UNIVERSITY OF OKLAHOMA
GRADUATE COLLEGE

ENHANCED CAPSULE-BASED NETWORKS AND THEIR APPLICATIONS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

ZHIHAO ZHAO

Norman, Oklahoma

2022

ENHANCED CAPSULE-BASED NETWORKS AND THEIR APPLICATIONS

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Samuel Cheng, Chair

Dr. Gregory MacDonald

Dr. Shangqing Zhao

Dr. Bin Zheng

Acknowledgments

First, I would like to specially thank my major advisor Dr. Samuel Cheng, who gives me insightful guidance and significant supports throughout my whole Ph.D. study and research. His comprehensive insights of scientific literacy have greatly helped me to become a qualified PhD student.

I would like to thank all my committee members, Dr. Gregory MacDonald, Dr. Shangqing Zhao, Dr. Bin Zheng, for their time and efforts reviewing and discussing my research. Their guidance is immeasurable both during the writing of this dissertation and over the entire course of my time at the University of Oklahoma. Especially, Dr. Shangqing Zhao went through my papers thoroughly many times and greatly improved them.

I cherish the numerous happy moments with my friends: Xuxin Chen, Eddie Clark, Dong Han, Raidel Martinez, Renee Wagenblatt, Karen Winfrey, Ken Winfrey, Haoliang Zhang, Lu Zhang, Jiaomiao Zhao, Wenqing Zhu.

Special thanks to my father and brother who greatly helped my mother to fight with her lung cancer. Hope my mother overcomes her cancer in the future.

Table of Contents

List of Tables

List of Figures

Abstract

Current deep models have achieved human-like accuracy in many computer vision tasks, even defeating humans sometimes. However, these deep models still suffer from significant weaknesses. To name a few, it is hard to interpret how they reach decisions and it is easy to attack them with tiny perturbations.

A capsule, usually implemented as a vector, represents an object or object part. Capsule networks and GLOM consist of classic and generalized capsules respectively, where the difference is whether the capsule is limited to representing a fixed thing. Both models are designed to parse their input into a part-whole hierarchy as humans do, where each capsule corresponds to an entity of the hierarchy. That is, the first layer finds the lowest-level vision patterns, and the following layers assemble the larger patterns till the entire object, e.g., from nostril to nose, face, and person.

This design enables capsule networks and GLOM the potential of solving the above problems of current deep models, by mimicking how humans overcome these problems with the part-whole hierarchy. However, their current implementations are not perfect on fulfilling their potentials and require further improvements, including intrinsic interpretability, guaranteed equivariance, robustness to adversarial attacks, a more efficient routing algorithm, compatibility with other models, etc.

In this dissertation, I first briefly introduce the motivations, essential ideas, and existing implementations of capsule networks and GLOM, then focus on addressing some limitations of these implementations. The improvements are briefly summarized as follows. First, a fast non-iterative routing algorithm is proposed for capsule networks, which facilitates their applications in many tasks such as image classification and segmentation. Second, a new

architecture, named Twin-Islands, is proposed based on GLOM, which achieves the many desired properties such as equivariance, model interpretability, and adversarial robustness. Lastly, the essential idea of capsule networks and GLOM is re-implemented in a small group ensemble block, which can also be used along with other types of neural networks, e.g., CNNs, on various tasks such as image classification, segmentation, and retrieval.

Chapter 1: Introduction

1.1 The limitations of convolutional neural networks

Convolutional neural networks (CNNs) have been very successful in a wide range of vision tasks in the past decade, such as image recognition [9, 10], object detection [11, 12], and instance segmentation [13]. This success is largely due to its architecture consisting of multiple layers of feature detectors that replicate the same set of learnable weights on numerous local patches [14, 15]. The stochastic gradient descent technique also plays an important role in CNNs' success, enabling training deep CNNs with a huge amount of data via optimizing the task-specific loss functions. Despite the success of CNNs in so many tasks, there are many limitations for CNNs as the following.

Firstly, it is hard to either explain why the structure of CNNs leads to a certain prediction or interpret how CNNs make their predictions in great detail, which makes CNNs work as black boxes, as in Figure 1.1. Although there exist some interpretation methods for CNNs, they mainly explore the interaction between the input and output (or a higher layer's activation maps). To name a few, finding images or image patches that maximize the target neurons' activation [16, 17, 18], displaying the loss function's gradient w.r.t the input image [19, 20], and fitting the network's prediction on a single image's various deformations using a simpler model, e.g., a linear classifier [21]. All these methods try to interpret the black-box network after they are already trained. In contrast, an ideal network should be a self-explanatory white box. Consequently, CNNs are not sufficient for high stake decisions, such as medical diagnosis assistance.

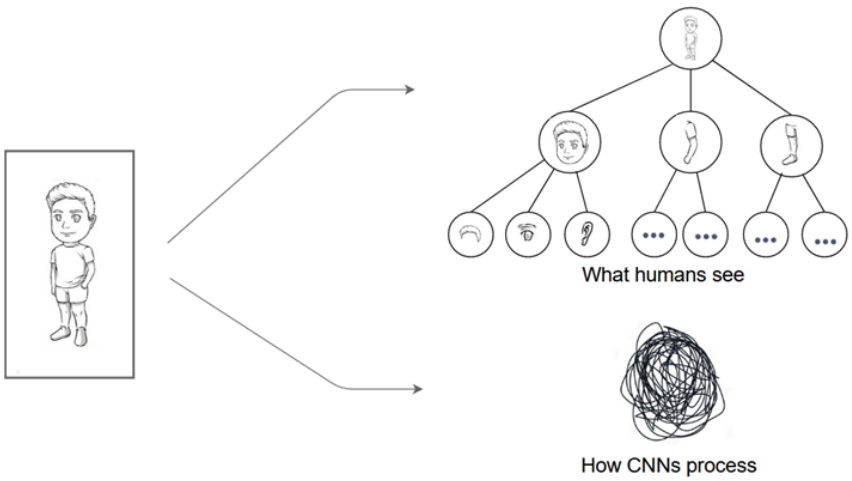


Figure 1.1. Comparison of how humans and convolutional neural networks process the same image.

Secondly, CNNs are vulnerable to adversarial attacks, which makes them potentially unsafe to be used in security-critical areas. A tiny amount of unpredictable shifts presented in a test image can fool the CNNs to make completely wrong predictions. This makes the adversarial attack possible and threatens the real-world applications [2]. As the example in Figure 1.2, the object person is correctly recognized without the attack. But after adding small perturbations to the original inputs, CNNs easily change their correct prediction to the wrong category gibbon. Interestingly, an adversarial example generated with a specific CNN can fool many other CNNs. However, the two images are apparently the same to humans: the vision system of human will identify each object as a panda.

Thirdly, CNNs are not robust to affine transformations. In other words, they cannot generalize their capability to every possible affine transformation of an image, as in Figure 1.3. Human beings watch a few samples of an object and can recognize it from any viewpoint next time. But CNNs are just "remembering" the annotation of human-fed samples and cannot recognize objects filmed at novel viewpoints that are not covered in the training set. Data augmentation partially alleviates this problem for 2D affine transformations, whereas the usual data augmentation methods can not imitate 3D affine transformation [3].

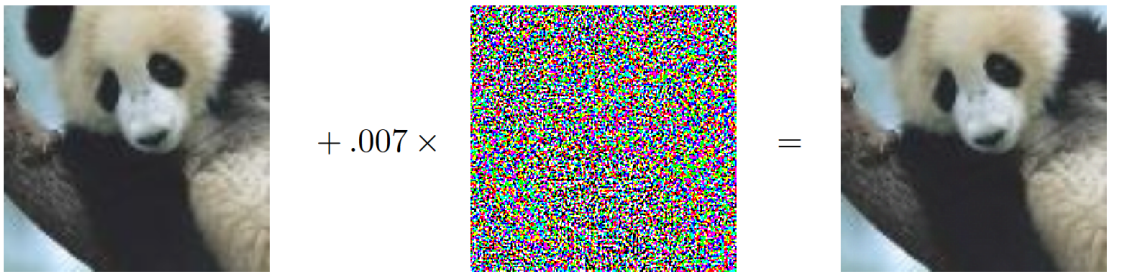


Figure 1.2. Illustration of adversarial attack to CNNs, borrowed from [2].

1.2 The big pictures of capsule networks and GLOM

As explained in the previous section, the three limitations of CNNs can be solved by human beings. If we build deep models that parse their inputs like humans, they will potentially solve these limitations of CNNs. There is strong psychological evidence that humans parse visual scenes into part-whole hierarchies [22, 23]. The part-whole hierarchy, take the person as an example, consists of parts (e.g., face), subparts (e.g., mouth), sub-subparts (e.g., a corner of the mouth), etc. Therefore, capsule networks and GLOM aim to parse objects into the part-whole hierarchy, where the adjacent layers are connected by high-dimensional coincidence filtering.

1.2.1 Parsing objects into the part-whole hierarchy

Capsule networks [24, 25, 5, 6, 4] and GLOM [4] aim to be human-like through the part-whole hierarchy. The capsules in capsule networks or embeddings in GLOM are supposed to spatially correspond to nodes of a parse tree that parses the network's input object into a part-whole hierarchy. A toy example can be that the person object is parsed into a three-layer capsule network or GLOM (in practice we may need more layers). Capsules or embeddings of the highest layer represent the entire person, capsules or embeddings of the

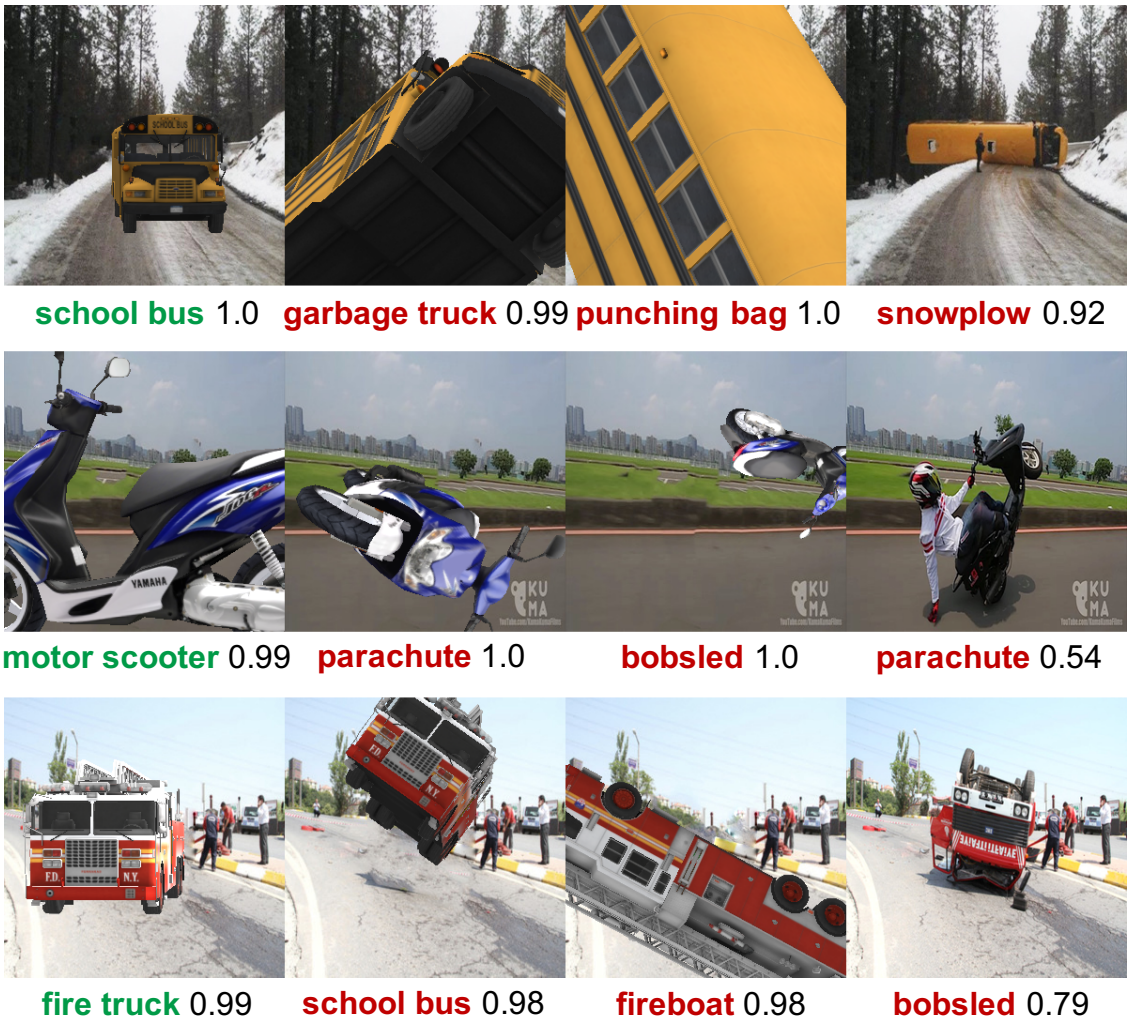


Figure 1.3. Illustration of CNNs' prediction to images after viewpoint change, borrowed from [3]. The first column shows canonical poses of objects, and the other columns show object after viewpoint change. The correct and wrong classifications are colored in green and red, respectively.

middle layer represent the mouth, nose, arm, etc., and capsules or embeddings of the lowest layer represent the line, circle, etc. The learnable weights in capsule networks or GLOM are supposed to learn the relationship between object parts represented by the capsules or embeddings in adjacent layers, e.g., the nose-to-person relationship.

The process that capsule networks and GLOM process their input is also named "inverse computer graphics" [26]. In computer graphics, a computer renders a 3D object onto a 2D

screen with a given viewpoint. Capsule networks and GLOM are supposed to work in the inverse direction that it first deduces the 3D coordinates from pixels, then multiplies the 3D coordinates by the inverse of the transformation matrix to get the pose of the whole object.

1.2.2 High-dimensional coincidence filtering

Capsule networks and GLOM activate capsules or columns in a different way than CNNs. The convolution operation in CNNs that connects the neurons in consecutive layers can be expressed as $\mathbf{w}^T\mathbf{x} + b$, where $\mathbf{w}$ and b are the convolution kernel and bias for the data $\mathbf{x}$ in a 2D local window. This convolution operation activates a neuron when sufficient evidence x_i has been found in $\mathbf{x}$. In contrast, capsule networks and GLOM use "high-dimensional coincidence filtering" that has essentially the same core idea as Hough transform. They activate the capsule or column by finding a bunch of similar predictions from a huge number of incoming predictions that are produced from the layer below. For example, when the layer below has detected the head, arm, and legs, these parts may produce predictions of a person or helmet (from the head), person or monkey (from the arm), person or dog (from the legs), then the prediction person is the coincidence.

1.2.3 Potential advantages

Capsule networks and GLOM are supposed to work like humans, as the general designing philosophy explained above. Therefore, they potentially have the following advantages as humans.

The first advantage is interpretability. One definition of interpretability is "the degree to which a human can understand the cause of a decision [27]." Because part-whole relationships are the way how a human makes decisions, models using part-whole relationships will be interpretable as long as there is a way to clearly visualize the correspondence between model components (both model activities and model weights) and the part-whole hierarchy. For

example, the situation that various faces in different images use the same transformation matrix to predict the person object means that this transformation matrix encodes an arm-to-person relationship. This interpretability further facilitates other possible applications. For example, the model may be pruned based on the importance of each part-whole relationship.

The second advantage is robustness to adversarial attacks. By adding a small perturbation to the original input, well-trained deep CNNs produce wrong predictions. However, this small perturbation will never mislead humans. Capsule networks and GLOM mimic humans, and thus potentially prevent adversarial attacks. Like humans, capsule networks and GLOM predict the object as the consensus between various object parts' predictions towards the full object. For example, if "head", "arm", "hand", "leg", and "foot" have the same prediction of "person", this agreement will activate the "person" prediction. This prediction fashion is robust as perturbations on the object parts will not be accumulated in the agreement quantification. This is because the person is activated by finding a coincidence instead of the sum-like convolution operations in CNNs. In addition, this coincidence between multiple parts' predictions still holds firmly when some parts are occluded, making it robust to occlusion.

The third advantage is equivariance with respect to the input object's affine transformation. Equivariance is originally the mathematical notion of symmetry for functions. A function f is said to be an equivariant map if the result of transforming the input x and then computing the function is the same as first computing the function and then transforming the output, i.e., $f(Tx) = Tf(x)$ where T is the matrix that performs the transformation. Capsule networks and GLOM with the following mechanism can be equivariant and generalize well to novel viewpoints that the network has never seen: i) the capsules or embeddings are supposed to be viewpoint-equivariant, whose activities contain the orientation information of the object or object parts; ii) the transformation weights are supposed to encode the viewpoint-invariant spatial relationship between a part and a whole as the coordinate trans-

formation between intrinsic coordinate frames assigned to the part and the whole. Then, when the viewpoint changes, the orientation of each object part or the entire object encoded in every capsule or column changes to the new viewpoint, so that the viewpoint-invariant spatial relationship learned from one viewpoint works for the new viewpoint.

1.3 Improve capsule networks and GLOM

Despite the promising future, the ambitious goals of capsule networks and GLOM have not been well achieved. Especially, GLOM is proposed as an imaginary system without any programming-level implementation. This leaves ample space to improve the existing capsule networks and practically achieve the imaginary design goals of GLOM.

1.3.1 A more efficient routing algorithm for capsule networks

First, the iterative routing algorithms in capsule networks are not time efficient, preventing their usage in very practical tasks such as medical image segmentation and ImageNet classification where large-size images, e.g., 512×512 or a large number of images are involved. One reason for this phenomenon is the inefficiency of iterative routing algorithms used in capsule networks. The iterative routing algorithms, e.g., dynamic routing [5] and EM routing [6], use multiple iterations to find the agreements between incoming predictions. As each iteration adds additional operations, the training and inference time can increase dramatically with more routing iterations, especially for large-size images. In contrast, non-iterative routing algorithms [28, 29] compute the capsule's content with a straight-through process. By simplifying the multiple iterations to a single forward pass, non-iterative routing procedures mitigate the computational burden of iterative routing procedures. Therefore, a non-iterative cluster routing is proposed for capsule networks, making it more convenient to apply capsule networks to large-size images, e.g., medical image segmentation.

1.3.2 Building equivariant capsule networks or GLOM

Second, the equivariance is not theoretically guaranteed. Most capsule networks and GLOM literature use intuitive heuristics to learn transformation-robust spatial relations among objects' components. But the following design goals are usually not met without introducing new inductive bias to the network: i) the transformation weight should encode the viewpoint-invariant spatial relationship between a part and a whole (or between a sub-part and a part), where the relationship can be, for example, the coordinate transformation between intrinsic coordinate frames of the part and the whole; ii) the capsule or embedding should encode viewpoint-equivariant orientation of corresponding part or whole that it represents. Crucially, the problem of correctly deducing 3D coordinates from the pixels of an image or video is not addressed. Capsule networks and GLOM use trainable transformation kernels applied to local receptive fields, but the receptive field coordinates are agnostic to the object's 3D pose.

The 3D coordinates, however, are directly provided in another kind of data, the point clouds. A point cloud is a collection of 3D data points representing a 3D object, usually scanned by 3D laser scanners. Capsule networks that take advantage of a point cloud's 3D coordinates have successfully shown equivariance [30, 31]. One goal of this dissertation is to achieve equivariance in GLOM using point clouds.

1.3.3 Building adversarial robust networks

Third, one design goal of capsule networks and GLOM is to increase the networks' robustness to adversarial attacks. CNNs are vulnerable to adversarial attacks due to their weight-sum fashion convolutional kernels, but the high-dimensional coincidence filtering is just the opposite of weight-sum-like processes. The coincidence filtering compares the vector's similarity, e.g., cosine similarity, in the high-dimensional vector space. A small per-

turbation to a vector will not change the similarity between two vectors, and this robust similarity is then the basis for deducing large object parts till the entire object. However, for capsule networks, the robustness is only evaluated on the smallNORB data set [6] and is challenged by vote attacks [32]. For GLOM, there is currently no work that explores its robustness to adversarial attacks in practice. Thus, there is plenty of room to improve the adversarial robustness of capsule networks and GLOM.

1.3.4 Encoding part-whole relationships into learnable transformation weights

Fourth, the part-whole relationships are not very clearly represented in transformation weights [33, 34]. For capsule networks, they have no module that guarantees the transformation weights to encode the part-whole relationship, which may probably lead to a fuzzy representation. The transformation weights are just learned through stochastic gradient descent to minimize the loss function. Whether the part-whole relationship is learned into the transformation weight has not been proven yet, not to mention quantifying to which extent the relationship is discovered. It is also possible that the capsules just learn a blended representation of various things, e.g., human eyes and car headlights. For GLOM, an imaginary propagation-fashion strategy is proposed for learning interpretable object parts, but how the part-whole relationship can be learned into the network is not explored. This leaves the blank to learn the part-whole relationship, visualize which relationship is learned in which transformation weight, and even quantitatively measure how precisely the weight encodes the relationship.

Especially, the point cloud is an ideal beginning among the various types of data. This is because the equivariant representation of object parts can be theoretically guaranteed as in [30, 31]. Then the part-whole relationships, if learned to be viewpoint-invariant, can generalize to novel viewpoints that are not covered in the training data set. One goal of

this dissertation is to learn and visualize viewpoint-invariant part-whole relationships in equivariant capsule networks or GLOM using point clouds.

1.3.5 Pushing performance on practical tasks

Fifth, the performance of capsule networks and GLOM on large and practical datasets need to be improved. CNNs have been successfully applied to almost every computer vision task. But capsule networks and GLOM have not beaten CNNs many times in terms of classification accuracy, segmentation precision, etc. This leads to some doubt about their practical superiority to CNNs. For capsule networks, one reason is that the original computationally-expensive routing algorithms, in terms of GPU memory and the network's forward propagation time, prevent building deep networks for high-resolution images. An image of size 224×224 needs a huge number of $(224-2) \times (224-2)$ parallel routing processes with a receptive field of 3×3 for just the network's first capsule layer, where the number of parallel routing processes is the same as the number of sliding windows during the convolution operation in CNNs. However, it is possible to apply capsule networks with the following two modifications. First, simplify the original iterative routing algorithms into one single iteration. Second, share parameters between different capsules in the same receptive field. One goal of this dissertation is to implement these two modifications on capsule networks and hopefully achieve high segmentation precision on medical images of size 512×512.

1.3.6 Pushing the compatibility of capsule networks and GLOM with other types of networks

Sixth, capsule networks and GLOM are attractive and promising, but there are still many other popular networks such as CNNs and transformers. Furthermore, there are many charming tasks that capsule networks and GLOM have not touched too much, such as self-supervised learning and neural radiance fields [35]. It would greatly increase the impact

of capsule networks and GLOM by incorporating them with other kinds of networks and extending their usage to more tasks. Although capsule networks and GLOM are different types of networks compared to CNNs and transformers, it is possible to isolate the core ideas of capsule networks and GLOM and reimplement these core ideas to be compatible with CNNs or transformers. One goal of this dissertation is to find out the essential ideas of capsule networks and GLOM and accomplish their compatibility with other types of neural networks.

1.4 Contribution and organization

The organization of this dissertation is summarized as the following.

- First, the introduction section gives the motivations, general ideas, and potential advantages of capsule networks and GLOM.
- Second, the original mechanisms of capsule networks and GLOM are illustrated in more detail, e.g., how the consecutive layers are connected.
- Third, a novel non-iterative routing procedure is proposed for capsule networks, which is for the motivations in Sections 1.3.1 and 1.3.5. It is significantly faster than the original iterative dynamic routing and EM routing while producing better accuracy at the same time. With the decrease in the number of computations, capsule networks can be used to segment images of a large size, e.g., 512×512.
- Fourth, an equivariant, interpretable, and adversarial robust network is proposed for point clouds, which is for motivations in Sections 1.3.2, 1.3.4, and 1.3.3.
- Fifth, a group ensemble block is proposed and applied to image classification, segmentation, and retrieval, which is for motivations in Section 1.3.6. In addition, this group

ensemble block can also be used along with other techniques such as self-supervised learning.

- The last section gives a conclusion and future directions.

The original works in the third, fourth, and fifth chapters are summarized as the following.

1.4.1 Capsule networks with fast non-iterative cluster routing for medical image segmentation

Capsule networks use routing algorithms to flow information between consecutive layers. In the existing routing procedures, capsules produce predictions (termed votes) for capsules of the next layer. In a nutshell, the next-layer capsule's input is a weighted sum over all the votes it receives. In this paper, we propose non-iterative cluster routing for capsule networks. In the proposed cluster routing, capsules produce vote clusters instead of individual votes for next-layer capsules, and each vote cluster sends its centroid to a next-layer capsule. Generally speaking, the next-layer capsule's input is a weighted sum over the centroid of each vote cluster it receives. The centroid that comes from a cluster with a smaller variance is assigned a larger weight in the weighted sum process. Compared with the state-of-the-art capsule networks, the proposed capsule networks achieve the best accuracy on the Fashion-MNIST and SVHN datasets with fewer parameters, and achieve the best accuracy on the smallNORB and CIFAR-10 datasets with a moderate number of parameters. The proposed capsule networks also produce capsules with disentangled representation and generalize well to images captured from novel viewpoints. The proposed capsule networks also preserve 2D spatial information of an input image in the capsule channels: if the capsule channels are rotated, the object reconstructed from these channels will be rotated by the same transformation.

1.4.2 An equivariant, adversarial robust, and interpretable architecture for point clouds

Borrowing part-whole relationships into neural networks provides a chance to access the internal activities of models. The recently proposed GLOM is an imaginary system that parses the part-whole hierarchy of an object into the model's internal embeddings. However, the parsing process of this hierarchy remains obscure by the black-box autoencoders in the original GLOM. In this dissertation, a new adversarial-robust and interpretable architecture is proposed for point cloud processing, named Twin-Islands. The proposed Twin-Islands not only parses the object into embedding islands, but also makes the parsing process transparent via encoding the real-world part-whole relationships with some transformation weight islands. The transparent parsing process makes the model intrinsically interpretable. Furthermore, the object is classified through plurality voting where the votes are produced from the embedding islands and transformation weight islands. The plurality voting rules out a number of incorrect votes, in contrast to the traditional models that accumulate the perturbations in every object part. The visualization experiment shows the correspondence between the real-world part-whole relationships and the transformation weight islands. One of its applications is also provided which prunes the transformation weights by the importance of corresponding part-whole relationships. Adversarial perturbation and deletion experiments show its superiority over CNN-based models.

1.4.3 Pushing the compatibility by group ensemble block

The neurons that conjointly compose a capsule collaborate with each other, but neurons of different capsules are isolated. This motivates us to propose a group ensemble block in the same spirit and apply it to other types of neural networks like CNNs. This group ensemble block essentially samples random subspaces from the original vector, and merges the processed results of these subspaces into the output vector. Operations such as parallel

computation and parameter sharing are adopted to decrease the number of parameters and computations. A brief introduction from the perspective of the random subspace is given in the next paragraph.

