GENERALIZATION OF DEEP NEURAL NETWORKS TO TRANSFORMATIONS THROUGH NOVEL AND HYBRID ARCHITECTURES

A Dissertation

by

Barış Özcan

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in the
Department of Computer Science

Özyeğin University
December 2022

GENERALIZATION OF DEEP NEURAL NETWORKS TO TRANSFORMATIONS THROUGH NOVEL AND HYBRID ARCHITECTURES

Approved by:

Assistant Professor Mustafa Furkan Kıraç, Advisor
Department of Computer Science
Özyeğin University

Assistant Professor Berk Gökberk
Department of Computer Engineering
Boğaziçi University

Professor Hasan Fatih Uğurdağ
Department of Electrical and Electronics Engineering
Özyeğin University

Associate Professor Emre Uğur
Department of Computer Engineering
Boğaziçi University

Date Approved: 22 December 2022

Assistant Professor Reyhan Aydoğan
Department of Computer Science
Özyeğin University

To those who believe nothing they hear and half that they see.

ABSTRACT

Object recognition is a foundational pillar for many computer vision tasks such as searching, tracking, navigating, scene understanding or information retrieval that require some kind of category knowledge at various levels. Even with the major advances in these tasks with the data-driven deep learning methods such as convolutional neural networks (CNNs), generalization to geometric variations and embedding part-whole relationships are still yet to be achieved when compared to human-level recognition. CNNs particularly fail to generalize to unseen viewpoints of a learned object even with substantial samples and are easily confused as the pooling operations lose the relation between existing entities in the input. Recently emerged capsule networks outperform CNNs in novel viewpoint generalization tasks even with significantly fewer parameters. Capsule networks group the neuron activations for representing higher-level attributes and their interactions for achieving equivariance to visual transformations. Capsules are designed to represent the pose of an existing visual entity and learned transformations are essentially pose transformations which are matrices. However, capsule networks have a high computational cost for learning the interactions of capsules in consecutive layers via the, so-called, routing algorithm in addition to the training stability problems. In this thesis, we propose to represent the pose information and transformations with quaternions in Quaternion Capsule Networks (QCNs). Quaternions are immune to the gimbal lock, have straightforward regularization of the rotation representation for capsules, and require a smaller number of parameters than matrices. QCNs directly inherit the existing EM-Routing for a fair comparison of the benefits of using quaternions instead of matrices. Experimental results show that QCNs generalize better to novel viewpoints with fewer parameters,

and achieve on-par or better performances with the state-of-the-art Capsule architectures on well-known benchmarking datasets. Building on this proposal, we aimed to reduce the computational burden and embed feature vectors to the capsules in addition to pose information. In this context we propose, *Alleviated Pose Attentive Capsule Agreement* (*ALPACA*) which is tailored for capsules that contain pose, feature and existence probability information together to enhance novel viewpoint generalization of capsules on 2D images. For this purpose, we have created a Novel ViewPoint Dataset (*NVPD*) a viewpoint-controlled texture-free dataset that has 8 different setups where training and test samples are formed by different viewpoints. In addition to *NVPD*, we have conducted experiments on the iLab2M dataset where the dataset is split in terms of the object instances. Experimental results show that *ALPACA* outperforms its capsule network counterparts and state-of-the-art CNNs on iLab2M and *NVPD* datasets. Moreover, *ALPACA* is 10 times faster when compared to routing-based capsule networks. It also outperforms attention-based routing algorithms of the domain while keeping the inference and training times comparable.

ÖZETÇE

Nesne tanıma, arama, izleme, gezinme, sahne anlama veya bilgi alma gibi çeşitli seviyelerde bir tür kategori bilgisi gerektiren birçok bilgisayarla görü problemi için temel bir dayanaktır. Konvolüsyonel sinir ağları (CNN'ler) gibi veri odaklı derin öğrenme yöntemleriyle bu görevlerde büyük ilerlemeler kaydedilmiş olsa da, geometrik varyasyonlara genelleme ve parça bütün ilişkilerinin gömülmesi, insan seviyesinde tanımayla karşılaştırıldığında geri kalmaktadır. CNN'ler özellikle öğrenilen bir nesnenin görülmeyen bakış açılarına önemli örneklerle bile genelleme yapamamakta ve havuzlama işlemleri girdideki mevcut parçalar arasındaki ilişkiyi kaybettiğinden kolayca yanlış yönlendirilebilmektedir. Yakın zamanda ortaya çıkan kapsül ağları, çok daha az parametreyle bile yeni görüş açısına genelleme konusunda CNN'lerden daha iyi performans göstermektedir. Kapsül ağları, daha yüksek seviyeli öznitelikleri temsil etmek için nöron aktivasyonlarını ve görsel dönüşümlere eşdeğişkenlik sağlamak için etkileşimlerini gruplandırır. Kapsüller mevcut bir görsel varlığın pozunu temsil etmek üzere tasarlanmıştır ve öğrenilen dönüşümler esasen matrisler olan koordinat sistemi dönüşümleridir. Bununla birlikte, kapsül ağları, eğitim kararlılığı sorunlarına ek olarak, yönlendirme algoritması olarak adlandırılan algoritma aracılığıyla ardışık katmanlardaki kapsüllerin etkileşimlerini öğrenmek için yüksek bir hesaplama maliyetine sahiptir. Bu tezde, Kuaterniyon Kapsül Ağlarında (QCN) poz bilgisini ve dönüşümleri kuaterniyonlarla temsil etmeyi öneriyoruz. Kuaterniyonlar gimbal kilidine maruz kalmamaktadır, kapsüller için rotasyon temsilinin basit bir şekilde düzenlenmesi vardır ve matrislerden daha az sayıda parametre gerektirir. QCN'ler, matrisler yerine kuaterniyon kullanmanın faydalarının adil bir şekilde karşılaştırılması için mevcut EM-Routing'i doğrudan devralarak test edilmiştir. Deneysel sonuçlar,

QCN'lerin daha az parametre ile yeni bakış açılarına daha iyi genelleme yaptığını ve ayrıca iyi bilinen kıyaslama veri kümelerinde en son teknoloji Kapsül mimarileriyle eşit veya daha iyi performanslar elde ettiğini göstermektedir. Bu öneriyi temel alarak, hesaplama yükünü azaltmayı ve poz bilgisine ek olarak kapsüllere özellik vektörleri yerleştirmeyi amaçladık. Bu bağlamda, kapsüllerin 2B görüntüler üzerinde yeni görüş açısı genelleştirmesini geliştirmek için poz, özellik ve var olma olasılığı bilgilerini birlikte içeren kapsüller için uyarlanmış *Alleviated Pose Attentive Capsule Agreement* (*ALPACA*) geliştirdik. Bu amaçla, eğitim ve test örneklerinin farklı bakış açıları tarafından oluşturulduğu 8 farklı düzeneğe sahip, bakış açısı kontrollü dokusuz bir veri kümesi olan Novel ViewPoint Dataset (*NVPD*) oluşturduk. *NVPD*'ye ek olarak, veri kümesinin nesne örnekleri açısından bölündüğü iLab2M veri kümesi üzerinde de deneyler gerçekleştirdik. Deneysel sonuçlar, *ALPACA*'nın iLab2M ve *NVPD* veri kümelerinde kapsül ağ muadillerinden ve son teknoloji CNN'lerden daha iyi performans gösterdiğini ortaya koymaktadır. Dahası, *ALPACA* yönlendirme tabanlı kapsül ağlarına kıyasla 10 kat daha hızlıdır. Ayrıca, çıkarım ve eğitim sürelerini karşılaştırılabilir tutarken, alanın dikkat tabanlı yönlendirme algoritmalarından daha iyi performans gösterir.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor Asst. Prof. Furkan Kıraç for his invaluable support, advice, and patience during my Ph.D. I would like to thank my thesis committee members Prof. Fatih Uğurdağ, Assoc. Prof Murat Şensoy and Asst. Prof. Reyhan Aydoğan, without their guidance I would not be able to finish this thesis. Particularly Prof. Fatih Uğurdağ, who made this possible for me by introducing me to my advisor. I would also like to thank Prof. Erhan Öztop, thanks to his insightful comments I was able to explore one of the foundational pillars of my thesis. I would like to mention Furkan Kınlı who is an outstanding colleague and a driving force, Doğa Yılmaz who was a great help in my publications and Mahir Atmış who has been a great companion. I also would like to mention Emre Göynügür, Osman Kaya, and Emir Arditi who have been great friends. Last but not least, I cannot express how grateful I am to my beloved wife and family for their understanding, as I could not have pulled through the years working on my thesis without them. Also, my friends mocked me for years about when will it end, I also thank them for being the whip in this journey. Finally, I would like to thank my dog, Mr.Spock for his valuable comments on my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Many computer vision task such as searching, tracking, navigating, scene understanding, or information retrieval requires some kind of category knowledge at various levels. Determining the category of a given visual is the problem known as image recognition. These visuals can be images or bounding boxes defining a region of interest in the image. Depending on the category types and levels, image recognition can also be defined as object recognition where the categories to be recognized are limited to objects (e.g. table, chair, car, plane, guitar, or buildings, etc.). One of the fundamental challenges in object recognition in terms of generalization is to handle geometric variations in the images such as scale, pose, viewpoint, and deformable parts to achieve human-level visual recognition and learning capabilities. For example, a human can easily recognize that each image in Figure 1 belongs to the same object without seeing the same object from various angles previously. Another example of the human ability to understand geometric variations is handwritten characters and digits. A human can to recognize a written character with any reasonable distortion (Figure 2) even after seeing a few samples. This ability to generalize to geometric variations even from a single sample is still yet to be achieved in visual recognition systems.

Previously, object recognition utilized engineered features that represent the pose such as SIFT [7], SURF [8] or ORB [9]. All these methods aim to find keypoints in the input such as edges or corners and describe it such that it is invariant to various conditions such as color, scale, or rotation. The most common pipeline for object recognition is to use a classifier over the extracted features such as support

Figure 1: The same star destroyer from different viewpoints in (a), (b), and (c).

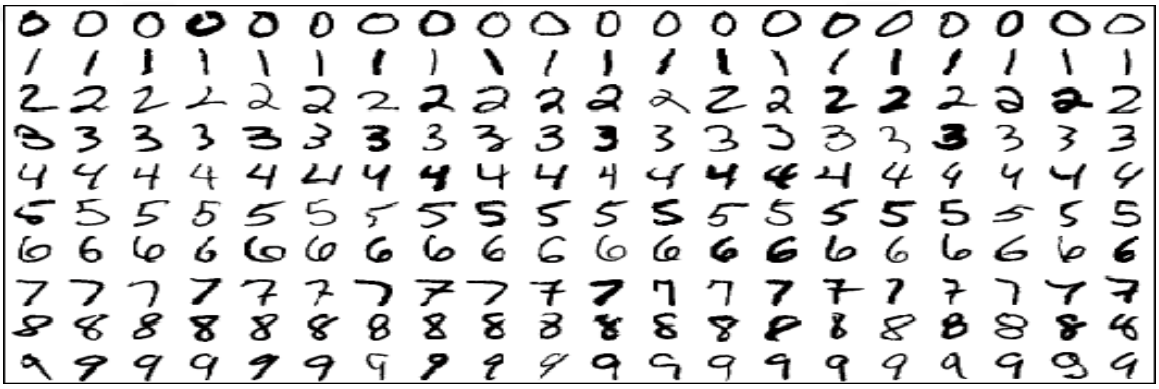


Figure 2: Handwritten digits with different distortions from the MNIST dataset.

vector machines [10] that will classify the input via these features in the image. Even though all these features are designed and tested thoroughly, object recognition is a highly nonlinear task and is very challenging to design a feature that can handle all variations in the data. Consequently, these descriptors dramatically affect recognition performance. This led to data-driven feature extraction methods where the features are learned from the data. Bag of words [11] approach, extracts features from the dataset at hand, finds clusters, and applies principal component analysis to find the dimensionality of the used features that provide the best split before training a classifier. Another data-driven method is random decision forests on shape recognition [12], where each pixel in an input image is classified by a loose but constrained feature design, in the end, these features aim to split the pixels with these learned features where each leaf contains as homogeneous as possible in each leaf and utilize majority

voting for classification.

