters of the latent distribution and provide a better fit to the data.

2.4.5 Other Latent Space Modelling

[10] models the latent space as a unit sphere and learns the correspondence from the sphere space to the data space without the adversarial loss. Such a trained model can achieve smooth linear interpolation output between any two random vectors on the unit sphere. This means that linear interpolations in the noise space lead to semantic interpolations in the generated images.

[16] proposed InfoGAN which in addition to the adversarial loss, it learns to maximise the mutual information between a subset of latent variables and the observation, i.e. $I(c; G(z, c))$, where c is the class of the real sample. A learnt InfoGAN model can disentangle discrete and continuous latent factors. Thus, linear interpo-

lation can be performed using the continuous latent code (e.g., from a thin digit to a wide digit); this is not possible for discrete codes, such as across categories of images.

2.5 Mini-Batch and SGD

Mini-batch and SGD are common tools used in machine learning algorithms. Often, the size of training data becomes so large that it is impractical to render batch optimisation (i.e., using the entire dataset in every iteration). For example, [53] trained a CNN to classify 1.2 million images from ImageNet, while [93] built a sequence-to-sequence neural translation model with 12 million sentences that contained 348 million French words and 304 million English words from the WMT'14 English to French dataset. Therefore, mini-batch training can be employed to reduce the communication cost and parallelise the learning process [61].

Despite its wide usage, the most standard form of selecting mini-batches in SGD is to sample data indices uniformly in each iteration and perform an update as follows:

$$w_{t+1} \leftarrow w_t - \frac{\eta}{n} \sum_{i=1}^n \nabla \phi_i(w_t), \quad (2.26)$$

where w_t is the parameter at time step t , ϕ is the loss function, η is the learning rate and n is the size of the mini batch. Standard SGD employs uniform sampling, so the stochastic gradient is an unbiased estimate of the true gradient. Nonetheless, it introduces variance between iterations, which negatively affects the performance [113].

Using large mini-batches can mitigate the problem by reducing the variance but this slows down the actual convergence [13]. Several attempts have been made to optimise the performance of mini-batch training; [61] added a conservative constraint

to the loss function to control differences between parameters across iterations:

$$w_t = \operatorname{argmin}[\phi(w) + \frac{\gamma}{2}\|w - w_{t-1}\|_2^2]. \quad (2.27)$$

Johnson and Zhang [45] proposed a method named stochastic variance reduced gradient (SVRG) that replaced the training target with a new random vector that had the same expectation but a smaller variance. Alain *et al.* [2] applied distributed importance sampling, in which the sampling weight was proportional to the L2-norm of the gradient. Gopal [32] proposed a similar idea by separating the data into bins using side-information and maintaining the distribution at a class level rather than at an instance level. Other researchers reduced variances by modifying sampling methods. These methods are introduced in section 2.5.4. Section 2.5.4 briefly outlines the DPP, which is the basis of our work in accelerating model convergence.

2.5.1 DPP

DPP is a random process used to model a subset \mathbf{Y} selected from a base set \mathcal{Y} . In particular, DPP encourages \mathbf{Y} to contain a diverse set of items; therefore, it is applied to some sampling and summarisation tasks in which diversity is preferred (e.g., detecting people and their poses in an image) [55].

Consider a random subset \mathbf{Y} drawn from \mathcal{Y} according to \mathcal{P} , for every subset $A \subseteq \mathcal{Y}$:

$$\mathcal{P}(\mathbf{Y} \supseteq A) = \det(K_A), \quad (2.28)$$

where K is called the marginal kernel which is a real, symmetric, positive semidefinite $N \times N$ matrix indexed by elements of \mathcal{Y} .

DPPs can be constructed alternatively with a real, symmetric matrix L indexed by the element of \mathcal{Y} instead of the marginal kernel K , in which case L is called L-ensembles. Let L be the Gram matrix, constructed from B , the feature vector for

each element in the base set \mathcal{Y} , i.e., $L = B^\top B$:

L -ensembles directly specifies the probability for a possible subset A as follows:

$$\mathcal{P}_L(\mathbf{Y} = A) = \frac{\det(L_A)}{\det(L + I)}, \quad (2.29)$$

where L_A is a part of L that is indexed by elements in A .

2.5.2 K-DPP

k-DPP models the distribution over subsets with size k from \mathcal{Y} [54], which ensures that the number of data sampled is of a fixed size.

$$P_L^k(Y) = \frac{\det(L_Y)}{\sum_{|Y'|=k} \det(L_{Y'})}. \quad (2.30)$$

2.5.3 Markov DPP

Markov DPP further encourages diversity between two subset selections. Suppose Y_t is the subset sampled at timestep t and Y_{t-1} is the subset sampled at timestep $t-1$, Markov DPP tries to maximise the conditional probability of sampling Y_t given Y_{t-1} . In other words, it tries to make the sample diverse from previous samples as shown in Figure 2.1.

The sampling strategies for normal DPP sampling, k-DPP sampling and Markov k-DPP sampling are demonstrated in [55] and [1].

2.5.4 DPP for Mini-Batch Sampling

Zhao and zhang [112] proposed an algorithm named SGD with stratified sampling (SGD-ss) that applied clustering to separate the dataset into clusters before uniformly sampling for mini batches within each cluster. The method requires data in each cluster to belong to the same category. The weight update formula now becomes:

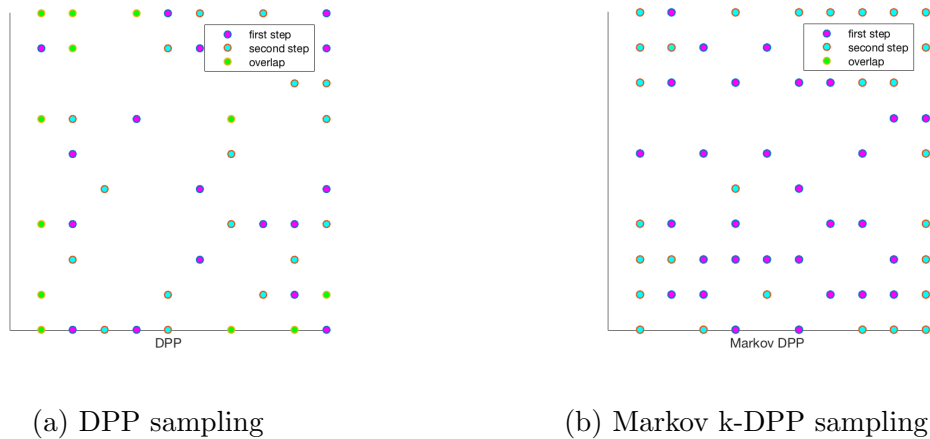


Figure 2.1 : In both figures, magenta points are drawn from k-DPP. Cyan points are its subsequent draw. Green points are samples selected in both steps, where (a) is using an independent k-DPP, and (b) is drawn from a Markov k-DPP, conditioning the first.

$$w_{t+1} \leftarrow w_t - \frac{\eta}{n} \sum_{s=1}^k \sum_{s=1}^b \nabla \phi_s(w_t), \quad (2.31)$$

where k is the total number of clusters and b is the number of samples to be selected within each cluster.

While most research in SGD variance reduction assumes the loss function to be convex and smooth [89], recent works by [45] and [80] also demonstrate that SVRG can be applied to non-convex functions. Although these works can theoretically accelerate the training process, there is still the requirement for an underlying data distribution to be known for the weight updates.

To improve the efficiency of SGD convergence without the assumption of any underlying data distributions, [108] proposed a strategy called DM-SGD in applying k-DPP to mini-batch sampling. The authors demonstrated that applying DPP sampling guarantees variance reduction and thereby provides faster convergence

rates relative to mini-batch SGD.

Given FC feature vectors W and corresponding one-hot labels H , DM-SGD calculates the feature vector for each data sample as a weighted concatenation between the two (i.e., feature vectors) are calculated as:

$$F = [(1 - w)W \quad wH], \quad 0 \leq w \leq 1. \quad (2.32)$$

The Gram matrix L for the sampling is defined as $L = FF^\top$. In addition, authors proved that for all data points x_i and x_j and all parameters θ , if:

$$\forall_{i \neq j} : C_{ij}g(x_i, \theta)^\top g(x_j, \theta) < 0. \quad (2.33)$$

Then DM-SGD has a lower variance than that of SGD. Here, g represents the gradient, C_{ij} is the correlation between the two data points, which is negative when data points are similar and positive when dissimilar. Therefore, applying k-DPP sampling guarantees variance reduction.

Authors have demonstrated the performance on several datasets including the MNIST [56] and Oxford Flower 102 datasets [75]. In the MNIST dataset, authors used the raw image pixels to define feature vectors. In the Oxford Flower 102 dataset, the feature vector for each image was defined using the first FC layer of the pre-trained VGG-16 network [91]. Authors then trained a linear softmax classification with off-the-shelf CNN features.

Chapter 3

Applications of Cross Domain Transfer

As explained in Chapter 1, the current attention mechanism built for cross domain transfer has certain limitations. Take text to image as an example, the attention structure extracts regular-grid region features and word embeddings to decide which words to examine when generating each image region. This does not work well when the images contain multiple objects and there are complex interactions between them because a single word does not provide enough descriptions for an object and a regular-grid region can contain a part of one or several objects.

Therefore, our work proposes building the attention structure on the phrase and object-grid region instead. To make this work, we also proposed novel mechanics to extract phrase features using a two-layer LSTM structure.

Another drawback of the cross domain algorithms is that they are usually designed to manage a pair of domains and a new set of parameters is required for a different pair of domains. This is time consuming and the network can be inevitably large when there are more than two domains involved in the system.

Thus, this chapter also introduces our second contribution, which proposes using a single pair of universal transfer functions to perform transfer across multiple domains. Our work manages datasets that contain “natural ordering”; such order is learnt during the training without any supervision. The details for both contributions are explained in Section 3.1 and 3.2.

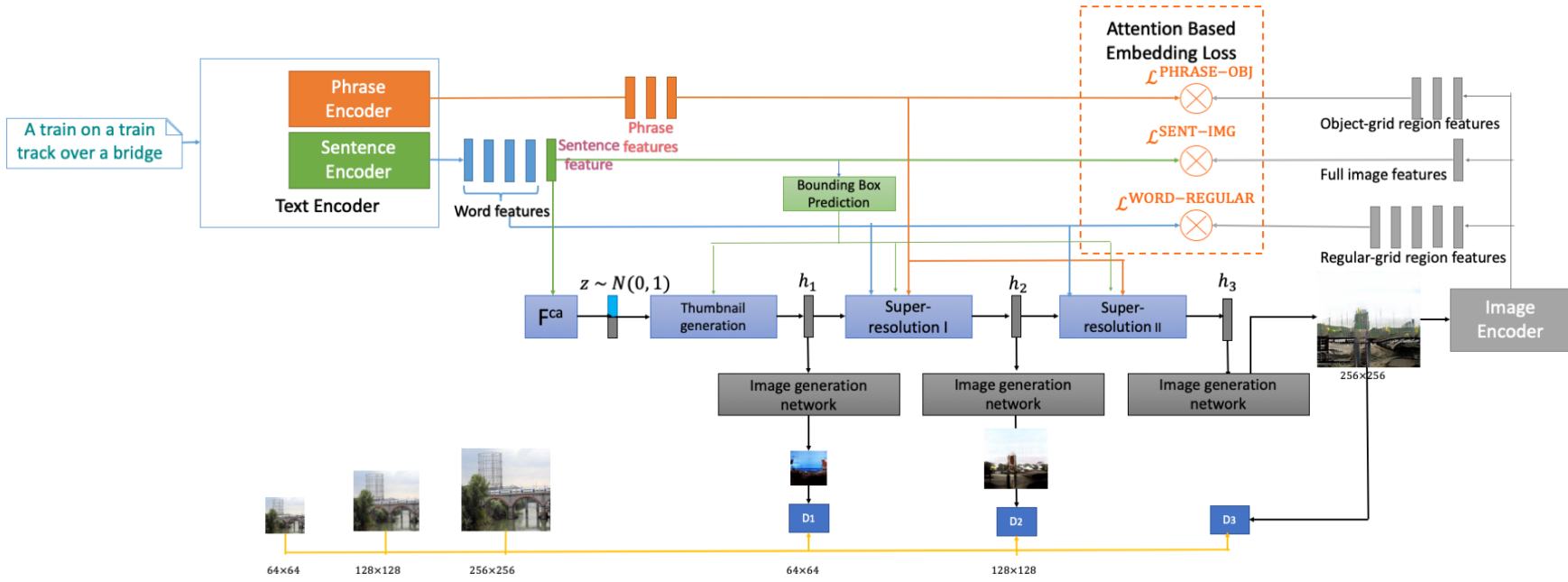
3.1 Realistic Image Generation using Region-Phrase Attention*

Our network is inspired by several recent studies, including those of the architecture of AttnGAN and the visual-semantic alignment [47]. Our work differs significantly because we introduce the attentions between phrase and object-grid features into the network. Compared with Obj-GAN, our framework introduces the phrases and the overall design is much simpler.

To this end, we show our overall model in Figure 3.1, which consists of an end-to-end text encoder network, a GAN framework and a bounding boxes prediction framework. The details for each module are explained in the following sections.

*This work was published as [38].

Figure 3.1 : Network structure for text embedding and GAN network.



3.1.1 Text Encoder

The text encoder of current text conditioned GAN network typically extracts a whole sentence representation and word representations using a bidirectional LSTM. In addition to those features, our proposed work also extracts the phrase features to be fed into our algorithm.

We define a phrase as a combination of the closest article (digit), adjective and noun. Such information can be extracted from applying part-of-speech tagging (POS-tagging) to raw sentences. For example, a sentence *“Two black horses standing with a cart attached to them.”* is tagged as [(“Two”, digit), (“black”, adjective), (“horses”, noun), (“standing”, verb), (“with”, preposition), (“a”, article), (“cart”, noun), (“attached”, verb), (“to”, preposition), (“them”, pronoun)]. We then group the nearest article (digit)-adjective-noun words as a phrase, which yields *“two black horses”* and *“a cart”*.

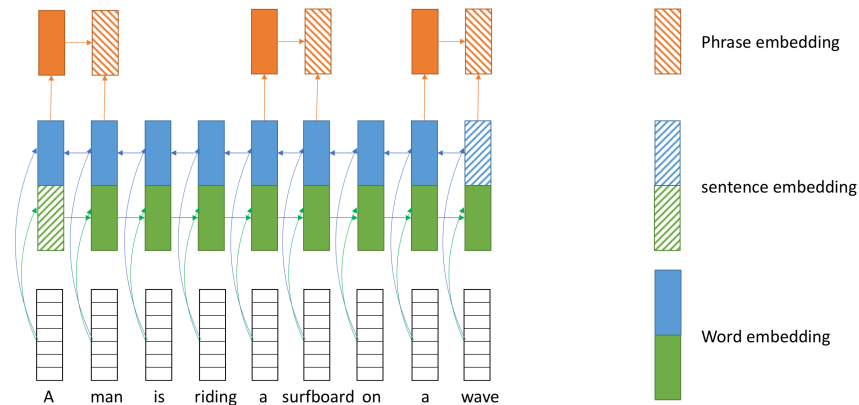


Figure 3.2 : Text embedding with a two-layer LSTM networks.

The full text encoder framework is shown in Figure 3.2, which in fact can be considered as a phrase encoder built on top of a sentence encoder. In addition, our design incorporates object-grid image features to assist the learning. Such image

features are extracted from an image encoder. The details are explained in the following sections.

Sentence Encoder

First, a bidirectional LSTM is applied to each sentence to extract word and sentence representations. Given a sentence $\{w_1, \dots, w_T\}$, the t^{th} word representation e_t is a concatenation of a forward e_t^f and a backward hidden state e_t^b , i.e., $e_t \equiv [e_t^f \ e_t^b]$. The full sentence embedding \bar{e} is defined using the last hidden states (i.e., $\bar{e} \equiv [e_T^f \ e_T^b]$).

Phrase Encoder

On top of the extracted word representations e where $e \equiv \{e_1, \dots, e_T\}$, phrase representations are extracted by applying a second LSTM in the following way. Given the t^{th} phrase, an LSTM is applied over the sequence of words in the phrase. The last hidden state is used as its feature representation which we refer to as p_t . An alternative way to extract phrase embeddings is by taking the average of word embeddings in each phrase. The performance of both methods is discussed in Section 3.1.5.

Our phrase-based embedding clearly has an advantage over the traditional word-based mechanism in which each word has one representation. For example, none of the *individual* words in the phrase “*a green apple*” portray an overall picture of the object; all three words work together to capture its visual meaning.

An Alternative Text Encoder Design

Apart from the proposed stacked LSTM text encoder, we design an alternative text encoder that is based on convolution layers and self-attention mechanism (i.e., no recurrent structure is required.) The network structure is shown in figure 3.3.

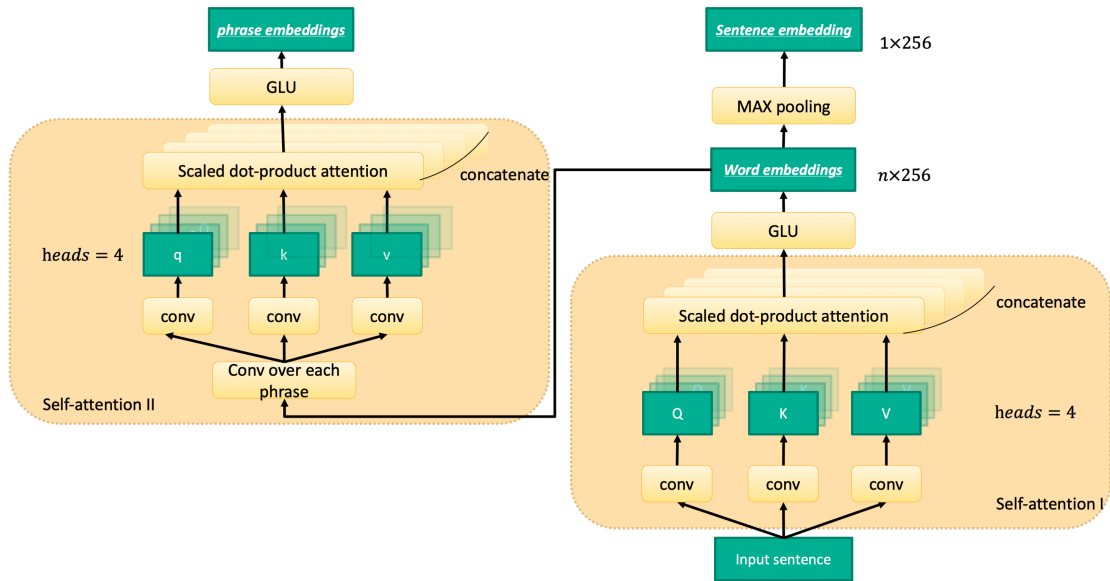


Figure 3.3 : Text embedding with CNN layers. n represents the number of words in the sentence. “heads = 4” means four scaled dot-product attention run in parallel.

It follows a similar logic as the LSTM version. The input sentence is fed through three separate convolution layers to generate different versions of query Q , key K and value V . Details on convolution operations on sequences are provided in Section 2.3.3. In this step, we apply the first multihead attention operation as the rectangular area “Self-attention I” shown in the figure. A gated linear unit (GLU) operation is applied to the concatenated attention outputs to derive the word embeddings. It is followed by a max pooling operation to derive the sentence embedding.

Phrase embeddings are derived via a second multihead attention from the word embeddings, which works in a very similar logic to the first one.

In addition, in each convolution operation in this network, we apply a normalisation function to the input before feeding it to the convolution. We compare between three methods: weight normalisation, layer normalisation and batch normalisation. Results are shown in Section 3.1.5.

Image Encoder

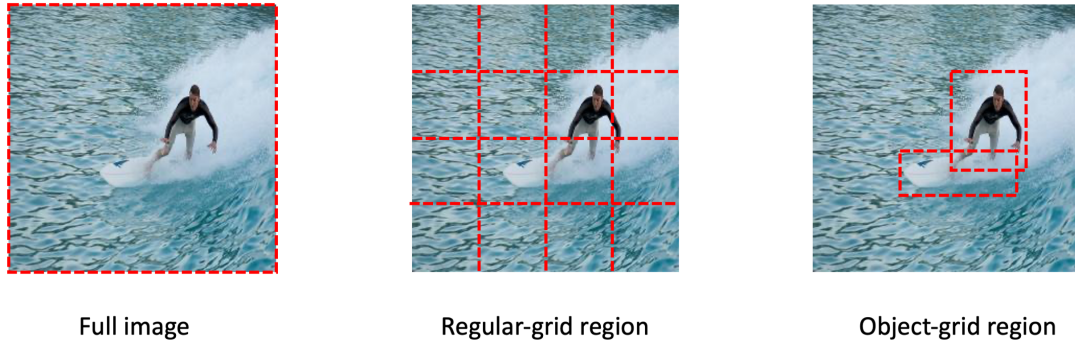


Figure 3.4 : Examples of full image region, the regular-grid region and object-grid region

The image encoder itself comes from the pre-trained Inception-v3 network [94] and is not further fine-tuned in our framework. We apply the image encoder to extract three types of image features from a single image: an *object-grid region feature*, a *regular-grid region feature* and a *full image feature*. As shown in Figure 3.4, an object-grid region is defined over a single object. Thus, the regions differ in sizes. Regular-grid regions have equal sizes and each of them can contain half an object or multiple objects.

Common to all features, each image first undergoes the Inception-v3 model. We use the last feature layer (i.e, “*mixed_6e*”) layer as the designated layer for the regular-grid region. The full image feature is obtained from the last average pooling layer. In addition, both the *regular-grid region feature* and *full image feature* are converted into vectors in the same semantic space using a trainable Fully Connected (FC) layer. Therefore, the resulting features have the following dimensions: a regular-grid region feature $v \in R^{289 \times D}$ where $289 = 17 \times 17$ is the dimension for the “*mixed_6e*” layer feature map. The image feature is denoted as $\bar{v} \in R^D$.

To obtain an *object-grid region feature*, the location and size of each region must

first be identified. In several open datasets, such as the Microsoft COCO (MSCOCO) dataset, the manually labelled bounding boxes of object(s) within an image are readily available. In cases in which the dataset does not provide such information, they can also be obtained from off-the-shelf image object detectors, such as RCNN [30]. This makes it possible to apply our algorithm to any image datasets with text annotations, including the CUB and the Oxford Flower 102 datasets.

The “*mixed_6e*” layer feature map and its bounding box information is fed through the Region of Interest (RoI) pooling to generate its object-grid region feature. These features are fed through a convolution operation with a kernel of an equivalent size, resulting in a vector in a common semantic space as text features. We denote the object-grid region feature as $\mathbf{b} \in R^{K \times D}$ where K is the number of bounding boxes in each image.

Attention-Based Embedding for Text

Text embedding and the perception layer for image and region features are bootstrapped prior to training the GAN. The training requires an overall loss function, which is defined as Equation 3.1:

$$\mathcal{L}^{\text{TEXT}} = \mathcal{L}^{\text{SENT-IMG}} + \mathcal{L}^{\text{WORD-REGULAR}} + \mathcal{L}^{\text{PHRASE-OBJ}}. \quad (3.1)$$

The above loss function comprises three separate losses; each of these follows [103]. Therefore, without loss of generality, the target of the loss function is to minimise the negative log posterior probability for the correct image-sentence pair. That is, for a batch of M image-sentence pairs $(S_i, I_i)_{i=1}^M$, for clarity, we drop the subscript for \mathcal{L} :

$$\mathcal{L} = -\sum_{i=1}^M (\log P(S_i|I_i) + P(I_i|S_i)), \quad (3.2)$$

where $P(S_i|I_i)$ is the conditional probability for a text data S_i to be matched with an image data I_i defined as:

$$P(S_i|I_i) = \frac{\exp(\gamma_1 \mathcal{R}(S_i, I_i))}{\sum_{q=1}^M \exp(\gamma_1 \mathcal{R}(S_q, I_i))}. \quad (3.3)$$

Here, $\mathcal{R}(S_i, I_i)$ gives the similarity score between the text and the image data. Text may refer to *sentence*, *word* or *phrase*, and the image may refer to their corresponding “entire image”, “regular-grid region” and “object-grid region” respectively. γ_1 is a manually defined smooth factor. The posterior probability $P(I_i|S_i)$ for an image being matched to a sentence is defined similarly.

The similarity score $\mathcal{R}(S_i, I_i)$ can be defined in multiple ways using off-the-shelf methods from the statistics community to suit each situation. In our work, we apply three \mathcal{R} values: $\mathcal{R}^{\text{SENT-IMG}}$, $\mathcal{R}^{\text{WORD-REGULAR}}$ and $\mathcal{R}^{\text{PHRASE-OBJ}}$ to Equation 3.2 and Equation 3.3, which derives three corresponding loss values $\mathcal{L}^{\text{SENT-IMG}}$, $\mathcal{L}^{\text{WORD-REGULAR}}$ and $\mathcal{L}^{\text{PHRASE-OBJ}}$.

Choices of \mathcal{R} for $\mathcal{L}^{\text{SENT-IMG}}$, $\mathcal{L}^{\text{WORD-REGULAR}}$ and $\mathcal{L}^{\text{PHRASE-OBJ}}$

$\mathcal{L}^{\text{SENT-IMG}}$ describes the similarity between text and image. We have chosen $\mathcal{R}^{\text{SENT-IMG}} = \phi(S_i, I_i)$ to be the cosine similarity between a sentence representation \bar{e} and a whole image feature \bar{v}_i .

$\mathcal{L}^{\text{WORD-REGULAR}}$ utilises the attention mechanism built between the regular-grid regions and the words. Its similarity score is chosen as $\mathcal{R}^{\text{WORD-REGULAR}} = \log\left(\sum_t^T \exp(\gamma_2 \phi(c_t, e_t))\right)^{\frac{1}{\gamma_2}}$. Here, T is the total number of words and γ_2 is a second smooth factor. $\phi(c_t, e_t)$ is the cosine similarity between a word embedding e_t and a regular-grid-region-context vector c_t . c_t is calculated as a weighted sum over regular-grid image features: $c_t = \sum_{j=0}^{289} \alpha_j v_j$, where α_j is the attention weight for the j^{th} regular-grid towards the t^{th} word and $\alpha_j = \frac{\gamma_3 \phi(c_t, e_t)}{\sum_k^{289} \exp(\gamma_3 \phi(c_t, e_k))}$.

$\mathcal{L}^{\text{PHRASE-OBJ}}$ is defined using the attention mechanism between the object-grid regions and the phrases. The similarity score is $\mathcal{R}^{\text{PHRASE-OBJ}} = \log\left(\sum_{t'}^K \exp(\gamma_2 \phi(c_{t'}, p_{t'}))\right)^{\frac{1}{\gamma_2}}$ where $c_{t'}$ is the object-grid-region context vector. $c_{t'}$ is calculated in a similar way as c_t except that it is a weighted sum over the object-grid region features instead of regular-grid features.

Another alternative to define \mathcal{L} is using the attention connection between the object-grid regions and words instead of phrases. We denote such a loss value $\mathcal{L}^{\text{WORD-OBJ}}$.

3.1.2 Overall Text Embedding Loss

Having defined the four loss values above, we applied them to our overall network architecture, as shown in Figure 3.1. In Section 3.1.5, we illustrated the performance of different combinations of \mathcal{L} . This further demonstrates that object-grid region and phrases are important conditional information in image generation.

3.1.3 Attentional Text-to-Image Generation

Inspired by previous network designs, our work constructs text-to-image generation as a multistage process. At each generation stage, images from small to large scales are generated from corresponding hidden representations. The first stage was named as “thumbnail generation” which takes sentence embedding \bar{e} as the input and generates images with the lowest resolution. In the following stages, images with higher resolution are generated from the hidden state of the last stage with an attention-based structure.

Thumbnail Generation

Thumbnail generation was inspired by the bounding box conditioned sentence to image design by GAWWN and was modified to suit our network design. The

generator structure is shown in Figure 3.5, while the discriminator structure is shown in Figure 3.6.