The conventional ensemble learning methods obtain the diversity among different ensembles' output through varying architectures or weights. However, these methods require too much computation and memory. Instead of varying network architectures or weights, we find that one can also achieve diversity from data difference, i.e., the difference among many random sampled subspaces. Moreover, compared to previous ensembling methods, one can dramatically reduce the overhead when projecting to random subspaces because the subspaces can be sampled not from the initial input, e.g., the input image, but just before the end of the network. Although the network layers before the projections are shared for every subspace, they receive diverse gradients from the diverse subspaces in the backpropagation. To implement our idea, we propose the group ensemble block to execute subspace sampling inside a small and easy-to-plug block that needs little computation. Furthermore, we show that our group ensemble block is complementary to existing methods by providing its integration with class-level and instance-level discrimination. Experiments show that the proposed group ensemble block achieves state-of-the-art accuracy on the CIFAR-100 and ImageNet datasets for the coarse-to-fine image retrieval problem, where the model is trained with coarse-level annotation (e.g., trees) and evaluated with fine-grained category (e.g., maple trees and oak trees).

1.4.4 Contributions

The contributions of this dissertation are summarized as the following:

- Efficiency: A fast non-iterative routing algorithm is proposed for capsule networks that largely relieves the computation burden of the iterative routing algorithms. This

routing algorithm increases capsule networks' performance on multiple tasks such as image classification and segmentation.

- Equivariance: The proposed new architecture, named Twin-Islands, is equivariant w.r.t the input object's orientation for point clouds.

- Interpretability: Visualization of the proposed Twin-Islands shows that model weights encode part-to-whole relationships and model activities encode the object's parts and whole.

- Adversarial robustness: By utilizing the high-dimensional coincidence filtering, the proposed Twin-Islands is robust to small perturbations produced in adversarial attacks.

- Compatibility. The capsule or GLOM idea is re-implemented in a group ensemble block. Its usage along with CNNs is provided for image classification and segmentation, and its usage along with self-supervised learning is provided for image retrieval.

Chapter 2: Capsule networks and GLOM

2.1 The general ideas of capsule networks

A capsule network is composed of several capsule layers and convolutional layers, where the convolutional layers are used to produce the primary capsule layer. Each capsule layer consists of usually a huge number of capsule individuals. Each capsule is a group of neurons that not only represents the category of an entity (such as an object or an object part), but also represents the properties of that entity. These properties may include position, size, orientation, deformation, velocity, albedo, hue, texture, etc. One very special property is the existence probability of the entity, which is represented by the extent of activation of the capsule. The prediction output of the capsule network is the category represented by the most activated capsule of the last capsule layer. This chapter later introduces the concept of the routing procedure that passes information between two consecutive capsule layers. In addition to the general ideas of the capsule and routing procedure, a complete implementation of capsule networks should determine a certain form of the capsule, routing procedure, and network structure that are clear enough for programming. In the following, two implementations in [5] and [6] by Hinton et al. are introduced.

2.1.1 The concept of the routing procedure

The routing procedure that passes information between two consecutive capsule layers, gets all the capsules of layer L as input and outputs all the capsules of layer $L + 1$. For convenience, if there is a connection between a capsule at layer L and another capsule at

layer $L + 1$, the former capsule is called a child capsule, and the latter is called a parent capsule.

In the routing procedure, a child capsule computes prediction for each of its possible parents by multiplying its own output by transformation matrices. An intuitive understanding of the capsule's prediction is that a capsule predicts its parent's properties by its own properties through the part-whole relationship that is encoded in the transformation matrix. Each prediction has a weight, which is called the routing coefficient, and the routing coefficients of a child capsule sum to 1.

A parent capsule gets predictions from all its child capsules, and its output is coupled from these predictions. More precisely, the output of a parent capsule is the weighted sum of the predictions with the weight being routing coefficient. If the output of a parent capsule is close to all or most of these predictions, these predictions must form a tight cluster. In other words, all or most of these predictions are close to the output of the parent capsule, which is named as agreement [5, 6]. A parent capsule is supposed to become active when multiple predictions agree.

Usually, the routing procedure contains the following two steps, and the two steps are run several times to optimize the routing coefficients and the parent capsules:

- For each possible parent, a capsule computes a prediction by multiplying its own output by a transformation matrix. Then the output of each parent capsules is coupled from the predictions of all its child capsules.

- If a prediction is close to the output of a possible parent, there is top-down feedback which increases the routing coefficient for that parent and decreasing it for other parents. This further increases the similarity between the capsule's prediction and that parent's output.

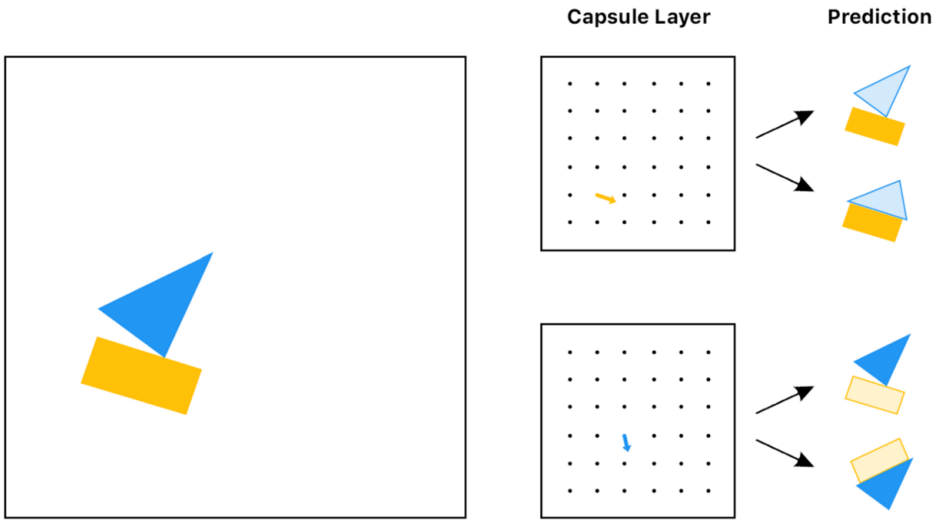


Figure 2.1. Example of the routing procedure with a simple boat. Multiple predictions are produced from each part of the whole, but only the boat prediction is agreed by both parts.

2.1.2 A routing procedure example

A routing procedure example is given in Figure 2.1. The boat in the input image consists of a triangle and a rectangle. The triangle and rectangle are rotated by 65° and 15° from the standard view, respectively. In this example, the triangle and rectangle have been detected by a certain capsule layer before the routing procedure. At the beginning of the routing procedure, the capsule representing the triangle predicts a boat and a house with equal routing coefficient, and so does the capsule representing the rectangle.

However, the two capsules only agree on the boat prediction. In other words, the predictions for the boat form a tighter cluster than the predictions for the house. The capsule representing the triangle thus favors the parent representing the boat more. This causes top-down feedback that increases the routing coefficient for the boat prediction. The same feedback happens to the capsule representing the rectangle for the same reason. After this feedback, the capsule representing the boat is more active than the capsule representing the house.

2.1.2.1 Advantages of the routing procedure

In CNNs, the ReLU function activates a neuron based on the matching score between a 3D local patch and the convolution filters. These filters are fixed after being learned in the training stage. In capsule networks, a capsule, the basic unit as a neuron in CNNs, is activated if it is supported by most of its children capsules. More precisely, a capsule is activated if the predictions coming from its child capsules match each other. This is an effective way to utilize the part-whole relationship and leads to models that generalize better.

Max pooling is used in CNNs to reduce the number of neurons representing the input image. It allows neurons in one layer to ignores all but the most active feature detector in a local pool in the layer below. The routing procedure is designed to not only produce the information for classifying the category of the input image but also many properties of each entity such as position, size, and orientation.

2.2 GLOM

Recent works [4, 36, 37] have taken a step forward in representing the part-whole hierarchy inside neural networks. The pioneering and representative method is GLOM [4] which leverages embedding islands residing in different layers to represent the part and whole. In this section, we first explain what are embedding islands and how to form embedding islands via propagation. Then we explain how to compose the contents of embeddings so that the model is able to generalize to objects placed at novel viewpoints that are never covered in the training data.

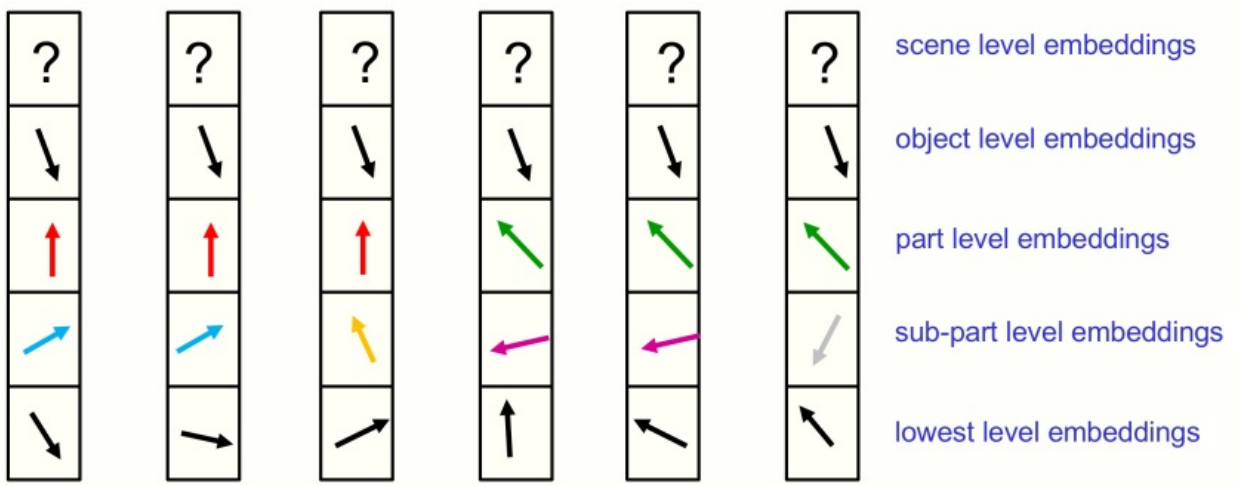


Figure 2.2. A demo illustration of embedding islands, borrowed from [4]. It contains six embeddings for each layer and 2-D vectors in the same color and direction form an island. The islands of identical vectors at the various levels shown in the figure represent a parse tree, where all of the locations shown belong to the same object and the scene level has not yet settled on a shared vector.

2.2.1 Embedding islands

2.2.1.1 What are embedding islands

Every layer of GLOM consists of multiple embeddings, which are vectors and initialized from the layer below. After the propagation, embeddings within the same layer that spatially corresponds to the same part of an object will converge to the identical vector. Each set of identical embeddings is named an embedding island because these embedding sets divide the object into its separate parts. The higher layer has few embedding islands that represent large parts of the object; the lower layer has many embedding islands that represent many small parts of the object, as shown in Figure 4.1. Denoting any two embeddings at one layer as $\mathbf{p}_i$ and $\mathbf{p}_r$, embedding islands are formed by the following,

$$\begin{cases} \mathbf{p}_i = \mathbf{p}_r, & \text{if } \mathbf{p}_i \text{ and } \mathbf{p}_r \text{ in the same part of an object} \\ \mathbf{p}_i \neq \mathbf{p}_r, & \text{otherwise.} \end{cases} \tag{2.1}$$

An embedding represents a car headlight given a car input will represent an eye when the input changes to a person. This is different from how CNNs or capsule networks work for different inputs. For example, CNNs are likely to have two neurons representing the headlight and eye respectively, which forces a hard decision that whether the car headlight and eye are really different parts.

2.2.1.2 How to form embedding islands

The embedding islands are supposed to be formed through propagation between embeddings. In each propagation round, an embedding blends information received from the layer below, the same layer, and the layer above. Specifically, in each propagation round, the i-th embedding $\mathbf{p}_i$ at a certain layer is updated with the following contributions:

- Itself at the previous propagation step $\mathbf{p}_i^{prev}$.
- The r-th embedding of the same layer $\mathbf{p}_r$ is passed to $\mathbf{p}_i$ with a confidence w_r that is positively related to its similarity with $\mathbf{p}_i$. This contribution blends an embedding with its similar neighbors.
- The j-th embedding $\mathbf{q}_j$ of the layer above is input to the top-to-down autoencoder $net^{t\to d}$, and the output is passed to $\mathbf{p}_i$. The contribution from the layer above carries the information about a larger part or the entire object and correctifies wrong embeddings at the current layer.
- The o-th embedding $\mathbf{o}_o$ of the layer below is input to the down-to-top autoencoder $net^{d\to t}$, and the output is passed to $\mathbf{p}_i$. The contribution from the layer below is also used to initialize the current-layer embeddings in the first propagation round.

In summary, $\mathbf{p}_i$ is updated with all the contributions as,

$$\mathbf{p}_i = \mathbf{p}_i^{prev} + \sum_{r \neq i} w_r \mathbf{p}_r + \sum_j net^{t \to d}(\mathbf{q}_j, \mathbf{p}_i^{prev}) + \sum_o net^{d \to t}(\mathbf{o}_o, \mathbf{p}_i^{prev}). \tag{2.2}$$

If $\mathbf{p}_i$ and $\mathbf{p}_r$ are spatially inside one object part, they will be initialized as roughly similar. The propagation blends $\mathbf{p}_i$ and $\mathbf{p}_r$ with each other and produces the same values for them.

Chapter 3: Capsule networks with non-iterative cluster routing

3.1 Introduction

Convolutional neural networks (CNNs) have been very successful in many computer vision tasks, such as image classification [9, 10], object detection [11, 12] and instance segmentation [13]. However, they sometimes fail to recognize an object captured from novel viewpoints which are not covered in the training data [38, 39]. One purpose of capsule networks is to overcome this problem [40, 8, 1]. Compared with CNNs, capsule networks have the following two major distinctions. First, the basic unit of capsule networks is a capsule composed of a group of neurons, while the basic unit of CNNs is a single neuron. A capsule thus can potentially represent multiple properties of an object, such as thickness and scale. Second, a data-dependent routing procedure is conducted between two consecutive capsule layers, while the flow of information in conventional CNNs is data-independent.

In the existing routing procedures, capsules produce predictions (termed votes) for the next-layer capsules. The input of a next-layer capsule is formulated as a weighted sum over all the votes it receives. Then its content may be computed from its input by a "squashing" function [8] or by layer normalization [41]. Iterative routing procedures alternately update the capsule's content and the weights used for formulating the capsule's input through several iterations [8, 1]. In contrast, non-iterative routing procedures compute the weights and capsule's content with a straight-through process [42, 7]. By simplifying the iterations to a single forward-pass, non-iterative routing procedures release the computational burden of iterative routing procedures.

We propose non-iterative cluster routing and apply it to capsule networks. In contrast to the existing routing procedures, in the proposed cluster routing, capsules produce vote clusters instead of individual votes for capsules of the next layer. A vote cluster comprises many votes, and each vote may be produced based on a different previous-layer capsule. A cluster's votes close to each other indicate that the same information is extracted from various previous-layer capsules. Thus the vote cluster's variance can be utilized to represent its confidence in the information it encodes. The input of a next-layer capsule is a weighted sum over the centroid of each vote cluster it receives, and the centroid that comes from a cluster with a smaller variance is assigned a larger weight. On several classification datasets, capsule networks with the proposed cluster routing achieve the best accuracy compared to the state-of-the-art capsule networks. Our capsule networks also preserve advantages of the previous types of capsule networks — producing capsules with disentangled representation [8, 7] and generalizing well to images captured from novel viewpoints [1, 42]. We also show that the proposed capsule networks preserve 2D spatial information such as the rotational orientation of an input image through a reconstruction experiment, where we first rotate the capsule channels by a transformation T, then observe if the reconstructed object is rotated by the same transformation T.

We outline the contributions of our work as the following:

- A novel non-iterative cluster routing is proposed for capsule networks. In the proposed cluster routing, capsules produce vote clusters instead of individual votes for next-layer capsules. The variance of a vote cluster is utilized to compute its confidence in the information it encodes. While computing a next-layer capsule's content, the vote cluster with smaller variance contributes more than other vote clusters.

- Compared with the state-of-the-art capsule networks, the proposed capsule networks achieve the best accuracy on the fashion-MNIST and SVHN datasets with the fewest

parameters. On the smallNORB and CIFAR-10 datasets, the proposed capsule networks achieve the best accuracy with a moderate number of parameters.

- The proposed capsule networks produce capsules with disentangled representation, generalize well to images captured from novel viewpoints, and preserve 2D spatial information of an input image in the capsule channels.

3.2 Related works

3.2.1 Capsule networks

Capsule networks were first introduced by Hinton et al. [40]. More recently, they developed capsule networks with dynamic routing [8] and EM (Expectation-Maximization) routing [1]. Capsule networks with dynamic routing yielded disentangled representation of an image; capsule networks with EM routing generalized well to images captured at novel viewpoints. However, these routing methods can be improved from the perspective of computational complexity. Li et al. [43] approximated the routing procedure with a master branch and an aide branch. Chen et al. [44] incorporated the routing procedure into the training process. Zhang et al. [45] improved the routing efficiency by using weighted kernel density estimation. Ahmed et al. [42] and Choi et al. [7] computed the coupling coefficients with a straight-through process. In addition to the works on releasing computational complexity, Ribeiro et al. [46] replaced the EM algorithm in EM-routing with Variational Bayes, which improved both the classification accuracy and novel viewpoint generalization. Tsai et al. [41] imposed layer normalization as normalization and replaced the sequential iterative routing with concurrent iterative routing. Wang et al. [47] interpreted the routing as an optimization problem that minimizes a combination of clustering-like loss and a Kullback–Leibler regularization term.

Capsule networks were combined with other techniques. Lenssen et al. [48] used group convolutions to boost the equivariance and invariance of capsule networks. Deliège et al. [49] embedded capsules in a Hit-or-Miss layer, which resulted in a hybrid data augmentation process and also detected potentially mislabeled images in the training data. Jaiswal et al. [50], Saqur et al. [51] and Upadhyay et al. [52] combined capsule networks with generative adversarial networks [53] to synthesize images.

Capsule networks were also extended to a wide range of applications. LaLonde and Bagci [54] extended capsule networks to object segmentation by introducing a deconvolutional capsule network. Durate et al. [55] developed capsule-pooling and applied capsule networks to action segmentation and classification. Zhao et al. [56] applied capsules to point clouds for 3D shape processing and understanding. Zhou et al. [57] applied capsule networks to visual question answering tasks with an attention mechanism.

3.2.2 Attention mechanism

The routing procedure is close to the attention mechanism of the Transformer [58], which produces data-dependent attention coefficients that capture the long-range interactions between inputs and outputs. Some capsule networks adopted the attention mechanism. Choi et al. [7] and Karim et al. [42] proposed attention-based routing procedures that compute the coupling coefficients between capsules without recurrence. Xinyi et al. [59] used an attention module in a capsule graph network to focus on critical parts of the graphs.

3.3 Capsule networks with dynamic routing

In contrast to a traditional neural network composed of artificial neurons, a capsule network comprises capsules. A capsule comprises a group of neurons that jointly represent an object or an object part. We present the classic dynamic routing capsule networks [8] among various types of capsule networks. In dynamic routing capsule networks, a capsule

is represented as a vector, and the capsule vector's length represents how active the capsule is. A capsule $\mathbf{u}_i \in \mathbb{R}^D$ at the lth layer is transformed to make "prediction vectors" $\hat{\mathbf{u}}_{j|i}$ for capsules of the $(l+1)$th layer, by multiplying with weight matrices $\mathbf{W}_{ij}$,

$$\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i, \tag{3.1}$$

where i and j are the indices of capsules of the lth and $(l+1)$ layer. A "prediction vector" is also named a vote for the next-layer capsules. The input $\mathbf{s}_j$ to a next-layer capsule is a weighted sum over all votes it receives, as in Eq 3.2. The capsule vector $\mathbf{v}_j$ of a next-layer capsule is "squashed" from its input such that the capsule vector's length is between zero and one, as in Eq 3.3. The dynamic routing iteratively updates the weights c_{ij}, the weighted sum $\mathbf{s}_j$ and the capsule vector $\mathbf{v}_j$ to the next-layer capsule by the following equations,

$$\mathbf{s}_j^{(t)} = \sum c_{ij}^{(t)} \hat{\mathbf{u}}_{j|i}, \tag{3.2}$$

$$\mathbf{v}_j^{(t)} = \frac{\|\mathbf{s}_j^{(t)}\|}{1 + \|\mathbf{s}_j^{(t)}\|} \cdot \frac{\mathbf{s}_j^{(t)}}{\|\mathbf{s}_j^{(t)}\|}, \tag{3.3}$$

and

$$c_{ij}^{(t)} = \frac{\exp(b_{ij} + \sum \mathbf{v}_j^{(t)} \cdot \hat{\mathbf{u}}_{j|i})}{\sum_k \exp(b_{ik} + \sum \mathbf{v}_j^{(t)} \cdot \hat{\mathbf{u}}_{k|i})}, \tag{3.4}$$

where t is the index of iteration, and b_{ik} is the log prior probability.

3.4 Capsule networks with vector capsules

In the capsule network implementation of [5], a capsule is implemented as a vector. The length of the vector represents the existence probability of the entity represented by the capsule. The orientation of the vector represents the properties of the entity such as position and size.

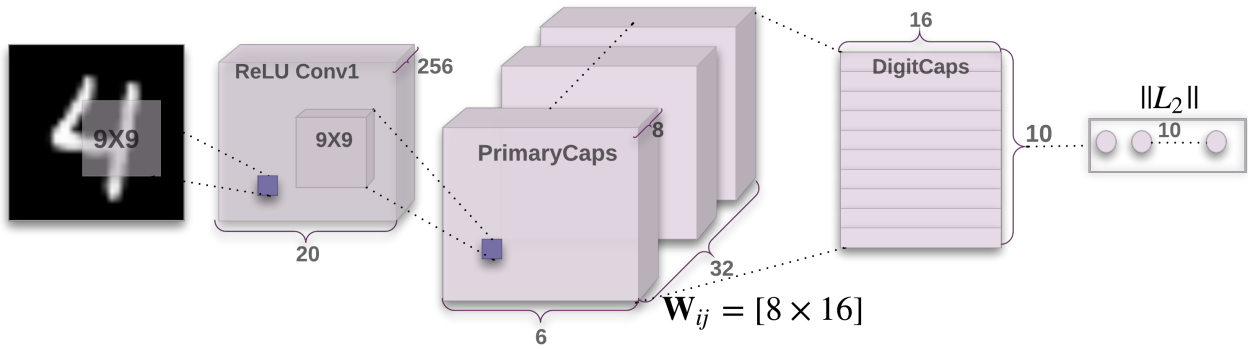


Figure 3.1. The architecture of a vector capsule network [5] used on the MNIST dataset. This network has one convolutional layer (ReLU Conv1) and two capsule layers (PrimaryCaps and DigitCaps). The routing procedure is conducted between the two capsule layers.

3.4.1 The network architecture

A capsule network with one convolutional layer and two consecutive capsule layers is shown in Figure 3.1. This network is used on the MNIST dataset. The routing procedure is conducted only between the two capsule layers.

The first layer (layer ReLU Conv1 in Figure 2) has 256 convolutional kernels and uses the ReLU activation. Each output channel is of size 20×20 since the input is an MNIST image of size 28×28, and the convolutional kernels of size 9×9 are used with a stride of 2 and no padding. This layer converts pixel intensities to the activities of local feature detectors that are then used as inputs to the first capsule layer (PrimaryCapsules).

3.4.2 The routing procedure

As introduced in above, a parent capsule gets predictions from all its child capsules, and the output of this parent capsule is the weighted sum of these predictions with the weight being routing coefficient. For all but the first layer of capsules (as the routing procedure is conducted between two consecutive capsule layers), the input to a capsule, $\mathbf{s}_j$, is a weighted sum over all "prediction vectors" $\hat{u}_{j|i}$ from the capsules in the layer below, and the weight of $\hat{u}_{j|i}$ is c_{ij}. Each "prediction vector" is produced by multiplying the output $\mathbf{u}_i$ of a capsule

in the layer below by a transformation matrix $\mathbf{W}_{ij}$,

$$\mathbf{s}_j = \sum_i c_{ij} \hat{u}_{j|i}, \qquad \hat{u}_{j|i} = \mathbf{W}_{ij} \mathbf{u}_i. \tag{3.5}$$

The output vector $\mathbf{v}_j$ of a capsule is calculated by the following non-linear "squashing" function from the input $\mathbf{s}_j$,

$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||} \tag{3.6}$$

The length of the output vector of a capsule should be between 0 and 1 since it represents the probability of existence. The non-linear "squashing" function ensures that a short input vector $\mathbf{s}_j$ gets shrunk to almost zero length and a long input vector $\mathbf{s}_j$ gets shrunk to a length slightly below 1.

The routing procedure used in vector capsule network [5] is shown in Procedure 1, where lines 4, 7 compute the routing coefficients

Algorithm 1 Routing algorithm given $\hat{\boldsymbol{u}}_{j|i}$, r, l.

for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
for r iterations **do**
  for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
  for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
  for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
end for
return $\mathbf{v}_j$

3.5 Capsule networks with matrix capsules

In contrast to the capsule network implementation of [5], the probability that the entity represented by the capsule exists is represented by a separate logistic unit in the implementation of [6]. In the implementation of [6], a capsule is implemented as a 4×4 matrix and a separate scalar unit. The separate scalar represents the probability that the entity exists.

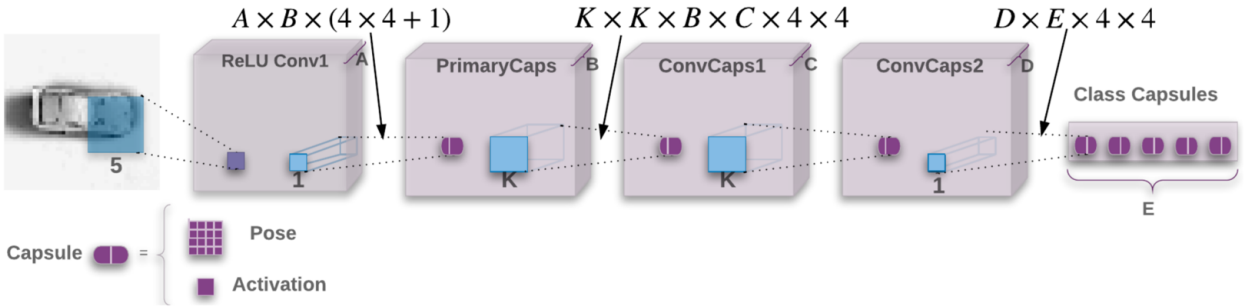


Figure 3.2. The architecture of a matrix capsule network [6] used on the SmallNORB dataset. This network has one convolutional layer and four capsule layers. The routing procedure is conducted between each adjacent pair of capsule layers.

The matrix is supposed to represent the pose (position and orientation in 3D space) of that entity and thus called pose matrix.

3.5.1 The network architecture

The structure of a capsule network [6] used on the SmallNORB dataset is shown in Figure 3.2. It has one convolutional layer and four capsule layers. The routing procedure is conducted between adjacent capsule layers.

