이학박사 학위논문

Deep Neural Network Latent Space Analysis using Decision Boundary and Attention Style CapsuleNet

(결정 경계와 어텐션 캡슐 네트워크를 이용한 잠재공간 특성백터 분석)

2022년 2월

서울대학교 대학원

협동과정 계산과학전공

서 현

Deep Neural Network Latent Space Analysis using Decision Boundary and Attention Style CapsuleNet

(결정 경계와 어텐션 캡슐 네트워크를 이용한 잠재공간 특성백터 분석)

지도교수 강 명 주

이 논문을 이학박사 학위논문으로 제출함

2021년 10월

서울대학교 대학원

협동과정 계산과학전공

서 현

서 현의 이학박사 학위논문을 인준함

2021년 12월

위 원 장 _____ (인)

부 위 원 장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

Deep Neural Network Latent Space Analysis using Decision Boundary and Attention Style CapsuleNet

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Hyun Seo

Dissertation Director : Professor Myungjoo Kang

Interdisciplinary Program of
Computational Science and Technology
Seoul National University

February 2022

Abstract

While the deep learning model produces overwhelming performance in many domains, it is not known what latent space the deep learning model embedding, what features it learns, and how it separates features. An accurate understanding of the learning process of deep learning is not perfect until now and is still an open problem.

In this thesis, we try to broaden our understanding of the latent space of the deep neural network in two ways. In the first chapter, we experimentally investigate the relationships with the vision boundary in the latent space of the deep neural network through several toy experiments. The decision boundary is obtained by using and adversarial attack methods in the latent space where deep neural network embeds. We analyze the relationship between the decision boundary and the latent space manual obtained by perturbing the image.

In the second chapter, the characteristics of the latent space is examined by constraining a network architecture design. We propose a new network module called an attention-style capsulenet with improved version of capsulenet. The value of each capsules is perturbed to determine which image feature is trained by the deep neural network.

Key words: Deep Learning, Latent Manifold, Adversarial Attack, Decision Boundary, CapsuleNet

Student Number: 2017-28074

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Chapter 1

Introduction

In recent days, deep neural networks have achieved a state-of-the-art performance in many domains and applications, including vision domain (*i.e* classification, object detection, segmentation and generation[38, 15, 14, 12, 20]), natural language processing domain (*i.e* language translation, sentence generation and speech recognition[34, 29, 5]).

Deep learning methodologies used in a wide range of domains generally learn internal parameters from data through the following processes 1.1. For a more detailed example, the general learning process of the deep neural model based on cnn is as follows.

1. The raw natural data(image) is extracted through a cnn-based model.
2. Convert the extracted feature to suit the task using mlp or another cnn-based model. (*i.e* classification segments)
3. Both the feature extract model and the task model have trainable weight and learn from data

All processes consist of differentiable structures and are trainable. Despite the great achievements in various fields of these deep learning models and methodologies, it is not known how the deep learning model solves the problem due to the nature of the black box.

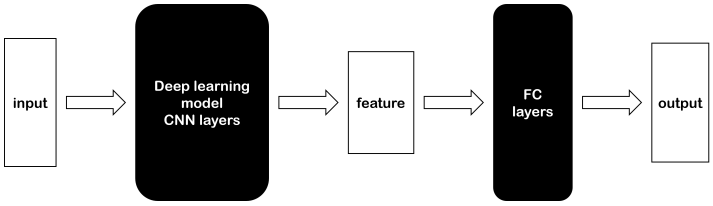


Figure 1.1: Overview of deep learning architecture

There have been several representative attempts to understand the deep learning methodology. One way to understand the deep learning model is to try to interpret the learned model. These methods include [49, 35, 30]. It is a method of visualizing the activation map to determine which features the deep neural network learned and which features influenced the results. The other attempt is universal approximation theorem. Universal approximation theorem means that any function can be approximated using a natural network composed of non-linear activation functions. Theorem was proved in 1989 by George Cybenko[10]. This proof of theorem guarantees that the natural network can approximate any function. However, it does not tell you how to learn the weight of the deep learning network.

While the deep learning model produces overwhelming performance in many domains, it is not known exactly what latent space the deep neural network embedding, what features it learns, and how it separates features. An accurate understanding of the learning process of deep learning is not

perfect until now and is still an open problem.

In this study, We try to broaden our understanding of the feature space learned by deep learning in two ways. In the first chapter, it is an attempt to understand, through several toy experiments, relationships with the vision boundary in the feature space of the deep learning model that has already been learned. The decision boundary is obtained using the manual and adversarial attack methods in the feature space where deep learning embeds. I analyze the relationship between the decision boundary obtained by adversarial attack and the feature space manual obtained by perturbing the image.

In the second chapter, the characteristics of the feature space is examined by constraining the feature space as a architecture design of the deep learning model. We proposed the network architecture which called an attention style capsule capsulenet that was improved by imitating the capsule net. The values of each capsule are perturbed to determine what the actual image space feature does the deep learning model analyzes and maps to the capsule feature space.

Chapter 2

Relation between Feature Manifold and Decision Boundary

There are some hints that led to assumption 1. Intuition is that adversarial attacks can generate adversarial examples in most deep learning models and in most image samples[13]. Here is even a study that suggests that the prediction result can be manipulated with only one pixel[42]. Because the noise is very small, the adversarial examples are located on the manifold around the original feature. It means that the feature manifold and the decision boundary are very close.

Assumption 1. *The latent manifold is locally parallel to the decision boundary.*

Recent study[36] try to explain that there is a decision boundary in

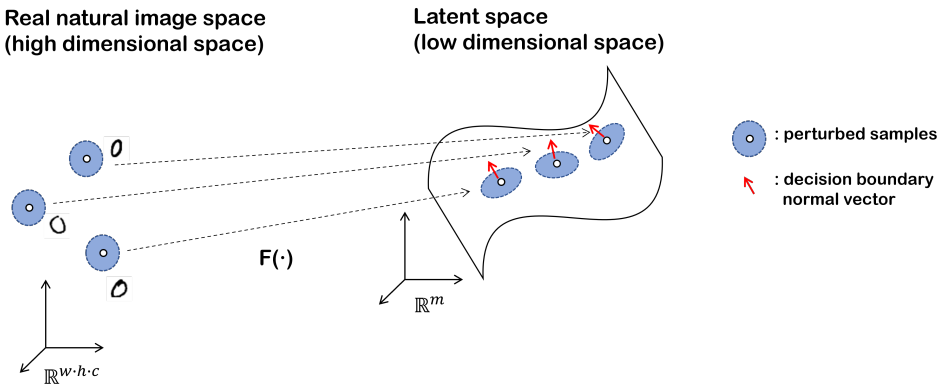


Figure 2.1: Overview of perpendicularity between local manifold and decision boundary normal vector

which the manifold has the same decision boundary in most areas and is dimpled in the part where actual data are present. Figure 2.2 shows the relationship between the decision boundary and the latent space assumed in [36]. When dealing with the classifier of machine learning, when the data of the red class and the blue class are embedded, the division boundary is considered to appear between each class and classify. The [36] argue that in three dimensions, the data manifold looks like Figure 2.2 (middle) in the latency space, the decision boundary is formed very close to the manifold, and only the part with actual data is formed in a dimpled shape to make the dish boundary classify.

In this study, more intuitive and extensive experiments were conducted on this assumption 1.

Figure 2.1 shows the overview of the experiments. First, prepare a deep learning model that has learned classification task. This model is fixed in all following processes. Choose Any source class(i.e number in MNIST dataset),

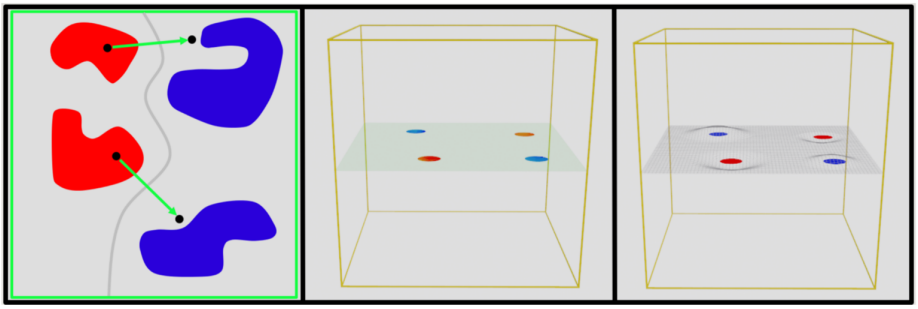


Figure 2.2: Classic visualization of the relationship between the late space and the vision boundary(left). dimpled manifold version[36] visualization(right). decision boundary visualization(middle). Red and blue mean each class group.

and prepare data of the class. Determine an arbitrary target class. Obtain an adversarial example that makes the deep learning model predict with the target label. The adversarial example is feed forward to the deep learning model to obtain the adversarial latent vector. The f_a denotes the original latent vector. Similarly, the original image is feed forward to the deep learning model to obtain original latent vector. The f_o denotes the original latent vector. We can get the direction vector of the decision boundary by subtracting the adversarial latent vector and the original latent vector. We denote this vector d_a.

$$d_a = f_a - f_o \tag{2.0.1}$$

This decision boundary plane determines the model's prediction between the source class and the target class.

The local direction of the latent manifold is obtained in the following way. The manifold vector is obtained in the following way. Several perturbed

samples are made by adding small Gaussian noise to the input image. The σ denotes the Gaussian noise level. These sampled images are feed forward to a deep learning model to obtain a perturbed latent vector. Because these sampled images are perturbed with very small noise, the embedding latent vectors by deep learning model are located on the on-latent manifold. Let the latent vector sampled with noise be f_s. We can get the direction vector of the decision boundary by subtracting the perturbed latent vector and the original latent vector. Let this latent manifold direction vector be d_m.

$$d_m = f_s - f_o \tag{2.0.2}$$

In this experiment, we intend to experimentally demonstrate the assumption 1 by the distribution of angles between direction vectors d_a and d_s according to multiple datasets and source class and target class.

2.1 Related Work

2.1.1 Manifold Hypothesis

The manifold hypothesis assumes that samples of high-dimensional space in a natural data are distributed in the form of low-dimensional space. This assumption is one of the core assumptions of deep learning methodology. Through learning, the deep learning model can embed natural high dimensional data (e.g., image, text, time series data, etc.) through various modules (convolution neural network layer, recurrent neural network layer, long short term memory layer, transformer layer, etc.) to low dimensional space. In ad-

dition, the deep learning model is trained by configuring a model so that output suitable for task (e.g., classification, segmentation, detection). For example, in the case of image classification problems, natural images ranging from size 28 x 28 x 1 to size 1024 x 1024 x 3 are given as inputs. The output of model is one hot encoding vector that classified class of given the input image. Most of these high-dimensional natural image spaces are empty. This fact can be confirmed by the following thought experiment. Based on the MNIST[23] dataset, for example, a vector of 784 dimensions is randomly sampled. The randomly sampled vector is reshaped to a size of $28\times28\times1$ to check as the image. Most of the sampled image will look like meaningless noise. As can be seen from this thought experiment, most of the high-dimensional data spaces in the natural image are empty.

It can be conjectured that there are very few areas where actual data is distributed, and meaningful features can be fully explained by low-dimensional space. The manifold hypothesis means that samples of such a high-dimensional space in a high dimensional natural image can be mapped in the form of a low-dimensional distribution and that data can be sufficiently explained only with a low-dimensional latent vector.