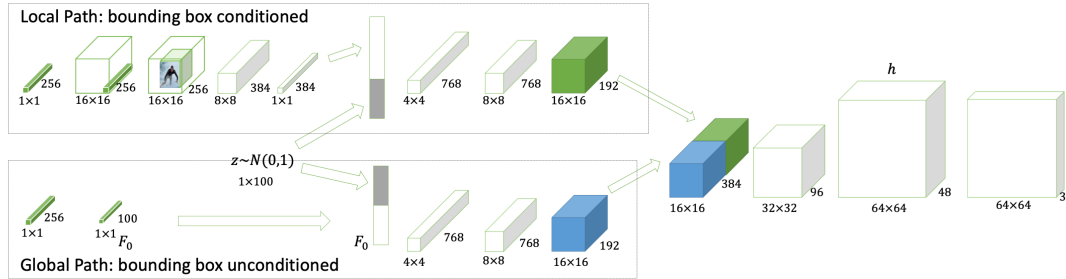


Figure 3.5 : Thumbnail generator

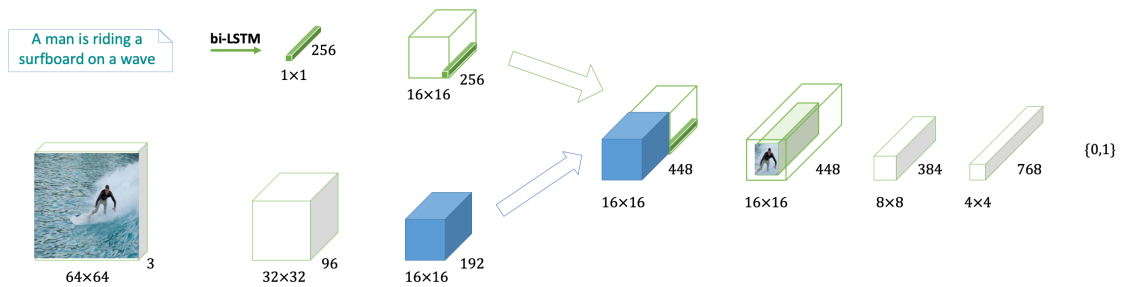


Figure 3.6 : Thumbnail bounding box conditioned discriminator

The generation process branches into two paths. The global path, which is not bounding box conditioned, takes the conditioning factor F_0 and the noise vector to produce a global feature tensor. F_0 itself is a Gaussian latent variable whose mean and diagonal covariance matrix come from F^{ca} which is a function of the sentence embedding. The local path instead uses the sentence embedding directly. It first combines the sentence embedding and the bounding box by spatially replicating e and zero out the region out of the bounding box. In cases in which multiple bounding boxes coexist, the resulting tensor is averaged. The local path then takes

the combined tensor through another several layers to generate a local feature tensor. Tensors from both paths are concatenated depth-wise to derive the first hidden representation h_1 .

In terms of the discriminator, one naive approach to incorporate the bounding box information is to follow GAWWN, through which features extracted from an image is to concatenate with the features extracted from the image-bounding boxes pair. The discriminator then evaluates this concatenated vector. However, the experiment results provided an unfavourable outcome using this approach. In our work, we introduce a three discriminators approach. The details are discussed in Section 3.1.3.

Super-Resolution I & II

Super-resolution enlarges the previously generated thumbnails by constructing the attention mechanism between the last hidden state and text features. At stage n , a hidden representation h_n is constructed from the last hidden state h_{n-1} . h_n is later translated to an image with the *image generation network* in Section 3.1.3.

We incorporate two sets of attentions in the proposed framework. The first is between individual words and regular-grid regions. The second is between phrases and object-grid regions.

Given the word embeddings \mathbf{e} where $\mathbf{e} \equiv e_1, \dots, e_T$ for T words in a sentence and phrase embeddings \mathbf{p} where $\mathbf{p} \equiv p_1, \dots, p_{T'}$ for T' phrases in a sentence, h_n is calculated as:

$$h_n = F_n(h_{n-1}, F_n^{attn1}(\mathbf{e}, h_{n-1}), F_n^{attn2}(\mathbf{p}, h_{n-1})). \quad (3.4)$$

Here, F_n is a deep neural network that constructs the hidden representation h_n from given inputs. F^{attn1} and F^{attn2} are the DNNs that construct the word-context

matrix and phrase-context matrix respectively.

The word-context matrix is constructed from word embeddings \mathbf{e} and regular-grid image region features from h_{n-1} . \mathbf{e} are first fed through a perceptron layer to be converted into the common semantic space as image features. The regular-grid region is defined here in a similar way to that explained in Section 3.1.1, except that the input feature map is not from the pre-trained Inception-v3.

Given j^{th} regular-grid region feature h_{n-1}^j , a word-context vector c_j is defined as the weighted sum over word embeddings:

$$c_j = \sum_t^T \varphi_{j,t} e_t. \quad (3.5)$$

Here, $\varphi_{j,t}$ is the attention weight between the t^{th} word and the j^{th} regular-grid region and $\varphi_{j,t} = \frac{\exp(h_{n-1}^j \top e_t)}{\sum_r \exp(h_{n-1}^j \top e_r)}$. Suppose there are J regular-grids, the final word-context matrix is then defined as the union of the c_j value for each regular-grid region, i.e., $F_n^{\text{attn1}}(\mathbf{e}, h_{n-1}) = (c_1, \dots, c_J)$.

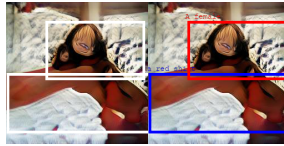
This phrase-context matrix is calculated in a similar way, except that word embeddings are replaced with phrase features and regular-grids are replaced with object-grid features. Here, object-grid features are derived from h_{n-1} by feeding it through the RoI pooling.

The resulting phrase-context matrix is of length K , where K is the number of object-grid regions defined in the image. To apply such a matrix to the network, we let each pixel inside the bounding box carry the same phrase-context vector, while pixels outside of bounding box carry zeros. As for regions in which multiple bounding boxes overlap, the phrase context vectors are averaged. Thus, the resulting phrase-context matrix is of the same shape as the previously defined word-context matrix.

Figure 3.7 illustrates examples of attention-weights mapping, from object-grid image regions to phrases. It is clear to see that our framework encourages the correct text information to be focused in generating each key objects.



(a) A picture of a stop and go light with a stop sign next to it.



(b) A female wearing a red shirt lies on a bed, resting.



(c) A metal counter topped with lots of cheesy pizzas.

Figure 3.7 : Example of attention being paid to a phrase when generating each object-grid region. White rectangles on the left figure highlight the object-grid regions in the image. The matched pair of phrase and object-grid image region is highlighted in the right image.

Image generation network: Hidden representation to images

As shown in Figure 3.1, the previous thumbnail generation and super-resolution stages do not produce images directly. Instead, they produce hidden representations that are fed through an additional convolution layer using kernel size and a depth dimension 3 to generate images.

Discriminators

In general, we use three types of discriminators. The first evaluates an entire image as being real or fake, which is named it D^{im} . The second evaluates a pair of image and sentence, which is termed $D^{\text{im-txt}}$. The third evaluates a group of image, sentence and bounding boxes, which is named $D^{\text{im-txt-bnd}}$. Collectively, the proposed discriminator set is: $\mathcal{D} \equiv \{D^{\text{im}}, D^{\text{im-txt}}, D^{\text{im-txt-bnd}}\}$

In addition, we incorporate the logic of matching-aware discriminator from [83], in which the latter two discriminators were fed through real, fake and unmatched samples. The value function for the generator and the discriminator at each stage is given below where we denote the bounding box information as b :

$$\begin{aligned} \min_G \max_D V(\mathcal{D}, G) = & \mathbb{E}_{x_i \sim p_{data}(x_i)} [\log \mathcal{D}(x_i)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - \mathcal{D}(G(z|\bar{e})))] \\ & + \mathbb{E}_{x_i \sim p_{data}(x_i)} [\log \mathcal{D}(x_i, \bar{e})] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - \mathcal{D}(G(z|\bar{e}), \bar{e}))] \\ & + \mathbb{E}_{x_i \sim p_{data}(x_i)} [\log \mathcal{D}(x_i, \bar{e}, b)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - \mathcal{D}(G(z|\bar{e}), \bar{e}, b))]. \end{aligned} \quad (3.6)$$

In Table 3.1 we report the detailed network architecture for the discriminator performed on the smallest 64×64 images. $D_d = 96$, $D_e = 256$ are the chosen hyper-parameters. f_D refers to features produced in the network. Upsampling consists of a nearest neighbour image resize, a convolution, a batch normalisation and a GLU layer. Down-sampling consists of a convolution, a batch normalisation and a leaky ReLU layer. The kernel size and the stride value used in both operations are shown in the parentheses. Identical function/framework applies to all stages with deeper network designs on larger images. The full network architecture can be found in the appendix.

3.1.4 Bounding Box Prediction

As the image generation relies on bounding box information which is not available in the testing phase, a separate bounding box prediction network is trained based on the sentence embedding. We define two prediction tasks in the network. The first predicts the coordinates for bounding boxes; the second predicts the total number of bounding boxes described in the sentence. Both predictions are structured as regression problems. Therefore, a sentence embedding is first fed through two multilayer neural networks, in which the final layer of both networks is a mean

Stage	Sub-stage	Name	Input Tensors	Output Tensors
Image + Sentence Discriminator		Convolution + leaky ReLU	$64 \times 64 \times 3$	$32 \times 32 \times D_d$
		Down-sampling (kernel=4, stride=2) $\times 3$	$32 \times 32 \times 96$	$f_D^{IMG}(4 \times 4 \times (D_d \times 8))$
	$\mathcal{D}(x)$	Convolution (Image only logits)	$4 \times 4 \times (D_d \times 8)$	1
	$\mathcal{D}(x, \bar{e})$	\textbf{Sentence conditioned logits}	f_D^{IMG}, \bar{e}	1
Image + Sentence + Bounding Box Discriminator		Convolution	$64 \times 64 \times 3$	$32 \times 32 \times D_d$
		Down-sampling (kernel=4, stride=2)	$32 \times 32 \times D_d$	$f_D^{IMG^2}(16 \times 16 \times (D_d \times 2))$
		Spatial Replicate	\bar{e}	$16 \times 16 \times D_e$
		Concatenation	$16 \times 16 \times D_e, f_D^{IMG^2}$	$16 \times 16 \times (D_e + D_d \times 2)$
		Apply bounding box mask	$16 \times 16 \times (D_e + D_d \times 2)$	$16 \times 16 \times (D_e + D_d \times 2)$
		Down-sampling (kernel=4, stride=2) $\times 2$	$16 \times 16 \times (D_e + D_d \times 2)$	$f_D^{IMG-BBOX}(4 \times 4 \times (D_d \times 8))$
	$\mathcal{D}(x, \bar{e}, b)$	\textbf{Sentence conditioned logits}	$f_D^{IMG-BBOX}, \bar{e}$	1
Sentence conditioned logits		Spatial replicate	\bar{e}	$4 \times 4 \times D_e$
		Concatenation	f_D^{IMG} or $f_D^{IMG-BBOX}, \bar{e}$	$4 \times 4 \times (D_e + D_d \times 8)$
		Down-sampling (kernel=3, stride=1)	$4 \times 4 \times (D_e + D_d \times 8)$	$4 \times 4 \times (D_d \times 8)$
		Convolution	$4 \times 4 \times (D_d \times 8)$	1

Table 3.1 : Network architecture for the discriminators on the 64×64 images.

squared error of the predicted value and the real value.

Our work adopts several processing steps for the data in the following manner. First, the coordinates of bounding boxes are normalised to the proportion of the full length, so that the maximum value is 1 regardless of the size of the bounding box or the image. Second, given a predicted number of bounding boxes, coordinates for the bounding boxes that outnumber the predicted value are considered “invalid”, and thus, are excluded in computing the loss. In addition, we define words such as “left”, “right” as position related words. Our work later compares the performance between using all sentences for the training and using only sentences that contain position related words.

3.1.5 Experiments

Below, we demonstrate the superior performance of our work, and the performance of each of the proposed components (i.e., the text encoder, the GAN network and the bounding box predictor from Sections 3.1.1, 3.1.3 and 3.1.4 respectively).

The dataset we used is the MSCOCO dataset, which includes various images that

involve natural scenes and complex object interactions. It contains 82,783 images for training and 40,504 for validation. Each image has five corresponding captions. Bounding boxes are provided for objects in 80 categories.

The text encoder is trained over 150 iterations and the learning rate is set as 0.0002. This pre-train phase is followed by the GAN training when the fine-tuned text encoder is used to train the GAN network over 120 iterations and the learning rate for both the generator and discriminator is set as 0.0002. Three metrics, inception scores, FID score and R-precision were utilized to perform the evaluation.

Metrics: Inception score, FID score and R-precision

It is difficult to measure the performance of image generation in a quantitative way. Thus, the *inception score* [86] and the FID score [35] were two popular metrics for automatic image quality evaluation.

The inception score is calculated as:

$$I = \exp(\mathbb{E}_x D_{\text{KL}}(p(y|x)||p(y))), \quad (3.7)$$

where x is a generated image and y is the label predicted by the inception model [94]. A higher inception score indicates that the generated images contain clear objects and include a high diversity of images (i.e., a better generation quality).

Salimans *et al.* [86] suggested that a good model should be able to generate diverse images that contain meaningful objects. Thus, the conditional label distribution $p(y|x)$ should have a low entropy while the marginal $\int p(y|x = G(z))dz$ should have a high entropy. As suggested in [86], the metric should be evaluated on a sufficiently large number of samples. Thus, the inception is collected from the 30,000 random validation images of the MSCOCO dataset.

The other metric, the FID score measures the difference between real images x

and generated images \hat{x} as below:

$$\text{FID}(x, \hat{x}) = \|\mu_x - \mu_{\hat{x}}\|_2^2 + \text{Tr}\left(\sum_x + \sum_{\hat{x}} - 2\left(\sum_x \sum_{\hat{x}}\right)^{\frac{1}{2}}\right). \quad (3.8)$$

Hence, a lower score indicates a better generation performance.

Both scores measure only the quality and diversity of images generated, but not the accuracy of the image in reflecting the description of a sentence. Therefore, a third metric called R-precision is used in the previous work [103].

R-precision is defined as the top r relevant text descriptions out of R retrieved texts for an image; candidate sentences are 1 relevant and 99 randomly selected sentences. Observations made through experiments were that when the sample size (i.e., number of candidate sentence) is small, the R value has very high variance. Therefore, two sample sizes were used at 100 and 30,000, which we named R-precision(100) and R-precision(30K) respectively.

Despite the limitations of both inception and FID scores as identified in [6] and [64], there is currently no completely unbiased way to measure the image quality. Thus the proposed work reports both scores and shows examples of the generation results versus those in the extant literature. In terms of R-precision, it requires a pre-trained image-to-text retrieval model that is not available in some previous methods, and is not reported for StackGAN-v1 [110] and StackGAN-v2 [111].

The experiment results demonstrated below were performed on 30,000 random samples from the validation set for the IS score and the R-precision values. The FID score is reported over the full validation set.

The Text Encoder

The Revised Text Encoder versus the Alternative CNN Based Encoder

To decide which structure the proposed text encoder should use, experiments were

performed on the proposed CNN based text encoders under different normalisation functions. These results are also compared with the proposed LSTM based text encoder. The performances are measured by the R-precision, $\mathcal{L}^{\text{SENT-IMG}}$, $\mathcal{L}^{\text{WORD-REGULAR}}$, $\mathcal{L}^{\text{PHRASE-OBJ}}$ achieved in the pre-train phase using the text encoders with real image and sentence pairs (i.e. the generative process is not involved).

Table 3.2 : Text encoder performance of the revised LSTM based text encoder and CNN based text encoders with three different normalisation functions: weight normalisation, batch normalisation and layer normalisation.

Algorithm	R-precision(100)(%)	$\mathcal{L}^{\text{SENT-IMG}}$	$\mathcal{L}^{\text{WORD-REGULAR}}$	$\mathcal{L}^{\text{PHRASE-OBJ}}$
CNN - WeightNorm	64.52	2.4421	2.4194	2.6214
CNN - BatchNorm	62.59	2.5392	2.7046	2.7741
CNN - LayerNorm	58.86	2.7337	2.8762	2.8357
Proposed LSTM	70.45	2.0486	1.7044	2.1659

As in Table 3.2, the CNN based text encoder under weight normalisation achieves the highest R-precision score, which is 3.08% higher than that using batch normalisation and 9.62% higher than the one using layer normalisation. However, it still receives 8.42% less R-precision than the proposed 2-layer LSTM network. Therefore, the final proposed model uses the LSTM based text encoder, as reported in the following sections.

The Revised Text Encoder In Table 3.3, we demonstrate the performance of multiple text embedding losses applied to the proposed LSTM based text encoder as introduced in Section 3.1.1. The comparison is made in terms of the final R-precision scores on the testing set. When calculating the R-precisions, the relevance between a pair (an image and a sentence) is calculated using the cosine similarity between the full feature of both.

Table 3.3 shows that applying $\mathcal{L}^{\text{SENT-IMG}} + \mathcal{L}^{\text{PHRASE-OBJ}}$ achieves the highest R-precision scores, which are 0.07% higher testing R-precision(100) and 0.12% higher R-precision(30K) than the baseline model. The score is also higher than that produced in the two experiments that applying word-regular-grid attention (i.e., using $\mathcal{L}^{\text{WORD-OBJ}}$) to the learning. This shows that firstly, using solely phrase and object-grid regions to construct the encoder loss is better than applying $\mathcal{L}^{\text{WORD-REGULAR}}$ and $\mathcal{L}^{\text{PHRASE-OBJ}}$ together; second, it indicates that introducing phrases is important in learning the text representation.

Table 3.3 : The R-precision(100) and R-precision(30K) on the testing set for the text encoders. LSTM-BASIC is the basic bi-LSTM used in AttnGAN which is our baseline model. LSTM and LSTM-PHRASE comes from the proposed method.

Experiment	R-precision(100)(%)	R-precision(30K)(%)
$\mathcal{L}^{\text{SENT-IMG}} + \mathcal{L}^{\text{WORD-REGULAR}}$ (baseline)	72.99 ± 4.50	4.732 ± 0.018
$\mathcal{L}^{\text{SENT-IMG}} + \mathcal{L}^{\text{WORD-REGULAR}} + \mathcal{L}^{\text{PHRASE-OBJ}}$	72.39 ± 4.26	4.481 ± 0.005
$\mathcal{L}^{\text{SENT-IMG}} + \mathcal{L}^{\text{WORD-OBJ}}$	71.91 ± 1.83	4.473 ± 0.017
$\mathcal{L}^{\text{SENT-IMG}} + \mathcal{L}^{\text{WORD-REGULAR}} + \mathcal{L}^{\text{WORD-OBJ}}$	70.69 ± 2.99	4.030 ± 0.013
$\mathcal{L}^{\text{SENT-IMG}} + \mathcal{L}^{\text{PHRASE-OBJ}}$	73.06 ± 4.05	4.851 ± 0.015

The GAN Network

Table 3.4 reports the R-precision, inception score (IS) and FID score achieved through the previous algorithms and the proposed methods. Apart from the metrics for the proposed method, the R-precision(30K) score and the FID score for AttnGAN, other scores came from previous literature [111]. As StackGAN did not provide a way to extract image features, so R-precision values were not reported.

We denote the proposed method which embeds the text information with LSTM-PHRASE in addition to utilising the object-grid regions and phrases in constructing

the attention mechanism as the method proposed. To demonstrate the importance of introducing phrase and object-grid attention, Table 3.4 also reports the result for first, using only word-regular-grid attention and second, using word-regular-grid and word-object-grid attention. These two are denoted as WORD-REGULAR and WORD-REGULAR+WORD-OBJ respectively. Both of them use the same text embedding as the *Proposed* method.

The proposed method achieves 4.2% higher R-precision (100) and 4.1% higher R-precision (30K) than the baseline, which shows that the proposed method is able to generate images that match more closely with the content described in the sentence. In addition, the proposed method achieves better performance than WORD-REGULAR and WORD-REGULAR + WORD-OBJ. This shows that it is important to introduce phrases to the learning.

Table 3.4 : R-precision, inception and FID score score between AttnGAN and the proposed method.

Method	R-precision(100)(%)	R-precision (30K)(%)	IS(30K)	FID
StackGAN-v1			8.45 ± 0.03	74.05
StackGAN-v2			8.30 ± 0.10	81.59
AttnGAN	85.47 ± 3.69	6.72 ± 0.15	25.89 ± 0.47	32.12
WORD-REGULAR	85.97 ± 3.01	8.20 ± 0.16	26.18 ± 0.30	40.54
WORD-REGULAR+WORD-OBJ	88.25 ± 3.01	10.37 ± 0.13	24.81 ± 0.21	36.51
Proposed	89.69 ± 4.34	10.80 ± 1.96	26.92 ± 0.52	34.52

Below, two aspects are reported with real samples that the generation result surpasses previous methods. First, as shown in Figure 3.5, the method can generate images that match closely with a given sentence, which is proven by the higher R-precision rate. This means that the proposed method is less likely to “miss” objects. For example, when “stop sign” and “go light” are both mentioned, the proposed method can generate both objects instead of only focusing on the “stop sign”. In

addition, when multiple objects / object-grid regions of the same type coexist in the image, the proposed method can generate the correct number of objects.

Second, the proposed method performs well in displaying identifiable main object, such as “A female” or “Large brown cow”. Through feeding the true-grid region information, the proposed method can focus the attention on a more precise image region instead of the entire image.

Experiments were also performed to decide whether self-attention layers should be inserted between convolution layers, as described in [95]. We compared the thumbnail generation results (i.e., generating 64×64 images) with and without such layers, which gives on average an inception score of 10.1138 and 10.1588 respectively. This suggests that inserting self-attention layers does not improve the proposed approach.

Bounding Box Prediction

In the phase of validation and training, the coordinates of each bounding box and number of bounding boxes are predicted by separate networks, as explained in Section 3.1.4. This section compares the performances of multiple alternatives of both predictions. In Figure 3.8, the two prediction tasks are denoted as “Coordinates Prediction” and “Number Prediction” respectively. Comparisons were made in terms of the validation losses over the training iterations.

The first comparison is whether to use all sentences in the prediction tasks or to only use those sentences with position related words. The second is in terms of the number of layers used for the prediction.

From Figure 3.8 , it is clear to see that the best practice is applying a one-layer neural network to both predictions, and that the outnumbered bounding boxes should be excluded when calculating the losses.

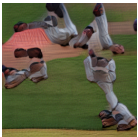











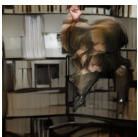

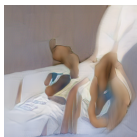
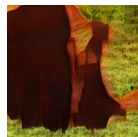








Caption	Baseball players run after a ball during a game	A picture of a stop and go light with a stop sign next to it	A kitchen filled with wooden cabinets and a microwave oven.	A metal counter topped with lots of cheesy pizzas.	A group of young men standing on top of a soccer field.	A pile of oranges sitting inside of a basket.
AttnGAN						
Proposed						
Caption	A woman in white shirt standing in kitchen area.	A busy traffic area on a street during the day	A female wearing a red shirt lies on a bed, resting	Large brown cow standing in field with small cow	A pizza with purple cabbage topping on a table next to white bowl	Black and white photo of a pedestrian at a suburban crosswalk
AttnGAN						
Proposed						

Table 3.5 : Examples of images generated by the proposed method and AttnGAN.

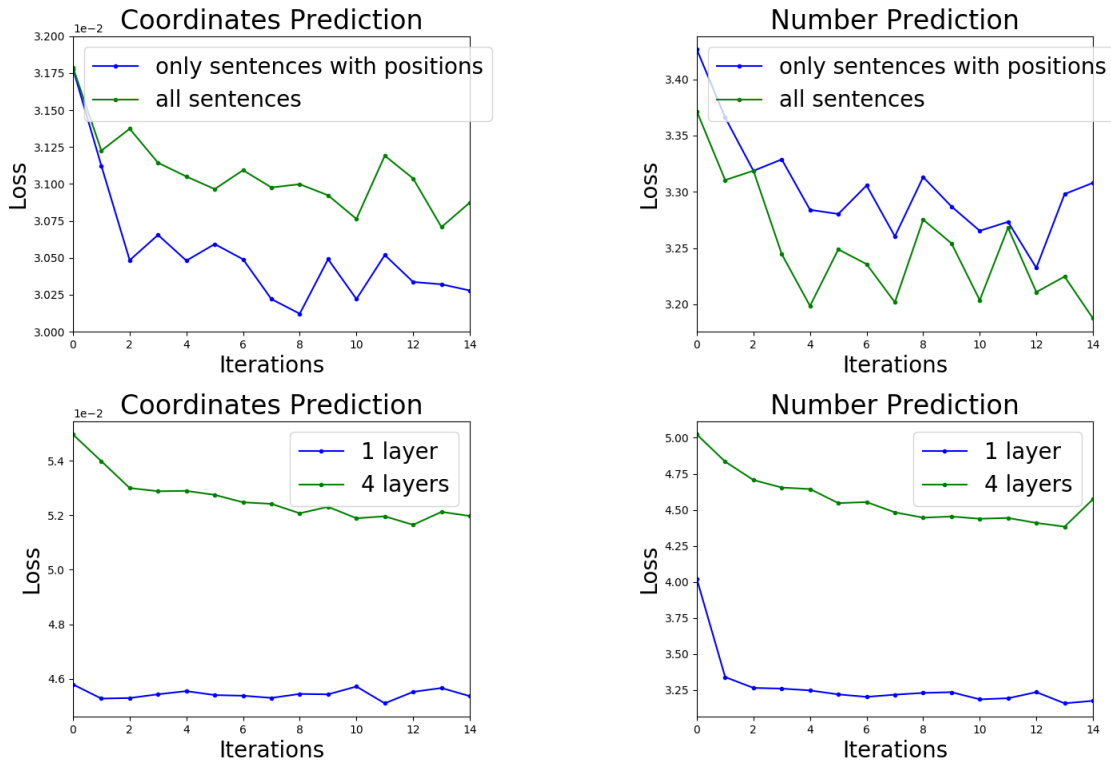


Figure 3.8 : Validation losses of coordinates prediction and number prediction in terms of whether to use the sentences that include position related words for the training.

3.2 Transfer of One Thousand Styles[†]

This section describes the proposed framework that performs style transfer over multiple image domains using a single pair of transfer function. The overall architecture is shown in Figure 3.9. In the proposed framework, we perform transfer over the N styles X_1, X_2, \dots, X_N using only a single pair of universal Generators: G in the forward direction and F in the backward direction. Obviously, the algorithm needs to find an optimal ordering, such that a universal generator pairs will suffice. In this work, no assumption are made for such ordering to be given. Instead, the order of styles is learnt. They are updated in each epoch as in Figure 3.9. In addition,

[†]This work is currently under review at Pattern Recognition Letters.

each style has separate discriminator D_1, D_2, \dots, D_N .

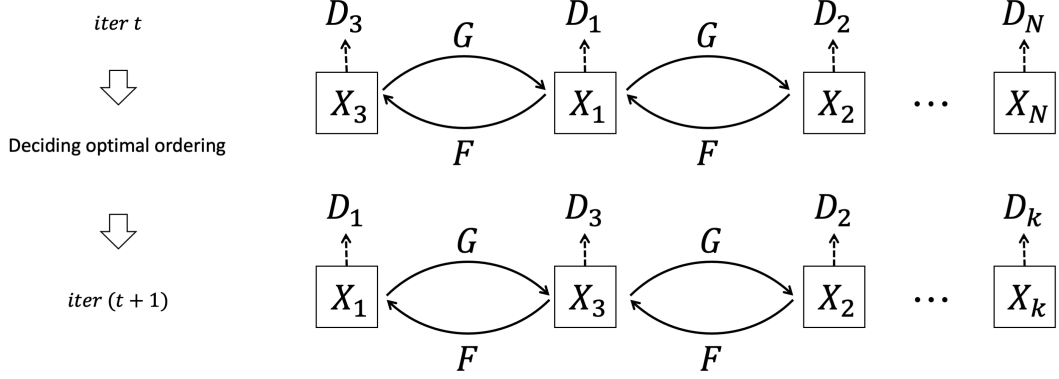


Figure 3.9 : Overall architecture of the proposed framework in two epochs. The graph shows the structure when in t^{th} epoch, the order of styles is $3, 1, 2, \dots, N$, the order is updated to $1, 3, 2, \dots, N$ in the next epoch. The style ordering shown in the graph is for illustration only; the actual ordering is decided during training.

We divide the model parameters into three separate sets: θ^{Odr} , θ^G and θ^D , where θ^G here refers to parameters of the universal generators (i.e. network G and F). They represent chain ordering, generator, and discriminator parameters respectively. The overall training flow of the proposed algorithm is summarised in Algorithm 1, and its details are elaborated upon in the subsequent sections.

3.2.1 Updating Chain Ordering

Neural network parameters are typically learnt through gradient descent. Given the previous trained parameter θ_t , a locally optimal direction $\Delta\theta_{t+1}$ for the next update can be computed through:

$$\theta_{t+1} = \theta_t + \lambda \times \left(\underset{\mathbf{u}, \mathbf{u}^\top \mathbf{u} = 1}{\operatorname{argmin}} \mathbf{u}^\top \nabla_{\theta} f(\theta_t) \right). \quad (3.9)$$

Algorithm 1 Overall training flow

for each epoch till convergence **do**

 update θ^{Odr} : Decide the optimal ordering of styles

- i. Decide potential orders through a chain structure search
- ii. Calculate \mathcal{L}^{Odr} for potential orders
- iii. The current order is decided as the one with the lowest \mathcal{L}^{Odr}

 update θ^{G} and θ^{D} : Perform GAN training under current order

end for

Therefore, in each epoch, conditioning on the fixed parameters associated with GAN, i.e., θ^{G} and θ^{D} , chain ordering parameters $\theta_{t+1}^{\text{Odr}}$ is also updated. However, unlike θ^{G} and θ^{D} , $\theta_{t+1}^{\text{Odr}}$ is not continuous. Therefore, we propose the following approximation alternative. Here, we let $\mathcal{L}^{\text{Odr}}(\cdot)$ to be the objective function after fixing θ^{G} and θ^{D} , i.e., we only update θ^{Odr} part of the parameters.

$$\theta_{t+1}^{\text{Odr}} = \underset{\text{NOD}(\mathbf{u}, \theta_t^{\text{Odr}})=1}{\text{argmin}} \{ \mathcal{L}^{\text{Odr}}(\mathbf{u}) \}, \quad (3.10)$$

where NOD stands for the “node ordering distance” between two chain configurations, meaning the maximum allowable hops required to swap between the two nodes.