However, most of the recent breakthroughs in object recognition are achieved by Deep Neural Networks (DNNs) where the features are learned from the data instead of being engineered. DNNs essentially learn the features together with the classifier such that learned features uplift the classifier's performance during the training. Particularly, Convolutional Neural Networks (CNNs) have achieved unprecedented performance improvements in various Computer Vision tasks such as object recognition [13], handwritten digits [14], faces [15], image classification [16] and semantic segmentation [17].

The generalization of neural networks is an actively debated topic in the literature and has cases for each application field such as vision, natural language processing, or tabular data. Until recently, a common approach is to avoid over-fitting on the training dataset to achieve better generalization by applying some form of weight regularization or reducing the model complexity. However, recent studies claim that there is a double descent phenomenon [18] where larger models achieve better generalization. Double descent is still an actively researched topic where the experiments are not limited to vision problems. On the other hand, another approach is to induce equivariance to the model architecture to achieve better generalization. However, human-level object recognition abilities are still yet to be achieved in terms of generalization to viewpoints, as CNNs fail to generalize to geometric variations.

Achievements of CNNs in visual recognition tasks are related to their ability of generalization to translation and scale variations in the images. CNN's search for the same feature on different locations in the image that benefit from translation symmetry, thus making CNNs equivariant to small translations in the input image. Additionally, stacking convolutional layers that have local connections that allow CNNs to capture the structure in different scales that result in scale invariance. This heuristic is valid for visual recognition tasks since a visual entity can appear in any

location on the image with exactly the same properties and far-apart pixels in an image are often uncorrelated. Therefore, CNNs can learn more features with fewer parameters and data since the search space for a good parameter set is reduced when compared to fully connected neural networks without sacrificing model capability. However, CNNs are not equivariant to all types of geometric variations such as object transformations or different viewpoints of the object of interest. This led to deeper CNNs that have millions of parameters and thus require a massive amount of data for training [19, 20, 21, 22] as CNNs are challenged with more complex recognition problems. While stacking layers and pooling operations provide invariance to a subset of geometric variations, this hinders the generalization ability of CNNs. Recently it is shown that very deep CNNs fail to generalize to novel viewpoints even when large amounts of data is provided [23].

Lastly, CNNs are concerned with the individual existence of a visual entity due to the pooling operation, thus cannot differentiate if a set of visual entities in an image are spatially distributed in a meaningful way (*e.g.* swapping an eye with a mouth in a face image). Whereas, without max pooling, CNNs can have a "place code" of the input that provides equivariance to translations. This "place code" roughly encodes the location of an input entity in the image which is bounded by the kernel and step sizes at each layer of a CNN. However, "place-code" is not enough for viewpoint equivariance as translation is only one of the possible transformations that occur during a viewpoint change. Therefore, it is fair to claim that CNNs fail to generalize to novel viewpoints. However, there is a deficiency of empirical studies on CNNs generalization capabilities to novel viewpoints in the literature that supports this claim.

Transformations between the pose of an object and its parts poses are constant under any viewpoint variations when the object of interest is assumed to have no deformable parts. Additionally, instantiations of these poses of the parts change as the

viewpoint changes while the transformations in between stay constant, which introduces equivariance to viewpoint transformations. An illustration of this assumption is given in Figure 16. To benefit from this assumption, a neural network should be able to represent the pose and any other required information, in addition, to being able to learn the in-between transformations of parts and wholes. For this purpose, capsules, which are the encapsulation of neuron activations are proposed. This allows capsules to create complex representations that are required to exploit the constant relationships between the parts and the wholes. Capsules can represent pose, feature vector, existence probability, or any other high-dimensional information in the given context.

Recently proposed *Capsule Networks* [24, 1] aim to address the above-mentioned deficiencies of CNNs by introducing viewpoint equivariance for better generalization to affine transformations and viewpoint variations while maintaining the relationship between the visual entities. Experimental results show that early capsule networks achieve better performance on MNIST [24] with a significantly lower number of parameters. More interestingly, capsule networks better generalize to unseen viewpoints due to their equivariant property on smallNORB dataset [1, 25]. However, none of these works were able to achieve performant results on larger databases. Since the relationship between the capsules is a computational bottleneck that causes instability during training and hinders them from scaling up to larger datasets [26]. Another shortcoming in the capsule literature is the way of conducting experiments on generalization to unseen viewpoints, compared to CNNs. This is done on a special setup of smallNORB where the dataset is split in terms of viewpoints [1]. To compare the generalization performance of capsule networks and CNNs to unseen viewpoints on smallNORB, training is stopped when training set accuracies of all compared methods are similar. This comparison is not fair since the training could be stopped before it is fully converged. Another drawback of smallNORB is the lack of class variability

and viewpoints.

1.1 Contributions an Outline

In this thesis, we aim to thoroughly analyze and improve the novel viewpoint generalization capabilities of neural networks. For this purpose, we address several deficiencies of capsule networks and provide a dataset that is specially tailored to test novel viewpoint generalization. The first improvement we introduce is the Quaternion Capsule Networks (QCN) [27], a novel form of capsules where the capsule orientations and their rotations are governed by quaternion algebra. Proposed QCN utilizes an already existing capsule routing algorithm to have a fair comparison of quaternions and matrix transformations. QCNs have four main contributions to both Capsule Network and Quaternion Network literature:

(i) Capsule representation with quaternions is proposed, and mapping between them in quaternion algebra is formulated.

(ii) *QCNs* can to learn the rotation axis along with the rotation angle, unlike the previous studies of feed-forward Quaternion Networks where the rotation axis is constant.

(iii) Extracting pose and activations from the input image is divided into two spatially aligned branches that allow using different sub-networks with different complexities for different modalities such as activation and pose.

Experimental results show that QCN achieves state-of-the-art performance in case of generalization to novel viewpoints. QCN also attains on-par performance with Matrix Capsules on well-known benchmarking datasets with only half of the parameters without any hyper-parameter tuning or architecture changes with respect to the dataset.

Building upon *QCN*, we introduce a novel capsule agreement algorithm *Alleviated Pose Attentive Capsule Agreement* (*ALPACA*) [28]. *ALPACA* is particularly tailored for capsules that contain pose, feature, and existence probability information together to enhance novel viewpoint generalization of capsules on 2D images. In addition to *ALPACA*, we introduce *Novel ViewPoint Dataset*(*NVPD*) that allows us to split the dataset in various setups to understand whether a given method fails in generalizing to distance, azimuth, and elevation differences or how the variance of the viewpoints available during the training affects the performance. An ideal dataset for development and analysis would require numerous experiments, thus we aimed for a lightweight but challenging dataset. *NVPD* accuracies of state-of-the-art methods do not saturate at maximum so that we can differentiate the performances. An overview of our contributions to this study is as follows:

(i) We propose a novel routing scheme called *Alleviated Pose Attentive Capsule Agreement* (*ALPACA*), which depends on the part-whole pose similarities in terms of the geodesic distance to compute the features of the parent capsules. Also, we employ a single vote calculation module for determining the parent capsule, instead of using learned parameters for each connection. This module reduces the number of parameters and computational complexity.

(ii) We propose a novel module for primary capsule extraction with the idea of composing the information from different heads.

(iii) We propose a novel activation for quaternion capsules where the normalized eigenvalue corresponding to the quaternion average and the total variation over the feature vector is used.

(iv) We introduce a new dataset, derived from ShapeNet, specialized for the novel viewpoint generalization task on 2D images called Novel Viewpoint Dataset (*NVPD*), which contains 10 classes, 50 instances per class, and 324 different

viewpoints per instance (*i.e.,* 18 azimuth angles, 6 elevations, and 3 distances). Each sample in the dataset set is without texture and background. *NVPD* provides 8 different training and test splits for further evaluating the generalization performance. Additionally, the benchmarks of the well-known capsule and CNN architectures are provided for this dataset.

(v) We provide the results of both our model and the state-of-the-art capsule networks on iLab2M [2] that is a toy vehicle objects dataset containing 2 million samples with background and different viewpoint variations.

(vi) While outperforming its counterparts in most of the test setups, *ALPACA* increases the average training and test speeds by ~10 fold when compared with the state-of-the-art approach.

ALPACA architecture outperforms its counterparts and CNNs both in *NVPD* and iLab2M datasets where the generalization to novel viewpoints is thoroughly analyzed. Particularly on the iLab2M dataset, *ALPACA* surpasses its counterparts by at least 4% in terms of accuracy. Likewise, *ALPACA* surpasses, *ResNet50* and *DenseNet* ensemble results which are trained on images with 8 times higher resolution. In addition to the accuracy results, we provide further details about the effects of the reconstruction, loss functions, and proposed activation on the performance of our proposed architecture.

This thesis consists of three chapters, Background and Motivation (Chapter 2), Methodology and Experimental Results (Chapter 3), and Conclusion (Chapter 4). In the Background and Motivation Chapter, we explain the reasoning that lead to Capsule Networks, the difference between invariance and equivariance, and quaternion-valued neural networks and present a general background related to both concepts in the context of neural networks to establish a basic understanding of the proposition in this thesis. Methodology and Experimental Results are divided into four sections

based on the two publications related to this thesis. Initially, a capsule notation is established in Section 3.1 to avoid confusion in the following sections. In Section 3.2, *QCN* [27] is explained and its experimental results on conventional datasets are provided while *NVPD* is presented in Section 3.3. Lastly, *ALPACA* [28] and its experimental results on *NVPD* and iLab2M [2] datasets. This section also provides results of state-of-the-art CNNs and Capsule Networks on *NVPD*. Related work to each method is presented in their respective sections for clarity.

CHAPTER II

BACKGROUND AND MOTIVATION

2.1 Invariance and Equivariance in Neural Networks

As mentioned previously, the success of CNNs relies upon the translational symmetry in the images. The same patterns can be found in the images in various spatial locations. Therefore, traversing the same filter on the image rather than learning new weights for every location is a beneficial induced bias in the model. This lead to convolutional layers which are parameter efficient when compared to fully connected layers. Sharing weights not only provides equivariance to translational symmetries in the image but simplifies the learning problem at hand by reducing the number of learnable parameters by a large margin. In contrast to invariant models, equivariant models provide additional information in its output. The existence of this additional information allows the training to allocate the learned parameters for essential patterns for recognition rather than compensating for the invariance. Deep CNNs, which enabled major advances in computer vision, basically stack these convolutional layers that are translation equivariant.

Although they are often used interchangeably, invariance and equivariance have an important difference. A function f that maps input image $I \in \mathbb{R}^{m \times n \times d}$ is invariant to the group of actions G if and only if when an action $g \in G$ applied to the input image the output of f stays the same. Whereas there is a tractable action between the outputs if the function f is equivariant to the group of actions G. Equivariance and invariance can formally be defined as in Equations (1),(2) and are illustrated in Figure 3.

$$f(g(I)) = f(I) \tag{1}$$

$$f(g(I)) = g(f(I)) \tag{2}$$

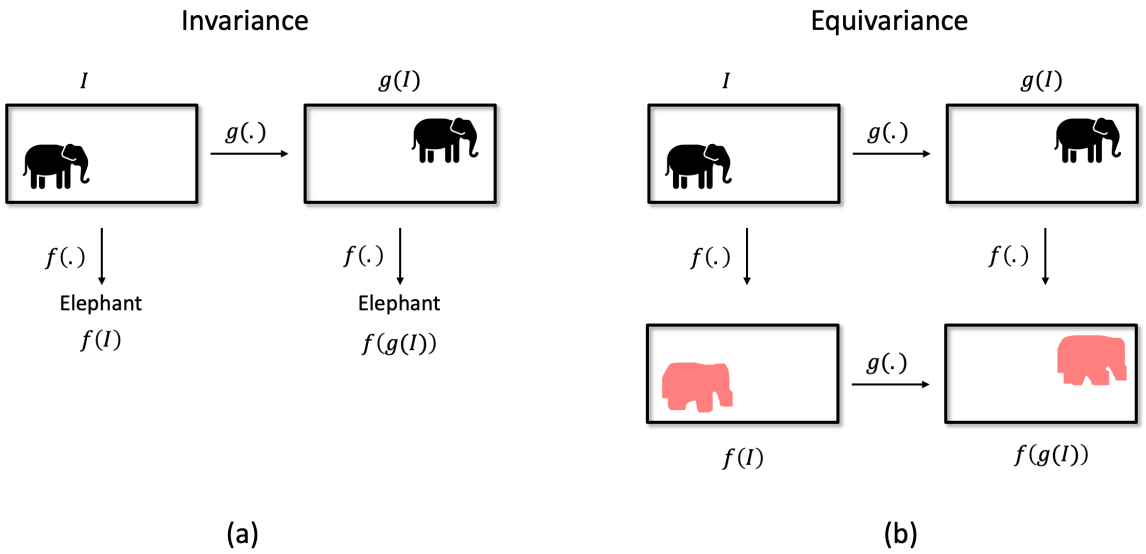


Figure 3: (a) invariance vs. (b) equivariance. $g(.)$ is a translation operation in this case where, $f(.)$ is the model and I is the input image.