2.1.2 Manifold Learning Methods

Manifold In short, a manifold is a space that can appear as a linear space locally. Figure 2.3 is in a three-dimensional space, but it is actually two-dimensional. The reason is that the rectangle is only rolled up in a round shape, and the cross-section is only a simple plane locally viewed. Unlike in topology, machine learning deals conceptually. The low-dimensional space

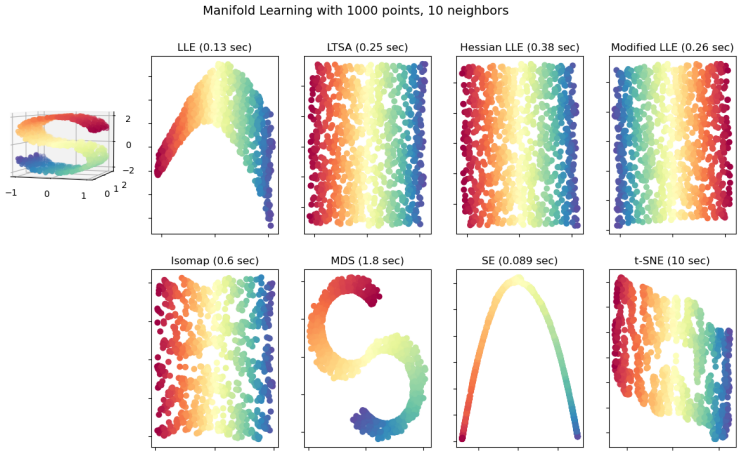


Figure 2.3: Manifold learning methods[28],

inherent in the high-dimensional space is called a manifold. The manifold usually has a nonlinear structure, and it is close to a linear structure if you look around a specific point. In machine learning, the sample in train dataset is on or near the manifold.

Isomap [43] Isomap is one of the oldest and most basic manual learning algorithms. Graph representation is learned based on linear distance between data points. The process is as follows.

1) Find the k-nearest neighbor for each point, calculate the Euclidean distance, and write it in the distance matrix M.

2) Calculate the values of 0 of M using the Floyd algorithm for the shortest path between the two points.

Find the eigen vectors of M, and select d_{low} eigen vectors in the order in which the eigenvalue is large. This eigen vector creates a new low-

dimensional space. Isomap should properly set k to obtain distance matrix M. If it's too big, you have to use the shortest path, but if you set it too small, there will be a problem of using the Euclidean distance, and if it's too small, there will be no path between sample pairs, making it a discontinuous space.

Locally Linear Embedding(LLE) [32] LLE has many similarities to Isomap, but it is a way to improve it. Instead of the distance matrix M, we use a weight matrix W that minimizes the function $l(W)$.

$$l(W) = \sum_{i=1}^{n} \|x_i - \sum_{x_j \in N_{x_i}} w_{ij} x_j\|_2^2 \tag{2.1.1}$$

We denote N_{x_i} is neighbor of x_i. Choose k points adjacent to x_i and find the weight W where the weighted sum is closest to x_i. The weight matrix W obtained in this way can be said to contain nonlinear structural information of data distributed in the original d_low-dimensional space. LLE uses W to find a point y in a new space in the dimension.

$$\phi(\mathbb{X}') = \sum_{i=1}^{n} \|y_i - \sum_{y_j \in N_{y_i}} w_{ij} y_j\|_2^2 \tag{2.1.2}$$

We denote N_{y_i} is neighbor of y_i. In this case, we use the equation 2.1.2 to find the set $\mathbb{X}'$ of points that minimize the objective function ϕ.

t-Distributed Stochastic Neighbor Embedding(t-SNE) [16] It is an algorithm that shows superior performance in various data ex-

periments. t-SNE measures the similarity between x_i and x_j as a conditional probability. This equation resembles a Gaussian distribution, which gives a high probability for samples close to x_i and a probability close to zero for distant samples.

$$p_{j|i} = \frac{exp(-\frac{\|x_i - x_j\|_2^2}{2\sigma_i^2})}{\sum_{k \neq i} exp(-\frac{\|x_i - x_k\|_2^2}{2\sigma_i^2})} \tag{2.1.3}$$

σ_i is determined by the distribution of data around x_i, and the more dense the data, the smaller the value. Based on this equation 2.1.2, redefine the equation to satisfy $p_{j|i} = p_{i|j}$.

$$p_{ij} = p_{ji} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{2.1.4}$$

Similarity in the transformation space uses the t-distribution instead of the Gaussian distribution. If the point in the transformation space is marked y, the similarity between y_i and y_j can be defined as follows.

$$q_{ij} = \frac{(1 + \|y_i - y_j\|_2^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|_2^2)^{-1}} \tag{2.1.5}$$

Since the data structure in the original space must be maintained in the transformation space, it should be close to the P probability distribution of the original space and the Q probability distribution of the transformation space. Therefore, we use KL-divergence of two probability distributions as an objective function. We denote transformed sample set is $\mathbb{X}' = y_1, y_2, \cdots, y_n$.

$$KL(P\|Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} log(\frac{p_{ij}}{q_{ij}}) \tag{2.1.6}$$

To find $\mathbb{X}'$ that minimizes equation 2.1.2, we use the gradient descent method.

2.1.3 Adversarial Attack

Adding very small noise to the input image with adversarial attacks can easily change the output of the deep learning network. It is known that the results can be manipulated even with high confidence score. Adversarial attacks can be classified into black box attacks and white box attacks according to attack scenarios. A targeted adversarial attack is a case of determining a target class to be changed. Untargeted adversarial attacks does not determine the target class, and the prediction of deep neural network will be an arbitrary class, not the original class. An image that adds noise obtained by a adversarial attack to the original image is called a adversarial example.

Many adversarial attack methodologies have been developed, and adversarial defense methodologies have been developed to prevent or mitigate this attack. Some white box adversarial attack methodologies are introduced below.

Fast Gradient Sign Method (FGSM) [13]

In 2014, [13] show how well the adversarial example generated in a very

linear attack scheme deceives the known deep neural network as nonlinear. The attack method used in this paper is called the Fast Gradient Sign Method.the equation is as follows.

$$\eta = \epsilon \cdot sign(\nabla_x J(\theta, x, y))$$

Let $J(\theta, x, y))$ be the cost function for a neural network, η the perturbation, ϵ the parameters, x the input, and y be the targets to the model. The image obtained by adding η to the original image becomes an adversarial model, and the equation is as follows.

$$\widetilde{x} = x + \eta$$

Projected Gradient Descent (PGD)[27] This is a application of the FGSM method introduced above, and it is to perform inner maximization under the specified norm by repeating the attack as many as n steps. In addition to the change in the number of steps, the size of the step called the learning rate was used to specify the size of the movement for each step.

$$x^{t+1} = \pi_{x+S}(x^t + \epsilon \cdots sign(\nabla_x J(\theta, x, y))$$

Local maxima, which was found through PGD-based attacks introduced in the paper, experimentally proved that it converges to similar loss values regardless of model or dataset. Based on this fact, they argue that PGD is the most effective of attacks using first-order alone to find the optimal solution to find local maximuma to induce misclassification of the model. In fact, it

(a) Grad-Cam [35]

(b) LIME[30]

Figure 2.4: Visualization methods. Each methodology visualizes which part of the input influenced the input of the model.

can be seen that the adversarial training model, which trained PGD samples in several papers, demonstrates consistent performance in any attack.

2.1.4 Explain AI (Visualization methods)

There are an interesting experimental attempt, and there was something like a Grad-CAM or a LIME, and the purpose was to visualize how the model makes an output. An overview of cam and lime methodology is shown in Figure 2.4. It can be divided into three ways to visualize how the deep learning model generates an output.

1. Backpropagation based methods
2. Activation based methods
3. Perturbation based methods

Backpropagation based method The backpropagation based methods are methods of expressing the importance of each pixel by calculating the loss of the deep learning model after feed forward the given image. Layer-wise Relevance Propagation[1], DeepLIFT[37], SmoothGrad[39] belong to

this method. Backpropagation based method have fast computational power and productivity to express detailed importance or relevance, but the quality of the image is poor and difficult to interpret. In order to evaluate confidence, it is necessary to pre-process, but it is difficult to obtain accurate results.

Activation Based Methods Activation based methods use a linearly combined weight of the activations from each convolutional layer for explanation. The most well-known method is the class activation map[49], followed by gradient class Activation Map(Grad-CAM) and Grad-CAM++[35][7]. Activation Based Method is an easy-to-see method of overlapping heatmap for areas that affect the existing image. However, it is not suitable for expressing more fine-grained evidence or color dependencies. It also has a disadvantage that it is difficult to guarantee that the results can be fully explained or that the decision-making process has been shown.

Perturbation Based Methods Perturbation based methods are a method of measuring importance through how predicted values change according to a small variation in input values. There is Local Interpretable Model-Agnostic Exploration (LIME)[30]. LIME explains which variables affect the input value through the variation perturbation, and obtains a regionally specific linear model near the input value. In the case of images, when predicting an image covering a part in a learned model, it can be seen that if the predicted value drops a lot, the hidden part has a great impact on actual prediction.

2.2 Distribution of angles between latent manifold and the decision boundary

2.2.1 Experiment detail

In this experiment, we focus on deep learning models trained for classification tasks on MNIST and CIFAR10 datasets. Simple CNN model, VGG-16[38], and ResNet-18[15] is used in this experiment. The Simple CNN model consists of the following. It consists of the 3x3 kernel 16 channels output, the 3x3 kernel 32 channels, the max pooling layer, the 3x3 kernel 64 channels, the 3x3 kernel 128 channels, the max pooling layer, the fully connected layer 128x64, and the fully connected layer 64x10. Learning rate is 0.1, epoch size is 20, SGD optimizer[31] is used for training. Step scheduling is used, and γ is 0.5.

FGSM[13], BIM[21], PGD[27], PGDL2[27], EOTPGD[25], FFGSM[46], TPGDM[48], and MIFGSM[11] are used as adversarial attack methods.

The α value is set to 2/255 which means that perturb step size is two pixels each iteration. The ϵ value is set to 32/255 which means that the max perturbs size is 32 pixels.

The latent vector f is extracted before the fully connected layer. The σ for obtaining manifold vector is 10^{-5}. The entire process is described in Algorithm 1. All experiments in this chapter use the above-described hyperparameter in the same way.

Algorithm 1 An process of finding the angular distribution.

Prepare : $X_{train}, Y_{train}, X_{test}, Y_{test}, M, Attack, \sigma$
▷ M is a model, *Attack* is an adversarial method
Train M with X_{train}, Y_{train}
$f_{ori} \leftarrow M(X_{test})$

Get decision boundary direction
$X_{adv} \leftarrow Attack(X_{test}, Y_{test})$ ▷ if untargeted attack
set source class : c_s and target class c_t ▷ if targeted attack
$X_{adv} \leftarrow Attack(X_{test}, Y_{test}, c_s, c_t)$
$f_{adv} \leftarrow M(X_{adv})$
$d_{adv} \leftarrow f_{adv} - f_{ori}$

Get manifold direction
$X_{perturb} \leftarrow X_{test} + N(0, \sigma^2)$
$f_{perturb} \leftarrow M(X_{perturb})$
$d_{perturb} \leftarrow f_{perturb} - f_{ori}$

Angle between manifold and decision boundary
$A \leftarrow \arccos^{-1}(\frac{d_{adv} \cdot d_{perturb}}{||d_{adv}||||d_{perturb}||})$

2.2.2 Experiment results

Table 2.1 shows the distribution of angles between the manifold vector and the decision boundary vector. The adversarial attack to obtain the decision boundary normal vector was performed in an untargeted target manner. As can be seen from the data, there are variations depending on the datasets, models, and attack methods, but it is distributed on average near 90 degrees. As a result of this, it can be inferred that the decision boundary is located almost close to that of the latent manifold. Figure 2.6 is a diagram visualizing the distribution of angles between the manifold vector and the decision boundary normal vector as a histogram.

Although there are variations depending on the source and target, it can be seen that it is distributed on average near 90 degree similarly to the target manner attack results.

As a result, we can confirm that assumption 1 is experimentally true. The latent manifold is locally parallel to the decision boundary. It can be assumed that the composition of the manifold and the decision boundary of the latent space is shown in Figure 2.5.