When $\text{NOD} = 1$, meaning only adjacent node pairs will be swapped (see Figure 3.10a). When setting $\text{NOD} \geq N - 1$, it means that $\binom{N}{2}$ combinations of node swaps will be allowed in each step of the chain search.

Updating θ^{Odr} causes a sequence of discrete jumps in the overall objective. For this reason, updating a single step in θ^{Odr} may completely undo the progress made from updating θ^{G} and θ^{D} previously. The objective values will fluctuate if we take

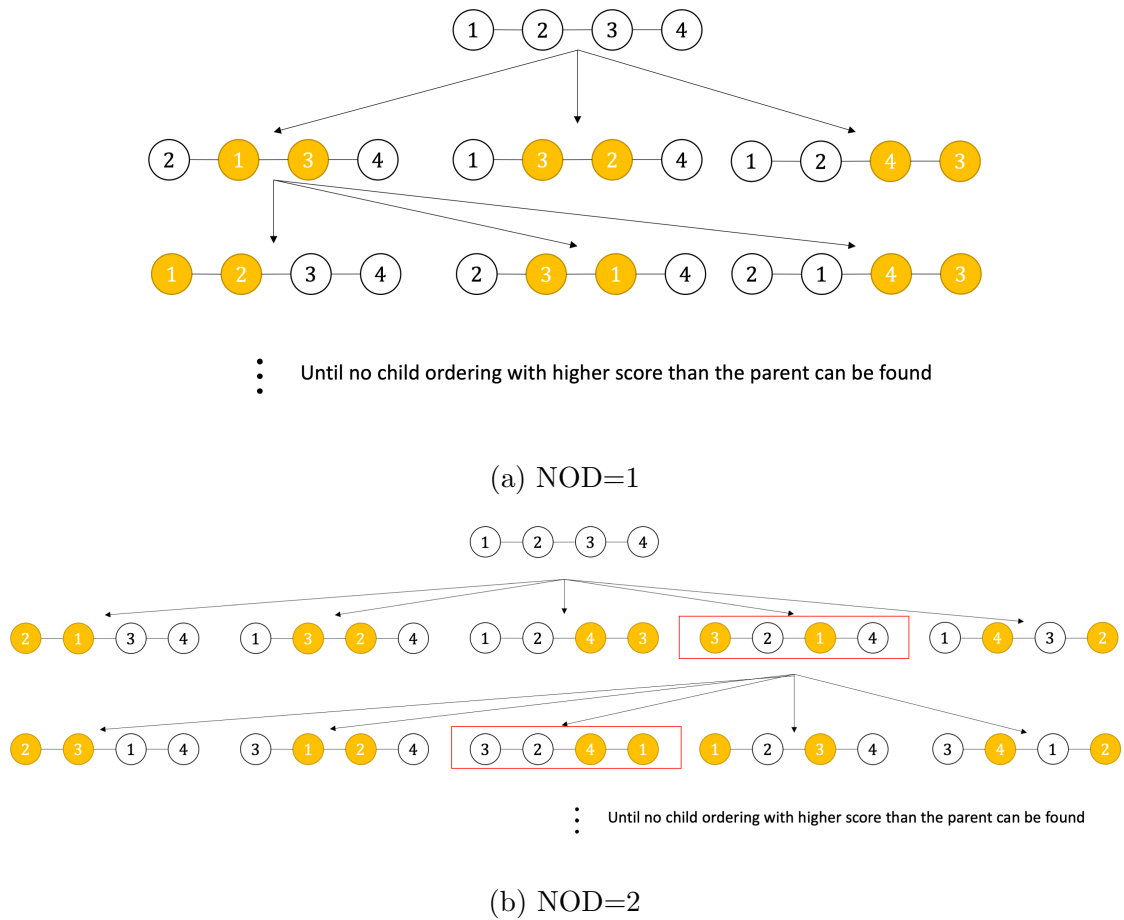


Figure 3.10 : NOD=1 and NOD=2. Numbers are the index of each style, the two nodes being swapped in each step are highlighted in yellow.

the conventional coordinate-descent approach (i.e., take one gradient descent step for each variable in turn).

Hence, the idea is to try not to “revisit” updating θ^{Odr} often. The proposed method achieves this by continuously update θ^{Odr} until it reaches (sub)-optimality before switching back to update θ^{G} and θ^{D} . This is illustrated in Figure 3.10 and Algorithm 2.

An alternative way to update θ^{Odr} is to compare $\mathcal{L}^{\text{Odr}}(\cdot)$ over all $N!$ possible orderings. This might be time-consuming when the total number of styles N is

Algorithm 2 Determine the order $\theta_{t+1}^{\text{Odr}}$ for $t + 1$

Require: order at time t : θ_t^{Odr} ; maximum allowable hops: d

```

current =  $\theta_t^{\text{Odr}}$ 

next = argmin  $\{\mathcal{L}^{\text{Odr}}(\mathbf{u})\}$ 
        NOD( $\mathbf{u}, \theta_t^{\text{Odr}}$ )= $d$ 
while  $\mathcal{L}^{\text{Odr}}(\text{current}) > \mathcal{L}^{\text{Odr}}(\text{next})$  do

    current = next

    next = argmin  $\{\mathcal{L}^{\text{Odr}}(\mathbf{u})\}$ 
            NOD( $\mathbf{u}, \theta_t^{\text{Odr}}$ )= $d$ 
end while

 $\theta_{t+1}^{\text{Odr}} = \text{current}$ 

```

large. However, in practice, when the number of styles is small, this approach is found to be quite manageable and consistent with a good performance.

Ways to Evaluate a Chain

The loss value of each chain order is calculated as the L_1 distance between real samples and the reconstructed samples, i.e. $\mathcal{L}^{\text{Odr}}(\cdot) = \mathcal{L}_{cyc}$. Take an example when the order is $(3, 1, 2, 4)$ over 4 styles. The score for each path is calculated as the average of cycle consistency loss between three pairs of image domains: $3 \leftrightarrow 1$, $1 \leftrightarrow 2$ and $2 \leftrightarrow 4$.

3.2.2 Alternative Mechanic using Scheduled Sampling

Suppose there are in total, N styles, in each epoch training is performed between $2(N - 1)$ pairs of styles. The full training objective is the combination of adversarial loss and the cycle consistency loss of $2(N - 1)$ pairs of image domains. The full training objective is defined as:

$$\mathcal{L}(G, F, \{D_{X_1}, D_{X_2}, \dots, D_{X_N}\}) = \sum_{i=1}^{N-1} \mathcal{L}_{\text{GAN}}(G, D_{X_{\text{Ord}_i}}, X_{\text{Ord}_i}, X_{\text{Ord}_{i+1}}) + \lambda \mathcal{L}(G, F), \quad (3.11)$$

where Ord_i indicates the index of an image domain in the current order.

Inspired by scheduled sampling [8] from natural language generation (NLG) literature: where during training, a word at position t can either be generated based on (1) the real target word at position $t - 1$, or (2) based on a previously generated word at $t - 1$. Scheduled sampling uses some random probability to determine the step.

A similar mechanic is proposed in our work. Concretely, probability α decides an input of G or F to be either from the synthetic generation or to use a real image from the style associated with the previous hop. However, experimental results show that using only real images from the previous hop provides a better performance as it may prevent some drift error.

3.2.3 Experiments

Experiment Setup

Experiments are performed on a machine with one 8GB GeForce GTX 1080 GPU and eight Intel(R) Core(TM) i7-5960X CPUs. Four datasets are used in the experiments: a toy dataset that consists of 2D points, rotated MNIST, cars observed from different angles and images with different brightness levels. Each dataset is trained with different network structures. Each experiment is trained over 500 epochs with a learning rate of 0.0002, and each result is summarised over 10 repeated experiments. The structure of each network, the statistics of each dataset, and experimental results are reported in the following sections.

Toy Dataset

The first dataset we use is a simple toy-dataset, which contains 4,000 training samples drawn from a homogeneous mixture of four Gaussians, centred as $(0.5, 0.5)$, $(1.5, 1.5)$, $(2.5, 2.5)$, $(3.5, 3.5)$. For the toy dataset, generators and discriminators are one-layer neural networks.

Figure 3.11 shows the generation result from the forward and backward direction (i.e., Given image from the first style x and the last order y). The synthetic samples are generated as $G(x)$, $G^2(x)$, $G^3(x)$ in the first direction and $F(y)$, $F^2(y)$, $F^3(y)$. The figure shows that the synthetic samples match the real data closely even when the transfer is performed with up to three steps.

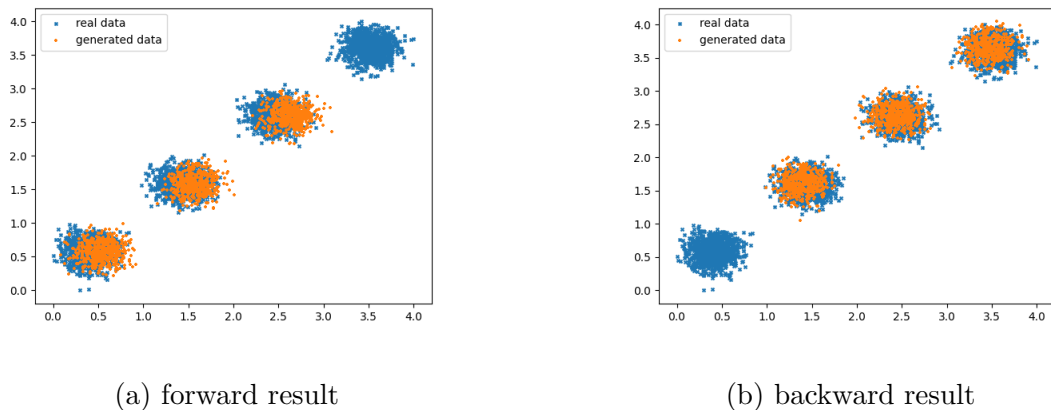


Figure 3.11 : Forward and backward generation results for the toy dataset. Real samples are in blue and the generated samples are in orange.

Table 3.6 reports the time cost spent to decide the θ^{Odr} in one epoch, the number of epochs required before the order is stable, number of orders compared in each epoch and the final MSE between real samples and synthetic samples. We define the order as “stable” when it does not change over a consecutive 50 epochs and is the final optimal order when the training is completed. The number of orders found is counted because all unique orders are compared in each epoch.

Table 3.6 : Performance comparison of NOD = 1, 2, and 3 when the total number of styles $N = 4$ on the toy dataset.

	duration of deciding ordering (seconds)	# epochs before stabilization	# orders found	MSE
NOD = 1	0.0529 ± 0.0069	28.6 ± 7.3	4.0459 ± 0.3768	0.0165 ± 0.0195
NOD = 2	0.0569 ± 0.0067	16.0 ± 6.8	5.7485 ± 0.7097	0.0081 ± 0.0067
NOD = 3	0.0861 ± 0.0137	9.8 ± 6.0	7.0347 ± 0.5623	0.0068 ± 0.0032
NOD = 1 (scheduled sampling, $\alpha = 0.3$)	0.0678 ± 0.0246	34.5 ± 7.5	4.0150 ± 0.2463	0.0322 ± 0.0236

In Table 3.6, NOD=3 achieves 58.79% lower final MSE error with a 0.0332 seconds longer time, to decide the optimal ordering in each epoch. This shows that with a larger NOD value, more orders are found in each epoch. Although the ordering comparison takes slightly longer, the larger NOD value achieves an earlier stabilisation of the optimal order and better model performance. In addition, Table 3.6 shows that training with the scheduled sampling does not perform as well as using only real images as the input of the universal generators.

Rotated MNIST

The second dataset we use contains 5,000 images of digit 7 from the MNIST dataset and their rotated versions (each image is rotated 15, 30, 45 degrees). Generators and discriminators are two-layer neural networks in which the intermediate feature has a length of 49, as shown in Table 3.7.

Table 3.7 : Network structure for Rotated MNIST

Network	Layer	Input Tensors	Output Tensors
Generator	Dense + leaky ReLU	$28 \times 28 \times 1$	7×7
	Dense + Reshape + tanh	7×7	$28 \times 28 \times 1$
Discriminator	Dense + leaky ReLU	$28 \times 28 \times 1$	7×7
	Dense	7×7	1

As Figure 3.12 shows, the synthetic samples are clearly identifiable as digit 7, even after three steps after the real image. Synthetic samples show a clear pattern of being rotated from the original images in the order of increasing angles.



(a) forward result



(b) backward result

Figure 3.12 : Forward and backward generation results on MNIST. The first row of both graphs is the real images; the following three rows are synthetic samples by taking 1, 2, and 3 steps from the real samples.

Table 3.8 reports the time cost for the ordering decision and the performance with different NOD values. Similar to the result from the toy dataset, $NOD = 3$ achieves the lowest MSE loss while spending 0.8247 more seconds to decide the optimal orders and compare 3.011 more orderings than $NOD=1$. Further, training with the scheduled sampling does not perform as well as using only real images despite achieving on average, slightly fewer epochs before the ordering is stable.

Table 3.8 : Performance comparison of $NOD = 1, 2,$ and 3 when the total number of styles $N = 4$ on the Rotated MNIST dataset.

	duration of deciding ordering (seconds)	# epochs before stabilization	# orders found	MSE
NOD = 1	1.5080 ± 0.3701	3.5 ± 1.8	4.0097 ± 0.2107	0.0381 ± 0.0015
NOD = 2	1.8408 ± 0.2665	1.3 ± 0.4	6.0127 ± 0.2668	0.0380 ± 0.0008
NOD = 3	2.3327 ± 0.3133	1.2 ± 0.4	7.0207 ± 0.3962	0.0373 ± 0.0017
NOD = 1 (scheduled sampling, $\alpha = 0.3$)	1.7663 ± 0.1205	2.8 ± 0.3	4.0090 ± 0.1699	0.0407 ± 0.0001

Different from the toy dataset, the average number of epochs before the order

becomes stable is much smaller. In most of our experiments, the order is stable from the beginning or after 1 epoch. We believe this is related to the nature of the dataset and the size of the network. Images can be clearly identified as coming from different domains after the first epoch and the optimal order that reflects the natural order of the dataset would achieve a smaller loss value than the alternatives. Training on the toy dataset for one epoch does not allow synthetic samples from different domains to be well separated. This causes the ordering decision to fluctuate.

Cars from Multiple Angles

Another experiment was performed on the cars observed from increasing angles from [26]. We use images in the dataset that have class 2, 3, 4, and 5. The dataset contains 676 images. The network structure follows the same structure used for CycleGAN [42], except the first two and last two layers in the generator are excluded. Therefore, it generates 64×64 images instead of 256×256 images in the original network.



Figure 3.13 : Forward and backward generation results on cars from multiple angles.

The synthetic samples show plausible shapes of cars and match the original image. In addition, the rotation with increasing angles is shown clearly in the result.

Table 3.9 reports the time cost of updating chain ordering and compares the

performance under different NOD values. It shows that on the car dataset, NOD=3 achieves 90.03% lower final MSE error with, on average, 0.0348 seconds longer time to decide the optimal ordering in each epoch. Similar to the rotated MNIST dataset, the optimal order stabilises from the beginning or after the first epoch.

Table 3.9 : Performance comparison of NOD = 1, 2, and 3 when the total number of styles $N = 4$ on the car dataset.

	duration of deciding ordering (seconds)	# epochs before stabilization	# orders found	MSE
NOD = 1	3.8782 ± 0.2639	1.2 ± 0.6	4.0100 ± 0.1608	0.0045 ± 0.0023
NOD = 2	4.0051 ± 0.2554	1.0 ± 0.0	6.0219 ± 0.3579	0.0038 ± 0.0035
NOD = 3	4.1130 ± 0.3267	1.0 ± 0.0	7.0304 ± 0.5124	0.0037 ± 0.0005

Images with different lighting conditions

The last dataset used comprises images with four levels of brightness, which contains in total 25,148 images: 6,287 for each category. The original photographs were collected from the CycleGAN paper and they are processed with TensorFlow to images with three different brightness levels. Images with four brightness levels were combined to be our dataset. The network structure used is the same one from CycleGAN [114], which generates 256×256 images from 256×256 images.

The synthetic samples demonstrate a clear pattern of increasing and decreasing brightness. However, images in the backward direction are not generated as well as those in the forward direction, especially images on the last row, which should theoretically be exactly the same as images with the original brightness. We attribute this to the notion that details of an image are already lost when adjusting the brightness to the highest level in the pre-processing step. Thus, it is difficult to recover the image by feeding the brightest image and performing generation in a backward fashion.



Figure 3.14 : Forward and backward generation results on images with multiple brightness.

Table 3.10 reports the performance over different NOD values. The higher NOD value still leads to a better generation output and larger time cost to decide the optimal ordering.

Table 3.10 : Performance comparison of NOD = 1, 2, and 3 when the total number of styles $N = 4$ on the multiple brightness dataset.

	Duration of deciding ordering (seconds)	# epochs before stabilisation	# orders found	MSE
NOD = 1	12.1168 ± 1.1031	1.7 ± 1.6	4.1111 ± 0.4581	0.0090 ± 0.0012
NOD = 2	19.6784 ± 3.4249	1.3 ± 0.4	7.2778 ± 2.5778	0.0068 ± 0.0008
NOD = 3	20.9056 ± 3.0545	1.0 ± 0.7	8.8254 ± 3.7651	0.0032 ± 0.0012

3.3 Summary

This chapter describes two studies undertaken in terms of applications of cross domain transfer. Our first work improves state-of-the-art attention based GAN network for text-to-image generation. Using *phrase* as an additional important encoding unit into image generation, our proposed work incorporates two innovations: First, we proposed a new design of text embedding which extracts additional phrase embedding. Second, we incorporate a new set of attentions computed between object-

grid regions and phrases and bring them into our GAN network design. Through experimentation on the MSCOCO dataset, the proposed approach is capable of generating more realistic and accurate images.

The second work addresses transfer across multiple (more than two) domains. Transfer between image domains is usually performed between a pair of image styles and transformation between N styles requires $2\binom{N}{2}$ generators. Observations suggest that a natural sequence of styles exists in some datasets, such as sets of objects observed from increasing angles or photographs taken under increasing lighting conditions. Therefore, we propose a methodology that automatically determines the optimal order of image domains, while studying the transfer functions between consecutive image domains. In this method, although translation between two domains might require up to $N - 1$ steps, only two generators are required to perform translation over N image domains. Experimental results show that the proposed methodology is capable of deciding the ordering over image domains while generating plausible images.

Chapter 4

Latent Space Modelling in Style Transfer

As introduced in Chapter 1, the current generative models usually model their latent spaces with standard Gaussian or uniform distributions. Use of such single mode distributions does not work well with datasets that contain diverse samples. Therefore, our work focuses on studying the effects and properties brought by applying mixture models to cross domain generation, as described in this chapter. We start with GMM. In particular, the proposed framework devised a novel posterior consistency module that calculates the responsibility probability of a sample. This allows exploration of properties on complex datasets that cannot be directly modelled with a GMM distribution. The performance is first demonstrated on unconditional image generation and later extended to text-to-image generation. To further demonstrate the effectiveness of replacing standard Gaussian with GMM, our work also investigates the network compression effect achieved for both image generation and anomaly detection.

In addition, as the number of components in the Gaussian mixture needs to be predetermined for the GMM based GANs, we also propose that DP-GAN uses DPGMM to model the latent space. DP-GAN enables the number of distinct classes studies simultaneously with the distribution parameters during training. As it is not trivial to study the parameters using a naive application of the DPGMM, we propose a set of mechanics to mitigate the problem. The details are explained in Section 4.1 to 4.4.

4.1 GAN-based GMM Responsibility Modelling*

The proposed architecture consists of three networks: First, we describe PCM, as well as the loss function that matches it with $p(k|z, \theta)$. Second, we have a generator G that produces synthetic samples from GMM random vectors. Third, a discriminator D which encodes samples to feature vectors and discriminates between real and synthetic samples. The overall architecture design is shown in Figure 4.1. In the following sections, we explain the details of the three networks.

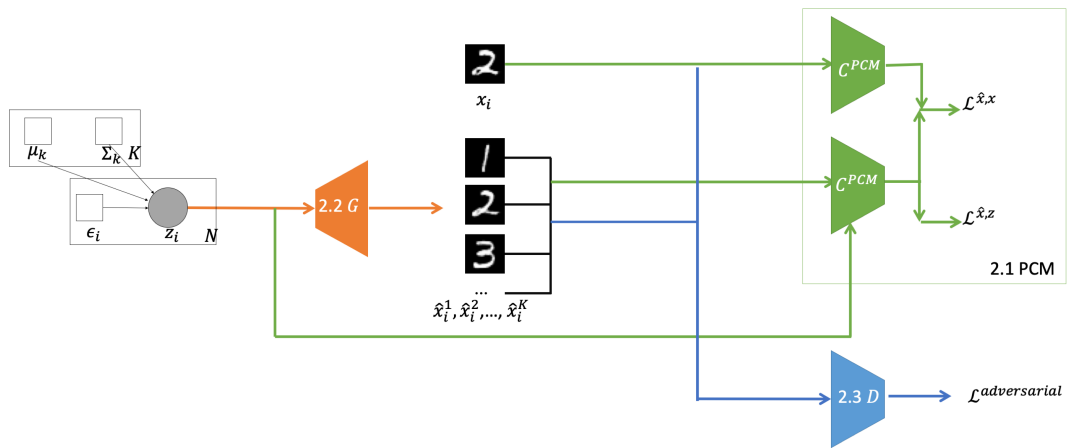


Figure 4.1 : The overall architecture. The feed-forward logic of the classifier C , the generator G and the discriminator D are marked with different colours.

4.1.1 Posterior Consistency Module (PCM)

The PCM comprises a few components. In its core, it comprises a classifier/generator C_{PCM} that outputs $p(k|x, \theta)$ given x . When a synthetic data \hat{x} is used as its input, the output approximates $p(k|\hat{x}, \theta)$ and it needs to be matched against:

- i. $p(k|z, \theta)$, this is to ensure that the responsibility probability condition on its latent variable z (from using Equation 4.5) is similar to the responsibility

*This work was accepted by ICPR 2020.

Table 4.1 : The overall network structure.

Stage	Sub-stage	Name	Input Tensors	Output Tensors
idrule C	Encoding Network (if shared)	Conv (kernel=5, stride=2) + LeakyReLU	$D_{img} \times D_{img} \times D_h$	$D_{img}/2 \times D_{img}/2 \times 64$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{img}/2 \times D_{img}/2 \times 64$	$D_{img}/4 \times D_{img}/4 \times 128$
	Encoding Network (if not shared)	Conv (kernel=5, stride=1) + ReLU	$D_{img} \times D_{img} \times D_h$	$D_{img} \times D_{img} \times 32$
		MaxPool (pool.size=2, stride=2)	$D_{img} \times D_{img} \times D_h$	$D_{img}/2 \times D_{img}/2 \times 32$
		Conv (kernel=5, stride=2) + ReLU + Flatten	$D_{img}/2 \times D_{img}/2 \times 32$	$D_{img}/4 \times D_{img}/4 \times 16$
		Linear + ReLU	$D_{img}/4 \times D_{img}/4 \times 16$	1024
	Classification Network (if shared)	Linear	$D_{img}/4 \times D_{img}/4 \times 128$	K
	Classification network (if not shared)	Linear	1024	K
	G	Linear + Batch norm + LeakyReLU + Reshape	$1 \times D_z$	$D_{img}/4 \times D_{img}/4 \times 256$
		Transposed Conv (kernel=5, stride=1) + Batch norm + LeakyReLU	$D_{img}/4 \times D_{img}/4 \times 256$	$D_{img}/4 \times D_{img}/4 \times 128$
Transposed Conv (kernel=5, stride=2) + Batch norm + LeakyReLU		$D_{img}/4 \times D_{img}/4 \times 128$	$D_{img}/2 \times D_{img}/2 \times 64$	
Transposed Conv (kernel=5, stride=2) + Tanh		$D_{img}/2 \times D_{img}/2 \times 64$	$D_{img} \times D_{img} \times D_h$	
D	Encoding Network	Conv (kernel=5, stride=2) + LeakyReLU	$D_{img} \times D_{img} \times D_h$	$D_{img}/2 \times D_{img}/2 \times 64$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{img}/2 \times D_{img}/2 \times 64$	$D_{img}/4 \times D_{img}/4 \times 128$
	Discriminator Network	Linear	$D_{img}/4 \times D_{img}/4 \times 128$	1

distribution dependent on x (from using the neural network). For obvious reason, the corresponding loss is named as:

$$\begin{aligned}
\mathcal{L}^{\hat{x}, z} &= \mathbb{E}_{\{z_1 \sim \mathcal{N}(\mu_1, \Sigma_1), \dots, z_K \sim \mathcal{N}(\mu_K, \Sigma_K)\}} \left[\frac{1}{K} \sum_{k=1}^K I(p(k|z_k, \theta), p(k|G(z_k), \theta)) \right] \quad (4.1) \\
&= \mathbb{E}_{\{z_1 \sim \mathcal{N}(\mu_1, \Sigma_1), \dots, z_K \sim \mathcal{N}(\mu_K, \Sigma_K)\}} \left[\frac{1}{K} \sum_{k=1}^K I(p(k|z_k, \theta), C_{\text{PCM}}^\theta(G(z_k))) \right].
\end{aligned}$$

where K is the total number of modes in the Gaussian mixture.

- ii. $p(k|x, \theta)$, this is to ensure that the distribution generated is similar to those generated by the real data x . The corresponding loss is named as:

$$\begin{aligned}
\mathcal{L}^{\hat{x}, x} &= \mathbb{E}_{x_i \sim p_{data}, \{z_1 \sim \mathcal{N}(\mu_1, \Sigma_1), \dots, z_K \sim \mathcal{N}(\mu_K, \Sigma_K)\}} \sum_{k=1}^K I(p(k|x_i, \theta), p(k|z_k, \theta)) \quad (4.2) \\
&= \mathbb{E}_{x_i \sim p_{data}, \{z_1 \sim \mathcal{N}(\mu_1, \Sigma_1), \dots, z_K \sim \mathcal{N}(\mu_K, \Sigma_K)\}} \left[\sum_{k=1}^K I(C_{\text{PCM}}^\theta(x_i), C_{\text{PCM}}^\theta(\hat{x})) \right],
\end{aligned}$$

where

$$I(X; Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p_{(X,Y)}(x, y) \log \left(\frac{p_{(X,Y)}(x, y)}{p_X(x) p_Y(y)} \right). \quad (4.3)$$

Essentially, all we need is a measure function to compute the distance between two distributions, so KL or DS divergence can also be used. However, in our experiment, mutual information loss provides the best result.

In terms of the neural network design, the classifier has two plausible alternatives: to share the feature encoding layers with the discriminator and to build a standalone network that contains multiple CNN layers to classify images. The shared feature encoding network encodes each image to a feature vector, and the classifier will be a simple standard linear softmax classifier built on top of the features.

Details of both network designs can be found below in Table 4.4. In Table 4.4, we use acronyms for operations in the table: “Conv” is the convolution operation, of which the kernel and stride size are in the bracket; “Batch norm” is short for batch normalisation; “Flatten” refers to the operation that flattens a tensor to 1D array. D_{img} , D_h and D_z are related to the datasets we use, the exact value of each are reported in the experiments.

In Section 4.1.5, we report the results of both classifier designs in terms of generation performance and computation costs. Parameters of the classifier are optimised by both the adversarial loss and the mutual information loss. Details of how the update is performed are introduced in Section 4.1.4.

The C_{PCM} is not required for generating new images after the model is fully trained. During testing, a random vector is sampled directly from the GMM model for the generation. The C_{PCM} can then be used to assign an unseen image to Gaussian and subsequently perform segmentation on the testing set.

4.1.2 The Generator

The ideal output of the trained generator is that all random vectors sampled from the same Gaussian of the GMM should generate similar images when fed through the generator. However, because the training is label independent, one cannot control the correspondence between which Gaussian is selected and the category of the generated sample during training. Therefore, instead of sampling a random vector from one Gaussian during training, our work samples one vector from all Gaussians during training and uses the classification output to weigh the loss values. The details are explained below.

One random vector is sampled from each Gaussian as $z_i = [z_i^1, \dots, z_i^K]$. These vectors are used to generate K synthetic images $[\hat{x}_i^1, \dots, \hat{x}_i^K]$. The classification output is used to weigh the adversarial loss calculated from each pair of a generated sample \hat{x}_i^k and the real image x_i .

In addition, because the design expects the training to be performed in an “end-to-end” fashion, the reparameterisation trick is applied to the z_i sampling process so that the back-propagation can be used to update the parameters μ_k and Σ_k of each Gaussian. Instead of sampling $z_i^k \sim \mathcal{N}(\mu_k, \Sigma_k)$, we define $z_i^k = \Sigma_k \epsilon + \mu_k \quad \forall k \in [1, K]$, where ϵ is sampled as $\epsilon \sim \mathcal{N}(0, I)$.

The generator structure used in our experiments is in Table 4.4. In the table, “LeakyReLU” (short for “leaky rectified linear unit”) and “Tanh” are activation functions.