In the case of translation, g is a translation operation (shifting the locations) which is simply moving the object of interest in the image while preserving every other visual aspect of the object. A translation-invariant function f's output will be the same regardless of the applied translation g. On the other hand, if f is equivariant to translations, there will be translation g between the responses of f to translated and original inputs. Convolutional Neural networks are equivariant to translations in images since the neuron activations depend on the locations of interesting features in the image. On the other hand, max-pooling or equivalent sub-sampling operations in convolutional neural networks make locally (kernel size) invariant layers to translations. This phenomenon is illustrated in Figure 4. One can observe that the neuron activations after max pooling in Figure 4 are not equal to the

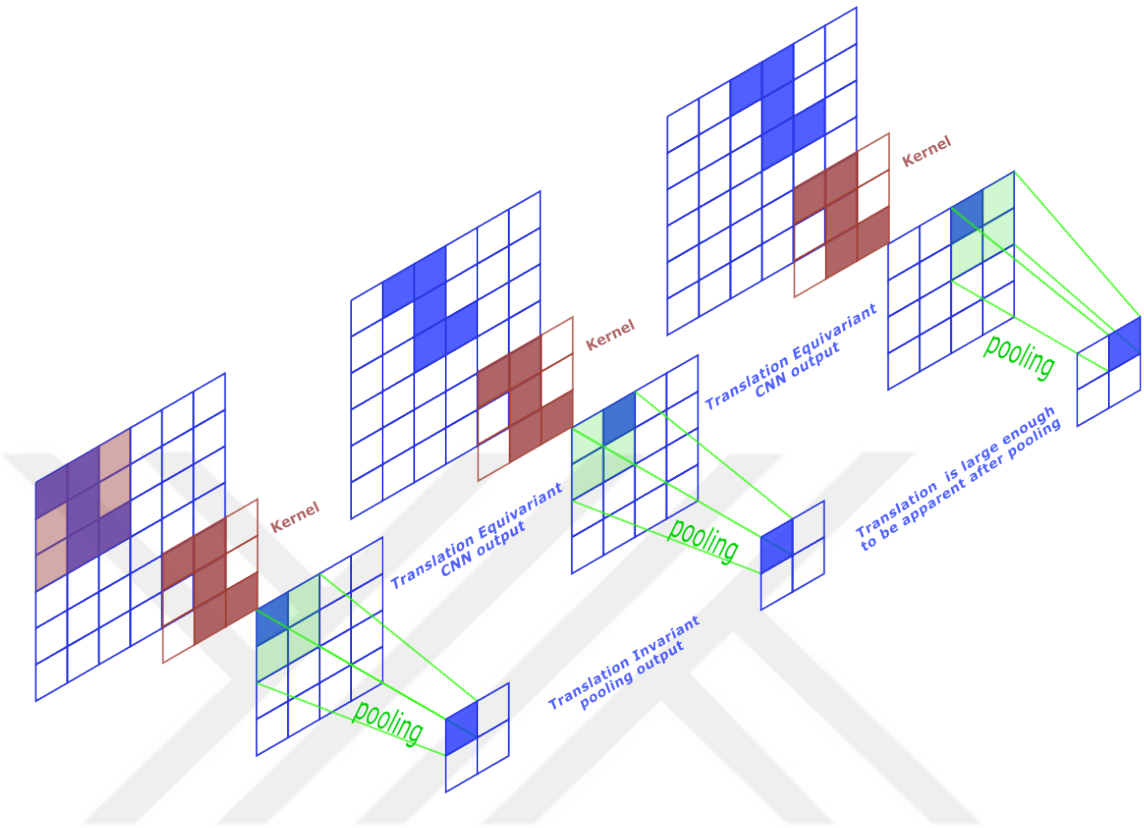


Figure 4: Illustration of partial equivariance to translation of CNNs. The translation in the input needs to be large enough to be encoded in the output of the pooling operation, otherwise, the output is invariant. On the other hand, outputs of convolutional layers prior to CNNs are translation equivariant by nature as the output translates in unison with the input pattern.

translation between its inputs. More precisely, the translation on the input should be large enough to be encoded if there is a sub-sampling operation as shown in Figure 4.

On the other hand, CNNs are neither invariant nor equivariant to rotations in the image. As CNNs are spatially translated in the image therefore any rotation in the input can only be detected by learning weights, especially for that rotation. A very simple example is illustrated in Figure 5 in 2D where each node is a learned rotation of the input image. However, note that learned rotations are only the filter weights and only output a scalar activation. Moreover, even without the max pooling

operation, it is impossible to know, for practical datasets, which weights are learned for which rotation-class pairs in a classic CNN for the moment.

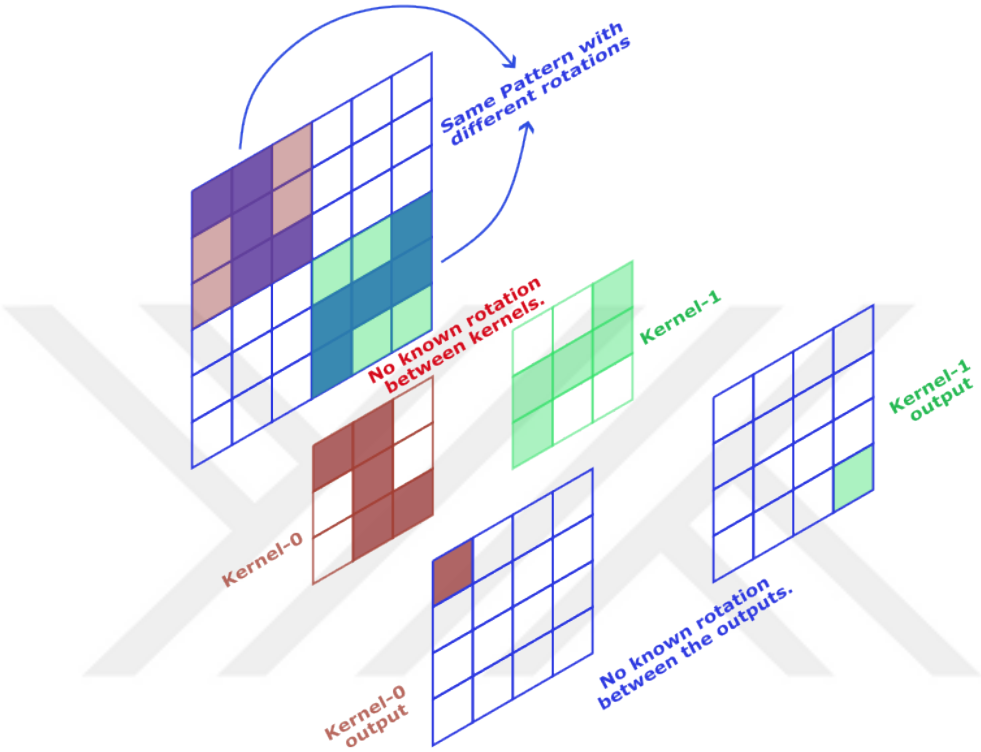


Figure 5: Illustration of how rotations on the input pattern are recognized by a CNN. To recognize pattern under rotations, CNNs require to learn kernels that is specialized for the rotated versions of the pattern of interest. In CNNs transformations between the kernels are unknown therefore the transformations between the outputs are also unknown. Consequently, CNNs are neither invariant nor equivariant to rotations in the input.

Similarly, a more challenging case is the viewpoint equivariance, where $g()$ is a viewpoint transformation $V(.)$(effect of a viewpoint transformation on the image is illustrated in Figure 6). Typically, a viewpoint transformation is a coordinate transformation such as the well-known bird-eye view projection from a given image using the camera model. A viewpoint transformation is a 4×4 matrix acting over a $3D$ pose vector. This mapping should be applicable to the outputs of the equivariant model $f(.)$. Therefore, outputs of this model should be $3d$ vectors, to satisfy the

equivariance property $f(V(I)) = V(f(I))$, which CNNs lack. Translation and rotation transformations are the subset of viewpoint transformations. As we know that CNNs are not invariant or equivariant to rotations, it can be deduced that CNNs lack these properties for viewpoint changes too. Currently, the only way to compensate for these deficiencies in viewpoint transformations is to learn respective weights to represent filters in convolutional layers for each possible viewpoint variant in the input. One can generalize this problem to any variation in the view or object properties such as scene lighting or object texture. Consequently, CNNs require huge amounts of data from various viewpoints to learn these filters thus leaning on augmentation methods and massive amounts of resources both in terms of GPU power and storage.

With that being said, recent studies on CNN visualizations show that there is some evidence of inherent equivariance up to a degree in CNNs that are trained on natural images [29]. However, these equivariant property is in between the neuron activations as mentioned previously, meaning that CNN has to learn a new feature for a range of transformations. Curve detectors [30] is a very good example of such a case. In this study, researchers visualize the neuron activations for rotated images. Results show that Curve detectors learned in CNNs, respond to a range of rotations that slightly overlap. Unfortunately, these neuron groups do not know their rotation or relation to other neuron groups. Still, this is an indication of CNNs huge success in vision problems compared to the earlier methods. Even though these indications, in this study we have experienced that state-of-the-art CNNs fail in viewpoint transformations that provide empirical evidence of the deficiency over the viewpoint transformations acted over the inputs.

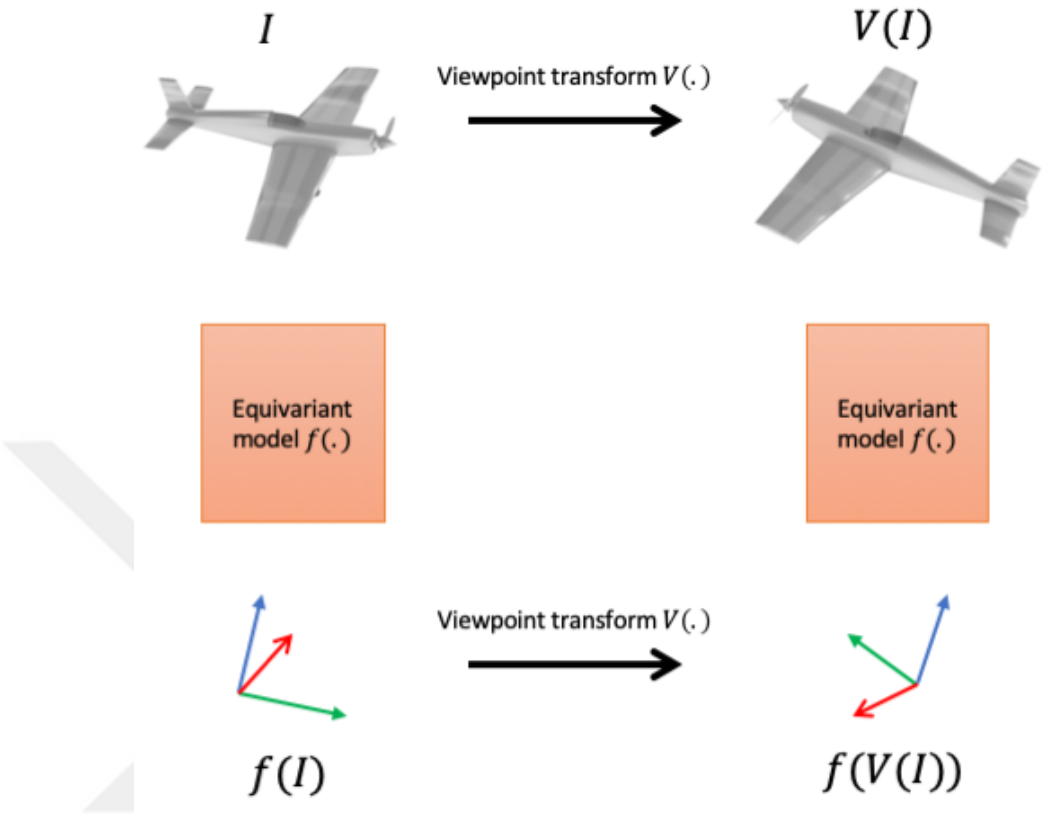


Figure 6: A viewpoint equivariant model outputs over transformed input images. Outputs to an image and its transformed version of this model should have a valid mapping in-between.