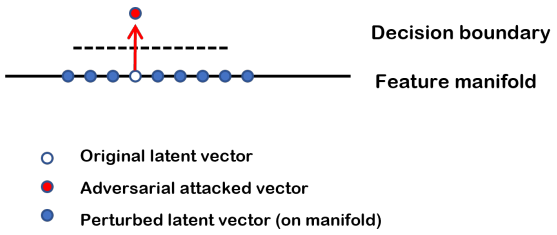
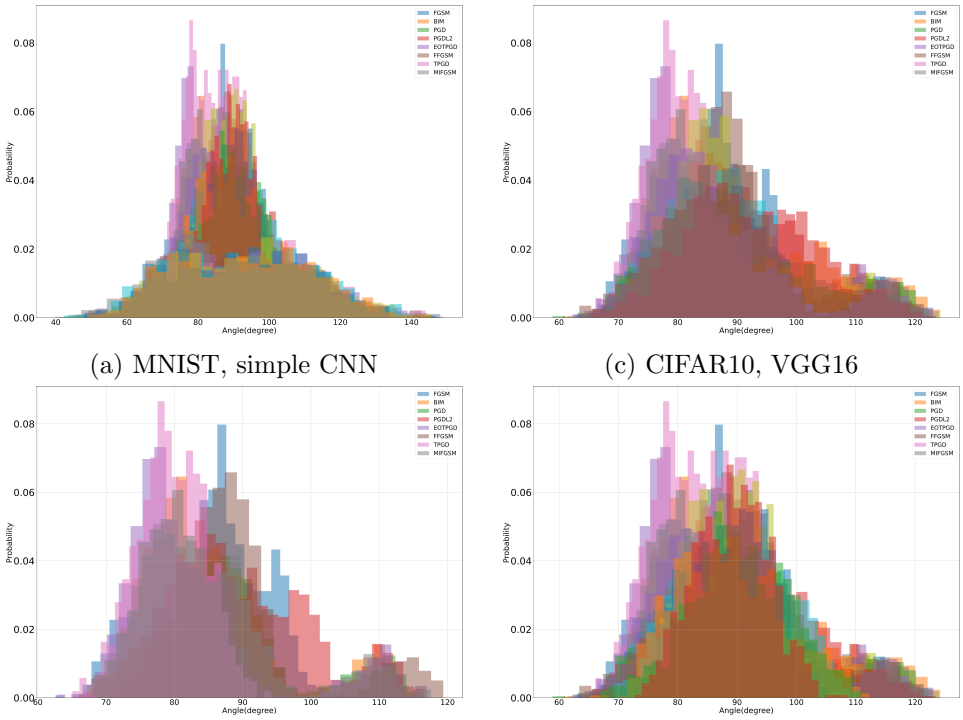


Figure 2.5: The configuration between the latent manifold and the decision boundary

Dataset	Model	Attack Method	Angle mean	Angle std
MNIST	Simple CNN	FGSM	92.2	19.2
		BIM	92.5	19.3
		PGD	93.9	19.2
		PGDL2	89.0	19.9
		EOTPGD	93.0	19.0
		FFGSM	92.2	19.8
		TPGD	92.6	18.9
		MIFGSM	92.8	19.1
CIFAR10	Simple CNN	FGSM	85.3	7.7
		BIM	85.0	9.8
		PGD	85.1	9.9
		PGDL2	90.3	9.6
		EOTPGD	82.8	10.3
		FFGSM	87.8	10.2
		TPGD	79.4	5.0
		MIFGSM	84.7	10.0
CIFAR10	VGG-16	FGSM	88.3	9.9
		BIM	88.6	11.3
		PGD	89.6	12.0
		PGDL2	91.5	13.3
		EOTPGD	89.1	12.4
		FFGSM	92.2	10.8
		TPGD	86.5	11.9
		MIFGSM	88.4	11.5
CIFAR10	ResNet-18	FGSM	88.9	5.4
		BIM	89.3	6.2
		PGD	89.6	6.3
		PGDL2	89.9	8.7
		EOTPGD	88.9	6.8
		FFGSM	88.6	8.8
		TPGD	90.3	8.3
		MIFGSM	89.1	6.2

Table 2.1: The mean and standard deviation of angle between manifold vector and decision boundary normal vector.



(a) MNIST, simple CNN

(c) CIFAR10, VGG16

(b) CIFAR10, simple CNN

(d) CIFAR10, ResNet18

Figure 2.6: Histogram of angle between manifold vector and decision boundary normal vector.

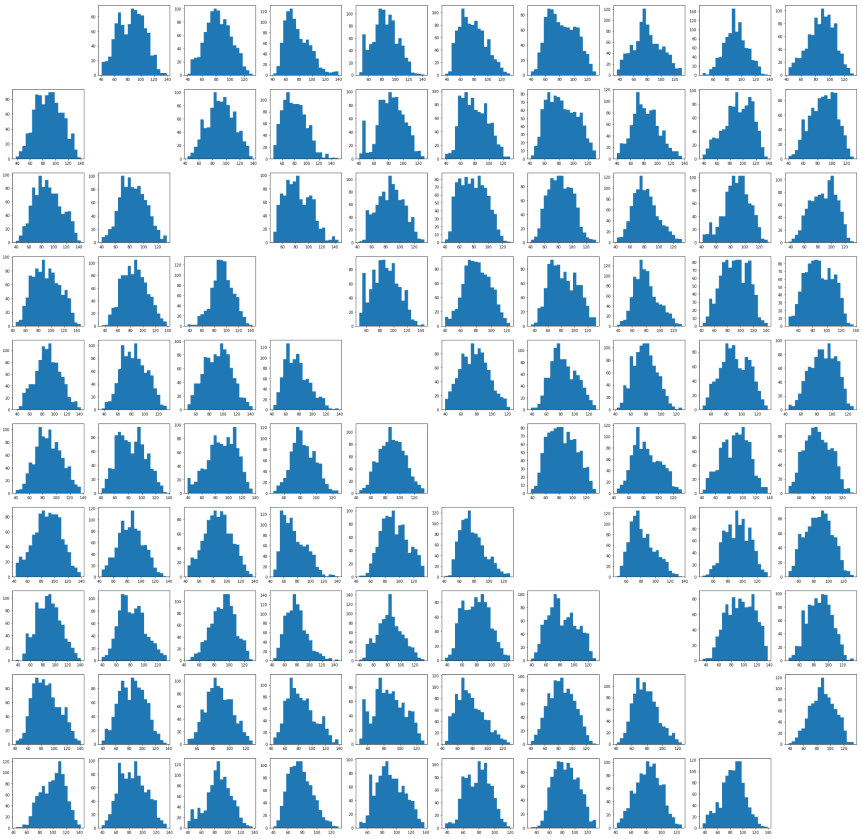


Figure 2.7: The angle distribution between manifold vector and decision boundary normal vector on targeted adversarial attack. x-axis is the source class and y-axis is the target class. Simple CNN, MNIST. FGSM attack.

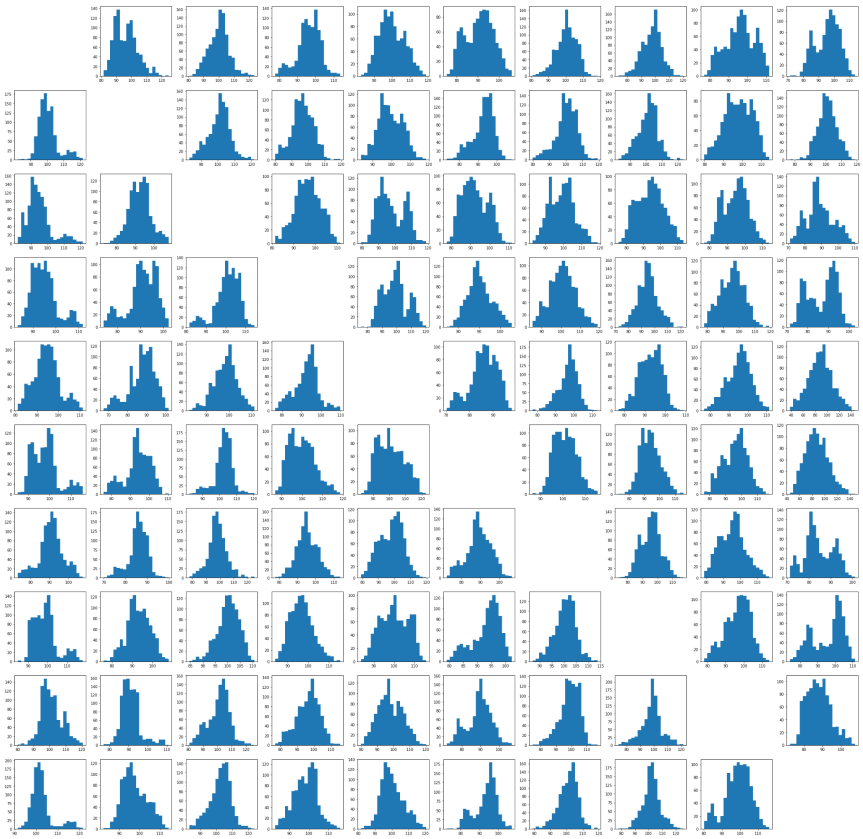


Figure 2.8: The angle distribution between manifold vector and decision boundary normal vector on targeted adversarial attack. x-axis is the source class and y-axis is the target class. Simple CNN, CIFAR10. BIM attack.

2.3 Near-local manifold curvature

2.3.1 Experiment detail

In Chapter 2.2, we intuitively hypothesized the relationship between the decision boundaries and the latent manifolds in a local area and experimentally confirmed that the assumptions were true. In this section, we would like to experimentally check the relationship between a wider range of latent manifolds and the decision boundaries. A manifold in a near-local area can be obtained by adjusting the degree to which the input image is perturbed. In Chapter 2.2, the degree of Gaussian noise is fixed to $\sigma = 10^{-5}$. In this experiment, the distribution of angles between manifold and boundary normal vectors is experimentally obtained by adjusting the degree of this Gaussian noise. Through the results, we check how the latent manifold is configured with the decision boundary. The experimental conditions excluding Gaussian noise are the same as in Chapter 2.2. Gaussian noise is set as follows. $(10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2})$

2.3.2 Experiment results

Figures 2.10 and Figures 2.11 show the distribution of angles between the decision boundary normal vector and the manifold vector according to the degree of Gaussian noise, respectively. The color depends on the attack method. The left side of Figure 2.10 is the result of the Simple CNN model in the MNIST dataset. The top distribution shows the distribution of angles when the σ is 10^{-5}. The bottom distribution shows the distribution of angles when the σ is 10^{2}. Figures 2.10 and Figures 2.11 are the results of Cifar10-Simple

CNN, Cifar10-VGG, and Cifar-ResNet, respectively. Looking at the results of figures 2.10 and figures2.10, although there are variations depending on the attack method, data set, and model type, the angle tends to decrease at 90 degree when noise is increased.

Figures 2.12 show the results of plotting the average of angles according to the degree of noise, respectively. The color of the graph means the attack method.