The first layer (layer ReLU Conv1 in Figure 3) has A ($A = 32$) convolution kernels of size 5×5 with a stride of 2 and uses ReLU activation. This layer converts pixel intensities to the activities of local feature detectors that are then used as inputs to the primary capsules of the first capsule layer (PrimaryCaps).

The second layer (PrimaryCaps) takes A convolutional channels as input which is the output of the first layer. It outputs B ($B = 32$) capsule channels where each capsule contains a 4×4 matrix and a scalar activation. In this layer, $(4 \times 4 + 1) \times$ B convolutional channels are first produced from the input by convolution kernels of size 1×1 with a stride of 1. Then the $(4 \times 4 + 1) \times B$ convolutional channels are rearranged to B capsule channels.

The first capsule layer is followed by two $K \times K$ (K=3) convolutional capsule layers (ConvCaps1 and ConvCaps2), each with 32 capsule types ($C = D = 32$) with strides of 2 and 1, respectively.

The final capsule layer (Class Capsules) has one capsule per output class. There are E (E=5) classes in total. The category represented by the most activated capsule of the final capsule layer is used as the prediction output of the capsule network.

3.5.2 The routing procedure

The routing procedure in the implementation of [6] is named EM routing as it is based on the Expectation-Maximization (EM) algorithm. In the EM routing procedure, each parent capsule corresponds to a Gaussian, and the pose of each active child capsule (converted to a vector) corresponds to a datapoint (or a fraction of a datapoint if the capsule is partially active). Recall the definition of the parent and child capsule: for convenience,if there is a connection between a capsule at layer L and another capsule at layer $L + 1$, the former capsule is called a child capsule, and the latter is called a parent capsule.

Each capsule i in layer L makes a "prediction pose" for the pose of capsule j in layer $L + 1$. A "prediction pose", also called a vote, is produced by multiplying the pose matrix $\mathbf{M}_i$ of capsule i by a 4×4 transformation matrix $\mathbf{W}_{ij}$,

$$\mathbf{V}_{ij} = \mathbf{M}_i \mathbf{W}_{ij}, \tag{3.7}$$

where $\mathbf{W}_{ij}$ is discriminatively learned and supposed to encode the part-whole relationship between $\mathbf{M}_i$ and $\mathbf{V}_{ij}$. The activations a_i and votes $\mathbf{V}_{ij}$ for all capsules $i \in \Omega_L$ and $j \in \Omega_{L+1}$ are input to the EM routing procedure to compute the activations a_j and poses $\mathbf{M}_j$.

An interesting property of the above equation is that if the following equation satisfies, the part-whole relationship encoded in $\mathbf{W}_{ij}$ applies to all viewpoints,

$$\mathbf{TV}_{ij} = \mathbf{TM}_i \mathbf{W}_{ij}, \tag{3.8}$$

where $\mathbf{T}$ is a 3D viewpoint transformation matrix applied to the input object, $\mathbf{TV}_{ij}$ and $\mathbf{TM}_i$ are representing the same object parts as $\mathbf{V}_{ij}$ and $\mathbf{M}_i$ but in different viewpoint.

3.5.3 Activation of a capsule

The choice of whether activating a capsule j is based on the cost of explaining datapoint i by using capsule j,

$$cost_{ij}^h = -\ln(P_{i|j}^h), \tag{3.9}$$

$$p_{i|j}^h = \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} \exp\left(-\frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}\right), \tag{3.10}$$

where v_{ij}^h is the h-th dimension of the vectorized vote $\mathbf{V}_{ij}$, μ_j is the mean of the fitted Gaussian, and σ_j is the variance of the fitted Gaussian under the assumption that μ_j has an axis-aligned covariance matrix. This assumption makes it possible to calculate the cost by summing over all dimensions of the cost of explaining each dimension, h, of the vote $\mathbf{V}_{ij}$.

The activation of the capsule j, a_j, is computed with the logistic function of the cost, $cost_j$. The h-th dimension of $cost_j$ is computed as,

$$\begin{aligned} cost_j^h &= \sum_i r_{ij} cost_{ij}^h \\ &= \sum_i -r_{ij} \ln(p_{i|j}^h) \\ &= \sum_i r_{ij} \big(\frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2} + \ln(\sigma_j^h) + \frac{\ln(2\pi)}{2} \big) \\ &= \frac{\sum_i r_{ij} (\sigma_j^h)^2}{2(\sigma_j^h)^2} + (\ln(\sigma_j^h) + \frac{\ln(2\pi)}{2}) \sum_i r_{ij} \\ &= \big(\ln(\sigma_j^h) + k \big) \sum_i r_{ij}, \end{aligned} \tag{3.11}$$

where k is a constant, c_{ij} is the routing coefficient from capsule i to capsule j, and $\sum_i r_{ij}$ is the amount of data assigned to capsule j. a_j is computed as,

$$a_j = sigmoid(\lambda(\beta_a - \sum_h cost_j^h)), \tag{3.12}$$

where β_a is discriminatively learned, λ is an inverse temperature parameter as a hyperparameter. Capsule j thereby is activated only when σ_j is small enough and $\sum_i r_{ij}$ is big enough. This corresponds to a tight cluster of votes with sufficient capsules from the layer below with high activations.

3.6 The proposed cluster routing

In contrast to the dynamic routing, the proposed cluster routing utilizes vote clusters instead of individual votes. A capsule $\mathbf{u}_i \in \mathbb{R}^D$ at the lth layer is multiplied with each weight matrix of a weight cluster $\mathbf{W}_i^{\{1,\cdots,K\}}$, resulting in a vote cluster $\hat{\mathbf{u}}_i^{\{1,\cdots,K\}}$ for a capsule at the $(l+1)$th layer. To reduce clutter in the notation, from now on we omit the index

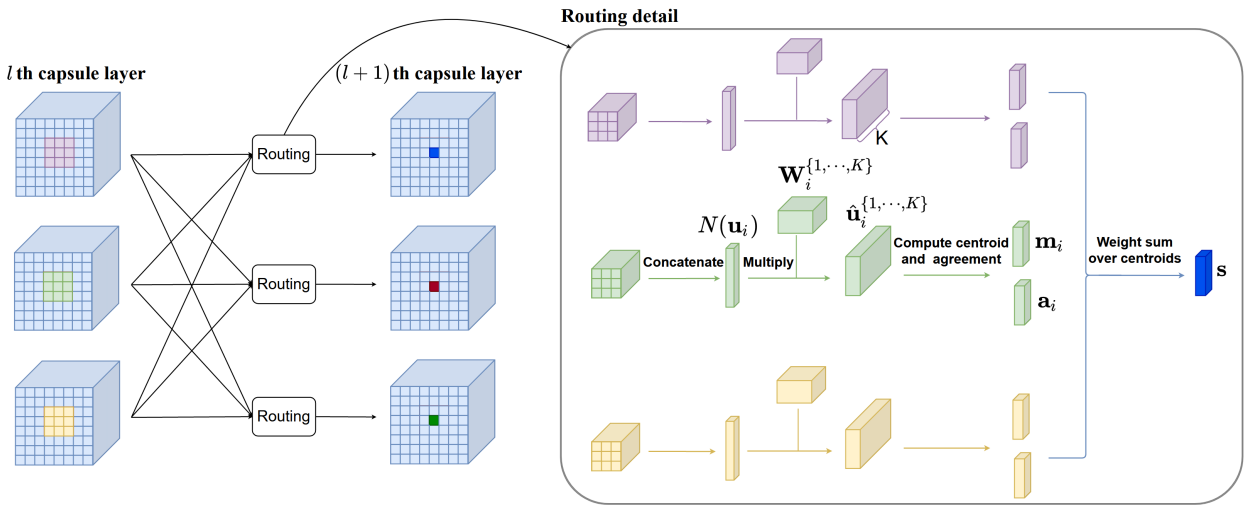


Figure 3.3. Illustration of the proposed cluster routing. A next-layer capsule of the $(l+1)$th layer gets input as a weighted sum over the centroids it receives, and larger weights are assigned to the centroid with larger corresponding agreement vector. The content of next-layer capsules are computed from their inputs by layer normalization, which is not shown in the figure for the purpose of clarity.

j for the next-layer capsules without introducing confusion. Then, for a cluster of weights $\mathbf{W}_i^{\{1,\cdots,K\}}$, for $k \in \{1, \cdots, K\}$, Eq 3.1 becomes

$$\hat{\mathbf{u}}_i^k = \mathbf{W}_i^k \mathbf{u}_i. \tag{3.13}$$

Each weight matrix $\mathbf{W}_i^k$ in a weight cluster may attend to a specific and distinct location of the capsule vector $\mathbf{u}_i$. It is supposed that after the training stage, if all vector locations represent the same object (or object part), each weight matrix will produce the same vote; if one vector location does not represent the same object as other vector locations, one or more votes will be different from others. Thus the agreement among these votes indicates if the capsule vector correctly represents a certain object.

Furthermore, we can replace $\mathbf{u}_i$ in Eq 3.13 by its neighborhood to increase the receptive field of each vote. In practice, we fix with the 3×3 neighborhood throughout this work and so we replace $\mathbf{u}_i$ by the concatenation $N(\mathbf{u}_i) \in \mathbb{R}^{9D}$ of its neighborhood. Then the k-th vote

in the cluster is produced as follows,

$$\hat{\mathbf{u}}_i^k = \mathbf{W}_i^k N(\mathbf{u}_i). \tag{3.14}$$

A vote cluster, $\hat{\mathbf{u}}_i^{\{1,\cdots,K\}}$, sends its centroid $\mathbf{m}_i = \frac{1}{K}\sum_{k=1}^{K} \hat{\mathbf{u}}_i^k$ and an agreement vector $\mathbf{a}_i$ to the next-layer capsule. The agreement vector $\mathbf{a}_i$ is computed by applying the negative log to the votes' standard deviation as follows,

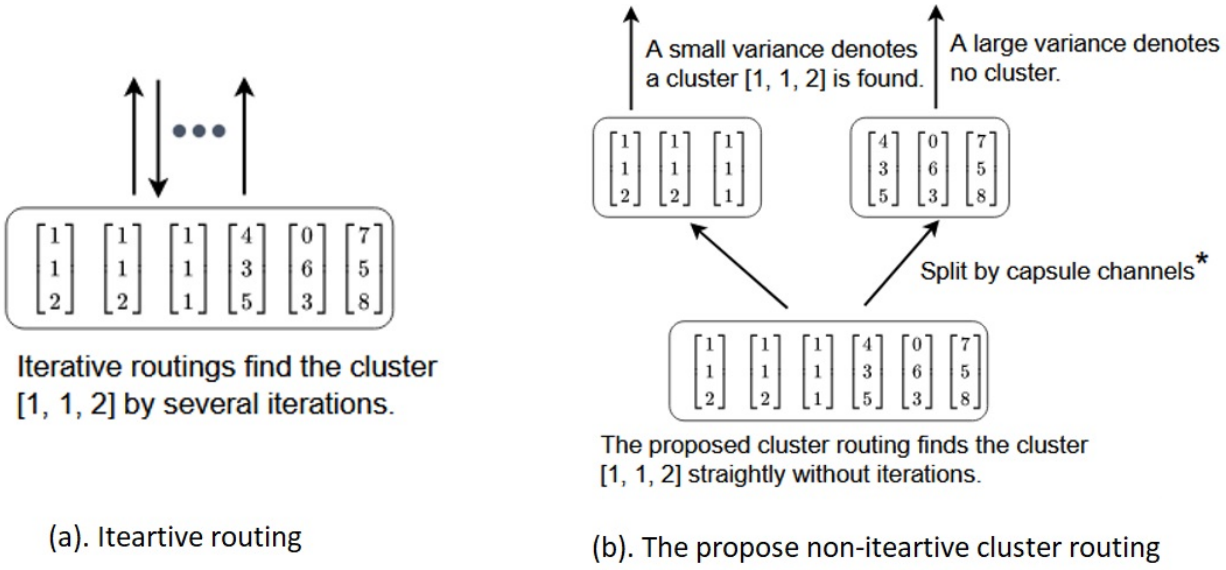
$$\mathbf{a}_i = -\log\left(\sqrt{\frac{1}{K}\sum_{k=1}^{K}(\hat{\mathbf{u}}_i^k - \mathbf{m}_i^k)\circ(\hat{\mathbf{u}}_i^k - \mathbf{m}_i^k)}\right), \tag{3.15}$$

where $\circ$ is the Hadamard (element-wise) product. Each of the C capsule channels in the lth layer produces a vote cluster for the next-layer capsule. Centroids of these vote clusters are weighted summed as follows,

$$\mathbf{s} = \sum_{i=1}^{C} \mathbf{c}_i \circ \mathbf{m}_i, \tag{3.16}$$

where $\mathbf{c}_i = \exp(\mathbf{a}_i) \oslash \sum_{i=1}^{C} \exp(\mathbf{a}_i)$ and $\oslash$ is the Hadamard division. We apply layer normalization [60] on the weighted sum $\mathbf{s}$, resulting in a capsule vector of the next layer as in [41].

Notice that the matrix product in Eq 3.14 can be implemented by D "conv" filters in popular deep learning libraries such as Tensorflow [61] and PyTorch [62]. This is also used in Choi et al.'s work [7], where the authors name it as convolutional transform. This decreases the difficulty to program a capsule network with the proposed routing, and also accelerates the running speed because the "conv" operation in Tensorflow and Pytorch is highly optimized.



(a). Iteartive routing

(b). The propose non-iteartive cluster routing

Figure 3.4. Comparison between iterative routing algorithms and the proposed non-iterative cluster routing algorithm.

3.6.1 Comparisons with related works

Although our weight matrix is implemented using convolutional filters, the proposed capsule networks achieve non-linearity by the proposed cluster routing instead of the ReLU activation as in CNNs. Table 3.1 lists the differences among different types of capsule networks.

The proposed cluster routing may remind the readers of the group normalization [63] which also utilizes the mean and standard deviation of a group. Group normalization divides the output channels of a convolutional layer into several groups, and normalizes each group with the group's mean and standard deviation, which is similar to other normalization techniques such as batch normalization [64], instance normalization [65] and layer normalization [60]. However, in contrast to the proposed cluster routing, the group normalization does not qualify as a routing algorithm, because it has no process similar to the following routing process: i) compute the data-dependent routing weights based on the agreement between

Table 3.1. Comparison on different types of capsule networks.

	Dynamic Routing [8]	EM routing [1]	Inverted dot-product attention routing [41]	The proposed cluster routing
Routing	sequential iterative	sequential iterative	concurrent iterative	non-iterative
Poses	vector	matrix	matrix	vector
Activations	n/a (norm of poses)	determined by EM	n/a	n/a
Non-linearity	Squash function	n/a	n/a	n/a
Normalization	n/a	n/a	Layer Normalization	Layer Normalization
Loss Function	Margin loss	Spread loss	Cross Entropy	Cross Entropy

votes; ii) compute the input of next-layer capsules as a weighted sum over the votes, where the routing weight are data-dependent.

3.7 Experiments

We evaluate the proposed capsule networks on the following tasks: classification, disentangled representation, generalization to images captured at novel viewpoints, and reconstruction from affine-transformed channels. We also visualize the routing weights $\boldsymbol{c}_i$, verifying that they are data-dependent as they should be.

3.7.1 Classification

Network architectures The proposed capsule networks' capacity is related to two hyperparameters, the number of a capsule vector's dimensions, D, and the number of a weight cluster's weight matrices, K. We design four variants of the proposed capsule networks by varying D and K while fixing the number of layers as five and the number of channels at each layer as four. The four variants are named *M-variant1-4* as in Table 3.2. During the experiments, we find that the proposed capsule networks also work well even if we use only one capsule channel at each layer. When using a single channel, we apply N weight clusters

Table 3.2. Test error rate comparisons with capsule networks literature and the baseline CNN. (·) denotes ensemble size.

Method	**smallNORB**		**Fashion-MNIST**		**SVHN**		**CIFAR-10**	
	Error (%)	Param	Error (%)	Param	Error (%)	Param	Error (%)	Param
Inverted dot-product attention routing [41]	-	-	-	-	-	-	14.83	560K
Attention Routing [7]	-	-	-	-	-	-	11.39	9.6M
STAR-CAPS [42]	-	-	-	-	-	-	8.77	≃318K
HitNet [66]	-	-	7.7	≃8.2M	5.5	≃8.2M	26.7	≃8.2M
DCNet [67]	5.57	11.8M	5.36	11.8M	4.42	11.8M	17.37	11.8M
MS-Caps [68]	-	-	7.3	10.8M	-	-	24.3	11.2M
Dynamic [8]	2.7	8.2M	-	-	4.3	≃1.8M	10.6	8.2M (7)
Nair *et al.* [69]	-	-	10.2	8.2M	8.94	8.2M	32.47	8.2M
FRMS [45]	2.6	1.2M	6.0	1.2M	-	-	15.6	1.2M
MaxMin [70]	-	-	7.93	≃8.2M	-	-	24.08	≃8.2M
KernelCaps [71]	-	-	-	-	8.6	≃8.2M	22.3	≃8.2M
FREM [45]	2.2	1.2M	6.2	1.2M	-	-	14.3	1.2M
EM-Routing [1]	1.8	310K	-	-	-	-	11.9	≃460K
VB-Routing [46]	1.6	169K	5.2	172K	3.9	323K	11.2	≃323k
Baseline CNN	3.76	3.30M	5.21	3.38M	3.30	3.38M	7.90	3.38M
S-variant1 (N4K4D13)	2.80±0.20	150K	5.19±0.15	152K	3.89±0.10	156K	13.33±0.78	156K
S-variant2 (N4K4D16)	2.58±0.32	217K	5.07±0.13	215K	3.77±0.11	219K	11.58±0.36	219K
S-variant3 (N8K8D16)	1.93±0.22	672K	4.79±0.13	686K	3.47±0.07	686K	8.58±0.15	686K
S-variant4 (N8K8D32)	**1.57**±0.13	2.53M	4.68±0.01	2.51M	**3.37**±0.03	2.55M	**7.37**±0.06	2.55M
M-variant1 (C4K5D6)	2.98±0.24	150K	5.17±0.07	146K	3.94±0.07	154K	12.16±0.30	154K
M-variant2 (C4K5D8)	3.09±0.19	246K	5.02±0.04	240K	3.63±0.11	252K	11.11±0.09	252K
M-variant3 (C4K8D16)	1.92±0.12	1.32M	4.84±0.07	1.30M	3.56±0.07	1.34M	8.55±0.12	1.34M
M-variant4 (C4K8D24)	1.95±0.12	2.87M	**4.64**±0.03	2.84M	3.48±0.14	2.89M	7.89±0.11	2.89M

on capsules of this single channel which produces N vote clusters for a next-layer capsule. We also design four variants with a single channel at each layer, named *S-variant1-4* as in Table 3.2.

Every *M-variant* and *S-variant* has 5 capsule layers, with a stride of 2 at the second and fourth layers. Each variant is trained for 300 epochs using cross-entropy loss with stochastic gradient descent. The initial learning rate is 0.1 with step decay at every 100 epochs, and the decay rate is 0.1. A batch size of 64 is used.

Datasets and data augmentation For each dataset, the hyperparameters for data augmentation are tuned by a validation set containing one-fifth of the training images. The models are then retrained with the full training set before testing. During the training stage, we add brightness and contrast jitter to an image to perturb its brightness and contrast. For a pixel at position x, its value $f(x)$ can be perturbed by $g(x) = \alpha f(x) + \beta$, where α and β control contrast and brightness, respectively. In this context, adding random brightness and contrast with a factor of 0.2 to an image means that α is in the range [0.8, 1.2] and β is in the range $[-0.2\frac{1}{N_x}\sum_x f(x),\ 0.2\frac{1}{N_x}\sum_x f(x)]$, where N_x is the total number of pixels and $\frac{1}{N_x}\sum_x f(x)$ is the mean value of all pixels.

Datasets and data augmentation For each dataset, the hyperparameters for data augmentation are tuned by a validation set containing one-fifth of the training images. The models are then retrained with the full training set before testing. During the training stage, we add brightness and contrast jitter to an image to perturb its brightness and contrast. For a pixel at position x, its value $f(x)$ can be perturbed by $g(x) = \alpha f(x) + \beta$, where α and β control contrast and brightness, respectively. In this context, adding random brightness and contrast with a factor of 0.2 to an image means that α is in the range [0.8, 1.2] and β is in the range $[-0.2\frac{1}{N_x}\sum_x f(x),\ 0.2\frac{1}{N_x}\sum_x f(x)]$, where N_x is the total number of pixels and $\frac{1}{N_x}\sum_x f(x)$ is the mean value of all pixels.

smallNORB [72] comprises 5 classes of 96×96 stereo images. The training and test sets both have 24,300 images. Following the steps in [1], we downsample each image to 48×48 pixels and normalize it to zero mean and unit variance. During training, we add random brightness and contrast with a factor of 0.2, pad to 56×56, randomly shift with a factor of 0.2, and randomly cropped to 32×32. At test time, we take the center 32×32 crop.

Fashion-MNIST [73] comprises 10 classes of 28×28 clothing items. The training and test sets have 60,000 and 10,000 images, respectively. During training, we add random brightness

and contrast with a factor of 0.2, pad to 36 × 36, take random 32 × 32 crop, and apply random horizontal flips with probability 0.5. At test time, we pad the images to 32 × 32.

SVHN [74] comprises 10 digit classes of 32 × 32 real-world house numbers. We trained on the core training set only, consisting of 73,257 images, and tested on the 26,032 images of the test set. During training, we add random brightness and contrast with a factor of 0.2, pad to 40 × 40, and take random 32 × 32 crop.

CIFAR-10 [75] comprises 10 classes of 32 × 32 real-world images. The training and test sets have 50,000 and 10,000 images, respectively. During training, we add random brightness and contrast with a factor of 0.2, pad to 40 × 40, take random 32 × 32 crop, and apply random horizontal flips with probability 0.5.

ImageNet [76] comprises 1000 classes of real-world images. The training and validation sets have 1,281,167 and 100,000 images, respectively. During training, we resize each image to 256 × 256 pixels, take random 224 × 224 crop and apply random horizontal flips with probability 0.5. During validation, we take the center 224 × 224 crop. Following the previous Ahmed et al.'s work [42], the test set is not used.

Accuracy comparisons with the state-of-the-arts The comparisons between the state-of-the-art capsule networks and the proposed *M-variants* and *S-variants* are listed in Table 3.2. On the Fashion-MNIST and SVHN datasets, the proposed capsule networks achieve better accuracy than other types of capsule networks with fewer parameters: i) on Fashion-MNIST, *M-variant1* achieves an error rate of 5.17% with 146K parameters, and *S-variant1* achieves an error rate of 5.19% with 152K parameters; ii) on SVHN, *M-variant2* achieves an error rate of 3.63% with 252K parameters, and *S-variant2* achieves an error rate of 3.77% with 219K parameters. For the smallNORB dataset, *S-variant4* achieves the best error rate of 1.57% with a moderate size 2.53M. For the CIFAR-10 dataset, *S-variant4* achieves the best error rate of 7.37% with a moderate size 2.55M; *M-variant4* achieves an error rate of 7.89% with a moderate size 2.89M.

Classification accuracy on ImageNet For the ImageNet dataset, we design a capsule network variant based on the *M-variant4*. Similar to the STAR-CAPS variant designed for ImageNet in [42], this variant starts with a 7×7 convolutional layer that outputs 64 channels, followed by a single bottleneck residual block with 256 output channels. Then the *M-variant4* is added after the residual block. The Top-1 validation accuracy on ImageNet is 63.87% and the Top-5 accuracy is 88.98%, which outperforms the accuracy of 60.07% and 85.66% produced by the STAR-CAPS network [42].

Accuracy comparisons with the baseline CNN We compare the variants *M-variant4* and *S-variant4* with a baseline CNN. The baseline CNN is designed as the following: 5 ReLU convolutional layers, layer normalization after the ReLU activation, 256 filters at each layer and 3.38M parameters in total. As shown in Table 3.2, the baseline CNN has more parameters, and achieves an higher error rate compared to either of the *M-variant4* and *S-variant4* networks.

Experiments on hyperparameters For the *M-variants*, we analyze the impact of the hyperparameters K and D, while fixing the number of channels at each layer as four. As shown in Table 3.3, there is a clear trend that both larger D and larger K lead to higher accuracy on the CIFAR-10 dataset.

Ablation study In the ablation experiment, we fix the routing weight c_i to a constant value $\frac{1}{C}$, which means the weight becomes data-independent. As shown in Table 3.4, after removing the data-dependence, the proposed capsule networks' performance drop significantly, which demonstrates the data-dependence is crucial.

3.7.2 Generalization to novel viewpoints

We validate the proposed capsule networks' generalization ability to images captured at novel viewpoints using the smallNORB dataset. Following the experiments in [1], we train the proposed capsule networks on one-third of the training data containing azimuths

Table 3.3. Analysis of the hyperparameters K and D on the *M-variants* using the CIFAR-10 dataset. Test error rate and the number of parameters are listed for each setting.

	D=6	D=8	D=16	D=24
K=5	12.16% 154K	11.11% 252K	9.06% 872K	8.06% 1.86M
K=8	11.29% 226K	10.24% 375K	8.55% 1.34M	7.89% 2.89M

Table 3.4. Ablation study on the proposed capsule network *M-variants*, where each variant's name is abbreviated, e.g., *M-v1*. We study the network's classification accuracy when the routing weights c_i are data-dependent or data-independent. The CIFAR-10 dataset is used.

Data-dependent routing	*M-v1*	*M-v2*	*M-v3*	*M-v4*
Yes	12.16	11.11	8.55	7.89
No	35.48	33.97	33.48	32.96

of (300, 320, 340, 0, 20, 40) and test on the test data containing azimuths from 60 to 280; for elevation viewpoints, we train on the 3 smaller and test on the 6 larger elevations. The validation set consists of images captured at the same viewpoints as in training. For the networks to be compared, we measure their classification accuracy on images captured at novel viewpoints (test set) after matching their classification accuracy on familiar viewpoints (validation set). The following networks are compared in Table 3.5: the baseline CNN model as in [1], EM-routing capsule networks [1], STAR-CAPS networks [42], and capsule networks with the proposed cluster routing. The proposed capsule networks achieve the best accuracy of 86.9% on novel azimuth viewpoints. On novel elevation viewpoints, the proposed capsule networks achieve an accuracy of 86.6%, which is slightly lower than the EM-routing capsule networks while outperforming the baseline CNN.

Table 3.5. A comparison of the smallNORB test error rate on images captured at novel viewpoints when all models are matched on error rate for familiar viewpoints. We use the same baseline CNN as in Hinton et al.'s work [1].