2.2 Capsule Networks

2.2.1 Motivation of Capsule Networks

Although there are empirical studies that suggest that CNNs tend to learn shapes and object parts in the deeper layers, recent findings prove that CNNs are heavily biased towards the texture unlike humans [3]. Even when an image is destroyed by induced textures, CNNs can recognize the object while humans fail. This also has a major drawback, when CNNs asked to recognize shapes without textures. As illustrated in Figure 7, a typical CNN confuses the existing object because of the texture in the image, however, it is very apparent for humans.

One major problem with CNNs is that they do not encode relationships between

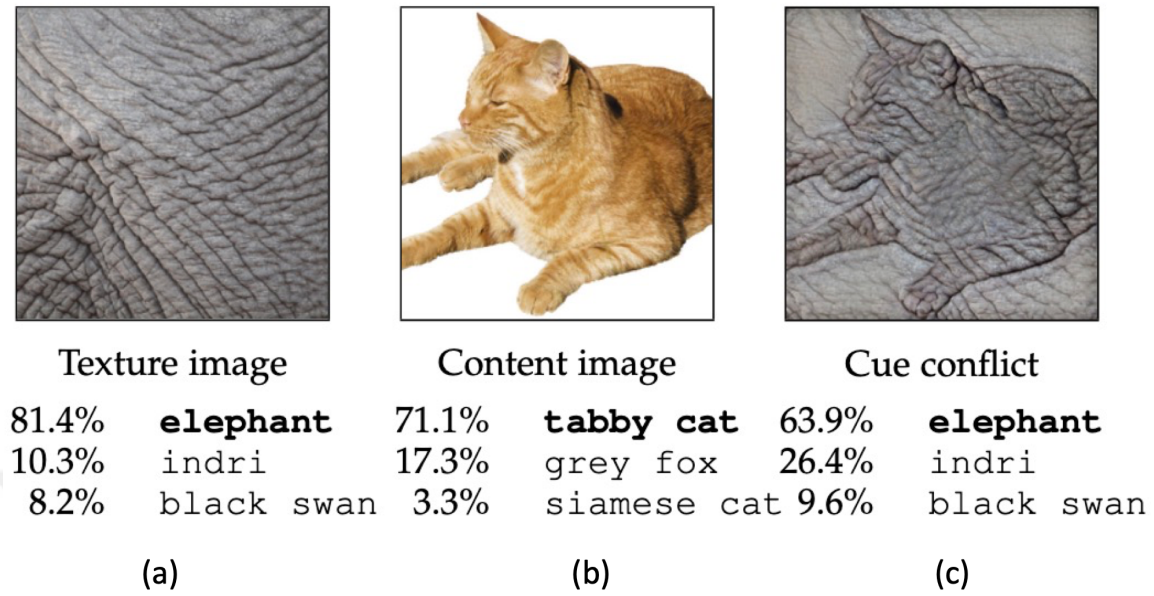


Figure 7: Texture bias example from [3], the response of a neural network to (a) only texture, (b) normal image, and (c) texture embed on the image.

the parts and objects. Since CNNs are working with scalars, they only consider the existence of a feature in the image and with every layer produce a weighted sum of features as the network gets deeper. This results in what's called the "Picasso Problem" [4], where the image contains every part of an object but in irrelevant orientations as shown in Figure 8.

Human perception is able to realize novel viewpoints of an object for very few observations, empirical studies show that it is because humans fit a combination of coordinate systems to the object of interest [31]. Evidence suggests that humans have a hierarchical coordinate system combination for objects. This approach to object recognition is explicitly modeled in capsule networks which can be considered as an inductive bias just as a convolution operation that leans on the fact that close-by pixels have more correlation in an image when compared to far-apart pixels. However, embedding the aforementioned concepts requires more sophisticated algorithms between layers that require higher dimensional information flowing between layers in contrast to scalars in CNNs.

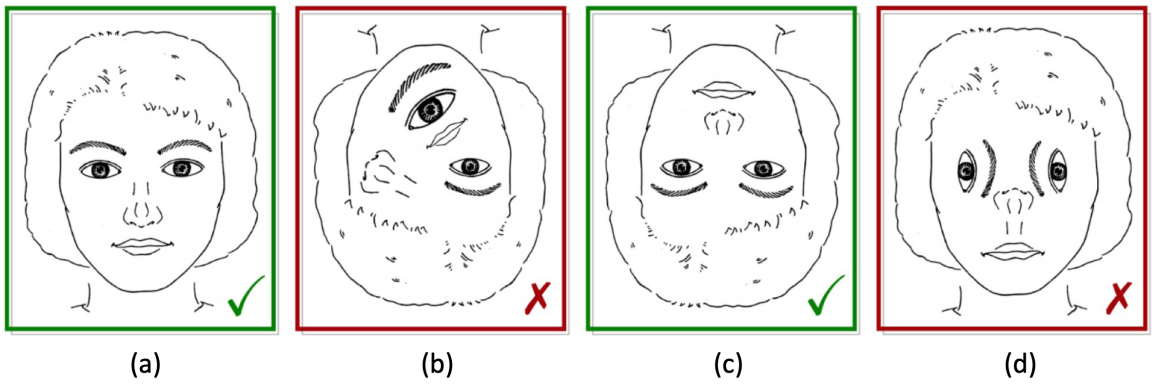


Figure 8: Picasso Problem figure from [4]. Examples of valid ((a) and (c)) and invalid ((b) and (d)) faces are where all the parts belong to a face of a different rotation.

In this context, capsules aim to represent a form of pose or feature information in each capsule to imitate this approach of human perception in neural networks. From another perspective, this relies on learning the linear manifold of transformations between the parts and the whole. This linear manifold exists if we assume the object of interest rigid as the transformations between the poses of parts and the object origin is constant under viewpoint variations in the object's coordinate frame. This is easy to learn via backpropagation, however, classical neural networks are incapable of such computations between layers therefore very first capsule models utilize a predefined set of transformations or learned vectors and matrices. Theoretically, to learn these transformations, the model should be able to represent everything in the object coordinate frame. This requires equivariance to viewpoints as you need to know the transformations on the input image. For a simplistic 2D case, viewpoint transformation can be reduced to 2D rotation. In Figure 9, this property of rigid objects is illustrated. The relation of the L object in the image is divided into a horizontal and a vertical rectangle. Note that, the angle between these rectangles is constant (90°) independent of the rotation on the input image.

Other properties of an object such as texture, color, or shape are a combination of these properties of its parts. A representative example is demonstrated in Figure 10

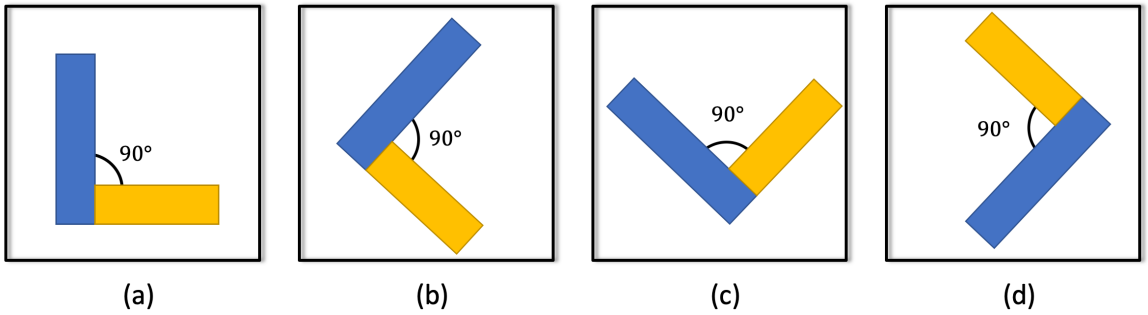


Figure 9: Simplistic illustration of constant transformations between parts independent of the transformation on the input image. Same shape with different rotations are given in (a), (b), (c) and (d). A 2D shape is used to reduce the complexity of the figure. One may clearly see that the angle between the horizontal part and the vertical part is always constant, independent of the rotation applied to the L object.

where a ship is divided into bow, stern, midship and sail parts. One can see from the figure that the color and shapes of each of these parts define the ship's texture and shape. If we were to define the shape of the ship at hand just from the blue-colored midship part, both ships would have the same texture and shape. The texture or shape information of an object and the relation of its parts in this context is also independent of the viewpoint variations. Therefore, if a network has the capability the learn this relation, it should be viewpoint invariant.

Other than the inspiration from human perception, capsule Networks are based on a few assumptions which are as follows:

1. Coordinate frames of parts and the object poses have a fixed transformation given the object of interest does not have any deformable parts.

2. High dimensional coincidences are effective feature detectors [32].

3. Part whole relationships are unique for most objects.

In theory, each capsule up to the last layer should represent the hypothetical part's pose and existence probability and other properties if applicable. This is modeled in the capsule networks however is not explicitly enforced during the learning process. In

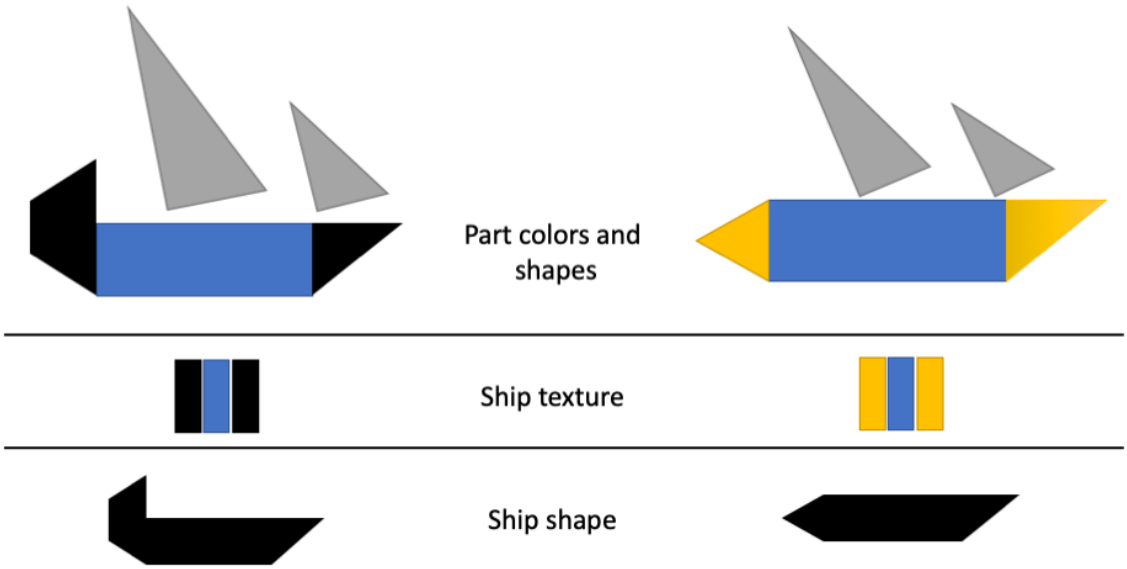


Figure 10: Shape and color of an object is defined by the combination of its parts. Even though the mid sections of both ships has the same shape and color, due to the difference in bow and stern, these ships are different in terms of shape and color.

the following pages few examples from the literature, that aim to enforce capsules to represent parts are provided, however, these examples work on 3d point clouds. Considering aforementioned assumptions, the capsule structure's main aim is to achieve an effective equivariance in terms of poses and part-whole agreements of an object. Consequently, allow neural networks to generalize better to novel viewpoints without the requirement of massive data from every possible viewing condition.

2.2.2 Background of Capsule Networks

Dealing with geometric variations in the input image has been an active problem for years in computer vision. This led to several tracks that aim to provide a form of robustness to these variations. While some researchers seek invariance, others aim to embed equivariance to neural networks as we know that CNNs success is dependent on its translational equivariance.

