東北大学機関リポジトリ
TOUR
Tohoku University Repository
東北大学

An Information-theoretic Analysis of Deep Neural Networks

著者	Zhang Yan
学位授与機関	Tohoku University
学位授与番号	11301甲第17786号
URL	http://hdl.handle.net/10097/00122847

TOHOKU UNIVERSITY
Graduate School of Information Sciences

An Information-theoretic Analysis of Deep Neural Networks

(深層ニューラルネットワークの情報理論的分析)

A dissertation submitted for the degree of
Doctor of Philosophy

Department of System Information Sciences

by

Zhang Yan

July 7, 2017

An Information-theoretic Analysis of Deep Neural Networks

Zhang Yan

Abstract

Although Deep Neural Networks (DNNs) have achieved impressive results on many tasks, our understanding of DNNs is limited because they are essentially complicated non-linear mapping functions. In order to obtain better understanding of DNNs and improve performance of DNNs, we analyze the statistical properties of representations learned by DNNs from an information theoretic perspective in different learning tasks. First, we make use of mutual information to regularize Auto-Encoders (AEs) in unsupervised learning tasks. Next, we propose a binary tree architecture to achieve better trade-off between parameter size and accuracy in the supervised learning tasks and then we use mutual information to analyze the optimality of learned representations by the proposed architecture. Moreover, we use mutual information to measure the dependency between object representations and material categories in transfer learning tasks.

To be more precise, for unsupervised learning, we propose a framework to utilize mutual information as a regularization criterion to learn AEs. In our proposed framework, AEs are regularized by minimization of the mutual information between input and encoding variables of AEs during the training phase. In order to estimate the entropy of the encoding variables and the mutual information, we propose a non-parametric method. We also give an information theoretic view of Variational AEs (VAEs), which suggests that VAEs can be considered as parametric methods that are used to estimate entropy. Experimental results show that the proposed non-parametric methods have more degree of freedom in terms of representation learning of features drawn from complex distributions such as Mixture of Gaussians, compared to methods which estimate entropy using parametric approaches, such as Variational AEs.

In supervised learning, we address a problem of improvement of parameter size and accuracy trade-off of wide DNNs. Thus, we propose a binary tree architecture to truncate architecture of wide networks by reducing width of the networks. More precisely, in the proposed architecture, the width of a wide network is incrementally reduced from lower layers to higher layers in order to increase the expressive capacity of network with a less increase on parameter size. Moreover, features obtained at different layers are concatenated to form an output of our architecture to ease the gradient vanishing problem. By employing the proposed architecture on a baseline wide network, we can construct

and train a new network with same depth but considerably less number of parameters. Information theoretic analyses also show that the proposed architecture can obtain better optimality of learned representations with a less increase of parameter size compared with baselines. In our experimental analyses, we observed that the proposed architecture enables us to obtain better parameter size and accuracy trade-off compared to baseline networks using various benchmark image classification datasets.

For transfer learning, we study how to select and integrate multiple features obtained using different pretrained models of DNNs. In this task, we address a material recognition problem. The motivation is to explore and utilize the relationships between material and object recognition problems to improve material recognition accuracy of the models. More precisely, we first extract features from images using pretrained models to select a set of samples which are best represented by the features. Then, we measure uncertainty of the features by computing entropy of class distributions for each sample set. Finally, we compute contribution of each feature to discrimination of classes for feature selection and integration. Experimental results show that the proposed method achieves state-of-the-art performance on two benchmark datasets for material recognition. Additionally, we introduce a new material dataset, named EFMD, which extends Flickr Material Database (FMD). By the employment of the EFMD for transfer learning, we achieve $\mathbf{84.0}\% \pm \mathbf{1.8}\%$ accuracy on the FMD dataset, which is close to the reported human performance $\mathbf{84.9}\%$.

Experimental results provided in this work indicate that information theoretic methods are efficient tools to analyze DNNs, and thus can be used to improve the training of DNNs in the unsupervised learning, supervised learning and transfer learning tasks.

Contents

Chapter 1

Introduction

1.1 Problems

Recently, Deep Neural Networks (DNNs) have achieved impressive results on many tasks [41, 65, 50, 69, 24, 70, 25, 68, 36, 18]. However, our understanding on DNNs is limited. DNNs are difficult to understand due to its hierarchical structure of multiple layers, each of which computes a non-linear function. Thus, a DNN is essentially a complicated non-linear mapping function which maps an input random variable to output random variables, i.e. deep representations. One way to understand a complicated non-linear function is to analyze the statistical properties of representations computed by the function. Information theory is a good choice for this task. However, it has not been used for this purpose so far with only a few exceptions, e.g. [1].

In this study, we apply information theory to obtain a better understanding on DNNs and improve performance of DNNs for different tasks. Specifically, we are interested in the following three learning tasks:

- *Unsupervised learning* of DNNs by Auto-Encoders: In this task, we are given training data without labels. The target is to learn the underlying useful structure of the data. In some realistic problems, it is usually difficult to obtain the labels and thus unsupervised learning methods are critical to solve these problems. Auto-encoders (AEs) [27] are popular unsupervised learning methods, in which neural networks are constructed in an encoding

decoding manner.

- *Supervised learning* of DNNs from scratch: For this task, we are given training data together with labels. Supervised learning is widely used in many practical application problems. Various architectures of DNNs can be used for learning in order to obtain good classification accuracy. However, we also need to consider parameter size and depth of DNNs, as the memory consumption and computational cost is critical for the practical applications.

- *Transfer learning* of DNNs from pre-trained models: In this task, we are given DNN models pre-trained on a dataset. The goal is to use the pre-trained models to solve the current task. Although the pre-trained models are trained on different tasks, it sometimes can provide useful information and assist solving the current task.

In order to address the aforementioned three tasks, we need to consider the following problems:

i) For unsupervised learning of DNNs using Auto-Encoders (AEs) [27], the encoding functions of AEs are implemented by neural networks, i.e. encoders. The encoders are coupled by decoders which aim to reconstruct an input variable from an encoding variable. AEs are learned by minimizing the reconstruction error. Using this learning scheme, the encoder is expected to capture the underlying *useful* structure of the input variable. However, a *trivial* solution to such an optimization problem can be obtained by setting an identity mapping between the input variable and the encoding variable. The learned encoder cannot be useful with a trivial solution. Thus, **regularization** methods are required to avoid trivial solutions and learn representations using AEs.

ii) For supervised learning tasks, one approach used to improve accuracy of a DNN is to widen each layer. For instance, it was empirically shown in [78] that a wide but relatively shallow DNN can obtain accuracy comparable to a narrow but relatively deep DNN on several classification tasks. There are two crucial benefits of wide-shallow DNNs. First, it usually runs *faster* than narrow-deep DNNs on parallel computing devices, e.g. GPUs, as illustrated in [78]. Also, a deep DNN with many layers may suffer from a *gradient vanishing problem*. Reducing the depth can ease this problem as shown in [29]. However, *parameter size* of DNNs may significantly increase with respect to improvement of accuracy by widening each layer. It is important to balance the *parameter size and*

accuracy trade-off.

iii) For transfer learning, we address a material recognition problem. We are particularly interested in transferring knowledge from object recognition tasks to material recognition tasks. By utilizing interconnection between material and object recognition, we aim to perform material recognition accurately. Various studies of psychophysics [64, 53] imply that material perception in human vision is interconnected with perception of object category. An observation is that human can perceive some material properties and categories of objects only after correct recognition of the object categories. However, it is challenging to model such dependencies, and utilize them to accurately perform a task of interest (material recognition in our case). It is important to select **useful** object and material features for the proposed material recognition task.

In this thesis, we analyze these problems from an information theoretic perspective. Specifically, for unsupervised learning of DNNs using AEs, we suggest a framework to make use of mutual information as a regularization criterion to train AEs. In the proposed framework, AEs are regularized by minimization of the mutual information between input and encoding variables of AEs during the training phase. In order to estimate the entropy of the encoding variables and the mutual information, we propose a non-parametric method. It is not required to make any assumption for a particular parameterized form of distribution of an encoding variable. This property gives more degree of freedom to learn the distribution of an encoding variable during the representation learning tasks. We also give an information theoretic view of Variational AEs (VAEs) [36], which suggests that the mutual information regularization used in VAEs can be considered as a mapping of all samples to a point (e.g. $\mathbf{0}$) in the encoding space. On the other hand, the proposed method minimizes the mutual information by reducing pairwise distances. Experimental results show that the proposed non-parametric models have more degree of freedom in terms of representation learning of features drawn from complex distributions such as Mixture of Gaussians [7], compared to methods which estimate entropy using parametric approaches, such as Variational AEs.

For supervised learning of DNNs from scratch, in order to both keep the benefits of wide networks and improve the parameter size and accuracy trade-off of wide networks, we propose a binary tree architecture. The proposed architecture is used to truncate architecture of wide networks

by reducing the width of the networks. More precisely, in the proposed architecture, the width is incrementally reduced from lower layers to higher layers in order to increase the expressive capacity of network with a less increase of parameter size. Also, in order to ease the gradient vanishing problem, the output of our architecture is formed by concatenation of features obtained at different layers. By employing the proposed architecture on a baseline wide network, we can construct and train a new network with same depth but considerably less number of parameters. Information theoretic analyses also show that the proposed architecture can obtain better optimality of learned representations with a less increase of parameter size compared with baselines. In our experimental analyses, we observe that the proposed architecture enables us to obtain better parameter size and accuracy trade-off compared to baseline networks using various benchmark image classification datasets. The results show that our model can be used to decrease the classification error of a baseline network from 20.43% to 19.22% on the Cifar-100 dataset using only 28% of parameters that baseline has.

In the transfer learning task, we propose a feature selection method to select and combine deep features learned using a transfer learning setting. First, we compute feature activations on images using learned feature representations to select a set of samples which are *best* represented by the features. Then, we measure uncertainty of the features by computing entropy of class distributions for each sample set. Finally, we compute contribution of each feature to class discrimination for feature selection and integration. In the experimental results, we obtain state-of-the-art performance by employing the features integrated using the proposed method on several benchmark datasets.

The rest of this thesis is organized as follows. In the next part of Chapter 1, we introduce elementary concepts of DNNs and Information Theory as the required background. In Chapter 2, we introduce an information theoretic framework for training of AEs. We introduce a binary tree architecture to improve the parameter size and accuracy trade-off for supervised learning tasks in Chapter 3. Chapter 4 gives the proposed method for transfer learning task. Finally, conclusion is given in Chapter 5.

1.2 Preliminaries

In this section, we first provide the notations used in this thesis, and then provide the required background on deep learning and information theoretic methods.

1.2.1 Terminology and Notation used in the Thesis

Unless specified, terminology and notation used in this thesis are employed as follows:

- A *Deep Neural Network* is defined as a (non-linear) mapping function which maps an input random variable to an output random variable. The mapping function is a composition of several (at least two) subfunctions.

- A *representation* is a random variable computed by a DNN.

- Random variables are denoted by bold letters. Random vectors and scalar variables are denoted by lowercase bold letters (e.g. $\mathbf{x}$), and random matrices are denoted by capital bold letters (e.g. $\mathbf{X}$).

- Realizations of random variables are denoted by normal typeface of letters (e.g. x, X).

- Deterministic scalar values are also denoted by normal typeface of letters (e.g. number of training samples: N, width of an image: w).

- Calligraphic letters (e.g. $\mathcal{X}$, $\mathcal{Y}$) denote sets. Usually, it is used to denote a set of training samples, e.g. $\mathcal{X} = \{x_i\}_{i=1}^{n}$. It is also used to denote a space to which a random variable belongs, i.e. a set of all possible realizations of a random variable, denoted by $\mathbf{x} \in \mathcal{X}$.

- Upper indices are used to index elements in the vectors and matrices, e.g. $\mathbf{X}^{(i,j)}$ is the value of element on the i-th column and j-th row of the matrix $\mathbf{X}$.

- Expectation over a distribution $p(\mathbf{x})$ is denoted by $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\cdot]$.

- $H(\cdot)$ denotes entropy, and $I(\cdot,\cdot)$ denotes mutual information.

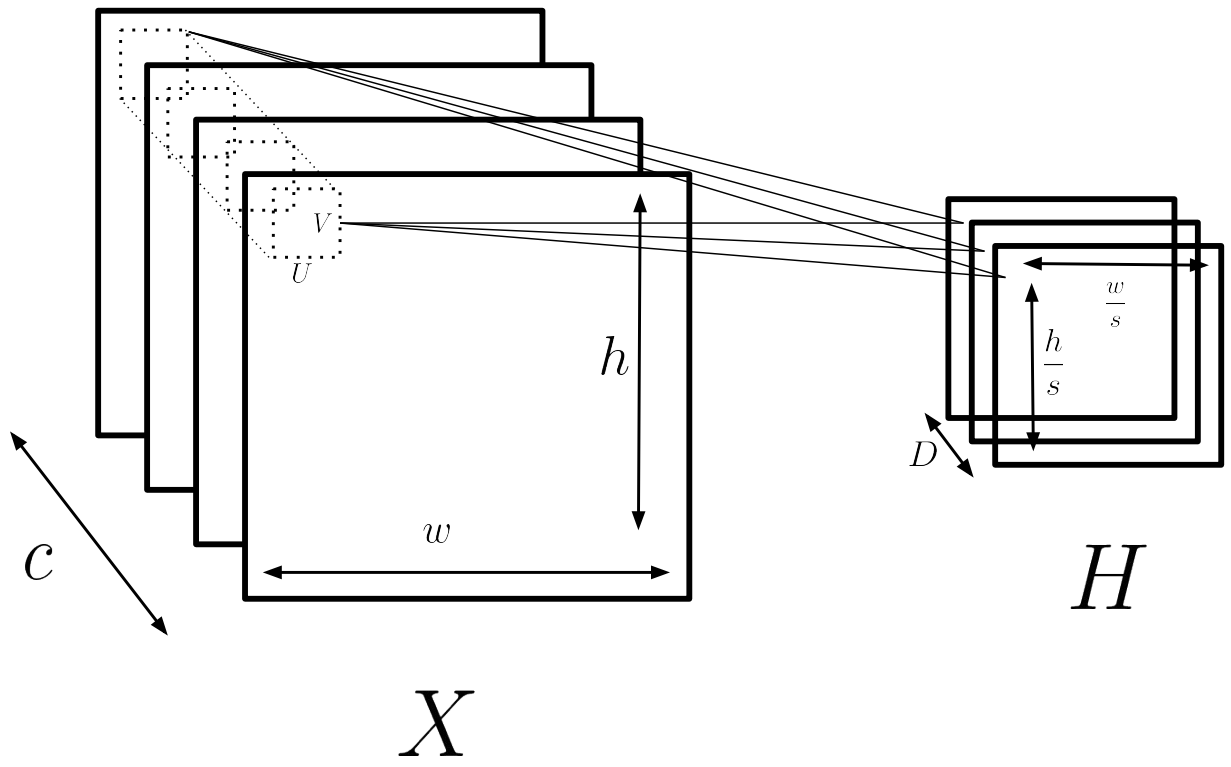


Figure 1.1: An illustration of convolutional module.

1.2.2 Deep Neural Networks

A DNN is a composition of several subfunctions.We denote each subfunction as a module. Commonly used modules are e.g. convolutional modules, fully connected modules etc.

Convolutional Modules: In this thesis, we consider two dimensional convolutional modules (unless specified, *convolution* always indicates a two dimensional convolution operation in this thesis). An input to a convolutional module is usually a multi-dimensional matrix, denoted by a *tensor* in this thesis. Let a tensor $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$ be an input to a convolutional module. The tensor $\mathbf{X}$ is also called an array of feature maps, where c is the number of channels of the feature maps, and w and h is the width and height of each channel, respectively. An output of a convolutional module is also a tensor, denoted as $\mathbf{H}$. Let the number of output channels be D, the stride be s, the width and height of each convolutional filter be U and V, respectively. Then, the i-th and j-th element of $\mathbf{H}$ on

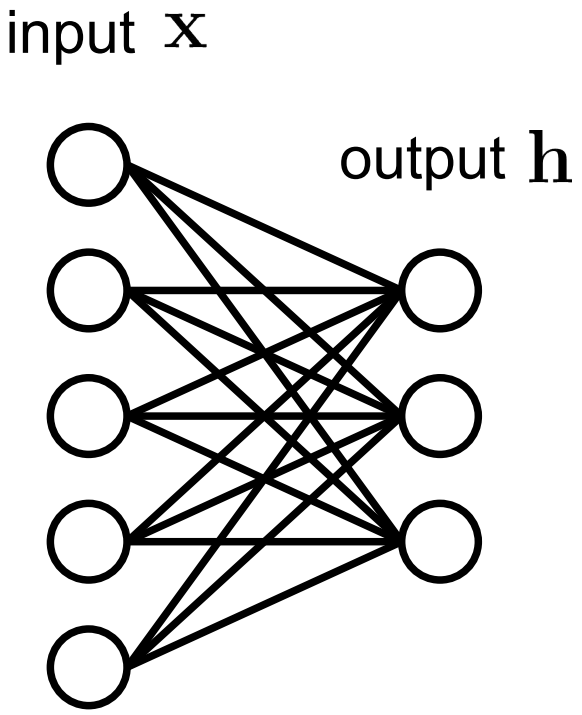


Figure 1.2: An illustration of a fully connected module.

its d-th channel is computed by

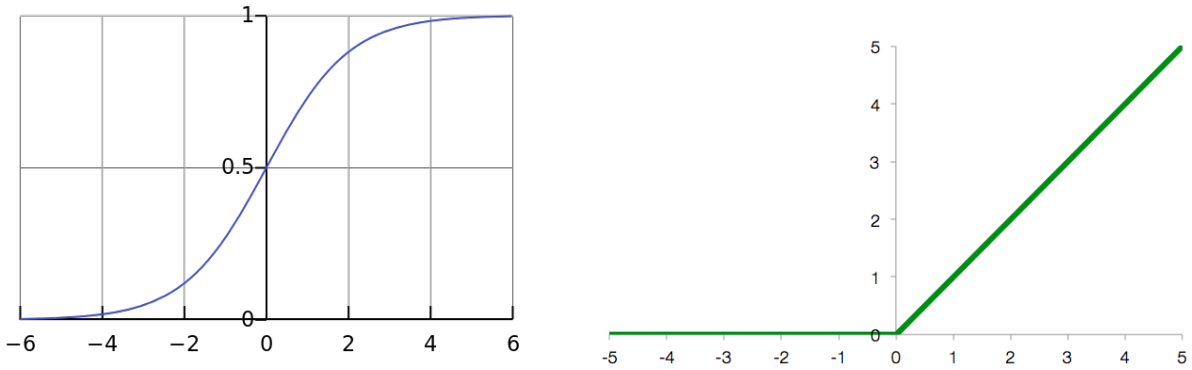
$$\mathbf{H}^{(i,j,d)} = \sum_{k=1}^{c} \sum_{u=1}^{U} \sum_{v=1}^{V} \mathbf{W}^{(u,v,k,d)} \mathbf{X}^{(s\times i+u, s\times j+v, k)} + \mathbf{b}^{(d)}, \tag{1.1}$$

where $\mathbf{W}$ and $\mathbf{b}$ are convolutional filters and biases. An illustration of a convolutional module is given in Figure 1.1.

Fully Connected Modules: An input to a fully connected module is usually a vector. Let an n dimensional vector $\mathbf{x} \in \mathbb{R}^n$ be an input variable. A fully connected layer can be formulated by

$$\mathbf{h} = \mathbf{w}\mathbf{x} + \mathbf{b}, \tag{1.2}$$

where $\mathbf{h} \in \mathbb{R}^m$ is an output variable, and $\mathbf{w} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ are weight and bias parameters, respectively. All elements of the input variable and all elements of the output variable are connected



(a) Sigmoid function. (b) ReLU function.

Figure 1.3: Sigmoid and ReLU functions.

in a fully connected module. An illustration of fully connected module is given in Figure 1.2.

Activation Functions: Activation functions are implemented using non-linear functions to compute the output of various modules of DNNs e.g. convolutional modules. Commonly used activation functions are sigmoid function, Rectified Linear Unit (ReLU) [54], Parametric ReLU [23], Exponential ReLU [13] and so on. Here, we give mathematical definitions of Sigmoid function and ReLU as they will be used in the following sections. The sigmoid activation function is computed by

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \tag{1.3}$$

and the ReLU is computed by

$$\text{relu}(x) = \max(0, x), \tag{1.4}$$

where x is a scalar. If input variables are vectors or matrices, then activation functions return element-wise results of (1.3) and (1.4). The curves computed by Sigmoid and ReLU are given in Figure 1.3a and Figure 1.3b, respectively.

Pooling Modules: Pooling modules are widely used in DNNs to reduce the size of feature maps. There are average pooling and max pooling modules. Given a tensor $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$, an average

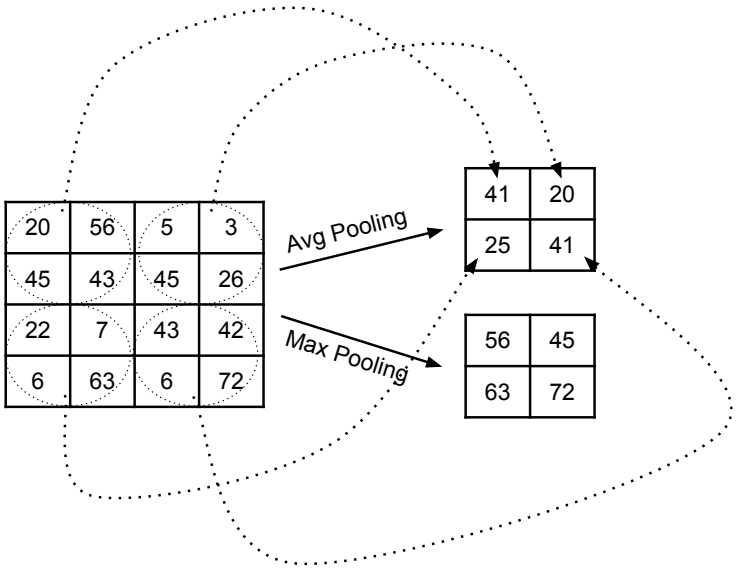


Figure 1.4: An illustration of average and max pooling modules with stride 2 and filter size 2.

pooling module with stride s and filter size U computes the following function on each k-th channel of $\mathbf{X}$;

$$\mathbf{H}^{(i,j,k)} = \frac{1}{U \times U} \sum_{u=1}^{U} \sum_{v=1}^{U} \mathbf{X}^{(s\times i+u, s\times j+v, k)}. \tag{1.5}$$

For a max pooling module, we compute

$$\mathbf{H}^{(i,j,k)} = \max(\mathbf{X}^{(s\times i+1, s\times j+1, k)}, \ldots, \mathbf{X}^{(s\times i+u, s\times j+v, k)}, \ldots, \mathbf{X}^{(s\times i+U, s\times j+U, k)}). \tag{1.6}$$

The size of each feature map belonging to $\mathbf{X}$ is reduced by a factor of s. An illustration of average and max pooling modules is given in Figure 1.4.

Batch Normalization Modules: Batch Normalization modules [31] aim to reduce the internal covariate shift in DNNs. The internal covariate shift is caused by the change of distribution of activations in each layer during training. Batch Normalization modules address this problem by

normalizing the mini batch inputs of each layer by

$$\begin{aligned} \mu &= \frac{1}{n}\sum_{i=1}^{n} x_i, \\ \sigma^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2, \\ \hat{x}_i &= \frac{x_i-\mu}{\sqrt{\sigma^2+\epsilon}}, \\ y_i &= \gamma\hat{x}_i + \beta, \end{aligned} \tag{1.7}$$

where x_i is the i-th sample obtained in a mini batch, ϵ is a small constant used to avoid zero division, γ and β are parameters to be learned. The above computations (1.7) are employed at fully connected layers. For convolutional layers, the mean μ and variance σ are computed for each feature map of mini batch samples. More precisely, given the i-th tensor $X_i \in \mathbb{R}^{w\times h\times c}$, we compute

$$\begin{aligned} t_i^{(k)} &= \frac{1}{w\times h}\sum_{u=1}^{w}\sum_{v=1}^{h} X_i^{(u,v,k)}, \quad \forall k = 1,\ldots,c, \\ \mu &= \frac{1}{n}\sum_{i=1}^{n} t_i, \\ \sigma^2 &= \frac{1}{n}\sum_{i=1}^{n}(X_i-\mu)^2, \\ \hat{X}_i &= \frac{X_i-\mu}{\sqrt{\sigma^2+\epsilon}}, \\ Y_i &= \gamma\hat{X}_i + \beta, \end{aligned} \tag{1.8}$$

where $t_i^{(k)}$ is the mean value of the k-th feature map of the i-th tensor in the mini batch. Batch Normalization enables us to train DNNs with larger initial learning rates and much less training steps to converge.

Residual Shortcut Modules: Residual shortcuts were introduced in [24]. Residual shortcuts enable us to train considerably deeper DNNs. The deepest ResNets, which were employed for classification using the ILSVRC12 [59] and Cifar datasets, have 200 and 1000 layers, respectively [24, 25], and they achieved impressive performance. A residual shortcut is implemented by adding an identity

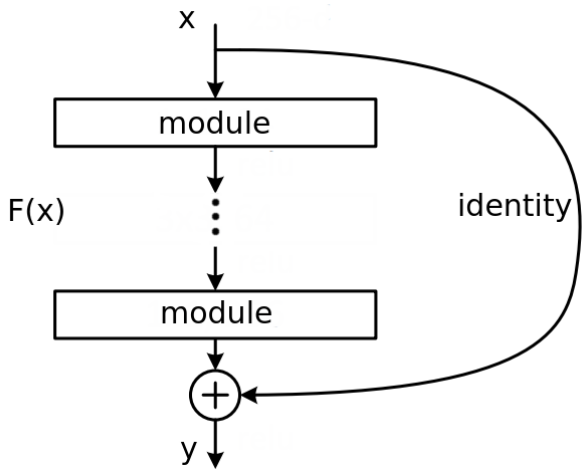


Figure 1.5: An illustration of a residual module.

mapping of lower layer features to higher layers. A residual module computes

$$\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x}, \tag{1.9}$$

where $\mathbf{x}$ is the input to a residual shortcut module, and $\mathcal{F}(\cdot)$ is a composition of several modules that are used to compute higher layer features. An illustration of a residual module is given in Figure 1.5.

Softmax Modules: Softmax modules of DNNs are usually used as classifiers. For a K-category classification task, a softmax module computes a K-dimensional vector, which represents class memberships, by

$$p(\mathbf{y} = k|\mathbf{x}) = \frac{e^{\mathbf{w}^{(k)}\mathbf{x}}}{\sum_{i=1}^{K} e^{\mathbf{w}^{(i)}\mathbf{x}}}, \tag{1.10}$$

where $\mathbf{x}$ is the input feature vector and $\mathbf{w}$ is the weight parameter. For various classification tasks [24, 68, 25, 78, 70], the cost function of DNNs is implemented by the cross entropy loss, which is

defined by

$$\mathcal{L} = -\frac{1}{n}\sum_{j=1}^{n}\sum_{k=1}^{K}\mathbb{1}\{\mathbf{y}_j = k\}\log\frac{e^{\mathbf{w}^{(k)}\mathbf{x}_j}}{\sum_{i=1}^{K} e^{\mathbf{w}^{(i)}\mathbf{x}_j}}, \tag{1.11}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function, and $\mathbf{y}_j$ is the ground truth label of the sample $\mathbf{x}_j$.

Next, we introduce two DNN models that are used in our thesis.

VGG: The VGG architecture [65] consists of convolutional layers, pooling layers, fully connected layers and softmax modules. A convolutional layer of a VGG is defined by a composition of; convolutional module $\rightarrow$ activation module (ReLU). Similarly, a fully connected layer is a composition of; fully connected module $\rightarrow$ activation module (ReLU).

ResNet: The ResNet architecture proposed in [24] consists of residual modules, convolutional layers, pooling layers, fully connected layers and Softmax modules. A convolutional layer in ResNets is a composition of; convolutional module $\rightarrow$ batch normalization module $\rightarrow$ activation module (ReLU). Similarly, a fully connected layer is a composition of; fully connected module $\rightarrow$ batch normalization module $\rightarrow$ activation module (ReLU). A typical residual module consists of two convolutional layers as shown in Figure 1.6.

VGG-16 and ResNet-34 architectures are given in Figure 1.7.

1.2.3 Information Theory

Information theoretic methods are widely used in many machine learning problems. In this thesis, we apply some commonly used information theoretic methods to analyze statistical properties of representations learned using DNNs.We first give some basic definition of information theoretic methods.

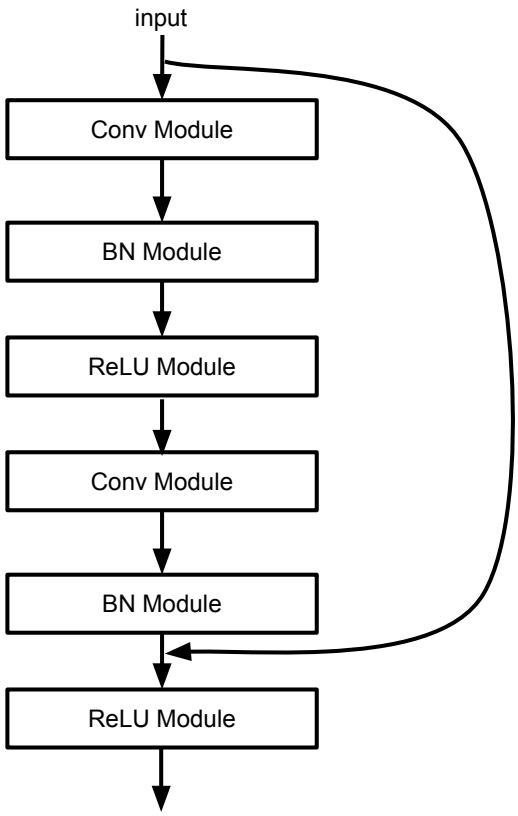


Figure 1.6: An illustration of a residual module proposed in [24].

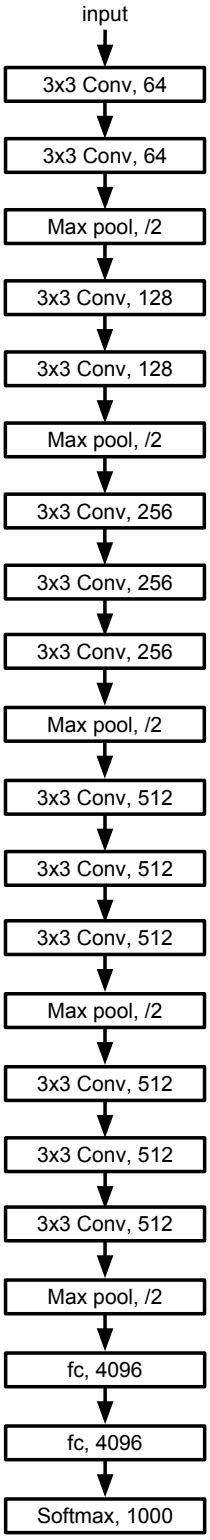
Entropy: Given a random variable $\mathbf{x}$ distributed by a distribution $p(\mathbf{x})$, the entropy [15] of $\mathbf{x}$ is defined by

$$H(\mathbf{x}) = -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \log p(\mathbf{x}). \tag{1.12}$$

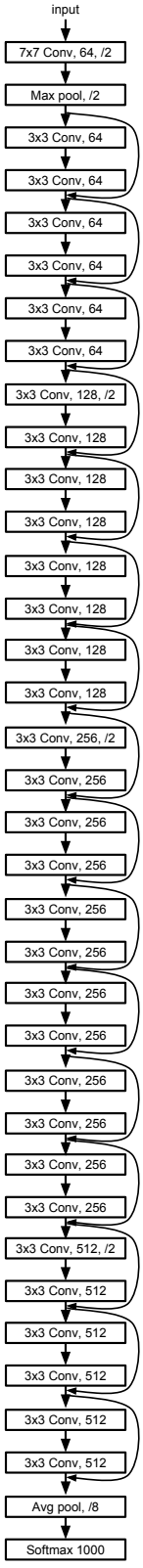
Usually, we are given a set of independent and identically distributed instances of realizations of $\mathbf{x}$, denoted by $\{x_i\}_{i=1}^N$. Then, the empirical value of $H(\mathbf{x})$ can be estimated by

$$H(\mathbf{x}) \simeq -\sum_{i=1}^{N} \log p(\mathbf{x} = x_i). \tag{1.13}$$

The entropy measures the uncertainty of distribution of $\mathbf{x}$. If the distribution $p(\mathbf{x})$ follows a uniform distribution, then the entropy is maximized.



(a) VGG-16.



(b) ResNet-34.

Figure 1.7: VGG-16 and ResNet-34 architectures. Each rectangle represents a layer. $/2$ denotes that stride 2 is used in the corresponding layer.

Mutual Information: Suppose that we are given two random variables $\mathbf{x}$, $\mathbf{y}$, and their distributions $p(\mathbf{x})$, $p(\mathbf{y})$, $p(\mathbf{x}, \mathbf{y})$. The mutual information [15] between $\mathbf{x}$ and $\mathbf{y}$ is defined by

$$\begin{aligned} I(\mathbf{x}; \mathbf{y}) &= H(\mathbf{y}) - H(\mathbf{y}|\mathbf{x}) \\ &= H(\mathbf{x}) - H(\mathbf{x}|\mathbf{y}), \end{aligned} \tag{1.14}$$

where the conditional entropy $H(\mathbf{y}|\mathbf{x})$ is computed by

$$H(\mathbf{y}|\mathbf{x}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim p(\mathbf{x},\mathbf{y})} \log p(\mathbf{y}|\mathbf{x}). \tag{1.15}$$

The mutual information measures the amount of information obtained from a variable $\mathbf{x}$ ($\mathbf{y}$) given the other variable $\mathbf{y}$ ($\mathbf{x}$). Mutual information is used in the information theoretic methods such as Information Bottleneck [72] and Rate Distortion theory [15].

Chapter 2

Information Potential Auto-Encoders

In this chapter, we suggest a framework to make use of mutual information as a regularization criterion to train Auto-Encoders (AEs) [27]. In the proposed framework, AEs are regularized by minimization of the mutual information between input and encoding variables of AEs during the training phase. In order to estimate the entropy of the encoding variables and the mutual information, we propose a non-parametric method. We also give an information theoretic view of Variational AEs (VAEs) [36], which suggests that VAEs can be considered as parametric methods that are used to estimate entropy. Experimental results show that the proposed non-parametric models have more degree of freedom in terms of learning of feature representations drawn from complex distributions, such as Mixture of Gaussians [7], compared to methods which estimate entropy using parametric approaches, such as Variational AEs.

2.1 Motivation

We address a representation learning problem as follows. Let $\mathbf{x}$ be an input (observed) random variable of an AE. In our problem of interest, we consider a representation learning task for learning of an encoding function $f(\cdot)$, where the input variable $\mathbf{x}$ is encoded to a new variable, $\mathbf{z} = f(\mathbf{x})$, called an encoding variable.