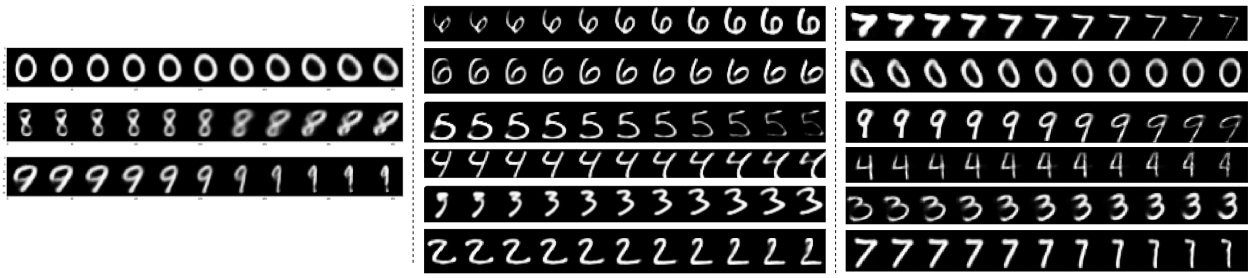
	Azimuth (%)				**Elevation** (%)			
Model	CNN	EM	STAR-CAPS	Ours	CNN	EM	STAR-CAPS	Ours
#Params	4.2M	316K	318K	246K	4.2M	316K	-	246K
Familiar	96.3	96.3	96.3	96.3	95.7	95.7	-	95.7
Novel	80.0	86.5	86.3	86.9	82.2	87.7	-	86.6

3.7.3 Disentangled representation

Capsule networks with dynamic routing [8] and capsule networks with attention routing [7] produce capsules with disentangled representation — each dimension of a last-layer capsule represents a digit's property, such as thickness, skew, and width. The proposed capsule networks also produce capsules with disentangled representation. Based on *M-variant2*, we change the number of last-layer capsules to 10, such that each last-layer capsule represents a class of the MNIST dataset. For an input image, we mask out all capsule vectors of the last layer except the one representing the input image's class. This capsule vector is input to a decoder that reconstructs the input image. The decoder has the same architecture as in [8], which consists of 3 fully connected layers with 512, 1024, and 784 (784 is the total number of pixels of an MNIST image) neurons, respectively. As shown in Figure 3.5, the proposed capsule networks produce capsules with disentangled representation.

3.7.4 Reconstruction from affine-transformed channels

We design a 3-layer capsule network for this task, where the second layer has a stride of 2. The output channels of the last layer are input into a reconstruction network. The reconstruction network consists of one upsample layer and two convolutional layers with ReLU and Sigmoid activation. The Sigmoid convolutional layer outputs the reconstructed image in the range [0, 1]. We transform the last layer's output channels by a transformation T and



(a) Attention routing (b) Dynamic routing (c) Proposed cluster routing

Figure 3.5. Dimension perturbations on capsules produced by capsule networks with attention routing [7] (left), dynamic routing [8] (middle), and the proposed clustering routing (right), respectively. Each row shows the reconstructed images when one dimension of the capsule representing the input digit is tweaked by intervals of 0.05 in the range [-0.25, 0.25]. All three capsule networks produce capsules with disentangled representation – each dimension of a certain capsule represents a digit's property, such as thickness, skew, and width.

observe the image reconstructed from the transformed channels. Ideally, the reconstructed image shall look like the input image transformed by the transformation T. The layer normalization is not used due to the simplicity of the MNIST dataset. The network is trained to perform both classification and reconstruction tasks.

The baseline CNN for this task has a similar architecture to the 3-layer capsule network. Table 3.6 lists the architecture of the two networks in detail. A capsule network with dynamic routing [8] (8.2M parameters) is also compared. For the dynamic routing network, we: i) transform the output channels of the second last capsule layer because each channel of the last layer contains only one capsule; ii) reconstruct the input image using three fully connected layers as in [8].

As shown in Figure 3.6, the dynamic routing capsule network seems to be always trying to reconstruct the original image. It produces low-quality reconstructions for large rotations (rotation with a degree from 90° to 270°), translations, and flips. The baseline CNN produces fine reconstructions for vertical flips, translations, scaling with a factor larger than 1, and gentle rotations 0°, 45°, 315° (-45°), but fails on large rotations and horizontal flip. The proposed capsule network produces fine reconstructions for almost all transformations except

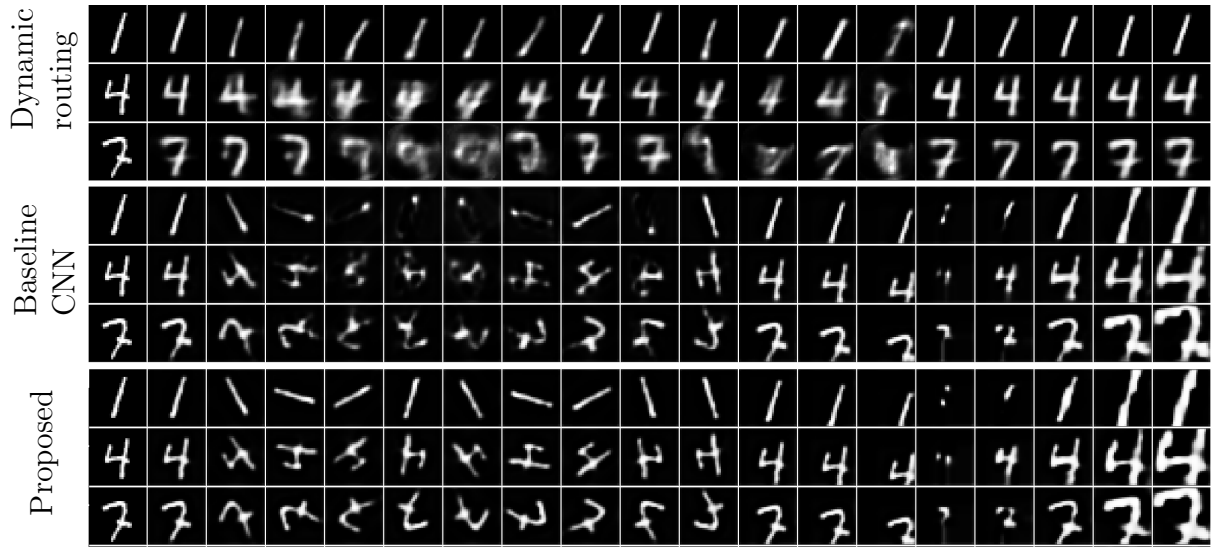


Figure 3.6. Reconstructed images from capsule channels output by the dynamic routing capsule network [8] and the proposed capsule network, and reconstructed images from convolutional channels output by the baseline CNN. The first column shows groundtruth. The other columns show reconstructions from capsule channels (or convolutional channels) applied with the following affine transformations: 2-9 col: rotation with 0, 45, 90, ..., 315 degrees; 10-11 col: horizontal and vertical flip; 12-14 col: shifting 1, 2, 4 pixels; 15-19 col: scaling by a factor of 0.5, 0.75, 1.2, 1.5, 2.

scaling with a factor less than 1. In short, the proposed capsule networks succeed in more transformation cases than the baseline CNN and the dynamic routing capsule network.

3.7.5 Analysis of routing weights

A data-dependent routing means that the routing weights $\mathbf{c_i}$ are dependent on the input image's visual content, unlike the weights matrices that are the same for any input. However, the reader may come up with a degenerate case where the proposed cluster routing may produce data-independent routing weights: suppose the weight matrices of a weight cluster are identical or very close to each other, the votes produced by this weight cluster will be identical or very close; then the routing weight for this vote cluster's centroid will always be almost 1, regardless of the input image's visual content. To examine if this degeneration happens, we visualize the routing weights.

Table 3.6. Parameters of a proposed capsule network and a baseline CNN used for reconstructing images from affine-transformed channels. Each filter of the baseline CNN is of size 3×3.

	Baseline CNN	Capsule network (C4K4D32)
First layer	960 (96 filters)	5,120
Second layer	83,040 (96 filters)	18,944
Third layer	13,840 (16 filters)	18,944
Linear classifier	31,370	31,370
First recons layer	4,640 (32 filters)	4,640
Second recons layer	289 (1 filter)	289
Total	134,139	79,307

We decide the stride and padding at each capsule layer such that each channel of the last layer contains only one capsule, then visualize routing weights for the last-layer capsules. Figure 3.7 shows the routing weights for the four vote clusters that a last-layer capsule receives. It can be seen from Figure 3.7 that the proposed capsule networks produce routing weights of the same distribution for images from the same class, but routing weights of different distributions for images from different classes. This demonstrates that the degenerate case does not happen — the proposed capsule networks use data-dependent routing weights.

3.7.6 Application in medical image segmentation

The dataset adopted here is the lung nodule analysis 2016 dataset (LUNA16). It consists of 885 labelled 3D CT scans of lung cancer screening patients collected from 7 academic centers and 8 medical imaging companies. Each 3D scan has 100$\sim$300 2D slices of size 512×512.

The proposed cluster routing is used to replace the iterative routing algorithm in the SegCaps model [77]. The group ensemble block proposed later in this dissertation is also adopted along with cluster routing. All the models are trained with the same setting on 2

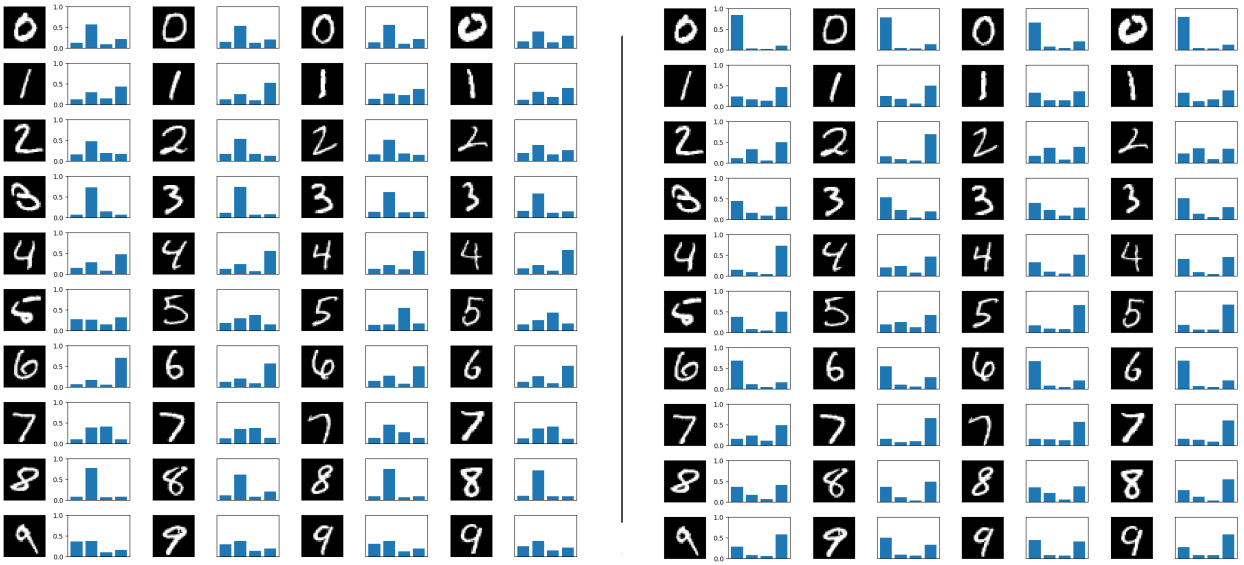


Figure 3.7. Visualization of routing weights used for last-layer capsules. Four bars show routing weights for the four vote clusters that a last-layer capsule receives. The left figure shows routing weights for the 8th dimension of the first channel's capsule; the right figure shows routing weights for the 2nd dimension of the second channel's capsule. Each channel of the last layer is designed to contain only a single capsule.

Table 3.7. Comparison of capsule network with iterative routing, cluster routing, and group ensemble block on the LUNA16 lung segmentation dataset.

Method	Dice coefficient mean/median	Precision mean/median
SegCaps with iterative routing [77]	0.9924/0.9958	0.9865/0.9916
SegCaps with cluster routing	0.9944/0.9979	0.9905/0.9958
SegCaps with cluster routing & group ensemble block	0.9992/0.9996	0.9942/0.9994

A5000 16G-memory graphic cards: batch size of 6, Adam optimizer, constant learning rate of 1e-5. As in Table 3.7, with using the proposed cluster routing, the dice score is increased from 0.9924 to 0.9944 and from 0.9958 to 0.9979 in terms of mean and median; the precision is increased from 0.9865 to 0.9905 and from 0.9916 to 0.9958 for mean and median metrics in terms of mean and median. With plugging the proposed ensemble block, the dice coefficient and precision are further improved.

3.8 Summary

We propose a non-iterative cluster routing algorithm for capsule networks. The proposed cluster routing adopts vote clusters instead of individual votes, and the variance of a vote cluster is used to compute its confidence in the information it encodes. A capsule vector is computed from the vote clusters it receives, where the vote cluster with larger confidence contributes more than other vote clusters. The experiments show that capsule networks with the proposed cluster routing achieve competitive performance on tasks including classification, disentangled representation, generalization to images obtained from novel viewpoints, and reconstructing images from affine-transformed channels. In the future, it will be interesting to explore whether some of the vote clusters can be pruned without affecting the performance.

Chapter 4: Twin-Islands: an adversarial-robust and interpretable architecture

Borrowing part-whole relationships into neural networks provides a chance to access the internal architecture of models. The recently proposed GLOM is an imaginary system that parses the part-whole hierarchy of an object into the model's internal embeddings. However, the parsing process of this hierarchy remains obscure by the black-box autoencoders. In this paper, we propose a new adversarial-robust and interpretable architecture for point cloud processing, named Twin-Islands. The proposed Twin-Islands not only parses the object into embedding islands, but also makes the parsing process transparent via encoding the real-world part-whole relationships with some transformation weight islands. The transparent parsing process makes the model intrinsically interpretable. Furthermore, the object is classified through plurality voting where the votes are produced from the embedding islands and transformation weight islands. The plurality voting rules out a number of incorrect votes, in contrast to the traditional models that accumulate the perturbations in every object part. The visualization experiment shows the correspondence between the real-world part-whole relationships and the transformation weight islands. One of its applications is also provided which prunes the transformation weights by the importance of corresponding part-whole relationships. Adversarial perturbation and deletion experiments show its superiority over CNN-based models.

4.1 Introduction

One goal of artificial intelligence is to create human-like agents. Current deep models have achieved human-like accuracy in almost every computer vision task, even defeating

humans sometimes. However, the process of how deep models produce their outputs are still very different from humans. If we want to have deep models entirely working like humans, they need to have human-like processing on their input too, in addition to the human-like accuracy.

Psychological researches have revealed that people parse an object into its part-whole hierarchy through the viewpoint-invariant part-whole relationships [23, 22]. The recently proposed GLOM [4] has attracted great attention since it demonstrates the feasibility to parse the part-whole hierarchy of an object into the model's internal embeddings. Specifically, it uses embedding islands at the lower and higher layer to represent the part and whole respectively, where every embedding island contains many identical embeddings and spatially matches an object part. The embedding islands in consecutive layers are connected by networks. Although the embedding islands clearly represent the parts and whole, the relationships from parts to whole remain obscure in the black-box networks.

In this paper, we propose a new architecture Twin-Islands, where we introduce a novel component called transformation weight islands to represent the relationship between parts and whole, in addition to the embedding islands in GLOM. Specifically, the transformation weight islands have the same shapes as the embedding islands of the lower layer of two consecutive layers, and each transformation weight island is performed to the same shape embedding island, producing votes for the entire object. The relationships from parts to the whole are transparent and stored in every transformation weight island.

One definition of interpretability is "the degree to which a human can understand the cause of a decision [27]." The proposed Twin-Islands is an intrinsically interpretable architecture whose model internals (e.g., learned weights) have a clear meaning. In contrast, most of the existing interpretability works for point cloud processing are conducted on non-intrinsically interpretable deep models. These methods identify the relationship between the input and output post hoc, such as finding which points contribute most to the classification

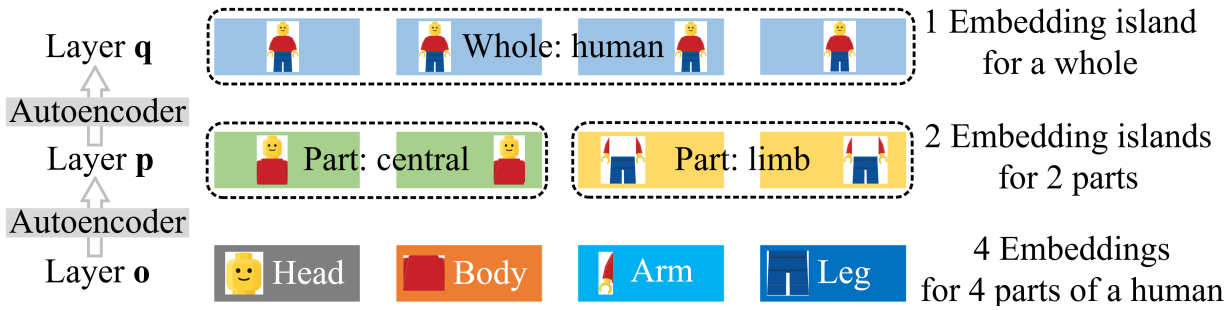


Figure 4.1. The embedding islands in GLOM that represent sub-parts, parts, and whole object of a LEGO toy.

decision [78] or approximating the classification decisions with the simple linear classifier for each point cloud [79].

Furthermore, the proposed Twin-Islands uses plurality voting to defend against adversarial perturbations. The votes for the entire object are firstly produced by applying the part-whole relationships encoded in transformation weight islands to the object parts encoded in embedding islands. The following plurality voting picks the top similar votes and discards many incorrect votes that contains the perturbations. This is in contrast to most of the existing methods that circumvent the perturbation accumulation inherent to the internal architectures of deep models, but import auxiliary techniques such as denoising [80, 81] and self-supervision [82].

The contributions are outlined as the following:

- The proposed Twin-Islands is the first work that explores the part-whole relationship for intrinsic model interpretability in point cloud processing. The model becomes intrinsically interpretable because every proposed transformation weight island represents a part-whole relationship while connecting two embedding islands for part and whole.

- The proposed Twin-Islands is the first work that explores the part-whole relationship for adversarial robustness in point cloud processing. We introduce plurality voting for classification that picks the top similar votes for the object and discards the perturbations in a larger number of incorrect votes.

- The visualization experiment shows a clear match between the real-world part-whole relationships and the model internals, i.e., the transformation weight islands. One application of the intrinsic interpretability is also provided which prunes transformation weights. The adversarial perturbation and point deletion experiments show that the proposed Twin-Islands is more robust than recent CNNs-based and capsule-based models.

4.2 Composing embeddings by quaternions

4.2.1 Check equivariance in capsule networks

In this section, the equivariance in capsule networks are checked. The short answer in advance is no, which is one motivation why one goal of the proposed Twin-Islands model is equivariance.

Equivariance with a transformation of an input image means that the transformation can be transferred to the representation output. Whether the capsule networks are viewpoint equivariant is checked by the following experiment. Denote a capsule network for image classification as a mapping function $f : X \Longrightarrow$ where X is the input image and $Y = f(X)$ the prediction output. Denote $P(X)$ as the pose matrix of the Y-th class capsule. A class capsule is the same as an ordinary capsule but at the final layer of a capsule network. For the MNIST dataset, there are 10 class capsules. Now, consider P as the pose estimate of an object as in this matrix capsule network implementation and T being a 3D viewpoint transformation due to position and orientation change of the camera. If the pose matrix is

invariant with respect to the transformation T, we expect $P(TX) = P(X)$. A model that works by viewpoint invariance discards the viewpoint information in the input image. In contrast, if the pose matrix is equivariant with respect to the transformation T, we expect $P(TX) = TP(X)$. The equivariance and invariance are checked by,

$$\frac{|\det\big(P(TX) - TP(X)\big)|}{|\det(P(X))|}, \tag{4.1}$$

$$\frac{|\det\big(P(TX) - P(X)\big)|}{|\det(P(X))|}, \tag{4.2}$$

respectively in the experiment described in the following. If the capsule networks work by equivariance, the former value should be small whereas the latter one should be large. If the capsule networks work by invariance, the contrast situation happens that the former value should be large and the latter one should be small.

In the experiment, a capsule network is trained on the MNIST dataset with augmented data of affined-transformed images. While checking invariance and equivariance, the image is transformed with novel affine transformations that are not used in the training stage. We used the novel affine transformation because we want to check how the capsule networks recognize images that they have never seen.

The average values of the first and second metrics are 0.965 and 0.111, respectively. This indicates that the matrix capsule implementation [6] preserves invariance more than equivariance.

4.2.2 Composing an embedding by unit quaternions

The part-whole relationship adopted by human beings is viewpoint-invariant, i.e., it is the relative relationship between the part and whole that does not change when the observer's viewpoint changes. What is connected by the viewpoint-invariant is the viewpoint-

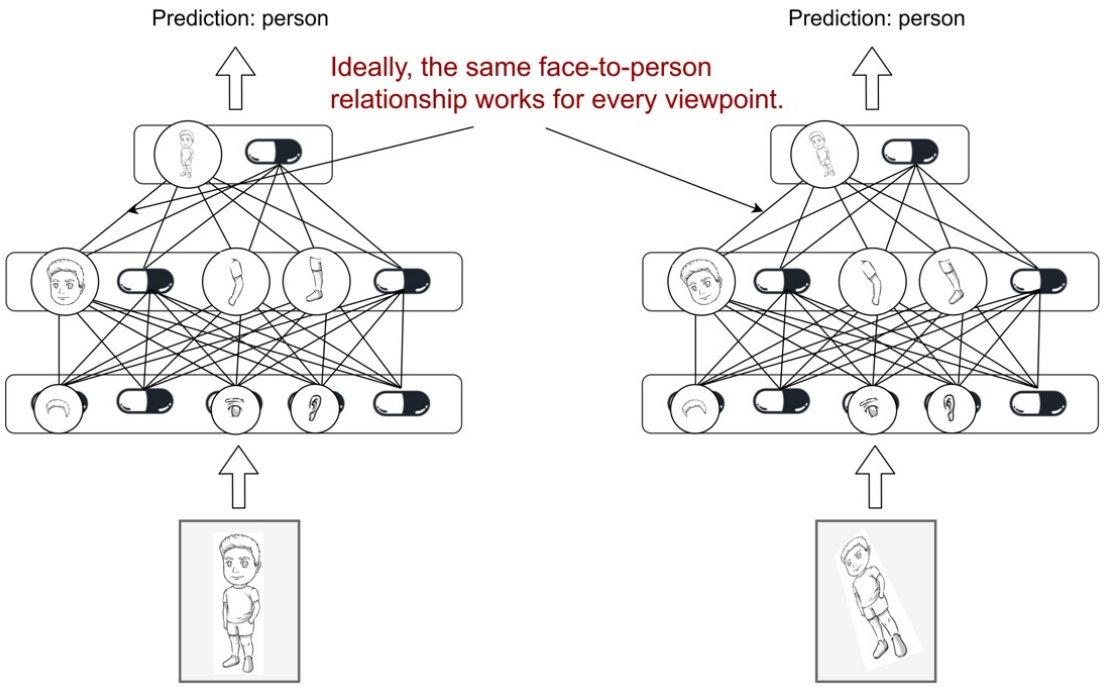


Figure 4.2. Illustration of why the viewpoint-invariant part-whole relationships generalize to novel viewpoints.

equivariant representations for the part and whole. When the observer's viewpoint changes, the observer uses the same part-whole relationship, but the orientations of the part and whole change according to the viewpoint change.

For a point cloud, its equivariant representation for each small patch can be computed into local reference frames (LFRs) [83]. Existing works convert LRFs to the more compact unit quaternion form, and build models based on the quaternion computation rules which reserve the equivariance property in the model's output. Following the equivariance works for point clouds [83, 84], an embedding can be formed as a concatenation of many unit quaternions for the purpose of equivariant representations for part and whole. Formally, an embedding is a concatenation of K unit quaternions as, $\mathbf{p}_i = [(\mathbf{p}_i)_1, \ldots, (\mathbf{p}_i)_K]$, where $(\mathbf{p}_i)_k = p_{ik}^1 + p_{ik}^2\mathbf{i} + p_{ik}^3\mathbf{j} + p_{ik}^4\mathbf{k}$ is a unit quaternion, and $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ are the imaginary units.

4.3 The proposed Twin-Islands

Although GLOM [4] describes a general framework for leveraging part-whole relationships, there are big gaps towards the proposed adversarial robustness and interpretability targets. We detail the gaps and propose corresponding solutions in the following subsections. Note that we remove the index j for the embedding of the last layer, since the last layer has only one embedding that represents the entire object.

4.3.1 Transformation weight islands for interpretability

4.3.1.1 Transformation weight islands

GLOM uses embedding islands to represent the part-whole hierarchy, but how the parts predict the whole is not interpretable because the autoencoder that predicts the whole from its parts is a black box. Instead, we apply similar transformation weights to similar embeddings. Since the similar embeddings that form an island represent a part of an object, their corresponding transformation weights that form a transformation weight island represent the part-whole relationship. Similar to the embedding islands in Eq 2.1, the proposed transformation weight islands are formed as the following,

$$\begin{cases} \mathbf{w}_i = \mathbf{w}_r, & \mathbf{p}_i \text{ and } \mathbf{p}_r \text{ in the same embedding island} \\ \mathbf{w}_i \neq \mathbf{w}_r, & \text{otherwise.} \end{cases} \tag{4.3}$$

4.3.1.2 Illustrative propagation example

Here we first give a 1D example about how to form the embedding islands and transformation weight islands through propagation, which is later extended to real-world point clouds. Let a 1D object, e.g., the umbrella in Figure 4.4, contain two parts located in the

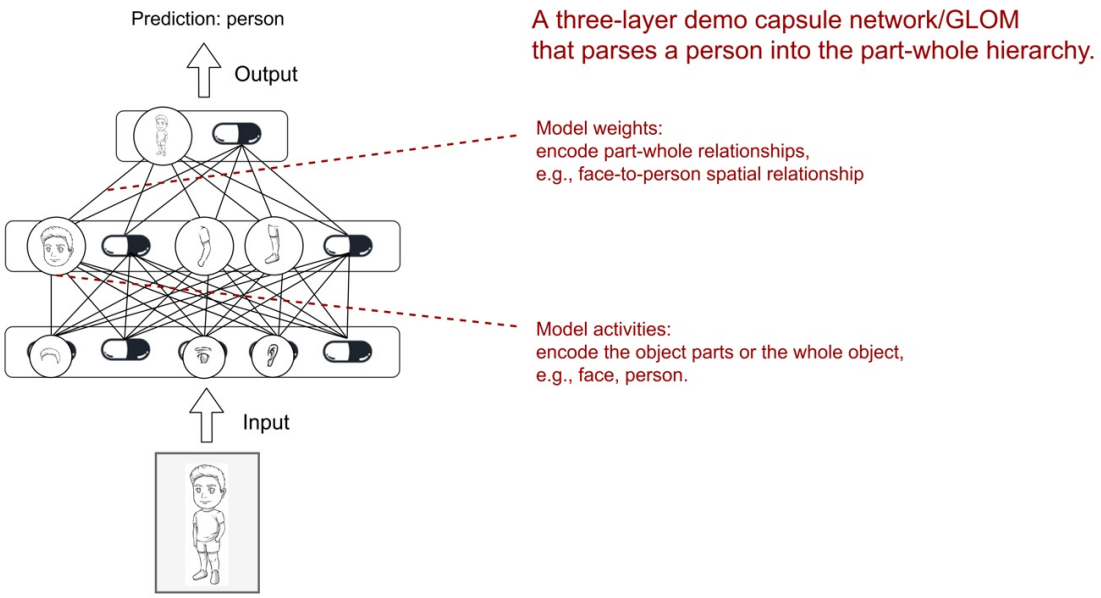


Figure 4.3. Illustration of model interpretability using the person object.

first and second half. Assume the 1D embeddings $p_i \in \mathbb{R}$ and 1D transformation weights $w_i \in \mathbb{R}$ have roughly captured the two parts, but far from perfect, as the initial time in Figure 4.4. Our goal is to let both the embeddings p_i and the transformation weights w_i be the same for each object part, as the final step in Figure 4.4. In each propagation round, the i-th embedding or the i-th transformation weight receives information from every other embedding or transformation weight as follows,

$$p_i \coloneqq p_i + \sum_{r \neq i} (\|x_i - x_r\| \cdot \|p_i - p_r\|)^{-1} \cdot p_r, \tag{4.4}$$

$$w_i \coloneqq w_i + \sum_{r \neq i} (\|x_i - x_r\| \cdot \|p_i - p_r\| \cdot)^{-1} \cdot w_r, \tag{4.5}$$

where $\|x_i - x_r\|$ is the Euclidean distance between the i-th and r-th embedding (or transformation weight) location and $\|p_i - p_r\|$ is the Euclidean distance between the embedding values. As in Figure 4.4, both the embeddings and transformation weights become stable finally.