Spatial Transformer Networks (STNs) and their extensions aim to achieve viewpoint invariance by applying affine transformations to the feature map[33, 34]. Particularly Deformable CNNs [5] learn transformations from the data by introducing two novel features to CNNs. The first feature, the deformable convolution, transforms the receptive field by adding offsets to the sampling grid of a standard convolution as illustrated in Figure 11 which transforms the sampling grid of the receptive field. Secondly, deformable region of interest pooling similarly deforms the pooling bins. Both offsets are learned from the preceding feature maps that enable adaptive localization for different objects.

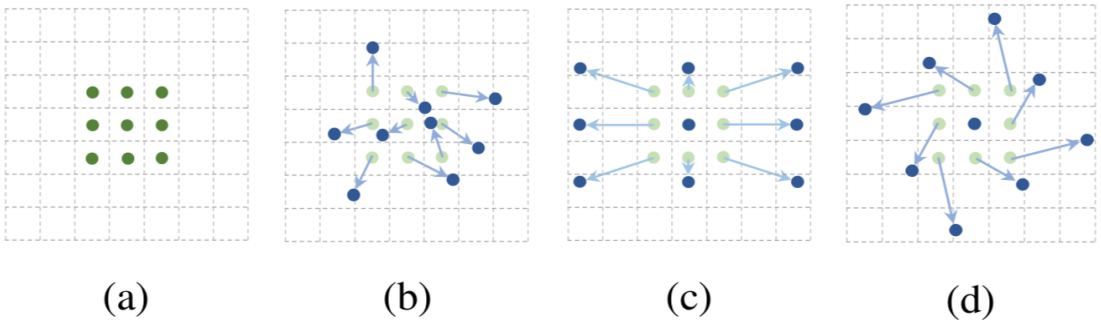


Figure 11: The deformable convolution example. (a) standard sampling grid, (b) deformed sampling grid, and (c) and (d) are special cases. Figure taken from [5]

Another approach is inspired by the weight sharing between the translational symmetries in the data of CNNS. Group equivariant CNNs [35] introduce rotations to the learned filters to provide a form of rotation equivariance in convolutional layers. However, this is limited to 90 degrees of rotations due to the computational burden of introducing larger symmetry groups. Steerable CNNs [36] provide a formulation to avoid learning multiple filters for each rotation and relieve the computational burden in Group CNNs for larger symmetry groups. Harmonic nets [37] improve this approach by exploiting the embedded rotation information in complex numbers.

Another approach is capsule networks. However, in contrast to previously mentioned methods that only aim for equivariance in terms of geometric variations, capsule networks also aim for a hierarchical consistency of an input image. Capsules are basically a structured grouping of multiple neuron activations to allow sophisticated representations. In contrast to classic neural networks where scalar values flow between layers, any type of tensors or tuples or grouping of representations flow between capsule layers. Initial capsule networks utilized vector activations and transformations in substitution with scalar activations. There is no limit to what the capsule represents as long as it is backpropagation compatible given an appropriate routing between capsule layers. Some examples are vector capsules, matrix capsules, or more complex capsules that consist of an existence probability along with a matrix.

The general approach in all Capsule Networks is analogous to parse trees, where small entities (child) in the image are combined to generate a representation of a larger and more complex entity (parent). While child capsules are considered as parts, parent capsules are considered to be a combination of these parts. In conventional capsule networks, capsules in lower layers are designed to be parents of the capsules from the previous layer that are connected to them. These connections can be in any form such as fully connected or convolutional. Child capsules cast a vote for its parent capsules state, which is a transformation of the child capsules state. Parent capsules are constructed by a mapping called *routing* $\Gamma(.)$ which essentially aggregates the incoming votes from child capsules. *Routing* $\Gamma(.)$ can either be a learned or a fixed mapping. The only mandatory property that Capsule Networks must have is that *routing* and any transformation to calculate votes to parent capsules must be compatible with backpropagation by design. Non-linearity between layers is often provided by the routing mechanisms in the literature, although it can be as simple as taking an average of the incoming votes.

The first example of using capsules instead of scalar neurons was introduced as

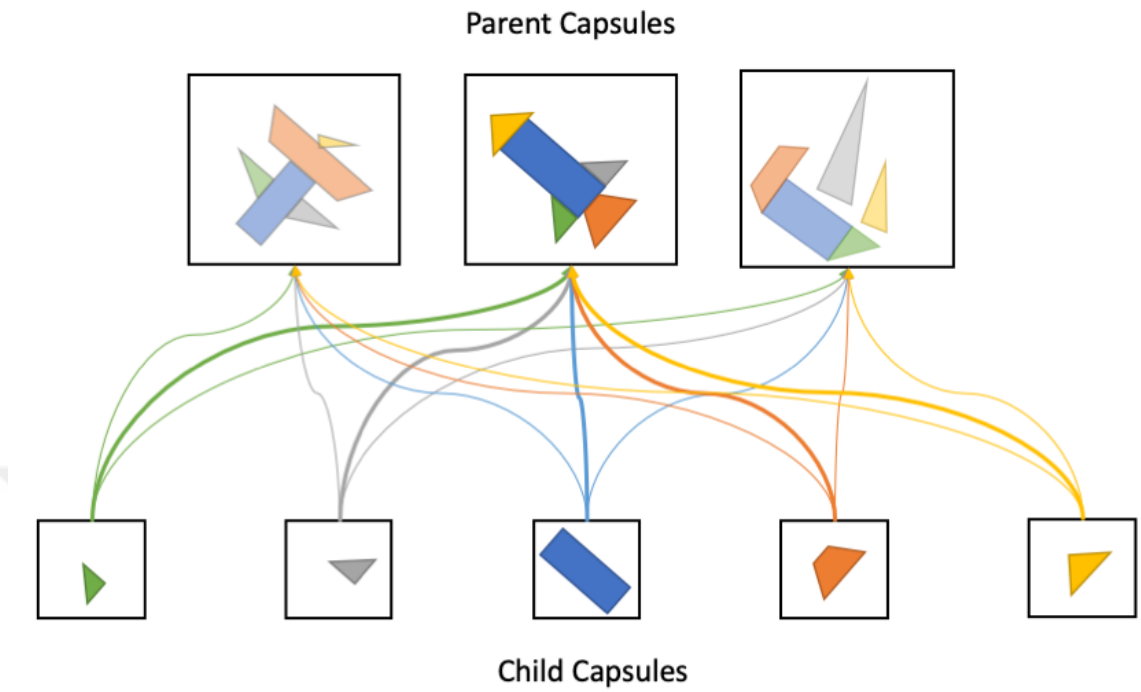


Figure 12: A simplistic routing illustration between the child capsules and a parent capsule. The darker the parent capsule is the more the agreement between the votes from child capsules.

Transforming Autoencoders [6] arguing that complex computations are necessary to achieve human-level recognition capabilities that include part-whole hierarchies between visual entities. Transforming autoencoders create a stereo pair of the input image by applying a transformation T and aim to reproduce the transformed version of the image where T is constrained to affine transformations from a predefined set. Capsules in transforming autoencoders have a recognition and a generation unit that contribute to the reproduced image depending on their existence probability. The Recognition unit estimates the pose information and existence probability while the generation unit generates aims to generate the transformed output. On the other hand, generation unit inputs are the pose estimates of the recognition unit transformed by the same T applied to the input image. A simple case of capsules that model translations is given in Figure 13 where p is the existence probability, x, and

y are the recognition unit estimates and δx and δy are the transformations from the predefined set.

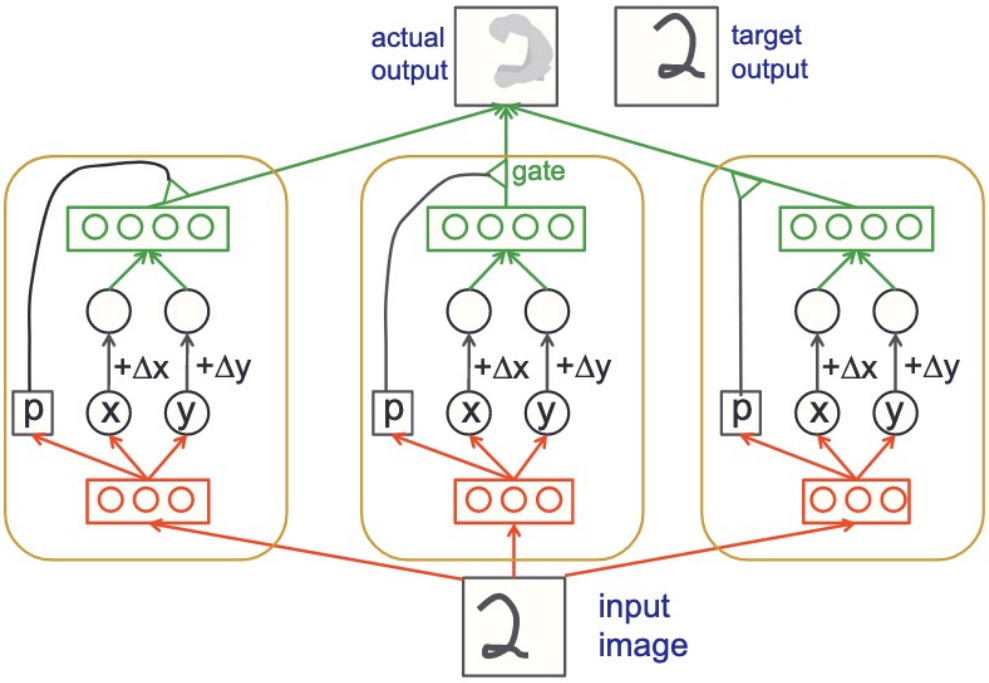


Figure 13: Illustration of transforming autoencoders that learns translations.[6]

Transforming Autoencoders are extended by learning the in-between transformations directly rather than learning a predefined by introducing the first routing by agreement method, via dynamic routing between capsules [24]. In this work, each capsule is a vector and the existence probability is defined by the magnitude of this vector. Dynamic routing models the part-whole relationship between the connected capsules at each layer by measuring the agreement among the child capsules about the feature vector of the parent. Dynamic routing between capsules has proved to be parameter-efficient, and it outperforms CNNs by a large margin where the input set has a large amount of visual transformations and overlapping digits on MNIST. Later introduced EM-Routing between Matrix Capsules [1] represented each capsule

as a matrix and activation pair. This reduced the number of parameters learned and removed the correlation between the existence probability and the feature vector. Additionally, the routing procedure is substituted with the Expectation-Maximization algorithm. These improvements yield a great performance upgrade in novel viewpoint generalization when compared to CNNs on a specific setup of the smallNORB dataset [38]. Both methods are covered in detail as they are closely related to *QCN* in Section 3.2.1.

Even though its success in minimal experiments, capsule networks are known for their problems with training stability and computationally expensive nature. Consequently, various routing mechanisms and capsule architectures have emerged. By aiming to improve the routing algorithm, Neural Network Encapsulation [39] proposed an approximation of the routing procedure with two branches to reduce the complexity. While the master branch receives information from its connected capsules, the aide branch provides information from the remainder capsules. In addition to these branches, a loss function is introduced to force higher-level capsules to reproduce the lower-level capsules.

On the other hand, Capsule Routing via Variational Bayes (*VBCaps*) [25] introduce Bayesian learning to the *EM-Routing* as a solution to variance-collapse singularities. *VBCaps* has priors and uncertainty on capsule parameters as the routing algorithm fits a mixture of transforming gaussians. This approach also allows to determine of active capsules automatically and control over the sparsity via capsule priors. Like *ALPACA*, *VBCaps* employ a reconstruction loss in its objective. Later, textitVBCaps is improved by Uncertainty in Capsule networks [32] that substitutes *VBCaps* which is local and iterative with the variational inference that benefits from latent variables that encourage the global context to be taken into the account in the objective function of the routing algorithm. Even though the speed gain and performance improvement, *VBCaps* is an iterative routing method and has matrix

pose representations in contrast to *ALPACA*.