4.1.3 The Discriminator

The design of the discriminator has two options: whether or not the image encoding layers are shared with the classifier. The image encoding network is followed by a standard linear logistic regression to identify the given image as real or fake.

The adversarial loss, which is calculated over K pairs of real and synthetic samples as:

$$\mathcal{L}^{\text{adversarial}} = \mathbb{E}_{x_i \sim p_{\text{data}}} \left(\frac{1}{K} \sum_{k=1}^K p(k|x_i, \theta) \times (\log D(x_i) + \log(1 - D(\hat{x}^k))) \right). \quad (4.4)$$

4.1.4 Training Parameters for the Prior Distribution

In our setting, the prior distribution is a GMM, the two trainable variables are means for K Gaussians $\mu \in \mathbb{R}^{K \times D_z}$ and standard deviations for K Gaussians $\sigma \in \mathbb{R}^{K \times D_z \times D_z}$. Both variables are updated by the adversarial loss in addition to the mutual information loss because the training is performed “end-to-end”. Algorithm 3 gives the pseudocode about how the updates are performed on each network in one iteration.

4.1.5 Experiments

This section evaluates the performance of the proposed method by comparing it with several baselines.

Experiment setup

The datasets used are the MNIST [57], Fashion-MNIST [101] and Oxford-102 Flower [52] datasets. Details are listed below in Table 4.2. In particular, only a subset of Oxford-102 is selected to perform the training, which is the images that belong to the first 10 classes. For experiments performed on each dataset, we used different hyper-parameters (see Table 4.2).

Linear Interpolation across Gaussian

Figure 4.2 shows samples generated by the proposed model trained on several datasets in a completely unsupervised manner. We set the number of Gaussians

Algorithm 3 Training the proposed model for 1 iteration

Require: $X = [x_1, x_2, \dots, x_M]$ - M training images in one batch

- 1: **for** $i = 1 \dots M$ **do**
 - 2: Classify x_i into K Gaussians
 - 3: **for** $k = 1 \dots K$ **do**
 - 4: $\epsilon \sim \mathcal{N}(0, I)$
 - 5: $z_k = \Sigma_k \epsilon + \mu_k$
 - 6: $\hat{x}_i^k \leftarrow G(z_k)$
 - 7: Classify \hat{x}_i^k into K Gaussians
 - 8: **end for**
 - 9: Calculate $\mathcal{L}^{\text{adversarial}}$ from x_i and $[\hat{x}_i^1 \dots \hat{x}_i^K]$ as in Equation 4.4
 - 10: Calculate $\mathcal{L}^{\hat{x}, z}$ as in Equation 4.1
 - 11: Calculate $\mathcal{L}^{\hat{x}, x}$ as in Equation 4.2
 - 12: Update the Discriminator with $\mathcal{L}^{\text{adversarial}}$
 - 13: Update both the Generator and the C_{PCM} with $\mathcal{L}^{\text{adversarial}}$
 - 14: Update the Classifier with $\mathcal{L}^{\hat{x}, x}$
 - 15: Update the Classifier, parameters for the latent distribution with $\mathcal{L}^{\hat{x}, z}$
 - 16: **end for**
-

Table 4.2 : Statistics of the different datasets used in the empirical evaluation.

Dataset	Number of Classes	Data Dimension	Train Samples	Validation Samples	Test Samples	Number of Epochs	Learning Rate γ	D_{img}	D_h
idrule MNIST	10	$28 \times 28 \times 1$	60,000	-	10,000	200	0.0002	28	1
Fashion-MNIST	10	$28 \times 28 \times 1$	60,000	-	10,000	200	0.0002	28	1
Oxford-102 Flower	102	$64 \times 64 \times 3$	1,020	1,020	6,149	10,000	0.0002	64	3

equal to the total number of classes of the dataset in all experiments. When we are performing the linear interpolation as in the right panels, the random vector z for each image generation is calculated as $z = \sum_k \epsilon + \mu_k \forall k \in \{1, \dots, K\}$, where ϵ is sampled as $\epsilon \sim \mathcal{N}(0, I)$. ϵ is kept the same for all images for each dataset.

Two conclusions can be drawn from the results in Figure 4.2. First, a fully trained proposed method can learn to “allocate” each class of image to a Gaussian. Second, the trained model can be used to perform smooth linear interpolation between Gaussians and even among more than two Gaussians. Figure 4.3 demonstrates the linear interpolation performed over three categories. The proportion of Gaussians of the synthetic images can be set manually.

Image Generation Quality

The generation performance is measured with two commonly used metrics: IS [86] and FID score ([35]). IS score is calculated as $I = \exp(\mathbb{E}_x D_{\text{KL}}(p(y|x)||p(y)))$ and a higher value generally indicates a better performance, where x is a generated image and y is the label predicted by the Inception model [94]. *FID score* is another metrics that measures the image generation quality. A lower value shows a better image quality and greater diversity. It calculates the difference between real images x and generated images g as $\text{FID}(x, g) = \|\mu_x - \mu_g\|_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}})$.

Limitations of both inception and FID scores have been identified in the extant literature [6, 64], and there are currently no “perfect” metrics at this moment. These two metrics are used as an indication rather than a hard measure.

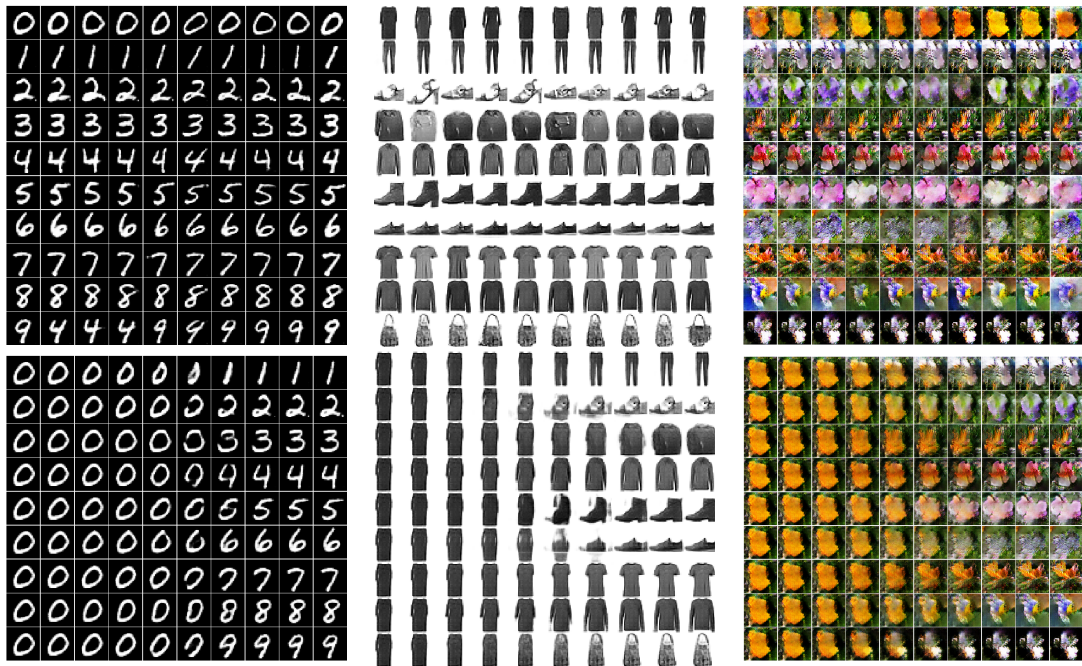


Figure 4.2 : Samples generated by the proposed models trained on the MNIST (left column), Fashion-MNIST (middle column) and CIFAR-10 (right column) datasets. The top row contains images generated using random vectors sampled from a different Gaussian. The bottom row shows the linear interpolation result from one Gaussian to another. The index of Gaussian does not necessarily correspond to the actual digit, generated images are reordered for demonstration purpose.

This evaluation is performed on the Oxford-102 dataset. As the inception score is suggested to be evaluated on a large enough number of samples, 5K synthetic samples are generated to calculate both values. Figure 4.4 plots inception scores and FID scores calculated over the training epochs. As the figure shows, the proposed framework, whether the encoding layers are shared or not, constantly achieves a higher inception scores and lower FID scores from the start. Applying shared encoding layers allows an even superior performance.

Table 4.3 reports the number of trainable parameters and the best inception and FID score achieved for each algorithm. The proposed framework, with separate

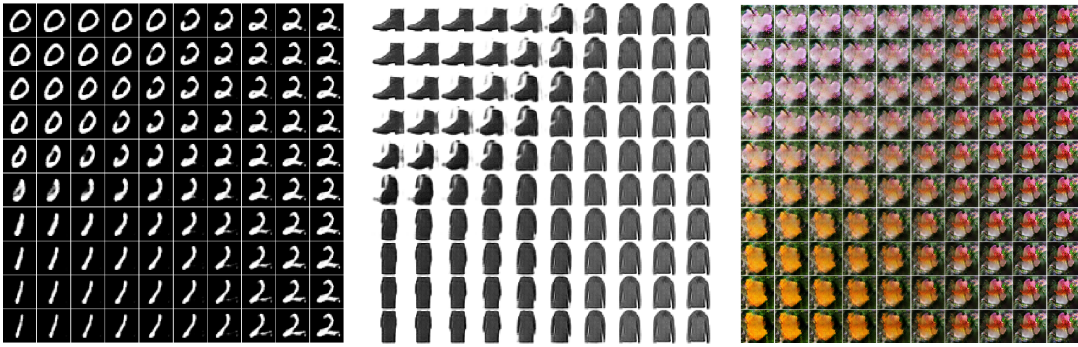


Figure 4.3 : Linear interpolation over three Gaussians on the MNIST, Fashion-MNIST and Oxford-102 datasets.

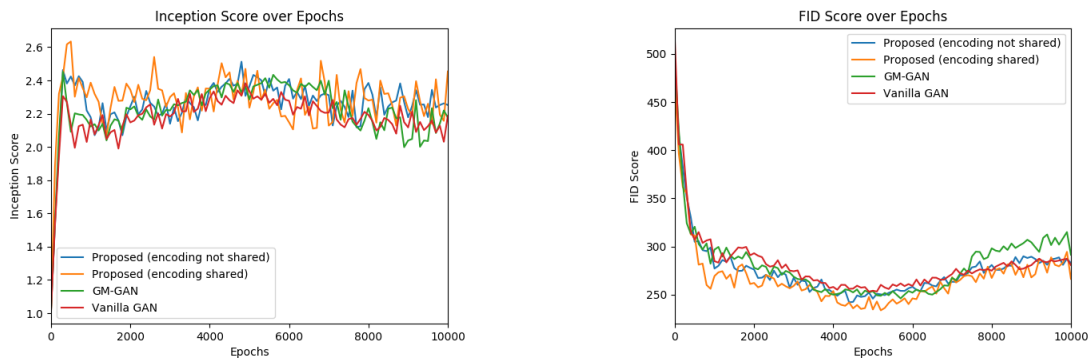


Figure 4.4 : Inception score and FID scores over training epochs on the Oxford dataset.

feature encoding layers, results in a 10.81% higher IS and 3.48% lower FID score compared with GM-GAN. The shared encoding layers provide a 5.74% higher IS and 10.85% lower FID score compared with the one without the shared layers.

From Figure 4.4 and Table 4.3, it is clear that the proposed method can outperform previous baseline models in terms of the image generation quality. The shared feature encoding layers would further improve the performance and reduce the size of the network.

Table 4.3 : Number of parameters, Inception scores and FID scores of the proposed method and the baselines.

	number of parameters	Inception Score \uparrow	FID score \downarrow
Proposed (encoding not shared)	13,005,411	2.9664 ± 0.2188	231.0577 ± 7.5371
Proposed (encoding shared)	8,794,835	3.1368 ± 0.1596	205.9776 ± 7.8587
GM-GAN	8,467,145	2.6770 ± 0.1079	239.3936 ± 6.7672
Vanilla GAN	8,366,145	2.4882 ± 0.1065	247.0610 ± 7.2361

Network Compression

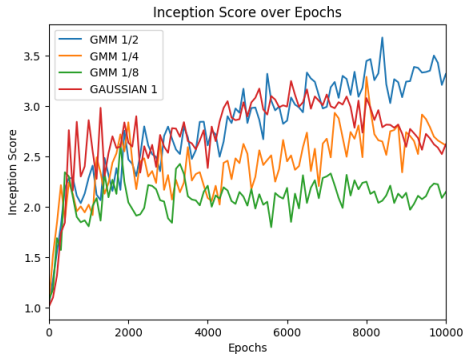
This section demonstrates that applying Gaussian mixture to GANs can significantly reduce the number of parameters while achieving a performance close to the vanilla GAN. Figure 4.5 plots inception and FID scores over training epochs of the proposed GMM-based GAN with 1/2, 1/4, and 1/8 the size of the original. The original size of the network is demonstrated in Table 4.4; 1/2 means that the number of parameters in each layer is reduced by half. The same applies for 1/4 and 1/8. These results are compared with the vanilla GAN, which uses the original number of parameters.

Figure 4.5 shows that the proposed approach, achieves a higher IS and lower FID score on both the CUB dataset and the Oxford-102 dataset compared with the vanilla GAN, while using only 1/2 of the number of parameters.

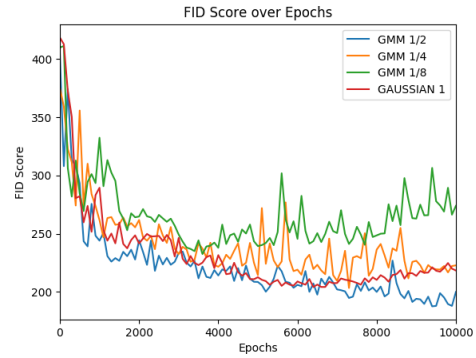
Performance on Highly Imbalanced Dataset

Training GAN networks often suffers from mode collapse. Therefore, we also demonstrate the performance on the highly imbalanced dataset (see Figure 4.6).

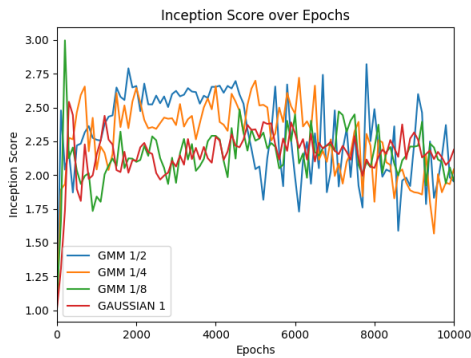
The imbalanced dataset we used is a subset of the original MNIST dataset. From digit 0 to digit 9, we randomly choose digit d as the dominant category. One thousand samples are randomly selected from all other categories and the category



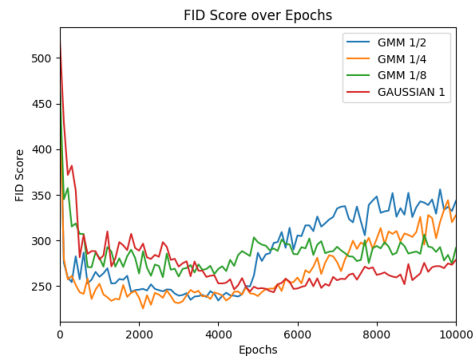
(a) Inception scores on the CUB dataset.



(b) FID scores on the CUB dataset.



(c) Inception scores on the Oxford-102 dataset.



(d) FID scores on the Oxford-102 dataset.

Figure 4.5 : Inception and FID scores over epochs for the proposed GMM-based GAN with various size of the network compared with the vanilla GAN.

d is kept the same. This becomes the imbalanced training dataset. Figure 4.6 shows the synthetic samples generated by the three algorithms when the digit 1 becomes the dominant category in the imbalanced dataset. While using both the standard Gaussian and GMM result in poor generation quality and collapsed output, the proposed framework is able to deliver digits with clear shape and categorisation.



(a) Gaussian as the latent distribution (b) GMM as the latent distribution (c) The proposed framework

Figure 4.6 : Performance on highly imbalanced dataset.

4.2 Compressing GANs with Gaussian Mixture Prior[†]

In this section, we investigate the network compression effect achieved by applying Gaussian mixture to GANs. In particular, this section presents a modified GM-GAN with a classifier to encourage the matching between the component from the latent distribution and the class from the synthetic data. The application of GM-GAN has been limited to image generation thus far, so our work further extends the GM-GAN to the anomaly detection problem.

4.2.1 Method

This section introduces details of the proposed framework, which consists of three modules: First, a classifier C classifies a given image as one of K classes and C shares the feature encoding layers with D . Second, a generator G produces synthetic samples from the GMM random vector z . Third, a discriminator D encodes samples to feature vectors and distinguishes between real and fake samples. The overall architecture of the proposed modified GM-GAN is shown in Figure 4.7.

[†]This work is currently under review at Pattern Recognition Letters.

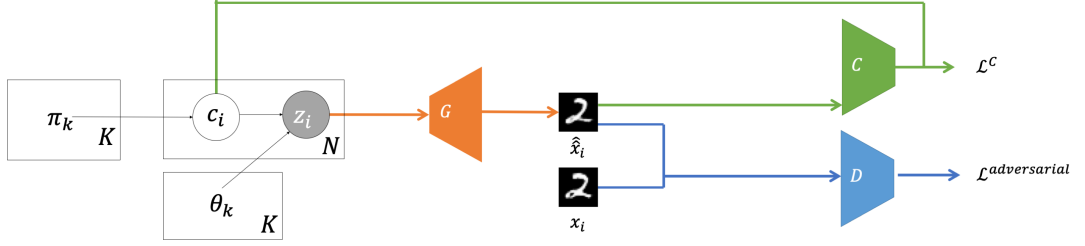


Figure 4.7 : The complete architecture of our model. K is the number of Gaussians in the GMM; z_i is the random vector sampled from the GMM. x_i and \hat{x}_i are the real and fake sample. G , C and D are the generator, discriminator and classifier.

Table 4.4 : The overall network structure of our model, including the input and output shape of tensors in each network.

Stage	Sub-stage	Name	Input Tensors	Output Tensors
C	Encoding Network (shared)	Conv (kernel=5, stride=2) + LeakyReLU	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64/r$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64/r$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128/r$
	Classification Network	Linear	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128/r$	K
G		Linear + Batch norm + LeakyReLU + Reshape	$1 \times D_z$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 256/r$
		Transposed Conv (kernel=5, stride=1) + Batch norm + LeakyReLU	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 256/r$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128/r$
		Transposed Conv (kernel=5, stride=2) + Batch norm + LeakyReLU	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128/r$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64/r$
		Transposed Conv (kernel=5, stride=2) + Tanh	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64/r$	$D_{\text{img}} \times D_{\text{img}} \times D_h$
D	Encoding Network (shared)	Conv (kernel=5, stride=2) + LeakyReLU	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64/r$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64/r$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128/r$
	discriminator Network	Linear	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128/r$	1

Classifier The classifier C is designed to encourage one-to-one correspondence between each component from the latent distribution and each class from the synthetic data. C outputs $p(k|\hat{x}, \theta)$ given a synthetic sample \hat{x} . $p(k|\hat{x}, \theta)$ needs to match against $p(k|z, \theta)$ which is the responsibility probability condition on the corresponding latent variable z . $p(k|z, \theta)$ can be computed analytically given parameters of the latent GMM:

$$w \equiv (w_1, \dots, w_K) = \left(\frac{\mathcal{N}(z|\mu_1, \Sigma_1)}{\sum_{k=1}^K \mathcal{N}(z|\mu_k, \Sigma_k)}, \dots, \frac{\mathcal{N}(z|\mu_K, \Sigma_K)}{\sum_{k=1}^K \mathcal{N}(z|\mu_k, \Sigma_k)} \right). \quad (4.5)$$

The loss function of the classifier is calculated as:

$$\mathcal{L}^C = \mathbb{E}_{z \sim p_z(z)} [I(p(k|z, \theta), p(k|\hat{x}, \theta))]. \quad (4.6)$$

Overall, the function I needs to be a measure function to compute the distance between two distributions, so KL or JS divergence can be used. However, mutual information loss provided the best result in our experiments.

C is chosen to share the feature encoding layer with the discriminator, during training, \mathcal{L}^C is applied to optimise both the generator G and the simple standard linear softmax classifier. The feature encoding layers are updated by the $\mathcal{L}^{\text{adversarial}}$ only.

Generator The generator G simply takes a random vector z sampled from the DPGMM to generate synthetic samples. As the training is performed in an “end-to-end” fashion (i.e. parameters of the GMM are updated together with the GAN network), so the reparameterisation trick is applied to the random vector sampling process. Instead of sampling $z_i^k \sim \mathcal{N}(\mu_k, \Sigma_k)$, we define $z_i^k = \Sigma_k \epsilon + \mu_k \quad \forall k \in [1, K]$, where ϵ is sampled as $\epsilon \sim \mathcal{N}(0, I)$. the structure of DCGAN is chosen to build the model, which is reported in Table 4.4.

Discriminator The discriminator D consists of multiple layers of CNN which is referred to as the feature encoding layers, followed by a standard linear logistic regression to identify the given image as real or fake. The design makes the discriminator and the classifier share the same feature encoding layers.

Table 4.4 reports the size and number of parameters used in each layer. Acronyms are used for certain operations in the table, “LeakyReLU” which stands for “leaky rectified linear unit” and “Tanh” are activation functions; “Conv” is the convolution operation, of which the kernel and stride size are in the bracket; “Batch norm” is short for batch normalisation; “Flatten” refers to the operation that flattens a tensor to 1D array. D_{img} , D_h and D_z are related to the datasets we use, the exact value of each are reported in Table 4.5.

The parameter r in Table 4.4, is the compression rate, which is defined as the ratio of the depth of each layer in a network out of the depth of each layer in the network used by the baseline model. Apart from the parameters for the latent distribution, $r = 2$ indicates a network to be half the size from the original network; $r = 3$ indicates 1/3 of the original size. The complete training strategy is summarised in Algorithm 4.

4.2.2 Experiments

This section empirically evaluates the performance of GM-GAN and the proposed modified algorithm for image generation and anomaly detection.

Performance on image generation

Datasets Experiments in terms of the image generation quality are performed on two datasets: the CIFAR-10 dataset [52] and the Oxford-102 dataset [76]. Details of each dataset and values that determine the size of the network in Table 4.4 are summarised in Table 4.5.

Original images provided by the Oxford-102 dataset are high-resolution images, which are resized into 64 by 64 to be our training images. In all our experiments, we set the number of Gaussians (i.e. K) as 10.

Algorithm 4 Training Strategy

Require: $X = [x_1, x_2, \dots, x_N]$ - N training images in the entire training set

K : number of components in the latent GMM distribution

d : the dimension of the latent space

c : the initial value range of Gaussian means

σ : scaling factor of covariance matrices

- 1: **for** $k = 1 \dots K$ **do**
 - 2: $\mu_k \sim U[-c, c]^d$ {Initialize the mean of k^{th} Gaussian}
 - 3: $\Sigma_k \leftarrow I_{d \times d}$ {Initialize the covariance matrix of k^{th} Gaussian}
 - 4: **end for**
 - 5: **for** each epoch till convergence **do**
 - 6: **for** $i = 1 \dots N$ **do**
 - 7: Sample $k \sim \text{Categ}(\frac{1}{K}, \dots, \frac{1}{K})$ {Select a Gaussian}
 - 8: $z_i \sim \text{GMM}(\mu_k, \Sigma_k)$ {Sample from k^{th} Gaussian}
 - 9: $\hat{x}_i \leftarrow G(z_i)$ {Generate a synthetic sample}
 - 10: Update G and D with $\mathcal{L}^{\text{adversarial}}(x_i, \hat{x}_i)$ using the loss function described in Equation 2.1
 - 11: Update G and C with \mathcal{L}^C using the loss function described in Equation 4.10
 - 12: **end for**
 - 13: **end for**
-

Table 4.5 : Details of the different datasets used in the empirical evaluation.

Dataset	# Classes	Samples Dimensions	# Train	# Val	# Test	D_{img}	D_h
CIFAR-10	10	$32 \times 32 \times 3$	50,000	-	10,000	32	3
Oxford-102	102	$64 \times 64 \times 3$	1,020	1,020	6,149	64	3

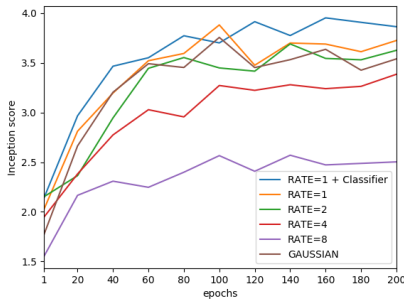
Evaluation metrics It is difficult to measure the performance of image generation in a quantitative way, so used two popular metrics were used for automatic image quality evaluation: the IS [86] and the FID score [35]. A higher IS and a lower FID score indicate a better generation performance.

Inception and FID Scores

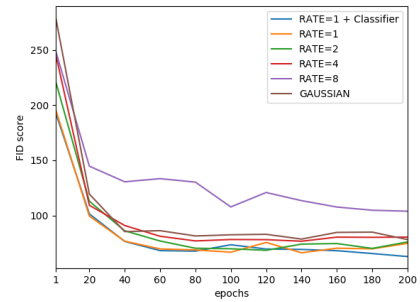
This section reports the inception and FID scores of GMM-based GAN with a various compression rates of r compared with the baseline vanilla GAN. Figure 4.8 shows the inception and FID scores over training epochs for Gaussian based and GM-GANs. Parameters of the GMM are trainable on all GM-GANs.

As shown in Figure 4.8 (a) for the CIFAR-10 dataset, when the networks have the same size (i.e. the compression rate $r = 1$), GM-GAN achieves a constantly higher IS from the beginning of training compared with the vanilla GAN. When r is increased to 2, GM-GAN has a similar IS as the vanilla GAN. In addition, our modified GM-GAN with the additional classifier and $r = 1$ achieves an even higher IS compared with the result without the classifier. Similar results of higher IS for GM-GMM and the highest IS for our model can be found in Figure 4.8 (c) on the Oxford-102 datasets.

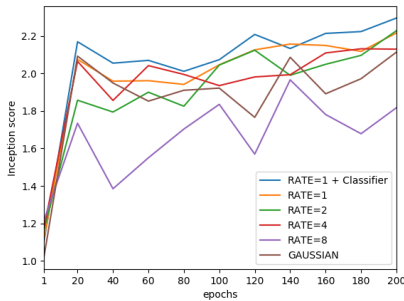
In terms of the FID scores on the CIFAR-10 dataset, GM-GAN constantly yields a lower FID score than does the Gaussian based GAN when using the same size of the network, and a network with $r = 2$. The network with $r = 4$ produces similar



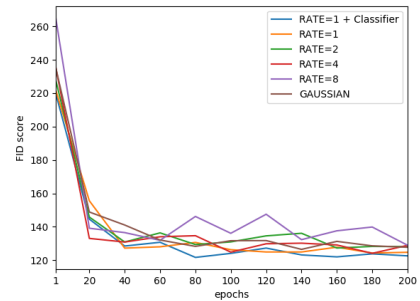
(a) Inception scores on the CIFAR-10 dataset



(b) FID scores on the CIFAR-10 dataset



(c) Inception scores on the Oxford-102 dataset



(d) FID scores on the Oxford-102 dataset

Figure 4.8 : Inception and FID scores over training epochs under different compression rates of GM-GAN and vanilla GAN on the CIFAR-10 and Oxford-102 dataset.

FID scores over training epochs compared with Gaussian based GAN, even though the network size of the latter is three times larger. FID scores on the Oxford-102 dataset also show that GM-GAN achieves a better performance under a smaller network size than the Gaussian based model (i.e., when $r = 2$).

The value of the exact number of parameters used, and on average the highest inception and lowest FID scores in each experiment are summarised in Table 4.6.

Table 4.6 indicates that when using the same size of the network, GM-GAN achieves a 3.4% higher inception score and 15.3% lower FID score on the CIFAR-10

Table 4.6 : Number of parameters, final Inception and FID score for each experiment.

Dataset	Compression Rate	# parameters	Inception score \uparrow	FID score \downarrow
CIFAR-10	1 + classifier	3,002,267	3.9523 ± 0.1874	62.7941 ± 3.3387
	1	2,920,337	3.8809 ± 0.1955	66.2328 ± 2.3344
	2	1,153,969	3.6899 ± 0.0734	68.4550 ± 3.2691
	4	501,185	3.3850 ± 0.1393	76.6934 ± 2.9099
	8	232,393	2.5700 ± 1.1925	103.9022 ± 8.3241
	GAUSSIAN	2,918,337	3.7555 ± 0.1285	78.1733 ± 2.8900
Oxford-102	1 + classifier	8,286,107	2.2961 ± 0.1751	121.6527 ± 4.0959
	1	7,958,417	2.2287 ± 0.0704	124.0927 ± 13.1122
	2	3,673,009	2.2164 ± 0.0704	124.2820 ± 8.1919
	4	1,760,705	2.1319 ± 0.0967	127.2721 ± 4.9043
	8	862,153	1.9660 ± 0.1780	128.9227 ± 7.2831
	GAUSSIAN	7,956,417	2.1142 ± 0.0357	126.4798 ± 1.3708

dataset and a 5.4% higher inception score and 1.9% lower FID score on the Oxford-102 dataset than the vanilla GAN. This indicates that GM-GAN outperforms the vanilla GAN with a higher generation quality for both datasets.