AEs are unsupervised representation learning methods, in which encoding functions are im-

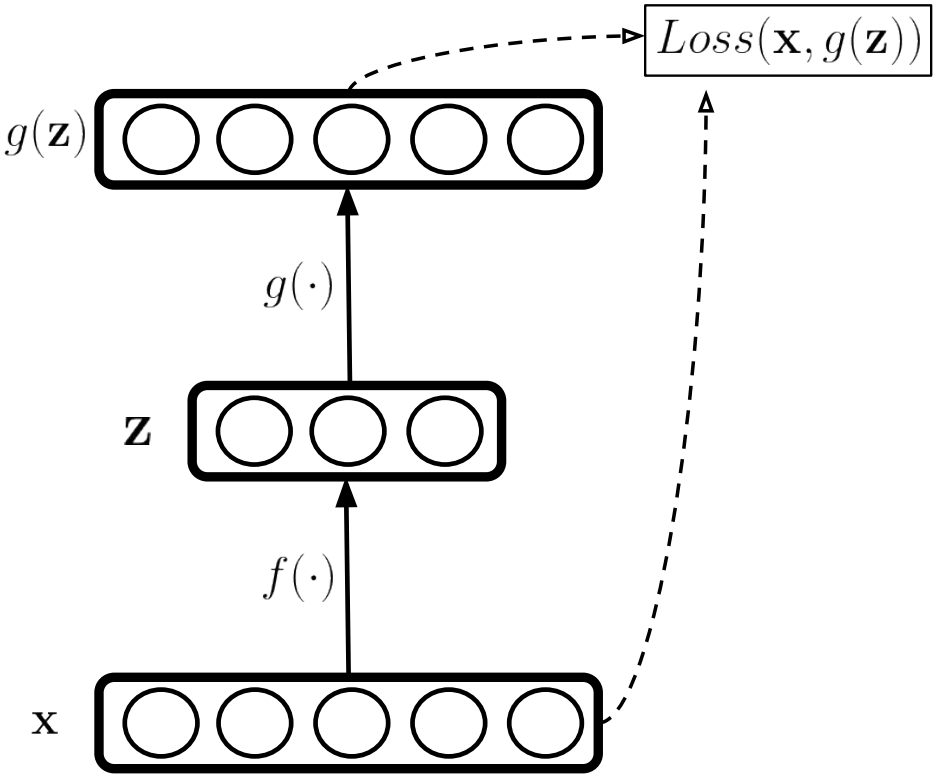


Figure 2.1: A framework of Auto-Encoder for unsupervised representation learning, where $\mathbf{x}$ is an input variable and $\mathbf{z}$ is an encoding variable. $f(\cdot)$ and $g(\cdot)$ are encoding and decoding functions to be learned. AEs can be learned by minimizing the reconstruction loss between input variable $\mathbf{x}$ and decoding variable $g(\mathbf{z})$.

plemented by neural networks, i.e. encoders. The encoders are coupled by decoders which aim to reconstruct the input variable from the encoding variable. AEs can be learned by minimizing the reconstruction error. A framework of AE is shown in Figure 2.1. Using this learning scheme, the encoder is expected to capture the underlying *useful* structure of the input variable. However, a trivial solution to such an optimization problem can be obtained by setting an identity mapping between the input variable and the encoding variable. The learned encoder cannot be useful with a trivial solution. Thus, regularization methods are required to avoid trivial solutions, and to learn representations using AEs.

In this work, we consider development of regularization methods for AEs from an information

theoretic perspective. Information theoretic methods have been widely used to develop regularizers or objectives utilized for learning of representations [72, 74, 16]. Mutual information $I(\mathbf{x};\mathbf{z})$ between an input variable $\mathbf{x}$ and an encoding variable $\mathbf{z}$ has been employed as a regularizer for representation learning. $I(\mathbf{x};\mathbf{z})$ quantifies the information content between $\mathbf{x}$ and $\mathbf{z}$. If we consider the encoding function as a compressor which compresses $\mathbf{x}$ into $\mathbf{z}$, then $I(\mathbf{x};\mathbf{z})$ can be seen as a measure of compression degree. Theoretical results given by [61] indicate that a boost of generalization performance can be obtained by regularizing supervised learning methods using $I(\mathbf{x};\mathbf{z})$ in supervised learning problems. In unsupervised learning tasks, mutual information has been also used to develop regularizers as shown in [74] and [16].

Estimation of mutual information $I(\mathbf{x};\mathbf{z})$ is a key task for employment of information theoretic regularizers. Therefore, $I(\mathbf{x};\mathbf{z})$ can be decomposed into two entropy terms; i) $H(\mathbf{z}|\mathbf{x})$, and ii) $H(\mathbf{z})$. The first term requires modeling of a conditional distribution $p(\mathbf{z}|\mathbf{x})$. Recently, a stochastic encoding mechanism has been introduced to train deep networks in [36]. In this encoding mechanism, we can obtain the distribution $p(\mathbf{z}|\mathbf{x})$, and thus $H(\mathbf{z}|\mathbf{x})$ can be analytically computed. Meanwhile the stochastic sampling from $p(\mathbf{z}|\mathbf{x})$ during the backpropagation can be efficiently performed by a re-parameterization trick. Thereby, $H(\mathbf{z}|\mathbf{x})$ is computed using a parametric model. As for $H(\mathbf{z})$, we need to estimate $p(\mathbf{z})$. A solution of this challenge is employment of a variational distribution $q(\mathbf{z})$ to obtain an upper bound of $H(\mathbf{z})$ as shown in [1]. However, in order to use the variational distribution, it is assumed that $p(\mathbf{z})$ has the same distribution as $q(\mathbf{z})$, which is a pre-defined distribution such as Gaussian. Thus, $H(\mathbf{z})$ is estimated under a parameteric model. In many cases, we cannot assure that an encoding variable should be drawn from a Gaussian distribution.

To this end, we propose a non-parametric method in order to compute $H(\mathbf{z})$. Our contributions can be summarized as follows:

1. In our proposed method, it is not required to make any assumption for a particular parameterized form of distribution $p(\mathbf{z})$ of an encoding variable. This property gives more degree of freedom to learn the distribution $p(\mathbf{z})$ during the representation learning tasks, as examined in the experimental analyses.

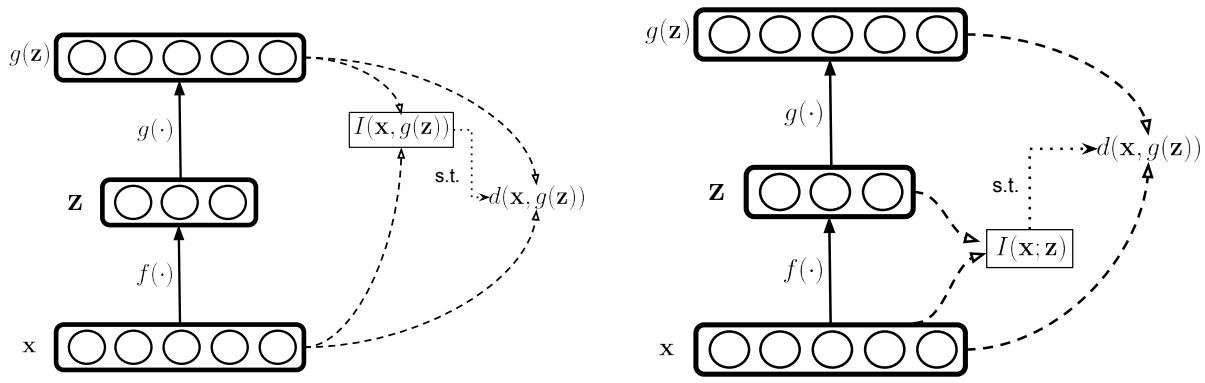
2. We employed the proposed method to train AEs with mutual information regularization. As shown in Section 2.2, by the employment of the proposed method, regularization of AEs by minimization of mutual information can be considered as minimization of the pairwise distance between samples in the encoding space. Therefore, we obtain an analogue to the *information potentials* scheme of physical particles [74] for training of AEs. The information potential scheme of AEs aims to map any two different samples from the input space to the same sample in the encoding space. It enables AEs to avoid trivial solutions of identity mapping. We call AEs trained by the proposed method, Information Potential AEs (IPAEs).

3. We provide an information theoretic view of Variational Auto-Encoders (VAEs). The regularization term employed by VAEs has the same form of minimization of mutual information used in parametric methods. We show that the mutual information regularization used in VAEs can be considered as mapping of all samples to a point (e.g. $\mathbf{0}$) in the encoding space. On the other hand, the proposed method minimizes the mutual information by reducing pairwise distances. Experimental results show that our method, which was proposed for regularization of AEs by minimization of mutual information, enables IPAEs to have more degree of freedom in terms of representation learning of features drawn from complex distributions such as Mixture of Gaussians, compared to VAEs.

2.2 Methods

2.2.1 Problem Definition

Let a random variable $\mathbf{x}$ be the input variable to an AE, and $f(\cdot)$ and $g(\cdot)$ be mapping functions of an encoder and a decoder, respectively. Encoding variable is denoted by $\mathbf{z} := f(\mathbf{x})$. Suppose that we are given a set of independent and identically distributed (i.i.d.) observations $\mathcal{X} = \{x_i\}_{i=1}^{N}$. Then, our framework proposed for training of AEs is as follows. We define an optimization objective by minimizing the mutual information between the input variable and the encoding variable under the



(a) AEs proposed in [17] (b) IPAE

Figure 2.2: The framework of AEs proposed in [17], and our proposed IPAEs.

constraint that the reconstruction error is smaller than a given threshold, which is formulated by

$$\begin{aligned} &\min_{f,g} I(\mathbf{x}; \mathbf{z}) \\ &\text{s.t.} \quad \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[d(\mathbf{x}, g(\mathbf{z}))] \leq D \end{aligned} \tag{2.1}$$

where $d(\cdot, \cdot)$ is a distance function, and D is a given threshold. The objective can be seen as a rate distortion problem, which aims to find the minimal number of bits (measured by $I(\mathbf{x}; \mathbf{z})$) required to encode $\mathbf{x}$ into $\mathbf{z}$ so that the source $\mathbf{x}$ can be reconstructed without exceeding a given distortion threshold D, with respect to the reconstruction error $d(\mathbf{x}, g(\mathbf{z}))$. The rate distortion theory has been used to learn AEs by [17]. The difference comes from the mutual information term; they aim to minimize $I(\mathbf{x}; g(\mathbf{z}))$ while our goal is to minimize $I(\mathbf{x}; \mathbf{z})$. A comparison of AEs proposed in [17] and our IPAE is given in Figure 2.2.

Our encoder can be considered as a lossy data compression model, where the degree of compression is quantified by the information content of the relationship between an input variable and the corresponding encoding variable. A large distortion threshold will give more freedom to AEs to compress the input variable into a smaller number of bits. The criteria of compressing information content will drive the encoder to extract *useful* information from $\mathbf{x}$, and to discard noisy part, and thus avoid trivial solutions.

By introducing a Lagrange multiplier, (2.1) can be reformulated by

$$\min_{f,g} \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[d(\mathbf{x}, g(\mathbf{z}))] + \beta\,(H(\mathbf{z}) - H(\mathbf{z}|\mathbf{x})). \tag{2.2}$$

where β is the inverse value of the introduced Lagrange multiplier controlling a trade-off between mutual information loss and reconstruction loss. In this work, we adopt a stochastic encoding scheme for the encoders, in which the backpropagation of a stochastic encoder can be efficiently performed by a re-parameterization trick as shown in [36]. More precisely, the encoder mapping function is formulated by a Gaussian stochastic mapping

$$\begin{aligned} \mathbf{z} &= f(\mathbf{x}), \\ &= \mu(\mathbf{x}) + \sigma(\mathbf{x}) \odot \boldsymbol{\epsilon}, \end{aligned} \tag{2.3}$$

where $\odot$ denotes element wise product, $\mu(\cdot)$ and $\sigma(\cdot)$ are computed by encoder networks, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. As a result, the conditional distribution of $\mathbf{z}$ given an input variable follows a Gaussian distribution $p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu(\mathbf{x}), \text{diag}(\sigma^2(\mathbf{x})))$. In this case, conditional entropy $H(\mathbf{z}|\mathbf{x})$ can be analytically computed. Next, we consider the problem of estimating $H(\mathbf{z})$.

2.2.2 Estimation of $H(\mathbf{z})$ via Parametric Methods

We first consider parametric methods used for estimating $H(\mathbf{z})$. One approach employed to solve this problem is to first identify a variational distribution $q(\mathbf{z})$. Then, its upper bound is obtained by using the property that Kullback-Leibler divergence is always positive, i.e.

$$\begin{aligned} & KL(p(\mathbf{z})||q(\mathbf{z})) \geq 0 \\ \Rightarrow & \sum_{\mathbf{z}} p(\mathbf{z}) \log p(\mathbf{z}) \geq \sum_{\mathbf{z}} p(\mathbf{z}) \log q(\mathbf{z}) \\ \Rightarrow & H(\mathbf{z}) \leq -\sum_{\mathbf{z}} p(\mathbf{z}) \log q(\mathbf{z}). \end{aligned} \tag{2.4}$$

In this case, the distribution of $q(\mathbf{z})$ can be assumed to be a pre-defined distribution, e.g. Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$, as suggested by [1]. As a result, the upper bound of mutual information $I(\mathbf{x}; \mathbf{z})$ can be computed by

$$\begin{aligned} I(\mathbf{x}; \mathbf{z}) &= H(\mathbf{z}) - H(\mathbf{z}|\mathbf{x}), \\ &= \mathbb{E}_{\mathbf{z},\mathbf{x}\sim p(\mathbf{z},\mathbf{x})}[-\log p(\mathbf{z}) + \log p(\mathbf{z}|\mathbf{x})] \\ &\leq \mathbb{E}_{\mathbf{z},\mathbf{x}\sim p(\mathbf{z},\mathbf{x})}[-\log q(\mathbf{z}) + \log p(\mathbf{z}|\mathbf{x})]. \end{aligned} \tag{2.5}$$

Note that this bound is exactly the same as the regularization term used by Variational Auto-Encoders (VAEs).

In (2.5), we first replace the expectation operation with summation over N empirical samples $\{x_i\}_{i=1}^N$. Next, we compute

$$\begin{aligned} p(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mu(\mathbf{x}), \text{diag}(\sigma^2(\mathbf{x}))) \\ &= \frac{\exp\big(-\frac{1}{2}(\mathbf{z}-\mu(\mathbf{x}))^T \text{diag}(\sigma^2(\mathbf{x}))^{-1}(\mathbf{z}-\mu(\mathbf{x}))\big)}{\sqrt{\big|2\pi \text{diag}(\sigma^2(\mathbf{x}))\big|}} \\ q(\mathbf{z}) &= \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ &= \frac{\exp(-\frac{1}{2}\mathbf{z}^T\mathbf{z})}{\sqrt{\big|2\pi\mathbf{I}\big|}} \end{aligned}, \tag{2.6}$$

where $|\cdot|$ denotes the matrix determinant. Then, we obtain an empirical formulation of the upper

bound of the mutual information using a parametric method as follows:

$$
\begin{aligned}
I(\mathbf{x};\mathbf{z}) &\leq \mathbb{E}_{\mathbf{z},\mathbf{x}\sim p(\mathbf{z},\mathbf{x})}[-\log q(\mathbf{z}) + \log p(\mathbf{z}|\mathbf{x})] \\
&\simeq \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[-\log q(\mathbf{z}) + \log p(\mathbf{z}|x_i)\big] \\
&= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[-\log \frac{\exp(-\frac{1}{2}\mathbf{z}^T\mathbf{z})}{\sqrt{\left|2\pi\mathbf{I}\right|}} \\
&\quad + \log \frac{\exp\big(-\frac{1}{2}(\mathbf{z}-\mu(x_i))^T \mathrm{diag}(\sigma^2(x_i))^{-1}(\mathbf{z}-\mu(x_i))\big)}{\sqrt{\left|2\pi \mathrm{diag}(\sigma^2(x_i))\right|}}\big] \\
&= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[\frac{1}{2}\mathbf{z}^T\mathbf{z} - \frac{1}{2}\log\left|\mathrm{diag}(\sigma^2(x_i))\right| \\
&\quad - \frac{1}{2}(\mathbf{z}-\mu(x_i))^T \mathrm{diag}(\sigma^2(x_i))^{-1}(\mathbf{z}-\mu(x_i))\big] \\
&= \frac{1}{2N}\sum_{i=1}^{N} \Big(\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[\mathbf{z}^T\mathbf{z}\big] \; - \; \log\left|\mathrm{diag}(\sigma^2(x_i))\right| \\
&\quad - \; \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[(\mathbf{z}-\mu(x_i))^T \mathrm{diag}(\sigma^2(x_i))^{-1}(\mathbf{z}-\mu(x_i))\big]\Big).
\end{aligned}
\tag{2.7}
$$

According to the definition of the covariance matrix Σ_z ,

$$\Sigma_z = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\mathbf{z}\mathbf{z}^T] - \mu_z\mu_z^T,$$

if Σ_z is a diagonal matrix, then we have

$$|\Sigma_z| = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\mathbf{z}^T\mathbf{z}] - \mu_z^T\mu_z. \tag{2.8}$$

Also, for a Gaussian distribution $p(\mathbf{z})$, we have

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[(\mathbf{z}-\mu_z)^T\Sigma_z^{-1}(\mathbf{z}-\mu_z)\big] = 1. \tag{2.9}$$

Thus, in (2.7), we have

$$\begin{aligned}\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[\mathbf{z}^T\mathbf{z}\big] &= \mu(x_i)^T\mu(x_i) + \big|\text{diag}(\sigma^2(x_i))\big|,\\ \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[(\mathbf{z}-\mu(x_i))^T\text{diag}(\sigma^2(x_i))^{-1}(\mathbf{z}-\mu(x_i))\big] &= 1.\end{aligned} \tag{2.10}$$

Applying (2.10) to (2.7), we have

$$\begin{aligned}I(\mathbf{x};\mathbf{z}) &\leq \frac{1}{2N}\sum_{i=1}^{N}\Big(\mu(x_i)^T\mu(x_i) + \big|\text{diag}(\sigma^2(x_i))\big| \;-\; \log\big|\text{diag}(\sigma^2(x_i))\big| - 1\Big)\\ &= \frac{1}{2N}\sum_{i=1}^{N}\Big(\big|\big|\mu(x_i)\big|\big|_2^2 + \big|\big|\sigma^2(x_i)\big|\big|_1 \;-\; \log\big|\text{diag}(\sigma^2(x_i))\big| - 1\Big),\end{aligned} \tag{2.11}$$

where $||\cdot||_2^2$ is the square of the ℓ_2 norm, and $||\cdot||_1$ denotes ℓ_1 norm.

2.2.3 Our Proposed Non-parametric Method

The bound (2.11) is obtained by assuming a parametric distribution for $p(\mathbf{z})$. In this paper, we do not make such an assumption. Thus, we propose a non-parametric method to estimate entropy $H(\mathbf{z})$. More precisely, we propose to estimate $p(\mathbf{z})$ non-parametrically by computing

$$\begin{aligned}p(\mathbf{z}) &= \sum_{\mathbf{x}} p(\mathbf{z}|\mathbf{x})p(\mathbf{x})\\ &\simeq \frac{1}{N}\sum_{j=1}^{N} p(\mathbf{z}|x_j).\end{aligned} \tag{2.12}$$

By substituting the above equation into $H(\mathbf{z})$ and applying the Jensen inequality, we obtain

$$\begin{aligned}H(\mathbf{z}) &= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[-\log p(\mathbf{z})]\\ &\simeq \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[-\log\frac{1}{N}\sum_{j=1}^{N}p(\mathbf{z}|x_j)\big],\\ &\leq \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|x_i)}\big[-\frac{1}{N}\sum_{j=1}^{N}\log p(\mathbf{z}|x_j)\big].\end{aligned} \tag{2.13}$$

We first employ the equality $p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu(\mathbf{x}), \text{diag}(\sigma^2(\mathbf{x})))$ and (2.3) in order to generate K samples of $\mathbf{z}$. Then, we have the following bound for $H(\mathbf{z})$,

$$H(\mathbf{z}) \leq \frac{1}{2KN^2} \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{j=1}^{N} \Big(\big(\mu(x_j) - \mu(x_i) - \sigma(x_i) \odot \epsilon_k\big)^2 \oslash \sigma^2(x_j) + \log \big|2\pi \text{diag}(\sigma^2(x_j))\big| \Big). \tag{2.14}$$

where $\epsilon_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $(\cdot)^2$ is element-wise square, $\oslash$ is the element-wise division of vectors[1].

The entropy of a Gaussian distribution with $p(\mathbf{z}) = \mathcal{N}(\mu_z, \Sigma_z)$ can be computed by

$$\begin{aligned}
H(\mathbf{z}) &= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[-\log p(\mathbf{z}) \big] \\
&= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[-\log \frac{\exp\big(-\frac{1}{2}(\mathbf{z}-\mu_z)^T \Sigma_z^{-1} (\mathbf{z}-\mu_z) \big)}{\sqrt{\big|2\pi\Sigma_z\big|}} \big] \\
&= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[\log \sqrt{\big|2\pi\Sigma_z\big|} \; + \; \frac{1}{2}(\mathbf{z}-\mu_z)^T \Sigma_z^{-1} (\mathbf{z}-\mu_z) \big] \\
&= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[\frac{1}{2} \log \big|2\pi\Sigma_z\big| \big] \; + \; \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[\frac{1}{2}(\mathbf{z}-\mu_z)^T \Sigma_z^{-1} (\mathbf{z}-\mu_z) \big] \\
&= \frac{1}{2} \log \big|2\pi\Sigma_z\big| \; + \; \frac{1}{2}.
\end{aligned} \tag{2.15}$$

As a result, conditional entropy $H(\mathbf{z}|\mathbf{x})$ can be analytically computed by

$$\begin{aligned}
H(\mathbf{z}|\mathbf{x}) &\simeq \frac{1}{N} \sum_{i=1}^{N} H(\mathbf{z}|x_i) \\
&= \frac{1}{2N} \sum_{i=1}^{N} \big(\log |2\pi \text{diag}(\sigma^2(x_i))| + 1 \big).
\end{aligned} \tag{2.16}$$

We combine (2.14) and (2.16), and compute the upper bound for the mutual information by

$$I(\mathbf{z}; \mathbf{x}) \leq \frac{1}{2KN^2} \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{j=1}^{N} \Big(\big(\mu(x_j) - \mu(x_i) - \sigma(x_i) \odot \epsilon_k\big)^2 \oslash \sigma^2(x_j) - 1 \Big). \tag{2.17}$$

Finally, we have the following objective for training an AE by using (2.17) in our optimization

[1] As a simplification for $\big(\mu(x_j) - \mu(x_i) - \sigma(x_i) \odot \epsilon_k\big)^T \text{diag}(\sigma^2(x_j))^{-1} \big(\mu(x_j) - \mu(x_i) - \sigma(x_i) \odot \epsilon_k\big)$.

objective in (2.2),

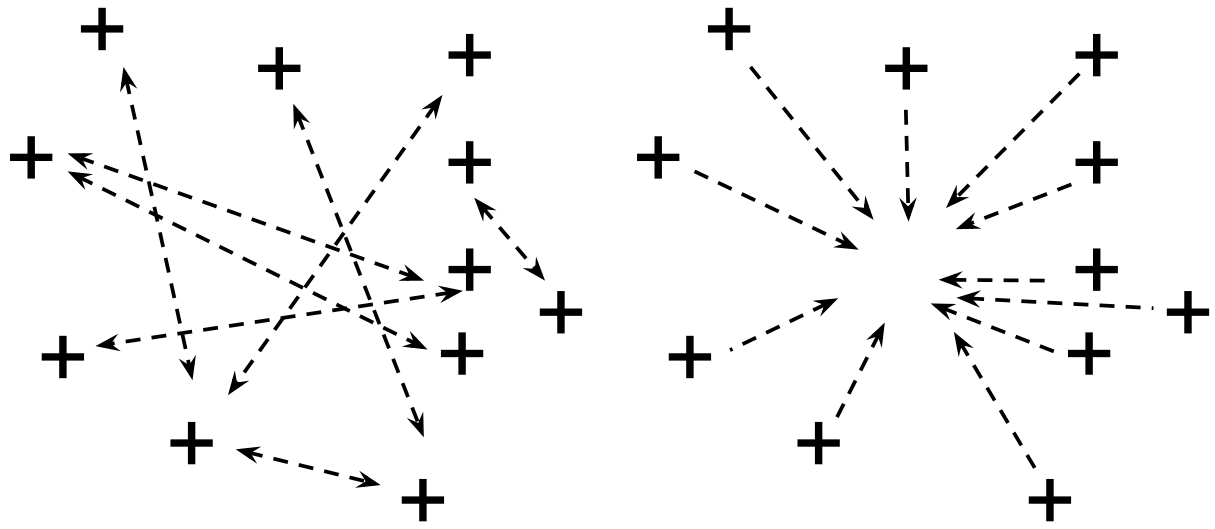
$$\min_{\mu,\sigma,g} \quad \frac{1}{2KN^2}\sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{j=1}^{N}\Big(d(x_i, g(\mu(x_i)+\sigma(x_i)\odot\epsilon_k))+ \\ \beta\left(\mu(x_j)-\mu(x_i)-\sigma(x_i)\odot\epsilon_k\right)^2 \oslash \sigma^2(x_j)\Big). \quad (2.18)$$

As we can see from the formula, minimization of the mutual information term can be considered as reduction of the pairwise distances between samples in the encoding space. The scheme of reducing pairwise distances between samples is depicted by *information potentials* in [74]. By reducing the pairwise distances between samples in the encoding space, the information potential aims to map two different samples from the input space to the same sample in the encoding space. If all samples are mapped to a single point in the encoding space, $I(\mathbf{x};\mathbf{z})$ will be minimized. However, the reconstruction error will be maximized. On the other hand, reduction of the reconstruction error term in (2.18) will map samples to different points in the encoding space, which may result in a trivial solution of identity mapping. Hence, mutual information can serve as a regularization method for AEs. These two errors are trade-offed by the coefficient β.

Although the motivation of VAEs is not based on information theory, the regularization term employed by VAEs has the same form as (2.11), which is a parametric estimation of $H(\mathbf{z})$. As we can see from (2.11), minimization of mutual information in (2.11) can be considered as minimization of the distances between all samples to a center point, which is $\mathbf{0}$, in the encoding space. Thus, from an information theoretic perspective, the difference between VAEs and IPAEs is that, the regularization of IPAEs aims to map any *two* different samples from input space to the same point in the encoding space while the regularization of VAEs is to map *all* samples residing in the input space to the same point in the encoding space. An illustration is given in Figure 2.3.

2.3 Related Work

AEs with Stochastic Encoding: Our proposed objective function is identical to that of Variational Auto-Encoders (VAEs) [36], if $\beta = 1$ and $p(\mathbf{z})$ is assumed to be a pre-defined distribution. However,

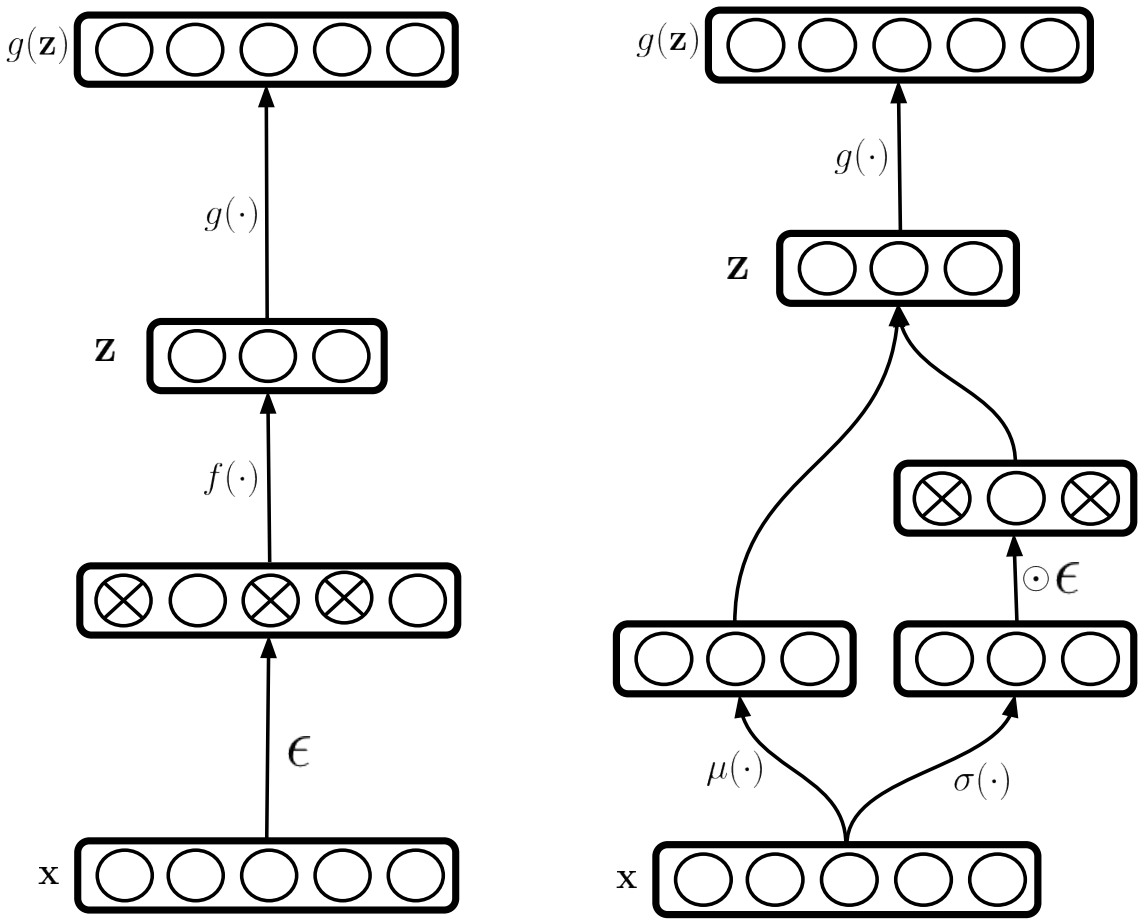


(a) IPAEs. (b) VAEs.

Figure 2.3: (a) The regularization of IPAEs aims to map any *two* different samples residing in the input space to the same point in the encoding space, while (b) the regularization of VAEs is to map *all* samples residing in the input space to the same point in the encoding space. Each + denotes a sample in the encoding space, and each arrow denotes the moving direction of each sample during training.

the motivation for constructing the corresponding objective function is quite different. VAEs are generative models. A common approach used to train VAEs is to maximize the marginal log-likelihood of the observed samples. $p(\mathbf{z})$ utilized in VAEs is a prior distribution that generates $\mathbf{x}$. If the real prior distribution is far way from the assumed prior distribution, then VAEs cannot learn *useful* representations. Denoising Auto-Encoders (DAEs) [75] avoid the trivial solutions by extracting encoding representations from a noise-corrupted input, and aim to reconstruct clean inputs from the extracted encoding representations. A comparison of DAEs and IPAEs from a perspective of noise corruption is given in Figure 2.4. With the denoising criteria, DAEs can learn useful edge detectors from images. In our model, noise is added to the encoding variable as shown in (2.3).

Mutual Information in Representation Learning: Previous works [74, 16, 17] on non-parametric estimation of mutual information reduce the aforementioned problem to estimation of Renyi Entropy. In [74], a Parzen window model was employed for density estimation. In [16, 17], the entropy estimation is based on kernel methods that can avoid explicit estimation of probability



(a) Denoising AEs.

(b) IPAEs.

Figure 2.4: A comparison of Denoising AEs and the proposed IPAEs from a perspective of noise corruption. ϵ means noise. The circles with cross depict corrupted neurons. (a) Denoising AEs add noise to the input variable, while (b) IPAEs add noise to encoding variable.

density. Although non-parametric methods do not make assumptions for the distributions, the disadvantages are also obvious. Non-parametric methods usually have to explore all pairs of samples. Parametric methods can be computationally less complex by making assumptions to probability density functions. For instance, variables were assumed to be drawn from Gaussian distributions in [10]. In addition, analytic solutions of the information bottleneck problem [72] can be found considering affine transformations of variables. In [1], a stochastic encoding method was adopted using a variational method for computation of the bound for the mutual information when variables are transformed in multi-layer deep networks endowed with non-linear activation functions. Our proposed method of estimating mutual information can be considered as a hybrid of parametric (for

$H(\mathbf{z}|\mathbf{x})$) and non-parametric (for $H(\mathbf{z})$) methods.

Other Regularization Methods for AEs: A well known regularization method used for training AEs is mapping a representation space of encoding variables to a lower dimensional space than that of an input space. Then, the lower-dimensional encoding variable can be considered as an under-complete representation of the input. Unlike the methods that perform regularization using lower dimensionality, sparse coding methods [55] employ a higher dimensionality on the encoding space to extract over-complete representations. Using higher-dimensional encoding variables, sparse coding methods impose sparsity constraints to AEs in order to avoid the trivial solutions. Note that, sparsity constraints cause inhibition of a larger number of hidden units on the output layer of the encoder network. Thus, AEs trained using sparsity constraints produce many zero activation values. As a result, the sparse coding can be considered as an implicit compression method used to avoid trivial solutions. In our proposed method, there is no dimensionality constraint. Winner-Take-All AEs [51] can also be considered as a sparsity regularization method. The sparsity is obtained by keeping the hidden units with top activation values and inhibiting the rest of units. Winner-Take-All scheme can learn Gabor-like filters in convolutional or fully connected AEs.

2.4 Experiments

In this section, we empirically examine the proposed Information Potential Auto-encoder (IPAE) method in comparison with VAEs. For a fair comparison of the results, the coefficient β is also used in VAEs to trade-off regularization and reconstruction, resulting in a similar model suggested in β-VAEs [26]. While generating $\mathbf{z}$ samples, K is set as 1 for both AEs in all experiments.

2.4.1 Toy Dataset

We first conduct experimental analyses on a toy dataset. We generated the toy dataset using a mixture of 25 Gaussian distributions in a two-dimensional space. Each Gaussian has the same covariance matrix $\left[\begin{smallmatrix} 0.1 & 0 \\ 0 & 0.1 \end{smallmatrix}\right]$, but with different means. We generate 200 samples from each Gaussian,

and obtain 5000 samples. The target is to reveal the underlying structure of the data, i.e. to estimate 25 centers of Gaussian distributions. The best result is obtained when all samples drawn from the same Gaussian collapse to the same point. If we consider 25 Gaussian distributions as distributions of 25 classes, the best result provides the best information bottleneck trade-off [72] by keeping useful information (class membership) and removing noise.

The network architecture, which was used to construct encoder and decoder, is as follows:

$$\mathbf{x} \xrightarrow{\text{relu}} 2048 \rightarrow \begin{cases} \mu(\mathbf{x}) \\ \sigma(\mathbf{x}) \end{cases} \dashrightarrow \mathbf{z} \xrightarrow{\text{relu}} 2048 \rightarrow 2$$

where "$\rightarrow$" denotes a fully connected layer, on top of which an activation function was used. The number given after each right arrow denotes the number of output units of the fully connected layer. $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ are both 16-dimensional. "$\dashrightarrow$" denotes a stochastic sampling used according to $\mathcal{N}(\mu(\mathbf{x}), \text{diag}(\sigma^2(\mathbf{x})))$. We use mean square error for a distortion function $d(\cdot, \cdot)$ in our objective (2.18) for this task.

For all trials of this dataset, we use the following training procedure. Adam [37] optimization method is employed for training. The initial learning rate is set as 0.001. Batch size is 512. The training is finished after 5000 batches.

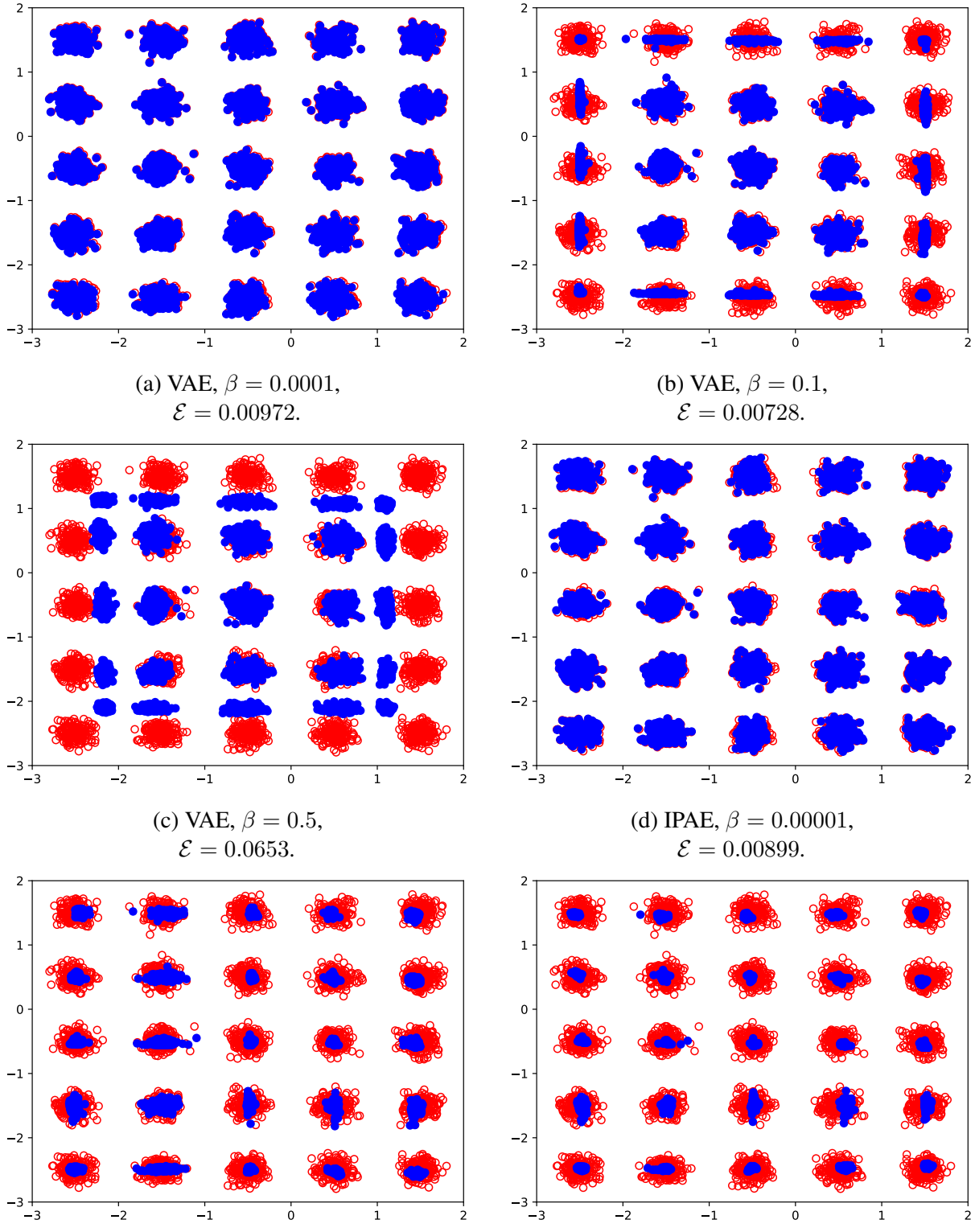
A comparative result obtained for IPAE and VAE is shown in Figure 2.5 and 2.6. In this experiment, the maximal value for j used in the objective (2.18) is set as 1 instead of N. To quantify the *performance* of reconstruction results, we also compute the average euclidean distance (denoted as $\mathcal{E}$ in Figure 2.5) of all reconstructed samples to the means of Gaussian from which they were drawn. The smaller $\mathcal{E}$ is better. As we can see from the figure, with small β value, both AEs failed to find the underlying structure by simply learning an identity mapping between input output (see 2.5a and 2.5d). As a result, $\mathcal{E}$ value is also large. As β increases, both AEs tend to compress different data points from the input space to the same point in the encoding space, and $\mathcal{E}$ value gets smaller, as shown in 2.5b and 2.5e. Further increasing the β value will give a stronger regularization. In the output results (2.5c) of VAEs, the data points start to deviate from the means, and $\mathcal{E}$ also increases.