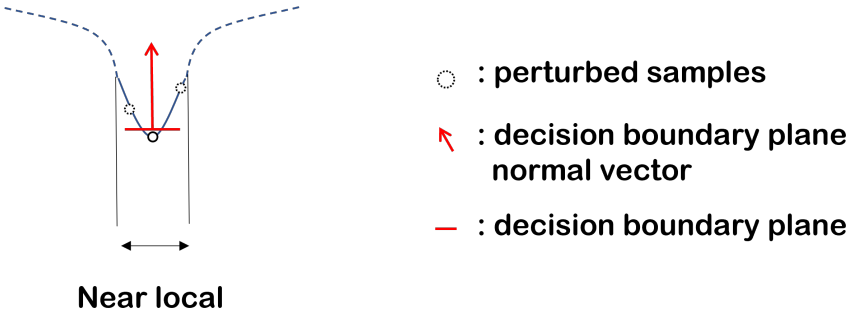
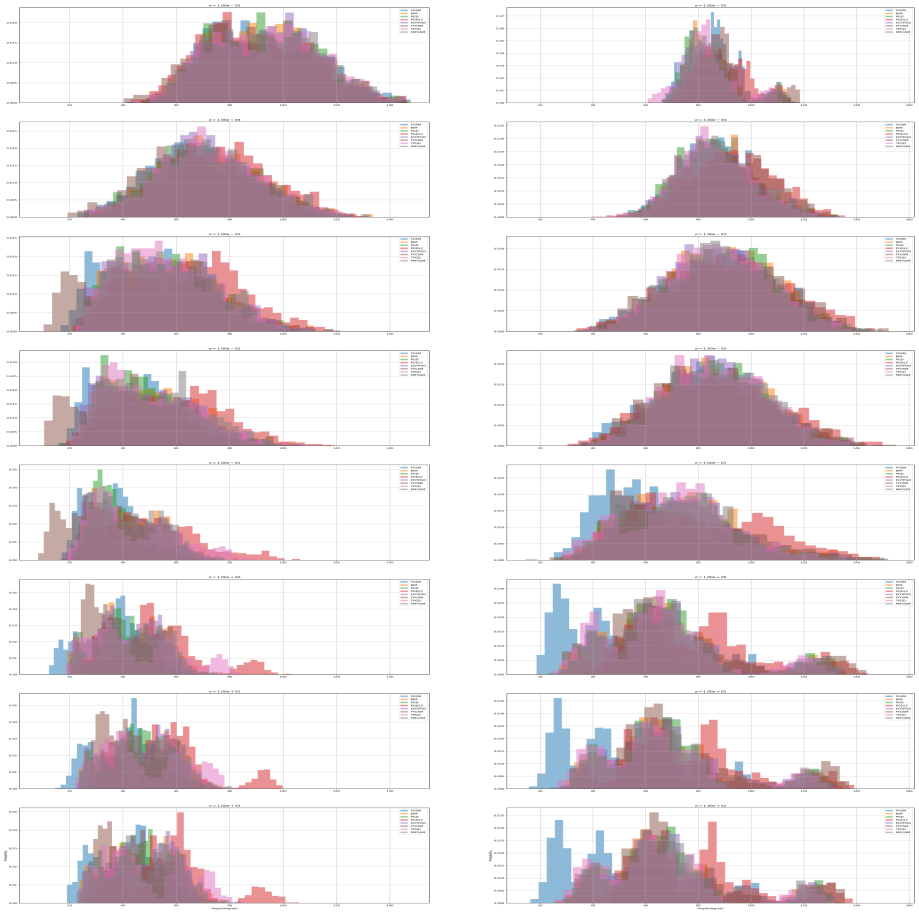


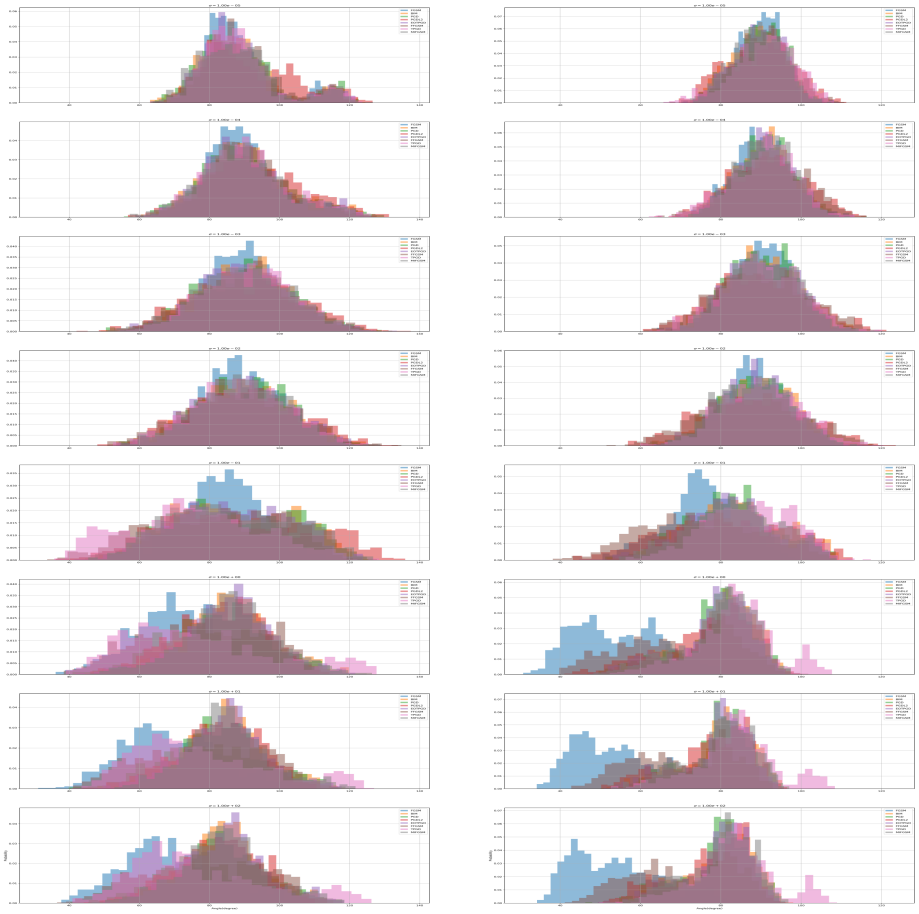
Figure 2.9: Configuration near local latent manifold

As a result of this experiment and the experiment result in Chapter 2.2, manifolds tend to be the same as decision boundaries in the local area, and It can be estimated that there is curvature in areas that are a little far away. The results of this experiment were schematized and visualized in figure 2.9.



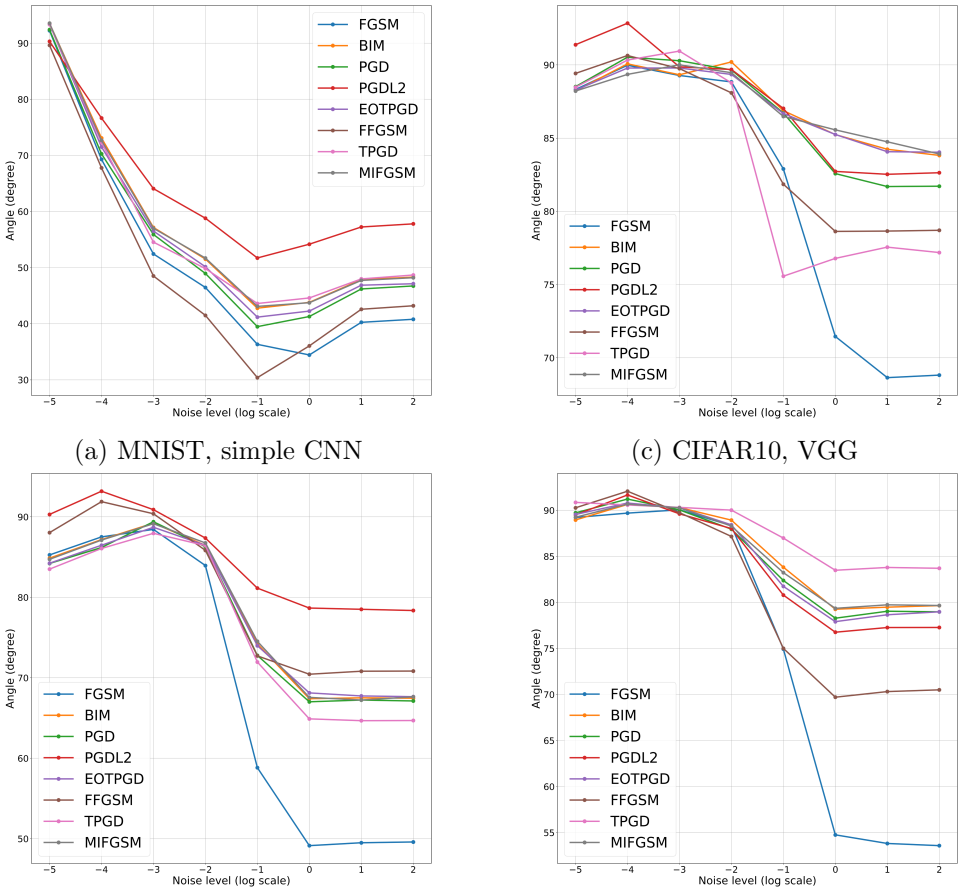
(a) MNIST, simple CNN (b) CIFAR10, simple CNN

Figure 2.10: Distribution of angles between manifold and boundary normal vectors according to manifold range(Gaussian noise level). Top noise level is 10^{-5}, bottom is 10^{2}



(a) CIFAR10, VGG (b) CIFAR10, ResNet

Figure 2.11: Distribution of angles between manifold and boundary normal vectors according to manifold range(Gaussian noise level). Top noise level is 10^{-5}, bottom is 10^{2}



(a) MNIST, simple CNN

(c) CIFAR10, VGG

(b) CIFAR10, simple CNN

(d) CIFAR10, ResNet

Figure 2.12: Trend of average angle between manifold and boundary normal vectors according to manifold range(Gaussian noise level) Top noise level is 10^{-5}, bottom is 10^{2}

2.4 Miscellaneous experiments

In this section, we consider some interesting questions about adversarial attacks and high-dimensional spaces and try to find the answers experimentally or mathematically.

2.4.1 Does adversarial attack really mean a vulnerability in deep learning models?

As machine learning becomes increasingly popular, privacy and security issues of deep learning methodology are important. But does adversarial attack really mean a vulnerability in deep learning models? To answer this question, let's look at the adversarial attack methodology. Whitebox attacks are difficult to apply in reality because of the constraints that all structures and parameters of the model must be known. So, many black-box attack methodologies[4][6][8], which are attacks in situations where the structure of the model and weights are not known, are also being studied. However, it is also questionable whether a blackbox attack is practically possible in a general situation rather than a malicious attack situation in that it uses multiple queries. In this section, we examine whether the deep learning methodology is actually vulnerable to small random noise.

First, we propose a simple thought experiment. To simplify the problem, let's deal with the classification problem. $D(X^{N\times N}, Y^K)$ is a dataset consisting of $X^{N\times N}$ and Y^K. N denotes the width and higtht of the image and K denotes the number of classes in dataset. Let X_i, Y_i denote the i-th image of dataset and the i-th ground truth class. Suppose that there is only

Noise Level	Misclassification rate (%)
0.01	0.856
0.02	2.218
0.03	4.256
0.04	6.626
0.05	7.97284

Table 2.2: Misclassification rate of random perturbed samples (adding noise of 0.01 to 0.05 uniform random noise).

one direction when an adversarial attack is performed with any class other than ground truth Y_i. Then, the probability that the deep learning model will produce incorrect results for any input noise is $1/2^{N \times N}$.

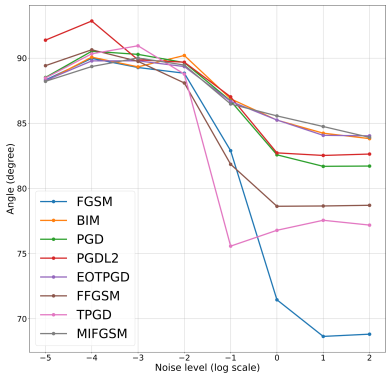
Of course, the above thinking experiment presupposes the assumption that there is only one way to change the class. Therefore, an experiment is conducted to determine whether the results of deep learning can actually be changed with random noise. The specific experimental setting is described below.

A model in which CIFAR-100 data is learned with Resnet 50 is used. Prepare 25 samples accurately predicted by deep learning. 100,000 perturbed samples are prepared for each sample by adding noise of about 0.01 to 0.05 uniform in the input space. Check if the predictions of the perturbed samples are different from the original prediction. The experimental result table 2.2 shows that the adversarial attack cannot be successful with random noise.

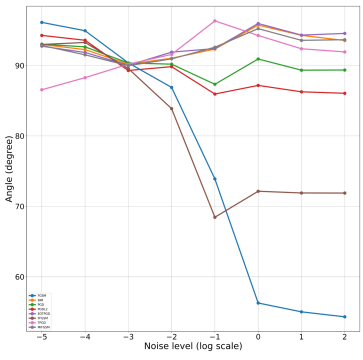
2.4.2 Is the manifold's shape related to the performance of the model?