In addition, GM-GAN when the compression rate is two, still achieves a 12.4% lower FID on the CIFAR-10 dataset, 4.8% higher inception score and 1.7% lower FID score on the Oxford-102 dataset compared with the vanilla GAN. The network size shrinks 39.5% and 46.2%, respectively.

GM-GAN with additional classifier outperforms those without the classifier for both datasets. It achieves a 1.8% higher inception score and 5.2% lower FID score on the CIFAR-10 and a 3.6% higher inception score, and 2.0% lower FID score on the Oxford-102 dataset.

The experiments in this subsection demonstrate that modelling the latent space with GMM can significantly reduce the size of the network by at least 50% while

generating images with similar quality.

Performance on Anomaly Detection

Experiment design The advantage of modelling the latent space with GMM to perform anomaly detection is further demonstrated on the MNIST [57], Fashion-MNIST [101], the CIFAR-10 dataset [52] and the KDD-99 10 percent dataset [34]. Statistics of these datasets are reported in Table 4.7.

Table 4.7 : Different datasets used in the empirical evaluation

Dataset	# Classes	Samples	Dimensions	# Train	# Val	# Test	D_{img}	D_h
MNIST	10	$28 \times 28 \times 13$		60,000	-	10,000	28	1
FashionMNIST	10	$28 \times 28 \times 13$		60,000	-	10,000	28	1
CIFAR-10	10	$32 \times 32 \times 3$		50,000	-	10,000	32	3
Oxford-102	102	$64 \times 64 \times 3$		1,020	1,020	6,149	64	3
KDD-99	2	121		198,361	-	247,011	-	-

The MNIST, Fashion-MNIST and CIFAR-10 datasets contain 10 classes so the training and evaluation processes are performed on each class individually (i.e. one class is considered as the anomaly class, and the remaining nine classes are the normal classes).

We follow the training strategy as described in AnoGAN [88] in our experiments. Therefore, only the normal data from the training set are used for training, and the whole test set is used in the evaluation. Once the GAN network is fully trained, 500 iterative steps are taken to map the input sample to the latent space. Given a sample x and its latent counterpart z , its anomaly score is calculated as the dissimilarity between x and the reconstructed sample $G(z)$ on the input and the feature level:

$$A(x) = (1 - \lambda) \cdot \|x - G(z)\|_1 + \lambda \cdot \|f(x) - f(G(z))\|_1, \quad (4.7)$$

where f is an intermediate layer of the discriminator, and λ is set to 0.1. A higher anomaly score indicates that the sample is more likely to be anomalous. In all experiments, the number of Gaussians (i.e. K) is set to be 10. Tests are made measuring the AUC value.

Evaluation metric As the anomaly score does not determine whether a sample is anomalous or not unless a threshold is set, we the Area under the ROC Curve (AUC) is chosen to evaluate the model performance. AUC measures the area under a receiver operating characteristic (ROC) curve, which plots true positive rates against false positive rates with varied discrimination thresholds.

AUC values Figure 4.9 plots the AUC value over anomaly classes 0 to 9 on the MNIST, FashionMNST and CIFAR-10 datasets.

As shown in Figure 4.9(a), in 6 out of 10 anomaly classes, GM-GAN achieves a higher AUC value than the vanilla GAN. In 5 out of 10 anomaly classes, GM-GAN with a compression rate at 2 still outperforms the vanilla GAN. In Figure 4.9(b), GM-GAN performs better in 8 out of 10 anomaly classes, and the compression rate = 2 achieves a higher AUC value in 5 out of 10 classes. A similar pattern can also be found in Figure 4.9(c), that GM-GAN performs no worse than the vanilla GAN in 6 out of 10 classes, even with a compression rate of two.

In addition, the additional classifier applied to the GM-GAN provides an even more enhanced anomaly detection result because it achieves the highest AUC value in 6 out of 10 anomaly classes on the MNIST and Fashion-MNIST dataset and 5 out of 10 on the CIFAR-10 dataset. The number of parameters for each algorithm and the average AUC value are reported in Table 4.8.

As Table 4.8 shows, GM-GAN can achieve, on average, 3.6%, 0.7%, and 0.8% higher AUC values on the MNIST dataset, when sizes of the networks are the same

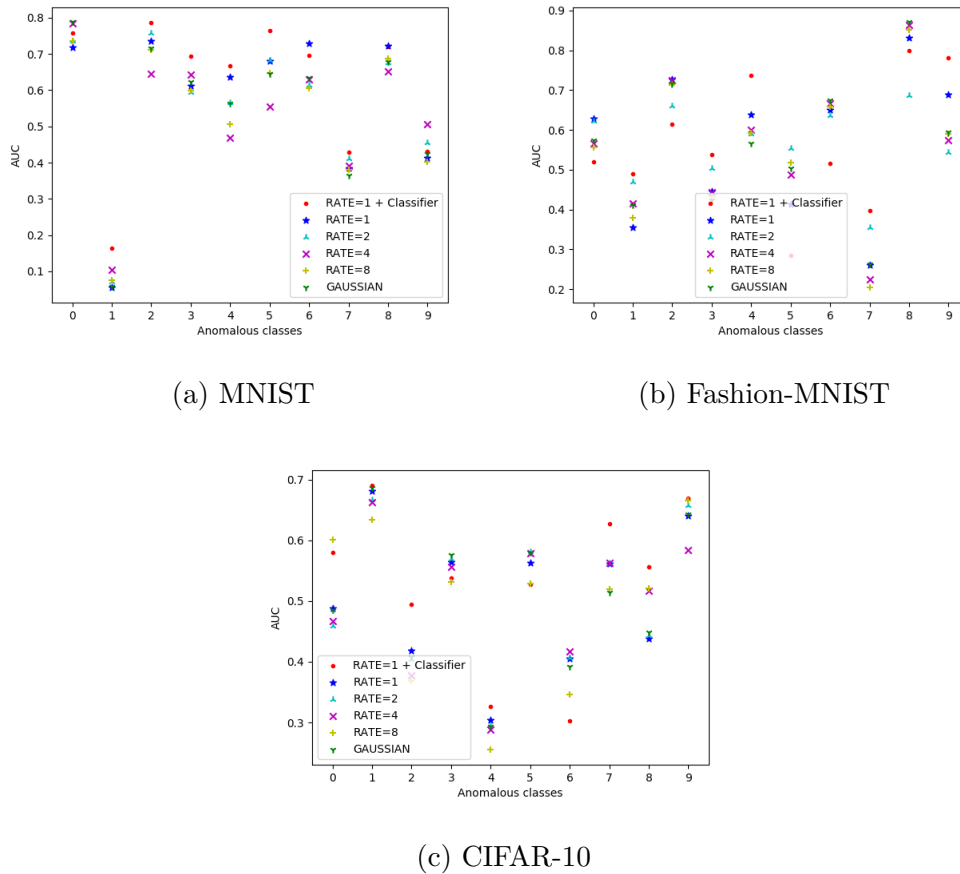


Figure 4.9 : Performance of anomaly detection measured by area under curve. (a) AUC over anomalous classes on the MNIST dataset; (b) AUC over anomalous classes on the Fashion-MNIST dataset; (c) AUC over anomalous classes on the CIFAR-10 dataset.

for both algorithms. GM-GANs can still achieve higher AUC values on all three datasets with a compression rate of 2. On the KDD-99 dataset, using GMM with a compression rate of 8 outperforms the original AnoGAN using only 6.9% of the number of parameters.

In addition, adding a classifier to the network allows a more accurate anomaly detection output with a minimal increase in the network size. As Table 4.8 shows, AUC value is further enhanced by 7.4%, 0.7%, and 4.9% on the three datasets

compared with the ones without the classifier.

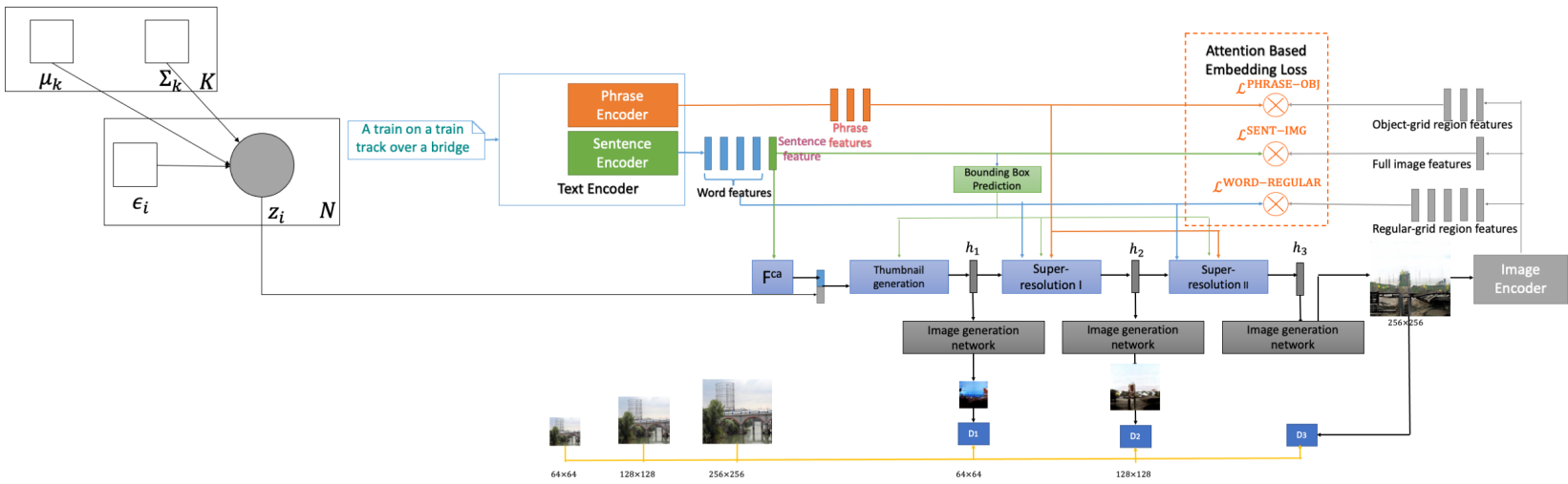
With the results of the experiments in this subsection, we can conclude that modelling the latent space with GMM allows GAN to achieve a better anomaly detection performance with only half the number of parameters. This is consistent with the findings in image generation that GM-GAN and its variant with much fewer parameters are superior to, or at least competitive with, Gaussian GAN.

4.3 GMM in Text-to-image Generation

This section describes how to apply GMM to domain transfer and report the performance. In the text-to-image generation, the concatenation of a noise vector z and sentence vector is fed to GAN network as the input. The vanilla GAN models z with standard Gaussian, and in our work, the distribution of z is modelled as a GMM.

The network structure, shown in Figure 4.10, is based on the one described in Section 3.1: images are generated through a multistage process and the attention structure built between image and text is utilised to further improve the generation quality. The attention structure is constructed as twofold: between word and regular-grid region and between phrase and object-grid region. The only difference to that described in Section 3.1 is that z is now sampled from a GMM distribution.

Figure 4.10 : Overall architecture of GMM based text-to-image generation.



The density function of the latent distribution is defined as $p(z) = \sum_{k=1}^K \alpha_k \mathcal{N}(z; \mu_k, \Sigma_k)$, where K is the total number of Gaussians in the mixture, and k^{th} component is characterised by a Gaussian distribution with weight α_k , mean μ_k and covariance matrix Σ_k . We assume that each Gaussian takes equal weight (i.e. $\alpha_k = \frac{1}{K} \quad \forall k = 1, \dots, K$).

In addition, our work also makes the parameters of the latent distribution, $\{\mu_1, \dots, \mu_K, \Sigma_1, \dots, \Sigma_k\}$, to be trainable, and the training is performed “end-to-end” with the GAN network. Therefore, instead of sampling from the GMM directly during training, the reparameterisation trick is applied so that the back-propagation can be used to update μ_k and Σ_k of each Gaussian.

Datasets

Experiments are performed on the CUB-200 [99] and MSCOCO [63] dataset. The statistics of both datasets are reported in Table 4.9. Processing and training the MSCOCO dataset are performed in the same way as described in Section 3.1, where the bounding boxes are used to collect object-grid region features. The test set in the MSCOCO dataset contains only images and no captions, so the validation set is used as the test set.

In terms of the bounding box processing, while each image in the MSCOCO dataset usually contains multiple objects and the dataset provides multiple bounding boxes accordingly, CUB-200 contains only one object in each image. Thus, it provides only one bounding box. The single bounding box outlines the main bird object from the background. In addition, captions from CUB-200 describe details about the bird, such as the colour of the beak or feathers, while the bounding box of the beak or the feather is not provided accordingly. Thus, the previous construction of the phrase-to-object-grid region, which expects multiple objects in the image would match multiple phrases in the caption, is not applicable to the CUB

dataset. Therefore, the object-grid region feature is replaced with the regular-grid region feature in the attention mechanism. In addition, because the bird object is the majority of an image and its caption, the single bounding box is used to crop the image before feeding to the network to simplify the training.

All experiments are performed with the dimension of D as 100 and learning rate as 0.0002 for both the generator and the discriminator. Experiments on the CUB dataset are performed over 600 epochs; experiments on the MSCOCO dataset are performed over 120 epochs.

Evaluation Metrics

The evaluation metrics used are the same as in Section 3.1, inception score [86] and FID score [35], which measures the generation quality, and R-precision [103], which measures how closely the generated image fits its exact description.

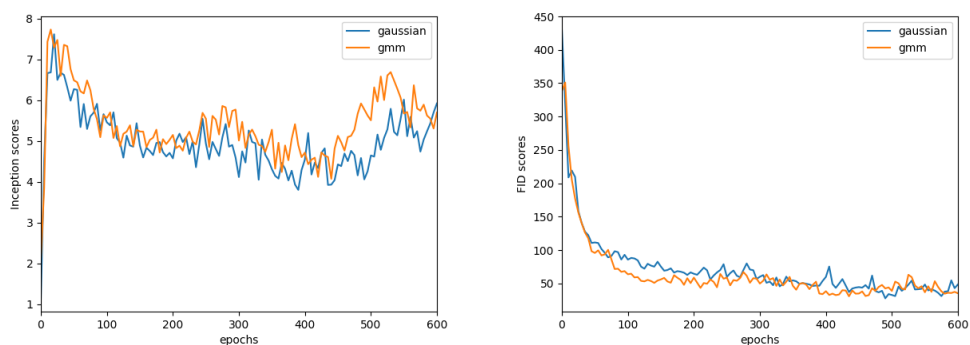
Experimental Results

Figure 4.11 shows the inception and FID scores of the Gaussian and GMM based GANs over epochs on the CUB and MSCOCO datasets.

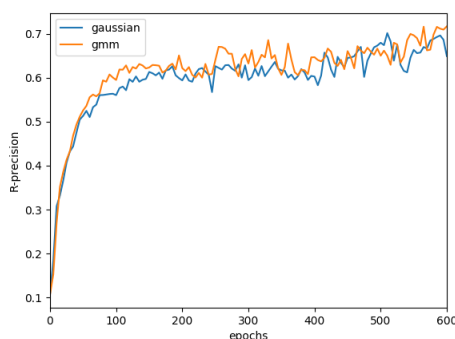
As in Figure 4.11, GMM based GAN achieves a higher IS and a lower FID score constantly throughout the entire training process compared with the vanilla GAN. This indicates that using GMM to model the latent space leads to a higher image generation quality.

4.4 Dirichlet Process GAN

The overall architecture of the proposed DP-GAN is shown in Figure 4.12. We describe in each iteration how the random noise z is generated via particle filter and how the parameters of the latent distribution and the network are updated. The overall training flow of the proposed algorithm is summarised in Algorithm 5, and



(a) Inception scores on the CUB dataset (b) FID scores on the CUB dataset



(c) R-precision on the CUB dataset

Figure 4.11 : Comparison between the Gaussian and GMM based GANs in terms of the inception and FID scores over epochs on the CUB and MSCOCO dataset.

its details are elaborated upon in the subsequent sections.

4.4.1 Generate z via Particle Filter

We first need data to train the DPGMM parameters. As explained previously, z instead of x is used as the data of the DPGMM. However, the vanilla GAN does not allow a direct transformation from x to its correspondence in latent space. Therefore, innovative ways are required to evaluate z using the x it generates. Inspired by particle filter, we generate and then evaluate z in which its overall flow is shown in Figure 4.13.

Algorithm 5 Overall training flow of the the proposed DP-GAN

for each epoch until convergence **do**

 Generate z via particle filter as in Section 4.4.1

- proposal (first-part) from DPGMM, $z \sim \text{DPGMM}(\theta)$
- proposal (second-part) $G(z)$
- weighting using $D(G(z))$
- resampling from $D(G(z))$

 Update DPGMM parameters

 Update rest of parameters

- The responsibility consistency module (RCM)
- Update the generator
- Update the discriminator

end for

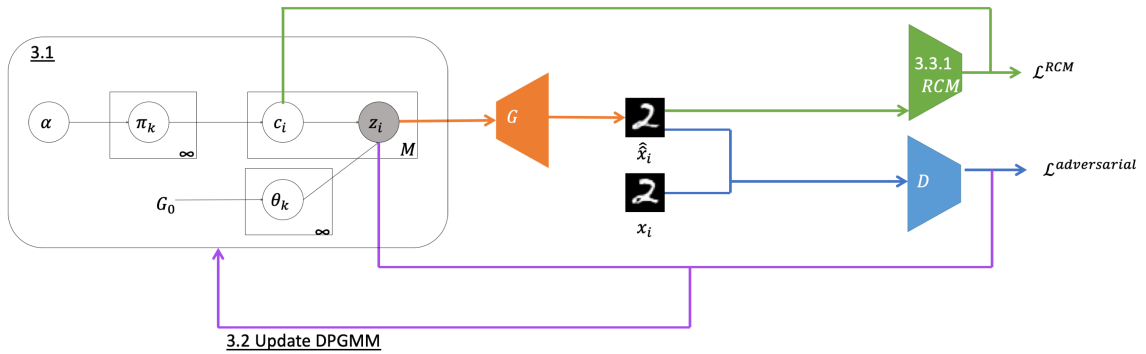


Figure 4.12 : The complete architecture of DP-GAN.

SMC or particle filter is a Monte Carlo technique used to solve Recursive Bayesian statistical inference. We use the simplest particle filter (i.e., condensation filter) to illustrate what occurs during each particle filter iteration:

- i. Provide a set of particles from the previous iteration $t - 1$, $\{z_{t-1}^i\}$
- ii. Sample from a proposal distribution: $z_t^i \sim p(z_t|z_{t-1}^i)$
- iii. Compute the weights $w_t^i \propto \pi_{t-1}^i p(x_t|z_t^i)$
- iv. Normalise weights $\pi_t^i = \frac{w_t^i}{\sum_{i=1}^N w_t^i}$

Here, z_i and x_t may also refer to the latent and observed.

To show the similarity between traditional particle filters versus our proposed work, we show them both side-by-side in 4.13 as well.

Proposal from DPGMM After randomly initialising parameters of the DPGMM, sampling z can be performed as described in Section 2.4.3. The two hyperparameters, α : the concentration parameter and D : the dimension of samples z are defined manually. Sampling results are fed to the generator $G(z)$ to obtain the proposed sample \hat{x} .

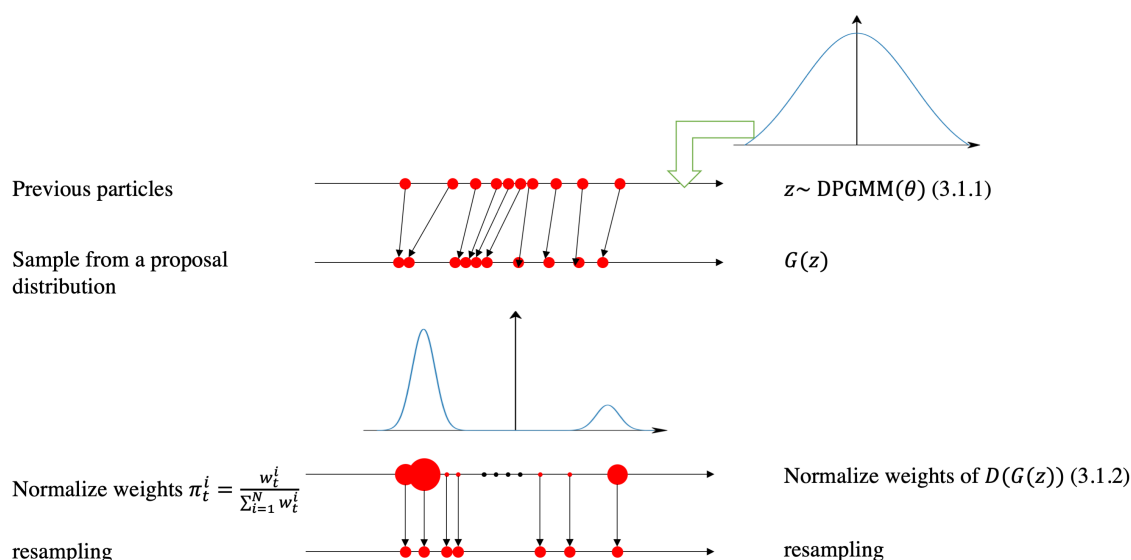


Figure 4.13 : The overall flow of how the latent DPGMM distribution is updated. Notations on the left are the traditional flow of particles; notations on the right are how it is applied in the proposed architecture.

Weighting and resampling Suppose all random vectors sampled in the last epoch are grouped as $Z \in R^{M \times D}$, where M is the size of the training set. Just like in particles, each needs to be weighed according to the current observation. Here, we use the trained discriminator D to do this (i.e., $D(G(Z))$ are scores produced by the discriminator D). Higher scores should reflect a better-generated samples, as judged by the current D . Subsequently, the algorithm increases the weights of those random vectors that lead to higher scores in the generation and decrease the weights of those with lower scores.

Just like particle filters, here the proposed approach also faces “particle collapsing”, a phenomenon where there exist only a few dominant particles with significant weight and the rest are close to zero. Therefore, the off-the-shelf resampling strategies described in [22] are used. Our work proposes to apply multinomial resampling to all random vectors sampled in the last epoch. The resampling result is used to

fit the DPGMM and update the parameters. Probabilities of selecting each vector are calculated by normalising the D score of its corresponding synthetic sample.

4.4.2 Updating DPGMM Parameters

There are many ways to update the DPGMM. Here, we have used the one described in [9], which is a mean-field variational algorithm for the DP mixture. The variational bound is written as:

$$\begin{aligned} \log p(x|\alpha, \lambda) &\geq E_q[\log p(\mathbf{V}|\alpha)] + E_q[\log p(\boldsymbol{\eta}^*|\lambda)] \\ &\quad + \sum_{n=1}^N (E_q[\log p(c_n|\mathbf{V})] + E_q[\log p(x_n|c_n)]) \\ &\quad - E_q[\log q(\mathbf{V}, \boldsymbol{\eta}^*, \mathbf{C})], \end{aligned} \quad (4.8)$$

where \mathbf{V} , $\boldsymbol{\eta}^*$, \mathbf{C} denote stick lengths, atoms, and the cluster assignment variables respectively.

Authors have proposed the following factorised family of variational distributions to approximate the distribution of the infinite-dimensional random measure G :

$$q(\mathbf{v}, \boldsymbol{\eta}^*, \mathbf{c}) = \prod_{t=1}^{T-1} q_{\gamma_t}(v_t) \prod_{t=1}^T q_{\tau_t}(\eta_t^*) \prod_{n=1}^N q_{\phi_n}(c_n), \quad (4.9)$$

where $q_{\gamma_t}(v_t)$ are beta distributions, $q_{\tau_t}(\eta_t^*)$ are exponential family distributions and $q_{\phi_n}(c_n)$ are multinomial distributions. For details of the algorithm, please refer to [9].

4.4.3 Updating Other Parameters

Responsibility Consistency Module We design the RCM to output $p(c|\hat{x})$, given a synthetic sample \hat{x} . $p(c|\hat{x})$ needs to match against $p(c|z) = \pi$ which is the responsibility probability condition on its latent variable z . The corresponding loss is calculated as:

$$\mathcal{L}^{RCM} = \mathbb{E}_{x_m \in p_{\text{data}}} [I(p(c|z), p(c|\hat{x}))]. \quad (4.10)$$

Essentially, all we need is a measurement function to compute the distance between two distributions, so KL or DS divergence can also be used. However, in the experiment, mutual information loss provides the best result.

RCM is chosen to share the feature encoding layer with the discriminator, during training, \mathcal{L}^{RCM} is applied to optimise both the generator G and the simple standard linear softmax classifier. The feature encoding layers are updated by the discriminator only.

Other GAN Parameter Updates

Generator The generator G simply takes a random vector z sampled from the DPGMM to generate synthetic samples. We chose the structure of DCGAN to build our model, which is reported in Table 4.10.

Discriminator The discriminator D consists of multiple layers of CNN, which we refer to as the feature encoding layers followed by a standard linear logistic regression to identify the given image as real or fake. The design makes the discriminator and the responsibility consistency module share the same feature encoding layers. Parameters of those layers are updated by the adversarial loss only.

GAN network architecture Details of the network design can be found below in Table 4.10. In Table 4.10, we use abbreviations for operations in the table: “Conv” is the convolution operation, of which the kernel and stride size are in the parenthesis; “LeakyReLU” is short for “leaky rectified linear unit” and “Tanh” are activation functions. “Batch norm” is short for batch normalisation; “Flatten” refers to the

operation that flattens a tensor to a 1D array. D_{img} , D_h and D_z are related to the datasets we use, the exact value of each are reported in Table 4.11.

4.4.4 Experiments

Performances of the proposed DP-GAN are demonstrated on a toy dataset, the MNIST [57] and the Fashion-MNIST [101] dataset. Details of each datasets are summarised as below in Table 4.11.

Toy Data Generation

The first dataset I use is a simple toy dataset. This dataset is constructed to gain more intuition regarding the properties of the DP-GAN model. The dataset contains 1,000 training samples which are generated by sampling from a homogenous mixture of L Gaussians. Below, I show the result when $L = 2$, $\forall k \in \{1, \dots, L\}, \Sigma_k = 1, \mu = \{5, -5\}$. For the toy dataset, we do not use the network structure shown in Table 4.4 to perform the training, the three networks, G , D and RCM each contains only a one-layer neural network.

Figure 4.14 shows samples from the data distribution and the synthetic samples. As shown in Figure 4.14c, most stick weights are concentrated on two Gaussians. Therefore, I also fit the synthetic samples to a homogenous mixture of two Gaussians in Figure 4.14 for demonstration purposes.

As in Figure 4.14a, samples generated by the vanilla GAN are mostly distributed between the two Gaussians of the data distribution, which makes the distribution of the synthetic data far from the data distribution. Figure 4.14b shows that the proposed DP-GAN is able to generate samples that match much closer to the original distribution than the vanilla GAN. In addition, Figure 4.14c shows that most stick weights are concentrated on two Gaussians after the model is fully trained. A threshold can be easily set to predict the correct number of classes even though the

training is performed in a completely unsupervised manner.

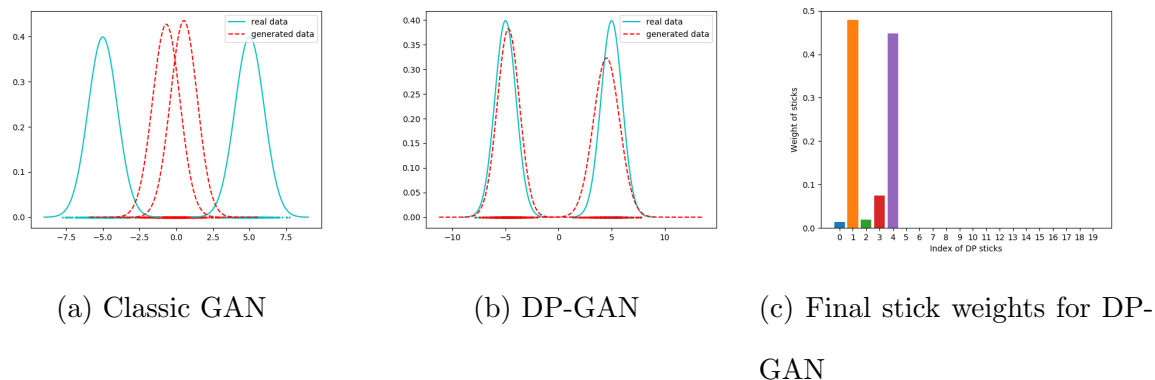


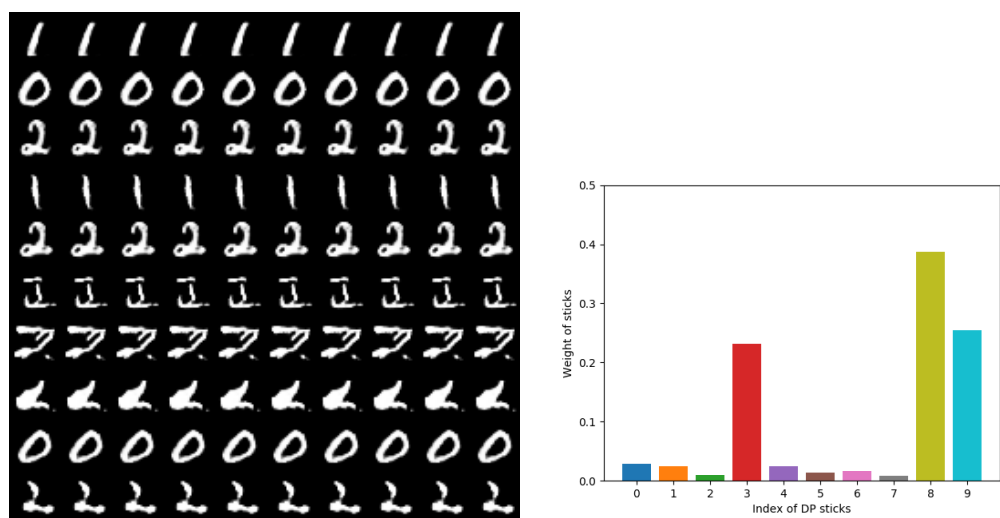
Figure 4.14 : Samples from the toy dataset and generated from (a) GAN and (b) DP-GAN. Samples from the training set and its PDF are plotted in blue and synthetic samples and the PDF of their distribution are plotted in red. (c) Stick weights for the DPGMM when the model is fully trained.