4.3.1.3 Transformation weight decomposition for clustering

The transformation weight islands in Eq 4.3 are formed when $\mathbf{w}_i = \mathbf{w}_r$. However, the exact identity among transformation weights is hard to achieve for real-world data, and a clustering process is required to divide the transformation weights into islands. Classic clustering algorithms are not sufficient because they require the number of islands as a hyperparameter, e.g., the value of K in K-Means. The proposed Twin-Islands uses an approach that does not affect the propagation process and removes the prior knowledge for the clustering process. Specifically, we introduce a spanning set $\{\mathbf{e}_h\}$ for the transformation weight space, where $\mathbf{e}_h$ is one basis. Every transformation weight is decomposed with a coefficient vector $\mathbf{a}_i = [a_{i,1}, \ldots, a_{i,H}] \in \mathbb{R}^H$ in the running time,

$$\mathbf{w}_i = \sum_h a_{i,h}\mathbf{e}_h. \tag{4.6}$$

Then two transformation weights $\mathbf{w}_i$ and $\mathbf{w}_j$ will be assigned into one cluster if ${}_h a_{i,h} =_h a_{j,h}$. For example, if $\mathbf{a}_1 = [0.1, 0.75, 0.05, 0.1]$ and $\mathbf{a}_2 = [0.05, 0.8, 0.05, 0.1]$, $\mathbf{w}_1$ and $\mathbf{w}_2$ belong to the same cluster. This decomposition requires a hyperparameter of the size of the spanning set, but the number of transformation weigh islands will be automatically determined for different objects without any prior knowledge.

4.3.1.4 Propagation for point clouds

Given the transformation weight decomposition Eq 4.6, passing a transformation weight is equivalently passing its coefficient vector. In each propagation round, a transformation weight's coefficient vector $\mathbf{a}_i$ is updated with two contributions:

1. The coefficient vector $\mathbf{a}_i^{prev}$ from the previous step.

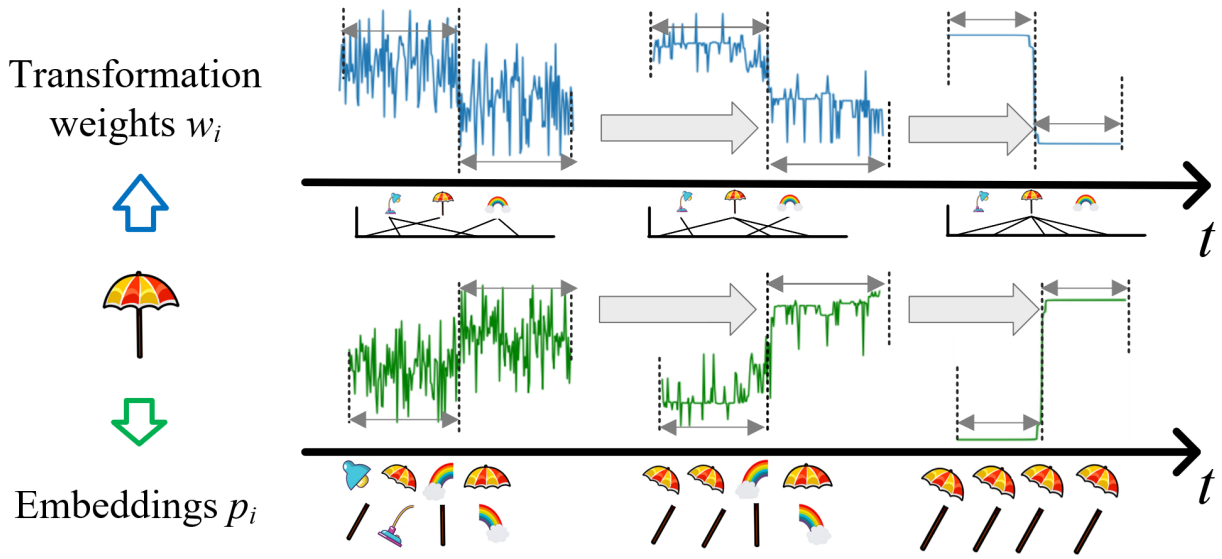


Figure 4.4. A 1D illustrative example on propagation among p_i and w_i. The initial values are generated as constant adding Gaussian noises. p_i and w_i are updated by Eqs 4.4 and 4.5, respectively, and then, p_i converges to two values, representing the shaft and panel; w_i converges to two values too, representing to the shaft-to-umbrella and panel-to-umbrella.

2. Neighbor coefficient vectors of the same layer,

$$\mathbf{a}_i^{nbrs} = \frac{1}{N} \sum_{r \neq i} s_{ir} c_{ir} \mathbf{a}_r, \tag{4.7}$$

where s_{ir} is the similarity between embeddings $\mathbf{p}_i$ and $\mathbf{p}_r$, and $c_{ir} = \|\mathbf{x}_i - \mathbf{x}_r\|^{-1}$ is the closeness between the 3D positions of $\mathbf{p}_i$ and $\mathbf{p}_r$.

The two contributions are summed and normalized as the updated coefficient vector,

$$\mathbf{a}_i = u(\mathbf{a}_i^{prev} + \lambda_a \, \mathbf{a}_i^{nbrs}), \tag{4.8}$$

where $u(\boldsymbol{x}) = \boldsymbol{x}/\|\boldsymbol{x}\|_1$ is the normalization function.

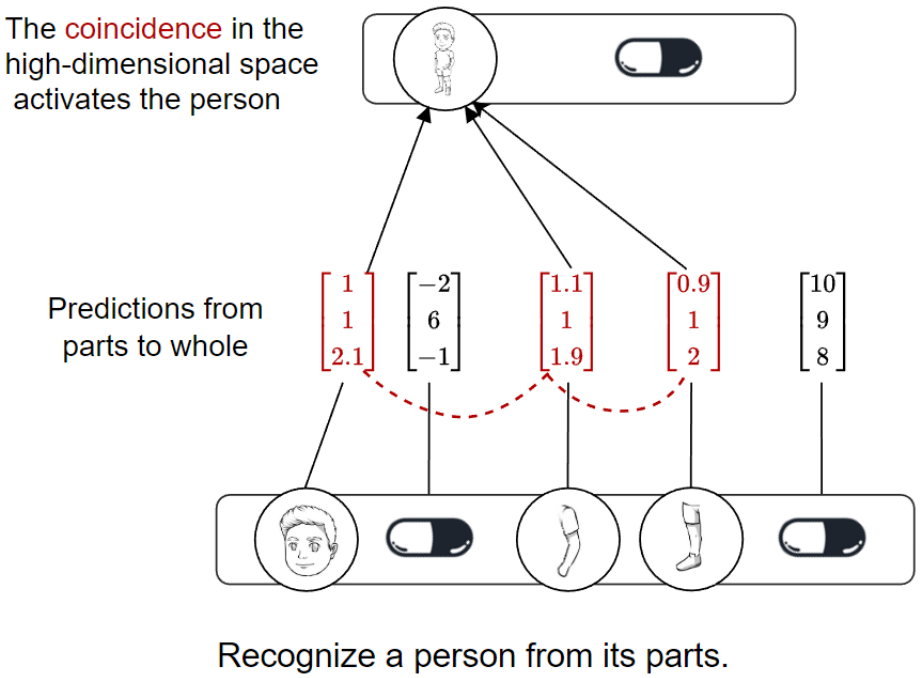


Figure 4.5. Illustration of the high-dimensional coincidence filtering.

Similar to Eqs 4.7 and 4.8, the embeddings $\mathbf{p}_i$ are updated as the following,

$$\mathbf{msg}_{r\to i} = s_{ir} c_{ir} \mathbf{p}_r, \tag{4.9}$$

$$\mathbf{p}_i = \text{mean}(\mathbf{p}_i^{prev}, \lambda_p \mathbf{msg}_{1\to i}, \ldots, \mathbf{msg}_{N\to i}), \tag{4.10}$$

where quaternion mean is used instead of the weighted-sum for maintaining equivariance. The mean of a set of unit quaternions $\{\mathbf{u}_i\}$ is $\text{mean}(\{\mathbf{u}_i\}) =_{\mathbf{u}} \mathbf{u}^\top \mathbb{M} \mathbf{u}$ [85], where $\mathbb{M} \in \mathbb{R}^{4\times 4}$ is defined as $\mathbb{M} := \sum_i \mathbf{u}_i \mathbf{u}_i^\top$. The mean quaternion is the eigenvector of $\mathbb{M}$ corresponding to the maximum eigenvalue and can be solved by SVD. This eigenvector-finding procedure is automatically differentiable using the PyTorch library.

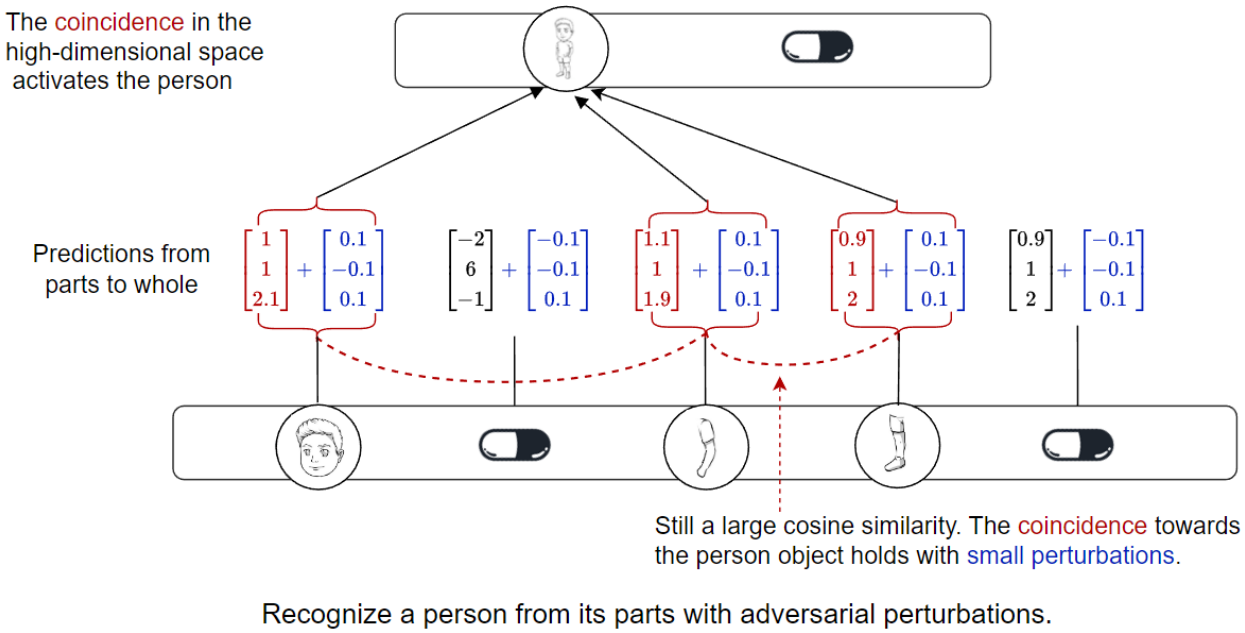


Figure 4.6. Illustration of why the high-dimensional coincidence filtering is robust to adversarial attacks.

4.3.2 Prediction as consensus for adversarial robustness

4.3.2.1 Motivation

After the propagation, each embedding $\mathbf{p}_i$ that represents a part is multiplied with a transformation weight $\mathbf{w}_i$ that represents a part-whole relationship, producing a vote $\mathbf{v}_i$ for the whole object. Each vote is a guess from an embedding of a part towards the embedding of the whole object. Similar votes indicate that they are a good embedding representation for the whole object. However, in the high-dimensional vote space, if a vote contains large noise information, it is hard to be similar to any others. Therefore, only votes that are similar to others, will be used for classifying the input object.

The design philosophy of most of the existing adversarial attacks is to add a small perturbation that misleads the classifier to yield a wrong result. Although the perturbation is small, its impact will be accumulated when making the final decision due to the weighted-sum fashion classification process. In the proposed Twin-Islands, a small perturbation cannot

significantly change the similarity between votes. Then we propose a plurality voting process by a clipping function that removes those noisy votes, thereby, perturbations cannot be accumulated over all votes.

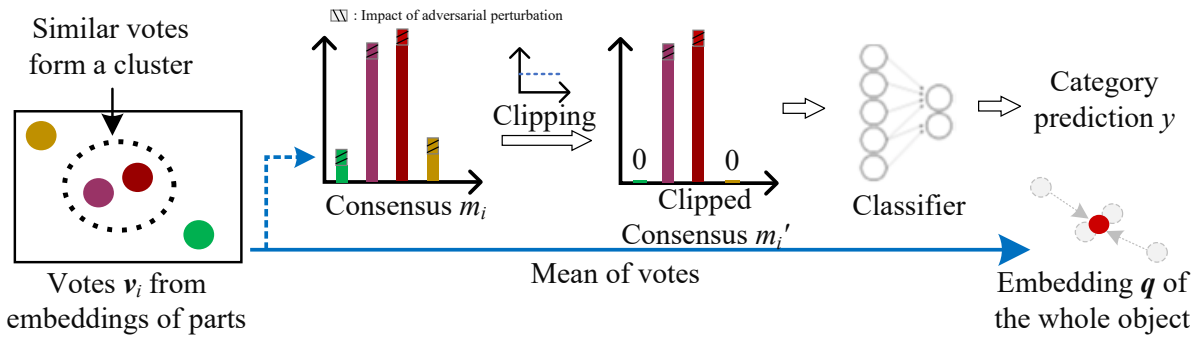


Figure 4.7. Predict the embedding $\mathbf{q}$ and category y of an object, using votes $\mathbf{v}_i$ towards $\mathbf{q}$. $\mathbf{q}$ is computed as the mean of the votes it receives. The i-th consensus value m_i is computed as the similarity between $\mathbf{v}_i$ and $\mathbf{q}$. The perturbation can only affect the large consensus values, excluding the small consensus values from noisy votes.

4.3.2.2 Implementation of motivation

The implementation of the above motivation is shown in Figure 4.7. Given the quaternions composition for each embedding, the vote $\mathbf{v}_i$ for the embedding of the object is computed by transforming each quaternion of $\mathbf{p}_i$ by quaternion multiplication $\circ$,

$$(\mathbf{v}_i)_k = (\mathbf{p}_i)_k \circ (\mathbf{w}_i)_k, \tag{4.11}$$

where $\mathbf{v}_i$, $\mathbf{p}_i$, $\mathbf{w}_i$ are all composed of K quaternions and the subscript k retrieves the k-th quaternion. The embedding $\mathbf{q}$ of the object is computed as the mean of the votes,

$$(\mathbf{q})_k = \text{mean}\big((\mathbf{v}_1)_k, \ldots, (\mathbf{v}_N)_k\big), \tag{4.12}$$

Algorithm 2 The propagation and classification process.

Input: Embeddings $\mathbf{p}_i$ of parts of the object.

Output: Embedding $\mathbf{q}$ of the object, category prediction y.

```
Initialize the coefficient vectors a_i {Eqs. 4.15 & 4.16}
for t = 1, ··· , T iterations do
    Update contributions: a_r to a_i, p_r to p_i {Eqs. 4.7 & 4.9}
    Update a_i and p_i using contributions {Eqs. 4.8 & 4.10}
    Update transformation weights w_i using a_i {Eq. 4.6}
    Update votes v_i = w_i p_i {Eq. 4.11}
    Update embedding q as mean of {v_i} {Eq. 4.12}
    Update clipped consensus vector m' {Eq. 4.13}
    Update classification decision y based on m'
end for
return q, y
```

A consensus vector $\boldsymbol{m} = [m_1, \ldots, m_N]$ is used to describe the similarity between the votes $\mathbf{v}_1, \ldots, \mathbf{v}_N$ and the embedding $\mathbf{q}$ of the object. The i-th consensus value m_i is measures the similarity between $\mathbf{v}_i$ and $\mathbf{q}$ as,

$$m_i = \frac{1}{K} \sum_k \big(1 - d((\mathbf{v}_i)_k, (\mathbf{q})_k\big). \tag{4.13}$$

where the function $d(\mathbf{a}, \mathbf{b}) = 2\cos^{-1}(|\langle \mathbf{a}, \mathbf{b} \rangle|)$ computes the distance between two unit quaternions $\mathbf{a}, \mathbf{b}$.

A small consensus value m_i indicates the i-th vote is different from other votes and tends to be noise. The small consensus values m_i are clipped to 0 as the following,

$$m_i' = \begin{cases} m_i, & m_i \geq \tau \\ 0, & \text{otherwise.} \end{cases} \tag{4.14}$$

This threshold τ is linearly increased from 0 to 0.5 in the first 10 epochs in programming. The clipped consensus vector contains information from the plural votes that agree with each other, and is input to a linear classifier for predicting the object category.

4.3.3 Implementation details

Algorithm 2 shows how to form the last layer from its previous layer. Forming other layers is similar to this process.

4.3.3.1 Coefficient vector initialization

The coefficient vector $\mathbf{a}_i$ should be viewpoint-invariant to pass the viewpoint-equivariance to the last-layer embedding $\mathbf{q}$. To this end, the coefficient vectors are initialized using viewpoint-invariant canonical point positions,

$$\mathbf{x}'_{ik} = \text{mean}^{-1}\big((\mathbf{p}_1)_k, \ldots, (\mathbf{p}_N)_k\big) \circ \mathbf{x}_i, \tag{4.15}$$

$$\mathbf{a}_i = \psi([\mathbf{x}'_{i1}, \cdots, \mathbf{x}'_{iK}]), \tag{4.16}$$

where $\mathbf{x}_i$ is the point position that embedding $\mathbf{p}_i$ locates, and $\psi(\cdot)$ is implemented as an MLP.

4.3.3.2 Network architecture

For generating the input, we first compute the LRFs that equivariantly describe every point's neighborhood. The LRFs extraction algorithm is chosen as the FLARE algorithm [83] following the work [30]. Every LRF is then converted to a quaternion and replicated K times to form the input of Twin-Islands. We stack two layers to let the first layer extract parts of the object and let the second layer capture the entire object. Every embedding in Twin-Islands is composed of $K = 128$ quaternions. The number of embeddings can be seen in the visualization experiments. Two propagation ($T = 2$) are performed for embeddings and transformation weights after the initialization. Both hyperparameters λ_a and λ_p used

Method	Params	NR/NR	NR/AR
PointNet	3.5M	88.45	12.47
PointNet++	1.5M	89.82	21.35
DGCNN	2.8M	**92.90**	29.74
KDTreeNet	3.6M	86.20	8.49
Point2Seq	1.8M	92.60	10.53
Spherical CNNs	0.5M	-	43.92
PRIN	1.5M	80.13	68.85
PPF-FoldNet	3.5M	70.16	70.16
QENet	0.4M	74.43	74.07
REQNN	2.9M	83.03	**83.03**
Proposed	**0.3M**	77.40	77.40

Table 4.1. Classification evaluation on the ModelNet40 dataset. NR/NR and NR/AR are train/test settings, where NR means not rotated and AR means arbitrarily rotated.

in the propagation process are set to 0.3 in the first propagation step and 1 in the second step. The hyperparameter H is set to 64.

4.4 Experiments

4.4.1 Experiment settings

The model is trained using classification supervision on the ModelNet40 dataset. The ModelNet40 dataset consists of 40 common objects in point cloud format. Its official split has 9,843 point clouds for training and 2,468 for testing. Since the proposed Twin-Islands is designed to be equivariant, it is trained with each point clout's default orientation. The model is trained with 150 epochs, a batch size of 24, Adam optimizer, and a constant learning rate of 5e-3. The code is written in PyTorch, and runs on two A5000 GPUs.

Method	avg	chair	bed	sofa	toilet	monitor
Mean LRF	0.35	0.32	0.36	0.34	0.41	0.34
PCA	0.67	0.69	0.70	0.67	0.68	0.61
PointNetLK	0.38	0.43	0.31	0.40	0.40	0.31
IT-Net	0.19	0.10	0.22	0.17	0.20	0.28
QENet	0.09	0.08	0.10	0.08	0.11	0.08
REQNN	0.16	0.13	0.17	0.14	0.19	0.16
Proposed	**0.04**	**0.04**	**0.03**	**0.03**	**0.05**	**0.04**

Table 4.2. Relative angular error (RAE) of orientation estimation on rotational asymmetry objects.

4.4.2 Equivariance evaluation

4.4.2.1 3D shape classification

This task evaluates Twin-Islands's classification ability for objects placed at novel orientations. More clearly, the training point clouds are not rotated (NR), and the testing point clouds are arbitrarily rotated (AR). For a thorough comparison, the testing accuracy on not rotated (NR) orientations is provided as well. The following state-of-the-art methods are compared in Table 4.1: PointNet [86], PointNet++ [87], DGCNN [88], KDTreeNet [89], Point2Seq [90], Spherical CNNs [91], PRIN [92], PPF-FoldNet [93], QENet [30], REQNN [84]. As in Table 4.1, most methods perform well on the familiar not rotated (NR) objects, but fail to recognize novel arbitrarily rotated objects (AR) that are not covered in the training set. In contrast, the proposed model achieves a decent accuracy of 77.40% for novel orientations, with the fewest parameters of 0.3M.

4.4.2.2 Orientation estimation

This task quantitatively evaluates how accurately the last-layer embedding $\mathbf{q}$ captures the input's orientation. Note that the orientation information is not learned with orientation supervision, but passed from the equivariant quaternion input through the quaternion

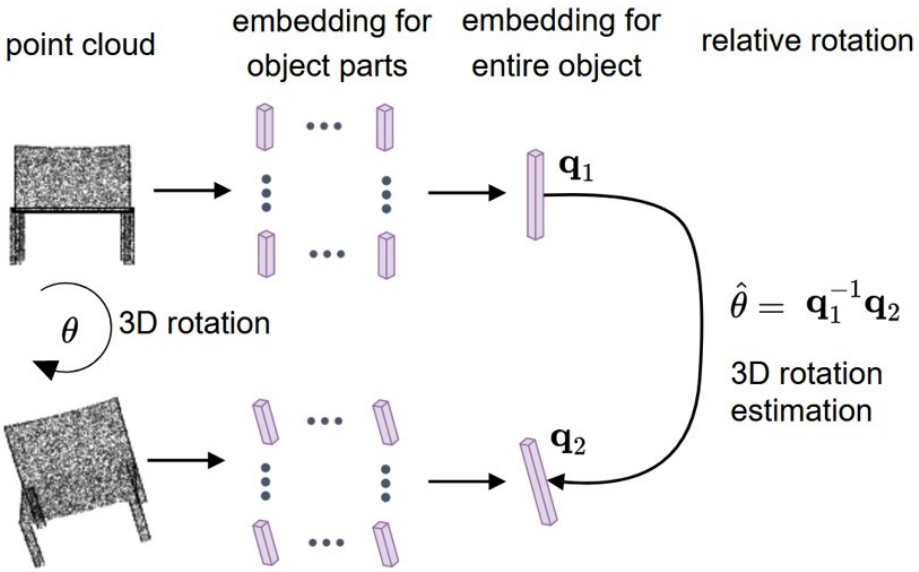


Figure 4.8. Illustration of estimating the object's orientation using quatenions-composed embeddings.

arithmetic. We test five rotational asymmetry objects, including chair, bed, sofa, toilet, and monitor. Two point clouds in canonical and rotated orientation $\boldsymbol{\theta}$ are input into the network, resulting in two embeddings $\mathbf{q}$ and $\mathbf{q}^{rot}$. The predicted orientation $\hat{\boldsymbol{\theta}}$ is the relative transform between the $\hat{k}$-th quaternion of $\mathbf{q}$ and $\mathbf{q}^{rot}$, with $\hat{k}$ being the quaternion that has the largest consensus over all votes. The metric is the relative angle in degree (RAE), computed by $d(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}})/\pi$. The reported RAE is averaged on five random rotations for each test point cloud.

The following methods are compared: a naive averaging of the LRFs (Mean LRF), the principal axis alignment (PCA), PointNetLK [94], IT-Net [95], QENet [30], and RE-QNN [84]. PointNetLK and IT-Net are 3D networks that iteratively align two given point sets. Methods invariant to rotations such as PointNet and PointNet++, cannot be used to estimate the orientation and are not evaluated. It can be seen from Table 4.2 that the proposed model achieves the best average RAE of 0.04.

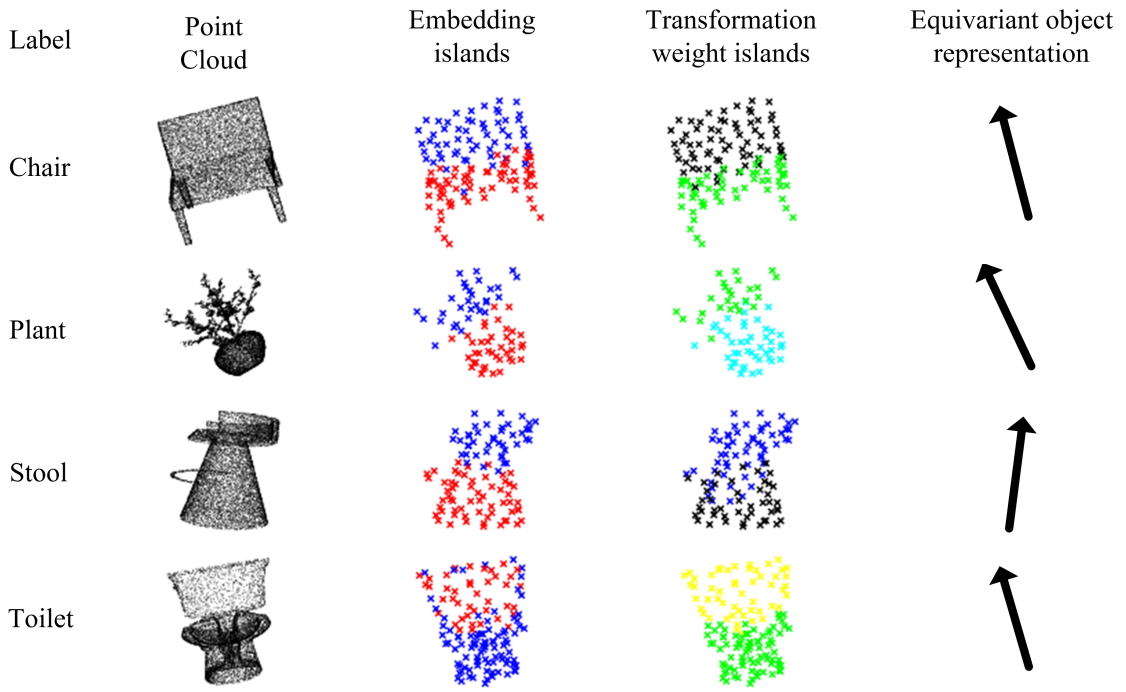


Figure 4.9. Visualization of Twin-Islands processing selected point clouds. The second and third columns are embedding islands and transformation weight islands, representing the parts and the part-whole relationships, respectively. The last column is one embedding representing the whole object, which is equivariant with respect to the inputs.