On the other hand, IPAE can keep the data points around Gaussian means while compressing them (see 2.5f), and thereby IPAE can obtain a minimal $\mathcal{E}$ value. The above results are obtained using the mixture of Gaussian distributions each of which has the same variance. We also construct a mixture of Gaussian with different variances, and the reconstruction results are shown in Figure 2.7. We use a larger dimension of encoding space, $\dim(\mathbf{z}) = 1024$, in the AEs compared with the previous ones. As we can see, our IPAE is robust to change of variance of the mixture of Gaussians. On the other hand, VAEs fail to find the centers. The different results obtained by IPAE and VAEs can be interpreted by different mutual information regularization methods used by two methods. As shown in Section 2.2.3, during the learning phase, VAE aims to map all samples to the same point in the encoding space due to employment of parametric mutual information regularization. On the other hand, IPAE aims to map any two samples in the input space to the same point in the encoding space, which gives more degree of freedom to learn the distribution of encoding features. Thus, for data drawn from a complex distribution, such as the proposed mixture of 25 Gaussian distributions, IPAE learns a better data structure compared to VAEs.

The main additional computational cost of IPAE compared to VAEs comes from maximal values of j (notated as N_j in the following) used in the objective (2.18). N_j is the number of samples that are used to approximate $p(\mathbf{z})$ in our method. By setting $N_j = N$, computational cost can be increased. However, our comparative experiments show that $N_j = 1$ works just fine. The reconstruction results and time consumption for other N_j values are given in Figure 2.8. Note that in the inference phase, our model is as fast as a VAE model, since N_j is no longer required.

2.4.2 Experimental Analyses on the MNIST Dataset

In the next experiment, we take a subset of the MNIST [46] to evaluate our method. We select 18000 training samples belonging to three classes of digits 1, 3 and 4. The network architecture, which



(a) VAE, $\beta = 0.0001$, $\mathcal{E} = 0.00972$.

(b) VAE, $\beta = 0.1$, $\mathcal{E} = 0.00728$.

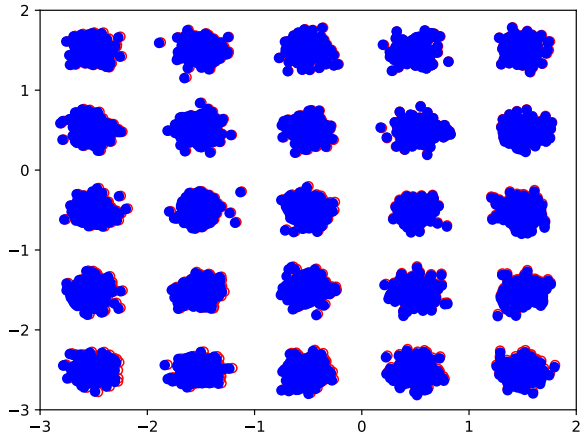
(c) VAE, $\beta = 0.5$, $\mathcal{E} = 0.0653$.

(d) IPAE, $\beta = 0.00001$, $\mathcal{E} = 0.00899$.

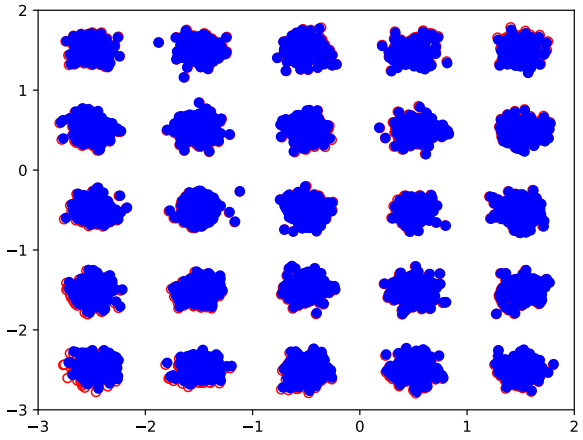
(e) IPAE, $\beta = 0.0005$, $\mathcal{E} = 0.00234$.

(f) IPAE, $\beta = 0.001$, $\mathcal{E} = 0.00204$.

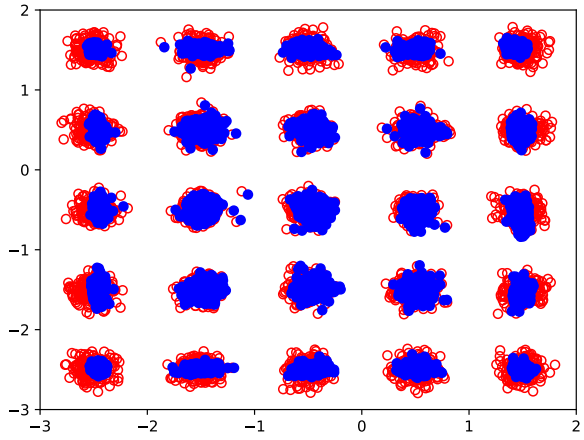
Figure 2.5: Reconstruction results obtained using VAE and IPAE for different β. In addition to the values illustrated in the figure, other values of β are also used to train VAEs (see Figure 2.6), but cannot obtain better results than that of the IPAE for $\beta = 0.001$. The red circles denote input data and the blue dots denote reconstructed data.



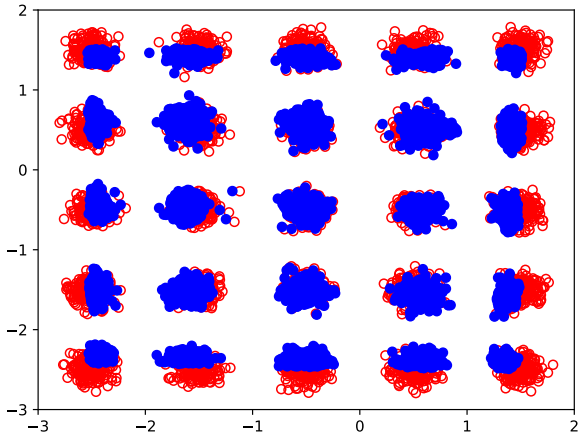
(a) VAE, $\beta = 0.0005$, $\mathcal{E} = 0.00987$.



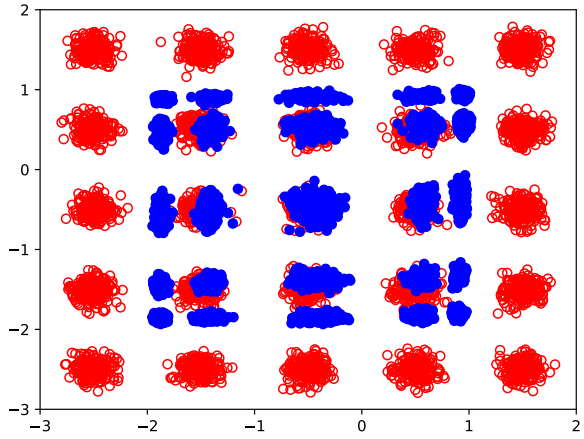
(b) VAE, $\beta = 0.001$, $\mathcal{E} = 0.009501$.



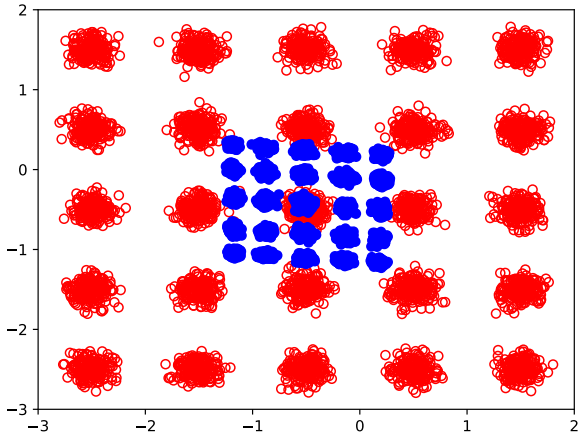
(c) VAE, $\beta = 0.01$, $\mathcal{E} = 0.006373$.



(d) VAE, $\beta = 0.25$, $\mathcal{E} = 0.013488$.

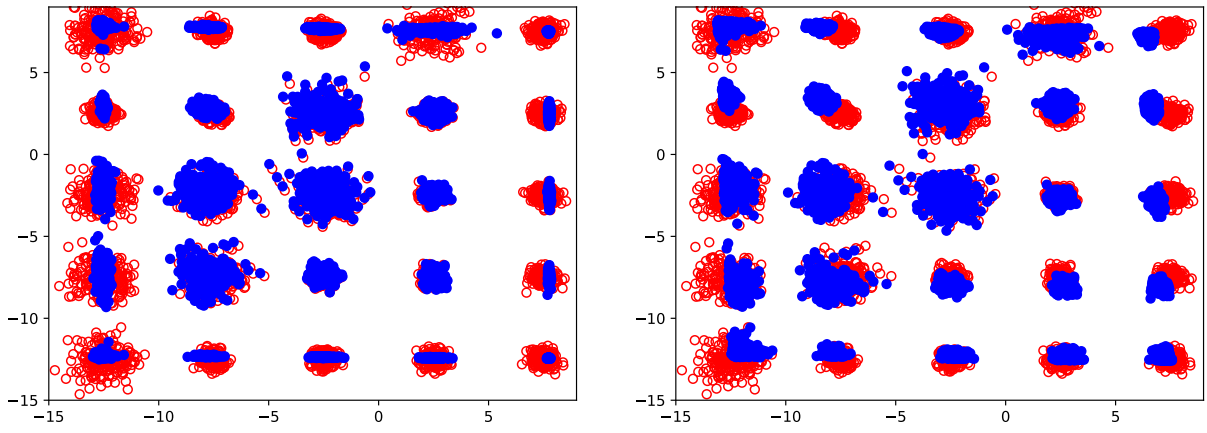


(e) VAE, $\beta = 0.75$, $\mathcal{E} = 0.159700$.



(f) VAE, $\beta = 1.5$, $\mathcal{E} = 0.860975$.

Figure 2.6: Reconstruction results obtained using VAEs for other values of β.



(a) VAE, $\beta = 0.01$, $\mathcal{E} = 0.190056$.

(b) VAE, $\beta = 0.05$, $\mathcal{E} = 0.413764$.

(c) VAE, $\beta = 0.1$, $\mathcal{E} = 1.190412$.

(d) IPAE, $\beta = 0.001$, $\mathcal{E} = 0.087569$.

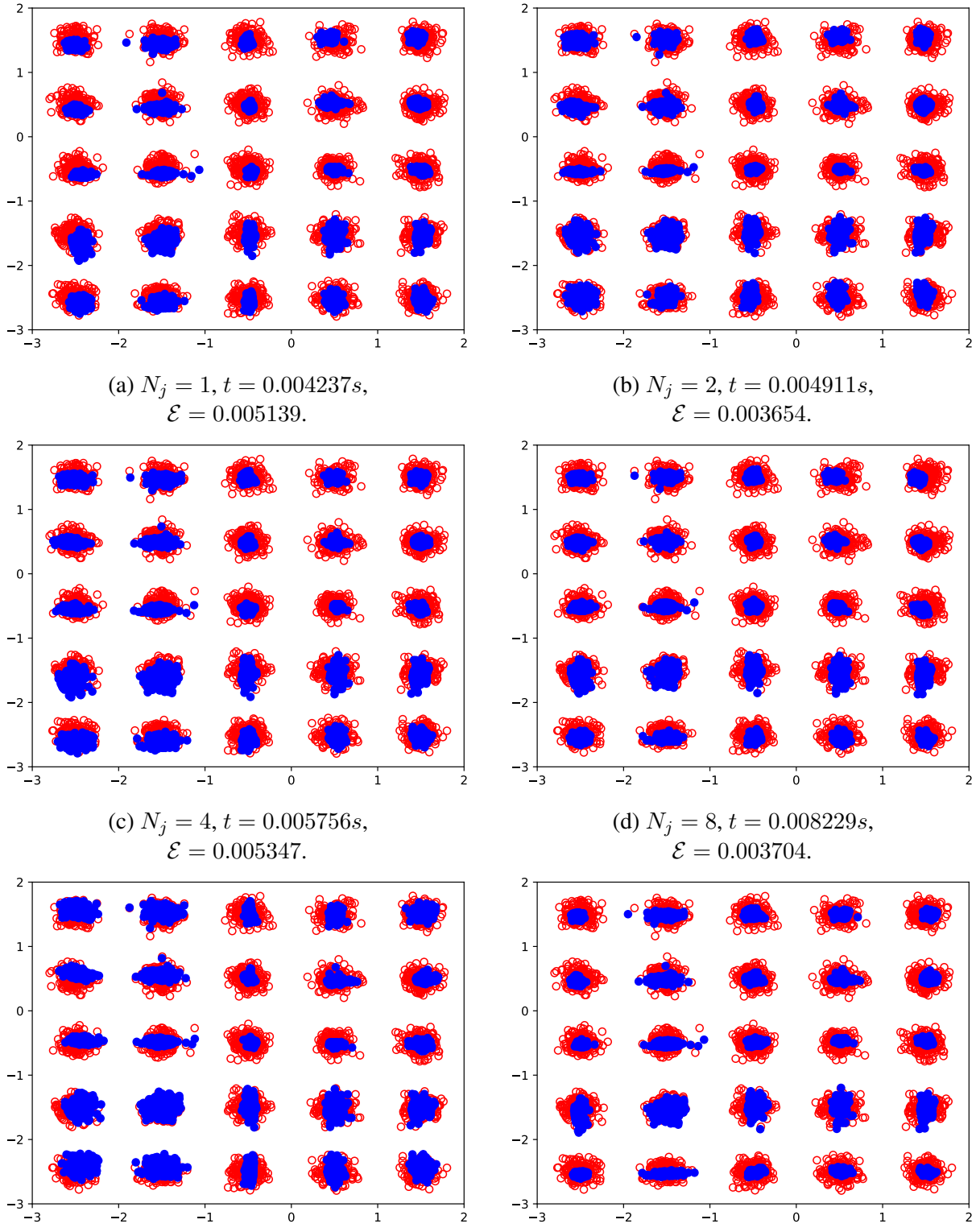
Figure 2.7: Reconstruction results obtained using different variance for each Gaussian distribution.

was used to construct an encoder and a decoder, is as follows:

$$\mathbf{x} \xrightarrow{\text{sigmoid}} 1024 \to \begin{cases} \mu(\mathbf{x}) \\ \sigma(\mathbf{x}) \end{cases} \dashrightarrow \mathbf{z} \xrightarrow{\text{sigmoid}} 1024 \to 784$$

where $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ are both 8-dimensional. In this task, the distortion function $d(\cdot, \cdot)$ used in our objective (2.18) is

$$d(\mathbf{x}, \tilde{\mathbf{x}}) = -\sum_{i=1}^{M} (\mathbf{x}^{(i)} \log \tilde{\mathbf{x}}^{(i)} + (1 - \mathbf{x}^{(i)}) \log(1 - \tilde{\mathbf{x}}^{(i)}))$$



(a) $N_j = 1, t = 0.004237s$, $\mathcal{E} = 0.005139$.

(b) $N_j = 2, t = 0.004911s$, $\mathcal{E} = 0.003654$.

(c) $N_j = 4, t = 0.005756s$, $\mathcal{E} = 0.005347$.

(d) $N_j = 8, t = 0.008229s$, $\mathcal{E} = 0.003704$.

(e) $N_j = 16, t = 0.011515s$, $\mathcal{E} = 0.005605$.

(f) $N_j = 32, t = 0.019438s$, $\mathcal{E} = 0.003131$.

Figure 2.8: Reconstruction results obtained using IPAE for $\beta = 0.0002$ and different N_j. t is the running time spent per mini batch during training.

where $\mathbf{x}^{(i)}$ denotes the i-th dimension of vector, and $\tilde{\mathbf{x}}$ is the reconstructed vector. This distortion used in VAE is interpreted as a Bernoulli distribution of $p(\mathbf{x}|\mathbf{z})$. The same learning procedure as that of the toy dataset is adopted. We visualize the distribution of encoding variable $\mathbf{z}$ of test samples by projecting it to 2-dimensional space using PCA. The results are shown in Figure 2.9, together with test classification error (%) obtained by training (without fine-tuning) a linear SVM on the learned encoding features. The experiments are repeated 10 times and average error is given (AEs are also re-trained at each time).

As we can see from the results, the proposed IPAEs provide better classification performance than VAEs. The best results obtained by IPAEs using $\beta = 0.00001$, $N_j = 8$ (see Figure 2.9g) outperform VAEs that use $\beta = 0.001$ (see Figure 2.9a). Note that, VAEs which are trained using $\beta \leq 0.0001$, failed in some experiments by obtaining too large $\sigma^2(\mathbf{z})$. In terms of visualization, the inter-class distance between samples tends to be larger, i.e. more linearly discriminative using smaller β values, since the data is less compressed. Thus the classification error gets lower. As we increase the values of β, the samples belonging to different categories are contracted, i.e. samples belonging to different categories are mapped to the same point (see e.g. the first and second row of the figure). It causes the samples in the encoding space become less discriminative using both VAEs and IPAE. The reason can be that a larger compression degree causes some discriminative features to be removed. As a result, the classification error also increases. Also, if we increase the N_j value in the IPAE model in order to increase the interaction between more pairs of samples, then the samples belonging to the same category have a more compact distribution, meanwhile the error can be almost unchanged. Therefore, the inter-class discrimination is preserved (see Figure 2.9g, 2.9h, 2.9i compared with Figure 2.9d). Without employment of pairwise interaction between samples, VAEs obtain a larger error if the compression degree is increased.

2.4.3 Qualitative Results on the MNIST Dataset

Next, we give some qualitative results on the MNIST dataset. We first visualize the filters learned using the proposed IPAE. We train an IPAE with the following structure:

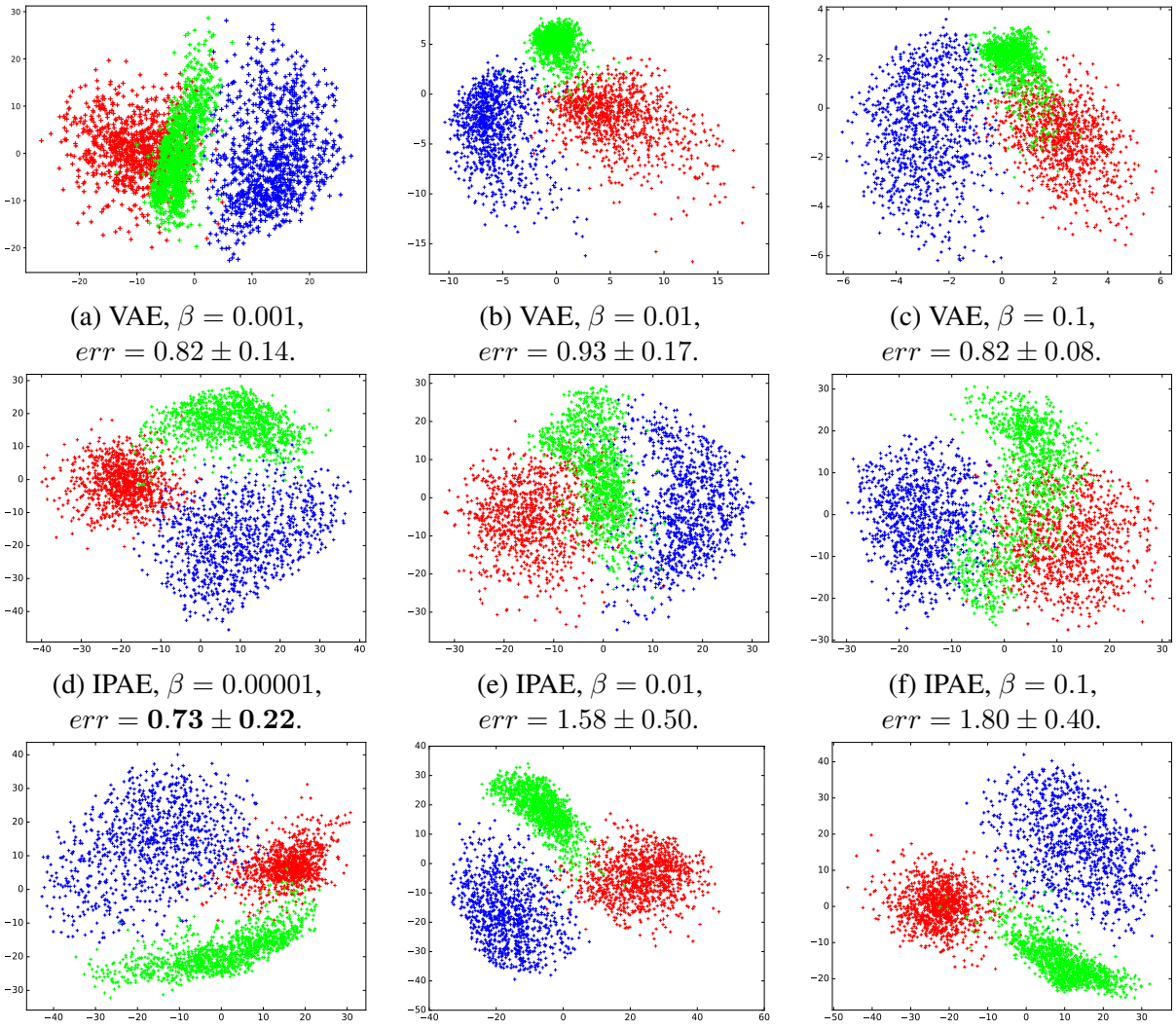
$$\mathbf{x} \xrightarrow{\text{sigmoid}} \begin{cases} \mu(\mathbf{x}) \\ \sigma(\mathbf{x}) \end{cases} \dashrightarrow \mathbf{z} \xrightarrow{\text{sigmoid}} 784$$

The dimension of the encoding variable $\mathbf{z}$ is 1024, thus there are 1024 learned filters. We visualize the learned filters in Figure 2.10. As we can see from Figure 2.10a, without any regularization, most of the learned filters of the AE appear to be noisy filters. With information potential regularization, IPAE shows more filters with shape patterns, e.g. stroke of digits.

We also train VAEs and IPAEs to show a manifold of features extracted from the MNIST dataset on a two dimensional space. The structure used for AEs is the following:

$$\mathbf{x} \xrightarrow{\text{sigmoid}} 512 \xrightarrow{\text{sigmoid}} 512 \begin{cases} \mu(\mathbf{x}) \\ \sigma(\mathbf{x}) \end{cases} \dashrightarrow \mathbf{z} \xrightarrow{\text{sigmoid}} 512 \xrightarrow{\text{sigmoid}} 512 \xrightarrow{\text{sigmoid}} 784$$

where the dimension of $\mathbf{z}$ is 2. Similarly, we use Adam optimizer with initial learning rate 0.001 and train for 10000 steps. We examine the results using different values of the coefficient β. The results are shown in Figure 2.11, 2.12, 2.13 and 2.14, where the left column is the distribution of test samples on the learned two dimensional encoding space. The right column shows the reconstructed samples computed using different values of $\mathbf{z}$. The value of $\mathbf{z}$ is equal to the coordinate of each reconstructed digit on the figure. As we can see from Figure 2.11, with relatively small value of $\beta = 0.01$, the distribution of encoding variable is deviated from the Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. As we strengthen the regularization by increasing the β value, the distribution in the encoding space is closer to the Gaussian distribution. Some of the reconstructed digits appear to be an average of two different digits, since its coordinate is located at a point belonging to a region in an intersection of category regions of two different digits in the encoding space.



(a) VAE, $\beta = 0.001$, $err = 0.82 \pm 0.14$.

(b) VAE, $\beta = 0.01$, $err = 0.93 \pm 0.17$.

(c) VAE, $\beta = 0.1$, $err = 0.82 \pm 0.08$.

(d) IPAE, $\beta = 0.00001$, $err = \mathbf{0.73 \pm 0.22}$.

(e) IPAE, $\beta = 0.01$, $err = 1.58 \pm 0.50$.

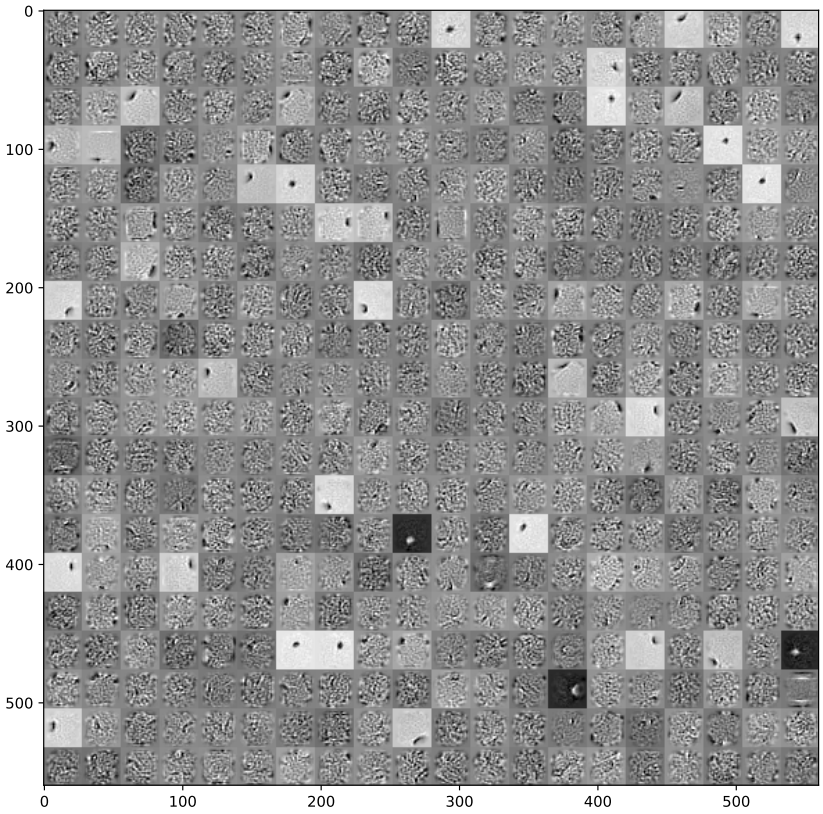
(f) IPAE, $\beta = 0.1$, $err = 1.80 \pm 0.40$.

(g) IPAE, $\beta = 0.00001$, $N_j = 8$, $err = \mathbf{0.70 \pm 0.15}$.

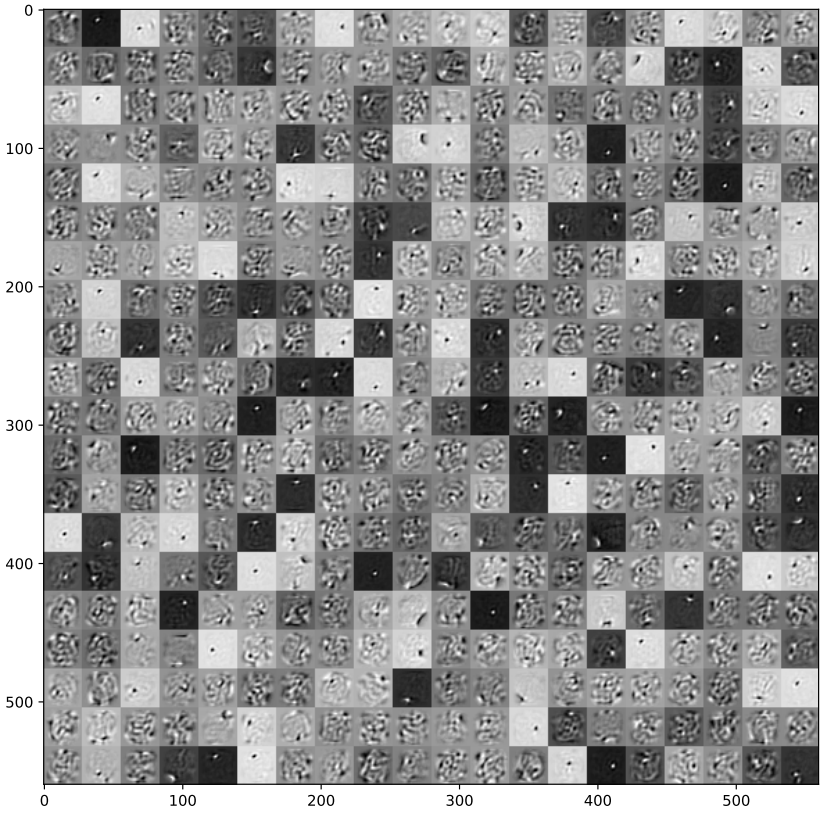
(h) IPAE, $\beta = 0.00001$, $N_j = 32$, $err = 0.80 \pm 0.24$.

(i) IPAE, $\beta = 0.00001$, $N_j = 64$, $err = 0.74 \pm 0.21$.

Figure 2.9: Visualization of encoding features $\mathbf{z}$ on test samples projected to 2-dimensional spaces. In the second row, we use $N_j = 1$. We also train a linear SVM using the learned encoding features (without fine-tuning). The experiments are repeated 10 times (including training of AEs), and average error err (mean$\pm$std error (%)) is given. The best two results are bolded. Note that, for $\beta \leq 0.0001$, VAEs failed in training phases in some experiments due to computation of large $\sigma^2(\mathbf{z})$.



(a) Filters learned using AE without regularization.



(b) Filters learned using IPAE with $\beta = 0.01$ and $N_j = 8$.

Figure 2.10: Filter visualization of IPAE compared with AEs without any regularization. IPAE can learn more filters with meaningful shapes.

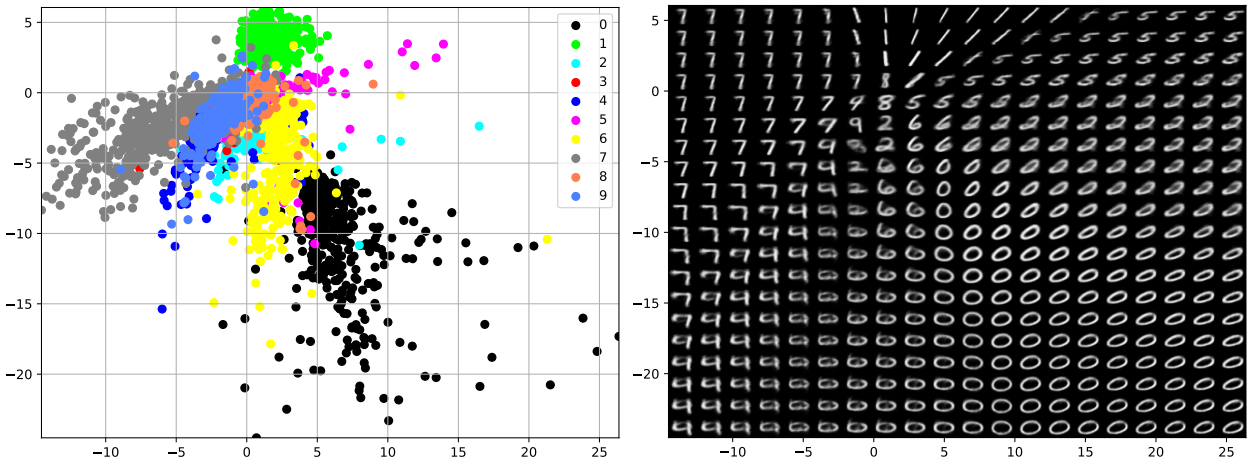
In Figure 2.12, we show the manifold learned by IPAEs with different values of β and $N_j = 1$. As we increase the values of β, the distribution of test samples on the encoding space does not change too much. However, some of the reconstructed digits become vaguer, see e.g. Figure 2.12c. This is because more samples are mapped to the same point in the encoding space as β increases. The reconstructed digits are depicted by an average of multiple digits.

We also investigate the effect of N_j on the results. Thus, the value of β is fixed as 0.001. The results obtained by different N_j are given in Figure 2.13 and 2.14. With larger value of $N_j = 32$, the reconstructed digits are basically similar to smaller ones. Thus, β has more impact on the results compared to N_j in this case.

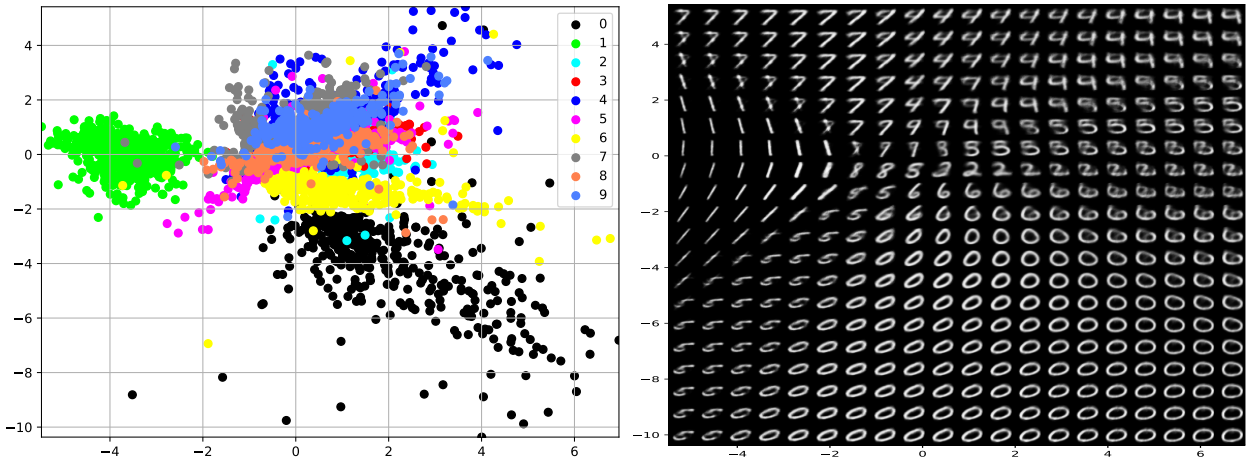
The dimensions of encoding variable also affect the results. In Figure 2.15 and 2.16, we illustrate the digits generated by decoders by setting random values to $\mathbf{z}$ with different dimensions. As is shown in the figures, with relatively less dimension (e.g. $\dim(\mathbf{z}) \leq 2$), the generated digits appear to be monotonous for both VAE and IPAE. Digits belonging to the same category show similar appearance. With larger dimension, the generated digits show more diversity for both VAE and IPAE. In this case, digits belonging to the same category show different writing styles. As we mentioned in Section 2.2.1, IPAE and VAE can be considered as a lossy data compression model, where the degree of compression is quantified by the mutual information $I(\mathbf{x}; \mathbf{z})$. The results given in the figures indicate that using the same regularization coefficient β, a smaller dimension of encoding variable $\mathbf{z}$ will give a stronger compression resulting in removal of redundant information content, such as the writing style. In the digit classification task, writing style contains redundant information which has no contribution to the classification.