MNIST Generation

I next demonstrate the performance on more complex datasets. The second dataset used is a subset of the MNIST dataset, which includes all 0, 1 and 2 digits in the dataset (i.e. the optimal number of clusters of the dataset is 3).

Figure 4.15a shows synthetic samples generated by the proposed DP-GAN. Each row contains samples generated from each Gaussian of the DPGMM. It shows clearly that even though the training is performed in a completely unsupervised manner, each mode in the latent space reflects a particular image class.

In addition, Figure 4.15b reports the final weights for each component of the DPGMM. Most weights are concentrated on three Gaussians with indices of 8, 9 and 3. A threshold can be easily set to predict the number of classes. Random vectors from these three Gaussians generate high-quality samples in the top three rows in Figure 4.15a.



(a) Synthetic Samples

(b) Final stick weights for DP-GAN

Figure 4.15 : (a) Samples generated by DP-GAN after trained on the MNIST dataset. Each row contains images sampled from a different Gaussian. Rows are sorted using the components' weights in the descending order. (b) Stick weights for the DPGMM when the model is fully trained.

Fashion-MNIST

The third dataset we use is the Fashion-MNIST dataset. Similar to the MNIST dataset, we select the first three categories of the original dataset as our training set, and the optimal number of clusters of the dataset is three.

Figure 4.16a shows synthetic samples generated by the proposed DP-GAN. It indicates that vectors sampled from each component of Gaussian generate a specific class of clothing images. Rows in Figure 4.16a are sorted using the final weights of components in the descending order. Top rows, which correspond to Gaussian indexed by 0, 4 and 7, contain images of higher-quality and the first three rows match against all three classes from the training set.

In addition, Figure 4.16b reports the final stick weights of the DPGMM. Most

weights are concentrated on three Gaussians with indices of 0, 4 and 7; a threshold can be easily set to predict the number of classes.

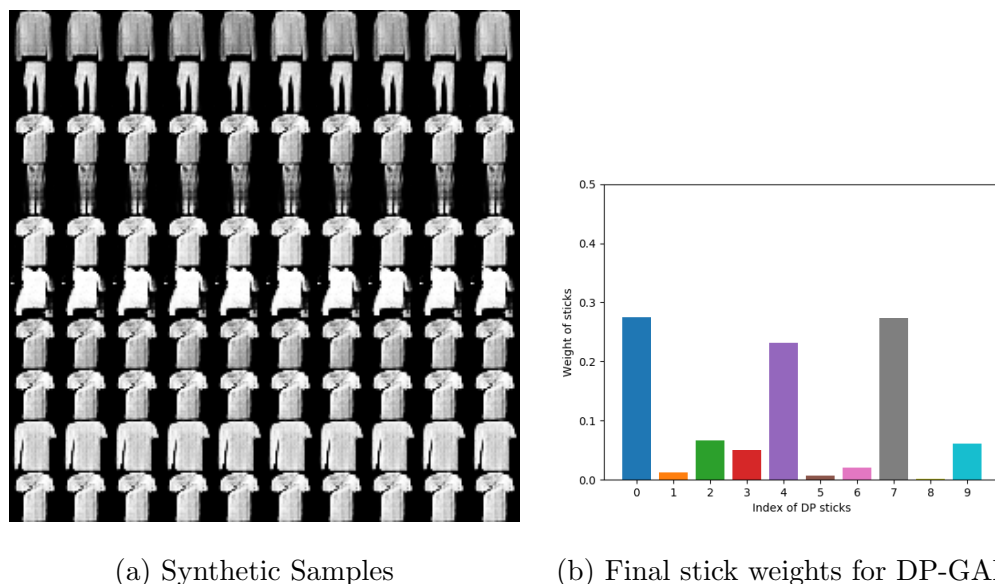


Figure 4.16 : (a) Samples generated by DP-GAN after training on the MNIST dataset. Each row contains images sampled from a different Gaussian. Rows are sorted using the components' weights in the descending order. (b) Stick weights for the DPGMM when the model is fully trained.

4.5 Summary

This chapter describes our contributions in terms of latent space modelling of GANs. In particular, our work studies the effects and properties learnt by applying GMM and DPGMM to GANs.

Given a data point x , a Gaussian Mixture Model (GMM) with K components may assume the existence of a random index $k \in \{1, \dots, K\}$ identifying with which Gaussian the particular data are associated with. In a traditional GMM paradigm, it is straightforward to compute in closed-form, the conditional likelihood $p(x|k, \theta)$ and the responsibility probability $p(k|x, \theta)$ describing the distribution weights for each

data. Computing the responsibility allows us to retrieve many important statistics of the overall dataset, including the weights of each mode/cluster. Modern large datasets often contain multiple unlabelled modes. For example, a paintings dataset may contain several styles or fashion images may contain several unlabelled categories. In its raw representation, the Euclidean distances between the data (e.g., images) do not allow them to form mixtures naturally, nor is it feasible to compute responsibility distribution analytically, making application of GMM not possible. Our work proposes to utilise the GAN framework to achieve a plausible alternative method to compute these probabilities. The key insight is they are computed at the data’s latent space z instead of x . However, this process of $z \rightarrow x$ is irreversible under GAN, which renders the computation of responsibility $p(k|x, \theta)$ infeasible. Our paper proposes a novel method to solve it by using a so-called PCM. PCM acts like a GAN, except its generator C_{PCM} does not output the data; instead, it outputs a distribution to approximate $p(k|x, \theta)$. The entire network is trained in an “end-to-end” fashion. Through these techniques, it allows us to model the dataset of very complex structures using GMM and subsequently to discover interesting properties of an unsupervised dataset, including its segments, and generate new “out-distribution” data by smooth linear interpolation across any combinations of the modes in a completely unsupervised manner.

Our work also investigates the network compression effect achieved by GM-GAN. In particular, we present a modified GM-GAN with a classifier to encourage the matching between the component from the latent distribution and the class from the synthetic data. We quantitatively compare the image generation quality of these GANs with different structures and sizes. We further demonstrate how anomaly detection is performed using improved GM-GAN with a classifier. Experimental results show that modelling the latent space with GMM together with the classifier in GAN can significantly reduce the number of parameters of GANs while achieving

a similar generation quality and performance in anomaly detection.

We further propose a novel GAN-based algorithm DP-GAN that models the latent space of GANs using the DPGMM distribution. DP has been popularly used in Bayesian non-parametric (BNP) statistics to simultaneously study the distribution parameters and number of distinct clusters. However, its application has been mostly restricted in areas in which the individual component likelihood $f_{\theta_i}(x)$ can be appropriately defined. Modern datasets, such as large image datasets, do not usually have ways to define $f_{\theta_i}(x)$, making DP or mixture models unable to apply. A logical solution is to apply the DP mixture model in the latent space instead. Many modern generative models offer us such a mechanism. One of them is the popular Generative Adversarial Networks (GAN). To this end, we have proposed a framework in which to use GAN to project data from its original representation x into its latent ones (i.e., z) so that the likelihood is to be defined over $f_{\theta_i}(z)$ instead. DNN-based GAN has been popularly used in data generation, where historically, only standard distribution $p(z)$ is used to model the latent space, not only making it non-informative, but also causes a number of issues, such as mode collapse, a phenomenon in which the samples generated look very similar. Therefore, the proposed approach has addresses two issues with one solution: First, we have designed a way to model a potentially unknown number of mixtures over a modern dataset. Second, improvements are made over the vanilla-GAN to achieve better sample diversities and have prevented mode collapse.

However, a naive application of the DPMM to z via GAN transformation is prohibitive because GAN, by construction, is only designed to transform latent z to x . The inverse transformation from data to its latent representation is not trivial. To solve this problem, we have proposed using a sample reweighting mechanism, often observed in SMC (i.e., particle filter methodologies). We use the GAN mechanism itself to “weigh” each sample of z by examining the quality of their corresponding

x generated. Further, to ensure that in each iteration, the distribution of both the data and the latent representation is in line, we have devised a novel Responsibility Consistency Module (RCM) to achieve this alignment. Through a series of experiments performed on both synthetic and real-world datasets, we demonstrate that our proposed approach can accurately and meaningfully model the dataset via its latent space: Each mode in the latent space reflects a particular data cluster in an unsupervised manner. Simultaneously, the model does not comprise GAN’s original purpose (i.e., it can also generate high-quality, realistic image data).

Table 4.8 : Number of parameters and AUC on the MNIST, FashionMNIST, CIFAR-10, KDD-99 dataset. Compression rate = 1, 2, 3, 4 refer to GM-GAN with the corresponding compression rate.

Dataset	Compression Rate	# parameters	AUC
MNIST	1 + classifier	2, 583, 067	0.6112 \pm 0.1922
	1	2, 520, 337	0.5693 \pm 0.2096
	2	953, 969	0.5563 \pm 0.1941
	4	401, 185	0.5382 \pm 0.1791
	8	182, 393	0.5345 \pm 0.1925
	GAUSSIAN	2, 518, 337	0.5493 \pm 0.2042
FashionMNIST	1 + classifier	2, 583, 067	0.5676 \pm 0.1581
	1	2, 520, 337	0.5636 \pm 0.1740
	2	953, 969	0.5621 \pm 0.0949
	4	401, 185	0.5564 \pm 0.1689
	8	182, 393	0.5491 \pm 0.1718
	GAUSSIAN	2, 518, 337	0.5598 \pm 0.1638
CIFAR-10	1 + classifier	3, 002, 267	0.5311 \pm 0.1234
	1	2, 920, 337	0.5063 \pm 0.1106
	2	1, 153, 969	0.5047 \pm 0.1153
	4	501, 185	0.5011 \pm 0.1071
	8	232, 393	0.4970 \pm 0.1258
	GAUSSIAN	2, 918, 337	0.5025 \pm 0.1155
KDD-99	1 + classifier	114, 452	0.4948 \pm 0.0128
	1	113, 162	0.4876 \pm 0.02970
	2	43, 306	0.4563 \pm 0.02970
	4	19, 130	0.4427 \pm 0.0351
	8	9, 730	0.4426 \pm 0.0748
	GAUSSIAN	111, 162	0.3987 \pm 0.0331

Table 4.9 : Statistics of the CUB-200 and MSCOCO dataset.

Dataset	train	test	validation	captions per image
CUB-200	5,994	5,794		10
MSCOCO	82,783	40,775	40,504	5

Table 4.10 : The overall network structure used for the real-world datasets.

Stage	Sub-stage	Name	Input Tensors	Output Tensors
<i>RCM</i>	Encoding Network (shared)	Conv (kernel=5, stride=2) + LeakyReLU	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$
	Classification Network	Linear	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$	K
<i>G</i>		Linear + Batch norm + LeakyReLU + Reshape	$1 \times D_z$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 256$
		Transposed Conv (kernel=5, stride=1) + Batch norm + LeakyReLU	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 256$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$
		Transposed Conv (kernel=5, stride=2) + Batch norm + LeakyReLU	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$
		Transposed Conv (kernel=5, stride=2) + Tanh	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$	$D_{\text{img}} \times D_{\text{img}} \times D_h$
<i>D</i>	Encoding Network (shared)	Conv (kernel=5, stride=2) + LeakyReLU	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$
	Discriminator Network	Linear	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$	1

Table 4.11 : Statistics of the different datasets used in the empirical evaluation.

Dataset	# Classes	Samples	Dimensions	# Train	# Test	D_{img}	D_h
Toy-Dataset	2	1		1,000	-	-	-
MNIST	10	$28 \times 28 \times 1$		60,000	10,000	28	1
FashionMNIST	10	$28 \times 28 \times 1$		60,000	10,000	28	1

Chapter 5

Training Acceleration in Style Transfer

As described in Chapter 1, DNN based cross domain training often suffers from large training costs and is extremely difficult to converge. Previously, researchers proposed DM-SGD which applied Determinantal Point Process (DPP) to shallow neural networks and achieved promising outcomes. However, there are two drawbacks of DM-SGD:

Firstly, when raw input features are used, it measures diversities in terms of the Euclidean distance in the input space. This is prohibitive; in fact, the very reason neural network is applied is to transform data non-linearly from its original space into a layer before the output is so that the features becomes a great deal more expressive. Therefore, if any features are to be used to guide the mini-batch selection, we need to use them before the last layer instead of before the first input. However, this also creates another problem in that the network parameters are not static and are updated throughout the iterations; Therefore, we must use the updated features at each iteration as opposed to what DM-SGD proposes.

Second, mini-batch selection using DPP may become prohibitively slow when the entire set of data Ω is used to compute its Gram matrix, even if this operation is only computed once. Therefore, the “fixed” feature approach is somewhat only theoretically plausible.

For these reasons, we must update the features to compute the Gram matrix at each iterations. Given sampling DPP once is already far too much computation, sampling at each iteration with an updated Gram matrix would further increase

the already-infeasible complexities. Therefore, it is natural to use an approximated DPP to dramatically reduce its computation time.

By observing this in the last or higher layer, we found that features within each class have a higher tenancy to stay closer and data between classes tend to separate. Therefore, we chose the last FC layer to construct the Gram matrix in our algorithms. We also took the so-called class-dependent DPP sampling by using hierarchical sampling to break down a single DPP sampling with large Gram-matrix into many DPP sampling of a much smaller Gram matrix. In the lower hierarchy, each sampling is to be performed on data within its own class. To further improve the computational efficiency, we also used Markov k-DPP to encourage diversity across iterations. Accordingly, we propose five separate mini-batch selection algorithms, which are explained in Section 5.1.2. We also show their empirical effectiveness when these mini-batch selection schemes are applied to classification problems on the Oxford 102 Flower [76], Stanford Dogs [49] and Caltech 101 dataset [25].

5.1 Efficient Diversified Mini-Batch Selection Using Variable High-Layer Features*

DNNs, such as CNN can be thought of as a process of data projection to serve the functions of its final layer (e.g., a softmax). The learning requires back-propagation, which can be highly computational when the data size is large. For this reason, a mini-batch is used instead at every iteration to approximate a “batch method”:

$$\theta_{t+1} = \theta_t - \eta \mathbb{E}[\nabla f(x, \theta)], \quad (5.1)$$

by using:

*This work was published as [36]

$$\{x^{(i)} \sim p(x)\} \quad \theta_{t+1} = \theta_t - \eta \frac{1}{M_{mb}} \sum_{i=1}^{M_{mb}} \nabla f(x^i, \theta), \quad (5.2)$$

where M_{mb} is the mini-batch size. In most instances, we do not know the underlying distribution $p(X)$. Therefore, a uniform sampling of data indices has been used instead. In our work, we use DPP to select a diversified set of data points within each category for each SGD iteration.

Although it may be possible to allow the size of mini-batches to vary in each iteration, it is practical to keep them the consistent across iterations. Therefore, in this work, we used a k-DPP, where each DPP draw generates a subset having equal cardinality k .

5.1.1 Gram-Matrix Construction from Variable Higher-Layer Features

DPP sampling requires a Gram-matrix which defines similarities between data points. The past literature offers two forms of features to construct a Gram matrix, namely the *raw* feature and *fixed higher layer* features.

The *raw* feature, it uses the input directly as feature vectors. For example, DM-SGD ([55]) uses the Gram-matrix from raw MNIST image pixels which are first reshaped into one dimension. This method does not require additional feature extractions, and the eigen-decomposition for the matrix only needs to be calculated once without the presence of neural network.

The *fixed higher layer* feature, it is obtained by feeding each data sample to a pre-trained DNN such as VGG-16. Like the raw data, the Gram matrix is not updated during training.

Variable higher-Layer features and the advantages

The first two constructions are computationally effective since the Gram-matrix and eigen-decomposition only need to be computed once. However, using the *raw*

feature, measuring diversities in terms of the Euclidean distance in the input space can be misleading because the purpose of the neural network is to project data from its original space to a layer before output, to better serve the function of the output layer. Therefore, we ought to use data of the last (or higher) layer instead of the input. Since the parameters changes over iterations, using the *fixed higher feature* to construct a fixed Gram-matrix, does not reflect the parameter update and makes it entirely dependent on the initial parameter guess.

Therefore, we used the output from the last FC layer as the feature vector to construct the Gram-matrix. These features are recomputed since the network is updated through back-propagation. Thus, they are named the *variable higher-layer* feature.

The advantage of were demonstrated using the *variable higher-layer* feature, and its remarkable expressive power over using *raw* or *fixed higher-layer* features. Figure 5.1 plots the feature vectors used to construct the three Gram-matrices for the Oxford Flower 102 datasets. In terms of *variable higher-layer* features, we used the value of the last iteration.

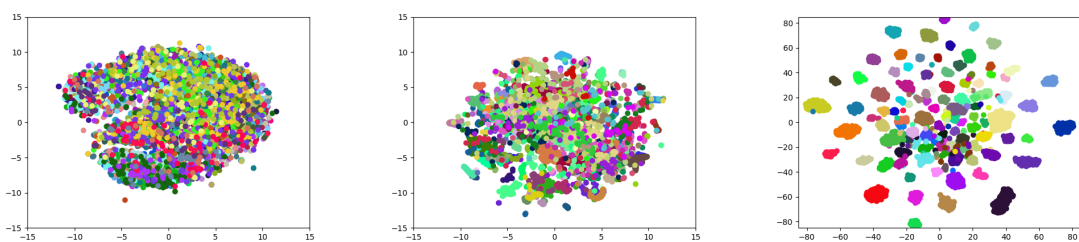


Figure 5.1 : A visualization of three feature vectors being used to construct Gram-matrices. From left to right, each figure represents *raw*, *fixed higher-layer* and *variable higher-layer* features respectively. Each category is represented in a unique colour. Features are dimension reduced using t-SNE.

The figure shows that *raw* or *fixed higher-layer* features provide very high intra-

class variances because data of the same class tend *not* to be close. Conversely, *variable higher-layer* features have much lower intra-class variance. Numerically, we measured the Calinski-Harabasz score for the three feature vectors as in Table 5.1. The Calinski-Harabasz score is defined as a ratio between the within-cluster dispersion and the between-cluster dispersion. A higher score shows dense and well separated clusters, and in a classification problem, the class labels of training data provide the cluster information. Not surprisingly, the *variable higher-layer* feature has much higher Calinski-Harabasz score than the other two.

Table 5.1 : Calinski-Harabasz score for *raw*, *fixed higher-layer* and *variable higher-layer* features in Oxford 102.

	Raw pixels	Fixed Features	Variable Features
Oxford 102	10.3827	26.7641	78.4164

Computationally feasible way to construct variable Gram matrix

Knowing that the method ought to construct the Gram-matrix using *variable higher-layer* features has barely solved any problems because generating random samples from a DPP with a large Gram-matrix can be prohibitively expensive, since the operation involves eigen-decomposition. For these reasons, we proposed five different mini-batch approximation schemes below, which is also the key contribution of our paper.

In each of these methods, the Gram matrix sizes are drastically reduced, allowing the algorithm to significantly accelerate training. Remarkably, while the Gram-matrix construction is updated at each iteration, the experiment shows that our method achieves much quicker convergence compared with previous approaches. In this paper, we provide two broad categories of approaches, namely, the full-set DPP

sampling and class-dependent DPP sampling.

Full-Set DPP Sampling

In this category, one does not consider the class label information when sampling DPP. Subsets are selected independent of their labels. There are two versions of algorithms, which we named FULL-SINGLE-DPP and FULL-MARKOV-DPP respectively:

FULL-SINGLE-DPP In here, in each i^{th} iteration, one first samples a larger subset \tilde{S} uniformly from the full training set Ω , before sampling a random subset S_i using k-DPP from \tilde{S} . Compared with DPP sampling from Ω directly, this method has drastically accelerated its computation. The choice of $|\tilde{S}|$ is arbitrary if it is larger than K : we have chosen it to be a multiple of K . Obviously, a larger $|\tilde{S}|$ results in a longer duration, but simultaneously, it also increases its performance in terms of a faster convergence and accuracy, and the Calinski-Harabasz score. Therefore, one may select an optimised and appropriate trade-off value given the size of the dataset. This approach is summarised in Algorithm 6.

In addition, in order to collect the feature matrix W_i , a feed-forward process is required for \tilde{S} . Thus, Gram-matrix for the k-DPP sampling is calculated as $W_i W_i^\top$.

Algorithm 6 FULL-SINGLE-DPP sampling

Require: Data Ω , mini-batch size K

- 1: **for** $i = 1$ to $MaxIter$ **do**
 - 2: $\tilde{S} \sim \text{Uniform}(\Omega)$
 - 3: Obtain the feature matrix W_i in $\text{feed-forward}(\tilde{S})$
 - 4: $S_i \sim \text{k-DPP}(W_i W_i^\top)$
 - 5: Perform updates according to equation 5.2
 - 6: **end for**
-

FULL-MARKOV-DPP FULL-SINGLE-DPP only provides samples with diversities within a single iteration. However, diversities are also needed between consecutive iterations. For this reason, we also proposed the cross-iteration diversity sampling scheme which we named FULL-MARKOV-DPP. In here, it uses conditional Markov k-DPP to perform sampling on S_i conditioning on all previous samples $\{S_{t<i}\}$. Therefore, in addition to encouraging samples in each mini batch to be as diverse as possible, it also encourages samples to be more diverse across the iterations. Details are outlined in Algorithm 7.

Algorithm 7 FULL-MARKOV-DPP sampling

Require: Data Ω , mini-batch size K

```

for  $i = 1$  to  $MaxIter$  do
2:    $\tilde{S} \sim \text{Uniform}(\Omega)$ 
      Obtain the feature matrix  $W_i$  in  $\text{feed-forward}(\tilde{S})$ 
4:   if  $iter = 1$  then
       $S_i \sim \text{k-DPP}(W_i W_i^\top)$ 
6:   else
       $S_i \sim \text{markov-kDPP}(W_i W_i^\top, \{S_{t<i}\})$ 
8:   end if
      Perform updates according to Equation 5.2
10: end for

```

The original Markov k-DPP works on a fixed Gram-matrix across time steps. However, as sampling is applied on a set of randomly selected data points, the Gram matrix also needs to be recalculated. From the second iteration, the Gram matrix is calculated from the uniform random subset \tilde{S}_i and conditioning factors $\{S_{t<i}\}$ that are mini batch samples from the previous iterations.

Class-Dependent Stochastic DPP

The second broad category of our proposed methods is class-dependent stochastic DPP. Given categories $1, \dots, C$, it takes a hierarchical approach to sampling: in the *first hierarchy*. A sampling is first performed within each class to obtain d initial data points. In the *second hierarchy*, a second k-DPP is applied to sample the final mini-batch with size K . This hierarchical approach of DPP sampling breaks down a single DPP sample with large Gram matrix into many DPP sampling of much smaller Gram-matrix, thereby enhancing its performance dramatically.

Three methods are proposed in this category as shown in Algorithm 8, differing only in terms of the *first hierarchy*; **CLASS-SINGLE-DPP** and **CLASS-MARKOV-DPP** applies k-DPP and Markov k-DPP respectively, as in the full-set DPP sampling case in Section 5.1.1.

CLASS-IMBALANCE: This handles scenarios in which there are class-imbalance problems. Some classes may have significantly more data points than other sets, therefore, DPP sampling is only applied to “smaller” class data, and uniform sampling is applied to the rest. Here, Markov k-DPP is only applied to classes with numbers of samples n_c below a manually set threshold δ . The class-dependent Stochastic DPP is summarised in Algorithm 8.

where

$$\text{SELECT}(W^c, \Omega^c) = \begin{cases} \text{kDPP}(W^c W_c^\top) & \mathbf{CLASS-SINGLE-DPP} \\ \text{markov-kDPP}(W^c W_c^\top, \{S_{t < i}\}) & \mathbf{CLASS-MARKOV-DPP} \\ \begin{cases} \text{markov-kDPP}(W^c W_c^\top), & n_c < \delta \\ U(\Omega^c), & n_c \geq \delta \end{cases} & \mathbf{CLASS-IMBALANCE} \end{cases}$$

In conclusion, the differences between the proposed methods relative to DM-SGD

Algorithm 8 Class-dependent stochastic DPP

Require: Data Ω , number of samples in each categories $\{n_1, \dots, n_C\}$

```

for  $i = 1$  to  $MaxIter$  do
  for  $c = 1$  to  $C$  do
3:   obtain  $W_i^c$  in feed-forward( $\Omega^c$ )
       $S_i^c \sim \text{SELECT}(W_i^c, \Omega^c)$ 
  end for
6:   Obtain the feature matrix  $W_i$  by concatenating features of  $[S_i^1, \dots, S_i^C]$ 
      sample  $S_i \sim \text{k-DPP}(W_i W_i^T)$ 
      Perform updates according to Equation 5.2
9: end for

```

lie in the following perspectives.

- i. The feature vectors W used to compute the Gram matrix are not coming from *raw* or *fixed higher-layer* features; instead, they are *variable higher-layer* features, which are updated as parameters change in each iteration.
- ii. Unlike DM-SGD, as in Equation 2.32, the label information is not used in defining the feature vector for each data sample.
- iii. Various techniques are employed to avoid DPP sampling on the full training set which significantly saves the time cost of performing sampling.
- iv. Markov k-DPP is employed to encourage diversity across iterations as well.

In terms of the variance reduction, [108] proved that DM-SGD, which employs k-DPP sampling, has a lower gradient variance than does vanilla SGD. The same proof can be easily applied to our proposed method. The proposed methods also

employ distance-dependent similarity kernel and sample diverse points, thus, they naturally inherit the properties of k-DPP.

5.1.2 Computation Complexity Analysis

This section provides the sampling duration computation complexity of the proposed methods and compares it with DM-SGD. Assume that the entire dataset Ω contains N data samples and each mini-batch contains K items. In each iteration, DM-SGD requires $O(N^3)$ to perform the eigen-decomposition and $O(NK + K^2)$ to perform the sampling in each iteration. The total time cost after I iterations is:

$$T_{\text{DMSGD}} = O(N^3 + (NK + K^2) \cdot I). \quad (5.3)$$

Full-set DPP performs sampling on a much smaller subset S_i in iteration i . Assume that the size of S_i is T of the original set Ω and $T \in (0, 1]$ (e.g. $T = 1/3$). The proposed methods do not require the eigen-decomposition to be performed on the $N \times N$ similarity kernel prior to the training. The computation cost in each iteration is $O((NT)^3 + NTK + K^2)$. The total computation cost for I iterations is:

$$T_{\text{FULLSET}} = O(((NT)^3 + NTK + K^2) \cdot I). \quad (5.4)$$

If we compare the two values T_{DMSGD} and T_{FULLSET} , we can easily derive that the condition for $T_{\text{DMSGD}} > T_{\text{FULLSET}}$ is:

$$(1 - T^3 I) \cdot N^2 + (1 - T) \cdot KI > 0. \quad (5.5)$$

As $(1 - T) \cdot KI$ is always positive, if we only focus on the first term, then if $T^3 < 1/I$, we can have $T_{\text{DMSGD}} > T_{\text{FULLSET}}$. For example, when I is 10,000, as long as S_i is smaller than $1/22$ of the entire dataset, FULL-set DPP is more time efficient

than DM-SGD. This is quite an easy constraint to meet. Considering the second term should further loosen this constraint.

Class-dependent DPP is slightly more complex because it performs separate sampling in each category of data. If we simplify the problem setting by assuming that C categories equally split the entire dataset, and the subset S_i^c sampled from category c contains T of total samples in this category, the total computation cost is

$$T_{\text{Class-dependent}} = \left(\left(\left(\frac{N}{C} \right)^3 + \left(\frac{N}{C} \right) \cdot \left(\frac{N}{C} \cdot T \right) + \left(\frac{N}{C} \cdot T \right)^2 \right) \cdot C + (NT)^3 + (NT) \cdot K + K^2 \right) \cdot I. \quad (5.6)$$

Therefore, the required condition for $T_{\text{DMSGD}} > T_{\text{Class-dependent}}$ is:

$$\left(\frac{I}{C^2} + I \cdot T^3 - 1 \right) \cdot N^2 + \left(\frac{T}{C} + \frac{T^2}{C} \right) \cdot NI + (T - 1) \cdot KT < 0. \quad (5.7)$$

If we take the Stanford Dogs dataset, whose statistics are shown in Table 5.2, as an example and we set $K = 50$, $T < 1/32$ is required such that class-dependent DPP is more computationally efficient than is DM-SGD.