4.4.3 Interpretability analysis

4.4.3.1 Visualization

This experiment shows that the part-whole relationships are successfully learned into the model's transformation weights. For visualizing part-whole relationships, the visualization paints points that mainly use the same weight basis with the same color, as in Figure 4.9. The embeddings are visualized with the help of K-Means clustering. Figure 4.9 shows some representative results. The overall results for 40 categories are shown in the Appendix.

Object	No pruning	Relationships pruning ratio		
		75%	50%	25%
bathtub	82.0	48.0	72.0	80.0
bed	94.0	86.0	88.0	94.0
chair	97.0	86.0	98.0	97.0
desk	87.2	83.72	90.7	89.5
dresser	96.5	58.14	95.4	95.4
avg.	**92.4**	75.4	90.5	**92.4**

Table 4.3. Classification accuracy on five objects after pruning the less frequently used transformation weight bases.

4.4.3.2 Part-whole relationship pruning

This experiment shows that some transformation weight bases can be pruned if they do not represent any important part-whole relationship. The importance of a transformation weight basis is counted by its frequency in the training set. In Table 4.3, we show the classification accuracy on the first five classes of ModelNet10, with removing the 25%, 50%, 75% less frequently used bases. The removal is achieved practically by setting corresponding coefficients $a_{i,h}$ to 0. It is important to note that the pruned network is directly evaluated without any fine-tune or retraining. Table 4.3 shows that the network achieves the same accuracy with pruning 25% basis. With a large 50% pruning ratio, the accuracy drops slightly from 92.4% to 90.5%. With an aggressive 75% pruning ratio, the accuracy drops drastically from 92.4% to 75.4%.

4.4.4 Adversarial robustness evaluation

This experiment compares Twin-Islands with the recently proposed CNN-based REQNN model and the capsule-based QENet model. Both models leverage quaternions for equivariance for point clouds as Twin-Islands. Since the model is designed to be equivariant, the attacks are performed on objects placed at novel orientations that are not covered in the training data.

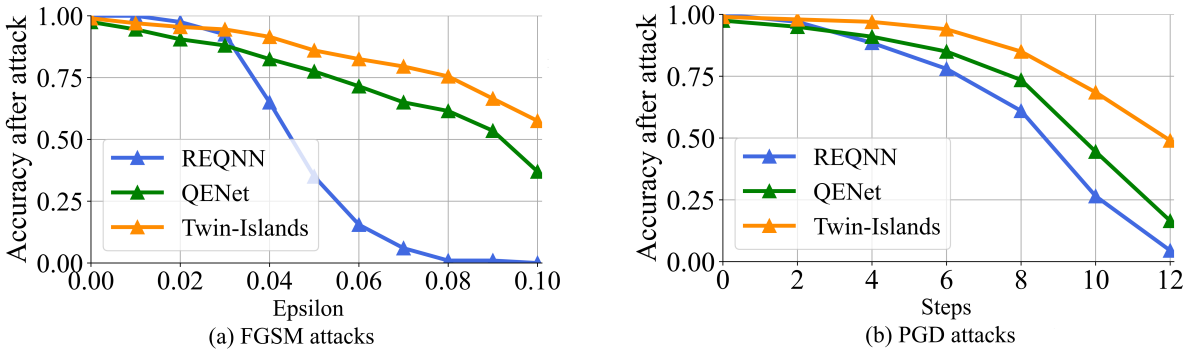


Figure 4.10. Classification accuracy after FGSM attacks (a) and PGD attacks (b). Attacks are performed on objects placed at novel orientations that are not covered in the training set.

4.4.4.1 Robustness to adversarial perturbations

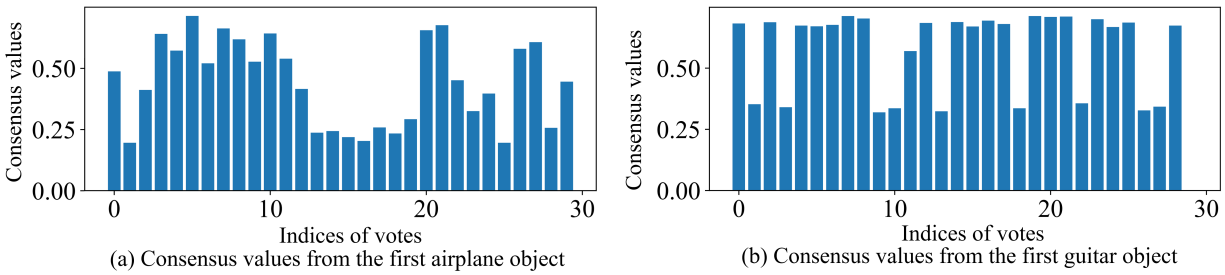
Our adversarial attacks adopt the popular fast gradient sign method (FGSM) [2] and projected gradient descent attack (PGD) [96]. Since the models are not performed well on some categories even before the attack, we perform the attacks on the categories where all three models REQNN, QENet, and Twin-Islands obtain more than 95% accuracy before the attack. As the results in Figure 4.10, the proposed Twin-Islands is more robust to both FGSM and PGD attacks than the CNN-based REQNN and capsule-based QENet.

4.4.4.2 Robustness to point deletion

This experiment evaluates the model's classification accuracy after removing random small 12-point-patches from the point cloud. This point removal on QENeT and Twin-Islands is achieved by simply deleting these points from the point cloud. For REQNN which always needs 1024 points as its input, we fill the removed patches with their neighbor points. In Table 4.4, the proposed Twin-Islands is more robust against point deletion than the CNN-based REQNN and capsule-based QENet.

Method	No deletion	Point deletion ratio				
		5%	10%	15%	25%	50%
QENet	74.07	75.07	66.38	57.70	40.91	13.01
REQNN	**83.03**	74.84	65.56	56.98	40.40	12.85
Twin-Islands	77.40	**76.29**	**74.75**	**72.61**	**67.51**	**42.34**

Table 4.4. Classification accuracy evaluation (%) after randomly removing part of the input point cloud.



(a) Consensus values from the first airplane object

(b) Consensus values from the first guitar object

Figure 4.11. The consensus value distribution computed from the first object of airplane and guitar category.

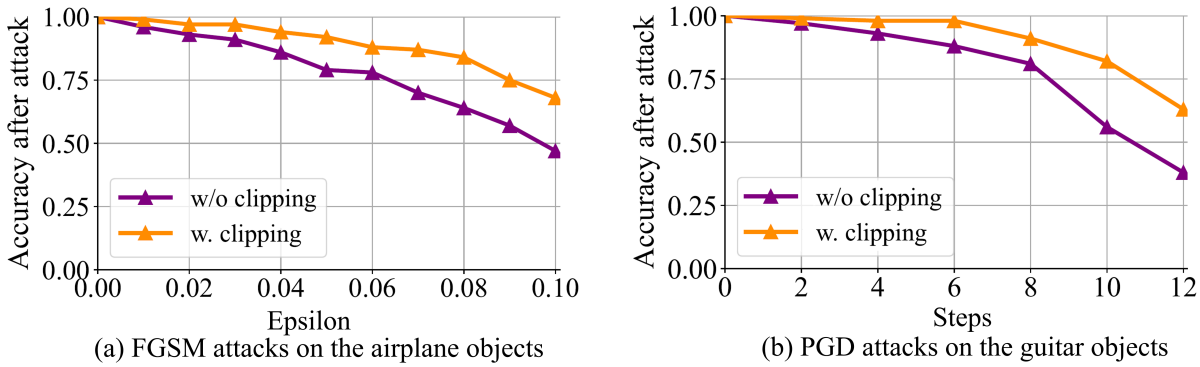
4.4.4.3 Analysis on clipping operation

Figure 4.11 shows the consensus values (Eq 4.13) produced from the first testing point cloud of the airplane and guitar class. It can be seen that there is a clear gap between the large and small consensus values. Figure 4.12 shows that there is a improvement on adversarial robustness after clipping the small consensus values.

4.5 Related works

4.5.1 Capsule networks and GLOM

Capsule networks [25, 5] aim to learn the part-whole hierarchy too, but they have a significant difference from GLOM. While capsule networks use multiple capsules to represent each of the possible object part types at a local patch, GLOM uses one embedding to represent the object part no matter what is this object part type by regressing the embeddings



(a) FGSM attacks on the airplane objects

(b) PGD attacks on the guitar objects

Figure 4.12. Classification accuracy on the airplane category, w. and w/o. clipping.

into islands. The recent work [36] is an implementation of GLOM for interpreting image classification, but the equivariance is not achieved and the robustness to adversarial attack is not explored. GlomFace transferred the part-whole hierarchy in the extreme occlusion problem in face landmarks alignment [37].

4.5.2 Network interpretation and adversarial attacks

Most of the interpretation methods explore the interaction between the input and output (or a higher layer's activation maps), including finding images or image patches that maximize the targets neurons' activation [16, 17, 18], displaying the loss function's gradient w.r.t the input image [19, 20], and fitting the network's prediction on every single image using a simpler model, e.g., a linear classifier [21]. Methods for interpreting point cloud processing mostly borrow generally the same ideas for images [80, 81, 82]. This paper explores intrinsic interpretability through part-whole relationships.

Adversarial attacks can be divided into white-box attacks [2, 96] and black-box attacks [97]. Generally speaking, white-box attacks have more threats than black-box attacks because the white-box attacks have the access to the model architecture and weights. Therefore, we evaluate the proposed model with the popular white-box attacks FGSM and PGD.

4.6 Conclusion and Future Work

This paper proposes a new architecture, named Twin-Islands, for point clouds. Visualization and pruning experiments demonstrate its interpretability, and adversarial perturbation and deletion experiments show its robustness. In the future, there is a broad space to explore. One direction is to reproduce similar results for images. But this is difficult because 3D coordinate needs to be inferred from 2D images instead of directly given as in point clouds, the random background needs to be excluded from objects, and significant object deformation needs to be properly handled. One direction is to further utilize interpretability, such as dissembling and assembling two or more models. One direction is to quantitatively measure the extent of interpretability via measuring how accurately the islands segment the object.

Chapter 5: Group ensemble block: subspace diversity improves coarse-to-fine retrieval

5.1 Introduction

Image retrieval systems find related database images for a user-input query sorted by their relevance scores. The relevance score quantifies how closely a database image matches the query, usually computed by their cosine similarity in the embedding space. Previous image retrieval methods usually improve the reliability of relevance scores through three approaches. The first approach utilizes class-level discrimination, mapping images of the same class to nearby locations in the embedding space [98, 99, 100, 101]. The second approach utilizes instance-level discrimination, leveraging self-supervised learning that pulls the embeddings of the same image's different augmentations together and pushes the embeddings of different images apart [102, 103, 104, 105]. The third approach is to take an ensemble average of several models trained under various conditions [106, 107, 108, 109].

Since the focused aspects of the three approaches are not mutually exclusive, it may be possible to combine the benefits of each approach. For example, the recent method [110] has combined class-level and instance-level discrimination, resulting in a performance gain. However, utilizing ensemble learning with other approaches meets practical challenges from the perspective of computation and memory costs. To support this claim, we summarize the cost of each approach as the following:

- The class-level discrimination usually requires the current training image and at least one image from the same or different category to maximize or minimize their embedding

similarity. The cost varies with the specific methods, and an inexpensive method Scalable Neighborhood Component Analysis (SNCA) [101] adds little computation and memory cost to the base network.

- The instance-level discrimination usually requires a siamese network that contains two identical or very similar subnetworks during training. The cost is about $2\times$ during training and $1\times$ (no additional cost) after the model is trained.

- The ensemble methods usually use N *entire* large deep networks, with each network learning a sub-embedding. The cost is $N\times$ for both training and testing stages.

As in the summary, the ensemble approach is expensive primarily due to the cost of ensembling several entire networks. Therefore, we must answer the following question: *instead of ensembling many independent networks, is it possible to execute ensemble learning in a low-cost module, so that it can be used with many other methods?* We propose that the random subspace technique has the potential of solving this question. Instead of model differences, diversity can also come from the differences between many random sampled subspaces of the data or feature representation, because the diversity between subspaces guarantees the diversity between their processed results. The effectiveness of random subspaces has already been shown for traditional models such as naive Bayes and decision trees [111, 112].

However, in existing literature, the usage of random subspaces has not been well explored for deep neural networks from the perspectives of cost and reusability, limiting its applications such as the image retrieval task. To fill this gap, we tailor the usage of random subspaces for deep neural networks from these two perspectives. First, in terms of cost, many computations can be executed in parallel, such as subspace sampling and subspace processing. With this parallelization, the usage of subspace sampling does not slow down the training process, even though it takes numerous iterations. We also empirically show that the parallel subspace processing can be implemented as a linear transformation for low computation cost, and its

weights can be shared across the processing of every subspace for low memory cost. Second, in terms of reusability, it is possible to integrate all necessary steps into one small block. Thus, this block will be flexible similar to batch normalization [64] and dropout [113] techniques in that all can be easily deployed elsewhere. For example, we replaced the ResNet50's last linear layer with the proposed ensemble block, which increased the CIFAR-10 classification accuracy from 94.82% to 95.76%.

We implement the random subspace method proposed above in the proposed group ensemble block which sufficiently utilizes the subspaces' diversity via the masking and merging processes. The masking process perturbs each subspace which further increases the diversity and performance. The merging strategy leverages the advantages of both averaging and concatenating: processed outcomes of subspaces inside the same group are averaged to increase the number of utilized subspaces, and sub-embeddings averaged from each group are concatenated to increase the diversity inside the concatenated embedding vector.

The proposed group ensemble block is evaluated on the coarse-to-fine image retrieval task where the training and testing settings have an apparent gap. The coarse-to-fine image retrieval task aims to retrieve images from the same fine-grained category (e.g., maple trees, oak trees, or palm trees) as the query after the model is trained with only coarse-level annotation (e.g., trees) available. In other words, the model needs to learn fine-grained classes with only coarse-level labels.

The contributions of our work are outlined as the following:

- The cost of random-subspace-based ensemble learning for deep models is significantly decreased by the proposed group ensemble block. In addition, one can effortlessly replace a linear layer in deep models with the proposed ensemble block, so that its applications are not limited to image retrieval. As an illustration, we provide an example of CIFAR-10 classification using ResNet50, where the accuracy increases from 94.82% to 95.76%.

- The proposed group ensemble block sufficiently utilizes the subspaces' diversity via masking and merging processes. The masking process perturbs each subspace which further increases both diversity and performance. The merging strategy combines and leverages the advantages of both averaging and concatenating strategies.

- Using the proposed group ensemble block, we achieve the state-of-the-art accuracy for the coarse-to-fine image retrieval problem on the CIFAR-100 and ImageNet datasets. Our quantitative evaluation shows that different subspaces' outcomes are both low-correlated and complementary to each other.

5.2 Related works

5.2.1 Coarse-to-fine problems

This subsection reviews coarse-to-fine literature where the testing-time labels are finer than those available at training time. [114] and [101] both study coarse-to-fine image classification and suppose that the models are trained with coarse-level labels, but the training set's fine-grained labels are available as ground truth at testing time. [114] trains the AlexNet on the coarse classes in the training stage. While testing, they use the output of the FC7 layer (second-to-last layer) as the embedding and then predict the fine-grained category with a kNN classifier. [101] proposes SNCA, which replaces the Softmax classifier of deep convolutional neural networks with a non-parametric classifier. The non-parametric classifier classifies an image into a specific class based on its nearest neighbor images. [115] studies the task of coarse-to-fine few-shot image classification. After training the model with coarse-level labels, they annotate one or a few fine-grained labels to support fine-grained classification. [110] studies coarse-to-fine image retrieval, combining class-level discrimination and instance-level discrimination. They adopt SNCA for class-level discrimination and the self-supervised learning method BYOL [104] for instance-level discrimination.

Many existing methods, e.g., [114, 101], can be adapted for the coarse-to-fine image retrieval problem though they were not created to do so. Actually, any network's neuron activations can be utilized as the embedding representation for the coarse-to-fine image retrieval task. However, only one existing publication [110] clearly and specifically targets the coarse-to-fine image retrieval problem to the authors' best knowledge. The authors of [110] combine class-level and instance-level discrimination. We further combine the ensemble learning approach by proposing a low-complexity group ensemble block.

5.2.2 Ensemble methods for image retrieval

Ensemble learning combines information from multiple sources to obtain better generalization performance. It includes many techniques such as bagging, boosting, and random subspaces. Before the deep learning era, ensemble learning had been well researched with traditional machine learning methods such as naive Bayes, decision trees, and support vector machines. However, unlike these traditional methods, deep models have much more complex architectures and many more parameters that are usually trained with much more data. Then utilizing ensemble learning with deep models may incur significant computation and memory increases. In the following, we name a few such works in the area of image retrieval. These methods divide the embedding into multiple sub-embeddings and produce each sub-embedding individually to achieve diversity among various sub-embeddings. [108] clusters the training set into several subsets and learns a model for each subset. [109] randomly groups classes into different superclasses and trains several models with these superclass labels. [116] shares many lower layers across the ensembles, but each ensemble has 15 non-shared layers, which is still expensive.

redThe random subspace method produces diversity among subspaces sampled from the data or feature representation. Random subspaces potentially require much lower computation and memory costs compared to many other ensemble learning methods, because the

model parts before the sampling operation can be shared across different subspaces. However, this approach has not been well explored for deep neural networks in terms of cost and reusability. We fill this gap by the proposed group ensemble block.

5.2.3 Class-level and instance-level discrimination for image retrieval

The class-level discrimination approach forces images of the same class to be mapped to nearby locations in the embedding space. Many methods of this approach minimize (or maximize) the similarity between a randomly sampled image pair if they are from different (or the same) classes [98, 99]. All these methods can also be extended to triplet [117, 118], and n-tuple [100]. The neighborhood component analysis methods [119, 101, 120] achieve class-level discrimination by classifying an image based on its distance to each image in the training set.

The instance-level discrimination is achieved through self-supervised learning (SSL) that maximizes the similarity between the embedding representation of an image's random augmented views. SSL methods usually use a siamese network's two branches to process an image's two augmentations and do not require any annotation. For the SSL methods, it is crucial to avoid the degenerate solution, where the representation of every image collapses to the same constant. To achieve that, both SimCLR [102] and MoCo [103] utilize negative pairs — repulsing different images (negative pairs) while attracting the same image's various augmentations (positive pairs). BYOL [104] practically shows that negative pairs are not necessary. Instead, they use a large batch size and a momentum encoder for one of the siamese network's two branches. SimSiam [105] further demonstrates that negative pairs, large batch size, and the momentum encoder are unnecessary, but a 'stop-grad' operation applied on one branch is crucial. We adopt the SimSiam method for SSL as it needs less computation and memory cost than other methods.

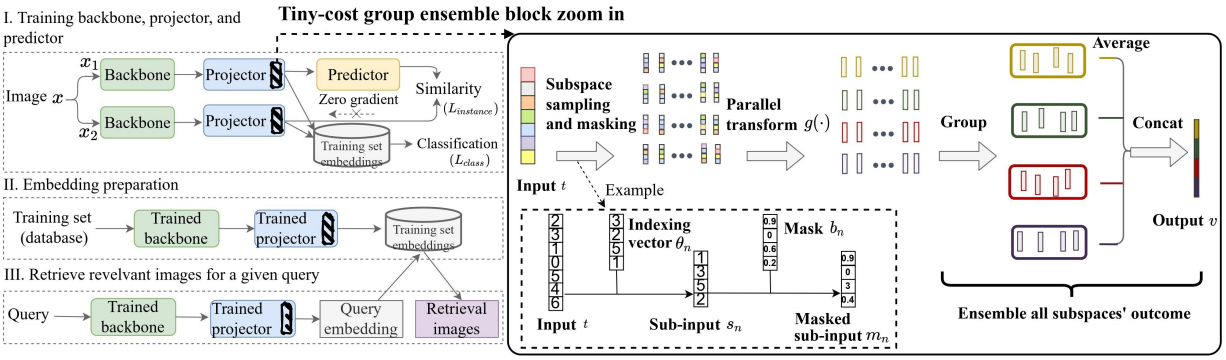


Figure 5.1. Illustration of the proposed low-complexity group ensemble block, and the workflow of combining class-level discrimination, instance discrimination, and ensemble block together for the coarse-to-fine image retrieval. In practice, we use more subspaces than this illustration, e.g., 1024.

5.3 Proposed method

The task of coarse-to-fine image retrieval aims to find relevant database images for a given query image. Its difference from the usual image retrieval task is that: while the training set only provides coarse-level annotations (e.g., trees), the retrieval images that are counted correct must be from the same fine-grained category (e.g., maple trees, oak trees, or palm trees) as the query. The database images are retrieved and ordered according to their semantic relevance to the query. The i-th database image $\boldsymbol{x}_i$'s relevance score to the query $\boldsymbol{x}$ is defined as the following,

$$Rel(\boldsymbol{x}, \boldsymbol{x}_i) = \frac{\boldsymbol{v}^T \boldsymbol{v}_i}{\boldsymbol{v}\boldsymbol{v}_i} + \log\left(\frac{p(y_{\boldsymbol{x}} = y_{\boldsymbol{x}_i})}{1 - p(y_{\boldsymbol{x}} = y_{\boldsymbol{x}_i})}\right), \tag{5.1}$$

where $\boldsymbol{v} \in \mathbb{R}^{D_{emb}}$ and $\boldsymbol{v}_i \in \mathbb{R}^{D_{emb}}$ are $\boldsymbol{x}$ and $\boldsymbol{x}_i$'s embeddings output by the same model trained with coarse-level labels, $y_{\boldsymbol{x}}$ and $y_{\boldsymbol{x}_i}$ are $\boldsymbol{x}$ and $\boldsymbol{x}_i$'s coarse-level annotations. In Eq 5.1, the first term computes the cosine similarity in the embedding space; the second term utilizes the probability $p(y_{\boldsymbol{x}} = y_{\boldsymbol{x}_i})$ that the query is from the same coarse-level category as the i-th

database image. Note that the second term only uses coarse-level labels about the training set and does not require any fine-grained label.

In this section, we first illustrate the proposed group ensemble block, then describe one of its applications, i.e., aggregating the instance-level discrimination method SimSiam [105] and class-level discrimination method SNCA [101] for the coarse-to-fine image retrieval.

5.3.1 Proposed group ensemble block

blackDenote the network's input image as $\boldsymbol{x}$, and denote the network's many layers before the group ensemble block as $f(\cdot)$. In our experiments, the layers $f(\cdot)$ include the ResNet50 backbone and the projector's two linear layers. The group ensemble block's output is utilized as the image's embedding for image retrieval. Our group ensemble block, as shown in Figure 5.1, consists of three steps as the following:

Table 5.1. Major symbols for formulating and processing random subspaces in Section 5.3.1.

$\boldsymbol{x}$	input image of the network
$f(\cdot)$	the network parts before the group ensemble block, including the ResNet50 backbone and the projector's two linear layers
$\boldsymbol{t}$	input of the group ensemble block, $\boldsymbol{t} = f(\boldsymbol{x}) \in \mathbb{R}^{D_{in}}$
N	number of sampled subspaces inside the group ensemble block
L	length of $\boldsymbol{\theta}_n$, $\boldsymbol{s}_n$, $\boldsymbol{b}_n$, $\boldsymbol{m}_n$, and $L \ll D_{in}$
$\boldsymbol{\theta}_n$	the indexing vector that samples the n-th subspace from $\boldsymbol{t}$
$\boldsymbol{s}_n$	the n-th sub-input sampled with $\boldsymbol{\theta}_n$
$\boldsymbol{b}_n$	mask for $\boldsymbol{s}_n$, consists of elements between [0, 1] (Eq 5.2)
$\boldsymbol{m}_n$	the n-th masked sub-input, $\boldsymbol{m}_n = \boldsymbol{s}_n \circ \boldsymbol{b}_n$
$g(\cdot)$	the parallel transformation applied to each masked sub-input $\boldsymbol{m}_n$

a) Subspace sampling: blackDenote the ensemble block's input as $\boldsymbol{t} = f(\boldsymbol{x}) \in \mathbb{R}^{D_{in}}$. N indexing vectors $\boldsymbol{\theta}_n \in {\mathbb{Z}_+}^L$ are randomly generated to define the N sampled subspaces. Each element of $\boldsymbol{\theta}_n$ takes a value from $\{1, 2, \cdots, D_{in}\}$, and the N indexing vectors are never updated once generated. Then N sub-inputs $\boldsymbol{s}_n \in \mathbb{R}^L$ are sampled from input $\boldsymbol{t}$ by taking the respective elements of $\boldsymbol{t}$ assigned by the indexing vectors $\boldsymbol{\theta}_n$.

blackEach sub-input $\boldsymbol{s}_n$ is further multiplied element-wise with a mask $\boldsymbol{b}_n \in \mathbb{R}^L$ to produce the masked sub-input $\boldsymbol{m}_n = \boldsymbol{s}_n \circ \boldsymbol{b}_n$, which further increases the diversity. In our default strategy, the masks $\boldsymbol{b}_n$ are initialized as the following,

$$\boldsymbol{b}_n = \frac{1}{1-\tau} \max(0,\ U(0,1)^L - \tau), \tag{5.2}$$

where $U(\cdot,\cdot)^L$ is a uniformly sampled vector of length L, $\max(0,\cdot)$ and τ are used to control the number of additional zeros added to $\boldsymbol{b}_n$ (the default value for τ in our experiments is 0.1), and $\frac{1}{1-\tau}$ normalizes the $\max(\cdot)$ function's output to the range $[0,1]$.

blackb) Parallel transformation: This step captures the distinctness among subspaces. As the number of sampled sub-inputs N grows to a large number, the parallel process $g(\cdot)$ independently applied to each masked sub-input $\boldsymbol{m}_n$ should be as simple as possible; otherwise, the memory and computation cost quickly become unaffordable. Therefore, we choose process $g(\cdot)$ as a linear transformation, sharing the learnable transformation weights for all sub-inputs.

c) Ensemble all subspaces' outcome: The transformation outcome of all sub-inputs can be ensembled by either averaging or concatenating. Assume the embedding $\boldsymbol{v}$ after ensemble is fixed to a short length D_{emb} for an efficient retrieval process; the dimension of each sub-input's transformation outcome will then be D_{emb} and $\frac{D_{emb}}{N}$ for the averaging and concatenating strategy, respectively. Apparently, concatenating requires a much lower memory and computation cost than averaging. However, it is not feasible to simply adopt concatenating. Since the embedding's dimension D_{emb} is usually limited to a small number, $\frac{D_{emb}}{N}$ goes to a small number or even less than 1 when N goes to a large number. A reasonable way to reduce memory and computation cost will be dividing the N sub-inputs into $\sqrt{N}$ groups, averaging inside every group, and concatenating between groups.