2.5 Conclusions

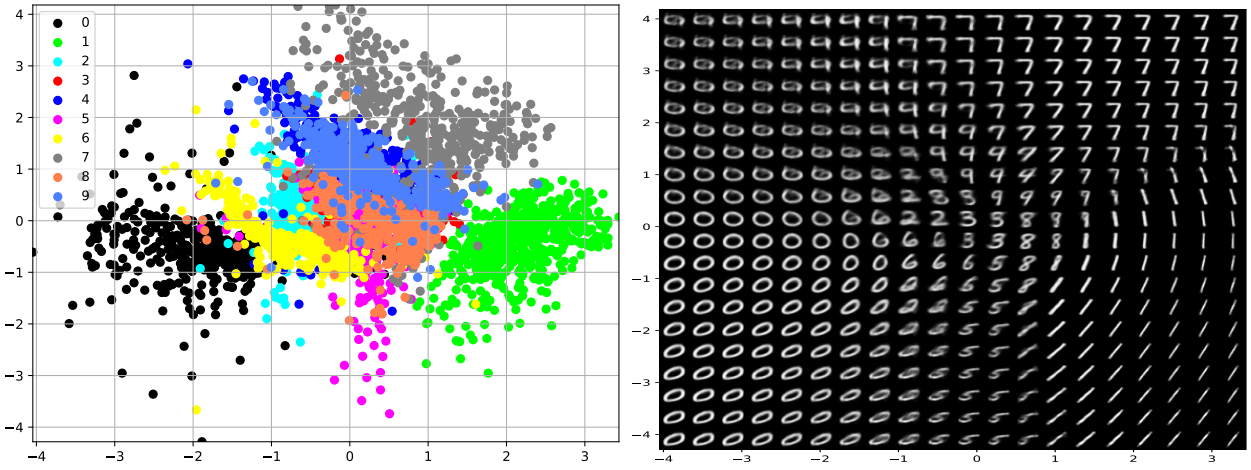
In this chapter, we introduce an information theoretic framework for training of Auto-Encoders. In order to estimate mutual information, we propose a non-parametric method that estimates the entropy of an encoding variable $H(\mathbf{z})$. We also give an information theoretic view of VAEs, which suggests that VAEs can be seen as parametric methods that estimate entropy. Experimental results



(a) $\beta = 0.01$.

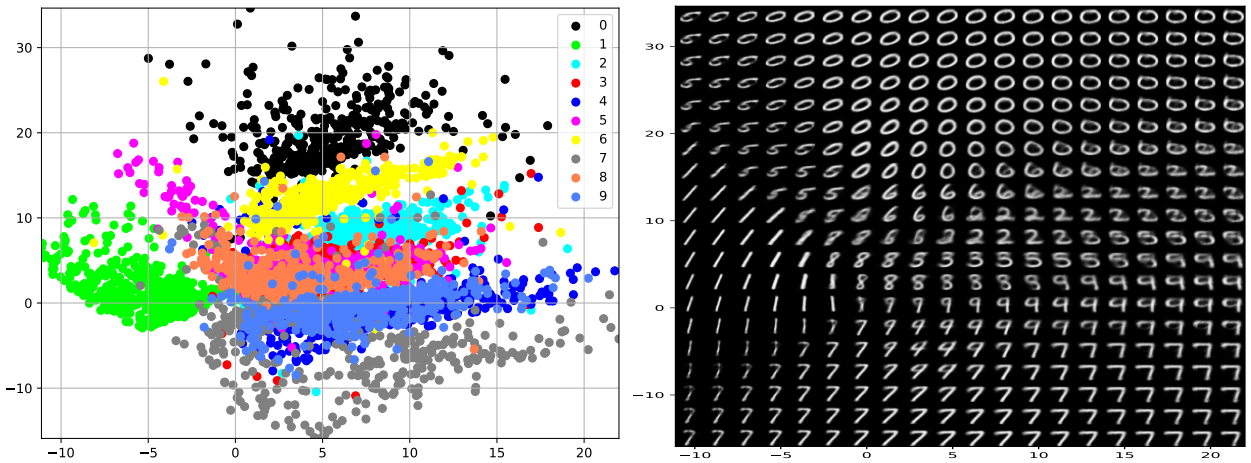


(b) $\beta = 0.1$.

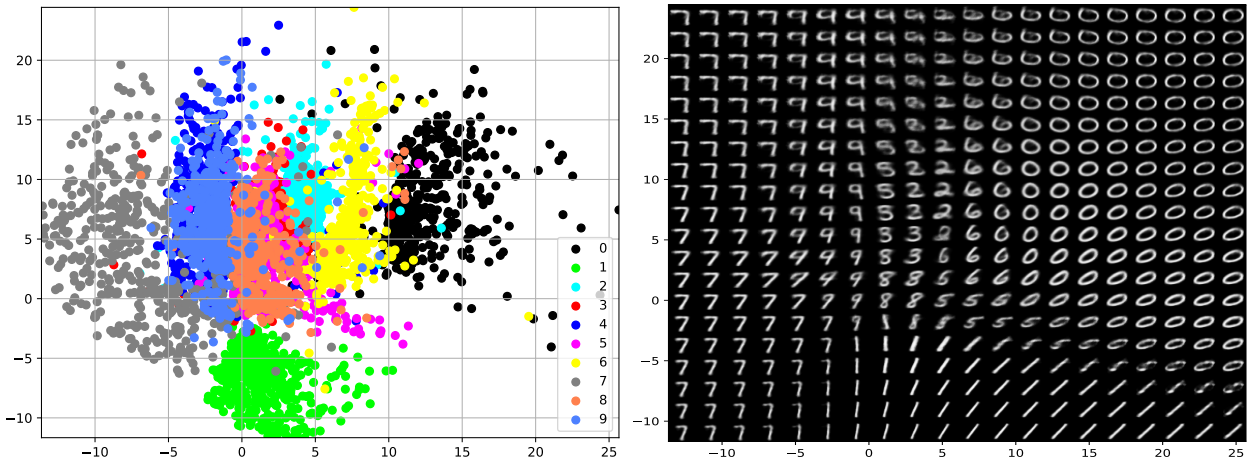


(c) $\beta = 1$.

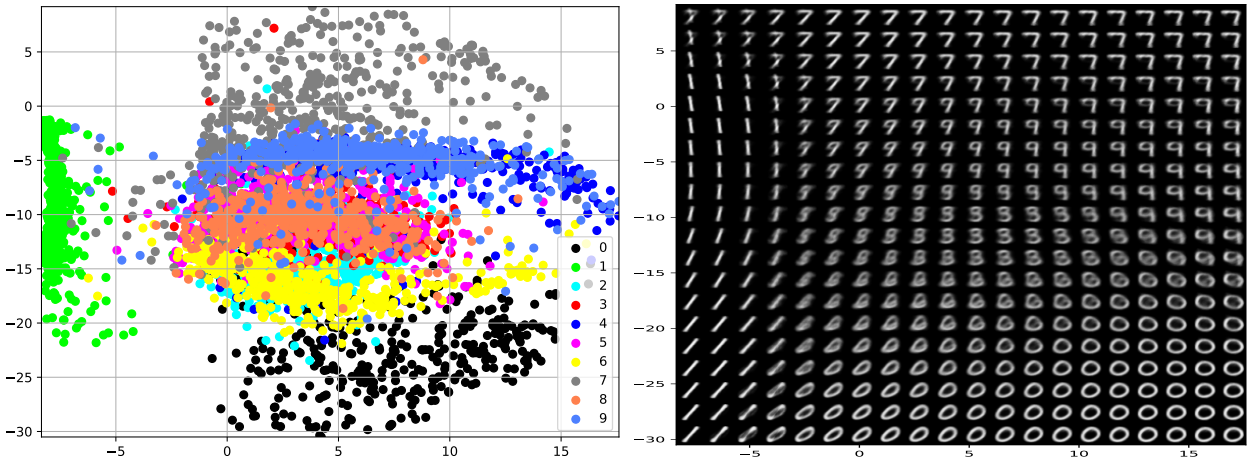
Figure 2.11: Manifold of learned features extracted from MNIST images using VAEs with different values of β. Left; distribution of test samples on the two dimensional encoding space. Right; samples reconstructed using different values of $\mathbf{z}$. The value of $\mathbf{z}$ is equal to the coordinate of each reconstructed digit on the figure.



(a) $\beta = 0.01$, $N_j = 1$.

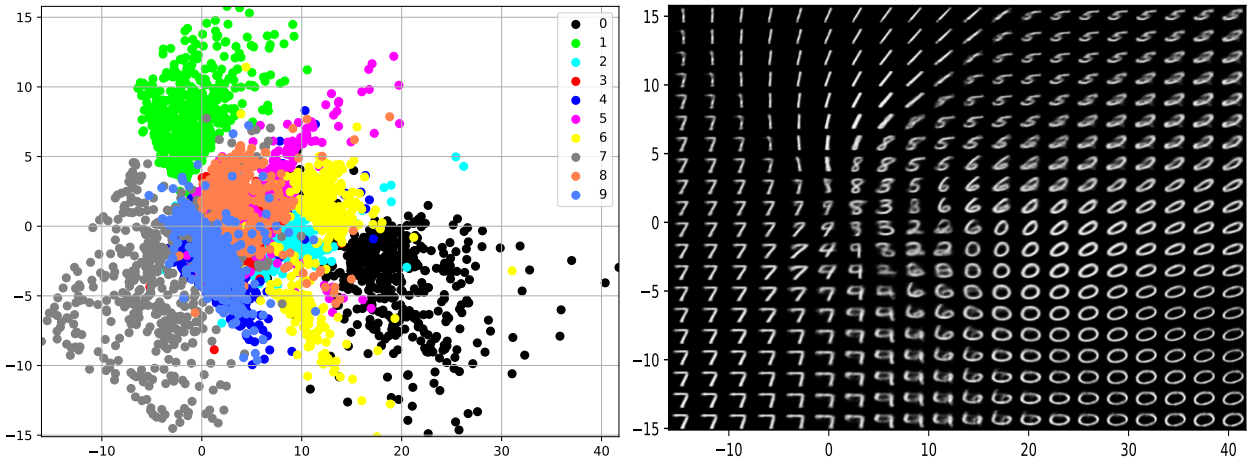


(b) $\beta = 0.1$, $N_j = 1$.

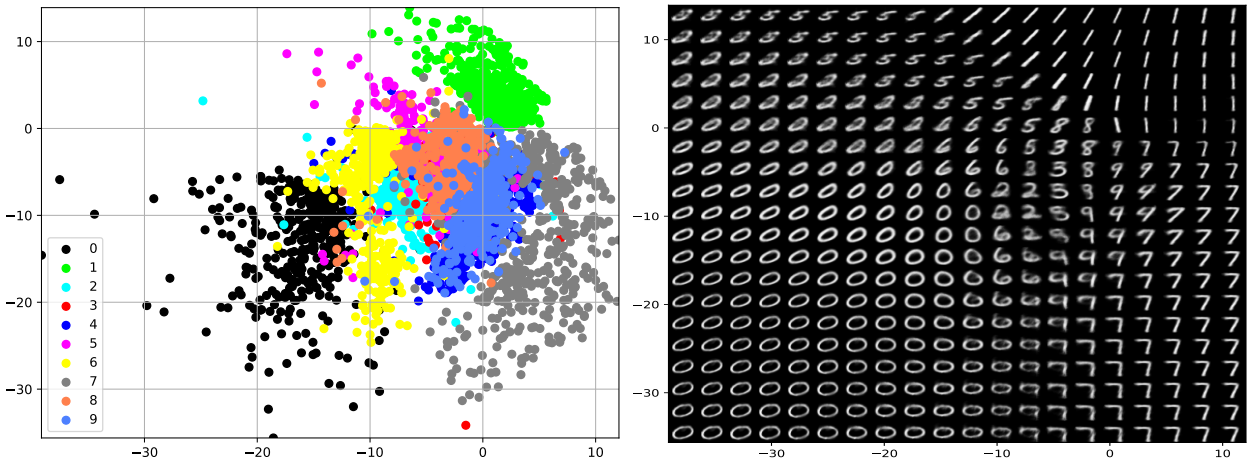


(c) $\beta = 1$, $N_j = 1$.

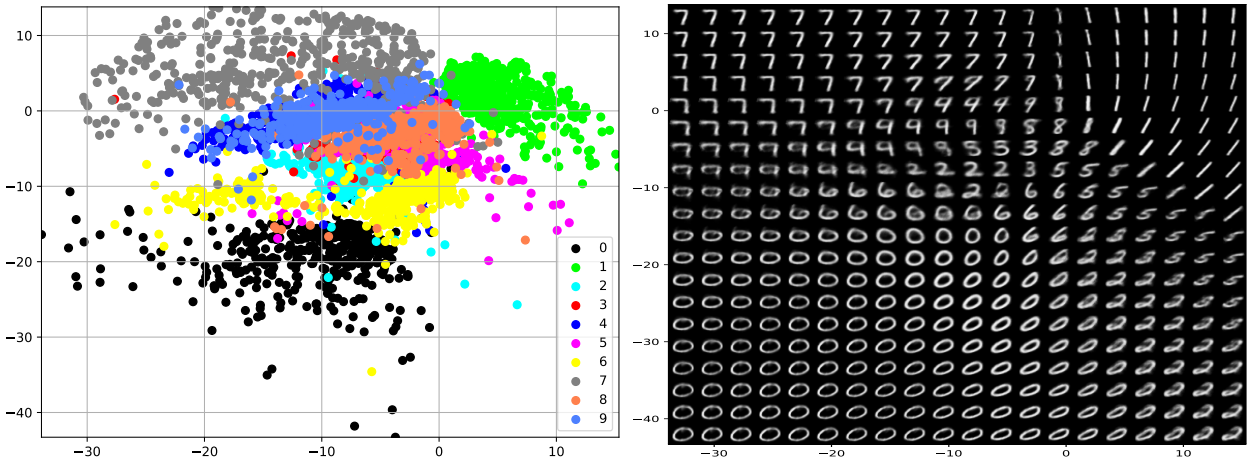
Figure 2.12: Manifold of learned features extracted from MNIST images using IPAEs with different values of β and $N_j = 1$. As we increase the values of β, the reconstructed digits are depicted by an average of multiple digits.



(a) $\beta = 0.001$, $N_j = 1$.

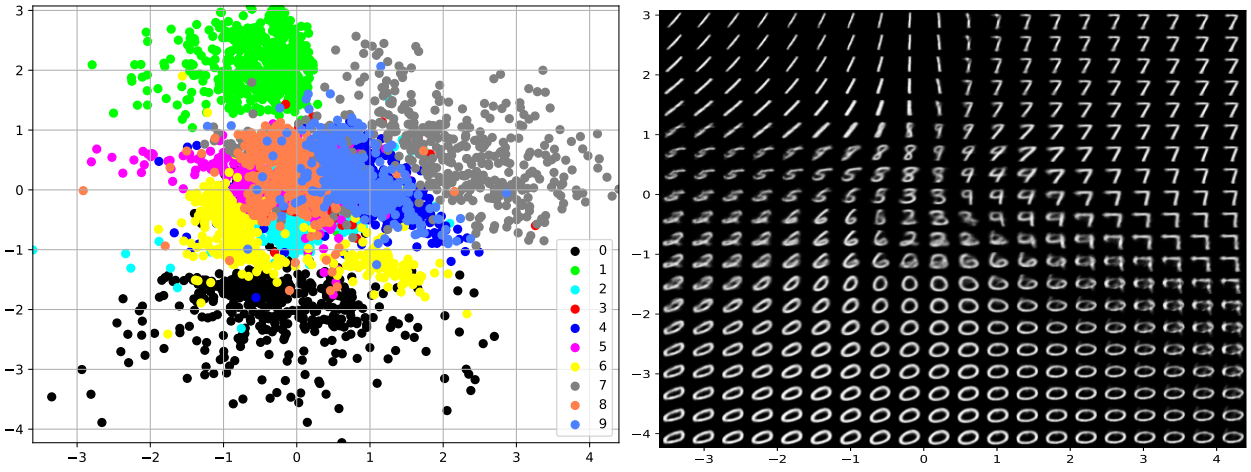


(b) $\beta = 0.001$, $N_j = 2$.

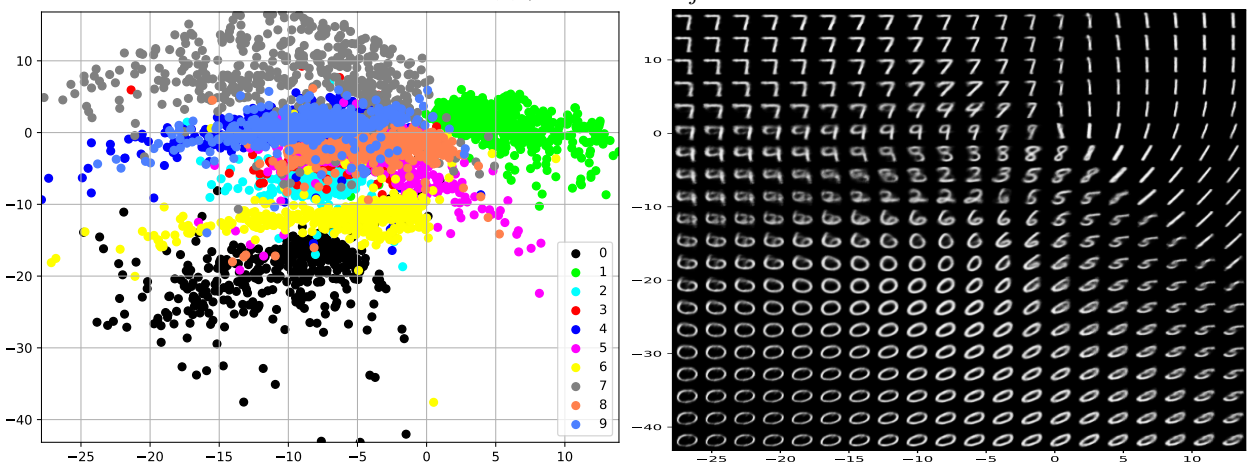


(c) $\beta = 0.001$, $N_j = 4$.

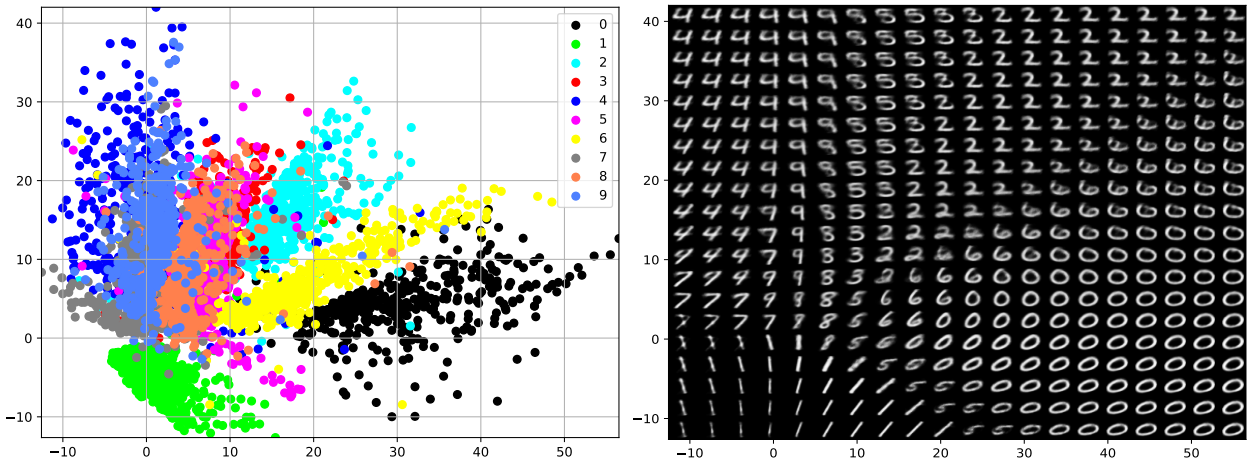
Figure 2.13: Manifold of learned features extracted from MNIST images using IPAEs with different values of N_j and $\beta = 0.001$.



(a) $\beta = 0.001$, $N_j = 8$.

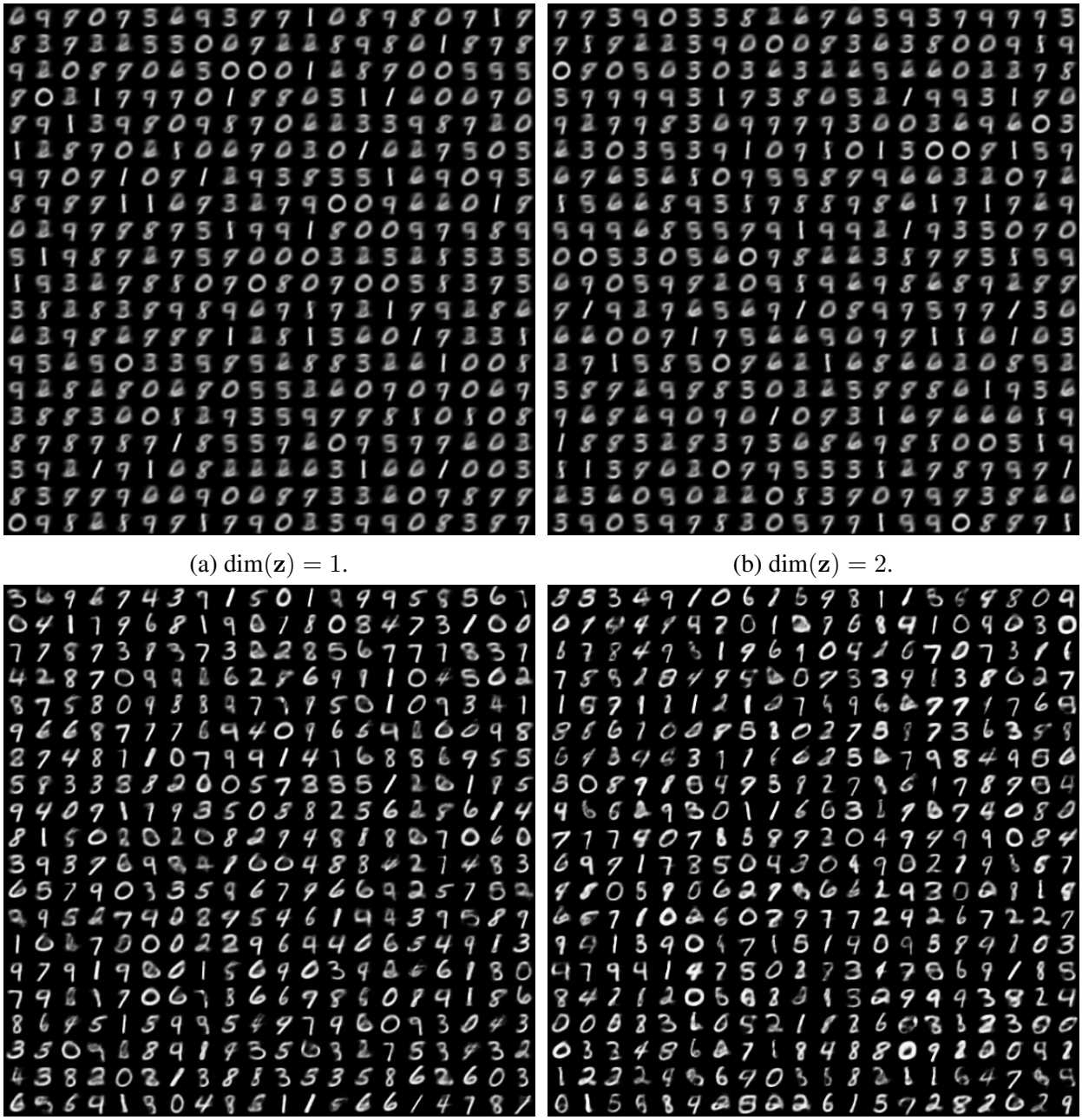


(b) $\beta = 0.001$, $N_j = 16$.



(c) $\beta = 0.001$, $N_j = 32$.

Figure 2.14: Manifold of learned features extracted from MNIST images using IPAEs with different values of N_j and $\beta = 0.001$.



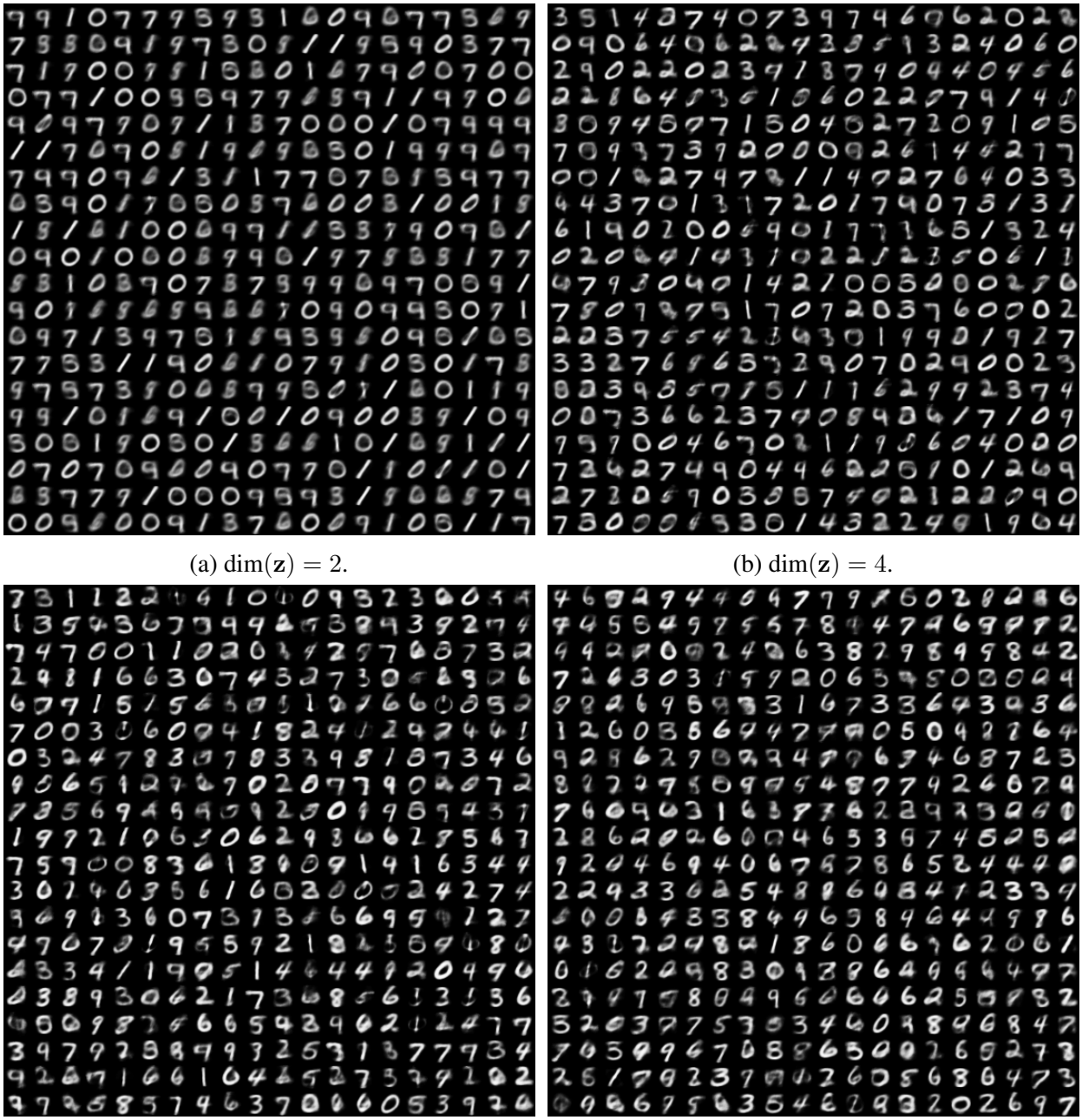
(a) $\dim(\mathbf{z}) = 1$.

(b) $\dim(\mathbf{z}) = 2$.

(c) $\dim(\mathbf{z}) = 8$.

(d) $\dim(\mathbf{z}) = 64$.

Figure 2.15: Digit generalization using VAEs with $\beta = 1$ and using different dimensions of encoding spaces.



(a) $\dim(\mathbf{z}) = 2$. (b) $\dim(\mathbf{z}) = 4$.

(c) $\dim(\mathbf{z}) = 8$. (d) $\dim(\mathbf{z}) = 16$.

Figure 2.16: Digit generalization using IPAE with $\beta = 1$ and $N_j = 1$, and using different dimensions of encoding spaces.

show that the proposed IPAEs have more degree of freedom in terms of representation learning of features drawn from complex distributions such as Mixture of Gaussians, compared to VAEs. In our future work, we will consider application of our proposed non-parametric information regularization method to other type of generative models such as generative adversarial networks [20].

Chapter 3

Truncating Wide Networks using Binary Tree Architectures

Recent study [78] shows that a wide deep network can obtain accuracy comparable to a deeper but narrower network. Compared to narrower and deeper networks, wide networks employ relatively less number of layers and have various important benefits, such that they have less running time on parallel computing devices, and they are less affected by gradient vanishing problems. However, the parameter size of a wide network can be very large due to use of large width of each layer in the network. In order to keep the benefits of wide networks meanwhile improve the parameter size and accuracy trade-off of wide networks, we propose a binary tree architecture to truncate architecture of wide networks by reducing the width of the networks. More precisely, in the proposed architecture, the width is incrementally reduced from lower layers to higher layers in order to increase the expressive capacity of network with a less increase on parameter size. Also, to ease the gradient vanishing problem, features obtained at different layers are concatenated to form the output of our architecture. By employing the proposed architecture on a baseline wide network, we can construct and train a new network with same depth but considerably less number of parameters. Information theoretic analyses also show that the proposed architecture can obtain better optimality of learned representations with a less increase of parameter size compared with baselines. In our experimental analyses, we observe that the proposed architecture enables us to obtain better parameter size and

accuracy trade-off compared to baseline networks using various benchmark image classification datasets. The results show that our model can decrease the classification error of baseline from 20.43% to 19.22% on Cifar-100 using only 28% of parameters that baseline has. Code is available at `https://github.com/ZhangVision/bitnet`.

3.1 Motivation

Recently, Deep Neural Networks (DNNs) have achieved impressive results for many image classification tasks [41, 65, 50, 69, 24, 70, 25, 68]. Various architectures of DNNs have been proposed in order to improve classification accuracy. One approach used to improve accuracy of a DNN is to widen each layer while keeping the depth unchanged. For instance, it is empirically shown in [78] that a wide but relatively shallow DNN can obtain accuracy comparable to a narrow but relatively deep DNN on several classification tasks. There are two crucial benefits of wide-shallow DNNs. First, it usually runs *faster* than narrow-deep DNNs on parallel computing devices, e.g. GPUs, as illustrated in [78]. Also, a deep DNN with many layers may suffer from a *gradient vanishing problem*. Reducing the depth can ease this problem as shown in [29]. However, parameter size of DNNs may significantly increase with respect to improvement of accuracy by widening each layer.

In this chapter, we address a problem of improvement of the parameter size and accuracy trade-off of the aforementioned wide-shallow DNNs. Specifically, given a baseline wide-shallow DNN, we aim to construct and train a new DNN equipped with the following two desirable properties:

- The new DNN has a depth not greater than the baseline wide-shallow DNN so that it can keep the aforementioned two benefits.

- The new DNN can achieve comparable accuracy using relatively less number of parameters compared to the baseline DNN, or can achieve better accuracy by using the same number of parameters as baseline DNN.

Toward this end, we propose a binary tree architecture for implementation of DNNs with a *better* trade-off. An illustrative comparison of our proposed binary tree architecture and conventional

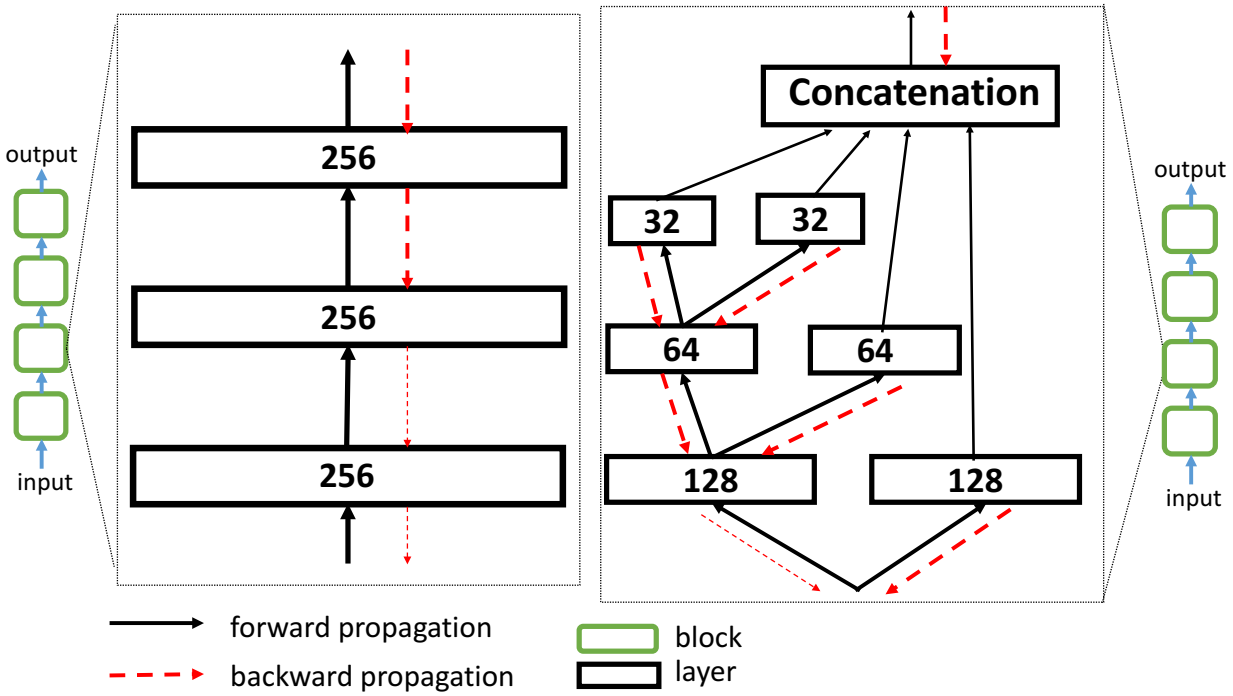


Figure 3.1: **Left**: A conventional architecture [24]. **Right**: Proposed binary tree architecture. Arrows indicate data flow. The numbers depicted in the rectangle denote the width of the layer. Both architectures have depth 3 and output width 256. Our architecture has considerably less number of parameters.

architectures [24, 78] is given in Figure 3.1. In conventional architectures, layers having same width (number of channels) are sequentially stacked. In the proposed binary tree architecture, the width of the k-th layer is $\frac{D}{2^{k-1}}$, where D is the width of the first layer ($k = 1$ at the input layer). Additionally, connections between layers of the proposed architecture are established as connections used in an asymmetric binary tree. At each k-th layer of a binary tree architecture, we have $C_k = \frac{D}{2^{k-1}}$ channels. Then, $\frac{C_k}{2}$ of channels are connected to the channels of the $(k+1)$-st layer. In addition, the remaining $\frac{C_k}{2}$ channels are directly concatenated to form the output of the architecture. Note that our binary tree architecture can be generalized to fully connected layers in which the width becomes the number of neurons used at the layer.

Our **motivation** for employment of the proposed binary tree architecture is twofold. First, we intend to increase the **expressive capacity of DNNs** with a relatively small increase of parameter size. In this chapter, we use the definition of expressive capacity proposed in the previous work [56, 52], where it is defined to be the maximal number of linear regions of (decision) functions computable by the given DNN. As shown in [56, 52], it reflects the complexity of class decision boundary computable by the DNN. Their results state that the maximal number of linear regions of a fully connected feed forward neural network endowed with ReLU [54] activation functions grows exponentially with respect to the depth of the network, and polynomially with respect to the width of the network (i.e., the number of neurons used at each layer of the network). Following this theoretical result, one can increase the expressive capacity with a small increase of the parameter size by simply stacking more layers with *small* width. This leads to the first characteristic structure of our binary tree architecture, which is obtained by incrementally decrease of the width from input layer to higher layers by a factor of 2^{-1}. With this specific structure, the expressive capacity grows with small increase of the parameter size. In our experiments, a binary tree used at convolutional layers of a DNN can increase classification accuracy with a small increase of parameter size of the DNN.