In Figure 5.2, we plot the training duration, which includes the sampling, feed-forward and back-propagation durations, over different T when the total iterations I are 10,000. All experiments are performed on the Oxford 102 dataset using the full VGG-16 network. DMSGD2 is the baseline model, the details and explanation of which can be found in Section 5.1.3. It is clear to see that the computation cost of the proposed algorithms is lower when T is larger. When $T < 1/10$, all four proposed methods are more efficient than the baseline model.

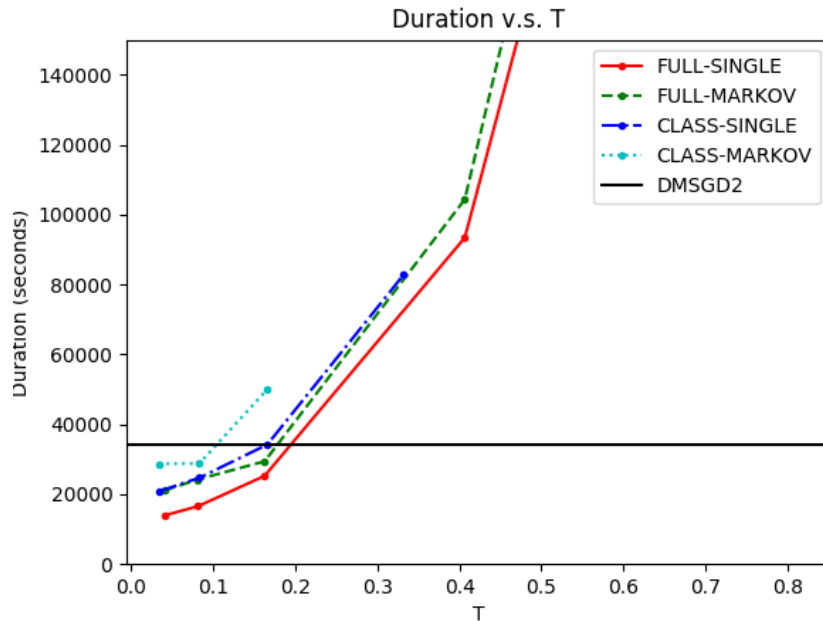


Figure 5.2 : Duration over T on the Oxford 102 Flower dataset.

5.1.3 Experiments

Datasets

Experiments were performed on several image classification problems using the Oxford 102 Flower [76], Stanford Dogs [49], Caltech 101 datasets [25] and MNIST datasets. Statistics of these datasets are as below.

Table 5.2 : Statistics of the Oxford 102 Flower, the Stanford Dogs, the Caltech 101 datasets, and the MNIST datasets.

Dataset	Oxford 102			Stanford Dogs			Caltech 101	MNIST	
#categories	102			120			101	10	
#samples	train	test	val	train	test	val	9,145	train	test
	1,020	6,149	1,020	12,000	8,580	8,580		60,000	10,000

Following the original paper from [108]. The training of the Oxford 102 Flower

dataset was performed on the 6,149 testing images and the testing was performed on the 1,020 training images (i.e. the training and testing sets were interchanged in our experiments). In addition, because the Caltech 101 dataset does not provide the train-test split, we randomly selected 80% of the data from each category as the training set and we equally split the rest of the data in each category as the testing and validation set. Lastly, performing eigen-decomposition on the entire Gram-matrix generated from MNIST image features is extremely time consuming. Therefore, we randomly selected 20% from each category to perform the training and another 20% as the validation set. We used the original test set for the testing. The results were collected from multiple experiments with random selections.

Baseline Models

We compared the proposed methods against two baseline models on the first three datasets. The first baseline model is the DM-SGD from [108], which uses the off-the-shelf last FC features and the label information to construct the Gram-matrix. The design from the original paper only trains the layer before the softmax layer, so we also compared the proposed algorithms with the second baseline model, which we named as “DMSGD2”. DMSGD2 allows all layers of DNNs to be trained while the sampling still relies on the off-the-shelf last FC features. For both DM-SGD and DMSGD2, Equation 2.32 is applied to calculate the feature vector F for each data sample. The proposed methods construct the Gram-matrix using the last FC features as shown in the pseudo-codes.

For the MNIST dataset, we compared the proposed methods against one baseline model DM-SGD. Following [108], DM-SGD uses the raw image pixels and the label information to construct a RBF kernel for the sampling. It also trains a full five-layer CNN network instead of only the last layer, as for the previous datasets. The proposed method still uses the Gram-matrix generated from the last FC features

and trains the same five-layer network.

Performance Evaluation on Oxford 102 Flower

This section evaluates the performances of the proposed methods, by comparing them with the baseline models. Comparisons were made in terms of accuracy, Calinski-Harabasz scores and time costs. Following the original paper, the results are demonstrated via fine tuning the pre-trained VGG-16 model. Experiments were performed with 10,000 iterations, learning rate was fixed as $1e - 5$ and the batch size was set to be $K = 50$. The same setting was applied to all datasets.

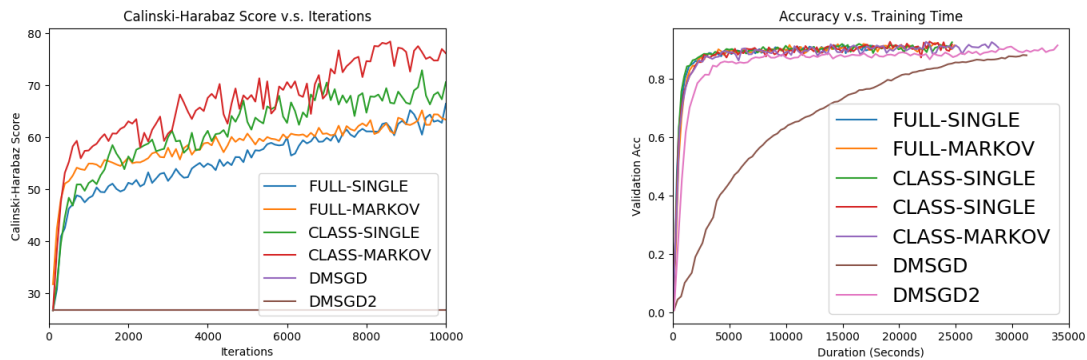


Figure 5.3 : Calinski-Harabasz scores over iterations and the validation accuracy over the training duration on the Oxford 102 Flower dataset.

In terms of the accuracy, Figure 5.3 and Table 5.3 shows that the proposed algorithms were able to achieve faster convergence rates and higher final testing accuracies. The proposed methods achieved up to 3.50% higher accuracy than baseline models.

In addition, compared with baseline models which used fixed Gram-matrices, we were able to achieve a much higher Calinski-Harabasz score during training.

Table 5.3 also reports validation accuracies over time costs and the total training duration. The displayed time costs for the proposed methods include both sampling

Table 5.3 : The final test accuracy, training duration and Calinski-Harabasz score achieved by the proposed sampling methods and the uniform sampling on the Oxford 102 dataset.

Experiment	Accuracy	Duration(seconds)	Calinski-Harabasz score
FULL-SINGLE	0.9010 ± 0.0027	$17,587.096 \pm 77.5$	66.4588 ± 1.6767
FULL-MARKOV	0.9080 ± 0.0065	$24,867.031 \pm 96.0$	68.1966 ± 2.5753
CLASS-SINGLE	0.9157 ± 0.0027	$24,671.139 \pm 106.3$	72.8985 ± 3.5527
CLASS-MARKOV	0.9333 ± 0.0040	$28,760.66 \pm 151.6$	78.4164 ± 2.0534
DM-SGD	0.8852 ± 0.0038	$31,257.426 \pm 82.8$	26.7737
DMSGD2	0.8983 ± 0.0044	$33,994.164 \pm 143.5$	26.7737

and back-propagation duration in each iteration, while omitting the data processing duration.

Results show that all four proposed methods spent less time in sampling than the DM-SGD, even when the eigen-decomposition needed to be performed in each iteration. In particular, FULL-SINGLE spends 56.27% of the time costs of the original DM-SGD and achieves a higher testing accuracy.

Compared with the four proposed methods on the four datasets, class dependent stochastic sampling achieves a better performance than full-set sampling on all four datasets. Methods that employs Markov-kDPP performs better than those using k-DPP.

Performance Evaluation on the Stanford Dogs Dataset

Figure 5.4 and Table 5.4 report the validation accuracy over training iterations and the training duration on the Stanford Dogs dataset. The best performing

CLASS-MARKOV were 5.5% higher in accuracy than DM-SGD and 3.0% higher than DMSGD2. In terms of the training duration, all four proposed methods require less training durations than DM-SGD. It is clear in the figure that the proposed algorithms have a faster learning curve and a clear advantage in computation efficiency.

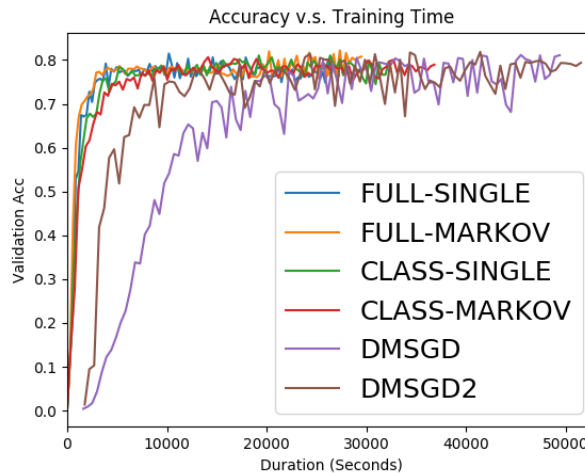


Figure 5.4 : Validation accuracy over the training duration by the proposed sampling methods and the uniform sampling on the Stanford Dogs dataset.

Table 5.4 : The final test accuracy and training duration achieved by the proposed sampling methods and the uniform sampling on the Stanford Dogs dataset.

Experiment	Accuracy	Duration(seconds)
FULL-SINGLE	0.7738 ± 0.0228	$25,268.7 \pm 110.1$
FULL-MARKOV	0.8114 ± 0.0135	$29,497.8 \pm 83.5$
CLASS-SINGLE	0.8130 ± 0.0072	$30,831.7 \pm 84.8$
CLASS-MARKOV	0.8165 ± 0.0032	$36,814.6 \pm 78.5$
DMSGD	0.7612 ± 0.0115	$47,817.7 \pm 118.0$
DMSGD2	0.7867 ± 0.0014	$49,763.4 \pm 79.1$

Performance Evaluation on the Caltech 101 Dataset

Figure 5.5 and Table 5.5 report the validation F_1 score over training iterations and the training duration on the Caltech 101 dataset. Since the dataset is highly imbalanced, the results are measured by F_1 score rather than accuracy. The best performing CLASS-SINGLE reported a 7.2% higher F_1 than DM-SGD and 2.4% higher than DMSGD2. In terms of the training duration, FULL-SINGLE requires approximately of the 49.8% time cost of DM-SGD, and all proposed methods spent less time than the two baseline models. It is clear to see that the proposed methods achieve a better performance on this highly imbalanced dataset.

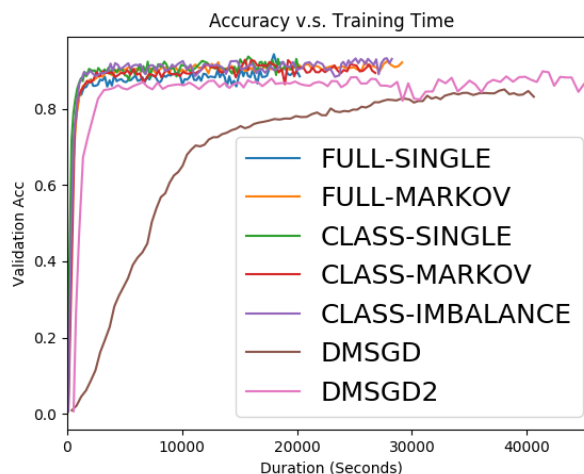


Figure 5.5 : Validation accuracy over the training duration by the proposed sampling methods and the uniform sampling on the Caltech 101 dataset.

In addition, it can be noted that class-dependent stochastic sampling methods performed better than the full-set sampling methods on this dataset, while CLASS-MARKOV performed slightly worse than CLASS-SINGLE in terms of the testing F_1 . This is because images in the Caltech 101 dataset have high intra-class variance. While class-dependent sampling performs separate sampling across categories, further encouraging diversity across iterations does not necessarily improve the per-

Table 5.5 : Final test accuracy and training duration achieved by the proposed sampling methods and the uniform sampling on the Caltech 101 dataset.

Experiment	F_1	Duration(seconds)
FULL-SINGLE	0.9165 ± 0.0317	$20,231.5 \pm 96.6$
FULL-MARKOV	0.9182 ± 0.0116	$20,464.8 \pm 108.0$
CLASS-SINGLE	0.9232 ± 0.0093	$26,548.0 \pm 82.1$
CLASS-MARKOV	0.9221 ± 0.0023	$26,809.6 \pm 143.3$
CLASS-IMBALANCE	0.9097 ± 0.0359	$26,250.9 \pm 65.6$
DM-SGD	0.8515 ± 0.0102	$40,604.2 \pm 84.5$
DMSGD2	0.8990 ± 0.0092	$45,491.3 \pm 167.0$

formance as with other datasets.

Performance Evaluation on MNIST

Figure 5.6 and Table 5.6 report the performance on the MNIST dataset. The proposed method still outperforms DM-SGD in terms of the final testing accuracy. As for the computational cost, class-dependent DPP actually requires a longer duration. This finding is also supported by our analysis in Section 5.1.2 because MNIST only has 10 categories. On the contrary, the proposed full-set DPP methods only require less than 30% of the total training duration of DM-SGD.

5.2 Diversified Mini-Batch Selection in Text-to-Image Generation

The proposed DPP based mini-batch sampling is further applied to the text-to-image generation. This section describes how it is applied and compares the performance of generation under uniform and DPP sampling.

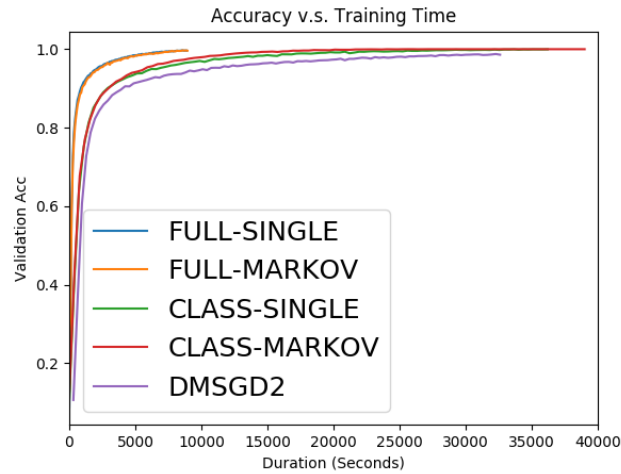


Figure 5.6 : Validation accuracy over training duration by the proposed sampling methods and the uniform sampling on the MNIST dataset.

Table 5.6 : Final test accuracy and training duration achieved by the proposed sampling methods and the uniform sampling on the MNIST dataset.

Experiment	Accuracy	Duration (seconds)
FULL-SINGLE	0.9704 ± 0.0025	$6,841.92 \pm 163.57$
FULL-MARKOV	0.9703 ± 0.0011	$11,109.3 \pm 160.67$
CLASS-SINGLE	0.9712 ± 0.0015	$36,204.4 \pm 120.07$
CLASS-MARKOV	0.9714 ± 0.0034	$52,640.0 \pm 247.16$
DM-SGD	0.9609 ± 0.0004	$32,224.8 \pm 202.95$

DPP sampling is applied to text-to-image generation in a similar fashion as described in Section 5.1, in which the mini-batch is sampled using DPP sampling instead of uniform sampling. In particular, we apply FULL-SINGLE DPP sampling in the experiments. Experiments are performed on the CUB dataset [99].

DPP sampling requires the construction of feature matrix which determines the similarity between every two samples. The proposed approaches in Section 5.1

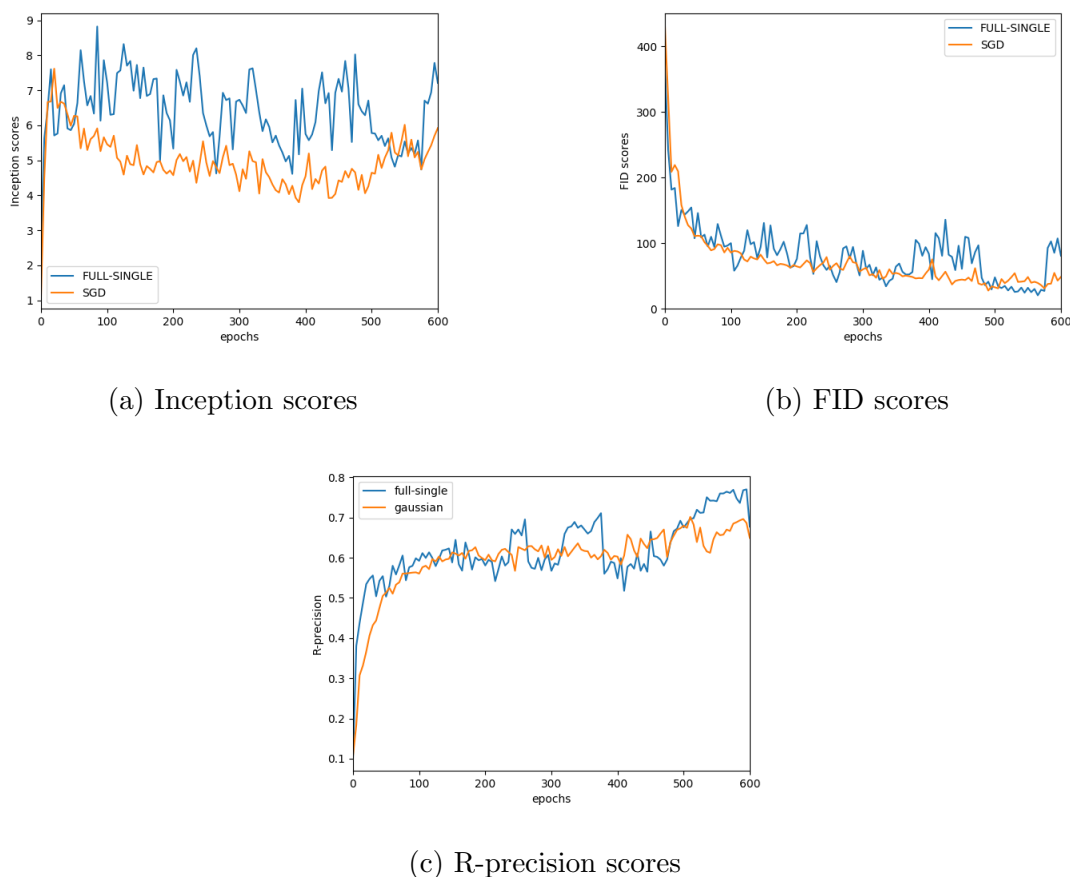


Figure 5.7 : Inception, FID and R-precision over epochs on the CUB dataset. “SGD” denotes the experiment performed using the conventional SG; “FULL-SINGLE” denotes the one performed using the FULL-SINGLE DPP sampling.

uses a combination of image feature and labels. As in this case each sample is a sentence-image pair and each image has a label. Here, the proposed model uses a concatenation of sentence feature, image feature and image label. The training is a two-phase process, in which the text encoder is first pre-trained with the true image-sentence pairs and the GAN network is trained next. The DPP sampling is only applied to the GAN training.

Figure 5.7 shows the inception, FID and R-precision scores over training epochs. Training using FULL-SINGLE DPP sampling constantly achieves a higher IS com-

pared with the one using the conventional SGD. Using FULL-SINGLE-DPP also results in a lower FID score and R-precision score, as in Figure 5.7. The result shows that applying DPP sampling to text-to-image generation also enhances the performance.

5.3 Summary

SGD has been widely adopted in training DNNs of various structures. Instead of using a full dataset, a so-called *mini-batch* is selected during each gradient descent iteration. This aims to accelerate the learning when a large amount of training data is present. Without the knowledge of its true underlying distribution, one often samples the data indices uniformly. Recently, researchers applied a diversified mini-batch selection scheme through the use of Determinantal Point Process (DPP), in order to avoid having highly correlated samples in one batch [108]. Despite its success, the attempts were restrictive in the sense that they used fixed features to construct the Gram matrix for DPP; using the raw or fixed higher-layer features limited the amount of potential improvement over the convergence rate. In this paper, we instead proposed using variable higher-layer features that are updated at each iteration when the parameter changes. To avoid the high computation cost, several contributions have been made to accelerate the computation of DPP sampling, including: (1) using hierarchical sampling to break down a single DPP sampling with large Gram-matrix into many DPP samplings of much smaller Gram matrix and (2) using Markov k-DPP to encourage diversity across iterations. Empirical results show a much more diversified mini batch in each iteration in addition to a much improved convergence compared with the previous approach.

Chapter 6

Conclusion

Our work aimed to identify effective approaches to improve the current cross domain translation algorithms. Based on studies on several applications of cross domain transfer, latent space modelling of GANs, and accelerating the training by improving mini-batch SGD, we introduced several methodologies that outperform the previous baseline models both in terms of the generation performance and the time cost.

The first part of our thesis was dedicated to cross domain transfer applications, in particular, text to image generation and transfer over multiple image domains. My work proposed a new design of text embedding that extracts additional phrase embedding and a new set of attentions computed between object-grid regions and phrases. The proposed method brings them into the GAN network design and can generate more realistic and accurate images on the MSCOCO dataset, as reflected by the inception scores, FID scores, and R-precision. We also proposed a methodology that automatically determines the optimal order of image domains, while studying the transfer functions between consecutive image domains. In this method, although translation between two domains might require up to $N-1$ steps, only two generators are required to perform translation over N image domains. Experimental results show that the proposed methodology is capable of deciding the ordering over image domains while generating plausible images.

The limitations of our work are that first, the proposed phrase-region attention structure is only applicable to text-to-image generations. In the future we hope to extend such a mechanism to other pairs of domains with more complex structures

(e.g., between sounds and videos or paragraphs and videos). These domains also consist of sequential data, so a similar attention mechanism is expected to be built. Secondly, the proposed multiple domain transfer algorithms work only on datasets that present a consistent change of details across the consecutive domains, as the universal transfer function is applied. Several alternatives that might enable the proposed method to have stronger generation variety, if the universal transfer function is built between hidden layers, and each image domain has its own output function. Therefore, the universal transfer function learns the latent correlation between domains, and the separate output functions can fine-tune the details of each domain. The second plan is that instead of using only one single pair of transfer functions, the model learns m transfer functions across N image domains and $m \ll N$. This way, although the network is slightly larger, the generation variety would be much higher. The third plan is to use a normalising flow model instead of a generator in the transfer function.

The second contribution we made in this thesis is the latent space modelling of GANs. Our first work proposed a novel framework to better capture the latent structure of a complex dataset. Under this framework, the GAN generator’s simple distribution was replaced by GMM. Subsequently, many technical innovations were proposed to make this framework trainable in an “end-to-end” fashion. Most noticeably, a PCM was innovated to help the model to better approximate GMM’s responsibility distribution when given the data. In addition to GMM modelling, we also demonstrated the multitude of benefits in the proposed approach:

- (1) We demonstrated through experiments that our proposed method retains GAN’s ability in sharper image generation compared with other GMM GAN methods.
- (2) The proposed approach can significantly save the computation cost as only

half the number of parameters is required to achieve the same image generation quality compared to classic DCGAN.

(3) Thanks to the mixed densities, the proposed method surpasses previous baselines when dealing with a highly imbalanced dataset.

The performance of applying Gaussian Mixture to GANs was also demonstrated on the CUB dataset for text-to-image generation. Compared to the vanilla GAN, Gaussian Mixture GAN delivers a higher image generation quality.

We also provided an empirical evaluation of the performance of modelling the GAN latent space with GMM. Experiments are performed both in terms of the generation quality and anomaly detection on multiple datasets. Results show that GMM-based GAN can significantly reduce the required number of parameters, while maintaining the same quality of generation. Such a property can bring about many new image generation functionalities, such as generation on portable devices with limited computation power. In addition, we proposed to add a classifier in the network to encourage $p(k|\hat{\mathbf{x}}, \theta)$ to be closer to $p(k|z, \theta)$ where $\hat{\mathbf{x}}$ is a synthetic sample and \mathbf{z} is the random vector the sample is generated from. The experimental result shows that this can further improve the generation performance with a minimum increase in the number of parameters.

The GMM expects the number of total components to be predetermined, so we also proposed a novel GAN-based algorithm DP-GAN that models the latent space of GANs using the DPGMM distribution. This was motivated by the difficulty of modelling large complex datasets with DP mixture models directly, and previous GANs do not perform well on unlabelled datasets that contain highly-diversified data and the number of distinct classes K is not feasible to be determined before training. Applying DP allows K to be inferred while the parameters of the distribution are learnt during training. However, because a vanilla GAN does not provide a

backward transformation to study the parameters of DP from the data distribution, we proposed learning the distribution in the following way. We “propose” a set of particles z from a proposal distribution $p(z)$, and z is weighed by the normalized $D(G(z))$. In addition, we encourage $p(c|z)$ and $p(c|x)$ to resemble with each other for any pair of (x, z) in the training, so that the learned G has a consistent posterior responsibility regardless of using x or z . In our empirical study performed on both synthetic and real-world datasets, we demonstrated that the proposed DP-GAN can accurately model the latent space while producing high-quality images. Given the weights of all components, a threshold can be easily set to infer the number of distinct classes after the model is fully trained. Each valid components in the mixture reflects a particular image class. This will allow generation from a specified image class by sampling from a specific Gaussian.

There are several extensions we hope to achieve in the latent space modelling of GANs. Firstly, the initial parameters of the Gaussian Mixture and DP Gaussian mixture distributions are randomly initialised in the current model. Some unsupervised algorithms can be applied to decide the initial parameters to accelerate the convergence. We also plan to apply other distributions to the cross domain transfer learning which enables more flexibility in the generation and useful properties can be discovered, such as scale mixtures.

The third contribution is in terms of accelerating network convergence by incorporating the DPP to mini-batch SGD. We proposed several fast, approximated DPP sampling strategies by taking advantage of available class-label information, which, in turn, allowed us to sample DPP using a Gram matrix that is constructed from the variable higher layer features, updated at each iteration as parameters change. We detailed the five approaches employed in this paper and the rationale behind how these methods significantly reduce the time costs. We demonstrated that all proposed algorithms were able to achieve a faster convergence rate and greater ac-

curacy compared with previous state-of-the-art “fixed” feature approaches. We also applied such DPP accelerated mini-batch sampling to perform text-to-image generation. Experimental results suggest an enhanced generation quality and an earlier convergence. Such a DPP based mini-batch sampling method is also applied to the text-to-image generation on the CUB dataset. The result showed a better image quality with a higher IS and lower FID score. The synthetic samples also reflected the description more accurately than the vanilla GAN because a higher R-precision was achieved.

The current experiment was only performed on the CUB dataset using the FULL-SINGLE-DPP sampling. Although, as demonstrated by the experimental results in Section 5.1.2, Markov-DPP based sampling methods and class-dependent stochastic DPP sampling methods are expected to achieve a faster convergence compared with FULL-SINGLE-DPP sampling. This will be verified in future work. We will also apply DPP based sampling to larger datasets, such as MSCOCO, and study the effects of weights between the image features, sentence features, and image labels. In addition, we plan to extend the proposed mini-batch sampling methods to other cross domain applications, such as text to sound and text to video.

Chapter 7

Appendix

7.1 Network Details for the Text-to-Image Generation

This section lists the hyperparameters chosen for the proposed framework, in addition to the detailed network design (displayed in tables).

7.1.1 Hyperparameters

The hyperparameters chosen for our proposed network are:

- i. $l_S = 15$: maximum number of words in a sentence
- ii. $l_P = 5$: maximum number of phrases in a sentence
- iii. $l_W = 5$: maximum number of words in a phrase
- iv. $l_O = 7$: maximum of objects (bounding boxes) in an image
- v. $D_h = 48$: depth of hidden states h_0 , h_1 and h_2
- vi. $D_e = 256$: dimension of sentence, phrase and word embedding
- vii. $D_d = 96$: a chosen hyper-parameter used in the discriminator
- viii. $D_z = 100$: dimension of the noise vector z and F^{ca}

7.1.2 Basic Network Blocks

Some network blocks used in our framework are:

- i. Upsampling block (kernel = a , stride = b): a nearest neighbour upsampling layer that up-scales the spatial size by 2, a convolution layer with kernel size a and stride size b , a batch normalisation and a GLU layer
- ii. Down-sampling block (kernel= a , stride= b): a convolution layer with kernel size a and stride size b , a batch normalization layer and a leaky ReLU layer
- iii. Spatial replicate: copy the input along an axis

In the below sections, I report the detailed network architecture for the proposed framework, including the operation in each layer, the input and output tensors.