In this section, we would like to check the possibility of using the manifold's shape as an indicator to evaluate the performance of the model. First, we prepare two models learned through augmentation using flip augmentation and a no augmentation vanilla model. The manifold near local curvature was checked using the methodology used in the section 2.3 for each model. Figure 2.13 shows the manifold near local curvature of each model. Although there are variations depending on the adversarial attack methodology, the curvature of the augmentation model tends to be flatter. In general, it is known that the generalization performance of deep learning improves when learning using augmentation. Therefore, we can expect that the flat the curvature, the better the generalization performance.

Further research is needed on the correlation between the manifold near local curvature and the performance of the deep learning model



(a) Vgg16 Cifar Vanilla model



(b) Vgg16 Cifar Vanilla model (flip aug train)

Figure 2.13: The manifold configuration of vanilla model and flip augmentation model

Chapter 3

Attention style Capsulenet

This chapter is based on the paper[9] accepted in ICCV 2019 workshop.

The convolutional layer is an effective method to extract local features due to its local connectivity and parameter sharing with spatial location. However, the convolutional layer has a limited ability to encode a transformation. For example, if the convolutional layer is combined with a max-pooling layer, the extracted feature is local translation invariant. As CNN models become deeper [15] [41], the receptive field of each feature is getting larger. Then, the information loss from the translation invariance also increases.

To overcome the transformation invariance of CNNs, the transforming autoencoder[17] uses the concept of "capsule". A capsule is a vector representation of a feature. Each capsule not only represents a specific type of entity but also describes how the entity is instantiated, such as precise pose and deformation. In other words, the capsules are transformation equivariant.

The CapsuleNet[33] is a novel method that implements the idea of the capsules. By introducing the dynamic routing algorithm and squash activation function 3.0.1, CapsuleNet uses vector-output capsules as a basic unit instead of scalar-output features.

$$\text{squash}(\mathbf{s_j}) = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||} \tag{3.0.1}$$

where $\mathbf{s_j}$ is a pre-activation capsule. However, CapsuleNet has a room for development. The number of parameters of CapsuleNet is much larger than that of comparable performance CNN-based models. Also, the dynamic routing is an iterative process. The reported accuracy of CapsuleNet on the benchmark datasets like CIFAR-10 is inferior to state-of-the-art CNN models[47].

In this Chapter, we propose a convolutional capsule network architecture comprised of building blocks of CNNs. We substitute the dynamic routing and squash capsule-activation function of CapsuleNet[33] with attention routing and capsule activation. In the attention routing, the log probabilities of agreement coefficients between the lth layer and the $(l+1)$th layer are learned by a scalar-product between the capsules of the lth layer and the kernel of convolution. The kernel of convolution serves as an approximation of the reference vector to perform routing. By replacing an iterative process of the dynamic routing with forward-pass convolution, the attention routing is fast while maintaining spatial information. Two important properties of squash activation function 3.0.1 is that the squash activation function preserves a vector orientation and is a capsule-wise activation function, not

an element-wise activation function such as ReLU or tanh. The dynamic routing is an unsupervised algorithm to find a centroid-like output capsule of the prediction capsules. Therefore, the squash activation function and its variant 3.0.2 [47] focus on preserving a capsules orientation.

$$\text{squash variant}(\mathbf{s_j}) = \left(1 - \frac{1}{\exp\left(||\mathbf{s}_j||\right)}\right) \frac{\mathbf{s}_j}{||\mathbf{s}_j||} \tag{3.0.2}$$

However, we focus on the capsule-wise operation rather than preserving orientation. The capsule activation performs an affine transform on the capsules and then applies an element-wise activation function. The capsules on the same capsule channel share parameters used in the affine transformation. Thus, the capsules on the same capsule channel are mapped to the same feature space, and the operation is parameter efficient. Therefore, the capsule activation is a capsule-wise function that does not preserve a vector orientation. Since the capsule activation applies a nonlinear transformation to a linear combination of the prediction capsules, parametrizing the routing process through the attention routing is compatible. We refer to our proposed model as *Attention Routing CapsuleNet (AR CapsNet).*

We evaluate the AR CapsNet on three datasets (MNIST, affNIST, and CIFAR-10). The AR CapsNet significantly outperforms CapsuleNet in the affNIST and CIFAR-10 classification task and shows a comparable performance in the MNIST dataset while being faster and using less than half parameters than CapsuleNet. Moreover, the AR CapsNet preserves the transformation equivariant property of CapsuleNet. As we perturb each element of the output capsule, the decoder attached to the output capsules shows

global variations as in [33]. Further experiment showed that the affine transformations on an input image cause the feature capsules to change in the significantly aligned direction. From these experiments, we prove that the AR CapsNet encodes an affine transformation on the input image in some basis of capsule space. In addition, our proposed architecture is constructed in a convolutional manner so that it can be easily extended to a deeper network structure.

- We propose a new architecture called AR CapsNet by introducing two modifications to the CapsuleNet [33]. These modifications are the *attention routing* and *capsule activation.*

- The capsule activation expands the concept of the existing capsule-wise activation functions such as the squash activation. The capsule activation performs an orientation-nonpreserving transform on the pre-activation capsules. The performance of the AR CapsNet demonstrates that the transformation equivariant features can be extracted even if the routing process is not restricted to the clustering approach and the capsule activation is not limited to the normalization.

- The AR CapsNet shows better results on the affNIST, and CIFAR-10 classification tasks and comparable results on the MNIST classification task while using much smaller parameters than CapsuleNet. Also, the AR CapsNet preserves the transformation equivariant property of the CapsuleNet. As we perturb each element of the output capsule, the decoder attached to the output capsule shows global variation as in [33].

- To investigate the transformation equivariance further, we suggest a new experiment. We observe the difference in the output capsule caused by applying transformations on an input image. In the AR CapsNet, these difference vectors are significantly aligned compared to a set of random vectors. These results demonstrate that transformation on an input image is encoded in some basis vector.

3.1 Related Works

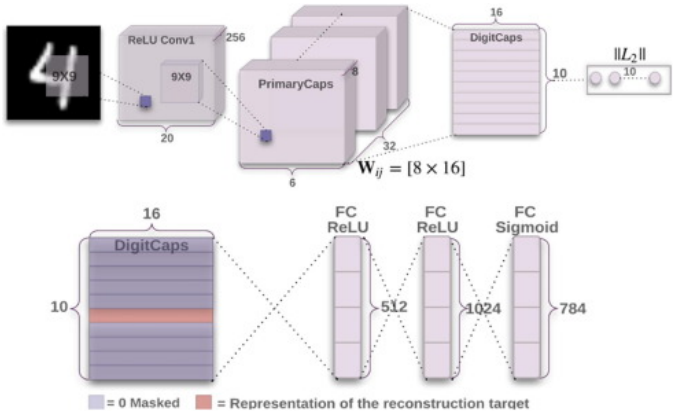


Figure 3.1: Overview of Capsulnet[33].

Let T be a transformation. A function f is invariant if $f(T(x)) = f(x)$. A function f is equivariant if $f(T(x)) = T(f(x))$. CNN(with maxpooling) is translation invariant. Max pooling ignores feature spatial information, especially in the deep layer. The CNN models that consist of convolutional layers and max-pooling layers have a local translation invariance. To overcome transformation invariance, CapsuleNet [33] uses vector-output capsules and

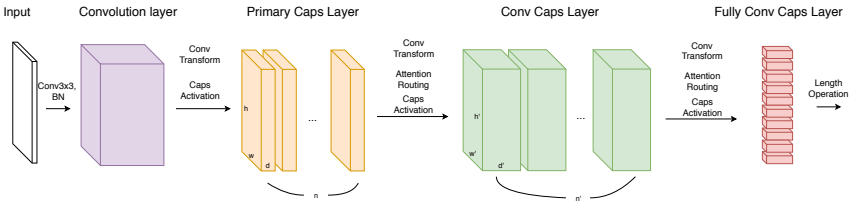


Figure 3.2: Overview of AR CapsNet. AR CapsNet is composed of primary caps layer, conv caps layer, and fully conv caps layer. BN denotes the batch normalization. Conv Transform and Caps Activation denotes the convolutional transform and capsule activation respectively.

the dynamic routing in place of scalar-output features and max-pooling. The [33] argues that scalar neurons are not suitable to contain more information like position, angle, affine transform information.

By demonstrating that the dimension perturbation of digit capsules leads to a global transformation of the reconstruction image, CapsuleNet claims to have transformation equivariance.

A number of methods to improve the performance of CapsuleNet have been proposed in [18] [45] [2] [22] [24]. In [45], they interpreted the routing-by-agreement process as an optimization problem of minimizing clustering loss. They proposed another routing process from the point of view of clustering. Their approach achieved better results on an unsupervised perceptual grouping task compared to [33]. The matrix capsules with EM routing [18] proposed another routing method called EM routing. The EM routing measures compatibility between matrix capsules by clustering matrix capsules through Gaussian distributions. The matrix capsules with EM routing achieved the state-of-the-art performance on a shape recognition task using the smallNORM dataset. The spectral capsule networks [2] is a variation of

[18]. Spectral capsule networks use a singular value to compute the activation of each capsule instead of the logistic function in [18]. Spectral capsule networks achieved better performance on a diagnosis dataset compared to [18] and deep GRU networks while showing faster convergence compared to [18].

The SegCaps [22] applied a capsule network to the object segmentation task. The SegCaps introduced two modifications to the CapsuleNet and devised the concept of deconvolutional capsules from these modifications. The two modifications are the locally connected dynamic routing and the sharing of transformation matrices within the same capsules channel. The sharing of transformation matrices is equivalent to the convolutional transform of our conv caps layer except for the addition of biases. The EncapNet [24] performs a one-time pass approximation of the routing process by introducing two branches. The master branch extracts a feature from the locally connected capsules as in [22] and the aide branch combines information from all the remaining capsules. Also, they introduced a Sinkhorn divergence loss which works as a regularizer. The EncapNet achieved competitive results on CIFAR-10/100, SVHN, and a subset of ImageNet.

Our proposed model uses attention architecture as a routing algorithm. The attention architecture learns a compatibility function between low-level features and high-level features. In [3], the output of attention architecture is a weighted sum of input features, and the weights are the compatibilities based on the input features and the RNN hidden state. The compatibility function is a feedforward neural network with a softmax activation function. In [26], they experimented on various kinds of attention architectures

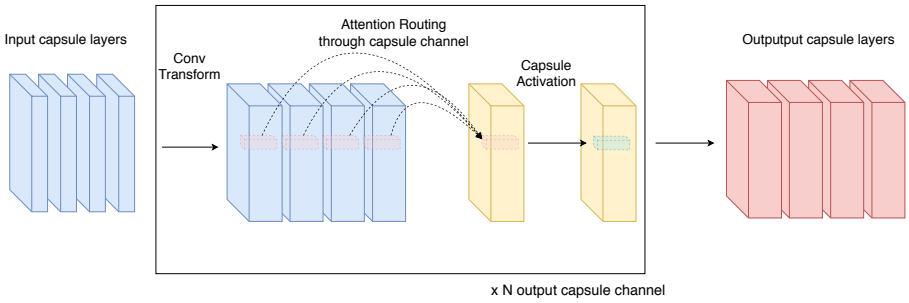


Figure 3.3: Detailed operation process of conv caps layer. Conv Transform denotes the convolutional transform. The convolutional transform performs a locally connected affine transform on each capsule channel. The attention routing learns the agreement between the convolutional transformed capsules for each spatial location. The capsule activation applies a capsule-wise activation function on each capsule channel.

from global attention to local attention and three compatibility functions. One of the three compatibility functions was a softmax output of the scalar-products between a target hidden state vector and source hidden state vector. The transformer network [44] uses a similar attention architecture as in [26]. Transformer performs a scaled scalar-product between the keys and values and then applies a softmax activation function. Our proposed attention routing computes the scalar product between capsules and a kernel.