The second motivation for employment of our binary tree architecture is to **ease a vanishing gradient problem observed in DNNs**. While training DNNs, the magnitude of gradient can be cumulatively reduced when it is propagated from higher layers to lower layers. Therefore, the more

layers are used for propagation, the weaker gradient will be obtained at the lower layers, which makes it difficult to train a DNN with many layers [4, 19]. Consequently, the gradient vanishing problem suggests reduction of the depth of a DNN, while increasing the depth may lead to efficient improvement of the expressive capacity as mentioned above. Thus, we need to trade-off between easing the gradient vanishing problem and increasing the expressive capacity, which motivates our second characteristic structure of our proposed binary tree architecture that is obtained by concatenation of features obtained at different layers. With this specific structure, gradients can propagate through short path to lower layers. An illustration is given in Figure 3.1, where flow of gradient propagation during backpropagation is depicted by red dash line. For a better illustration of vanishing gradients, we use thicker red line to show stronger gradient. Our empirical analyses in Section 3.4.4 show that concatenation of features at lower layers can ease the gradient vanishing problem.

Our contributions are summarized as follows:

- We propose a binary tree architecture to improve the trade-off between parameter size and classification accuracy of given baseline wide-shallow DNNs. Meanwhile, the depth of baseline wide-shallow DNNs is not increased to keep the benefit of running speed on parallel computing devices.

- Our experimental results show that, on the Cifar datasets, one can construct a DNN using the proposed binary tree architecture to achieve better accuracy but using considerably less number of parameters. On the Cifar-100 dataset, our models can outperform corresponding baselines by using only approximately 50% of baseline's parameter size. One of our models decreases the classification error of baseline from 20.43% to 19.22% by using only 28% of parameters that baseline has. On ILSVRC12 task, we construct and train two DNNs using the proposed binary tree architecture which also provide better parameter size and classification accuracy trade-off than baseline models.

- We also provide a theoretical analysis of the expressive capacity of DNNs endowed with our proposed binary tree architecture as a function of its depth and width. The theoretical results

indicate that expressive capacity of DNNs endowed with our proposed binary tree architecture can grow with small increase of the parameter size.

- We give an information theoretic analysis on the optimality of learned representations. Results show that the proposed architecture can obtain better optimality of learned representations with a less increase of parameter size compared with baselines.

The rest of this chapter is organized as follows. The second section of this chapter provides the related work. In section 3.3, we introduce our binary tree architecture. Experimental results and analyses are given in section 3.4. Section 3.5 concludes the chapter.

3.2 Related Work

Recently, various architectures of convolutional neural networks (CNNs) have been proposed [41, 50, 69, 65, 70, 68]. In [30], connections between random forest [9] and CNNs are investigated. Inspired by random forest, they embedded routing functions to CNNs and obtain Conditional CNNs. As shown in their experiments, Conditional CNNs with highly branched tree architectures can improve the accuracy-efficiency trade-off. Conditional CNNs can be considered as symmetric full tree architectures. On the other hand, we use an asymmetric tree architecture and concatenate features from different layers. Another related work, a *fractal architecture* used by FractalNet was proposed in [45]. The relationship between FractalNets and our BitNets is as follows. Fractal architecture can also be considered as a tree architecture. And they both try to ease the gradient vanishing problem by joining lower-layer features to higher-layer features. The difference is that BitNets aim to reduce the parameter size by incrementally reducing the width as depth increases, while FractalNets do not. As a result, FractalNets have more number of parameters compared to BitNets for the same depth. The motivation for development of BitNets considers the expressive capacity which is not investigated in FractalNets. Experimental results on benchmark datasets also show that BitNets outperform FractalNets by using less number of parameters.

Previous work [32, 39, 48] also explored the tree structure in the context of deep neural networks.

All these related work and our work enable an end-to-end training of tree structure in DNNs. Our method is different from them in the following aspects. In [39], a decision forest classifier was introduced to replace a classifier used at the last classification layer (usually Softmax) of a baseline DNN; other layers, e.g., conv. layers, are left untouched. Thus, the parameter size is not reduced unlike our method. In [32], the encoding and decoding functions of Auto-Encoders were implemented using 'soft' decision trees. Their motivation is to recursively direct samples from parent nodes to all its child nodes according to probabilities computed by gating functions, and the output of tree is the average of output of all leaves weighted by gating values. In [48], a tree structure was used in pooling layer, the region being pooled was fed into the leaves of tree structure and results were obtained at the root of tree structure. Therefore, they employ a tree architecture different from (in an inverse way to) our tree structure.

As shown in [33, 80], the parameter size of a trained CNN can be reduced by constructing a new CNN with less redundancy of weights. However, these methods may cause a drop on accuracy after a compression of model. With our architecture, we can boost the accuracy with less number of parameters.

In [24], a residual architecture is proposed to construct CNNs (ResNets). ResNets enable us to train considerably deeper CNNs. The deepest ResNets employed for classification using the ILSVRC12 [59] and Cifar datasets have 200 and 1000 layers, respectively [24, 25], and achieved impressive performance. However, a deep ResNet with many layers may suffer from two problems. First, the running speed on parallel computing devices may be slow compared to the speed of shallower ones. Thus, it takes more time to train a very deep ResNet. Also, the gradient vanishing problem [4, 19] may also be observed on a very deep ResNet with many layers. To address these problems, a novel training procedure called stochastic depth is introduced in [29]. It enables one to train shallow ResNets during training and use deep ResNets for testing. Shallow ResNets can ease the gradient vanishing problem during training, and reduce the training time. Their experimental results show that their proposed training procedure can improve the test accuracy of baseline ResNets with constant depth. This indicates easing the gradient vanishing problem is crucial to train a very deep ResNet. However, during the test time, the depth of ResNets is the same as the depth of

baseline which makes inference speed slower.

Intuitively, a simpler way to avoid gradient vanishing problem and accelerate the running speed during training and inference is to use a shallow network. In [78], Zagoruyko and Komadakis use a shallow ResNet, and increase the width to make the expressive capacity comparable with narrow-deep ones. Interestingly, they observe that wide-shallow CNNs can outperform its deeper-narrower peer CNNs which have the same parameter size on the Cifar-10/100 classification datasets. On the ILSVRC12 classification task, wide ResNets can also obtain comparable accuracy with smaller depth. Their results draw our attention to consider the wide-shallow DNNs. In their method, the width of each layer is symmetrically increased by the same factor. This will significantly increase the parameter size. Therefore, in this work, the proposed binary architecture truncate architecture of wide networks by reducing width of the networks considering their parameter size and accuracy trade-off. Our motivation for employment of binary tree architecture also considers gradient vanishing problem, which has been shown to be crucial for training DNNs in [29].

3.3 Binary Tree Architecture

In this section, we give an overview of conventional blocks used in previous works [24, 78], and introduce our proposed binary tree architecture. Then, we theoretically analyze expressive capacity and parameter size of our proposed binary tree architecture considering its depth and width.

3.3.1 Conventional Blocks

In some CNNs [65, 24, 25, 78], a block is constructed by a stack of K convolutional layers. At each k-th layer, $k = 1, 2, \ldots, K$, we compute

$$\mathbf{X}_k = f_k(\mathbf{X}_{k-1}; \mathcal{W}_k), \tag{3.1}$$

where $\mathbf{X}_0 := \mathbf{X} \in \mathbb{R}^{w \times h \times c}$. $\mathbf{X}$ is an input tensor of features given to the block, $\mathbf{X}_k \in \mathbb{R}^{\frac{w}{s} \times \frac{h}{s} \times D}$ is the tensor of features obtained at the output of k-th layer, c is the number of channels, w is the $width$

and h is the $height$ of a feature map. We assume that the number of convolutional filters used at each layer is D, and down-sampling is performed at the first convolutional layer by stride s. $f_k(\mathbf{X}_k; \mathcal{W}_k)$ is computed by a composition of a convolution operation, batch normalization [31] and nonlinear function σ, where $\mathcal{W}_k$ denotes a set of trainable parameters used at the k-th layer, e.g. the filter weights. The output feature tensor of the block is obtained at the last layer $\mathbf{Z} := \mathbf{X}_K \in \mathbb{R}^{\frac{w}{s} \times \frac{h}{s} \times D}$. In (3.1), we omit shortcut connection and element-wise addition used in [24] to simplify the notation. We refer to the block defined in (3.1) as a *conventional block* (ConvenBlock) with width D and depth K in the following sections. An illustration of ConvenBlock with width 256 and depth 3 is given in Figure 3.1 (left).

3.3.2 Binary Tree Blocks

We define a *binary tree block* (BitBlock) by a binary tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes residing in the block, and $\mathcal{E}$ is the set of edges that connect the nodes. The architecture of a BitBlock is determined by its width D and depth K, where D is the channel number, i.e. number of feature tensors provided by the BitBlock at the output, and K is the depth of the binary tree $\mathcal{T}$, as follows:

- At the root of a BitBlock $\mathcal{T}$ denoted by $v_0 \in \mathcal{V}$, we have a feature tensor denoted by $\mathbf{X}_{0,l} := \mathbf{X} \in \mathbb{R}^{w \times h \times c}$, where $\mathbf{X}$ is given as an input to the block.

- At each k-th layer (level) of a BitBlock, we apply two mapping functions $f_{k,l}$ and $f_{k,r}$ to the input feature tensor $\mathbf{X}_{k-1,l}$. Then, we compute a feature tensor $\mathbf{X}_{k,l}$ having $\frac{D}{2^k}$ channels on a left child node of v_{k-1}, and a feature tensor $\mathbf{X}_{k,r}$ having $\frac{D}{2^k}$ channels on a right child node of v_{k-1} at the k-th layer of $\mathcal{T}$. The feature tensor $\mathbf{X}_{k,l}$ is further fed to the next $(k+1)$-st layer. More precisely, at each k-th level, we compute

$$\begin{aligned} \mathbf{X}_{k,l} &= f_{k,l}(\mathbf{X}_{k-1,l}; \mathcal{W}_{k,l}), \\ \mathbf{X}_{k,r} &= f_{k,r}(\mathbf{X}_{k-1,l}; \mathcal{W}_{k,r}). \end{aligned} \tag{3.2}$$

- Finally, feature tensors computed at all right child nodes at each k-th level of $\mathcal{T}$, and the

feature tensors computed at the left child node at the last K-th level of $\mathcal{T}$ are concatenated across channels to construct the output feature tensor of the BitBlock by

$$\mathbf{Z} = \text{concat}(\mathbf{X}_{1,l} \bullet \mathbf{X}_{2,l} \bullet \cdots \bullet \mathbf{X}_{K,l} \bullet \mathbf{X}_{K,r}). \tag{3.3}$$

The size of concatenated tensor $\mathbf{Z}$ is $(\frac{w}{s} \times \frac{h}{s} \times D)$. In the case of down-sampling, we apply a stride s at the first layer as utilized in ConvenBlocks.

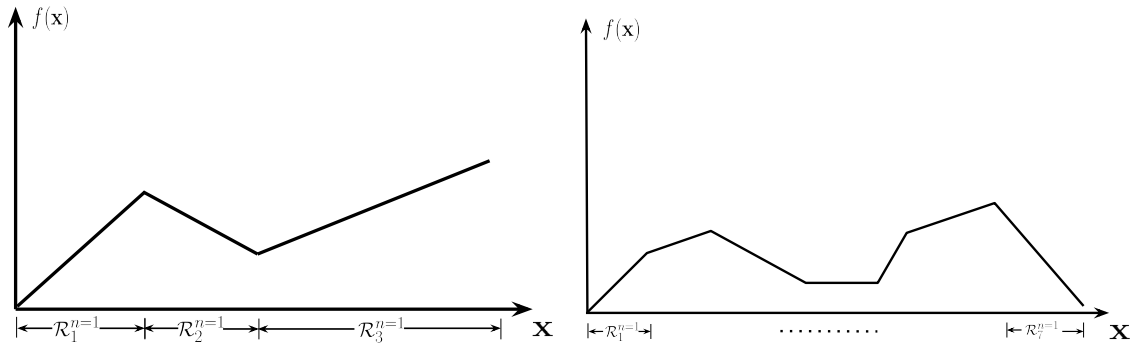
We note that D is divided by 2^K. Moreover, if $K = 1$, then the BitBlock $\mathcal{T}$ reduces to a single convolution layer. Our BitBlocks can also be applied to fully connected layers using a feature tensor $\mathbf{X} \in \mathbb{R}^{1\times1\times C}$. We denote it by fully connected BitBlock. An illustration of proposed BitBlock with width 256 and depth 3 is given in Figure 3.1 (right). We can construct a DNN by stacking multiple BitBlocks. A DNN endowed with BitBlocks is called as a BitNet in this paper.

3.3.3 A Theoretical Analyses of Expressive Capacity of Binary Tree Blocks

In this section, we theoretically analyze expressive capacity and parameter size of the proposed binary tree block with respect to its depth and width when it is used at fully connected layers. We use the definition of expressive capacity proposed in the previous work [56, 52]. Precisely, expressive capacity of a DNN is defined by the maximal number of linear regions of (decision) functions computable by the given DNN. The formal definition of linear regions of a function is given as follows.

Definition 3.3.1 (Linear Region). Given a function $f(\cdot)$ with n-dimensional input space $\mathbb{R}^n$, a linear region $\mathcal{R}^n_k \subseteq \mathbb{R}^n$ of $f(\cdot)$ is a subspace of its input space, such that $f(\cdot)$ computes a linear mapping on that linear region, i.e. $\forall \mathbf{x} \in \mathcal{R}^n_k$, $f(\mathbf{x}) = \mathbf{w}_k\mathbf{x} + \mathbf{b}_k$, and $\forall \mathbf{x} \notin \mathcal{R}^n_k$, $f(\mathbf{x}) \neq \mathbf{w}_k\mathbf{x} + \mathbf{b}_k$.

Consider a decision function $f(\cdot)$ computed by a DNN, then the more number of linear regions $f(\cdot)$ has, the more complex input-output mapping the DNN can compute. An example is illustrated in Figure 3.2. In other words, the DNN can compute more complex decision boundary and solve more complex classification tasks. A multilayer perceptron network implementing a linear activation



(a) (b)

Figure 3.2: There are three linear regions in the function of (a), and seven linear regions in the function of (b). Thus, function given in (b) can compute more complex input-output mapping.

function computes a linear mapping between input and output. Thus, it only has 1 linear region, i.e. the whole input space.

As investigated in [52], DNN can map different linear regions of lower layer space to the same output of higher layer space. As a result, the number of linear regions computed by DNN increases exponentially as the depth increases. The way DNN mapping different linear regions to the same output is similar to the space folding operation as shown in Figure 3.3.

We first briefly review how to implement space folding by DNNs. Let the input space be n-dimensional, and the input variable be $\mathbf{x} \in \mathbb{R}^n$. Then, we can fold each i-th axis of input space into p linear regions at the first layer by the following function:

$$\mathbf{z}_1^{(i)} = \max(w\mathbf{x}^{(i)}, 0) + \sum_{j=1}^{p-1}(-1)^j \max(2w\mathbf{x}^{(i)} - j, 0). \tag{3.4}$$

Figure 3.4b shows an example of folding function with $p = 4$. Note that, (3.4) can be implemented by a DNN using ReLU activation functions. An illustration is given in Figure 3.4a, where the activation values of intermediate layer can be denoted by $\mathbf{z}_1^{(i)}$. As we can see, p neurons are used in the hidden layer to fold an axis into p linear regions. Each axis can be further folded by feeding $\mathbf{z}_1^{(i)}$ to the next layer resulting in more number of linear regions using

$$\mathbf{z}_2^{(i)} = \max(w\mathbf{z}_1^{(i)}, 0) + \sum_{j=1}^{p-1}(-1)^j \max(2w\mathbf{z}_1^{(i)} - j, 0). \tag{3.5}$$

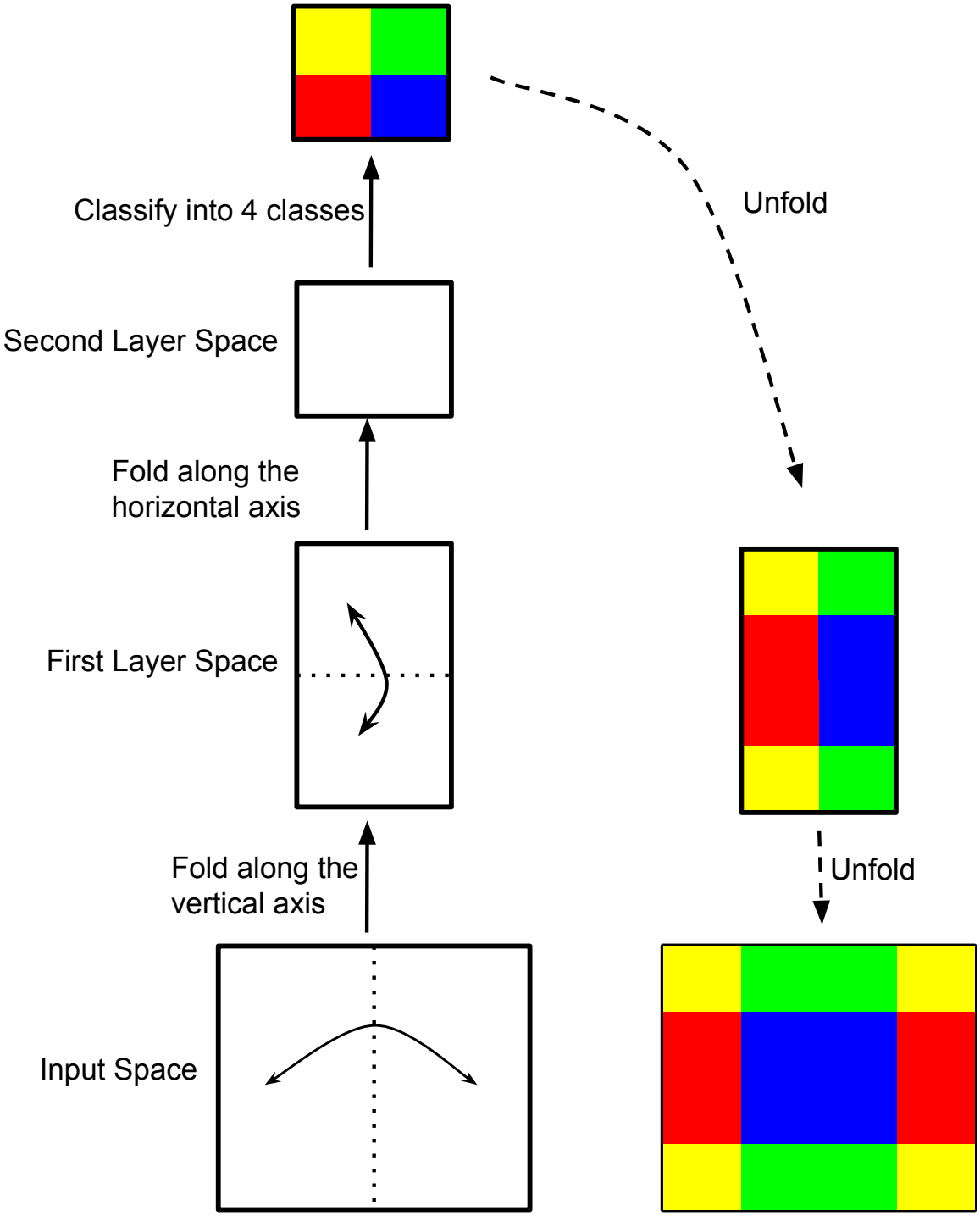


Figure 3.3: An illustration of space folding.

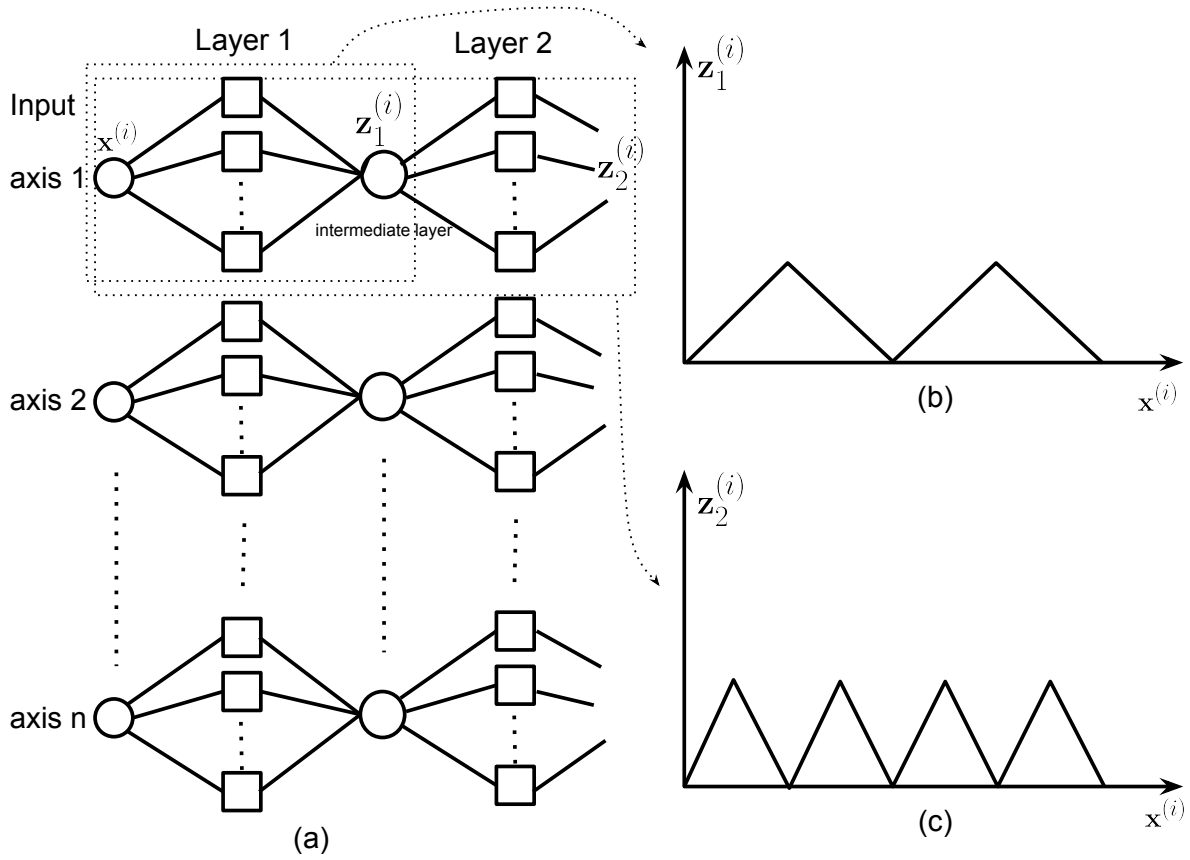


Figure 3.4: Space folding implemented by fully connected DNN. Each circle denotes a neuron without activation function, each rectangle denotes a neuron with ReLU activation function. (b) shows an example of space folding along the first axis by first layer. (c) is the space folding by second layer. As depth increases, each axis can be folded into more number of linear regions.

The folding function computed for mapping $\mathbf{x}^{(i)}$ to $\mathbf{z}_2^{(i)}$ is given in Figure 3.4c. Formally, a DNN with n input neurons and L hidden layers of width D can fold each axis to $(\frac{D}{n})^L$ linear regions. With n axes, the DNN can compute functions that have $\Omega\big((\frac{D}{n})^{nL}\big)$ linear regions [52].

We provide the following two theoretical results regarding i) computation of the parameter size, and ii) computation of the maximal number of linear regions of functions computable by a fully connected conventional network and a BitNet. We first provide the results for conventional networks that can be easily derived using the results given in [52].

Corollary 3.3.2. *Suppose that we are given a fully connected neural network stacked by L fully connected ConvenBlocks each having width D and depth K. The parameter size of the given network is $\Omega\big(LKD^2\big)$. The maximal number of linear regions of functions that can be computed by the given*

network in an n-dimensional ($D \geq n$) input space is lower bounded by $\Omega\big((\frac{D}{n})^{nKL}\big)$. ■

Proof. The total depth of the given network is LK. Thus, the parameter size is $\Omega\big(LKD^2\big)$. Also, according to Corollary 5 of [52], the maximal number of linear region of functions that can be computed by the given network in an n-dimensional ($D > n$) input space is lower bounded by $\Omega\big((\frac{D}{n})^{n(KL-1)}D^n\big)$, which can be simplified by a looser bound $\Omega\big((\frac{D}{n})^{nKL}\big)$.

□

The expressive capacity of our proposed BitNet is given as follows.

Proposition 3.3.3. *Suppose that we are given a BitNet stacked by L fully connected BitBlocks each having width D and depth K. The parameter size of the given BitNet is $\Omega\Big(\frac{4}{3}L\big(1-\frac{1}{4^K}\big)D^2\Big)$. The maximal number of linear regions of functions that can be computed by the given BitNet in an n-dimensional ($D \geq 2^K n$) input space is lower bounded by $\Omega\Big(\big(\frac{D}{2^K n}\big)^{nKL}\Big)$.* ■

Proof. According to the definition of BitBlock, the width of k-th layer in a BitBlock is $\frac{D}{2^{(k-1)}}$. Thus, the parameter size of the given BitBlock can be computed by,

$$\begin{aligned}&\sum_{k=1}^{K}\Big(\frac{D}{2^{(k-1)}}\Big)^2\\ =&\frac{4}{3}\big(1-\frac{1}{4^K}\big)D^2.\end{aligned}$$

Thus, the parameter size of L stacked BitBlocks is $\Omega\Big(\frac{4}{3}L\big(1-\frac{1}{4^K}\big)D^2\Big)$.

To compute the number of linear regions, we simply ignore all right nodes in a BitBlock. As a result, given a BitBlock with width D and depth K, we approximate it as a standard DNN with depth K and width $D/2^k$ at k-th layer. According to Theorem 4 of [52], the maximal number of linear regions of functions that can be computed by a given BitBlock in an n-dimensional ($D \geq 2^K n$)

input space is lower bounded by

$$\begin{aligned}
&\prod_{k=1}^{K}\Big(\frac{D}{2^k n}\Big)^n \\
=&\Big(\frac{D}{n}\Big)^{nK} 2^{\frac{-n(1+K)K}{2}} \\
>&\Big(\frac{D}{n}\Big)^{nK} 2^{-nKK} \\
=&\Big(\frac{D}{2^K n}\Big)^{nK}.
\end{aligned}$$

As a result, with L BitBlocks, the maximal number of linear region is bounded by $\Omega\Big(\big(\frac{D}{2^K n}\big)^{nKL}\Big)$.

The approximation of BitBlock as standard DNN is valid. There are $K+1$ paths connected to the last layer of BitBlock. Take the BitBlock depicted in Figure 3.1 for example: 128, 128-64 and two 128-64-32. Each path can be considered as a standard network. The number of linear regions computed by the i-th path (with depth i) is

$$\prod_{j=1}^{i}(\frac{D}{2^j n})^n.$$

Since an output of a BitBlock is concatenation of output of all paths, the number of linear regions for the BitBlock is

$$(\sum_{i=1}^{K}\prod_{j=1}^{i}\frac{D}{2^j n})^n.$$

Thus, by using the big Ω notation, we have

$$\Omega\big((\prod_{j=1}^{K}\frac{D}{2^j n})^n.\big)$$

This result is basically equivalent to the case where all right nodes are ignored. This is due to the fact that the number of linear regions grows exponentially with depth as shown in [52]. Therefore, the asymptotic bound (big Ω) is dominated by the deepest path of the BitBlock.

The number of linear regions obtained by concatenation is computed using the space folding

method. Specifically, let the number of linear regions for i-th path is $(R_i)^n$, indicating that $i-th$ path can fold the input space along each axis by R_i segments. Concatenation of each path can fold each axis by $\sum_i R_i$ segments. As a result, the number of linear regions is $(\sum_i R_i)^n$. □

As we observe from these results, when D and L are fixed, as K increases, the expressive capacity of BitNets can grow with a small increase of the parameter size. Although the expressive capacity of a fully connected conventional network can grow faster as K increases, its parameter size also grows faster than that of a BitNet. Although these theoretical results are obtained for fully connected layers, our experimental observations reflect that the results can be applied also for the convolutional layers. In our experiments, a binary tree used at convolutional layers of a DNN can increase classification accuracy with a small increase of parameter size of the DNN.

3.3.4 An Information Theoretic Analysis

Next, we analyze the proposed method from an information theoretic perspective. Let $\mathbf{x}$ be an input random variable of a deep network, $\mathbf{z}$ be a hidden layer representation computed by the deep network, and $\mathbf{y}$ be a target random variable of the task (e.g. the ground truth label of an input image). The Information Bottleneck (IB) method [72] provides a principle to learn the representation by considering two factors: *compression* and *relevance*. Compression reflects the compactness of a learned representation $\mathbf{z}$ with respect to source $\mathbf{x}$, and is measured by Mutual Information $I(\mathbf{x};\mathbf{z})$. Relevance indicates the amount of information preserved by $\mathbf{z}$ that is useful to predict $\mathbf{y}$, and is measured by $I(\mathbf{z};\mathbf{y})$. Smaller value of $I(\mathbf{x};\mathbf{z})$ reflects that $\mathbf{z}$ is more compactly compressed, and larger value of $I(\mathbf{z};\mathbf{y})$ indicates that $\mathbf{z}$ is more relevant about $\mathbf{y}$.

IB principle can be employed as a measurement of optimality of learned representations [73]. Given a DNN which computes a representation $\mathbf{z}$, *optimality* of $\mathbf{z}$ was measured by its compression gap ΔG_c, and relevance gap ΔG_r to the optimal representation by

$$\Delta G_c = I(\mathbf{x};\mathbf{z}) - I(\mathbf{x};\mathbf{z}^*) \quad \text{and} \quad \Delta G_r = I(\mathbf{z}^*;\mathbf{y}) - I(\mathbf{z};\mathbf{y}), \tag{3.6}$$

where $\mathbf{z}^*$ is an optimal representation [72]. An optimal representation is a minimal sufficient statistic of $\mathbf{x}$ with respect to $\mathbf{y}$, meaning that $\mathbf{z}^*$ is maximally compressed from $\mathbf{x}$, while being maximally informative about $\mathbf{y}$. Observation of smaller gaps indicates that $\mathbf{z}$ is closer to $\mathbf{z}^*$. Minimal sufficient statistic is usually unachievable, because compressing $\mathbf{x}$ usually causes loss of information about $\mathbf{y}$. Smaller gaps ΔG_c and ΔG_r indicate that $\mathbf{z}$ is a better representation from an information theoretic perspective. Usually, we do not know the values of $I(\mathbf{x};\mathbf{z}^*)$ and $I(\mathbf{z}^*;\mathbf{y})$. Since $I(\mathbf{x};\mathbf{z}^*)$ and $I(\mathbf{z}^*;\mathbf{y})$ are fixed when given a dataset $(\mathbf{x},\mathbf{y})$ and a learned DNN, we can use $I(\mathbf{x};\mathbf{z})$ and $I(\mathbf{z};\mathbf{y})$ to analyze the optimality of representations computed by the DNN.

Note that compression amount $I(\mathbf{x};\mathbf{z})$ can be computed by

$$I(\mathbf{x};\mathbf{z}) = H(\mathbf{x}) - H(\mathbf{x}|\mathbf{z}). \tag{3.7}$$

Since $H(\mathbf{x})$ is a constant value, the compression is determined by the conditional uncertainty of predicting $\mathbf{x}$ given $\mathbf{z}$. Intuitively, the conditional uncertainty $H(\mathbf{x}|\mathbf{z})$ is determined by the number of different realizations of $\mathbf{x}$ residing in the input space $\mathcal{X}$ which are mapped to the same realization of $\mathbf{z}$ in the representation space $\mathcal{Z}$. Formally, given a feed forward DNN, we denote the representation computed at the i-th layer by $\mathbf{z}_i$, and the mapping function used for computing $\mathbf{x} \to \mathbf{z}_i$ by $f_{0,i}$. The distribution for $\mathbf{x}$ is denoted by $p(\mathbf{x})$. Then we have

$$p(\mathbf{x} = x|\mathbf{z}_i = z_i) = \frac{p(\mathbf{x} = x) \cdot \mathbb{1}\{f_{0,i}(x) = z_i\}}{\sum_{\{\tilde{x}\in\mathcal{X}: f_{0,i}(\tilde{x})=z_i\}} p(\mathbf{x} = \tilde{x})}, \tag{3.8}$$

where x, $\tilde{x}$, z_i are realizations of $\mathbf{x}$, $\mathbf{z}$ in the space $\mathcal{X}$, $\mathcal{Z}$ and $\mathbb{1}\{f_{0,i}(x) = z_i\}$ is a function which indicates whether the realization x is mapped to z_i.

Given a realization $z_i \in \mathcal{Z}_i$ computed at the i-th layer, there can be several different realizations of $\mathbf{z}_{i-1}$ obtained at the $(i-1)$-th layer being mapped to z_i, denoted by

$$\tilde{\mathcal{Z}}_{i-1}(z_i) = \{z_{i-1} \in \mathcal{Z}_{i-1} : f_{i-1,i}(z_{i-1}) = z_i\},$$

where $f_{i-1,i}$ is the mapping function of the i-th layer used for computing $\mathbf{z}_{i-1} \to \mathbf{z}_i$. Recursively,

for each realization in $\tilde{\mathcal{Z}}_{i-1}(z_i)$, there are several realizations from the $(i-2)$-th layer being mapped to it. Thus, (3.8) can be re-formulated in the following lemma.

Lemma 3.3.4. *Given a feed forward DNN with input $\mathbf{x} \in \mathcal{X}$, we denote the representation obtained at the i-th layer by $\mathbf{z}_i \in \mathcal{Z}_i$, and the mapping functions are $f_{0,i} : \mathbf{x} \to \mathbf{z}_i$ and $f_{i-1,i} : \mathbf{z}_{i-1} \to \mathbf{z}_i$. Then, the conditional probability is computed by*

$$p(\mathbf{x} = x|\mathbf{z}_i = z_i) = \frac{p(x) \cdot \mathbb{1}\{f_{0,i}(x) = z_i\}}{\sum_{\tilde{z}_{i-1} \in \tilde{\mathcal{Z}}_{i-1}(z_i)} \sum_{\tilde{z}_{i-2} \in \tilde{\mathcal{Z}}_{i-2}(\tilde{z}_{i-1})} \cdots \sum_{\tilde{z}_1 \in \tilde{\mathcal{Z}}_1(\tilde{z}_2)} \sum_{\{\tilde{x} \in \mathcal{X} : f_{0,1}(\tilde{x}) = \tilde{z}_1\}} p(\tilde{x})}, \quad (3.9)$$

where $\tilde{\mathcal{Z}}_{i-1}(z_i) = \{z_{i-1} \in \mathcal{Z}_{i-1} : f_{i-1,i}(z_{i-1}) = z_i\}$.

As we can see, as the depth increases, there can be more realizations of an input random variable in the input space being mapped to the same realization in the space $\mathcal{Z}_i$ at the last layer due to the recursive employment of maps, which gives the following corollary.

Corollary 3.3.5. *Suppose that we are given an L-layer DNN f_L which computes $\mathbf{x} \to \mathbf{z}_L$. Then, we can obtain $I(\mathbf{x}; \mathbf{z}_{L+1}) \leq I(\mathbf{x}; \mathbf{z}_L)$ by stacking an additional layer which computes $\mathbf{z}_L \to \mathbf{z}_{L+1}$ on top of the DNN f_L.*

As we mentioned in Section 3.3.3, DNNs can map different regions in the input space to the same region in the output space by space folding. Lemma 3.3.4 and Corollary 3.3.5 provide an information theoretic perspective of space folding, as illustrated in Figure 3.5.

If regions sharing the same class label in the input space are mapped to the same region in the output space, the compression amount of mutual information $I(\mathbf{x}; \mathbf{z})$ can be reduced, while relevance mutual information $I(\mathbf{z}; \mathbf{y})$ can be kept unchanged. As a result, we can obtain *better optimality* defined in (3.6).

Consider a binary classification problem with a source random variable $\mathbf{x}$ which is uniformly distributed in an n-dimensional cubic $[-1, 1]^n$ and a target random variable $\mathbf{y} \in \{0, 1\}$. Each dimension of the input space is uniformly divided into M intervals $\{U_m\}_{m=1}^{M}$, where $U_m = [\frac{2m-2}{M} - 1, \frac{2m}{M} - 1]$. Given a realization x, if the i-th dimensional value of x belongs to the $f(i)$-th interval i.e. $x^{(i)} \in U_{f(i)}$.