When the lower layers $f(\cdot)$ are complex enough, e.g., ResNet50, we assume that $f(\boldsymbol{x})$ preserves almost all the semantic information of $\boldsymbol{x}$. Then building the ensemble process upon $f(\boldsymbol{x})$ is nearly equivalent to building it upon $\boldsymbol{x}$ while significantly mitigating the computation and memory burden. While building the group ensemble block upon $f(\boldsymbol{x})$, the backward-propagation propagates the "diversity" between sub-inputs backward so that the lower layers will be updated with diverse gradients. Formally, the lower layers' parameters W_f are updated with diverse gradients as the following,

$$W_f := W_f + \sum_n \frac{\partial L}{\partial \boldsymbol{m}_n} \frac{\partial \boldsymbol{m}_n}{\partial W_f}, \tag{5.3}$$

where both $\frac{\partial L}{\partial \boldsymbol{m}_n}$ and $\frac{\partial \boldsymbol{m}_n}{\partial W_f}$ are different among various subspaces because the masked sub-inputs $\boldsymbol{m}_n$ are different from each other.

5.3.2 Combining group ensemble block with class-level and instance-level discrimination

Due to the low cost of the proposed group ensemble block as analyzed in Section 5.4.2, it is possible to combine all three approaches, including class-level discrimination, instance-level discrimination, and ensemble learning. How the three approaches are connected is shown in Figure 5.1: the SSL method SimSiam (the instance-level discrimination) adds a siamese architecture to the model for stage I; the proposed group ensemble block (the ensemble learning approach) is placed at the top of the projector; the SNCA (the class-level discrimination) classifies images based on the group ensemble block's output. This combination requires three steps for the coarse-to-fine retrieval problem, as shown in Figure 5.1. The first step trains the backbone, projector, and predictor with the siamese architecture. The second step generates an embedding vector for every training image. The third step retrieves relevant images for a given query by comparing embeddings' similarities. Note that

the class-level and instance-level methods (SimSiam and SNCA) are only used in the first step.

For utilizing the class-level discrimination, we adopt the Scalable Neighborhood Component Analysis (SNCA) method [101]. Suppose we are given a dataset of n examples $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ with corresponding coarse-level labels $cy_1, cy_2, \cdots, cy_n$. Example $\boldsymbol{x}_i$ selects example $\boldsymbol{x}_j$ as its neighbor with probability p_{ij}, based on their cosine similarity c_{ij} in the embedding space,

$$p_{ij} = \frac{\exp(c_{ij}/\sigma)}{\sum_{k \neq i} \exp(c_{ik}/\sigma)}, \quad p_{ii} = 0, \tag{5.4}$$

where $p_{ii} = 0$ indicates that each example cannot select itself as its neighbor. The parameter σ, set as 0.05 in practice [101], is used to scale the cosine similarity. Let $\Omega_i = \{j | cy_j = cy_i\}$ denote the indices of training examples that share the same coarse-level label with $\boldsymbol{x}_i$, and define p_i as the probability of $\boldsymbol{x}_i$ being correctly predicted. The objective is to minimize the expected negative log-likelihood over the dataset,

$$p_i = \sum_{j \in \Omega_i} p_{ij}, \quad \mathcal{L}_{class} = -\frac{1}{n} \sum_{i=1}^{n} \log(p_i). \tag{5.5}$$

For utilizing instance-level discrimination, we adopt the SimSiam method [105]. The SimSiam method first randomly augments an image $\boldsymbol{x}$ to T views $\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^T$. Then each augmented view is processed with the same backbone, projector, and predictor, where the projector and predictor output $\boldsymbol{z}_i$ and $\boldsymbol{u}_i$, respectively. SimSiam maximizes the cosine similarity between every view pair ($\boldsymbol{x}_i$, $\boldsymbol{x}_j$) through minimizing the loss term $\mathcal{L}_{ij}$,

$$\mathcal{L}_{ij} = -\left(\frac{\boldsymbol{u}_i}{\|\boldsymbol{u}_i\|} \frac{\boldsymbol{z}_j}{\|\boldsymbol{z}_j\|} + \frac{\boldsymbol{u}_j}{\|\boldsymbol{u}_j\|} \frac{\boldsymbol{z}_i}{\|\boldsymbol{z}_i\|}\right). \tag{5.6}$$

Note that the gradients of $\mathcal{L}_{ij}$ w.r.t. $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ are set to 0, and named as the stop gradient [102]. Summing $\mathcal{L}_{ij}$ over every view pair gives the instance-level discrimination loss $\mathcal{L}_{instance}$,

$$\mathcal{L}_{instance} = \frac{1}{T(T-1)} \sum_{1 \leq i \neq j \leq T} \mathcal{L}_{ij}. \tag{5.7}$$

The total loss is a sum over the class-level loss and the instance-level loss,

$$\mathcal{L} = \mathcal{L}_{class} + \mathcal{L}_{instance}. \tag{5.8}$$

5.4 Performance and cost analysis

Denote $\boldsymbol{r}_n = [r_n^1, r_n^2, \cdots, r_n^K] \in \mathbb{R}^K$ as a query's largest relevance score to K fine-grained categories output by the n-th sub-input, where the superscript represents an element of the vector $\boldsymbol{r}_n$. We further normalize it ($||\boldsymbol{r}_n|| = 1$) so that it represents the probability of which category the current retrieval will be from. Let $\boldsymbol{r} \in \mathbb{R}^K$ be the average of the N sub-inputs' prediction: $\boldsymbol{r} = \frac{1}{N} \sum \boldsymbol{r}_n$. Let $E_y(\boldsymbol{r})$ further be the mean of $\boldsymbol{r}$ conditioned on fine-grained class y, which means $E_y(\boldsymbol{r})$ is the mean of all $\boldsymbol{r}$ generated by images from fine-grained class y.

Theorem 1 *Denote the query and retrieval image's fine-grained category as Y and $\hat{Y}$, respectively. All the relevance vectors $\boldsymbol{r}$ that correctly retrieve a y-th fine-grained class image form a set $S_y = \{\boldsymbol{r} \colon \arg\max_d r^d = y\}$. Let ϕ_y be the Euclidean distance from $E_y(\boldsymbol{r})$ to ∂S_y, the boundary of S_y. The upper error bound that a retrieval image is not from the same fine-grained category as the query is given by the following,*

$$Pr(\hat{Y} \neq y | Y = y) \leq \frac{\gamma_y + \eta_y / N}{\phi_y^2} = \frac{2(\gamma_y + \eta_y/N)(K-1)^2}{(K E_y(\boldsymbol{r}) - 1)^2}, \tag{5.9}$$

where

$$\eta_y = \frac{1}{N}\sum_{d=1}^{K}\sum_{n=1}^{N} Var(r_n^d|Y=y) \tag{5.10}$$

is the variance of a single sub-input's relevance score prediction, and

$$\gamma_y = \frac{1}{N^2}\sum_{d=1}^{K}\sum_{n\neq m} Cov(r_n^d, r_m^d|Y=y). \tag{5.11}$$

is the covariance between different sub-inputs' relevance score predictions.

This proof follows the same idea as in the proof of Section 7.2 of [121]. The fact that $||\boldsymbol{r} - E_y(\boldsymbol{r})|| < \phi_y$ leads to $\arg\max_d r^d = y$ implies that $P(||\boldsymbol{r} - E_y(\boldsymbol{r})|| \geq \phi_y|Y=y)$ gives the upper error bound,

$$\Pr(\hat{Y} \neq y|Y=y) \leq \Pr(||\boldsymbol{r} - E_y(\boldsymbol{r})|| \geq \phi_y \ |Y=y) \tag{5.12}$$

Using Chebyshev's inequality, we have

$$\Pr(||\boldsymbol{r} - E_y(\boldsymbol{r})|| \geq \phi_y \ |Y=y) \leq \frac{1}{\phi_y^2} Var(\boldsymbol{r}|Y=y), \tag{5.13}$$

and $Var(\boldsymbol{r}|Y=y)$ can be computed by the variance-covariance matrix as the following,

$$\begin{aligned} Var(\boldsymbol{r}|Y=y) &= Var(\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{r}_n|Y=y) \\ &= \frac{1}{N^2}\sum_{d=1}^{K}[\sum_{n=1}^{N} Var(\boldsymbol{r}_n^d|Y=y) \\ &\quad + \sum_{n\neq m} Cov(r_n^d, r_m^d|Y=y)] \end{aligned} \tag{5.14}$$

Combining Eqs. 5.10, 5.11, 5.13, 5.14 gives

$$\Pr(\hat{Y} \neq y | Y = y) \leq \frac{\gamma_y + \eta_y/N}{\phi_y^2}. \tag{5.15}$$

$\frac{\gamma_y+\eta_y/N}{\phi_y^2} = \frac{2(\gamma_y+\eta_y/N)(K-1)^2}{(KE_y(\boldsymbol{r})-1)^2}$ uses the same rules as in the proof of Section 7.2 of [121].

5.4.1 Performance analysis

The error upper bound shown in Theorem 1 is positively related to η_y/N and negatively related to the covariance γ_y. We assume that when N goes to a large number, η_y/N should be small. In this subsection, we evaluate the effect of two factors, N and γ_y.

Figure 5.2 shows the effect of embedding size N using dataset CIFAR-100. As shown in the figure, the accuracy on two tasks that are detailed in Section 5.5.1.1, increases as the ensemble size N increases and saturates around $N = 1024$.

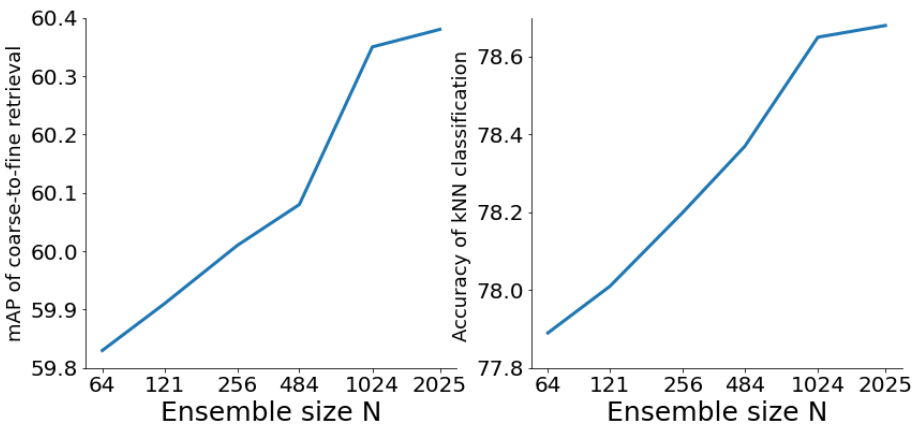


Figure 5.2. Performance analysis on the ensemble size N using dataset CIFAR-100.

For the covariance, we compare two models that use and do not use the group ensemble block, named ensemble model and baseline model. Note that even if there is no ensemble block in the baseline model, the embedding can still be cut into several parts so that we can analyze the covariance between these parts. An insignificant difference between the two

models' evaluated covariance will indicate that the ensemble block is not working. Otherwise, the ensemble model's covariance being significantly lower than the baseline model indicates a good ensemble block. We use the following setting to have a fair comparison between the two models: i) for the ensemble model, the group ensemble block cuts its input into several (here, we use four for convenient comparison) evenly distributed and non-overlapping sub-inputs without using inside-group averaging; ii) for the baseline model, its embedding is also cut into four parts for computing covariance. As shown in Table 5.2, the ensemble model produces a consistently lower correlation than the baseline model on fine-grained classes.

Table 5.2. The covariance comparison between the baseline and ensemble model. The covariance is computed on the first 9 fine-grained classes of CIFAR-100.

Method	$y=1$	$y=2$	$y=3$	$y=4$	$y=5$	$y=6$	$y=7$	$y=8$	$y=9$
Baseline	0.244	0.077	0.222	0.210	0.114	0.167	0.215	0.261	0.150
Ensemble	0.223	0.072	0.161	0.159	0.804	0.159	0.181	0.149	0.118

5.4.2 Cost analysis

With choosing $g(\cdot)$ as a linear transformation, the group ensemble block needs far fewer parameters and computations than the lower layers, making it low-complexity. We show this by setting N, D_{in}, D_{ens} to be large enough, e.g., 1024, 4096, and 1024. Note that the dimension of a sub-input and its output are $L = \frac{D_{in}}{\sqrt{N}}$ and $\frac{D_{ens}}{\sqrt{N}}$.

In terms of parameters, the group ensemble block contains $N \times L$ parameters for storing the indexing vectors $\boldsymbol{\theta}_n$, $N \times L$ parameters for storing the masks $\boldsymbol{b}_n$, and $(L+1) \times \frac{D_{ens}}{\sqrt{N}}$ parameters for storing the transformation weights. Then the total parameter amount is $1024 \times \frac{4096}{\sqrt{1024}} + 1024 \times \frac{4096}{\sqrt{1024}} + (\frac{4096}{\sqrt{1024}} + 1) \times \frac{1024}{\sqrt{1024}} \approx 0.27M$, far less than the lower layers' parameter amount, e.g., the $f(\cdot)$'s $\sim 30M$ parameters.

In terms of computation, the group ensemble block requires $N \times L$ FLOPs for masking the sub-inputs and $N \times 2 \times L \times \frac{D_{ens}}{\sqrt{N}}$ FLOPs for the linear transformation. The total computation

amount is $1024\times\frac{4096}{\sqrt{1024}} + 1024\times2\times\frac{4096}{\sqrt{1024}}\times\frac{1024}{\sqrt{1024}} \approx 8.52M$ FLOPs, far less than the lower layers' computation amount, e.g., the ResNet50's $3.87G$ FLOPs.

In comparison, the traditional ensemble approaches usually need to ensemble N entire networks for N ensembles, which needs $N\times30M$ parameters and $N\times3.87G$ FLOPs. These numbers quickly grow beyond control when the ensemble size N goes to a large number.

5.5 Experiment

5.5.1 Experiment settings

5.5.1.1 Evaluation tasks

Coarse-to-fine image retrieval finds the top relevant fine-grained images for a given query using Eq. 5.1. We report the mean average precision (mAP), which averages the average precision (AP) over all queries. The AP for a single query is computed as the following [122],

$$AP = \frac{\sum_{k=1}^{n} P(k)\cdot rel(k)}{R}, \tag{5.16}$$

where R denotes the number of relevant images for the query, n denotes the number of images retrieved by the model, $P(k)$ denotes the precision of top-k retrieval results, and $rel(k)$ is a binary indicator function equal to 1 when the kth retrieved result is relevant to the query and 0 otherwise.

On-the-fly kNN classification assumes that coarse-level labels are available during training, and fine-grained labels are available as ground truth during testing. This task evaluates whether the model has really learned fine-grained categories from coarse-level labels. For a test image $\boldsymbol{x}$, denote its k nearest database images and corresponding fine-grained labels as $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k$ and $y_1, \cdots, y_k$ respectively. The probability that $\boldsymbol{x}$ is classified into

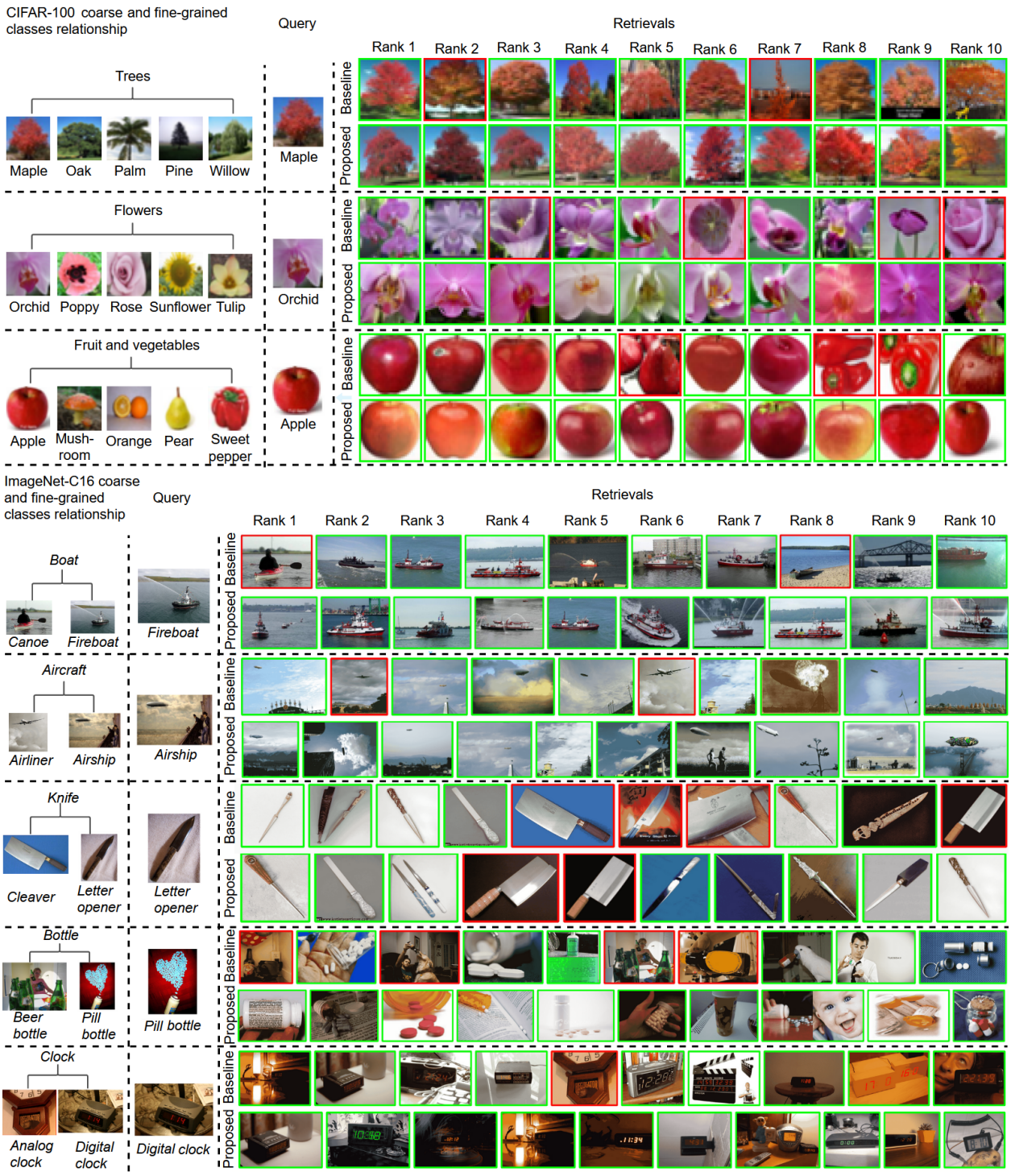


Figure 5.3. The top-10 retrievals on datasets CIFAR-100 (upper) and ImageNet-C16 (lower), using (proposed) and not using (baseline) the proposed group ensemble block. For this coarse-to-fine retrieval task, while the model is trained with only coarse-level annotations available, the retrievals counted as correct must be from the same fine-grained category as the query. The right side of the figure shows the retrievals, where Green and red boxes mark the correct and incorrect retrievals.

fine-grained category y is computed as the following,

$$p(y_x = y) \propto \sum_{i=1}^{k} 1(y_i = y) \exp(\frac{1}{\sigma} \frac{\boldsymbol{v}^T \boldsymbol{v}_i}{\boldsymbol{v}\boldsymbol{v}_i}), \tag{5.17}$$

where the indicator function $1(\cdot)$ takes 1 when the i-th nearest database image falls in the y-th fine-grained category otherwise takes 0. The probability $p(y_x{=}y)$ is normalized so that $\sum_y p(y_x{=}y){=}1$. The hyper-parameter k is set to 20, which is cross-validated from {10, 15, 20, 25, 30}. The top-1 accuracy is reported for the on-the-fly kNN classification.

5.5.1.2 Datasets

The CIFAR-100 dataset [123] contains 20 coarse classes and each coarse class has 5 fine-grained classes. For example, the coarse class *trees* has the fine-grained classes *maple trees*, *oak trees*, *palm trees*, *pine trees*, and *willow trees*.

For the ImageNet dataset [124], we use the Robustness library [125] that subsumes fine-grained classes into coarse classes. We use its *geirhos_16* that provides 16 coarse classes, with each containing 2 fine-grained classes. For example, the coarse class *boat* consists of fine-grained classes *canoe* and *fireboat*. For convenience, we name this subset ImageNet-C16.

5.5.1.3 Training details

Our models use the ResNet-50 [126] as the backbone. The projector contains three 2048, 2048, and 256 length linear layers, and the predictor contains two 2048 and 256 length linear layers. The models are trained with the SGD optimizer and cosine learning rate for 600 epochs. The initial learning rate is 0.1, with a 5-epoch warmup [127]. The weight decay is 5e-4 for CIFAR-100 and 1e-4 for ImageNet-16. The batch size is 128 for CIFAR-100 and 64 for ImageNet-C16. We use a single NVIDIA P100 GPU with 16G memory.

Table 5.3. Comparison with the state-of-the-art on CIFAR-100, ImageNet-C16, and ImageNet-1K. The top-1 accuracy (%) is reported for the kNN classification, and the mAP (%) is reported for the coarse-to-fine image retrieval.

Method	Class discrim.	Instance discrim.	Ensemble	#Params	FLOPs	CIFAR-100		ImageNet-C16		ImageNet-1K	
						kNN	mAP	kNN	mAP	kNN	mAP
SNCA [101]	✓	-	-	24.0M	3.8669G	72.2	35.9	72.6	43.2	55.4	31.8
ClusterFit [130]	✓	-	-	54.2M	3.9273G	72.5	23.0	73.3	32.4	59.5	12.7
Grafit [110]	✓	✓	-	32.9M	3.8848G	77.7	55.7	80.8	65.6	69.1	42.9
Basline	✓	✓	-	32.4M	3.8837G	77.7	55.7	80.9	65.8	69.1	43.0
Baseline w. group ensemble	✓	✓	✓	32.0M	3.8840G	**78.7**	**60.4**	**83.2**	**69.2**	**71.9**	**45.3**

The data augmentation consists of AutoAugment [128], random crop, and random erasing [129]. Due to the large amount of GPU memory required by SimSiam, we use four augmented views T=4 for CIFAR-100 and two augmented views T=2 for ImageNet-C16.

5.5.2 Experiment results and comparisons

In the following, we list all methods that we evaluate. All the methods below adopt the ResNet-50 architecture for a fair comparison.

SNCA [101] solely uses the class-level discrimination as explained in Section 5.3.2.

Grafit [110] uses both class-level and instance-level discrimination. The instance-level discrimination is achieved through the self-supervised method BYOL [104].

ClusterFit [130] first uses the k-means algorithm to cluster the training images' feature vectors that are output by a pretrained model. It then trains a model from scratch, with the k-means' cluster assignments as pseudo-labels. It's shown [130] that a model trained with these pseudo labels produces a more transferable feature representation than models trained with human-annotated labels.

Our baseline uses class-level and instance-level discrimination, similar to Grafit. The difference is that Grafit uses BYOL for SSL while our baseline uses SimSiam. Using SimSiam

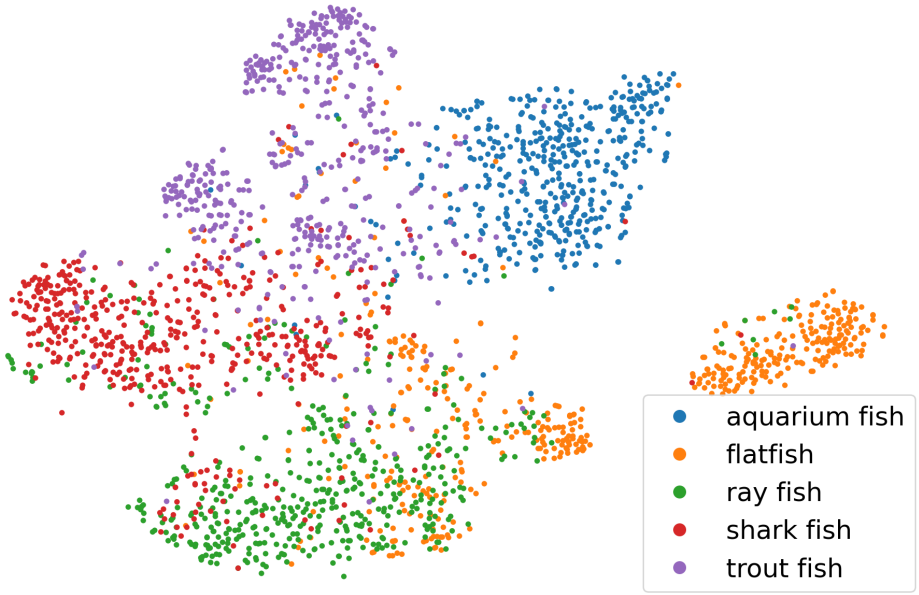


Figure 5.4. t-SNE representations of embeddings from the CIFAR-100's *fish* coarse category. The *fish* category consists of five fine-grained categories — *aquarium fish*, *flatfish*, *ray fish*, *shark fish*, and *trout fish*.

achieves comparable accuracy to BYOL while requiring significantly less GPU memory. For example, Grafit and our baseline require ∼35G and ∼15G GPU memory, respectively, using the CIFAR-100 dataset with a batch size of 128 and four augmented views (T=4). We deem our baseline to be a more feasible version of Grafit.

The proposed ensemble model utilizes the proposed group ensemble block, class-level discrimination, and instance-level discrimination. It replaces the last layer of the baseline projector with a group ensemble block.

As shown in Table 5.3, the ensemble model achieves the best performance for every evaluation task. For the kNN classification, the group ensemble block pushes the best accuracy from 77.7% to 78.7% on CIFAR-100, from 80.9% to 83.2% on ImageNet-C16, and from 69.1% to 71.9% on ImageNet-1K. For the coarse-to-fine image retrieval task, the group ensemble block pushes the best accuracy from 55.7% to 60.4% on CIFAR-100, from 65.8% to 69.2% on ImageNet-C16, and from 43.0% to 45.3% on ImageNet-1K.

Figure 5.3 shows the top-10 retrieval images on CIFAR-100 and ImageNet-C16 when using (proposed) and not using (baseline) the ensemble block. It can be seen that using the ensemble block produces consistently better retrieval results. Figure 5.4 presents the t-SNE visualization for CIFAR-100's coarse category *fish* using the group ensemble block.