3.2 Proposed Method

Our proposed architecture consists of *primary caps layer*, *conv caps layer*, and *fully conv caps layer*. We denote the lth capsule layer as $\mathbf{u}^l_{w,h,d,n}$, where w, h, d, and n index the spatial width axis, spatial height axis, capsule dimension axis, and capsule channel axis, respectively. We refer to the capsules

with the same capsule channel index as a *capsule channel* $\mathbf{u}^l_{(:,:,:,n)}$ [1].

3.2.1 Primary Caps Layer

We denote the primary capsule layer as the 0th capsule layer. Before entering the primary caps layer, we extract local features $\tilde{x}$ from the input image x by performing the convolution blocks composed of convolution layer and batch normalization.[19] We consider the local features $\tilde{\mathbf{x}}$ as a single capsule layer. In our primary caps layer with N channels of D dimensional output capsules, 3x3 convolution with kernels of filter size D and stride 2 is performed on the input capsules $\tilde{\mathbf{x}}$ N times independently. Each output of a convolution layer is a capsule channel.

$$\mathbf{s}^0_{(:,:,:,n_0)} = \text{ReLU}\left(\text{Conv}_{3\times 3}\left(\tilde{x}\right)\right) \tag{3.2.1}$$

Note that this is equivalent to performing a 3x3 convolution of $N \times D$ kernels and then reshaping the features to (B, W, H, D, N) where B denotes the batch size and (W, H) denotes the spatial size of the capsule layer. Then, the capsule activation is applied to each capsule channel instead of the squash activation function in [33].

3.2.2 Capsule Activation

The capsule activation takes an affine transformation on each capsule channel and then applies tanh activation function. The capsules on the same capsule channel share parameters of the affine transformation. Thus, the

[1] $\mathbf{u}^l_{(:,:,:,n_0)} := \{u^l_{w,h,d,n} | n = n_0\}$

capsule activation is equivalent to taking 1x1 convolution with a kernel of filter size D and tanh activation function on each capsule channel.

$$\mathbf{u}_{(:,:,:,n_0)} = \tanh\left(\mathrm{Conv}_{1\times1}\left(\mathbf{s}_{(:,:,:,n_0)}\right)\right) \tag{3.2.2}$$

Each element of the output capsules of the capsule activation depends on the corresponding input capsule. Therefore, the capsule activation is a capsule-wise activation function. The tanh activation function normalizes each element of capsules, thus stabilizes the lengths of the capsules.

3.2.3 Conv Caps Layer

We denote the input to the lth conv caps layer as $\mathbf{u}^{l-1}_{w,h,d,n}$ which is the output of the $(l-1)$th conv caps layer. We first perform a *convolutional transform* on each capsule channel. The convolutional transform is a locally-connected affine transformation sharing parameters within the same capsule channel. In particular, the convolutional transform is a 3x3 convolution of D^l kernels without activation function, where D^l denotes the capsule dimension of the lth conv caps layer.

$$\tilde{\mathbf{s}}^{l,n}_{(:,:,:,m)} = \mathrm{Conv}_{3\times3}\left(\mathbf{u}^{l-1}_{(:,:,:,m)}\right) \tag{3.2.3}$$

Each output of the convolutional transform is fed to the attention routing. The output of attention routing is a linear combination of the convolutional

transformed capsules with the same spatial location.

$$\mathbf{s}^{l}_{(w,h,:,n)} = \sum_{m=1,\cdots,N^{l-1}} c^{l,n}_{(w,h,m)} \cdot \tilde{\mathbf{s}}^{l,n}_{(w,h,:,m)} \tag{3.2.4}$$

where the capsules $\mathbf{s}^{l}_{(w,h,:,n)}, \tilde{\mathbf{s}}^{l}_{(w,h,:,m)} \in \mathbb{R}^{D^l}$. The weights $c^{l,n}_{(w,h,m)} \in \mathbb{R}$ are computed by the attention routing. The log probabilities $b^{l,n}_{(w,h,m)}$ are the scalar-product between a concatenation of capsules $[\tilde{\mathbf{u}}^{l}_{w,h,:,1}, \tilde{\mathbf{u}}^{l}_{w,h,:,2}, \cdots, \tilde{\mathbf{u}}^{l}_{w,h,:,N^{l-1}}]$ and a parameter vector $w^{l}_{n} \in \mathbb{R}^{D^l \times N^{l-1}}$. This operation can be implemented efficiently by 3D convolution on the convolutional transformed capsule layers with kernels $w^{l}_{n} \in \mathbb{R}^{1\times 1\times D^l \times N^{l-1}}$, stride=(1,1,1), and valid padding.

$$b^{l,n}_{(:,:,:)} = \mathrm{Conv3D}_{1\times 1\times D^l}\left(\tilde{\mathbf{s}}^{l}_{(:,:,:,:)}\right) \tag{3.2.5}$$

The weights $c^{l,n}_{(w,h,m)}$ are softmax outputs of the log probabilities $b^{l,n}_{(w,h,m)}$ along the capsule channel axis.

$$c^{l,n}_{w,h,m} = \frac{\exp(b^{l,n}_{(w,h,m)})}{\sum_{1\le m\le N^{l-1}} \exp(b^{l,n}_{(w,h,m)})} \tag{3.2.6}$$

Note that the attention routing adjusts the weight $c^{l,n}_{w,h,m}$ for each spatial location (w,h) corresponding to the convolutional transformed capsules $\{\tilde{\mathbf{u}}^{l}_{(w,h,:,m)}\}_m$ with the same spatial location.

Finally, the capsule activation is performed on each capsule channel $\mathbf{s}^{l}_{(:,:,:,n)}$. A set of convolutional transform, attention routing, and capsule activation is performed independently N^l times. (*i.e.*, each output of the

convolutional transform, attention routing, and capsule activation is a capsule channel $\mathbf{u}^l_{(:,:,:,n)}$)

$$\mathbf{u}^l_{(:,:,:,n)} = \tanh\left(\mathrm{Conv}_{1\times 1}\left(\mathbf{s}^l_{(:,:,:,n)}\right)\right) \tag{3.2.7}$$

Intuitively, the dynamic routing uses a centroid of the transformed capsules as the reference vector to measure agreement by scalar-product. As the dynamic routing process iterates, the capsule with the higher agreement has a larger weight, and the reference vector evolves in that capsule direction. On the other hand, since the capsule activation in the conv caps layer do not preserve vector orientation, the output capsule $\mathbf{u}^l_{(:,:,:,n)}$ cannot approximate the centroid of transformed capsules $\{\tilde{\mathbf{u}}^l_{(w,h,:,n)}\}$. Instead of measuring agreement between the transformed capsules and the output capsule $\mathbf{u}^l_{(:,:,:,n)}$, the attention routing parametrizes the routing process. The parameter vector w^l_n which is the kernel of convolution serves as an approximation of the reference vector to perform routing.

We propose replacing the dynamic routing of [33] with the convolutional transform and attention routing. Compared to dynamic routing, our proposed operation is faster and more parameter efficient. Since dynamic routing is constructed in a fully connected manner, the transform weight matrices are assigned for each pair of the input capsule and output capsule. We share the weight matrices across the spatial location and keep the translation equivariance by performing 3x3 convolution on the lth layer in the convolutional transform. Besides, the dynamic routing has an iterative routing process to compute the weight $c^l_{w,h,n}$. On the other hand, by introducing

a trainable parameter vector, our proposed operation is a fast forward-pass.

Moreover, dynamic routing applies a softmax activation function along the capsule channel axis of output capsule.[2] In attention architectures [3] [26] [44], the softmax activation function is applied on the input channel axis in order to keep the scale of higher features stable. By treating capsule layers in a convolutional manner, attention routing applies a softmax activation function along the input capsule channel axis. By applying the softmax activation function for each spatial location along the input capsule channel, the spatial information is preserved and the scale of capsule values becomes stable.

3.2.4 Fully Conv Caps Layer

The fully conv caps layer is almost the same as the conv caps layer and serves as the output layer of AR CapsNet. The convolutional transform combines capsule features from the all spatial location by applying a kernel of the same spatial size as the input with valid padding. Therefore, the output of the fully conv caps Layer has a shape of $(1, 1, D^L, N^L)$.

3.2.5 Margin Loss and Reconstruction Regularizer

We adopt the margin loss and reconstruction regularizer in [33]. Since the output capsules of capsule activation have a length of up to $\sqrt{D^L}$ where D^L denotes the capsule dimension, we use the normalized length to predict the

[2] If $c^l_{i,j}$ denotes the weight between capsule i in the $(l-1)$th layer and capsule j in the lth layer, $\sum_j c^l_{i,j} = 1$.

Algorithm 2 The process of Attention Routing

while $\ell = 1, \cdots, L$ **do**
  while $n = 1, \cdots, N^{\ell}$ **do**
  Convolutional transformation for each capsule channel
    while $m = 1, \cdots, N^{\ell-1}$ **do** $\tilde{\mathbf{s}}^{\ell,n}_{(:,:,:,m)} \leftarrow \texttt{Conv2D}_{3\times3}(\mathbf{u}^{\ell-1}_{(:,:,:,m)})$
    Attention through capsule channel
  $\mathbf{b}^{l,n} \leftarrow \texttt{Conv3D}_{1\times1\times D^{\ell}}(\tilde{\mathbf{s}}^{\ell})$
    while $w, h = 1, \cdots, W^{\ell}, H^{\ell}$ **do**
  $\mathbf{c}^{n,l}_{w,h,:N^{\ell-1}} \leftarrow \texttt{softmax}(\mathbf{b}^{\ell,n}_{w,h,:N^{\ell-1}})$
  $\mathbf{s}^{\ell}_{w,h,:,n} \leftarrow \sum_{m=1,\cdots,N^{l-1}} c^{n,l}_{w,h,m} \cdot \tilde{\mathbf{s}}^{\ell,n}_{(w,h,:,m)}$
  Capsule activation for each capsule channel
  $\mathbf{u}^{\ell}_{(:,:,:,n)} \leftarrow \tanh(\texttt{Conv2D}_{1\times1}(\mathbf{s}^{\ell}_{(:,:,:,n)}))$

probability of the corresponding class of the dataset.

$$||\mathbf{u}^L_n||_{\text{nor}} = \frac{||\mathbf{u}^L_n||}{\sqrt{D^L}} \tag{3.2.8}$$

where $||\mathbf{u}^L_n||$ denotes the output capsules of the fully conv caps layer and n indexes the capsule channel axis. We applied the Margin loss, L_n, for each class n on the $||\mathbf{u}^L_n||_{\text{nor}}$.

$$\begin{aligned} L_n = T_n \max(0, m^+ - ||\mathbf{u}^L_n||_{\text{nor}})^2 \\ + \lambda(1 - T_n) \max(0, ||\mathbf{u}^L_n||_{\text{nor}} - m^-)^2 \end{aligned} \tag{3.2.9}$$

where $T_n = 1$ iff the corresponding class of output capsule is present and $m^+ = 0.9$ and $m^- = 0.1$.

The output capsules $\{\mathbf{u}^L_n\}_{n=1,\cdots,N}$ are fed to the reconstruction decoder. We used a decoder consisting of 3 fully connected layers as in [33] except that our decoder has (512, 512, the number of input image pixel) nodes. We

refer to the mean of L2 loss between an input image and the decoder output as a reconstruction loss. We add the reconstruction loss that is scaled down by 0.3 to the margin loss as a regularization method.[3]

3.3 Experiments

We evaluate our model on the MNIST, affNIST, and CIFAR-10 datasets. For each dataset, we split the training images into a training set (90%) and a validation set (10%). We choose the model with the lowest validation error and evaluate the model on the test set. Then, we compare the results with CapsuleNet [33]. We use a Keras implementation[4] for CapsuleNet.