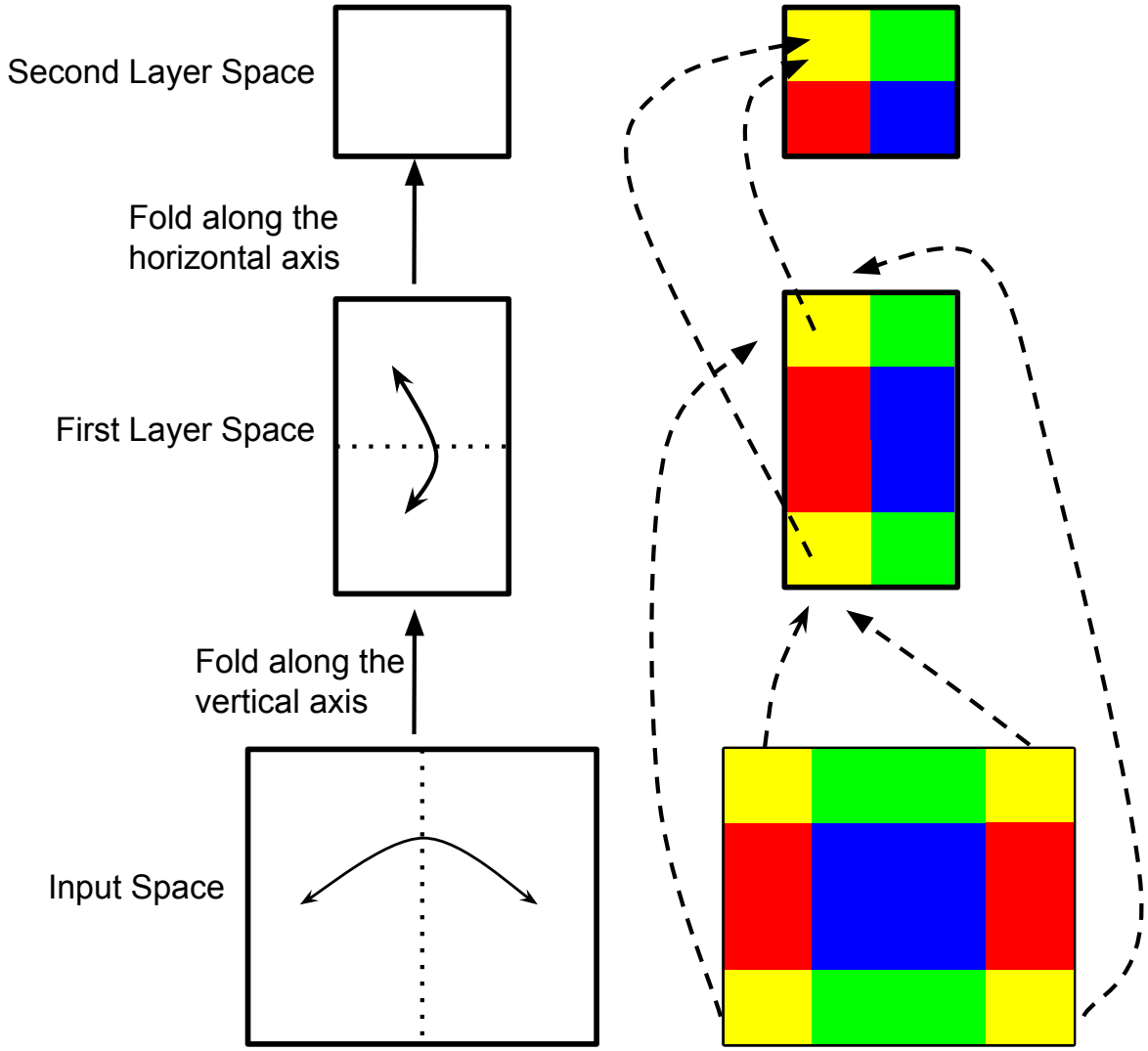


Figure 3.5: An information theoretic perspective of space folding. Different yellow regions residing in the input space are mapped to the same region in the space of the second layer.

Then, the label of x is determined by,

$$\mathrm{y} = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} f(i) \text{ is even} \\ 0 & \text{else} \end{cases} . \tag{3.10}$$

In this approach, labels of realizations distributed imply as a checkerboard pattern in the space. An example of a checkerboard distribution in the two-dimensional space is illustrated in Figure 3.6. The classification tasks on checkerboard distributions have been investigated in previous work [6, 5, 71]

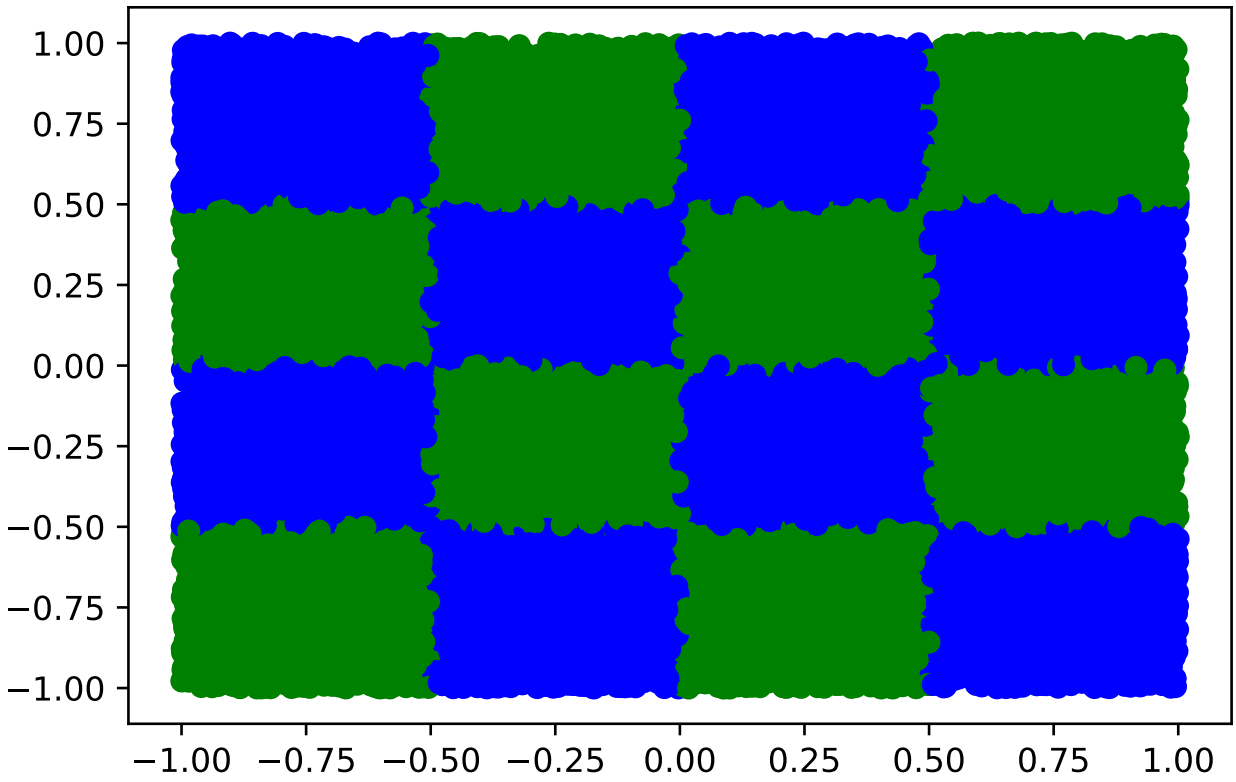


Figure 3.6: A checkerboard distribution in the two-dimensional space. $M = 4$. Samples residing in different regions (depicted by different color) have different labels.

to demonstrate the advantages of DNNs compared to local models such as decision trees and local kernel models. In this work, we analyse DNNs for solving checkerboard tasks from an information theoretic perspective.

Proposition 3.3.6. *Suppose that* $(\mathbf{x}, \mathbf{y})$ *follows an* n*-dimensional checkerboard distribution. Then, there exists a DNN stacked by* L *fully connected ConvenBlocks each having width* D *and depth* K*, and DNN computes a representation* $\mathbf{z}$ *at the* KL*-th layer, and we have*

$$I(\mathbf{z}; \mathbf{y}) = \log 2,$$
$$I(\mathbf{x}; \mathbf{z}) = n \log 2 \; - \; nKL \log \left(\frac{D}{n}\right),$$

where $(\frac{D}{n})^{KL} \geq M$.

Proof. From the definition of the checkerboard distribution given in (3.10), at each i-th axis, $(\mathbf{x}^{(i)}, \mathbf{y})$ also follows a one-dimensional checkerboard distribution. We can use the space folding method to

fold each axis, i.e. by constructing a network in Figure 3.4 and applying (3.4) to construct a mapping function. The mapping function constructed for each axis is illustrated in Figure 3.7. As a result, at each axis, samples residing in the green regions are mapped to $[0, 0.5]$, and those belonging to the blue regions are mapped to $[0.5, 1]$. All samples can be correctly classified by a representation $\mathbf{z}$. Thus, we can obtain

$$\begin{aligned} I(\mathbf{z};\mathbf{y}) &= H(\mathbf{y}) - H(\mathbf{y}|\mathbf{z}) \\ &= -2 \times 0.5 \log 0.5 + \log 1 \\ &= \log 2. \end{aligned}$$

Note that $p(\mathbf{x}|\mathbf{z})$ is determined by the number of different samples which reside in the input space, and which are mapped to the same point in the representation space of $\mathbf{z}$. According to the mapping function given in Figure 3.7, the number of different samples being mapped to the same sample in the $\mathbf{z}$ representation space is equal to the number of linear regions of the mapping function. By using Corollary 3.3.2, we can obtain,

$$\begin{aligned} I(\mathbf{x};\mathbf{z}) &= H(\mathbf{x}) - H(\mathbf{x}|\mathbf{z}) \\ &= -\int_{\mathbf{x}} \frac{1}{2^n} \log \frac{1}{2^n} + \int_{\mathbf{x}} \frac{1}{2^n} \log \left(\frac{D}{n}\right)^{-nKL} \\ &= n \log 2 - nKL \log \left(\frac{D}{n}\right). \end{aligned}$$

□

Thus, as the depth and width of DNN increase, the relevance mutual information $I(\mathbf{z};\mathbf{y})$ remains, while the compression mutual information $I(\mathbf{x};\mathbf{z})$ is reduced. The gap ΔG_c is reduced resulting in a better representation. Also, the depth has more impact on reducing the compression mutual information than width, which is consistent with the previous results, i.e. depth plays a more important role in growth of the expressive capacity.

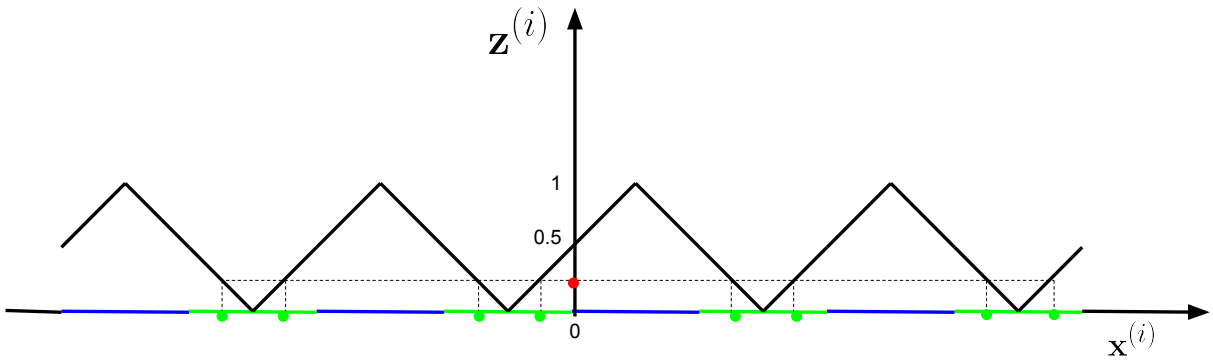


Figure 3.7: Mapping function constructed for one-dimensional checkerboard distribution. The samples residing in the green regions are mapped to $[0, 0.5]$, and those in the blue regions are mapped to $[0.5, 1]$. Given a point (e.g. the red point) in the $\mathbf{z}^{(i)}$ axis, there are multiple points (the green points) in the $\mathbf{x}^{(i)}$ axis being mapped to it.

Similarly, we can obtain the following theoretical results for BitNets:

Proposition 3.3.7. *Suppose that* $(\mathbf{x}, \mathbf{y})$ *are drawn from an* n*-dimensional checkerboard distribution. Then, there exists a BitNet stacked by* L *fully connected BitBlocks each having width* D *and depth* K*, and the DNN computes representation* $\mathbf{z}$ *at the* KL*-th layer with the following properties:*

$$I(\mathbf{z}; \mathbf{y}) = \log 2,$$
$$I(\mathbf{x}; \mathbf{z}) = n \log 2 \; - \; nKL \log \big(\frac{D}{2^K n}\big),$$

where $(\frac{D}{2^K n})^{KL} \geq M$ *and* $D > 2^K n$*.*

Proof. We can construct the same mapping function for each axis as shown in Figure 3.7. Thus, the number of different samples being mapped to the same sample in the representation space of $\mathbf{z}$ is equal to the number of linear regions of the mapping function. The proof follows by using the Proposition 3.3.3 with this property. □

As we observe from this proposition, when D and L of a BitNet are fixed, as K increases, the relevance mutual information $I(\mathbf{z}; \mathbf{y})$ does not change, while the compression mutual information $I(\mathbf{x}; \mathbf{z})$ is reduced, resulting in a *better* representation. However, the parameter size only increases slightly as analyzed in Proposition 3.3.3.

3.4 Experimental Results

We empirically analyze the proposed binary tree architecture using various baseline ResNets and several benchmark image classification datasets. We also analyze the gradient vanishing problem of the proposed binary tree architecture. In the implementation of BitNets, given a baseline ResNet, we can construct a BitNet of the same depth and the same block width with considerably fewer parameters compared to the baseline (ResNet). We can also increase the block width but keep the depth of baseline model by the proposed binary tree architecture and obtain a BitNet with a similar number of parameters. The configuration details of BitNets are given for each classification task.

3.4.1 Cifar-10 and Cifar-100

Cifar-10 and Cifar-100 [40] are image datasets consisting of 50,000 training images and 10,000 test images. The spatial size of each image is 32×32. Cifar-10 and Cifar-100 consist of 10 and 100 categories, respectively. The architectures of BitNets and ResNets used to perform analyses on the Cifar datasets are given in Table 3.1. As we can see from the table, the depth of a net is determined by the number of blocks in each group, i.e. n, and the depth of each block, i.e. k. The width of each block is determined by d.

We first compare performance of our proposed BitNets and that of Wide ResNets [78] by constructing nets with same depth and block width. Specifically, given a Wide ResNet with fixed value of d, k, and n, we use the same block width d for BitNet. Then, we set values of k and n for BitNet, such that the total depth of BitNet equals to the total depth of the given Wide ResNet. We can use different value combinations of k and n for BitNet as long as $k \times n$ value for BitNet equals $k \times n$ value for the given Wide ResNet. For example, given a 38-layer Wide ResNet with $(d = 4, k = 2, n = 6)$, we can construct a BitNet with $(d = 4, k = 3, n = 4)$ or a BitNet with $(d = 4, k = 4, n = 3)$, both implying 38 layers.

For fare comparison with Wide ResNets, we use the same training and testing setting as employed in [78]. Specifically, for data augmentation, we use a common method used in previous works [21, 50, 24, 78]. More precisely, 4 pixels with zero values are padded on each side of the original image

Group Name	Configuration	Output Size
conv1	conv, 16 channels	32×32
conv2	block$(d \times 16, k) \times n$	32×32
conv3($\downarrow$)	block$(d \times 32, k) \times n$	16×16
conv4($\downarrow$)	block$(d \times 64, k) \times n$	8×8
gap	global average pooling	1×1
fc	10 or 100-way softmax	

Table 3.1: The CNN architecture employed for classification using the Cifar-10 and Cifar-100. BitNets use BitBlock as block type, and ResNets use ConvenBlock with residual connection. k denotes the depth of each block ($k = 2$ for all ResNets used in this paper). d determines the width of each block. n denotes a stack of n blocks. All convolutional layers use filters of size 3×3. Batch Normalization is used at every convolutional layer before ReLU. Down-sampling is performed by applying stride 2 at the first convolutional layer of the first block in the groups conv3 and conv4.

to make a 40×40 image, from which a 32×32 patch is randomly cropped and randomly flipped horizontally. For testing, the original 32×32 image is used. Batch size is 128 that is split on two GPUs. The initial learning rate is 0.1, and is reduced by 0.2 on the 60-th, 120-th and 160-th epoch. The training is finished at the 200-th epoch.

Training and testing errors of BitNets used in our experiments for Cifar-100 classification task are given in Table 3.2. As shown in [56, 52], the expressive capacity reflects the complexity of class decision boundary computable by a DNN. We use training errors to quantify the expressive capacity of a DNN. As we can see from the table, the training errors of BitNets are close to that of the Wide ResNets, which indicates that by using considerably less number of parameters, the proposed BitNets can obtain expressive capacity similar to that of the Wide ResNets.

Table 3.3 provides the comparative results for several Wide ResNets and BitNets, which are designed with the same width and depth. As we can see from the results, using the same depth and width, BitNet has considerably less number of parameters and FLOPs. Moreover, BitNets can outperform Wide ResNets using considerably less number of parameters. We compare BitNets with four baseline Wide ResNets. In the analyses, we obtained the following results:

(1) A Wide ResNet having $(d = 4, k = 2, n = 6)$ is the deepest and narrowest architecture among

Model	Number of Parameters	Test Error	Train Error
Wide ResNet (d=4,k=2,n=6)	8.9M	22.89	0.018
BitNet (d=4,k=3,n=4)	3.7M	22.19	0.018
BitNet (d=4,k=4,n=3)	2.7M	22.60	0.022
BitNet (d=4,k=2,n=6)	5.4M	23.22	0.026
BitNet (d=4,k=6,n=2)	1.7M	23.87	0.028
Wide ResNet (d=10,k=2,n=2)	17.1M	21.59	0.018
BitNet (d=10,k=2,n=2)	9.6M	20.48	0.018
BitNet (d=10,k=4,n=1)	3.9M	23.88	0.020
Wide ResNet (d=10,k=2,n=3)	26.8M	20.75	0.018
BitNet (d=10,k=2,n=3)	15.6M	19.29	0.018
BitNet (d=10,k=3,n=2)	10.2M	19.37	0.018
Wide ResNet (d=12,k=2,n=4)	52.5M	20.43	0.018
BitNet (d=12,k=2,n=4)	31.2M	19.06	0.018
BitNet (d=12,k=4,n=2)	14.9M	19.22	0.018

Table 3.2: Cifar-100 classification error (%) of the BitNets and Wide ResNets used in our experiments.

four baseline architectures. The BitNet $(d = 4, k = 4, n = 3)$ and the BitNet $(d = 4, k = 3, n = 4)$ can obtain comparable performance with this baseline and the parameter size is only 30% and 41% of that of baseline, respectively. The BitNet $(d = 4, k = 2, n = 6)$ is constructed by using more number of blocks but reduce the depth of each block. As shown in Section 3.4.4, this BitNet suffers from a gradient vanishing problem during the training due to use of small k and a large n. As a result, the performance slightly degrades. BitNet $(d = 4, k = 6, n = 2)$ has least parameter size, i.e. only 19% of baseline's parameter size. However, it also performs worst.

(2) Compared with first baseline ResNet $(d = 4, k = 2, n = 6)$, the baseline ResNet $(d = 10, k = 2, n = 2)$ is wider and shallower. The BitNet $(d = 10, k = 2, n = 2)$ outperforms it by 1% in the Cifar-100 task by using approximately 56% of baseline ResNet's parameter size. Another configuration of BitNet $(d = 10, k = 4, n = 1)$ uses only one BitBlock at each group, resulting in

Model	Depth	Param	FLOP	Cifar-10	Cifar-100
NIN [50]	-	-	-	8.81	35.67
ELU [13]	-	-	-	6.55	24.28
DSN [47]	-	-	-	7.97	34.57
AllCNN [66]	-	-	-	7.25	33.71
ResNet [24]	1202	10.2M	-	4.91	–
preact-ResNet [25]	1001	10.2M	-	4.62	22.71
Stochastic Depth ResNet [29]	110	1.7M	-	5.25	24.98
FractalNet [45]	40	22.9M	-	5.24	22.49
Wide ResNet (d=4,k=2,n=6) [78]	38	8.9M	1.34×10^9	4.97	22.89
BitNet (d=4,k=3,n=4)	38	3.7M	0.53×10^9	4.82	<u>22.19</u>
BitNet (d=4,k=4,n=3)	38	2.7M	0.39×10^9	<u>4.65</u>	22.60
BitNet (d=4,k=2,n=6)	38	5.4M	0.78×10^9	5.31	23.22
BitNet (d=4,k=6,n=2)	38	1.7M	0.24×10^9	4.77	23.87
Wide ResNet (d=10,k=2,n=2) [78]	14	17.1M	2.64×10^9	4.56	21.59
BitNet (d=10,k=2,n=2)	14	9.6M	1.32×10^9	<u>4.17</u>	<u>20.48</u>
BitNet (d=10,k=4,n=1)	14	3.9M	0.49×10^9	4.97	23.88
Wide ResNet (d=10,k=2,n=3) [78]	20	26.8M	4.06×10^9	4.44	20.75
BitNet (d=10,k=2,n=3)	20	15.6M	2.21×10^9	<u>**3.78**</u>	<u>19.29</u>
BitNet (d=10,k=3,n=2)	20	10.2M	1.41×10^9	3.81	19.37
Wide ResNet (d=12,k=2,n=4) [78]	26	52.5M	7.87×10^9	4.33	20.43
BitNet (d=12,k=2,n=4)	26	31.2M	4.45×10^9	<u>4.07</u>	<u>**19.06**</u>
BitNet (d=12,k=4,n=2)	26	14.9M	2.06×10^9	4.11	19.22

Table 3.3: Classification error (%) of CNNs for the Cifar-10/100 datasets. Using the same depth and block width, our BitNets can outperform Wide ResNets with considerably less number of parameter size. Underlined numbers indicate the best performance among models having the same depth and same block width. Bold numbers denote the best performance obtained for all models. Definition of d, k and n are given in Table 3.1. All of Wide ResNets and our BitNets are trained using data augmentation and without using dropout.

only 23% of baseline's parameter size. However, the performance is also degraded by more than 1% using the Cifar-100 dataset. For the Cifar-10 dataset, both BitNets obtain similar performance compared to the baseline.

(3) For the Wide ResNet $(d = 10, k = 2, n = 3)$, both BitNets obtain more than 1% performance boost using the Cifar-100 dataset. For the Cifar-10 dataset, the performance boost is more than 0.5%. Notably, the parameter size of the BitNet $(d = 10, k = 3, n = 2)$ is only 38% of the parameter size of the baseline.

(4) The Wide ResNet $(d = 12, k = 2, n = 4)$ has the largest parameter size among four baseline models. Our two BitNets both outperform the baseline by more than 1% accuracy. Note that the parameter size of the BitNet $(d = 12, k = 4, n = 2)$ is only 28% of that of the baseline.

We emphasize that the performance of baseline Wide ResNets are already close to the state-of-the-art. Thus more than 1% boost of the accuracy is obtained. To summarize, most of our BitNets can achieve better or approximately equal accuracy using less number of parameters, which indicates that our binary tree architecture can improve the parameter size and accuracy trade-off of baseline Wide ResNets. There are two BitNets whose accuracy are roughly 1% lower than that of their baselines. This is possibly because they use too less number of BitBlocks causing insufficient expressive capacity. The rest of BitNets obtain sufficient expressive capacity with a relatively less number of parameter size. Compared with other previous models such as Stochastic Depth ResNet [29] and FractalNet [45], our BitNets can outperform them using less number of parameter size.

3.4.2 ILSVRC12

To evaluate the proposed architecture on a large scale image classification task, we also use the training and validation dataset of ILSVRC12 [59], which consists of 1.3M training images and 50,000 validation images belonging to 1000 categories. During training, data augmentation and image preprocessing methods are used as follows. The image is cropped by scale and aspect ratio augmentation method [69], and then resized to 224×224. A random horizontal flip is also applied. The input images are mean subtracted and variance normalized on each RGB channel. The color

Group	BitNet-26	BitNet-34	Output Size
conv1	conv 7×7, 64 channels, stride 2		112×112
conv2	max pooling 3×3, stride 2		56×56
	$\mathrm{b}(128, 3) \times 2$	$\mathrm{b}(256, 4) \times 2$	
conv3	$\mathrm{b}(192, 3) \times 1$ $\mathrm{b}(256, 3) \times 1$	$\mathrm{b}(384, 4) \times 2$	28×28
conv4	$\mathrm{b}(384, 3) \times 2$	$\mathrm{b}(512, 4) \times 2$	14×14
conv5	$\mathrm{b}(512, 3) \times 1$ $\mathrm{b}(768, 3) \times 1$	$\mathrm{b}(768, 4) \times 2$	7×7
gap	global average pooling		1×1
fc	1000-way softmax		
Param.	12.79M	22.99M	
FLOP	2.8×10^9	7.8×10^9	
Depth	26	34	

Table 3.4: Structure of BitNets used for ILSVRC12 classification task. $\mathrm{b}(d, k) \times n$ refers to a stack of n BitBlocks with width d and depth k. All convolutional layers employed in each BitBlock use filters of size 3×3. The output size is reduced by applying stride 2 at the first convolutional layer of first block in some groups.

distortion methods proposed in [41] and [28] are both used. For validation, the image is resized such that its shorter side is 256, and a center crop of 224×224 are used to test. Batch size is 256 split on 8 GPUs. The initial learning rate is 0.1 and is reduced by 10^{-1} at each 30 epoch. The training is finished at the 90-th epoch. Following [24], stochastic gradient descent (SGD) with momentum 0.9 is used as our optimizer and the weight decay is set as 0.0001. Batch Normalization is used in every convolutional layer before ReLU. We didn't use dropout [67] in any BitNet.

In this task, we first construct two BitNets (BitNet-26 and BitNet-34) in order to compare their performance with that of ResNet-34 [24]. The details of models are given in Table 3.4. Then we construct several versions of BitNets each of which has the same depth and width as wide ResNet-18 and wide ResNet-34 have. The details of wide BitNets are given in Table 3.5 and 3.6.

Classification results are given in Table 3.7. The results show that our BitNets have better

Group	Wide ResNet-18	Wide BitNet-18	Output Size
conv1	conv 7×7, 64 channels, stride 2		112×112
conv2	max pooling 3×3, stride 2		56×56
	r$(64 \times d, 2) \times 2$	b$(64 \times d, 2) \times 2$	
conv3	r$(128 \times d, 2) \times 2$	b$(128 \times d, 2) \times 2$	28×28
conv4	r$(256 \times d, 2) \times 2$	b$(256 \times d, 2) \times 2$	14×14
conv5	r$(512 \times d, 2) \times 2$	b$(512 \times d, 2) \times 2$	7×7
gap	global average pooling		1×1
fc	1000-way softmax		
Depth	18		

Table 3.5: Structure of 18-layer wide ResNets/BitNets used for ILSVRC12 classification task. r/b$(64 \times d, k) \times n$ refers to a stack of n Residual/BitBlocks with width $64 \times d$ and depth k. All convolutional layers employed in each BitBlock use filters of size 3×3. The output size is reduced by applying stride 2 at the first convolutional layer of the first block in some groups.

Group	Wide BitNet-34-A	Wide BitNet-34-B	Wide BitNet-34-C	Output Size
conv1	conv 7×7, 64 channels, stride 2			112×112
conv2	max pooling 3×3, stride 2			56×56
	b$(128, 3) \times 2$	b$(128, 3) \times 2$	b$(128, 2) \times 2$	
conv3	b$(256, 4) \times 2$	b$(256, 4) \times 2$	b$(256, 2) \times 3$	28×28
conv4	b$(512, 3) \times 4$	b$(512, 3) \times 4$	b$(512, 3) \times 4$	14×14
conv5	b$(1024, 3) \times 2$	b$(1024, 2) \times 3$	b$(1024, 3) \times 3$	7×7
gap	global average pooling			1×1
fc	1000-way softmax			
Param.	33.88M	44.50M	46.91M	
FLOP	5.28×10^9	5.80×10^9	6.39×10^9	
Depth	34			

Table 3.6: Structure of 34-layer wide BitNets used for ILSVRC12 classification task. b$(d, k) \times n$ refers to a stack of n BitBlocks with width d and depth k. All convolutional layers employed in each BitBlock use filters of size 3×3. The output size is reduced by applying stride 2 at the first convolutional layer of the first block in some groups.

parameter size and accuracy trade-off compared to ResNets. Specifically, BitNet-34 outperforms ResNet-34 B by 1% using the same parameter size, while their depth is the same. Another smaller BitNet-26 obtains 1% higher error compared to ResNet-34 B. However it is shallower and the parameter size is approximately 50% of ResNet-34 B. Compared to model FractalNet-34, BitNet-34 also outperforms it. BitNet-26 outperforms ResNet-18 B by approximately 2% accuracy using the same number of parameters. Increasing the width of ResNet-18 by methods proposed in [78] can improve the accuracy. However the cost for the improvement is also huge. BitNet-26 and ResNet-18 B width $\times 1.5$ have comparable accuracy, but the number of parameters of our BitNet is only 49% of parameters of ResNet. Similarly, the parameter size of BitNet-34 is only 50% of that of ResNet-18 width $\times 2$.

In terms of comparison between wide versions, the proposed Wide BitNets obtain better classification accuracy and parameter size trade-off than Wide ResNets. Specifically, Wide BitNet-18 Width $\times 2$ outperforms Wide ResNet-18 Width $\times 1.5$ by 0.36% accuracy using the same number parameters of ResNet. As the width further increases, the advantage of our BitNets seems more notable. Wide BitNet-18 Width $\times 3$ outperforms Wide ResNet-18 Width $\times 2$ by 0.86% on the single crop test by using $1.27\times$ parameters of ResNet.

We also investigate different configurations of depth and number of blocks on the ILSVRC12 task. As shown in Table 3.6, we construct three types of Wide BitNet-34 by using different number of BitBlocks and depth, tagged as A, B, C. Wide BitNet-34-A has the least number of parameters due to the least number of BitBlocks but with relatively larger depth for each block. As a result, the accuracy of Wide BitNet-34-A is the lowest. By reducing the depth for each block and using larger number of blocks, we can obtain better performance by the cost of slight increase of parameter size.

To summarize, the proposed BitNet models obtain better parameter size and accuracy trade-off compared to baseline ResNets. In some structures, BitNets can obtain better performance than ResNets even with considerably less number of parameters. Our proposed BitNet architectures also outperform tree structure net FractalNet-34.

Model	Single Crop	Ten Crop	Param
FractalNet-34 [45]	–	24.12	–
ResNet-18 B [24]	30.43	28.22	13.1M
Width $\times 1.5$ [78]	27.06	–	25.9M
Width $\times 2$ [78]	25.58	–	45.6M
Width $\times 3$ [78]	24.06	–	101.8M
ResNet-34 B [24]	26.73	24.76	23.2M
Width $\times 1.5$ [78]	24.50	–	48.6M
Width $\times 2$ [78]	23.39	–	86.0M
BitNet-26	27.74	25.83	12.8M
BitNet-34	25.46	23.77	23.0M
Wide BitNet-18 Width $\times 2$	26.70	24.61	26.10M
Wide BitNet-18 Width $\times 3$	24.72	22.77	57.89M
Wide BitNet-18 Width $\times 4$	24.10	22.16	102.18M
Wide BitNet-34-A	25.02	23.00	33.88M
Wide BitNet-34-B	24.88	22.92	44.50M
Wide BitNet-34-C	24.73	22.55	46.91M

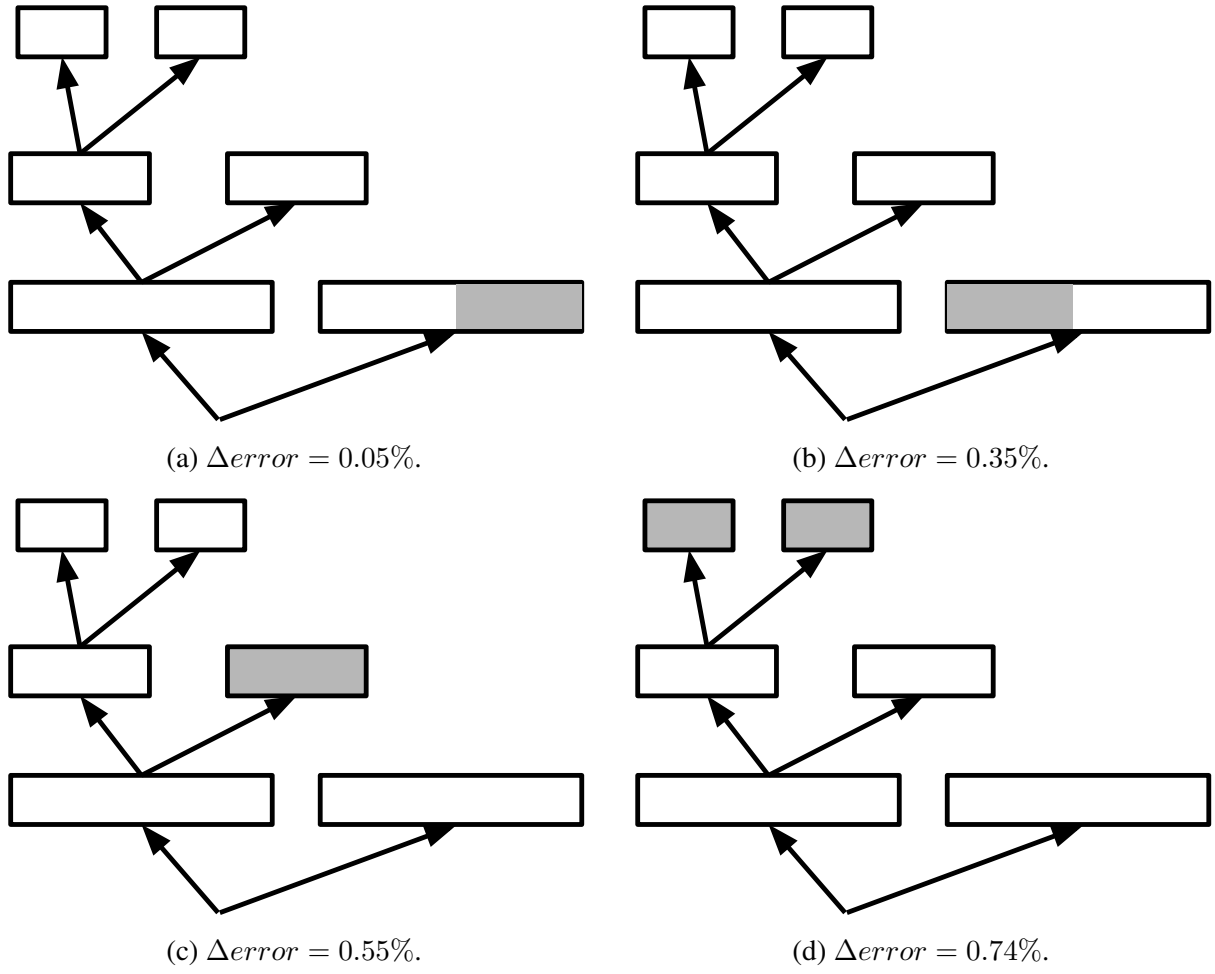
Table 3.7: Single model, Top-1 classification error (%) obtained using the ILSVRC12 validation dataset.

3.4.3 Experimental Analyses of the Effect of Depth of BitBlocks to Classification Performance

In this subsection, we observe that BitBlocks having different depth may provide different performance using the same width d and same number of BitBlocks n (see BitNet $(d = 10, k = 3, n = 2)$ and BitNet $(d = 10, k = 2, n = 2)$ in Table 3.3). Thus, we further analyze how performance of BitNets changes with respect to the depth of BitBlocks. Specifically, we evaluate BitNet $(d = 4, k, n = 4)$ (a deep-narrow one) and BitNet $(d = 12, k, n = 2)$ (a relatively shallower-wider one) by setting different values to k. The results are given in Table 3.8. As illustrated in the

table, as k increases, both BitNets gain a performance boost due to an increase on the expressive capacity. Note that for the BitNet $(d = 12, k, n = 2)$ employed using the Cifar-100, there is almost a 4% performance boost from $k = 1$ to $k = 2$ with a 33% increase of parameter size. We observe that the performance boosts further as k increases. These observations match our theoretical analyses provided in Section 3.3.3, which states that as d and n are fixed, increasing k can increase the expressive capacity of a BitNet with a small increase of parameter size. However, the boosting trend tends to be saturated after $k \geq 3$, and the increase of parameter size is also neglectable. This can be explained as follows. As k increases, the width of convolutional layer also decreases in the binary tree architecture resulting in a saturated expressive capacity. Additionally, we also observe that wider BitNets ($d = 12$) gain more performance boost than a narrower Bitnet ($d = 4$) with the same increase on k. This is simply because the increased layers in the wider BitNet are wider than narrower one, thus it can increase more expressive capacity.