The computational burden of the capsule networks is due to the iterations of the unsupervised clustering applied in the routing algorithm at each layer. Clustering may also be affected by the noise in the data. Acting upon these observations Self-routing Capsules (*SRCaps*) [40] remove the agreement from capsule layers. Inspired by the similarity between Mixture-of-Experts (MoE) and capsule networks, each capsule in *SRCaps* determine its votes and its routing coefficients itself. routing coefficients are obtained by a single layer perceptron in each capsule with a SoftMax function acting on the output. The parent capsule is determined by the weighted average of the incoming votes. This reduces the complexity of the routing procedure as it is replaced with a simple weighted average and the authors show that the performance of their model is not degrading when more layers are stacked. Other notable studies focusing on the routing procedure are Kernel Density Estimation (KDE) Routing [41] and Subspace Capsules [42, 43]. Attention-based routing procedures are explained in Section 3.4.1 since it is closely related to *ALPACA*.

Group Equivariant Capsule Networks [44] combine the Group Equivariant CNNs with capsule networks and prove that dynamic routing algorithms guarantee equivariance given certain constraints. Moreover, they provide spatial aggregation of receptive fields and combine them with group convolutions. On the other hand, Stacked Capsule Autoencoders (*SCAE*) [45] focus only on discovering parts and objects in a given image but provide unsupervised classification results which are state of the art. Each part capsule in *SCAE* represents rotations, translations, scale and shear, existence probability, and unique identity. On the other hand, object capsules are formed by combining the information in parts. During training, part capsules are also predicted by the object capsules.

In addition to the papers that address deficiencies of capsule networks. Initial papers led to several applications of Capsule Networks on different problems, such as

image segmentation [46, 47], video action detection [48], fashion image understanding [49, 50], forged image and video detection [51], hyperspectral image classification [52] and text classification [53, 54, 55].

2.3 Quaternions in Neural Networks

Quaternions are first proposed by Hamilton in 1883 as an extension of the complex numbers that represent an efficient way of computing rotations when compared to the Euler angles and rotation matrix. A quaternion q is a hyper-complex number that can be considered as a 4-dimensional vector but governed by different algebraic rules than the standard vectors. Quaternions are defined as in (3), where i, j, k denotes the imaginary axes.

$$q = q_0 + q_1 i + q_2 j + q_3 k, \quad q_0, q_1, q_2, q_3 \in \mathbb{R} \tag{3}$$

Quaternions are often divided into two parts, a scalar part s and a vector part $\mathbf{v}$ as given in (4), to be able to define algebraic operations in a compact form.

$$q = \begin{bmatrix} s_q, & \mathbf{v_q} \end{bmatrix} \tag{4}$$

The governing algebraic rules over quaternions are as follows:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{5}$$

$$ij = k, jk = i, ki = j, ji = -k, kj = -i, ik = -j \tag{6}$$

To formulate quaternion rotation, quaternion conjugate and quaternion product must be defined for two quaternions q and p.

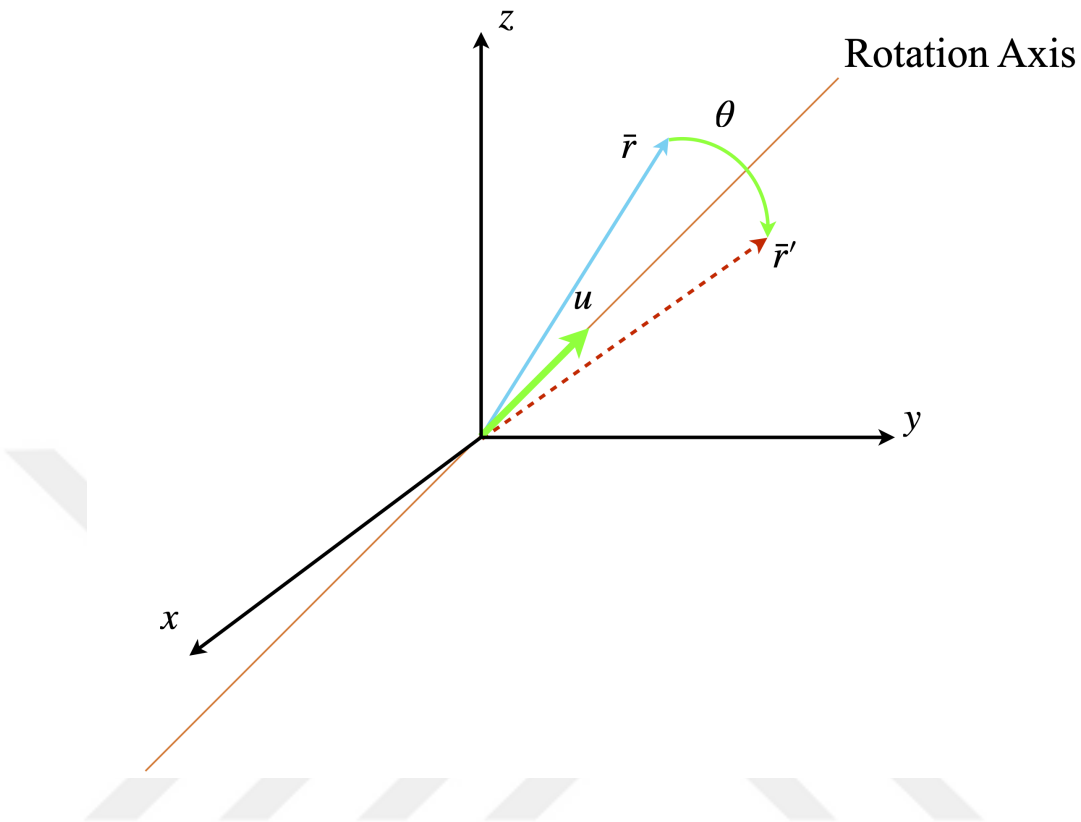


Figure 14: Illustration of Equation (10). Rotation of a pure quaternion $\bar{r}$ by θ along the rotation axis $\mathbf{u}$.

$$q = \begin{bmatrix} s_q, & \mathbf{v_q} \end{bmatrix}, \quad p = \begin{bmatrix} s_p, & \mathbf{v_p} \end{bmatrix} \tag{7}$$

Quaternion conjugate:

$$q^* = \begin{bmatrix} s_q, & -\mathbf{v_q} \end{bmatrix} \tag{8}$$

Quaternion product:

$$q * p = \begin{bmatrix} s_q s_p - < \mathbf{v_q}, \mathbf{v_p} > \\ s_q \mathbf{v_p} + s_p \mathbf{v_q} + \mathbf{v_q} \times \mathbf{v_p} \end{bmatrix}^T \tag{9}$$

where $< \mathbf{v_q}, \mathbf{v_p} >$ and $\mathbf{v_q} \times \mathbf{v_p}$ are the dot product and the cross product between vectors $\mathbf{v_q}$ and $\mathbf{v_p}$, respectively. The rotation of a quaternion $\bar{r}$ by $\frac{\theta}{2}$ around the axis $\mathbf{u}$ is represented in (10) (see Figure 14), where q is a unit vector in the form of (11), q^* is the quaternion conjugate and $\bar{r}$ is a pure quaternion whose scalar part equals to 0, as given in (12).

$$\bar{r}' = q * \bar{r} * q^* \tag{10}$$

$$q = \begin{bmatrix} \cos\theta, & \mathbf{u}\sin\theta \end{bmatrix} \tag{11}$$

$$\bar{r} = \begin{bmatrix} 0, & r \end{bmatrix} \tag{12}$$

For a valid quaternion rotation, where the rotation is by $\frac{\theta}{2}$ and the norm of $\bar{r}$ is preserved, q must be a unit quaternion and $\bar{r}$ must be a pure quaternion. Therefore, using quaternion rotation, instead of transformations in capsules, implicitly regularize the network weights by constraining them to satisfy unit quaternion property. Since quaternion average is only used in *ALPACA*, its definition, solution and equivariant properties are explained in Section 3.4.

Complex-valued neural networks have been an active research field in the last decades [56, 57], but with a limited influence until the applications on RNNs proving that complex-valued RNNs learn faster [58] and avoid vanishing gradients [59]. Moreover, earlier studies show that complex-valued neural networks are easier to optimize [60], and have better generalization characteristics [61]. One of the earliest studies of complex numbers in feed-forward NNs is proposed by Trabelsi *et al.* [62], namely Deep Complex Networks. Following up on this study, Deep Quaternion Network was proposed by defining quaternion convolution operation, but not utilizing

quaternion rotations [63]. On the other hand, another variant of Quaternion Convolutional Networks [64] represents each pixel in a colored image with a quaternion by considering RGB channels as imaginary parts. In this study, quaternion rotations are utilized with a constant rotation axis of $\sqrt{3}\begin{bmatrix} i & j & k \end{bmatrix}$, and thus only the rotation angles are learned during training. In the case of RNNs, it is demonstrated that QRNNs reduce the number of parameters by a wide margin on speech recognition tasks while maintaining the test accuracy [65], [66]. From a Computer Graphics perspective, quaternions are widely used for rotations due to their useful properties such as smooth rotation and computational efficiency [67]. Moreover, quaternions are closely related to the Capsule idea in the context of the part-whole relationship since an object can be represented by the part-whole transformations in Computer Graphics.

2.3.1 Motivation of the Thesis

Since their emergence, capsule networks have always been considered to be compatible with human vision systems and are believed to have potential as the empirical evidence suggests in the literature. However, research on capsule networks is still in its early stages and some major problems are yet to be solved. First of all, despite being data efficient in terms of training, capsule networks have a considerable computational burden due to the routing algorithm and learned transformations. Another well-known fact is that the training process of capsule networks is not stable particularly when a reconstruction of the input image is not used as a regularizer. Lastly, an extensive study on the novel viewpoint generalization performance of capsule networks when compared to state-of-the-art CNNs is yet to be found in the literature. Being motivated by the problems of capsule networks, this thesis aims to reduce the computational burden, increase stability and performance and thoroughly analyze the novel viewpoint generalization performance of capsule networks.

CHAPTER III

METHODOLOGY AND EXPERIMENTAL RESULTS

In this section, we first establish an abstract capsule notation to avoid confusion between *QCN* and *ALPACA*. In Section 3.2 we explain the Quaternion Capsule Networks [27] and provide experimental results published in this paper. The next two sections 3.3 and 3.4 focus on Generalization to Unseen Viewpoint Images of Objects via Alleviated Pose Attentive Capsule Agreement [28]. Section 3.3 describes the Novel ViewPoint Dataset (*NVPD*) while Section 3.4 describe and provide the experimental results of *ALPACA* and numerous state-of-the-art methods along with it on the *NVPD*.

3.1 Capsule Layers and Notation

The generalized capsule notation provided in this section roughly follows the notation in [39]. Before getting into the notations, a coarse view of the concepts and pipeline in a capsule layer is given in 15. Capsule-based methods covered in this thesis follow a similar architectural logic while the contents of these abstract concepts may differ. The first few layers of capsule networks are the so-called "capsule extractors" that provide the first capsule layers (primary capsules). These capsule extractors can be any type of feature extractors such as CNNs or transformers or fully connected networks. Common practice is to group the extracted features according to the capsule contents (*e.g.* vectors, matrices) to form the primary capsules. The last layer of the capsule network is called the class capsules, where the architecture has a capsule for each class in the dataset it has been trained on. Please note that the last layer depends on the task at hand, and the provided last layer is for image classification as it is focus of this thesis.

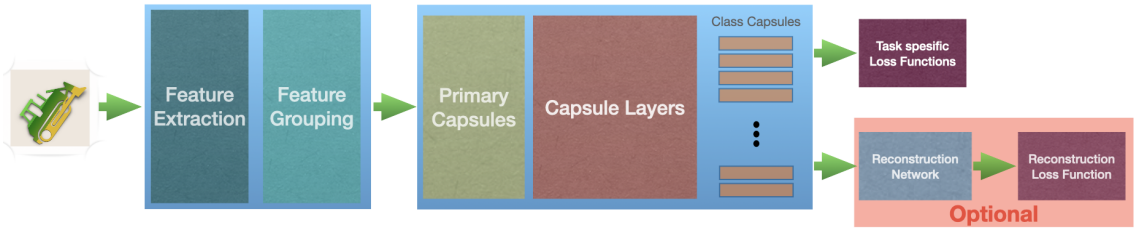


Figure 15: General structure of a typical capsule network architecture. Where the number of capsules in the last layer is equal to the classes in the dataset. Using a reconstruction loss commonly used in the literature as a regularizer if not in every architecture.