7.1.3 Network Architecture for the Generator

Table 7.1 : Network architecture for the generator

Stage	Sub-stage	Name	Input Tensors	Output Tensors
G_0	Local Path	Spatial Replicate	$1 \times D_e$	$16 \times 16 \times D_e$
		Apply bounding box mask	$16 \times 16 \times D_e$	$l_O \times 16 \times 16 \times D_e$
		Average	$l_O \times 16 \times 16 \times D_e$	$16 \times 16 \times D_e$
		Down-sampling (kernel=4, stride=2) $\times 4$	$16 \times 16 \times D_e$	$1 \times 1 \times (D_h \times 8)$
		Concatenate with noise vector z	$1 \times 1 \times (D_h \times 8), z$	$1 \times 1 \times (D_h \times 8 + D_z)$
		Linear + batch norm + GLU	$1 \times 1 \times (D_h \times 8 + D_z)$	$4 \times 4 \times (D_h \times 16)$
		Up-sampling (kernel=3, stride=1) $\times 2$	$4 \times 4 \times (D_h \times 16)$	$16 \times 16 \times (D_h \times 4)$
		Apply bounding box mask	$16 \times 16 \times (D_h \times 4)$	$l_O \times 16 \times 16 \times (D_h \times 4)$
		Average	$l_O \times 16 \times 16 \times (D_h \times 4)$	$16 \times 16 \times (D_h \times 4)$
	Global Path	Concatenation	z, F^{ca}	$1 \times (D_h \times 2)$
		Linear + batch norm + GLU	$1 \times (D_h \times 2)$	$4 \times 4 \times (D_h \times 16)$
		Up-sampling (kernel=3, stride=1) $\times 2$	$4 \times 4 \times (D_h \times 16)$	$16 \times 16 \times (D_h \times 4)$
		Concatenate local and global outputs		$16 \times 16 \times (D_h \times 8)$
		Upsampling (kernel=3, stride=1) $\times 2$	$16 \times 16 \times (D_h \times 8)$	$64 \times 64 \times D_h$
	G_1	Regular-grid-Word Attention	Linear	e
F_n^{attn1}			$h_0, l_s \times D_h$	$64 \times 64 \times D_h$
Object-grid-Phrase Attention		Linear	p	$l_p \times D_h$
		F_n^{attn2}	$h_0, l_p \times D_h$	$64 \times 64 \times D_h$
		Concatenation	$F_n^{attn1}, F_n^{attn2}, h_0$	$64 \times 64 \times (3 \times D_h)$
Up-sampling (kernel=3, stride=1)	$64 \times 64 \times (3 \times D_h)$	$128 \times 128 \times D_h$		
G_2	Regular-grid-Word Attention	Linear	e	$l_s \times D_h$
		F_n^{attn1}	$h_1, l_s \times D_h$	$128 \times 128 \times D_h$
	Object-grid-Phrase Attention	Linear	p	$l_p \times D_h$
		F_n^{attn2}	$h_1, l_p \times D_h$	$128 \times 128 \times D_h$
		Concatenation	$F_n^{attn1}, F_n^{attn2}, h_0$	$128 \times 128 \times (3 \times D_h)$
Up-sampling (kernel=3, stride=1)	$128 \times 128 \times (3 \times D_h)$	$256 \times 256 \times D_h$		

7.1.4 Network Architecture for the Discriminators

D_0

Table 7.2 : Network architecture for D_0

Stage	Sub-stage	Name	Input Tensors	Output Tensors
Image + Sentence Discriminator		Convolution + leaky ReLU	$64 \times 64 \times 3$	$32 \times 32 \times D_d$
		Down-sampling (kernel=4, stride=2) $\times 3$	$32 \times 32 \times 96$	$f_D^{IMG}(4 \times 4 \times (D_d \times 8))$
	$\mathcal{D}(x)$	Convolution (Image only logits)	$4 \times 4 \times (D_d \times 8)$	1
	$\mathcal{D}(x, \bar{e})$	\textbf{Sentence conditioned logits}	f_D^{IMG}, \bar{e}	1
		Convolution	$64 \times 64 \times 3$	$32 \times 32 \times D_d$
Image + Sentence + Bounding Box Discriminator		Down-sampling (kernel=4, stride=2)	$32 \times 32 \times D_d$	$f_D^{IMG^2}(16 \times 16 \times (D_d \times 2))$
		Spatial Replicate	\bar{e}	$16 \times 16 \times D_e$
		Concatenation	$16 \times 16 \times D_e, f_D^{IMG^2}$	$16 \times 16 \times (D_e + D_d \times 2)$
		Apply bounding box mask	$16 \times 16 \times (D_e + D_d \times 2)$	$16 \times 16 \times (D_e + D_d \times 2)$
		Down-sampling (kernel=4, stride=2) $\times 2$	$16 \times 16 \times (D_e + D_d \times 2)$	$f_D^{IMG-BBOX}(4 \times 4 \times (D_d \times 8))$
Sentence conditioned logits	$\mathcal{D}(x, \bar{e}, b)$	\textbf{Sentence conditioned logits}	$f_D^{IMG-BBOX}, \bar{e}$	1
		Spatial replicate	\bar{e}	$4 \times 4 \times D_e$
		Concatenation	f_D^{IMG} or $f_D^{IMG-BBOX}, \bar{e}$	$4 \times 4 \times (D_e + D_d \times 8)$
		Down-sampling (kernel=3, stride=1)	$4 \times 4 \times (D_e + D_d \times 8)$	$4 \times 4 \times (D_d \times 8)$
		Convolution	$4 \times 4 \times (D_d \times 8)$	1

D_1

Table 7.3 : Network architecture for D_1

Stage	Sub-stage	Name	Input Tensors	Output Tensors
Image + Sentence Discriminator		Convolution + ReLU	$128 \times 128 \times 3$	$64 \times 64 \times D_d$
		Down-sampling (kernel=4, stride=2) $\times 4$	$64 \times 64 \times 96$	$f_D^{IMG}(4 \times 4 \times (D_d \times 16))$
		Down-sampling (kernel=3, stride=1)	$f_D^{IMG}(4 \times 4 \times (D_d \times 16))$	$f_D^{IMG}(4 \times 4 \times (D_d \times 8))$
	$\mathcal{D}(x)$	Convolution (Image only logits)	$4 \times 4 \times (D_d \times 8)$	1
	$\mathcal{D}(x, \bar{e})$	\textbf{Sentence conditioned logits}	f_D^{IMG}, \bar{e}	1
Image + Sentence + Bounding Box Discriminator		Convolution	$128 \times 128 \times 3$	$64 \times 64 \times D_d$
		Down-sampling (kernel=4, stride=2) $\times 2$	$64 \times 64 \times D_d$	$f_D^{IMG^2}(16 \times 16 \times (D_d \times 4))$
		Spatial Replicate	\bar{e}	$16 \times 16 \times D_e$
		Concatenation	$16 \times 16 \times D_e, f_D^{IMG^2}$	$16 \times 16 \times (D_e + D_d \times 4)$
		Apply bounding box mask	$16 \times 16 \times (D_e + D_d \times 2)$	$l_0 \times 16 \times 16 \times (D_e + D_d \times 4)$
Sentence conditioned logits		Average	$l_0 \times 16 \times 16 \times (D_e + D_d \times 4)$	$16 \times 16 \times (D_e + D_d \times 4)$
		Down-sampling (kernel=4, stride=2) $\times 2$	$16 \times 16 \times (D_e + D_d \times 2)$	$4 \times 4 \times (D_d \times 16)$
		Down-sampling (kernel=3, stride=1)	$f_D^{IMG-BBOX}(4 \times 4 \times (D_d \times 16))$	$f_D^{IMG-BBOX}(4 \times 4 \times (D_d \times 8))$
	$\mathcal{D}(x, \bar{e}, b)$	\textbf{Sentence conditioned logits}	$f_D^{IMG-BBOX}, \bar{e}$	1
		Spatial replicate	\bar{e}	$4 \times 4 \times D_e$
Sentence conditioned logits		Concatenation	f_D^{IMG} or $f_D^{IMG-BBOX}, \bar{e}$	$4 \times 4 \times (D_e + D_d \times 8)$
		Down-sampling (kernel=3, stride=1)	$4 \times 4 \times (D_e + D_d \times 8)$	$4 \times 4 \times (D_d \times 8)$
		Convolution	$4 \times 4 \times (D_d \times 8)$	1

D_2 Table 7.4 : Network architecture for D_2

Stage	Sub-stage	Name	Input Tensors	Output Tensors
Image + Sentence Discriminator		Convolution + leaky ReLU	$256 \times 256 \times 3$	$128 \times 128 \times D_d$
		Down-sampling $\times 3$ (kernel=4, stride=2)	$128 \times 128 \times D_d$	$f_D^{IMG}(16 \times 16 \times (D_d \times 8))$
		Down-sampling (kernel=3, stride=1) $\times 2$	$f_D^{IMG}(4 \times 4 \times (D_d \times 16))$	$f_D^{IMG}(4 \times 4 \times (D_d \times 8))$
	$\mathcal{D}(x)$	Convolution (Image only logits)	$4 \times 4 \times (D_d \times 8)$	1
	$\mathcal{D}(x, \bar{e})$	Sentence conditioned logits	f_D^{IMG}, \bar{e}	1
Image + Sentence + Bounding Box Discriminator		Convolution + leaky ReLU	$256 \times 256 \times 3$	$128 \times 128 \times D_d$
		Down-sampling $\times 3$ (kernel=4, stride=2)	$128 \times 128 \times D_d$	$f_D^{IMG^2}(16 \times 16 \times (D_d \times 8))$
		Spatial Replicate	\bar{e}	$16 \times 16 \times D_e$
		Concatenation	$16 \times 16 \times D_e, f_D^{IMG^2}$	$16 \times 16 \times (D_e + D_d \times 8)$
		Apply bounding box mask	$16 \times 16 \times (D_e + D_d \times 8)$	$l_O \times 16 \times 16 \times (D_e + D_d \times 8)$
		Average	$l_O \times 16 \times 16 \times (D_e + D_d \times 8)$	$16 \times 16 \times (D_e + D_d \times 8)$
		Down-sampling $\times 2$ (kernel=4, stride=2) $\times 2$	$16 \times 16 \times (D_e + D_d \times 2)$	$4 \times 4 \times (D_d \times 32)$
		Down-sampling (kernel=3, stride=1) $\times 2$	$f_D^{IMG-BBOX}(4 \times 4 \times (D_d \times 32))$	$f_D^{IMG-BBOX}(4 \times 4 \times (D_d \times 8))$
$\mathcal{D}(x, \bar{e}, b)$	Sentence conditioned logits	$f_D^{IMG-BBOX}, \bar{e}$	1	
Sentence conditioned logits		Spatial replicate	\bar{e}	$4 \times 4 \times D_e$
		Concatenation	f_D^{IMG} or $f_D^{IMG_BBOX}, \bar{e}$	$4 \times 4 \times (D_e + D_d \times 8)$
		Down-sampling (kernel=3, stride=1)	$4 \times 4 \times (D_e + D_d \times 8)$	$4 \times 4 \times (D_d \times 8)$
		Convolution	$4 \times 4 \times (D_d \times 8)$	1

Bibliography

- [1] R. H. Affandi, A. Kulesza, and E. B. Fox, “Markov determinantal point processes,” *CoRR*, vol. abs/1210.4850, 2012. [Online]. Available: <http://arxiv.org/abs/1210.4850>
- [2] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio, “Variance Reduction in SGD by Distributed Importance Sampling,” *ArXiv e-prints*, 2015.
- [3] Y. Alami Mejjati, C. Richardt, J. Tompkin, D. Cosker, and K. I. Kim, “Unsupervised attention-guided image-to-image translation,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 3693–3703. [Online]. Available: <http://papers.nips.cc/paper/7627-unsupervised-attention-guided-image-to-image-translation.pdf>
- [4] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.
- [5] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *CoRR*, vol. abs/1409.0473, 2014. [Online]. Available: <http://arxiv.org/abs/1409.0473>
- [6] S. Barratt and R. Sharma, “A note on the inception score,” *arXiv preprint arXiv:1801.01973*, 2018.
- [7] M. Ben-Yosef and D. Weinshall, “Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised cluster-

- ing of images,” *CoRR*, vol. abs/1808.10356, 2018. [Online]. Available: <http://arxiv.org/abs/1808.10356>
- [8] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, “Scheduled sampling for sequence prediction with recurrent neural networks,” *CoRR*, vol. abs/1506.03099, 2015. [Online]. Available: <http://arxiv.org/abs/1506.03099>
- [9] D. M. Blei and M. I. Jordan, “Variational inference for dirichlet process mixtures,” *Bayesian Anal.*, vol. 1, no. 1, pp. 121–143, 03 2006. [Online]. Available: <https://doi.org/10.1214/06-BA104>
- [10] P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam, “Optimizing the latent space of generative networks,” *arXiv preprint arXiv:1707.05776*, 2017.
- [11] A. Brock, J. Donahue, and K. Simonyan, “Large scale GAN training for high fidelity natural image synthesis,” *CoRR*, vol. abs/1809.11096, 2018. [Online]. Available: <http://arxiv.org/abs/1809.11096>
- [12] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. Lafferty, R. L. Mercer, and P. S. Roossin, “A statistical approach to machine translation,” *Computational linguistics*, vol. 16, no. 2, pp. 79–85, 1990.
- [13] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, “Sample size selection in optimization methods for machine learning,” *Mathematical programming*, vol. 134, no. 1, pp. 127–155, 2012.
- [14] H. Chang, J. Lu, F. Yu, and A. Finkelstein, “Pairedcyclegan: Asymmetric style transfer for applying and removing makeup,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 40–48.
- [15] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua, “Stylebank: An explicit representation for neural image style transfer,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1897–1906.

- [16] X. Chen, Y. Duan, R. Houthoofd, J. Schulman, I. Sutskever, and P. Abbeel, “Infogan: Interpretable representation learning by information maximizing generative adversarial nets,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2172–2180.
- [17] X. Chen, C. Xu, X. Yang, L. Song, and D. Tao, “Gated-gan: Adversarial gated networks for multi-collection style transfer,” *IEEE Transactions on Image Processing*, vol. 28, no. 2, pp. 546–560, 2018.
- [18] J. Cheng and M. Lapata, “Neural summarization by extracting sentences and words,” *CoRR*, vol. abs/1603.07252, 2016. [Online]. Available: <http://arxiv.org/abs/1603.07252>
- [19] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: <http://arxiv.org/abs/1409.1259>
- [20] S. Chopra, M. Auli, and A. M. Rush, “Abstractive sentence summarization with attentive recurrent neural networks,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 93–98.
- [21] B. Dai, D. Lin, R. Urtasun, and S. Fidler, “Towards diverse and natural image descriptions via a conditional GAN,” *CoRR*, vol. abs/1703.06029, 2017. [Online]. Available: <http://arxiv.org/abs/1703.06029>
- [22] A. Doucet and A. Johansen, “A tutorial on particle filtering and smoothing: Fifteen years later,” *Handbook of Nonlinear Filtering*, vol. 12, 01 2009.
- [23] D. Eck and J. Schmidhuber, “A first look at music composition using lstm recurrent neural networks,” *Istituto Dalle Molle Di Studi Sull Intelligenza*

Artificiale, vol. 103, 2002.

- [24] H. Eghbal-zadeh and G. Widmer, “Probabilistic generative adversarial networks,” *CoRR*, vol. abs/1708.01886, 2017. [Online]. Available: <http://arxiv.org/abs/1708.01886>
- [25] L. Fei-Fei, R. Fergus, and P. Perona, “Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories,” *Computer vision and Image understanding*, vol. 106, no. 1, pp. 59–70, 2007.
- [26] S. Fidler, S. Dickinson, and R. Urtasun, “3d object detection and viewpoint estimation with a deformable 3d cuboid model,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 611–619. [Online]. Available: <http://papers.nips.cc/paper/4562-3d-object-detection-and-viewpoint-estimation-with-a-deformable-3d-cuboid-model.pdf>
- [27] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [28] J. Gauthier, “Conditional generative adversarial nets for convolutional face generation,” *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, vol. 2014, no. 5, p. 2, 2014.
- [29] R. Girshick, “Fast r-cnn,” in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.
- [30] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

- [31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>
- [32] S. Gopal, “Adaptive sampling for sgd by exploiting side information,” in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 364–372. [Online]. Available: <http://proceedings.mlr.press/v48/gopal16.html>
- [33] G. Hadjeres and F. Pachet, “Deepbach: a steerable model for bach chorales generation,” *CoRR*, vol. abs/1612.01010, 2016. [Online]. Available: <http://arxiv.org/abs/1612.01010>
- [34] S. Hettich and S. D. Bay, “The uci kdd archive,” 1999.
- [35] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a nash equilibrium,” *CoRR*, vol. abs/1706.08500, 2017. [Online]. Available: <http://arxiv.org/abs/1706.08500>
- [36] W. Huang, R. Y. D. Xu, and I. Oppermann, “Efficient diversified mini-batch selection using variable high-layer features,” in *Proceedings of The Eleventh Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, W. S. Lee and T. Suzuki, Eds., vol. 101. Nagoya, Japan: PMLR, 17–19 Nov 2019, pp. 300–315. [Online]. Available: <http://proceedings.mlr.press/v101/huang19b.html>

- [37] X. Huang and S. Belongie, “Arbitrary style transfer in real-time with adaptive instance normalization,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1501–1510.
- [38] W. Huang, R. Y. D. Xu, and I. Oppermann, “Realistic image generation using region-phrase attention,” in *Proceedings of The Eleventh Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, W. S. Lee and T. Suzuki, Eds., vol. 101. Nagoya, Japan: PMLR, 17–19 Nov 2019, pp. 284–299. [Online]. Available: <http://proceedings.mlr.press/v101/huang19a.html>
- [39] L. Hui, X. Li, J. Chen, H. He, C. Gong, and J. Yang, “Un-supervised multi-domain image translation with domain-specific encoders/decoders,” *CoRR*, vol. abs/1712.02050, 2017. [Online]. Available: <http://arxiv.org/abs/1712.02050>
- [40] J. Hutchins, “Warren weaver and the launching of mt,” *Early Years in Machine Translation*, ed. W. John Hutchins. Amsterdam: John Benjamins. 2000, pp. 17–20, 2000.
- [41] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: <http://arxiv.org/abs/1502.03167>
- [42] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” *CoRR*, vol. abs/1611.07004, 2016. [Online]. Available: <http://arxiv.org/abs/1611.07004>
- [43] U. Jain, Z. Zhang, and A. G. Schwing, “Creativity: Generating diverse questions using variational autoencoders,” *CoRR*, vol. abs/1704.03493, 2017. [Online]. Available: <http://arxiv.org/abs/1704.03493>

- [44] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Z. Fang, “Towards the automatic anime characters creation with generative adversarial networks,” *CoRR*, vol. abs/1708.05509, 2017. [Online]. Available: <http://arxiv.org/abs/1708.05509>
- [45] R. Johnson and T. Zhang, “Accelerating stochastic gradient descent using predictive variance reduction,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 315–323. [Online]. Available: <http://papers.nips.cc/paper/4937-accelerating-stochastic-gradient-descent-using-predictive-variance-reduction.pdf>
- [46] N. Kalchbrenner and P. Blunsom, “Recurrent continuous translation models,” in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 1700–1709.
- [47] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [48] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” *CoRR*, vol. abs/1812.04948, 2018. [Online]. Available: <http://arxiv.org/abs/1812.04948>
- [49] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, “Novel dataset for fine-grained image categorization,” in *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.
- [50] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, “Learning to discover cross-domain relations with generative adversarial networks,” *CoRR*, vol. abs/1703.05192, 2017. [Online]. Available: <http://arxiv.org/abs/1703.05192>

- [51] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [52] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Tech. Rep., 2009.
- [53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [54] A. Kulesza and B. Taskar, “k-dpps: Fixed-size determinantal point processes,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1193–1200.
- [55] A. Kulesza, B. Taskar *et al.*, “Determinantal point processes for machine learning,” *Foundations and Trends® in Machine Learning*, vol. 5, no. 2–3, pp. 123–286, 2012.
- [56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [57] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database,” *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, vol. 2, 2010.
- [58] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, “Photo-realistic single image super-

- resolution using a generative adversarial network.” in *CVPR*, vol. 2, no. 3, 2017, p. 4.
- [59] C. Li, H. Xie, K. Mengersen, X. Fan, R. Y. Da Xu, S. A. Sisson, and S. Van Huffel, “Bayesian nonnegative matrix factorization with dirichlet process mixtures,” *IEEE Transactions on Signal Processing*, vol. 68, pp. 3860–3870, 2020.
- [60] J. Li, M. Luong, and D. Jurafsky, “A hierarchical neural autoencoder for paragraphs and documents,” *CoRR*, vol. abs/1506.01057, 2015. [Online]. Available: <http://arxiv.org/abs/1506.01057>
- [61] M. Li, T. Zhang, Y. Chen, and A. J. Smola, “Efficient mini-batch training for stochastic optimization,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 661–670.
- [62] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, “Diversified texture synthesis with feed-forward networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3920–3928.
- [63] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll’ar, and C. L. Zitnick, “Microsoft COCO: common objects in context,” *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: <http://arxiv.org/abs/1405.0312>
- [64] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, “Are gans created equal? a large-scale study,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 700–709. [Online]. Available: <http://papers.nips.cc/paper/7350-are-gans-created-equal-a-large-scale-study.pdf>

- [65] M. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” *CoRR*, vol. abs/1508.04025, 2015. [Online]. Available: <http://arxiv.org/abs/1508.04025>
- [66] L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool, “Pose guided person image generation,” in *Advances in Neural Information Processing Systems*, 2017, pp. 406–416.
- [67] H. H. Mao, T. Shin, and G. Cottrell, “Deepj: Style-specific music generation,” in *Semantic Computing (ICSC), 2018 IEEE 12th International Conference on*. IEEE, 2018, pp. 377–382.
- [68] J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille, “Deep captioning with multimodal recurrent neural networks (m-rnn),” *CoRR*, vol. abs/1412.6632, 2014. [Online]. Available: <http://arxiv.org/abs/1412.6632>
- [69] T. Marwah, G. Mittal, and V. N. Balasubramanian, “Attentive semantic video generation using captions,” *CoRR*, vol. abs/1708.05980, 2017. [Online]. Available: <http://arxiv.org/abs/1708.05980>
- [70] M. Mathieu, C. Couprie, and Y. LeCun, “Deep multi-scale video prediction beyond mean square error,” *CoRR*, vol. abs/1511.05440, 2015. [Online]. Available: <http://arxiv.org/abs/1511.05440>
- [71] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: <http://arxiv.org/abs/1411.1784>
- [72] E. Nalisnick, L. Hertel, and P. Smyth, “Approximate inference for deep latent gaussian mixtures,” in *NIPS Workshop on Bayesian Deep Learning*, vol. 2, 2016.
- [73] E. Nalisnick and P. Smyth, “Stick-breaking variational autoencoders,” *arXiv preprint arXiv:1605.06197*, 2016.

- [74] R. Nallapati, B. Xiang, and B. Zhou, “Sequence-to-sequence rnns for text summarization,” *CoRR*, vol. abs/1602.06023, 2016. [Online]. Available: <http://arxiv.org/abs/1602.06023>
- [75] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.
- [76] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.
- [77] Y. Pan, Z. Qiu, T. Yao, H. Li, and T. Mei, “To create what you tell: Generating videos from captions,” in *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 2017, pp. 1789–1798.
- [78] Y. Pan, Z. Qiu, T. Yao, H. Li, and T. Mei, “To create what you tell: Generating videos from captions,” *CoRR*, vol. abs/1804.08264, 2018. [Online]. Available: <http://arxiv.org/abs/1804.08264>
- [79] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *CoRR*, vol. abs/1511.06434, 2015. [Online]. Available: <http://arxiv.org/abs/1511.06434>
- [80] S. J. Reddi, A. Hefny, S. Sra, B. Póczos, and A. Smola, “Stochastic variance reduction for nonconvex optimization,” *arXiv preprint arXiv:1603.06160*, 2016.
- [81] S. E. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee, “Learning what and where to draw,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 217–225.

- [Online]. Available: <http://papers.nips.cc/paper/6111-learning-what-and-where-to-draw.pdf>
- [82] S. E. Reed, Z. Akata, B. Schiele, and H. Lee, “Learning deep representations of fine-grained visual descriptions,” *CoRR*, vol. abs/1605.05395, 2016. [Online]. Available: <http://arxiv.org/abs/1605.05395>
- [83] S. E. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” *CoRR*, vol. abs/1605.05396, 2016. [Online]. Available: <http://arxiv.org/abs/1605.05396>
- [84] D. Rezende and S. Mohamed, “Variational inference with normalizing flows,” ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1530–1538. [Online]. Available: <http://proceedings.mlr.press/v37/rezende15.html>
- [85] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization,” *CoRR*, vol. abs/1509.00685, 2015. [Online]. Available: <http://arxiv.org/abs/1509.00685>
- [86] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, “Improved techniques for training gans,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2234–2242. [Online]. Available: <http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf>
- [87] T. Salimans and D. P. Kingma, “Weight normalization: A simple reparameterization to accelerate training of deep neural networks,” in *Advances in Neural Information Processing Systems 29*, D. D.

- Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 901–909. [Online]. Available: <http://papers.nips.cc/paper/6114-weight-normalization-a-simple-reparameterization-to-accelerate-training-of-deep-neural-networks.pdf>
- [88] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” *CoRR*, vol. abs/1703.05921, 2017.
- [89] O. Shamir, “Making gradient descent optimal for strongly convex stochastic optimization,” *CoRR*, vol. abs/1109.5647, 2011. [Online]. Available: <http://arxiv.org/abs/1109.5647>
- [90] R. Shetty, M. Rohrbach, L. A. Hendricks, M. Fritz, and B. Schiele, “Speaking the same language: Matching machine to human captions by adversarial training,” *CoRR*, vol. abs/1703.10476, 2017. [Online]. Available: <http://arxiv.org/abs/1703.10476>
- [91] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014.
- [92] K. Sohn, H. Lee, and X. Yan, “Learning structured output representation using deep conditional generative models,” in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3483–3491. [Online]. Available: <http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf>
- [93] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q.

- Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: <http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>
- [94] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *CoRR*, vol. abs/1512.00567, 2015. [Online]. Available: <http://arxiv.org/abs/1512.00567>
- [95] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.
- [96] J. S. Vincent Dumoulin and M. Kudlur, “A learned representation for artistic style,” in *Proceedings of the International Conference on Learning Representations*, 2017.
- [97] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.
- [98] L. Wang, A. Schwing, and S. Lazebnik, “Diverse and accurate image description using a variational auto-encoder with an additive gaussian encoding space,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5756–5766.
- [99] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, “Caltech-UCSD Birds 200,” California Institute of Technology, Tech. Rep. CNS-TR-2010-001, 2010.
- [100] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa,

- K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: <http://arxiv.org/abs/1609.08144>
- [101] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *CoRR*, vol. abs/1708.07747, 2017. [Online]. Available: <http://arxiv.org/abs/1708.07747>
- [102] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2048–2057. [Online]. Available: <http://proceedings.mlr.press/v37/xuc15.html>
- [103] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, “Attngan: Fine-grained text to image generation with attentional generative adversarial networks,” *CoRR*, vol. abs/1711.10485, 2017. [Online]. Available: <http://arxiv.org/abs/1711.10485>
- [104] S. Yamamoto, A. Tejero-de Pablos, Y. Ushiku, and T. Harada, “Conditional video generation using action-appearance captions,” *arXiv preprint arXiv:1812.01261*, 2018.
- [105] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, “Midinet: A convolutional generative adversarial network for symbolic-domain music generation,” *arXiv preprint arXiv:1703.10847*, 2017.
- [106] L. Yang, S. Chou, and Y. Yang, “Midinet: A convolutional generative adversarial network for symbolic-domain music generation using 1d and

- 2d conditions,” *CoRR*, vol. abs/1703.10847, 2017. [Online]. Available: <http://arxiv.org/abs/1703.10847>
- [107] D. Yoo, N. Kim, S. Park, A. S. Paek, and I. Kweon, “Pixel-level domain transfer,” *CoRR*, vol. abs/1603.07442, 2016. [Online]. Available: <http://arxiv.org/abs/1603.07442>
- [108] C. Zhang, H. Kjellström, and S. Mandt, “Stochastic learning on imbalanced data: Determinantal point processes for mini-batch diversification,” *CoRR*, vol. abs/1705.00607, 2017. [Online]. Available: <http://arxiv.org/abs/1705.00607>
- [109] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” *arXiv preprint arXiv:1805.08318*, 2018.
- [110] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas, “Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks,” *arXiv preprint arXiv:1612.03242*, 2016.
- [111] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, “Stackgan++: Realistic image synthesis with stacked generative adversarial networks,” *CoRR*, vol. abs/1710.10916, 2017. [Online]. Available: <http://arxiv.org/abs/1710.10916>
- [112] P. Zhao and T. Zhang, “Accelerating minibatch stochastic gradient descent using stratified sampling,” *arXiv preprint arXiv:1405.3080*, 2014.
- [113] P. Zhao and T. Zhang, “Stochastic optimization with importance sampling for regularized loss minimization,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1–9.

- [114] J. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” *CoRR*, vol. abs/1703.10593, 2017. [Online]. Available: <http://arxiv.org/abs/1703.10593>
- [115] S. Zhu, S. Fidler, R. Urtasun, D. Lin, and C. C. Loy, “Be your own prada: Fashion synthesis with structural coherence,” in *International Conference on Computer Vision (ICCV)*, 2017.