The output of a BitBlock is a concatenation of leaf nodes of our binary tree architecture. As leaf nodes have different depth to the root node of a BitBlock, we analyze the contribution of different leaf nodes to the classification performance. To this end, we disable a given leaf node by setting the activation values of the leaf node as zero during the inference time. Then, the decreased classification accuracy is used to measure the contribution of the given leaf node in the BitBlock. The more accuracy is decreased, the more contribution the given leaf node has. Note that similar methods are used in Random Forest [9] models to analyze the importance of features. Specifically, we first train a baseline BitNet on the Cifar-10 dataset using an architecture BitNet $(d = 4, k = 3, n = 3)$. The test classification error of the baseline is 6.32%. We then disable leaf nodes in the last BitBlock (the one closest to the classification layer) of the baseline BitNet. The disabled leaves and the corresponding increased test errors are shown in Figure 3.8. We perform analyses using four disabling scenarios, in each of which we disable $\frac{D}{4}$ neurons (D is the total number of neurons of the concatenated output) from different leaves. In the first two scenarios shown in Figure 3.8a and 3.8b, we disable neurons of a leaf node with the lowest depth. As a result, the error increases by 0.05% and 0.35%, respectively. In Figure 3.8c, we disable the leaf node computed from the intermediate layer, and the error increases by 0.55%. In Figure 3.8d, the leaves used at the highest layer are disabled, and the



(a) $\Delta error = 0.05\%$. (b) $\Delta error = 0.35\%$.

(c) $\Delta error = 0.55\%$. (d) $\Delta error = 0.74\%$.

Figure 3.8: An analysis of contribution of leaf nodes to the classification performance. The gray zone denotes the neurons disabled during the test phase by setting their activation values to zero. $\Delta error$ is the increased error obtained after disabling the neurons.

error increases by 0.74%. Thus, the leaves used at the higher layers have more contribution than those used at the lower layers in terms of the classification performance.

3.4.4 Experimental Analyses of Gradient Vanishing Problem

In this section, we analyze our BitNets considering the gradient vanishing problem. Specifically, during the training of a BitNet or Wide ResNet employed using the Cifar-100, we compute the mean magnitude (L2-norm) of gradient obtained at the first convolutional layer per epoch. The results are given in Figure 3.9. We analyze nine models having different depth but the same width $d = 4$. In

	BitNet $(d = 4, k, n = 4)$		BitNet $(d = 12, k, n = 2)$	
	Param.	Cifar-10/100	Param.	Cifar-10/100
$k = 1$	2.7M	4.98/23.54	10.3M	6.02/23.80
$k = 2$	3.5M	**4.68**/22.72	13.8M	4.02/19.86
$k = 3$	3.7M	4.82/**22.19**	14.7M	**3.98**/18.97
$k = 4$	3.7M	4.69/22.65	14.9M	4.11/19.22
$k = 5$	3.7M	4.69/22.86	15.0M	4.09/**18.95**
$k = 6$	3.7M	4.77/22.69	15.0M	4.19/19.51

Table 3.8: Cifar-10/100 classification error (%) of two BitNets with respect to k. Definition of d, n, and k are given in Table 3.1.

the figure, wide resnet-38 refers to the Wide ResNet $(d = 4, k = 2, n = 6)$ having 38 layers, and plainnet-38 is the one designed without using residual shortcut connections. Models denoted by b$(d, k, n) - m$ are the proposed BitNets and m is the total depth.

As we can see from the figure, for all models having 38 layers, our BitNet b$(4, 6, 2) - 38$ shows the strongest magnitude, even stronger than wide resnet-38. This result indicates that using concatenation in the proposed binary tree architecture can ease the gradient vanishing problem. We also observe that gradient becomes weaker as the number of blocks n is increased and the depth k of each BitBlock is reduced. For instance, for all 38 layers BitNets, the magnitude can be roughly sorted by b$(4, 2, 6) - 38$ weaker than b$(4, 3, 4) - 38$ weaker than b$(4, 4, 3) - 38$ weaker than b$(4, 6, 2) - 38$ according to the strength of the magnitude. This observation reflects that as n increases and k decreases, features obtained at less number of lower layers are concatenated to form the output of each BitBlock. In general, the gradients propagate more layers to reach lower layers.

We also analyze how the gradient magnitude changes with respect to BitBlock's depth k if d and n are fixed. As we can see, b$(4, 1, 2) - 8$ shows the strongest magnitude of gradient among all nine models as expected because it is the shallowest model. By increasing k, we observe that for BitNet b$(4, 2, 2) - 14$ and b$(4, 4, 2) - 26$, the magnitude is decreased because more layers are used to propagate gradient in BitBlocks.

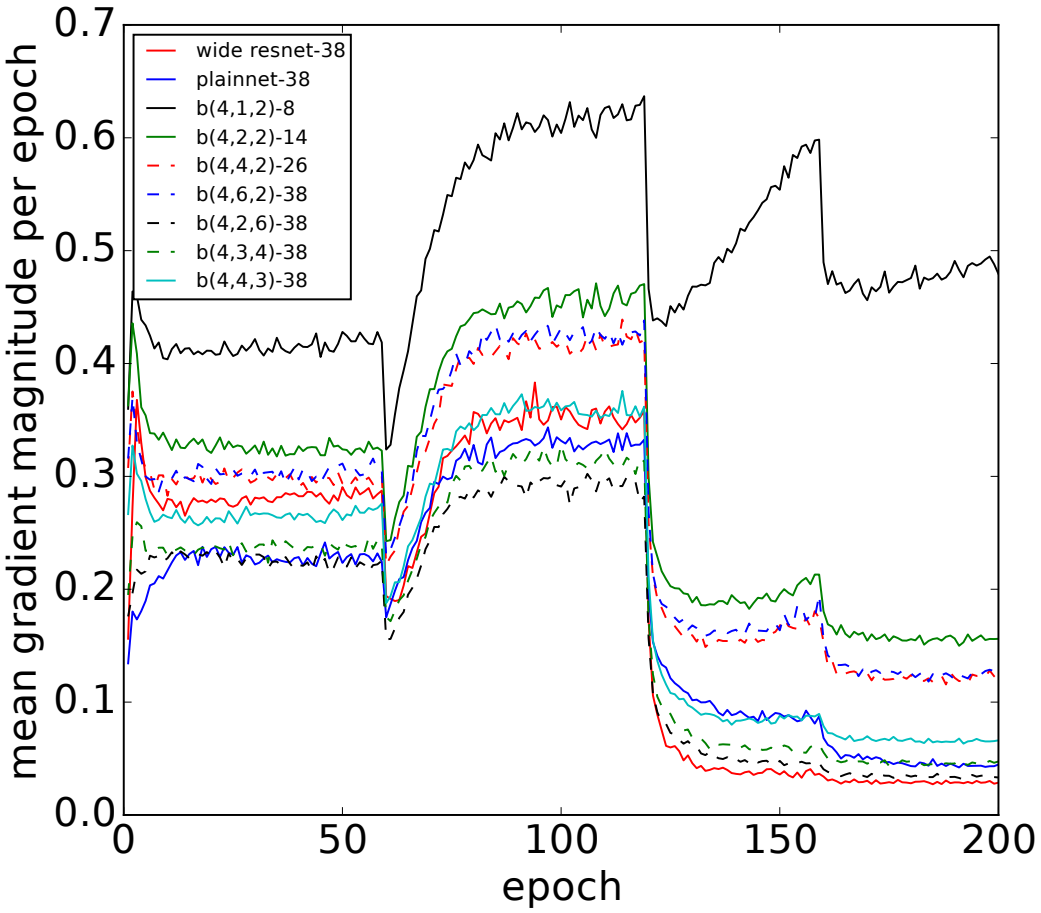


Figure 3.9: Mean magnitude (L2-norm) of gradient of the first convolutional layer computed per epoch during training phase. b$(d, k, n) - m$ refers to an m-layer BitNet with structure configuration (d, k, n) defined in Table 3.1. Best viewed in color print.

The errors obtained for nine models are given in Table 3.9. Although b$(4, 1, 2)-8$ and b$(4, 2, 2)-14$ show stronger gradient magnitude than wide resnet-38, they provide higher classification errors. This is mainly because the depth of these two BitNets is too small resulting in insufficient expressive capacity. BitNet b$(4, 4, 3) - 38$ obtains comparable classification performance by using larger depth. The gradient magnitude of BitNet b$(4, 4, 3) - 38$ is also comparable with that of resnet, which benefits from the proposed binary tree architecture. Without using binary tree architecture, the gradient magnitude of plainnet-38 is weaker than that of resnet-38 and the final classification error is larger.

Model	Param.	Error
wide resnet-38	8.9M	22.89
plainnet-38	8.9M	29.13
b$(4, 1, 2) - 8$	1.2M	29.19
b$(4, 2, 2) - 14$	1.5M	25.45
b$(4, 4, 2) - 26$	1.7M	22.72
b$(4, 6, 2) - 38$	1.7M	23.87
b$(4, 2, 6) - 38$	5.4M	23.22
b$(4, 3, 4) - 38$	3.7M	22.19
b$(4, 4, 3) - 38$	2.7M	22.60

Table 3.9: Cifar-100 classification error (%) of the models illustrated in Figure 3.9.

3.5 Conclusions

In this chapter, we introduce and analyze a binary tree architecture to truncate architecture of wide networks considering their parameter size and accuracy trade-off. In the proposed architecture, the width at each layer is incrementally reduced from lower layers to higher layers. Also, features obtained at different layers are concatenated to form the output of our architecture. In our experiments, the networks which are designed using the proposed architecture, called BitNets, can obtain better parameter size and accuracy trade-off on several benchmark datasets compared to baseline networks endowed with conventional architectures. Additionally, in our experimental analyses, we observe that the concatenation structure can ease the gradient vanishing problem. We also provide a theoretical analyses of the expressive capacity of BitNets and the optimality of learned representations by BitNets. In our future work, we plan to use BitNets for object detection tasks.

Chapter 4

Integrating Deep Features for Material Recognition

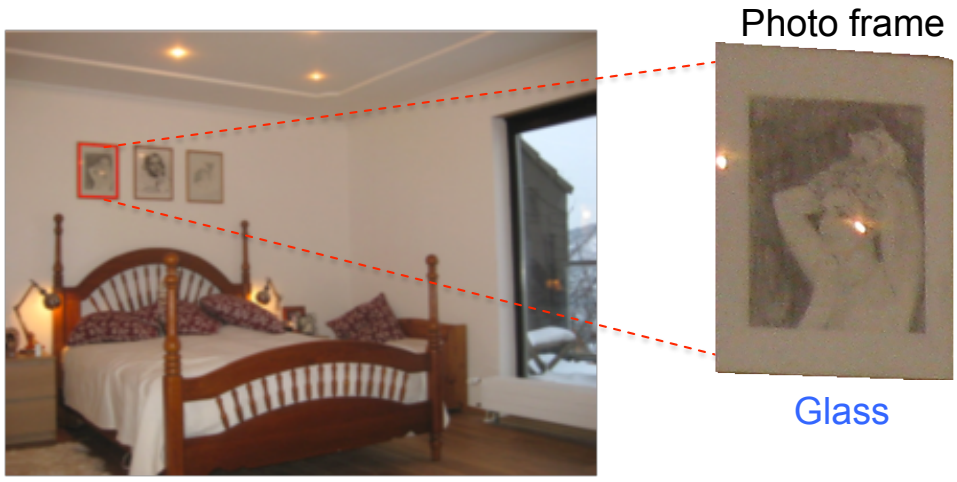
This chapter considers the problem of material recognition. Motivated by observation of close interconnections between material and object recognition, we study how to select and integrate multiple features obtained by different models of Convolutional Neural Networks (CNNs) trained in a transfer learning setting. To be specific, we first compute activations of features using pre-trained CNNs on images to select a set of samples which are *best* represented by the features. Then, we measure uncertainty of the features by computing entropy of class distributions for each sample set. Finally, we compute contribution of each feature to classes discrimination for feature selection and integration. Experimental results show that the proposed method achieves state-of-the-art performance on two benchmark datasets for material recognition. Additionally, we introduce a new material dataset, named EFMD, which extends Flickr Material Database (FMD). By the employment of the EFMD for transfer learning, we achieve $\mathbf{84.0\% \pm 1.8\%}$ accuracy on the FMD dataset, which is close to the reported human performance $\mathbf{84.9\%}$.

4.1 Motivation

In this chapter, we consider a material recognition problem, which is to identify material categories such as *glass* or *fabric* of an object (i.e., the material from which the object is made) using its single RGB image. We are particularly interested in employment of interconnection between material and object recognition, by utilizing which we aim to perform material recognition accurately. More precisely, we aim to develop a method that can efficiently transfer feature representations learned for different tasks/datasets including object recognition to a target task of material recognition. Toward this end, using convolutional neural networks (CNNs) [41, 65], we study how to select and integrate multiple features obtained by different models of CNNs trained using different datasets to perform different tasks.

Various studies of psychophysics [64, 53] imply that material perception in human vision is interconnected with perception of object category. An observation is that human can perceive some material properties and categories of objects only after correct recognition of the object categories. An example is shown in Figure 4.1a. This dependency can be reversed; object category of some objects can be correctly recognized only after accurate perception of material category of the objects. In this work, we conjecture that there exist mutual dependencies between perception of objects and materials. This interpretation can be further generalized to wider contexts such as perception of scene and texture, although we focus on the relationship between object and material categories in this work.

However, it is challenging to model such dependencies, and utilize them to accurately perform a task of interest (material recognition in our case). For example, it is possible that an image belonging to an object category *flower* may belong to one of the material categories; *plastic*, *paper*, and *foliage* (see Figure 4.1b). In addition to such category-level dependency, we consider representation-level dependency, which is much more complicated. It is obvious that not every representation learned for object recognition is beneficial for material recognition, due to divergent appearance that a material category may exhibit. Hence, it is important to select *useful* object and material features for the material recognition task.



(a)

(b)

Figure 4.1: (a) An example of integration of knowledge of object and material representations for material recognition. We can first recognize a target image (the image patch residing within the red rectangle) as a photo frame (object), and then we infer the photo frame as *glass* category (material). (b) As shown in the first row, *circle* shape appears in three different material categories. The second row shows that features that represent *flowers* are not merely peculiar to *foliage*.

In this chapter, we propose a feature selection method to select and combine deep features learned using a transfer learning setting. The contributions of this chapter are summarized as follows:

- We propose a method for material recognition by selecting and integrating multiple features of different CNN models. They are pre-trained on different datasets/tasks and, if possible and necessary, they are further fine-tuned on the target task/dataset in advance.
- We introduce an extended version of the benchmark material dataset (namely, FMD [62]), called EFMD which is ten times larger than the FMD dataset. The images of EFMD are selected according to surface properties of objects observed in the images that are similar to that of FMD. By the employment of EFMD for transfer learning, we achieve $84.0\% \pm 1.8\%$ accuracy on FMD, which is close to human performance (84.9%) [63].

The rest of this chapter is organized as follows. We summarize related work in Section 4.2. The proposed method is introduced in Section 4.3. Section 4.4 provides experiments conducted on several benchmark datasets. Section 4.5 concludes the section.

4.2 Related Work

Surface properties of materials, and the relationship between perception of material and object categories are analyzed in [63, 64]. They first proposed a well-designed benchmark dataset called FMD. Then, they designed descriptors to extract hand-crafted features for representation of various surface properties such as color, texture and shape for material recognition in [63]. Moreover, the relationship between object and material recognition was analyzed for accurate and fast perception of materials in [64].

In [12], a filter bank based method was developed using CNNs for texture recognition. The authors achieved state-of-the-art performance using several benchmark datasets for texture recognition and material recognition, including 82.4% accuracy on the FMD dataset. In [60], a method was proposed to discover local material attributes from crowdsourced perceptual material distances. They show that without relying on object cues (e.g., outlines, shapes), material recognition can

still be performed by employing the discovered local material attributes. In this work, we focus on utilizing object cues for material recognition.

Commonly used feature selection methods can be categorized into wrapper methods [38, 35, 58] and filter methods [8, 77, 22, 49, 57, 2, 44]. Wrapper methods are used to select features by training a classifier on a subset of features, and determine the utility of these features according to the accuracy of the classifier. Features are ranked according to particular criteria using filter methods. For instance, correlation coefficients were used in [77] to measure importance of each individual feature. Mutual information is also a popular criterion used for ranking features [2, 44, 57]. A max-relevance and min-redundancy criterion was proposed in [57]. However, in this method, they compute mutual information of each pair of features at each iteration to evaluate redundancy of features. Therefore, the computation cost of the algorithm was increased with the number of candidate features. In [76], a feature selection method was proposed based on a variant of Adaboost. In this method, each feature is considered as a weak classifier. At each iteration, a weak classifier that performs best is selected, and the weights assigned to samples are updated. An advantage of boosting based feature selection methods is reduction of running time for the training phase. In this paper, we propose a filter-based feature selection method in order to select the most discriminative features efficiently by minimizing an entropy criterion in a boosting scheme.

4.3 Feature Selection and Integration for Deep Representations

In this section, we propose a method to analyze and integrate multiple features extracted by different CNNs pretrained on different tasks.

4.3.1 Outline of the Method

In our approach, we first extract features from material images using multiple CNN models trained using/for different datasets/tasks. We identify a feature by an activation value of a unit (neuron) computed at a layer of a CNN. If it is possible and necessary, CNN models can be fine-tuned on the training samples of the target task of material recognition. We then consider each individual feature

as a weak classifier, although we do not explicitly train a weak classifier for each feature unlike employed in [76]. Instead, we utilize the class entropy as a criterion to measure how discriminative each feature is. As suggested in [18, 79], the images maximizing a feature value are determined as the most representative samples for utilization of the feature. Thus, we calculate class entropy for each feature using a set of images on which the activation value of the feature is maximized, and then use a weighted sum of the class entropy over the image set. A weight is given to each sample in the training data. Initially, we assign equal weights for all the samples. Then, the proposed method updates the weights, and selects features, iteratively. At each iteration, the most discriminative feature is selected for integration according to the weighted class entropy. Meanwhile, the weights of images on which an integrated feature is activated are penalized according to the class entropy. This procedure enables us to select a set of the most discriminative features for a given set of features.

4.3.2 Details of the Algorithm

Suppose that we are given N image datasets $\mathcal{D}_n = \{(I_i, Y_i)\}_{i=1}^{M_n}$, $n = 1, 2, \ldots, N$, where $I_i \in \mathcal{X}_n$ is the i-th image, $Y_i \in \mathcal{Y}_n$ is the corresponding class label, and the samples $\{(I_i, Y_i)\}_{i=1}^{M_n}$ are independent and identically distributed (i.i.d.) according to a distribution $\mathcal{P}_n$ on $\mathcal{X}_n \times \mathcal{Y}_n$. We then train a CNN using each dataset $\mathcal{D}_n$ for the associated task. Our method can be used in case CNNs have identical or different network architectures. Given a layer (or a combination of layers) of a CNN trained on $\mathcal{D}_n$[1], we use Φ_n to denote the mapping from an input image to the activation value of the layer(s) computed by the CNN. Given a new dataset $\mathcal{D}_b = \{(I_i, Y_i)\}_{i=1}^{M_b}$, we express the activation value of an image I_i of the sample $(I_i, Y_i) \in \mathcal{D}_b$ by $\mathbf{x}_{nb,i} = \Phi_n(I_i)$.

In this work, we aim to extract features from a given set of material images $\mathcal{D}_b$ by employing different CNNs independently trained using images of objects and materials. Therefore, we consider only datasets $\mathcal{D}_n, \forall n$, consisting of images of objects and/or materials. For the sake of simplicity of the notation and concreteness, we denote $\mathbf{x}_m \triangleq \Phi_n(I \in \mathcal{D}_b)$ and $\mathbf{x}_o \triangleq \Phi_{n'}(I \in \mathcal{D}_b)$ by features extracted from an image $I \in \mathcal{D}_b$ employing Φ_n learned using a dataset of material images $\mathcal{D}_n$, and employing $\Phi_{n'}$ using a dataset of object images $\mathcal{D}_{n'}$, respectively. Then the feature vectors $\mathbf{x}_o$ and

[1] In our experiments, we used VGG-D-16 [65] and the fc7 layer.

$\mathbf{x}_m$ are concatenated to obtain a feature vector $\mathbf{x}_c = [\mathbf{x}_o, \mathbf{x}_m]$.

In the proposed method, we model class discriminative features using a classifier that will be employed on concatenated features $\mathbf{x}_c$ by improving discriminative properties of features $\mathbf{x}_m$ and $\mathbf{x}_o$, for classification of material images belonging to a dataset $\mathcal{D}_b$. For this purpose, we consider each individual feature as a weak classifier as suggested in [76]. However, we do not explicitly train a weak classifier for each feature. Instead, we first analyze the discriminative property of each individual feature belonging to $\mathbf{x}_m$ and $\mathbf{x}_o$.

In CNNs implemented using ReLU activation functions ([41, 65]), the features that are represented in an individual neuron are visualized by analyzing and selecting the images on which the activation value of the neuron is maximized [18, 79]. In our method, we also employ this idea to measure contribution of features to discrimination of classes. Suppose that we are given a set of realizations of concatenated features $\mathcal{C} = \{(x_{c,i}, Y_i)\}_{i=1}^{M_b}$ extracted from $\mathcal{D}_b$, where $x_{c,i}$ denotes a realization of concatenated feature extracted form i-th image in $\mathcal{D}_b$. The j-th individual feature $\mathbf{x}_c^{(j)}$ of $\mathbf{x}_c$ is investigated by searching for top K samples that have maximal positive feature values on $\mathbf{x}_c^{(j)}$:

$$\begin{aligned} \mathcal{T}_j &= \{(x_{c,k}, Y_k)\}_{k=1}^{K} \\ s.t. \quad x_{c,k}^{(j)} &> x_{c,i}^{(j)}, \quad x_{c,k}^{(j)} > 0, \\ &\forall (x_{c,k}, Y_k) \in \mathcal{T}_j, \\ &\forall (x_{c,i}, Y_i) \in \mathcal{C} \setminus \mathcal{T}_j. \end{aligned} \tag{4.1}$$

An example of top samples (Girshick et al.[18]) is shown in Figure 4.2.

Note that, the samples belonging to $\mathcal{T}_j$ can reveal some properties of the feature $\mathbf{x}_c^{(j)}$. Then, we compute the discriminative property of $\mathbf{x}_c^{(j)}$ by computing the class entropy in the set $\mathcal{T}_j$ by

$$H_j(Y) = - \sum_{(x_{c,k}, Y_k) \in \mathcal{T}_j} p(Y_k) \log p(Y_k). \tag{4.2}$$

In the next part of the proposed method, we select most discriminative features for discrimination

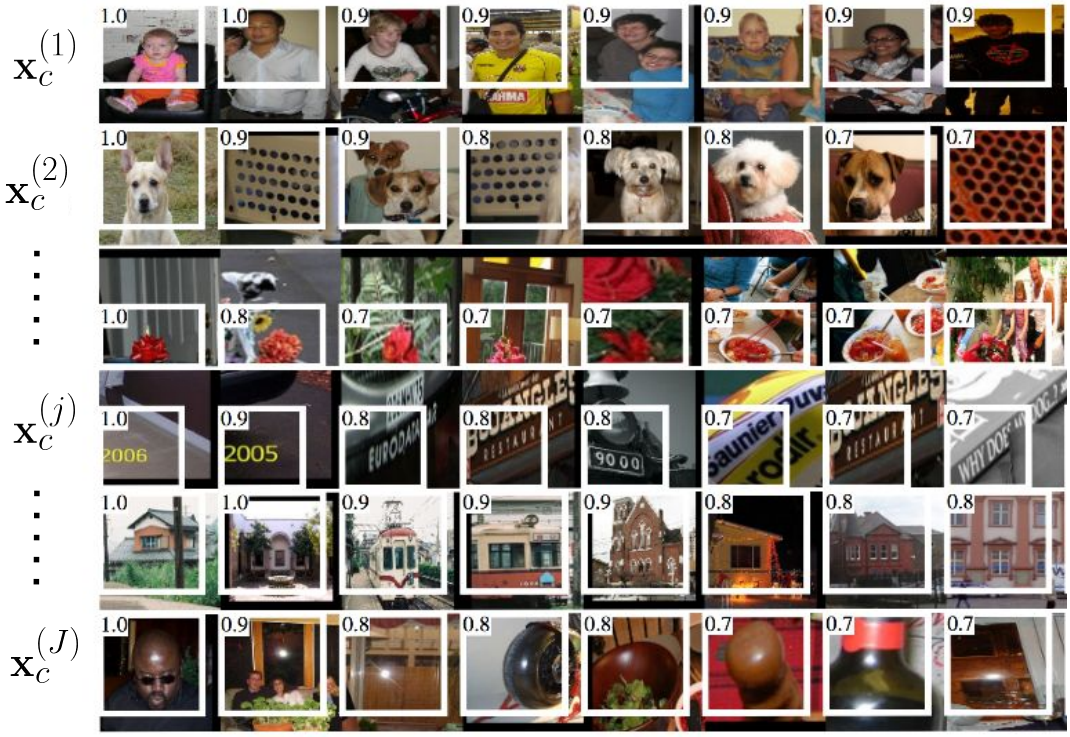


Figure 4.2: An example of top images (Girshick et al.[18]), the white bounding boxes show the receptive fields and activation values. Each row shows top eight images that have maximal activation values for each feature.

of the whole dataset. Given a set of individual features $\mathcal{F} = \{\mathbf{x}_c^{(j)}\}_{j=1}^{J}$ ($J = \dim(\mathbf{x}_c)$), we initialize a set of features to be integrated $\mathcal{S} = \emptyset$, and assign equal weights $w_i = 1$ to each sample $(x_{c,i}, Y_i) \in \mathcal{C}$. Next, we select a feature that has minimum weighted class entropy by

$$\mathbf{x}_c^{(\sigma)} = \arg\min_{\mathbf{x}_c^{(j)} \in \mathcal{F}} \sum_{(x_{c,k}, Y_k) \in \mathcal{T}_j} w_k * H_j(Y), \tag{4.3}$$

and update the sets $\mathcal{S} = \mathcal{S} \cup \{\mathbf{x}_c^{(\sigma)}\}$, $\mathcal{F} = \mathcal{F} \setminus \{\mathbf{x}_c^{(\sigma)}\}$. Then, we penalize the top K samples $(\mathbf{x}_{c,k}, Y_k) \in \mathcal{T}_\sigma$ of the integrated feature using

$$w_k = w_k * (1 + H_\sigma(Y)^{-1}). \tag{4.4}$$

A pseudo-code of our algorithm is shown in Algorithm 1.

Example 4.3.1. We give a simple example to explain our method more intuitively. Suppose that we

Input :

- $\{\Phi_n\}_{n=1}^{N}$: A set of feature extractors each of which is learned using a CNN on a dataset $\mathcal{D}_n$.
- $\mathcal{D}_b = \{(I_i, Y_i)\}_{i=1}^{M_b}$: A dataset of images that will be used for inference of representations.
- T: The number of integrated features.
- K: The number of samples that have maximal activation values for each feature.

Output :

- $\mathcal{S}$: A set of integrated features.

Initialization :

- For each $(I, Y) \in \mathcal{D}_b$, extract features using a feature extractor $\Phi_n(\cdot)$ such that $\mathbf{x}_n = \Phi_n(I), \forall n = 1, 2, \ldots, N$.
- Concatenate $\mathbf{x}_n, \forall n$ and construct $\mathbf{x}_c = [\mathbf{x}_n]_{n=1}^{N}$.
- Define $\mathcal{C} = \{(x_{c,i}, Y_i)\}_{i=1}^{M_b}$, and set equal weight $w_i = 1$ to each sample belonging to $\mathcal{C}$.
- For each individual feature of $\mathbf{x}_c$, define the set $\mathcal{F} = \{\mathbf{x}_c^{(j)}\}_{j=1}^{J}$ ($J = \dim(\mathbf{x}_c)$), and $\mathcal{S} = \emptyset$.

for $j \leftarrow 1$ **to** J **do**
    Construct K samples that have maximal feature values on $\mathbf{x}_c^{(j)}$ using (4.1);
    Compute class entropy on the set of top K samples using (4.2);
end
for $t \leftarrow 1$ **to** T **do**
    Normalize the sample weights: $w_i^{t+1} = \frac{w_i^t}{\sum_{j=1}^{M_b} w_j^t}, \forall i$;
    Select the feature that minimizes the weighted class entropy using (4.3);
    Penalize the samples using (4.4);
end

Algorithm 1: Integration of features extracted using deep representations of CNNs.

are given two features $\mathbf{x}_c^{(1)}$ and $\mathbf{x}_c^{(2)}$. We first select top 10 samples for each feature, and obtain their labels:

	Label									
$\mathbf{x}_c^{(1)}$	1	1	1	1	1	1	1	1	1	2
$\mathbf{x}_c^{(2)}$	1	1	1	1	2	2	2	3	3	3

Then, we compute the weighted class entropy for the *top* samples using (4.2) and (4.3) according to their weights and labels:

	Label, Weight									
$\mathbf{x}_c^{(1)}$	1,w_1	1,w_2	1,w_3	1,w_4	1,w_5	1,w_6	1,w_7	1,w_8	1,w_9	2,w_{10}
$\mathbf{x}_c^{(2)}$	1,w_1	1,w_2	1,w_3	1,w_4	2,w_5	2,w_6	2,w_7	3,w_8	3,w_9	3,w_{10}

Next, we select the feature that minimizes the weighted class entropy using (4.2), and penalize the top samples of the selected feature ($\mathbf{x}_c^{(1)}$) using (4.4):

	Label,Weight									
$\mathbf{x}_c^{(1)}$	1,$w_1\uparrow$	1,$w_2\uparrow$	1,$w_3\uparrow$	1,$w_4\uparrow$	1,$w_5\uparrow$	1,$w_6\uparrow$	1,$w_7\uparrow$	1,$w_8\uparrow$	1,$w_9\uparrow$	2,$w_{10}\uparrow$
$\mathbf{x}_c^{(2)}$	1,w_1	1,w_2	1,w_3	1,w_4	2,w_5	2,w_6	2,w_7	3,w_8	3,w_9	3,w_{10}

After repeating the previous steps by T times, we can select top T features.

4.4 Experimental Analysis

4.4.1 Datasets

In the experiments, we used three benchmark datasets (see Table 4.1). FMD dataset [62] consists of 10 material categories each of which contains 100 images. In the FMD, the samples were selected manually from Flickr covering different illumination conditions, colors, texture and their composition. MINC is a large scale material recognition dataset [3]. It contains 3 million images belonging to 23 categories.

In addition, we introduce a new dataset called EFMD which is an extended version of the FMD. EFMD consists of the same categories that are used in FMD, each of which contains 1,000 images. Thus, the size of EFMD is 10 times larger than the size of FMD. While constructing EFMD, we aim to make it as *similar* to FMD as possible in the context of visual perception and recognition. Specifically, we first pick 100,000 images from Flickr by text searching. Then we ask Amazon Mechanical Turkers to choose the images that are *similar* to FMD images. In each Human Intelligence Task, we present 10 candidate images to a Turker along with three good examples (FMD images) and six bad examples, and ask the Turker to select good ones. Publishing each Human Intelligence Task for three Turkers, we only select images that are selected by all the Turkers. We then manually crop these images to adjust the scale of an object appearing in the images. Note that the images belonging to FMD will not be selected and merged into EFMD to make sure that there is no overlapping between them. In Figure 4.3, we list some example images of FMD and EFMD.

Each of FMD and EFMD datasets is randomly split into two subsets of equal size; one is used

Datasets	# Categories	# Samples	# Train	# Test	# Splits
FMD	10	1,000	500	500	10
EFMD	10	10,000	5,000	5,000	10
MINC	23	3 Million	Training	Validation/Test	-

Table 4.1: Datasets used in experiments.

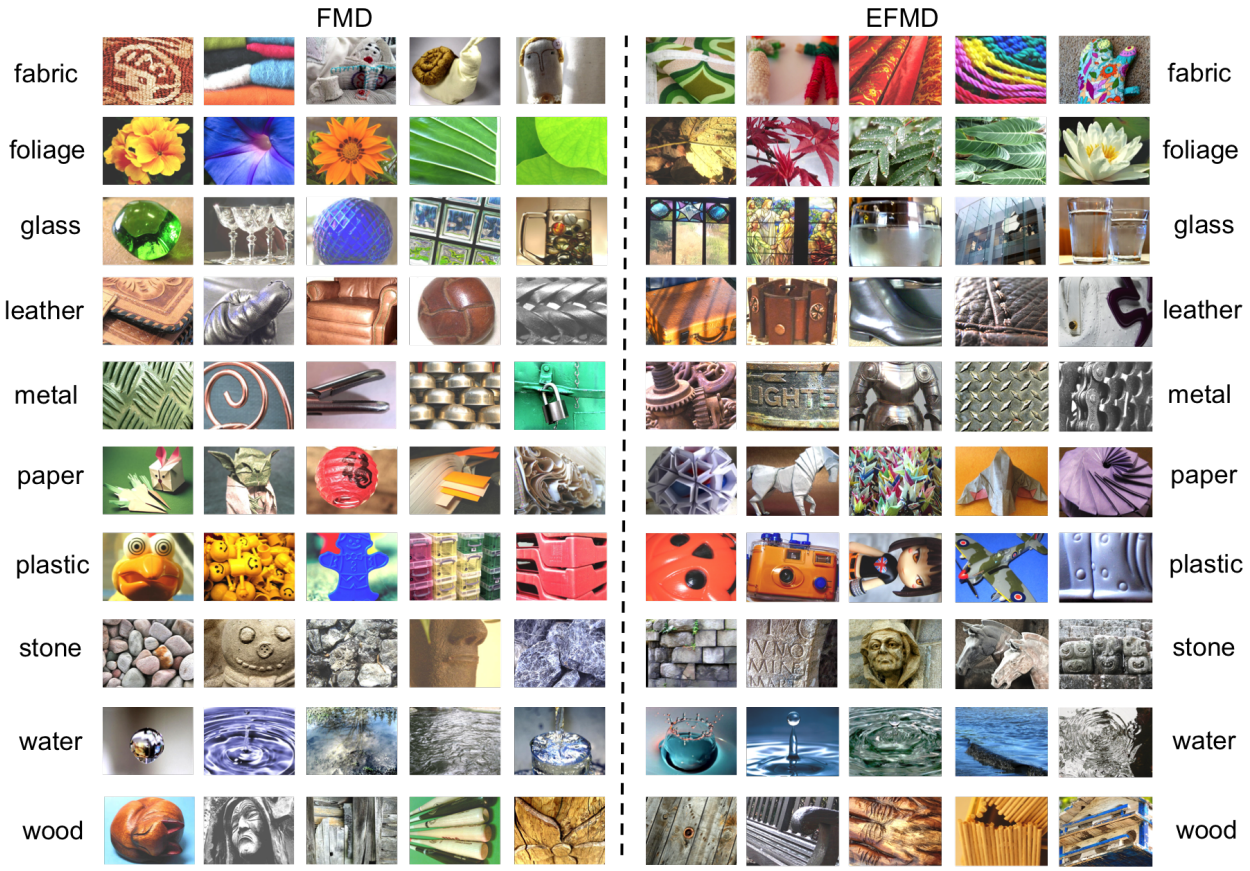


Figure 4.3: Some example images belonging to the FMD and EFMD.

for training and the other is used for testing. This scenario is performed 10 times and the average classification accuracy was reported. For MINC dataset, we directly use the originally provided train, validation and test sets. The details of each dataset and the corresponding experimental setup are provided in Table 4.1.

4.4.2 Details of Experimental Setups

In our experiments, we consider four different tasks, which we will refer to as FMD, FMD-2, EFMD, and MINC(-val/test), respectively. In each of them, we construct a material representation Φ_n and an object representation Φ'_n following *different* procedures as described below.