The set of capsules C_L in layer L is connected to the capsules in the next layer C_{L+1} via a routing algorithm $\Gamma(.)$ whose inputs are votes V_L calculated from C_L. Contents of a capsule in layer L, represented by c_L^i where $c_L^i \in C_L$, is not limited to any structure it can either represent a single property or multiple at once if it has a suitable routing algorithm. Some of the examples of capsule contents from the literature are, pose vector, pose matrix and an activation, feature vector, pose matrix and activation, or quaternion and activation. Class capsules in the last layer are denoted as c_κ, where $\kappa \in K$ is the class index and K is the set of all classes in the dataset. A capsule c_L^i only sends votes to its connected parents $v_{i|k}^L$ where these connections can be of any nature (*e.g.* convolutional, fully connected). Calculation of $v_{i|k}^L$ from capsule c_L^i and $\Gamma(.)$ is only limited by the backpropagation algorithm, therefore they need to be differentiable. Learned parameters of a capsule network are the vote calculation parameters and parameters of $\Gamma(.)$ if exist. Both the vote calculation and how they are handled via Γ depend on the capsule network, therefore they are explained specifically for *QCN* and *ALPACA* in the following sections.

3.2 Quaternion Capsule Networks

Capsule architectures, prior to Quaternion Capsule Networks (*QCN*), aimed to learn the linear manifold between the object and a 4×4 pose matrix or a feature vector

of its parts. However, this representation has various drawbacks. Particularly, the feature vector of an object and its part does not guarantee a linear manifold of the transformation space that capsule layers rely on while learning the transformations. In the case of using a pose matrix, to guarantee the linear manifold between capsules, it should be assumed that connected capsules only learn the rotations in-between higher and lower-level parts. From the perspective of a visual entity, all its parts should agree on the orientation and the origin of the coordinate system of this entity.

In the case of convolutional connections, the agreement on the origin (translation agreement) can be discarded with the assumption of the origin of a visual entity and all its parts are defined with respect to the center of the same receptive field, and hence the agreement on the origin is safe to ignore. When the visual entity does not have any deformable parts, there is a fixed mapping from each part to the parent entity in the form of rotation, as illustrated in Figure 16. Moreover, convolutionally connected Capsule Networks that only represent the orientation and its learned transformation set, which is restricted to the rotations, can still generalize to novel viewpoints. In this way, the computational burden and the size of the network are significantly reduced as the network seeks a solution in a smaller space in case of transformations. In parallel with this intuition, Stacked Capsule Autoencoders ($SCAE$) [45] represent the relationships in 3D space with 3×3 matrices.

In the case of the representation of the rotations, using quaternions have several advantages compared to the rotation matrix. First, quaternions do not suffer from gimbal lock [68, 69], as distinct from rotation matrices, Euler angles, and exponential maps. Second, rotation matrices must be orthogonal, and quaternions must be a unit vector for a proper representation of the rotation. However, neither can guarantee to keep this property after the weight updates. At this point, it is harder to ensure the orthogonality in the rotation matrix, and better to normalize quaternions. Additionally, in the previous studies, it is demonstrated that quaternions are successful

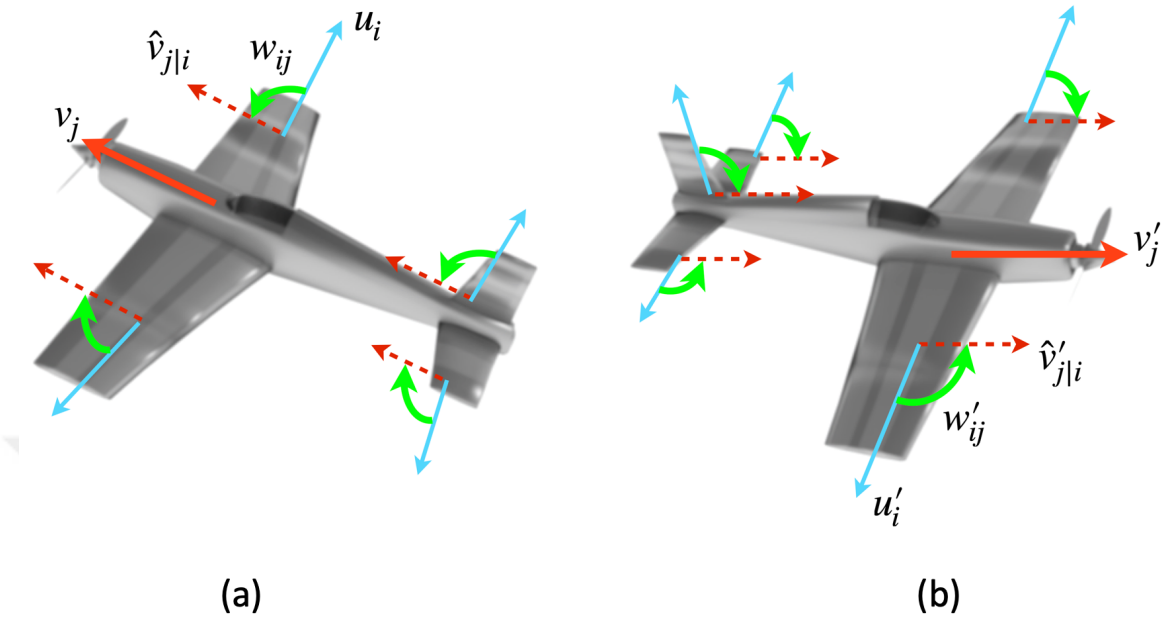


Figure 16: An illustration of rotations between child capsules and their parent object capsule. For both orientations of the plane in (a) and (b), the rotations between the child capsules and the parent capsule should stay the same ($w_{i|j} = w'_{i|j}$). **Blue:** Child capsule pose, **Red:** Parent capsule pose, **Dashed Red:** Votes for parent capsule, **Green:** Intrinsic rotation between child and parent poses.

in restoring the spatial relations [70], and can be extracted from images [62, 63]. Lastly, using quaternions reduces the number of parameters for each 3D rotation to be learned from 9 to 4.

*QCN*s are formed by the quaternion capsules where the capsule orientations and their rotations are governed by quaternion algebra. Quaternion capsules have three main contributions to both Capsule Network and Quaternion Network literature. First, capsule representation with quaternions is proposed, and mappings between them in quaternion algebra are formulated. Secondly, the rotation axis is a constant in the previous studies of feed-forward Quaternion Networks. However, *QCN*s have the ability to learn the rotation axis along with the rotation angle as in Recurrent Quaternion Neural Networks (*QRNN*) [65]. Lastly, extracting pose and activations from the input image is divided into two spatially aligned branches that allow using

different sub-networks with different complexities for different modalities. Experimental results show that Quaternion Capsules achieve state-of-the-art performance in case of generalization to novel viewpoints. *QCN*s also attains on-par performance with Matrix Capsules on well-known benchmarking datasets with only half of the parameters without any hyper-parameter tuning or architecture changes with respect to the dataset.

The following sections are structured as follows, Section 3.2.1 provides the literature that is closely related to *QCN*s. Vote calculation and the routing procedure are explained in Section 3.2.2 while the model architecture is explained in 3.2.3. Lastly, experimental results are given in 3.2.5.

3.2.1 Related Work

*QCN*s combines quaternion rotations and capsule networks by forming capsules as quaternions which has several advantages. First, exploiting the partial equivariance of convolutions to translations, we reduced the capsule representation and learned transformations only to pose and rotations respectively. Where one can result in 3×3 matrix representation as in Stacked Capsule Autoencoders (*SCAE*) [45]. Through this assumption, capsules and learned rotations in *QCN*s are quaternions that only require 4 parameters, therefore reducing the required parameters in the network to represent pose and transformations, when compared to 3×3 matrix in *SCAE* or 4×4 matrix in Matrix Capsules that has 9 and 16 parameters respectively. Moreover, due to the rotation formulation of quaternions, the network is implicitly regularized by using quaternions. In quaternion rotation, rotor quaternions, which are learnable parameters in Quaternion Capsule Networks, must be unit quaternions. Therefore, a simpler version of weight normalization [71] must be applied to the weights to ensure proper representation of the rotation. As a side effect of its computationally efficient nature as well as the immunity to ambiguity and gimbal lock, quaternions are better

suited for optimization, in contrast to Euler angles.

*QCN*s aim to improve Dynamic routing between Capsules (*Dynamic-Routing*) [24] and Matrix Capsules with *EM-Routing* [1] that can learn the transformations between parts and the whole object via back-propagation. *QCN*s utilize the convolutional connections and the exact routing procedure in *EM-Routing*. Considering *EM-Routing* is a direct improvement over *Dynamic-Routing*, both methods are covered in the following sections.

3.2.1.1 Dynamic Routing Between Capsules

The first proposition to learn these transformations are Dynamic routing between Capsules whose capsules consist of a feature vector $c_L^i = \left\{ f_i^L \right\}$. These vectors are transformed by learned weights $W_{i|j}^L$ to calculate votes $v_{j|i}^L$ and parent features are calculated by squashing (Eq. 13) the weighted sum of the incoming votes given in 14. Weights $\gamma_{i|j}^L$ are called coupling coefficients and instantiated by a SoftMax over logits, b_{ij} often initialized to 0 at each routing process, as defined in Eq. 15. Dimensionality d_L of capsule vectors in layer L is a hyper-parameter, therefore learned transformations $W_{i|j}^L$ is a $d_L \times d_{L+1}$ matrix.

$$f_j^{L+1} = \frac{\left\| s_j^{L+1} \right\|^2}{(1 + \left\| s_j^{L+1} \right\|^2)} \frac{s_j^{L+1}}{\left\| s_j^{L+1} \right\|} \tag{13}$$

$$v_{j|i}^L = W_{i|j}^L f_i^L, \quad s_j^{L+1} = \sum_i \gamma_{i|j}^L v_{j|i}^L \tag{14}$$

$$\gamma_{i|j}^L = \frac{exp(b_{ij})}{\sum_j exp(b_{ij})} \tag{15}$$

$$a_{i|j}^L = f_j^{L+1} . v_{j|i}^L \tag{16}$$

Dynamic-Routing is an iterative clustering algorithm over the votes, that weights

the incoming votes according to their similarity with their parents. The measure of similarity is chosen to be cosine similarity between the parent feature vector f_k^{L+1} and a vote $v_{i|k}^L$ (Eq. 16). This architecture is fully connected at all capsule layers, therefore, a child capsule votes to all the capsules in the next layer which essentially means that $k = j$. Proposed *Dynamic-Routing* runs for 3 iterations, it is shown that running for more iterations does not provide a significant improvement [24]. Steps of *Dynamic-Routing* is given in Alg. 1.

Algorithm 1: $DynamicRouting(v_{j|i}^L)$

$\forall b_{ij} \leftarrow 0$

for r *iterations* **do**

$$\forall \gamma_{i|j}^L \leftarrow \frac{exp(b_{ij})}{\sum_j exp(b_{ij})} \tag{15}$$

$$\forall s_j^{L+1} \leftarrow \sum_i \gamma_{i|j}^L v_{j|i}^L \tag{14}$$

$$f_j^{L+1} = \frac{\left\|s_j^{L+1}\right\|^2}{(1+\left\|s_j^{L+1}\right\|^2)} \frac{s_j^{L+1}}{\left\|s_j^{L+1}\right\|} \tag{13}$$

$\forall b_{ij} \leftarrow v_{j|i}^L . f_j^{L+1}$

end

return $C_{L+1} = \left\{ \left[f_j^{L+1} \right] \right\}_j^{n_{L+1}}$

Dynamic-Routing minimize Margin Loss given in Eq. 17. As the magnitudes of each class capsule f_κ is considered to be its existence probability, Margin Loss aim to increase the difference in magnitudes of feature vectors between the ground truth class capsule and other capsules.

$$L = \sum_\kappa^K \{T_c(max(0, m^+ - \|f_\kappa\|)^2 + \lambda(1 - T_\kappa)max(0, \|f_\kappa\| - m^-)^2\} \tag{17}$$

In Eq. 17, T_κ is the existence indicator, meaning that, $T_\kappa = 1$ if κ is the ground truth class, else $T_\kappa = 0$. While λ is the weight for negative (non-existent) classes in each image, m^+ and m^- are the margins for positive and negative classes. Weighting

parameter λ, margins m^+ and m^- are set to 0.5, 0.9 and 0.1 respectively in [24] that experiment on MNIST dataset.