Before training the model on the image dataset, we divide each pixel value by 255 so that it is scaled in the range of 0 to 1. Then, we extract the local features $\tilde{x}$ from an input image through two convolutional layers with batch normalization(BN) [19] and ReLU activation function. These two convolutional layers use 3x3 kernels with a stride 1. Then, the features go through the AR CapsNet to obtain vector outputs. For each conv caps layer and fully conv caps layer, the dropout layer [40] of keep probability 0.5 is applied to the input capsules before the convolutional transform.

We use the RMSprop optimizer with rho of 0.9 and decay of 1e-4 to minimize the loss defined in Section 3.2.5. We set the learning rate as 0.001 and batch size as 100

[3] CapsuleNet [33] scaled the reconstruction loss by 0.392. Since we use the mean of L2 loss and CapsuleNet use the sum of L2 loss, $0.392 = 0.0005 \times 784$.

[4] https://github.com/XifengGuo/CapsNet-Keras

Method	MNIST	MNIST+	affNIST	C10	C10+
CapsuleNet[33]	99.45*	99.75 (99.52*)	79.0	63.1*	69.6*
CapsuleNet+ensemble(7)	-	-	-	-	89.4
Ours	99.46	99.46	91.6	87.19	88.61
Ours+ensemble(7)	-	-	-	88.94	90.11

Table 3.1: Test accuracy (%) on the MNIST, affNIST, and CIFAR-10 classification tasks. C10 represents the CIFAR-10 dataset. + denotes training with data augmentation. We adopted translation for MNIST+ and translation, rotation, and horizontal flip for C10+. * indicates the results from our experiment.

3.3.1 Classification Results on MNIST and affNIST

Dataset The MNIST dataset is composed of 28×28 handwritten digit images. We adopted 0.1 translation as a data augmentation for the MNIST dataset. The affNIST dataset consists of 40×40 images, which are obtained by applying various affine transformations such as rotation and expansion to the images from MNIST. For the affNIST classification task, we trained our model with randomly translated MNIST images in horizontal or vertical directions up to shift fraction 0.2 as in [33]. Any other affine transformations like rotations were not used in the training process. The affNIST dataset has a separate validation set, thus we chose the model with the lowest validation error based on the affNIST validation set. Then, we tested our model with the affNIST test set.

Implementation For the MNIST and affNIST datasets, we used the AR CapsNet which consists of a primary caps layer, one conv caps layer and fully conv caps layer. Before entering the AR CapsNet, an input image goes through two convolutional layers of 64 channels (3x3 Conv - BN - ReLU).

The primary caps layer has eight channels of 16-dimensional capsules, the conv caps has eight channels, and the fully conv caps layer has ten channels. Each capsule channels in the conv caps layer and fully conv caps layer has 32 dimensions in the MNIST and 16 dimensions in the affNIST. We decreased the spatial size of the capsule features by applying a 3x3 convolution of stride 2 in the convolutional transform of the conv caps layer. We trained our model for 20 epochs.

Accuracy Our model shows a comparable accuracy with the substantial decrease in the number of parameters and training time. Our model with 5.31M parameters achieved 99.45% accuracy on the MNIST dataset without any data augmentation and 99.46% accuracy with data augmentation. (Table 3.1) The CapsuleNet with 8.21M parameters achieved 99.45% accuracy without any data augmentation and 99.52% with data augmentation. The reported accuracy of CapsuleNet on the MNIST dataset with translation augmentation is 99.75% [33]. Also, the training took 37.2 seconds per epoch for our proposed model and 199.5 seconds per epoch for CapsuleNet when we experimented on GTX 1080 GPUs.

In the affNIST experiments, there are two options to generate training images from the MNIST dataset. The first option is to create a larger dataset by generating a set of all the possible augmented data before training. The second option is to apply translation over the original dataset for each epoch. The reported accuracy of CapsuleNet is 79% and that of the baseline CNN model is 66% in [33]. The experiment is performed on the former option.[5] Our proposed model achieved 91.6% accuracy for the lat-

[5]https://github.com/Sarasra/models/tree/master/research/capsules

ter option. Under the comparable experiment, our model outperformed the CapsuleNet and the baseline CNN model. Since our model is transformation equivariant (Section 3.3.4), our model is robust to affine transformations.

3.3.2 Classification Results on CIFAR-10

Dataset The CIFAR-10 dataset is a 32×32 colored natural images in 10 classes. We adopted 0.1 translation, rotation up to 20 degrees, and horizontal flip as a data augmentation for CIFAR-10.

Implementation For the CIFAR-10 classification task, we added four conv caps layer between a primary caps layer and fully conv caps layer. We decreased the spatial size of the capsule features in the first conv caps layer as in Section 3.3.1. Each conv caps layer has eight channels of 32-dimensional capsules and is connected to the next conv caps layer with a residual connection [15]. Note that the residual connection in [15] connects the lth layer and $(l+2)$th layer, but our residual connection connects the lth conv caps layer and $(l+1)$ conv caps layer. We trained our model for 200 epochs.

Accuracy The results in Table 3.1 show that our model outperforms CapsuleNet with and without data augmentation. CapsuleNet with 11.74M parameters shows 63.1% accuracy in C10 and 69.6% accuracy in C10+. However, our proposed model with 9.6M parameters shows 87.19 % accuracy in C10 and 88.61% accuracy in C10+. In [33], an ensemble of 7 models achieves 89.4% accuracy when the models are trained with 24×24 patches of images and the introduction of a *none-of-the-above* category. However, an

Conv caps layer	Caps dim	Params	C10	C10+
0	16	7.3M	77.51	81.89
	32	12.6M	77.44	81.97
1	16	3.5M	81.96	82.83
	32	7.7M	82.39	83.92
2	16	3.7M	84.48	84.77
	32	8.4M	85.46	87.01
3	16	3.8M	85.56	86.93
	32	8.9M	86.56	87.91
4	16	4.0M	86.37	87.21
	32	9.6M	87.19	88.61

Table 3.2: Test accuracy (%) on the MNIST and CIFAR-10 for various hyperparameters. In each experiment, we trained a model for 200 epochs and chose the model with the lowest validation error. For each hyperparameter setting, AR CapsNet shows stable performance without showing severe degradation.

ensemble of 7 AR CapsNet models trained with C10+ achieved 90.11% test accuracy. Note that C10+ only uses rotation, shift, and horizontal flip as data augmentation and not the cropping or the *none-of-the-above* category.

3.3.3 Robustness to hyperparameters

Implementation The AR CapsNet requires a set of hyperparameters, such as the number of conv caps layer and the capsule dimension of each capsule layer. To test the robustness to hyperparameters, we evaluate the AR CapsNet in the CIFAR-10 classification tasks according to the various setting of hyperparameters. The evaluated AR CapsNet architecture is the same as the models mentioned in Section 3.3.1 and 3.3.2. The primary caps layer has eight channels of 16-dimensional capsules, and the conv caps layer has eight capsule channels. In the setting of hyperparameters, the *Conv caps layer* denotes the number of conv caps layer between the primary caps layer and fully conv caps layer. In every model with at least one conv caps layer, the first conv caps layer decreases the spatial size of the capsule layer by adopting a 3x3 convolution of stride 2 in the convolutional transform. The *Capsule dim* denotes the capsule dimension of the conv caps layer and fully conv caps layer.

Robustness All the AR CapsNet models trained with CIFAR-10 dataset show decent performance in Table 3.2. Increasing capsule dimension and the number of conv caps layer lead to an improvement in the test accuracy. The AR CapsNet model with four conv caps layer shows 86.37% accuracy with 16-dimensional capsules and 87.19% accuracy with 32-dimensional capsules. The AR CapsNet with four conv caps layer and 32-dimensional capsules shows the best results of 87.19% in C10 and 88.61% in C10+. Also, the AR CapsNet model with no conv caps layer has more parameters than the model with four conv cap layer and shows the worst performance. The features in

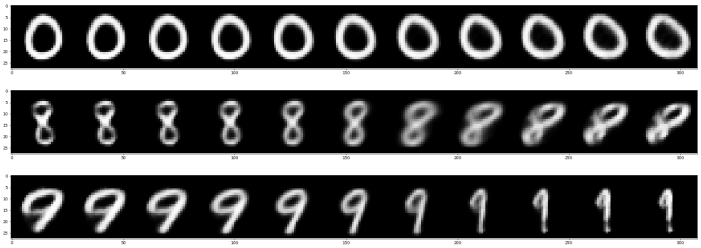


Figure 3.4: Decoder outputs according to dimension perturbations. We observed the variations of decoder output as we perturbed one dimension of the output capsules by steps of $0.05\sqrt{D^L}$ from $-0.25\sqrt{D^L}$ to $+0.25\sqrt{D^L}$. The perturbation leads to the combination of variations in the decoder output images. (e.g., rotation, thickness, etc.).

the primary caps layer has a large spatial size. Thus, the fully conv caps layer connected to the primary caps layer assigns excessive parameters, and this causes overfitting.

3.3.4 Transformation Equivariance

Dimension perturbation To prove that our proposed model is transformation equivariant, we executed experiments on the MNIST model as in [33]. We observed the variations of decoder output as we perturbed one scalar element of the output capsules (Figure 3.4). The experiments in the [33] perturbed one scalar element from -0.25 to 0.25. Since the output capsules of the AR CapsNet have lengths of up to $\sqrt{D^L}$ compared to 1 in [33], we perturbed one scalar element from $-0.25\sqrt{D^L}$ to $0.25\sqrt{D^L}$ where D^L denotes the capsule dimension of output capsules. Figure 3.4 shows that some dimensions of the output capsules represent variations in the way the digit of the corresponding class is instantiated. Some dimensions of the output

Digit	Rot+	x+	y+	Rot-	x-	y-
8	0.89	0.89	0.91	0.86	0.87	0.86
5	0.89	0.78	0.73	0.84	0.88	0.86
avg	0.88 (0.89)	0.86 (0.88)	0.84 (0.80)	0.83 (0.81)	0.83 (0.84)	0.84 (0.84)

Table 3.3: The average of relative ratio $\{r_i\}$ for each combination of digit and transformation. The avg represents the average of all 10,000 test samples for each affine transformation. We report the results of models trained on the MNIST+ dataset. A high relative ratio implies the difference vectors are strongly aligned. For random vectors, the average of relative ratio $\{r_i\}$ is 0.311 and standard deviation is 0.262.

capsules represent the localized skew in digit 0, the rotation and the size of the higher circle in digit 8, and the rotation, thickness, and skew in digit 9.

Alignment ratio Each scalar element of the output capsules represents a combination of variations such as rotation, thickness, and skew. (Digit 9 in Figure 3.4) Since the length of the output capsules is basis-invariant, the transformation on an input image could be represented in coordinates of any basis. To further test the transformation equivariance of the AR CapsNet, we tested whether the difference in the output capsules caused by applying a transformation on an input image is aligned in one direction.