5.5.3 More analyses and ablation studies

black **Cost of the group ensemble block** Table 5.4 lists the memory and FLOPs cost of the proposed group ensemble block with varying ensemble sizes. It can be seen that the ensemble block consumes 524.32K parameters and 1.31M FLOPs even with a large ensemble size of 16384, which is negligible compared to the baseline model's 32.40M parameters and 3.88G FLOPs. In terms of inference speed, using an NVIDIA RTX A5000 graphic card, the baseline model takes 0.0688s for a batch size of 64, and the model with ensemble size of 1024 and 16384 take 0.0694s and 0.0699s respectively, showing that using the group ensemble block adds almost no latency.

black

Table 5.4. Cost of the group ensemble block using varying numbers of ensemble size N, with $D_{in} = 2048$ and $D_{emb} = 256$.

Cost	64	256	484	1024	4096	16384	baseline
#Params	40.99K	67.60K	91.21K	131.59K	262.28K	524.32K	32.40M
FLOPs	1.06M	1.08M	1.09M	1.11M	1.18M	1.31M	3.88G

black **Effect of masking strategy** This ablation study investigates different masking strategies, in addition to the default strategy as in Eq.5.2. The "no masking" strategy removes the masking step, which is equivalent to using a mask with all elements being 1. This experiment also evaluates the performance of generating each element of the mask using the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ or the uniform distribution $U(a, b)$. For the parameter τ that sets mask elements smaller than it to zero, we evaluate with $\tau = 0$, $\tau = 0.1$, $\tau = 0.25$.

It can be seen from Table 5.5 that using either Gaussian or uniform masking is better than no masking. For the parameter τ, it is better to set it larger than 0, e.g., 0.1 or 0.25. The Gaussian and uniform masking strategies produce about the same accuracy.

Table 5.5. Ablation study on masking strategy using ImageNet-C16.

No masking		Threshold	$\mathcal{N}(0.5, 0.25^2)$		$U(0,1)$	
kNN	mAP		kNN	mAP	kNN	mAP
82.38	68.34	$\tau = 0$	82.69	68.95	83.15	69.02
		$\tau = 0.1$	82.56	69.27	83.19	69.24
		$\tau = 0.25$	83.19	69.34	82.69	68.95

black **Effect of sampling strategy** This ablation study investigates different sampling strategies, in addition to the "random sampling" as in Figure 5.1. The "no sampling" strategy does not apply any sampling strategy, and each sub-input $\boldsymbol{s}_n$ is the same as the input $\boldsymbol{t}$, which can also be understood as "complete sampling" that samples the entire input. The "ordered sampling" strategy samples each sub-input sequentially, i.e., the first draws the first L elements, the second draws the next L elements, and so forth. When all elements are exhausted, samples will be drawn from the beginning again. As shown in Table 5.6, the "no sampling" strategy does not produce an apparent accuracy increase compared to the baseline. The "ordered sampling" strategy is much worse than the baseline, because these ensemble groups significantly overlap due to the sequential sampling. With this overlap, the outputs of $\sqrt{N}$ groups are not diverse from each other, and the embedding of length D_{emb} is composed of $\sqrt{N}$ highly correlated sub-embeddings of a short length $\frac{D_{emb}}{\sqrt{N}}$. Consequently, the "random sampling" strategy outperforms the others by a large margin.

Effect of various transformation processes This ablation study investigates different transformation processes, in addition to the default linear transformation. As shown in Table 5.7, using a three-layer MLP with 256 hidden neurons is slightly worse than the default linear transformation while utilizing more parameters and computations. The potential

Table 5.6. Ablation study on sampling strategy, where the default strategy uses random sampling as in Figure 5.1.

Sampling strategy	CIFAR-100		ImageNet-C16	
	kNN	mAP	kNN	mAP
Baseline	77.72	55.70	80.88	65.84
No sampling	77.95	56.06	80.83	65.99
Random sampling	78.65	60.35	83.19	69.24
Ordered sampling	69.98	46.87	73.56	53.56

reason may be that the MLP overfits the coarse-level annotation and cannot generalize well to the fine-level image retrieval.

Table 5.7. Ablation study on transformation process. The MLP has three layers with 256 hidden neurons.

Sampling strategy	CIFAR-100		ImageNet-C16	
	kNN	mAP	kNN	mAP
Linear	78.65	60.35	83.19	69.24
MLP	78.53	60.08	82.94	68.70

Effect of embedding vector length Figure 5.5 shows the effect of embedding vector length using dataset CIFAR-100. The accuracy increases as the embedding vector length increases and is finally saturated around the length of 1024. Note that a small embedding length is favorable in practice for smaller storage size and faster retrieval speed.

Same or distinct transformation weights This ablation study investigates whether using distinct transformation weights for various sub-inputs would improve the model's performance. As the evaluated result shown in Table 5.8, using distinct weights for distinct sub-inputs does not produce higher accuracy. This may be because distinct weights are overfitted to the coarse-level annotations and cannot generalize well to the fine-grained categories.

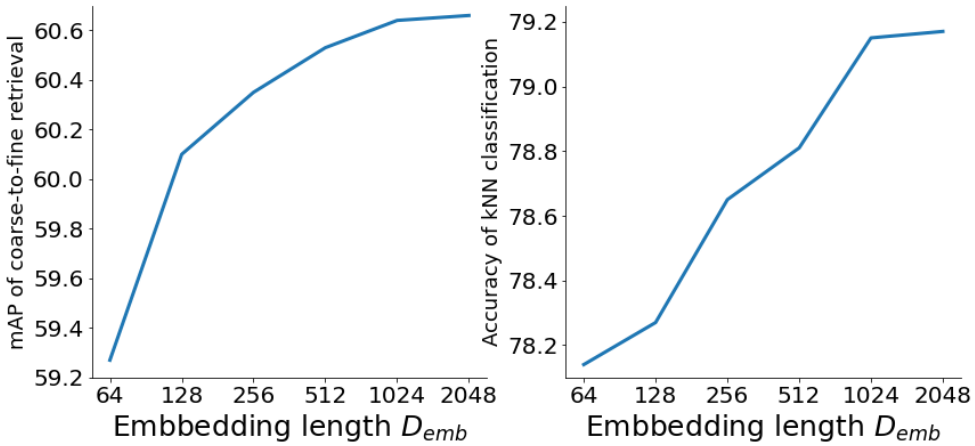


Figure 5.5. Effect of embedding length on the coarse-to-fine retrieval and kNN classification using CIFAR-100.

Table 5.8. Ablation study on using same or distinct transformation weights for various sub-inputs.

Same transformation	CIFAR-100		ImageNet-C16	
	kNN	mAP	kNN	mAP
Yes	78.65	60.35	83.19	69.24
No	77.69	58.50	82.13	68.26

Effect of number of group ensemble blocks In Figure 5.1, the last linear layer of the baseline projector is removed and the group ensemble block is added. To check if more ensemble blocks will produce higher accuracy, we remove all the baseline projector's three linear layers, and add three ensemble blocks. As shown in Table 5.9, using three ensemble blocks does not improve the accuracy.

Table 5.9. Effect of the number of group ensemble blocks.

Number of ensemble blocks	CIFAR-100		ImageNet-C16	
	kNN	mAP	kNN	mAP
1	78.65	60.35	83.19	69.24
3	77.08	57.01	81.88	68.33

Complementariness analysis To measure complementariness between sub-inputs, we observe the model's performance when we ensemble the outcome of various sub-inputs. To compare the ensemble model with the baseline model, we use the same setting (use four sub-inputs in total) as in the covariance measurement of Section 5.4.1. As shown in Table 5.10, when solely using one sub-input's outcome, the ensemble model produces a lower accuracy than the baseline. This is because each sub-input's outcome only sees part of the ensemble block's input; but its counterpart of the baseline sees the entire input. When combining two or more sub-inputs' outcomes, the ensemble model produces a higher accuracy than the baseline. Noticeably, when combining all four sub-inputs' outcomes, the ensemble model produces an mAP of 0.5961, which is much higher than the baseline's 0.5503.

Table 5.10. Mean average precision (mAP) on CIFAR-100, when leveraging the transformation outcomes of 1, 2, 3, and 4 sub-inputs. 4 sub-inputs are used in total.

Sub-input 1	Sub-input 2	Sub-input 3	Sub-input 4	Baseline	Ensemble
✓	-	-	-	0.5329	0.5192
-	✓	-	-	0.5303	0.5162
-	-	✓	-	0.5181	0.5077
-	-	-	✓	0.5213	0.5199
✓	✓	-	-	0.5473	0.5677
-	-	✓	✓	0.5355	0.5628
✓	✓	✓	-	0.5486	0.5847
-	✓	✓	✓	0.5453	0.5822
✓	✓	✓	✓	0.5503	0.5961

5.5.4 Group ensemble block in other tasks

This experiment shows that it is easy to deploy the proposed group ensemble block in other tasks and improve the model's performance. We provide an example of CIFAR-10 classification using ResNet50 [126]. The usage of the proposed group ensemble block is effortless: we replace the last linear layer of ResNet50 with a group ensemble block that has the same input and output dimension as the replaced linear layer. This ensemble block

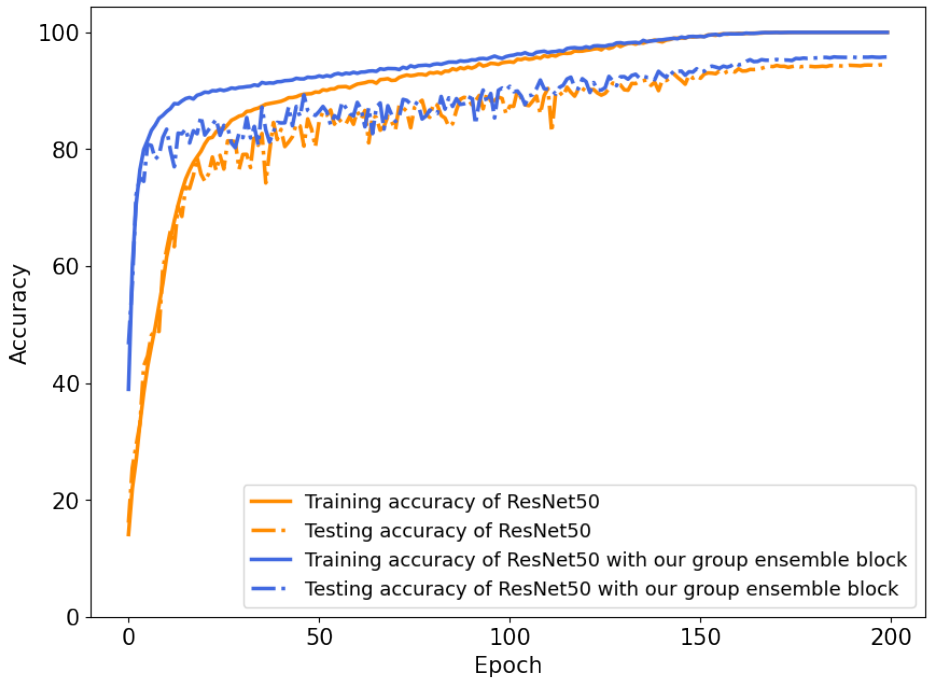


Figure 5.6. CIFAR-10 training and testing accuracy by ResNet50 and its variant where its last linear layer is replaced with a group ensemble block.

contains 10 groups as CIFAR-10 contains 10 classes. Each group has 10 subspaces, consistent with our other experiments where the number of groups is equal to the number of subspaces in each group. This modification results in a negligible increase to the number of parameters and computations compared to the network's other parts, as analyzed previously in Section IV-B and V-C. The training time is also not increased after the modification: using one NVIDIA P100 graphics card, both models take 4.5 hours for 200 epochs with a batch size of 256. The accuracy on the test set is noticeably increased from 94.82% to 95.76%, averaged over 5 runs. The model also converges significantly faster in the first 50 epochs, as shown in Figure 5.6.

5.6 Conclusion

We study the special task of coarse-to-fine image retrieval, where the retrieval images are encouraged to be from a category that is finer than the annotations. We propose a low-complexity group ensemble block that leverages numerous subspaces of the block's input, and make it possible to combine ensemble learning with other approaches such as class-level discrimination and instance-level discrimination. Quantitative evaluation proves that various subspaces' process outcomes are low-correlated and complementary to each other. State-of-the-art accuracy on CIFAR-100, ImageNet-C16, and ImageNet-1K shows the effectiveness of the proposed group ensemble block. Future applications of the proposed method could be extending the proposed group ensemble block to other tasks.

Chapter 6: Conclusion and future directions

6.1 Conclusion

- In the first chapter, I start with the limitation of the nowadays popular CNNs, serving as a background of capsule networks and GLOM. Then I introduce how the general ideas of capsule networks and GLOM potentially solve the problems of CNNs, followed by the weaknesses of current capsule networks and GLOM. The first chapter ends with a summary of the contributions of this dissertation.

- The second chapter gives a brief introduction to capsule networks and GLOM, which details the general ideas introduced in the first chapter.

- The third chapter illustrates the proposed non-iterative cluster routing for capsule networks, which accelerates capsule networks and facilitates the usage of segmentation on large-size images.

- The fourth chapter illustrates the proposed Twin-Islands model, which achieves many desired properties of GLOM such as equivariance, model interpretability, and adversarial robustness for point clouds.

- The fifth chapter illustrates the proposed group ensemble block, which extends the usage of capsule networks and GLOM. It is used with the CNNs-based self-supervised learning method, and applied to tasks such as image classification and retrieval.

- The last chapter concludes this dissertation and also gives future directions.

6.2 Future directions

Capsule networks and GLOM are still quickly developing at this time. There are many promising future directions as the following:

- For part-whole hierarchy, once what the intermediate layers have learned become clear, it may be possible to have further applications, e.g., prune and assemble models. Noticeably, it may be possible to do all basic vision tasks simultaneously using the same part-whole hierarchy, including classification, detection, and segmentation, even with capturing the rich attributes of every object part such as color and shape.

- For model interpretability, it should be possible to quantitatively measure to what extent the model is interpretable. Generally, this can be measured by comparing the transformation weight and embedding islands with the segmentation ground truth.

- For the adversarial attacks, it would be interesting to theoretically prove how much the high-dimensional coincidence filtering technique improves compared to other types of neural networks, e.g., CNNs.

- In terms of rapid GLOM development in the community, it will be very helpful to explore an efficient propagation strategy and also find a robust baseline GLOM model that achieves a decent accuracy on the well-recognized benchmark dataset, e.g., ImageNet.

References

[1] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International conference on learning representations*, 2018.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[3] Michael A Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4845–4854, 2019.

[4] Geoffrey Hinton. How to represent part-whole hierarchies in a neural network. *arXiv preprint arXiv:2102.12627*, 2021.

[5] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *NIPS*, 2017.

[6] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International conference on learning representations*, 2018.

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

[8] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

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

[14] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

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

[16] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

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

[18] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. Understanding deep networks via extremal perturbations and smooth masks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2950–2958, 2019.

[19] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *In Workshop at International Conference on Learning Representations*. Citeseer, 2014.

[20] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018.

[21] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " 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*, pages 1135–1144, 2016.

[22] Daniel Kahneman, Anne Treisman, and Brian J Gibbs. The reviewing of object files: Object-specific integration of information. *Cognitive psychology*, 24(2):175–219, 1992.

[23] Geoffrey Hinton. Some demonstrations of the effects of structural descriptions in mental imagery. *Cognitive Science*, 3(3):231–250, 1979.

[24] Geoffrey Hinton. Taking inverse graphics seriously, 2013.

[25] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[26] Geoffrey Hinton, Alex Krizhevsky, Navdeep Jaitly, Tijmen Tieleman, and Yichuan Tang. Does the brain do inverse graphics. In *Brain and Cognitive Sciences Fall Colloquium*, volume 2, 2012.

[27] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[28] Karim Ahmed and Lorenzo Torresani. Star-caps: Capsule networks with straight-through attentive routing. *Advances in neural information processing systems*, 32:9101–9110, 2019.

[29] Zhihao Zhao and Samuel Cheng. Capsule networks with non-iterative cluster routing. *Neural Networks*, 143:690–697, 2021.

[30] Yongheng Zhao, Tolga Birdal, Jan Eric Lenssen, Emanuele Menegatti, Leonidas Guibas, and Federico Tombari. Quaternion equivariant capsule networks for 3d point clouds. In *European Conference on Computer Vision*, pages 1–19. Springer, 2020.

[31] Weiwei Sun, Andrea Tagliasacchi, Boyang Deng, Sara Sabour, Soroosh Yazdani, Geoffrey Hinton, and Kwang Moo Yi. Canonical capsules: Unsupervised capsules in canonical pose. *arXiv preprint arXiv:2012.04718*, 2020.

[32] Jindong Gu, Baoyuan Wu, and Volker Tresp. Effective and efficient vote attack on capsule networks. In *International Conference on Learning Representations*, 2021.

[33] Jan Eric Lenssen, Matthias Fey, and Pascal Libuschewski. Group equivariant capsule networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8858–8867, 2018.

[34] Sai Raam Venkataraman, S. Balasubramanian, and R. Raghunatha Sarma. Building deep equivariant capsule networks. In *International Conference on Learning Representations*, 2020.

[35] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[36] Nicola Garau, Niccolò Bisagno, Zeno Sambugaro, and Nicola Conci. Interpretable part-whole hierarchies and conceptual-semantic relationships in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13689–13698, 2022.

[37] Congcong Zhu, Xintong Wan, Shaorong Xie, Xiaoqiang Li, and Yinzheng Gu. Occlusion-robust face alignment using a viewpoint-invariant hierarchical network architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11112–11121, 2022.

[38] Logan Engstrom, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 1(2):3, 2017.

[39] Michael A Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses

of familiar objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4845–4854, 2019.

[40] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[41] Yao-Hung Hubert Tsai, Nitish Srivastava, Hanlin Goh, and Ruslan Salakhutdinov. Capsules with inverted dot-product attention routing. *arXiv preprint arXiv:2002.04764*, 2020.

[42] Karim Ahmed, Lorenzo Torresani, and Advances. Star-caps: Capsule networks with straight-through attentive routing. In *in Neural Information Processing Systems*, pages 9101–9110, 2019.

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

[44] Zhenhua Chen and David Crandall. Generalized capsule networks with trainable routing procedure. *arXiv preprint arXiv:1808.08692*, 2018.

[45] Suofei Zhang, Quan Zhou, and Xiaofu Wu. Fast dynamic routing based on weighted kernel density estimation. In *International Symposium on Artificial Intelligence and Robotics*, pages 301–309. Springer, 2018.

[46] Fabio De Sousa Ribeiro, Georgios Leontidis, and Stefanos D Kollias. Capsule routing via variational bayes. In *AAAI*, pages 3749–3756, 2020.

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

[48] Jan Eric Lenssen, Matthias Fey, and Pascal Libuschewski. Group equivariant capsule networks. In *Advances in Neural Information Processing Systems*, pages 8844–8853, 2018.

[49] Adrien Deliege, Anthony Cioppa, and Marc Van Droogenbroeck. Hitnet: a neural network with capsules embedded in a hit-or-miss layer, extended with hybrid data augmentation and ghost capsules. *arXiv preprint arXiv:1806.06519*, 2018.

[50] Ayush Jaiswal, Wael AbdAlmageed, Yue Wu, and Premkumar Natarajan. Capsulegan: Generative adversarial capsule network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[51] Raeid Saqur and Sal Vivona. Capsgan: Using dynamic routing for generative adversarial networks. In *Science and Information Conference*, pages 511–525. Springer, 2019.

[52] Yash Upadhyay and Paul Schrater. Generative adversarial network architectures for image synthesis using capsule networks. *arXiv preprint arXiv:1806.03796*, 2018.

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

[54] Rodney LaLonde and Ulas Bagci. Capsules for object segmentation. *arXiv preprint arXiv:1804.04241*, 2018.

[55] Kevin Duarte, Yogesh Rawat, and Mubarak Shah. Videocapsulenet: A simplified network for action detection. In *Advances in Neural Information Processing Systems*, pages 7610–7619, 2018.

[56] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point capsule networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1009–1018, 2019.

[57] Yiyi Zhou, Rongrong Ji, Jinsong Su, Xiaoshuai Sun, and Weiqiu Chen. Dynamic capsule attention for visual question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9324–9331, 2019.

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

[59] Zhang Xinyi and Lihui Chen. Capsule graph neural network. In *International conference on learning representations*, 2018.

[60] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[61] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[62] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[63] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[64] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[65] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[66] Adrien Deliège, Anthony Cioppa, and Marc Van Droogenbroeck. An effective hit-or-miss layer favoring feature interpretation as learned prototypes deformations. *arXiv preprint arXiv:1911.05588*, 2019.

[67] Sai Samarth R Phaye, Apoorva Sikka, Abhinav Dhall, and Deepti Bathula. Dense and diverse capsule networks: Making the capsules learn better. *arXiv preprint arXiv:1805.04001*, 2018.

[68] Canqun Xiang, Lu Zhang, Yi Tang, Wenbin Zou, and Chen Xu. Ms-capsnet: A novel multi-scale capsule network. *IEEE Signal Processing Letters*, 25(12):1850–1854, 2018.

[69] Prem Nair, Rohan Doshi, and Stefan Keselj. Pushing the limits of capsule networks. *Technical note*, 2018.

[70] Zhen Zhao, Ashley Kleinhans, Gursharan Sandhu, Ishan Patel, and KP Unnikrishnan. Capsule networks with max-min normalization. *arXiv preprint arXiv:1903.09662*, 2019.

[71] Taylor Killian, Justin Goodwin, Olivia Brown, and Sung-Hyun Son. Kernelized capsule networks. *arXiv preprint arXiv:1906.03164*, 2019.

[72] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–104. IEEE, 2004.

[73] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[74] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[75] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[76] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[77] Rodney LaLonde, Ziyue Xu, Ismail Irmakci, Sanjay Jain, and Ulas Bagci. Capsules for biomedical image segmentation. *Medical image analysis*, 68:101889, 2021.

[78] Binbin Zhang, Shikun Huang, Wen Shen, and Zhihua Wei. Explaining the pointnet: What has been learned inside the pointnet? In *CVPR Workshops*, pages 71–74, 2019.

[79] Hanxiao Tan and Helena Kotthaus. Surrogate model-based explainability methods for point cloud nns. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2239–2248, 2022.

[80] Daniel Liu, Ronald Yu, and Hao Su. Extending adversarial attacks and defenses to deep 3d point cloud classifiers. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2279–2283. IEEE, 2019.

[81] Hang Zhou, Kejiang Chen, Weiming Zhang, Han Fang, Wenbo Zhou, and Nenghai Yu. Dup-net: Denoiser and upsampler network for 3d adversarial point clouds defense. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1961–1970, 2019.

[82] Jiachen Sun, Yulong Cao, Christopher B Choy, Zhiding Yu, Anima Anandkumar, Zhuoqing Morley Mao, and Chaowei Xiao. Adversarially robust 3d point cloud recognition using self-supervisions. *Advances in Neural Information Processing Systems*, 34:15498–15512, 2021.

[83] Alioscia Petrelli and Luigi Di Stefano. A repeatable and efficient canonical reference for surface matching. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 403–410. IEEE, 2012.

[84] Wen Shen, Binbin Zhang, Shikun Huang, Zhihua Wei, and Quanshi Zhang. 3d-rotation-equivariant quaternion neural networks. In *European Conference on Computer Vision*, pages 531–547. Springer, 2020.

[85] F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.

[86] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[87] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[88] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[89] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE international conference on computer vision*, pages 863–872, 2017.

[90] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8778–8785, 2019.

[91] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning so (3) equivariant representations with spherical cnns. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–68, 2018.

[92] Yang You, Yujing Lou, Qi Liu, Yu-Wing Tai, Lizhuang Ma, Cewu Lu, and Weiming Wang. Pointwise rotation-invariant network with adaptive sampling and 3d spherical voxel convolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12717–12724, 2020.

[93] Haowen Deng, Tolga Birdal, and Slobodan Ilic. Ppf-foldnet: Unsupervised learning of rotation invariant 3d local descriptors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 602–618, 2018.

[94] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7163–7172, 2019.

[95] Wentao Yuan, David Held, Christoph Mertz, and Martial Hebert. Iterative transformer network for 3d point cloud. *arXiv preprint arXiv:1811.11209*, 2018.

[96] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[97] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[98] Filip Radenović, Giorgos Tolias, and Ondřej Chum. Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples. In *European conference on computer vision*, pages 3–20. Springer, 2016.

[99] Yifan Sun, Liang Zheng, Yi Yang, Qi Tian, and Shengjin Wang. Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline). In *Proceedings of the European conference on computer vision (ECCV)*, pages 480–496, 2018.

[100] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in neural information processing systems*, pages 1857–1865, 2016.

[101] Zhirong Wu, Alexei A Efros, and Stella X Yu. Improving generalization via scalable neighborhood component analysis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 685–701, 2018.

[102] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[103] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[104] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Pires, Zhaohan Guo, Mohammad Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*, 2020.

[105] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[106] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. Hard-aware deeply cascaded embedding. In *Proceedings of the IEEE international conference on computer vision*, pages 814–823, 2017.

[107] Michael Opitz, Georg Waltner, Horst Possegger, and Horst Bischof. Deep metric learning with bier: Boosting independent embeddings robustly. *IEEE transactions on pattern analysis and machine intelligence*, 42(2):276–290, 2018.

[108] Artsiom Sanakoyeu, Vadim Tschernezki, Uta Buchler, and Bjorn Ommer. Divide and conquer the embedding space for metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 471–480, 2019.

[109] Hong Xuan, Richard Souvenir, and Robert Pless. Deep randomized ensembles for metric learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 723–734, 2018.

[110] Hugo Touvron, Alexandre Sablayrolles, Matthijs Douze, Matthieu Cord, and Hervé Jégou. Grafit: Learning fine-grained image representations with coarse labels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 874–884, 2021.

[111] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[112] Zhiwen Yu, Daxing Wang, Jane You, Hau-San Wong, Si Wu, Jun Zhang, and Guoqiang Han. Progressive subspace ensemble learning. *Pattern Recognition*, 60:692–705, 2016.

[113] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[114] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[115] Guy Bukchin, Eli Schwartz, Kate Saenko, Ori Shahar, Rogerio Feris, Raja Giryes, and Leonid Karlinsky. Fine-grained angular contrastive learning with coarse labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8730–8740, 2021.

[116] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. Attention-based ensemble for deep metric learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 736–751, 2018.

[117] Albert Gordo, Jon Almazán, Jerome Revaud, and Diane Larlus. Deep image retrieval: Learning global representations for image search. In *European conference on computer vision*, pages 241–257. Springer, 2016.

[118] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[119] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17, 2004.

[120] Yair Movshovitz-Attias, Alexander Toshev, Thomas K Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 360–368, 2017.

[121] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997.

[122] Wengang Zhou, Houqiang Li, and Qi Tian. Recent advance in content-based image retrieval: A literature survey. *arXiv preprint arXiv:1706.06064*, 2017.

[123] A Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

[124] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[125] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019.

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

[127] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[128] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[129] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13001–13008, 2020.

[130] Xueting Yan, Ishan Misra, Abhinav Gupta, Deepti Ghadiyaram, and Dhruv Mahajan. Clusterfit: Improving generalization of visual representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6509–6518, 2020.