In all tasks, we initialize the experiments using two CNNs pre-trained on MINC and ILSVRC2012 [59], for which we use publicly available pre-trained Caffe [34] models of VGG-D consisting of 16 layers [65], as shown in Figure 4.4a. The structure of VGG-D is shown in Figure 1.7a. We then

Task	Samples used for fine-tuning	Number of samples
FMD	FMD(train)	500
FMD-2	FMD(train) + EFMD(all)	10,500
EFMD	EFMD(train)	5,000
MINC-val/test	None	N/A

Table 4.2: Two CNNs pretrained on MINC(M) and ILSVRC2012(O) are fine-tuned for each task using the datasets shown below, from which Φ_n and $\Phi_{n'}$ are obtained, respectively.

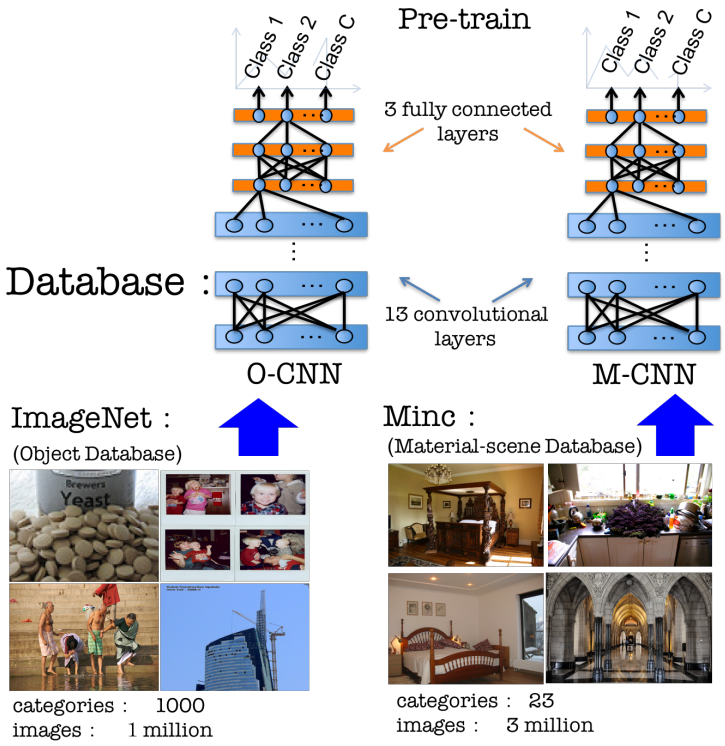
fine-tune these CNNs for the tasks FMD, FMD-2, and EFMD using different samples as shown in Table 4.2. In the fine-tuning phase, weights of Conv1-Conv4 (in total 9) layers are fixed, and weights of the higher layers are updated as shown in Figure 4.4b. We observed that this method used for fixing/updating layers performs the best for FMD and EFMD. We do not conduct fine-tuning for MINC dataset, since MINC is a larger dataset enabling us to train a CNN from scratch. Thus we used the pre-trained model as is.

In the inference phase of each task, we extract material features $\mathbf{x}_m \triangleq \Phi_n(I \in \mathcal{D}_b)$ and object features $\mathbf{x}_o \triangleq \Phi_{n'}(I \in \mathcal{D}_b)$ using the representations learned at the fc7 layer (after ReLu) of the two CNNs, respectively. The concatenated object and material features $\mathbf{x_c} = [\mathbf{x}_o, \mathbf{x}_m]$ are used to obtain a set of integrated features $\mathbf{x_s}$. We set K to 10% of the size of training images, and T to 3,000 throughout the experiments. The integrated features are fed into support vector machines (SVM) with radial basis function (RBF) kernels (parameter: $\gamma = 0.0001, C = 1000$) [14] for training and testing of classifiers. The overall experimental setups are given in Figure 4.4.

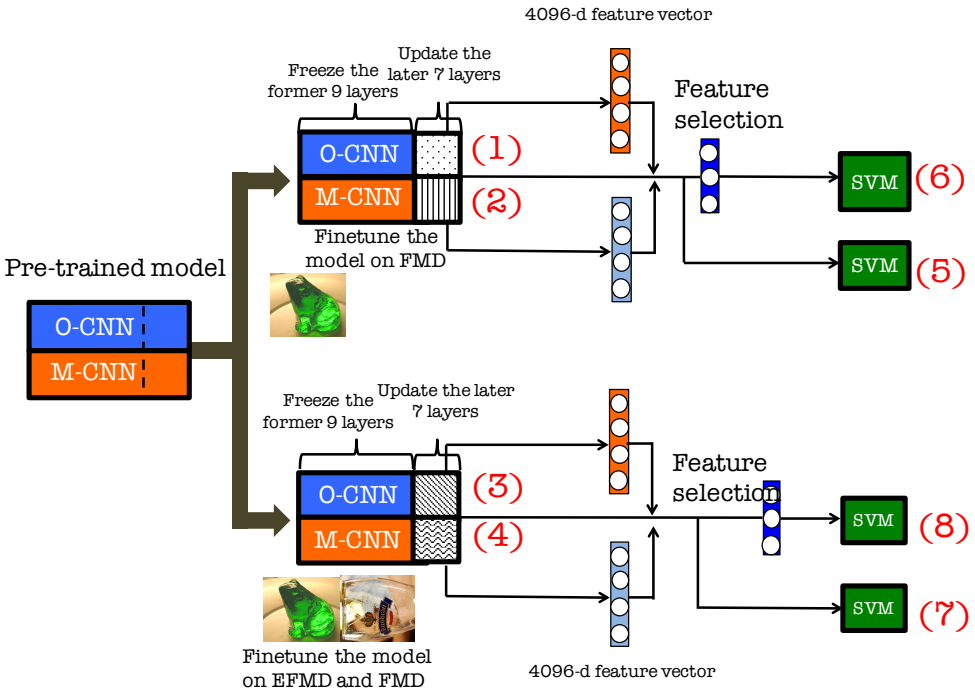
4.4.3 Performance Analysis

The results are shown in Table 4.3; i) *O*, ii) *M*, iii) *MO* and iv) *SMO* columns denote the results for the case where only i) object features ($\mathbf{x}_o$), ii) material features ($\mathbf{x}_m$), iii) concatenated features ($\mathbf{x_c} = [\mathbf{x}_o, \mathbf{x}_m]$), and iv) the features integrated by proposed method ($\mathbf{x_s}$), are used, respectively.

First, we observe that the concatenated features (*MO*) provide better performance than the individual material (*M*) and object (*O*) features for all the tasks, indicating that object features



(a) Two VGG-16 are pretrained on ILSVRC2012 (O-CNN) and MINC (M-CNN).



(b) Transfer learning and feature selection.

Figure 4.4: Experimental setups.

Task	*M* (%)	*O* (%)	*MO* (%)	*SMO* (%)
FMD	80.4 ± 1.9	79.6 ± 2.1	79.1 ± 2.5	$\mathbf{82.3 \pm 1.7}$
FMD-2	82.5 ± 2.0	82.9 ± 1.6	83.9 ± 1.8	$\mathbf{84.0 \pm 1.8}$
EFMD	88.7 ± 0.2	88.8 ± 0.3	$\mathbf{89.7 \pm 0.13}$	$\mathbf{89.7 \pm 0.16}$
MINC-val	82.45 [3]	68.17	83.48	**83.93**
MINC-test	82.19 [3]	68.04	83.12	**83.60**

Table 4.3: Performance (accuracy) comparison for different tasks. *M*: Material features learned using MINC. *O*: Object features learned using ILSVRC2012. *MO*: Concatenated material and object features ($\mathbf{x}_c \in \mathcal{F}$). *SMO*: Features integrated using the proposed method ($\mathbf{x}_c \in \mathcal{S}$).

contribute *significantly* to material classification. Next, features integrated by our method (*SMO*) further boost the performance obtained using the concatenated features for the task FMD and MINC-val/test, indicating the effectiveness of the proposed method. However, there is practically no difference in accuracy between *MO* and *SMO* for the task FMD-2 and EFMD.

It should also be noted that $82.3 \pm 1.7\%$ obtained for FMD and $84.0 \pm 1.8\%$ obtained for FMD-2 are better than the state-of-the-art performance reported in the literature. A method proposed in [11] achieves $82.4 \pm 1.5\%$ on FMD, where they combine fc7 features and Fisher Vectors (FV) features pooled from the conv5_3 layer of VGG-D-19. However, we should note that their results are obtained using different setups, and thus we cannot make direct comparisons. Differences between the setups are that they do not use MINC or EFMD dataset, whereas they use a multi-scale approach, i.e., images with different scales are used to compute FV features. It is nonetheless important to note that $84.0 \pm 1.8\%$ is the best performance obtained for FMD, and is close to human vision (84.9%) [63]. Additionally, $83.93\%/83.60\%$ obtained for MINC val/test also outperforms the previous results [3]. In their work, $83.83\%/83.4\%$ are obtained using GoogLeNet, and $82.45\%/82.19\%$ are achieved using VGG-D-16. Note that their CNN models are pre-trained using ILSVRC2012[59].

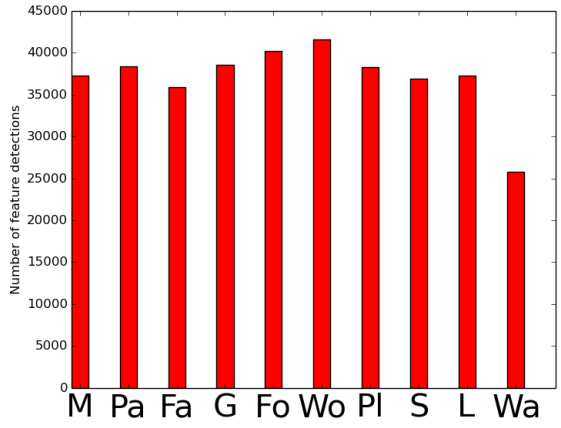
Next, we observe that the methods perform better using FMD-2 compared to FMD. This result can be attributed to the use of more training samples in FMD-2, where 10,000 EFMD samples, in addition to 500 FMD samples, are used for training. This observation also indicates the *similarity* between images belonging to EFMD and FMD, which is consistent with our intention for design of

EFMD. Moreover, we obtain higher accuracy for methods that are trained using a half of EFMD, and tested on the rest of EFMD. Thus, we can consider that features learned using EFMD are *more separable and discriminative* compared to features learned using FMD.

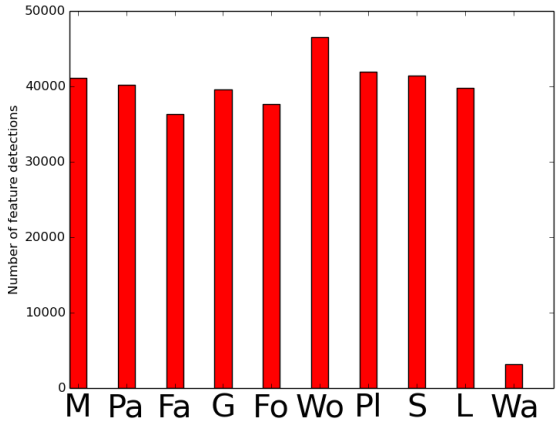
We analyze this observation by measuring statistical properties of representations learned using FMD and EFMD datasets, and decisions of classifiers employed on the representations. In order to analyze *shareability* of class representations among features, we first compute distribution of feature detections on Top-K samples that are computed in (4.1), and employed for computation of entropy (4.2) for concatenated $\mathbf{x}_c^{(j)} \in \mathcal{F}$, and integrated $\mathbf{x}_c^{(j)} \in \mathcal{S}$ features. The results are averaged over 10-splits and provided in Figure 4.5. The results show that class representations are *better shared* for FMD than FMD-2 and EFMD. For instance, the number of detections on the samples belonging to the water class is ~ 25000 for $\mathbf{x}_c^{(j)} \in \mathcal{F}$ and ~ 6000 for $\mathbf{x}_c^{(j)} \in \mathcal{S}$ for FMD, however, ~ 3000 for $\mathbf{x}_c^{(j)} \in \mathcal{F}$ and ~ 700 for $\mathbf{x}_c^{(j)} \in \mathcal{S}$ for FMD-2. In other words, the representations learned using FMD dataset represent water class better than the representations learned using FMD-2 dataset.

In addition, the results show that the proposed method (*SMO*) does *not* improve the performance for concatenated features (*MO*) using FMD-2 and EFMD. We explore this result by first analyzing *shareability* of class representations among features. For this purpose, we compute average entropy $\hat{H}^{\mathcal{F}} = \frac{1}{|\mathcal{F}|} \sum_{i=1}^{|\mathcal{F}|} H_i(Y)$ and $\hat{H}^{\mathcal{S}} = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} H_i(Y)$. The results given in Table 4.4 show that entropy decreases as we employ the EFMD for training CNNs. However, we observe that the difference between entropy values of $\hat{H}^{\mathcal{F}}$ and $\hat{H}^{\mathcal{S}}$ is ~ 0.21 for FMD and FMD-2, and ~ 0.22 for EFMD.

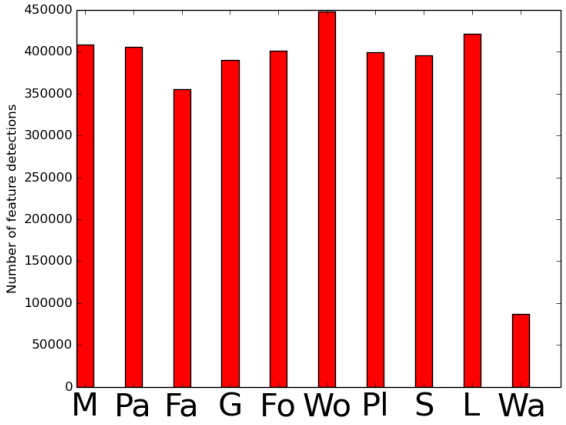
In order to further analyze the shareability of class representations, we computed diversity of decisions of classifiers employed on features ($\mathbf{x}_m$ and $\mathbf{x}_o$) extracted using individual representations of objects (O) and materials (M). For this purpose, we employed five statistical measures [43, 42], namely i) inter-rater agreement (κ) to measure the level of agreement of classifiers while correcting for chance, ii) Q statistics to measure statistical dependency of classifiers, iii) the correlation coefficient ρ, iv) Kohavi-Wolpert variance to measure variance of agreement, v) measurement of disagreement, vi) generalized diversity to measure causal statistical diversity of classifiers, and vii) coincidence failure diversity to measure whether all classifiers are simultaneously either correct or wrong.



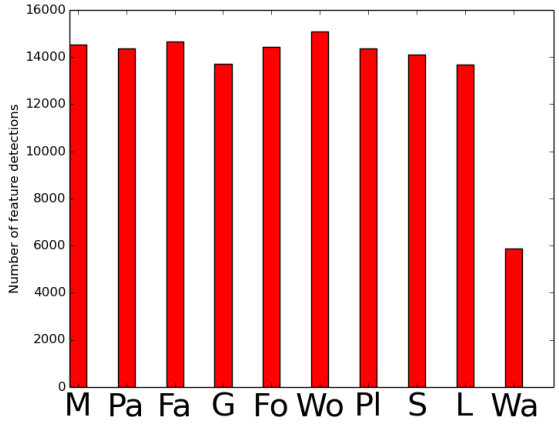
(a) $\mathbf{x}_c^{(j)} \in \mathcal{F}$ for FMD.



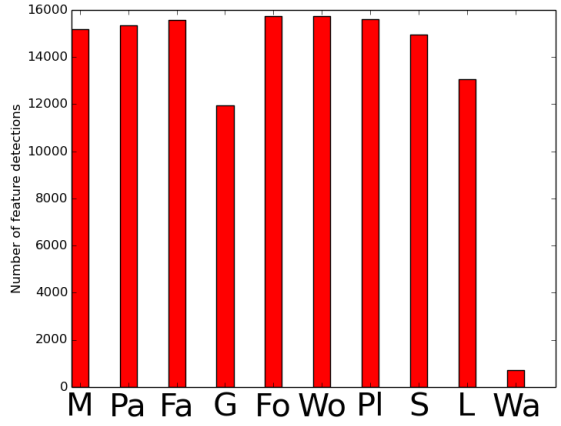
(b) $\mathbf{x}_c^{(j)} \in \mathcal{F}$ for FMD-2.



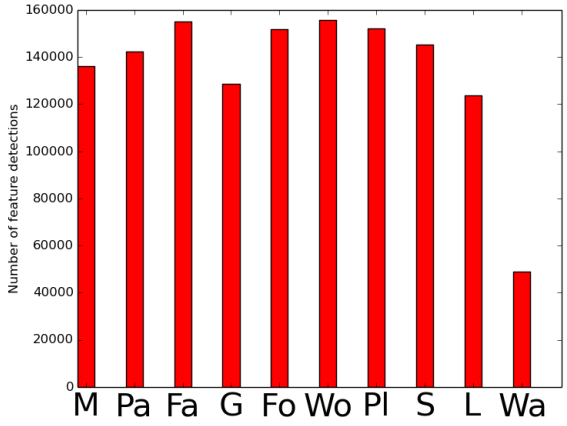
(c) $\mathbf{x}_c^{(j)} \in \mathcal{F}$ for EFMD.



(d) $\mathbf{x}_c^{(j)} \in \mathcal{S}$ for FMD.



(e) $\mathbf{x}_c^{(j)} \in \mathcal{S}$ for FMD-2.



(f) $\mathbf{x}_c^{(j)} \in \mathcal{S}$ for EFMD.

Figure 4.5: Distribution of feature detections on Top-K samples among classes that are computed in (4.1). Distribution of 8192-dimensional features is given for (a) FMD, (b) FMD-2, and (c) EFMD; and that of 3000-dimensional integrated features is given for (d) FMD, (e) FMD-2, and (f) EFMD. M:*metal*, Pa:*paper*, Fa:*fabric*, G:*glass*, Fo:*foliage*, Wo:*wood*, Pl:*plastic*, S:*stone*, L:*leather*, Wa:*water*.

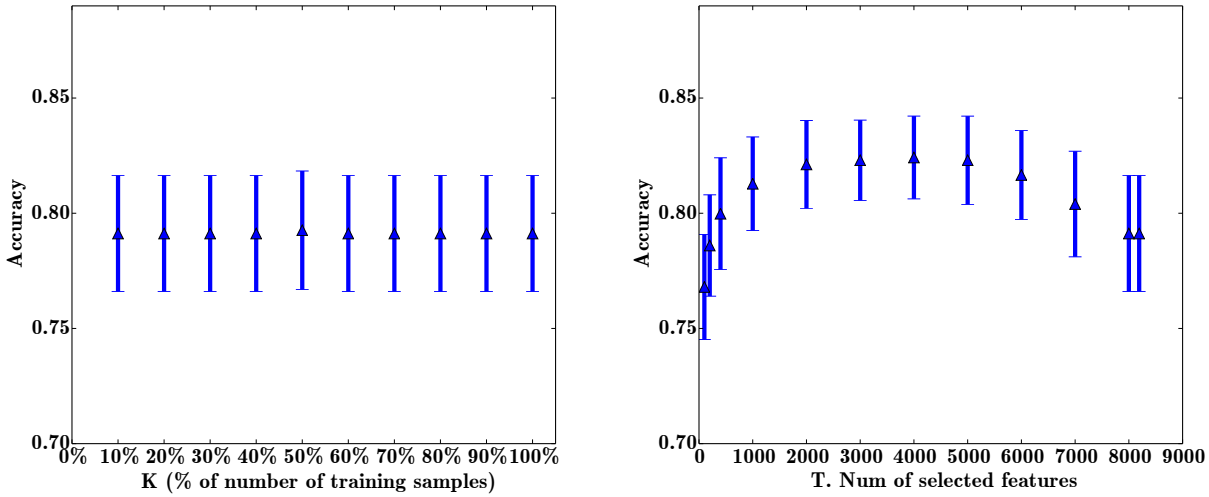
Diversity Measures	FMD	FMD-2	EFMD
MO ($\hat{H}^{\mathcal{F}}$)	2.40	2.29	2.21
SMO ($\hat{H}^{\mathcal{S}}$)	2.19	2.08	1.99
κ ($\downarrow$)	0.6471	0.7103	0.7116
Q Statistics ($\downarrow$)	0.9860	0.9944	0.9996
The correlation coefficient ρ ($\downarrow$)	0.6556	0.7145	0.7193
Kohavi-Wolpert Variance ($\uparrow$)	0.0070	0.0043	0.0002
Disagreement ($\uparrow$)	0.0279	0.0172	0.0008
Generalized Diversity ($\uparrow$)	0.3383	0.2892	0.2795
Coincidence Failure Diversity ($\uparrow$)	0.5000	0.4314	0.4278

Table 4.4: Average entropy values of distributions of detections of concatenated and integrated features (MO and SMO). In addition, we provide a diversity analysis of decisions of classifiers employed on individual feature sets (O and M), where $\downarrow$ ($\uparrow$) indicates that the smaller (larger) the measurement, the larger the diversity.

The results are given in Table 4.4. Note that the performance boosts using the proposed method as the diversity of the classifiers employed on the features extracted using individual representations (O and M) increases. For instance, the diversity of classifiers employed on the feature sets extracted using FMD dataset is larger than that of the classifiers employed using FMD-2 and EFMD datasets. In addition, the performance difference between classifiers employed on the integrated features (SMO) and the concatenated features (MO) is 3.2%, 0.1% and 0% for FMD, FMD-2 and EFMD, respectively (see Table 4.3). Therefore, we observe that the performance boost obtained using our proposed method increases as diversity of the classifiers employed on individual representations increases.

4.4.4 Robustness analysis

We analyze how the number of Top-K selected samples and the number (T) of integrated features affect the classification performance on the FMD dataset; see Figure 4.6a and Figure 4.6b. In Figure 4.6a, the number of integrated features is fixed to 3000. Note that only the samples with positive activation are considered as the top selected samples. For example, if K is selected to cover



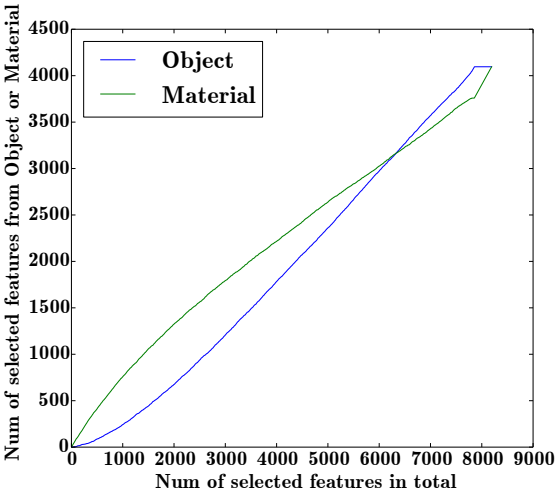
(a) K vs. Accuracy.

(b) T vs. Accuracy.

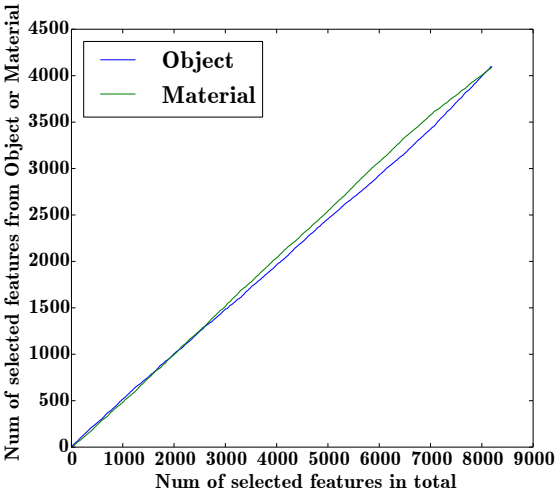
Figure 4.6: Analysis of classification performance for (a) different number of Top-K samples, and (b) different number (T) of integrated features on the FMD.

100% of the samples, then the ratio of samples selected by the algorithm may be less than 100%. This can be observed if most of the activation values of FMD is 0. In order to analyze the effect of T on the performance, K is fixed to 10%. Although selection of $T = 3000$ features provides the best performance, less number of integrated features, e.g. 100-400, provides comparable accuracy.

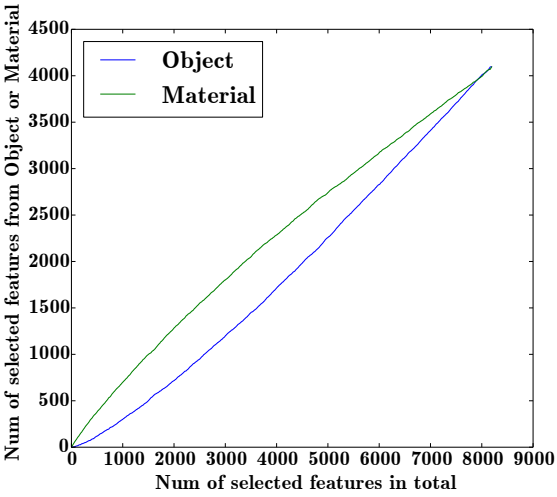
We also analyze the relationship between features that are extracted using representations of materials and objects for feature selection. We leave out one split of FMD, and record the number of integrated material and object features using our feature selection method. A comparative result is shown in Figure 4.7. As we can see from the figure, there is a gap between the number of selected material and object features in FMD (Figure 4.7a), EFMD (Figure 4.7c) and MINC (Figure 4.7d). However, in Figure 4.7b, we observe that the difference between the number of material and object features is less for FMD2 compared to the other datasets. In FMD2, both object and material features are fully fine-tuned using 10,000 EFMD images. If the fine-tuned features are discriminative for material classification, then the features are equally selected by the proposed method using FMD2. On the other hand, if the features are not *sufficiently* fine-tuned, then material features may be more discriminative than object features for material recognition. For instance, Table 4.3 shows the results for the case where material and object features are not fine-tuned for MINC-val. We obtain 82.45%



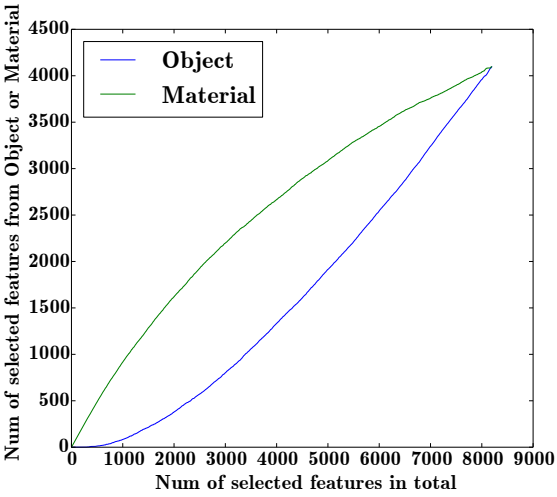
(a) FMD.



(b) FMD2.



(c) EFMD.



(d) MINC.

Figure 4.7: Comparison of number of object and material features belonging to the set of selected features. We show the number of selected object and material features in (a) FMD, (b) FMD2, (c) EFMD, and (d) MINC.

using individual material features, while we obtain 68.17% using individual object features. Hence, we may obtain larger number of material features than object features as shown in Figure 4.7d.

4.5 Conclusions

In this work, we propose a method to integrate deep features extracted from multiple CNNs trained on images of materials and objects for material recognition. For this purpose, we first employ a feature selection and integration method to analyze and select deep features by measuring their contribution to representation of material categories. Then, the integrated features are used for material recognition using SVM classifiers. In the experimental results, we obtain state-of-the-art performance by employing the features integrated using the proposed method on several benchmark datasets. In future work, we plan to investigate theoretical properties of the proposed methods for integration of deep representations using various deep learning algorithms such as autoencoders, to perform other tasks such as scene analysis, image classification and detection.

Chapter 5

Conclusions

In this thesis, we propose information theoretic methods to analyze Deep Neural Networks (DNNs) for different learning tasks: unsupervised learning, supervised learning and transfer learning.

We first introduce an information theoretic framework for unsupervised learning of Auto-Encoders. To estimate the entropy of encoding variable $H(\mathbf{z})$, we propose a non-parametric method. Additionally, we give an information theoretic view of VAEs, which suggests that VAEs can be seen as parametric methods that estimate entropy. Experimental results show that the proposed IPAEs have more degree of freedom in terms of representation learning of features drawn from complex distributions such as Mixture of Gaussians, compared to VAEs.

In the second chapter, we introduce a binary tree architecture to truncate architecture of wide networks considering their parameter size and accuracy trade-off. In the proposed architecture, the width of each layer is incrementally reduced from lower layers to higher layers. Also, features obtained at different layers are concatenated to form the output of our architecture. In our experiments, the networks, which are designed using the proposed architecture, can obtain better parameter size and accuracy trade-off on several benchmark datasets compared to baseline networks endowed with conventional architectures.

For analysis of transfer learning tasks of DNNs, we suggest a method to select and integrate multiple features obtained using different pretrained models of DNNs trained on images of materials and objects for material recognition in Chapter 3. We first employ a feature selection and integration

method to analyze and select deep features by measuring their contribution to representation of material categories. The contribution of a given feature is computed by the entropy of class distributions for a sample set which are best represented by the given feature. Then, the integrated features are used for material recognition using SVM classifiers. In the experimental results, the proposed method obtain state-of-the-art performance on several benchmark datasets by employing the integrated features.

In our future work, we will consider application of our information theoretic methods to other tasks. For instance, we will extend the proposed non-parametric information regularization method to state-of-the-art generative models, such as generative adversarial networks [20]. In addition, BitNets can be employed for object detection tasks. For transfer learning, we consider other computer vision tasks, such as scene analysis.

Bibliography

[1] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *Proceedings of International Conference on Learning Representations*, 2017.

[2] R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, 5(4):537–550, Jul 1994.

[3] Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. Material recognition in the wild with the materials in context database. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 3479–3487, June 2015.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[5] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[6] Yoshua Bengio, Olivier Delalleau, and Clarence Simard. Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4):449–467, 2010.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[8] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1–2):245 – 271, 1997.

[9] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[10] Gal Chechik, Amir Globerson, Naftali Tishby, and Yair Weiss. Information bottleneck for gaussian variables. *Journal of Machine Learning Research*, 6:165–188, 2005.

[11] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, and Andrea Vedaldi. Deep filter banks for texture recognition, description, and segmentation. *CoRR*, abs/1507.02620, 2015.

[12] Mircea Cimpoi, Subhransu Maji, and Andrea Vedaldi. Deep filter banks for texture recognition and segmentation. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 3828–3836, June 2015.

[13] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units(elus). In *Proceedings of International Conference on Learning Representations*, 2016.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[15] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[16] Luis Gonzalo Sánchez Giraldo and José C. Príncipe. Information theoretic learning with infinitely divisible kernels. In *Proceedings of International Conference on Learning Representations*, 2013.

[17] Luis Gonzalo Sánchez Giraldo and José C. Príncipe. Rate-distortion auto-encoders. In *Proceedings of International Conference on Learning Representations*, 2014.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 580–587, June 2014.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2010.

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014.

[21] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. Maxout networks. In *International Conference on Machine Learning*, 2013.

[22] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Machine Learning Research*, 3:1157–1182, March 2003.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of International Conference on Computer Vision*, 2015.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2016.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of European Conference on Computer Vision*, 2016.

[26] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *Proceedings of International Conference on Learning Representations*, 2017.

[27] Geoffrey E Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Proceedings of Advances in Neural Information Processing Systems 6*, pages 3–10. 1994.

[28] Andrew G. Howard. Some improvements on deep convolutional neural network based image classification. In *Proceedings of International Conference on Learning Representations*, 2014.

[29] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *Proceedings of European Conference on Computer Vision*, 2016.

[30] Yani Ioannou, Duncan P. Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. arXiv:1603.01250, 2016.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of International Conference on Machine Learning*, 2015.

[32] Ozan Irsoy and Ethem Alpaydin. Autoencoder trees. In *Asian Conference on Machine Learning*, 2014.

[33] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of British Machine Vision Conference*, 2014.

[34] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[35] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *International Conference on Machine Learning*, pages 121–129. Morgan Kaufmann, 1994.

[36] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. In *Proceedings of International Conference on Learning Representations*, 2014.

[37] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273 – 324, 1997.

[39] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò . Deep neural decision forests. In *Proceedings of International Conference on Computer Vision*, 2015.

[40] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, 2012.

[42] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[43] Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, May 2003.

[44] N. Kwak and Chong-Ho Choi. Input feature selection for classification problems. *IEEE Transactions on Neural Networks*, 13(1):143–159, Jan 2002.

[45] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *Proceedings of International Conference on Learning Representations*, 2017.

[46] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. 2001.

[47] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2015.

[48] Chen-Yu Lee, Patrick Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2016.

[49] Lewis and D. David. Feature selection and feature extraction for text categorization. In *Proceedings of the Workshop on Speech and Natural Language*, pages 212–217, 1992.

[50] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *Proceedings of International Conference on Learning Representations*, 2014.

[51] Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In *Proceedings of Advances in Neural Information Processing Systems 28*. 2015.

[52] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, 2014.

[53] Takehiro Nagai, Toshiki Matsushima, Kowa Koida, Yusuke Tani, Michiteru Kitazaki, and Shigeki Nakauchi. Temporal properties of material categorization and material rating: visual vs non-visual material features. *Vision Research*, 115, Part B:259 – 270, 2015.

[54] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of International Conference on Machine Learning*, 2010.

[55] B. Olshausen and D. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

[56] Razvan Pascanu, Guido Montúfar, and Yoshua Bengio. On the number of inference regions of deep feed forward networks with piece-wise linear activations. In *Proceedings of International Conference on Learning Representations*, 2014.

[57] H. Peng, Fulmi Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, Aug 2005.

[58] Juha Reunanen. Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3:1371–1382, March 2003.

[59] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, pages 1–42, April 2015.

[60] Gabriel Schwartz and Ko Nishino. Automatically discovering local visual material attributes. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, June 2015.

[61] Ohad Shamir, Sivan Sabato, and Naftali Tishby. Learning and generalization with the information bottleneck. In *Algorithmic Learning Theory: 19th International Conference*, 2008.

[62] L. Sharan, R. Rosenholtz, and E. Adelson. Material perception: What can you see in a brief glance? *Journal of Vision*, 14(9), 2014.

[63] Lavanya Sharan, Ce Liu, Ruth Rosenholtz, and Edward H. Adelson. Recognizing materials using perceptually inspired features. *International Journal of Computer Vision*, 108(3):348–371, 2013.

[64] Lavanya Sharan, Ruth Rosenholtz, and Edward H. Adelson. Accuracy and speed of material categorization in real-world images. *Journal of Vision*, 14(10), 2014.

[65] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations*, 2015.

[66] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *Proceedings of International Conference on Learning Representations workshop*, 2015.

[67] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[68] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv:1602.07261, 2016.

[69] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2015.

[70] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2016.

[71] Matus Telgarsky. Benefits of depth in neural networks. abs/1602.04485, 2016.

[72] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. In *37th Allerton Conference on Communication and Computation*, 1999.

[73] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015.

[74] Kari Torkkola. Feature extraction by non parametric mutual information maximization. *Journal of Machine Learning Research*, 3:1415–1438, 2003.

[75] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[76] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001.

[77] Jason Weston, André Elisseeff, Bernhard Schölkopf, and Mike Tipping. Use of the zero norm with linear models and kernel methods. *Machine Learning Research*, 3:1439–1461, March 2003.

[78] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of British Machine Vision Conference*, 2016.

[79] MatthewD. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proceedings of European Conference on Computer Vision 2014*, volume 8689, pages 818–833. 2014.

[80] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, June 2015.

Acknowledgments

This work was carried out under the supervision of Professor Takayuki Okatani and Professor Mete Ozay.

First, I would like to express my sincere appreciation to my advisors Professor Takayuki Okatani and Mete Ozay for their priceless advice on my research and for their encouragement at my difficult days. Professor Okatani has been very supportive during the last three years and has given me the freedom to pursue what I am interested in. Professor Ozay is an enthusiastic and talent researcher. He has spent several sleepless nights with students to catch the deadline of submission. I have learn much invaluable knowledge from the two professors, including writing skills, ways of scientific thinking and mathematical theories etc. In addition to academic knowledge, my advisors have provided excellent examples of being a respectful and successful man which can also be really helpful to my future career. It is an honor to be their student.

I would also like to thank my friends who have been helping me through all kind of difficulties.

Our lab members also help me a lot during the preparation of this thesis. So I want to thank Zhun Sun, Pongsate Tangseng, Xing Liu, Hiroto Date, Hu Junjie, Liang Xu, Han Zou, Shao MingZhen, Wang Haichao, Shuang Liu etc.

I would like to give special thanks to Liu Yexin who is a kind girl and helps me a lot on my difficult days.

Lastly, I want to thank my family for their unconditional love, for my parents who raise me and for my brother who cares me.

Thank you all!