Let $\{T_i\}_{i=1,\cdots,N}$ be a set of affine transformations on an input image x. We denote the difference between the output capsules $\mathbf{u}_n^L(T_i(x))$ and $\mathbf{u}_n^L(x)$ as $\mathbf{v}_i(x)$ where n denotes the corresponding class of x.

$$\mathbf{v}_i(x) = \mathbf{u}_n^L(T_i(x)) - \mathbf{u}_n^L(x) \tag{3.3.1}$$

We denote the concatenation of $\mathbf{v}_i(x)$ along the row axis as $\mathbf{V}$. In order to

obtain a representative unit vector $\tilde{v}$ of $\{\mathbf{v}_i(x)\}$, we apply a Singular-Value Decomposition(SVD) on matrix $\mathbf{V}$.

$$\mathbf{c}, \tilde{v} = \arg\min_{c_i, \tilde{v}} \sum_i ||\mathbf{v}_i(x) - c_i \cdot \tilde{v}||_2^2 \tag{3.3.2}$$

$$= \arg\min_{c_i, \tilde{v}} ||\mathbf{V} - \mathbf{c} \cdot \tilde{v}^T||_F^2 \tag{3.3.3}$$

where F denotes the Frobenius norm, $\mathbf{c} = (c_1, \cdots, c_N)^T$, and $c_i \in \mathbb{R}$. The exact solution of this low rank approximation problem is the first right-singular vector $\tilde{v}$ of $\mathbf{V}$. This experiment is similar to the Principal Component Analysis(PCA) except that we do not subtract the mean for each columns of $\mathbf{V}$. The align vector $\tilde{v}$ corresponds to the principal vector of PCA. We observed the relative ratio r_i of principal component of $\mathbf{v}_i(x)$ to the vector norms $||\mathbf{v}_i(x)||_2$.

$$r_i = \frac{|\mathbf{v}_i(x) \cdot \tilde{v}|}{||\mathbf{v}_i(x)||_2} \tag{3.3.4}$$

We randomly chose 10,000 images from the test set. For each test image, we generated five images by applying an affine transformation and observed the relative ratio r_i. In Table 3.3, Rot($\pm$) represents $\pm\{5, 10, 15, 20, 25\}$ degrees rotations and x($\pm$) represents a horizontal translation up to $\pm$5 pixels. y($\pm$) represents a vertical translation up to $\pm$5 pixels as well. We observed the average of relative ratio r_i for each combination of digit and transformation. Table 3.3 shows the average of relative ratio r_i for two digits (highest : digit 8, lowest : digit 5) and the average for 10,000 test samples for each transformation. As a reference, we generated five random vectors from the standard multivariate normal distribution. We conducted the same exper-

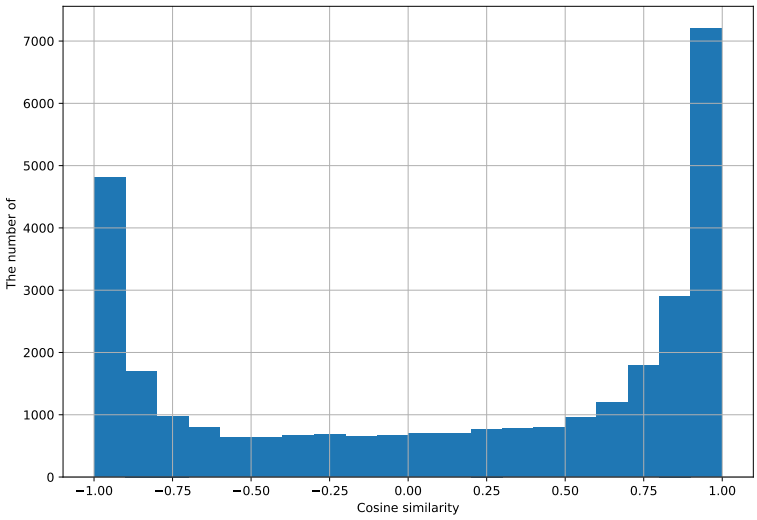


Figure 3.5: Distribution of cosine similarity between unit align vectors $\tilde{v}$ of positive and negative affine transformations.

iment for random vectors for 1,000 times as well. We obtained an average of 0.311 and a standard deviation of 0.262 for random vectors. Even for the worst-case digit 5, every transformation shows a significantly higher relative ratio r_i of 0.73 in y+ than the random vectors. This result implies that the difference vectors are strongly aligned in one direction. Therefore, AR CapsNet encodes affine transformations on an input image by some vector components. Also, we report the results of models trained on the MNIST+ dataset in the (). The models trained on the MNIST+ show comparable relative ratio r_i to those trained on the MNIST. This result shows that AR CapsNet encodes affine transformations even without observing transformations during training.

An interesting observation is that AR CapsNet is transformation equiva-

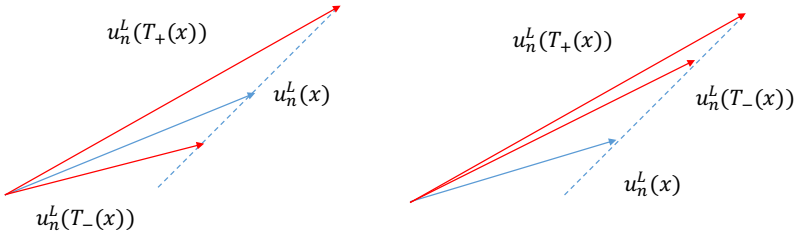


Figure 3.6: Output capsules $\mathbf{u}_n^L$ when the cosine similarity between align vectors of positive and negative transformations is -1 or 1. T_+ and T_- represent the positive and negative transformation. **Left:** cosine similarity -1. **Right:** cosine similarity 1.

lent but do not distinguish the positive and negative transformations. Figure 3.5 shows the histogram of the cosine similarity between the align vectors of positive and negative transformation.[6] We observed two peaks around -1 and 1. The cosine similarity of -1 and 1 imply that positive and negative transformations are encoded in one dimension. However, the cosine similarity of 1 suggests that the difference vectors of positive and negative transformations have the same direction.(Figure 3.6) We leave the explanation of this observation to future work.

[6]The direction of the first right-singular vector $\tilde{v}$ is given by $\mathbf{v}_i(x) \cdot \tilde{v} > 0$ in for each positive and negative transformation.

Chapter 4

Conclusion and Future Works

In this thesis, We try to broaden our understanding of the latent space of the deep neural network. In the first chapter, We experimentally investigate the relationships with the decision boundary in the latent space of the deep neural network through several toy experiments. The decision boundary is obtained by using and adversarial attack methods in the latent space where deep neural network embeds. We analyze the relationship between the decision boundary and the latent space manual obtained by perturbing the image. As a result, we can experimentally confirm that The latent manifold is locally parallel to the decision boundary. It can be estimated that there is curvature in areas that are a little far away from the original manifold.

In the second chapter, the characteristics of the latent space is examined by constraining a network architecture design. We proposed a new network

module called an attention-style capsule net with improved capsulenet. By introducing the attention routing and capsule activation, AR CapsNet obtained a higher accuracy compared to CapsuleNet while using fewer parameters and less training time. The attention routing is an effective way to route between capsules because it only compares capsules of the same spatial location. The values of each capsule are perturbed to determine what the actual image space latent does the deep neural network model analyzes and maps to the capsule latent space. The capsule activation is based on the assumption that the capsule-scale activation can extract transformation equivariant features even if it is not orientation-preserving. This assumption distinguish the capsule activation from the squash activation function and its variant. By showing that the difference vectors are strongly aligned in one direction, we proved that AR CapsNet encodes transformation information in some dimensions.

We experimentally analyze the relationship between latent space and decision boundaries. Based on the experimental results, the theoretical analysis of the latent space of the deep learning network remains.

Bibliography

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one 10*(7), e0130140.

[2] Mohammad Taha Bahadori (2018). Spectral capsule networks.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[4] Wieland Brendel, Jonas Rauber, and Matthias Bethge (2017). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*.

[5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

[6] Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll

(2019). Guessing smart: Biased sampling for efficient black-box adversarial attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4958–4966.

[7] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian (2018). Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pp. 839–847. IEEE.

[8] Jianbo Chen, Michael I Jordan, and Martin J Wainwright (2019). Hopskipjumpattack: A query-efficient decision-based attack. arxiv 2019. *arXiv preprint arXiv:1904.02144 3*.

[9] Jaewoong Choi, Hyun Seo, Suii Im, and Myungjoo Kang (2019). Attention routing between capsules. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0.

[10] George Cybenko (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*(4), 303–314.

[11] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li (2018). Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9185–9193.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014).

Generative adversarial nets. *Advances in neural information processing systems 27*.

[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

[16] Geoffrey Hinton and Sam T Roweis (2002). Stochastic neighbor embedding. In *NIPS*, Volume 15, pp. 833–840. Citeseer.

[17] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang (2011). Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pp. 44–51. Springer.

[18] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst (2018). Matrix capsules with em routing.

[19] Sergey Ioffe and Christian Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119.

[21] Alexey Kurakin, Ian Goodfellow, and Samy Bengio (2016). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.

[22] Rodney LaLonde and Ulas Bagci (2018). Capsules for object segmentation. *arXiv preprint arXiv:1804.04241*.

[23] Yann LeCun, Corinna Cortes, and CJ Burges (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist 2*.

[24] Hongyang Li, Xiaoyang Guo, Bo DaiWanli Ouyang, and Xiaogang Wang (2018). Neural network encapsulation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 252–267.

[25] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh (2018). Adv-bnn: Improved adversarial defense through robust bayesian neural network. *arXiv preprint arXiv:1810.01279*.

[26] Minh-Thang Luong, Hieu Pham, and Christopher D Manning (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

[27] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris

Tsipras, and Adrian Vladu (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog 1*(8), 9.

[30] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144.

[31] Herbert Robbins and Sutton Monro (1951). A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

[32] Sam T Roweis and Lawrence K Saul (2000). Nonlinear dimensionality reduction by locally linear embedding. *science 290*(5500), 2323–2326.

[33] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton (2017). Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pp. 3856–3866.

[34] Hasim Sak, Andrew W Senior, and Françoise Beaufays (2014). Long

short-term memory recurrent neural network architectures for large scale acoustic modeling.

[35] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pp. 618–626.

[36] Adi Shamir, Odelia Melamed, and Oriel BenShmuel (2021). The dimpled manifold model of adversarial examples in machine learning. *arXiv preprint arXiv:2106.10151*.

[37] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje (2017). Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pp. 3145–3153. PMLR.

[38] Karen Simonyan and Andrew Zisserman (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[39] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg (2017). Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*(1), 1929–1958.

[41] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber (2015). Training very deep networks. In *Advances in neural information processing systems*, pp. 2377–2385.

[42] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation 23*(5), 828–841.

[43] Joshua B Tenenbaum, Vin De Silva, and John C Langford (2000). A global geometric framework for nonlinear dimensionality reduction. *science 290*(5500), 2319–2323.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008.

[45] Dilin Wang and Qiang Liu (2018). An optimization view on dynamic routing between capsules.

[46] Eric Wong, Leslie Rice, and J Zico Kolter (2020). Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*.

[47] Edgar Xi, Selina Bing, and Yang Jin (2017). Capsule network performance on complex data. *arXiv preprint arXiv:1712.03480*.

[48] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan (2019). Theoretically principled trade-off

between robustness and accuracy. In *International Conference on Machine Learning*, pp. 7472–7482. PMLR.

[49] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929.

국문초록

딥러닝 모델이 많은 도메인에서 압도적인 성능을 내는데 반해 딥러닝 모델이 어떤 잠재 공간을 만드는지, 어떤 특징를 학습 하는지 정확히 알려지지 않고 있다. 딥러닝의 학습 과정에 대한 정확한 이해는 현재까지 완벽하지 않고 열린 문제이다.

이번 연구에서는 크게 2가지 방법으로 딥러닝이 학습한 잠재공간에 대해 이해를 넓히고자 하였다. 첫번째 챕터에서는 딥러닝 모델의 잠재공간에서 결정경계를 이용하여 잠재공간의 특성을 여러 실험들을 통해 분석하였다. 적대적 공격 방법을 이용해서 결정경계 벡터를 구하고 인풋에 노이즈를 추가하여 잠재공간의 다양체 벡터를 구하였다. 결정경계 벡터와 잠재공간의 다양체 벡터 사이의 관계를 통해 잠재공간의 다양체의 구성을 분석하였다.

두번째 챕터에서는 캡슐이라는 특수한 설계로서 잠재공간을 제한하여 학습된 잠재공간의 특징을 살펴보았다. 네트워크 구조는 캡슐넷을 개선한 어텐션 스타일의 캡슐넷을 제안하고, 각 캡슐들을 값을 변동시켜 딥러닝 모델이 실제 이미지공간의 어떤 특징을 분석하여 캡슐 잠재공간으로 매핑하는지 확인하고 인풋 데이터에서의 변형에 대한 캡슐값의 변동을 분석하였다.

주요어휘: 잠재공간 다양체, 적대적 공격, 결정 경계, 캡슐넷

학번: 2017-28074