3.2.1.2 Matrix Capsules with EM-Routing

In contrast, capsules consist of feature vectors in Dynamic Routing between Capsules, Matrix Capsules with EM-Routing is composed of a 4 by 4 pose matrix M and a scalar value that represents the activation $a_i \in \mathbf{a_L}$ of that capsule ($c_L^i = \left\{M_i^L, a_i^L\right\}$). The main purpose of using a matrix is to reduce the number of parameters in the learned transformations to calculate the votes. Matrices are also one of the well-known tools for pose representations and transformations. The squash function to calculate the existence probability of a capsule was replaced with an activation term a_i^L that allows typical objective functions to minimize during routing [1]. Additionally, cosine similarity is not suited for differentiating between good agreements, therefore in *EM-Routing* variance of a Gaussian cluster is utilized. Lastly, capsule layers are convolutionally connected in *EM-Routing*. Votes $v_{j|i}^L \in V_L$ are calculated via the matrix transformation in Eq. 18.

$$v_{j|i}^L = W_{i|j}^L M_i^L \tag{18}$$

EM-Routing is an expectation maximization *EM* based clustering of the incoming votes to each parent capsule. Similar to *EM*, it has expectation and maximization steps (M-step and E-step). The general structure of the algorithm is given in Alg. 2. *EM-Routing* fits Gaussian distributions on each element ${}^h v_{i|j}^L$ of the incoming votes where $h \in H$ and H is 16 as each vote is a 4×4 pose matrix calculated from Eq. 18. Activations a_j of parent capsules are dependent on the variance μ_j^h over each element on the incoming votes, where β_u and β_a are learned parameters. Lastly, R_{ij} refers to the routing weights and is equally initialized for each parent capsule at each forward pass. *EM-Routing* also often runs for 3 iterations at maximum, in [1], the effect of

Algorithm 2: $EM - Routing(a, V)$

$\forall b_{ij} \leftarrow 0$

$\forall i \in C_L, j \in C_{L+1} : R_{ij} \leftarrow 1/|C_{L+1}|$

for r *iterations* **do**

  $\forall \in C_{L+1}$:M-STEP$(\mathbf{a_L}, R, V_L)$

  $\forall \in C_L$:E-STEP$(\mu, \sigma, \mathbf{a_L}, V_L)$

end

return $C_{L+1} = \left\{ \left[M_j^{L+1}, a_j\right] \right\}_j^{n_{L+1}}$

`M-step(`$\mathbf{a_L}, R, V_L, j$`):`

  $\forall i \in C_L : R_{ij} \leftarrow R_{ij} * a_i$

  $\forall h : \mu_j^h \leftarrow \frac{\sum_i R_{ij}\, {}^h v_{i|j}^L}{\sum_i R_{ij}}$

  $\forall h : (\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} ({}^h v_{i|j}^L - \mu_j^h)^2}{\sum_i R_{ij}}$

  $cost^h \leftarrow (\beta_u + log(\mu_j^h)) \sum_i R_{ij}$

  $a_j \leftarrow sigmoid(\lambda(\beta_a - \sum_h cost^h))$

  return

`E-step(`$\mu, \sigma, \mathbf{a_L}, V_L$`):`

  $\forall j \in C_{L+1} : p_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} \exp\left(-\sum_h^H \frac{({}^h v_{i|j}^L - \mu_j^h)^2}{2(\sigma_j^h)^2}\right)$

  $\forall j \in C_{L+1} : R_{ij} \leftarrow \frac{a_j p_j}{\sum_{m \in C_{L+1}} a_m p_m}$

  return

more iterations is discussed and empirically found that more than 4 iterations are obsolete. Moreover, *EM-Routing* is very slow therefore each iteration has a drastic effect on the run time.

EM-Routing minimize Spread loss instead of margin loss that aims to increase the discrepancy between the ground truth class capsule activation and other class capsule activations in the last layer, the formal definition is provided in Eq. 19. Practical

details of how the parameter m is scheduled during the training of QCNs are provided in Section 3.2.5.

$$L = \sum_{i \neq t} (max(0, m - (a_t - a_i)))^2, \quad L = \sum_{i \neq t} L_i \tag{19}$$

3.2.2 QCN Capsules

A quaternion capsule in layer L is defined as $c_L = [q_i, a_i]$ where q_i is a pure quaternion ($q_i = \begin{bmatrix} 0, & \bar{\mathbf{q}}_\mathbf{i} \end{bmatrix}$) and a_i is the activation. The votes for parent capsules are calculated by quaternion rotation operation in (10). Learned quaternion rotations between capsules are denoted as $w_{i|j}$. Thus, the vote from i^{th} capsule in layer L to j^{th} capsule in layer $L+1$ is calculated as follows:

$$v_{i|j} = w_{i|j} * q_i * w_{i|j}^* \tag{20}$$

To comply with the constraint of quaternion rotation, $\bar{w}_{i|j}$ must be a unit quaternion, therefore is defined as given in (21).

$$w_{i|j} = \begin{bmatrix} \cos\theta_{i|j}, & \sin\theta_{i|j} \frac{\bar{\mathbf{w}}_{\mathbf{i|j}}}{\left\| \bar{\mathbf{w}}_{\mathbf{i|j}} \right\|} \end{bmatrix} \tag{21}$$

where $\bar{\mathbf{w}} = \begin{bmatrix} w_1 i & w_2 j & w_3 k \end{bmatrix}$ which refers to a similar weight normalization procedure as in [71] by re-parameterizing the weights where the scale of the weight vector is set to 1 to ensure that the weight vector is a unit vector in the rotor quaternion. Moreover, θ_w, w_1, w_2, and w_3 for each capsule are learned via backpropagation during training. Since the division of θ by 2 in the original quaternion formulation is a constant, it is discarded on the weight formulation. Note that Quaternion Capsules can learn the rotation axis $\bar{w}_{i|j}$, in addition to the rotation angle $\theta_{i|j}$, which is a novel feature for Quaternion Networks. Finally, once the votes are obtained, the parent capsules $\begin{bmatrix} v_{i|j}, a_j \end{bmatrix}$ are calculated by EM Routing in Alg. 2.

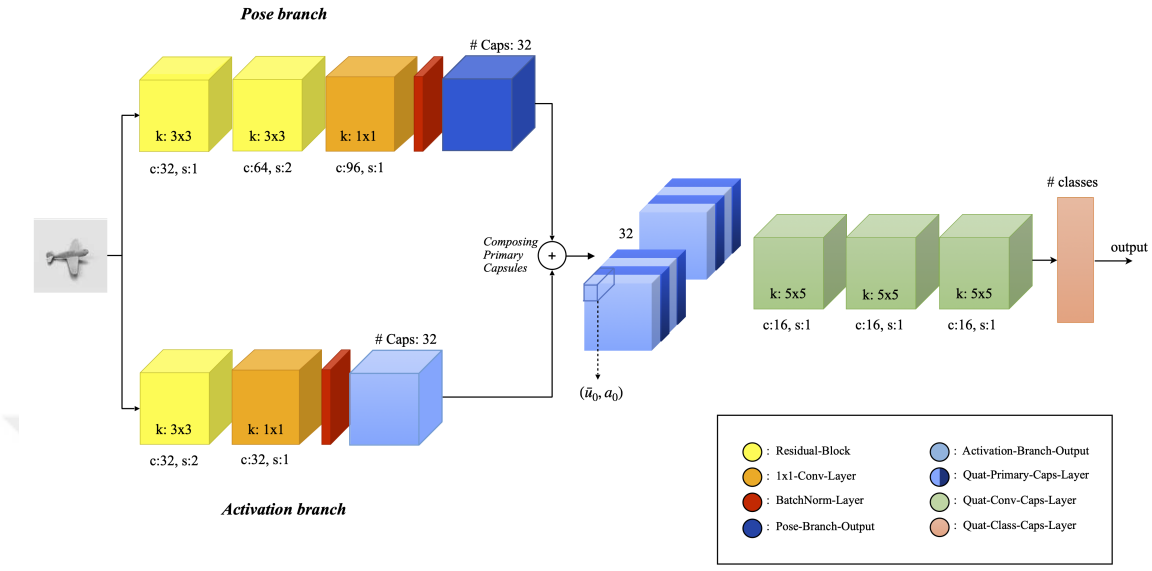


Figure 17: Overview of QCN architecture. The input image is processed on Pose and Activation branches separately, later to be merged to compose Primary capsules. Starting from Primary capsules, the network consists of three convolutional and one fully connected capsule layer.

3.2.3 Architecture

In the literature, the activations and pose information for capsules are extracted by the same network even though the pose of a visual entity is independent of its existence probability. Our architecture inherently differs from other Capsule Networks in this manner. Instead of using two convolutional layers as in the original architecture [1], pose q_i and activations a_i are extracted with the help of isolated branches that output spatially aligned activations and pose for each capsule, as illustrated in Figure 17. This design allows us to independently specify the size, or even the type of the network for pose and activations before constructing Primary capsules. In our design, a relatively deeper network is used in the pose branch since extracting pose information may be assumed to be a more complex problem than extracting the activations. Spatial alignment of the branches is ensured by keeping kernel sizes and

strides coherent for both branches.

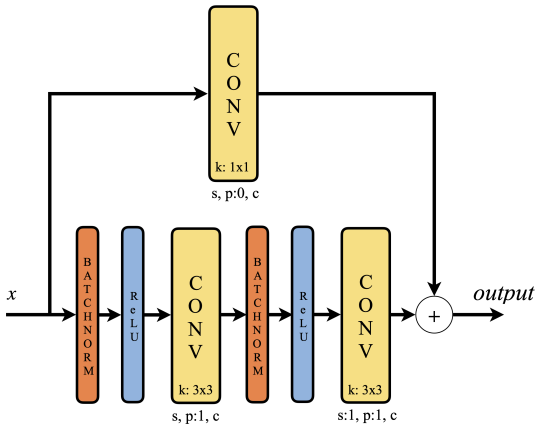


Figure 18: Illustration of a single residual block used in our design. Stride s of the residual block is applied in the first convolutional layer and in the skip connection. The number of channels c of the residual block is applied on each convolutional layer with the same padding p.

Pose branch consists of two residual blocks followed by a *1x1* convolutional layer and a batch normalization. Residual blocks are designed as the same blocks that learn the imaginary parts in Deep Complex Networks [62]. The detailed view of a single residual block is given in Figure 18. While both residual blocks have *3x3* kernels, the first one has 32 channels with stride 1, whereas the second has 64 channels with stride 2. The following *1x1* convolutional layer increases the dimensionality in feature space to match the dimensionality of Primary capsules. Since capsule poses are represented as pure quaternions requiring 3 imaginary parts, this layer has $N \times 3$ channels where N refers to the number of capsules in the primary layer and is set to 96. Then, batch normalization [72] is applied to the output before forming Primary capsules.

Activation branch is relatively shallower as assumed that the existence probability can be simply extracted. Therefore, only one residual block is employed before the same pipeline in *Pose branch* with N channels. To ensure that the spatial alignment with *Pose branch* is preserved, the residual block has *3x3* kernel with stride 2. The number of channels in this block is set to 32.

Pose and activation outputs corresponding to the same receptive fields are grouped to compose Primary capsules $[q_i, a_i]$. Thus, each Primary capsule refers to the pose and activation of a visual entity in a particular receptive field. Consecutive 3 convolutional capsule layers have 16 capsules and 1 stride with *5x5* kernels. The last convolutional layer is fully connected to the Class capsule layer where the number of capsules is dependent on the number of classes in the dataset (*e.g* for MNIST and smallNORB, the number of capsules are 10 and 5, respectively). All capsule layers are connected by EM Routing with 2 routing iterations and the spread loss (19) is used without any manipulation on the original version in [1].

3.2.4 Implementation Details

The implementation details regarding *QCN*s are explained in this section. First, a quaternion $q = q_0 + q_1 i + q_2 j + q_3 k$ is isomorphic to real-valued matrices ${}^{R}Q$ and ${}^{L}Q$ in case of quaternion product from the right and left, as can be seen in (22).

$$ {}^{R}Q = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix}, {}^{L}Q = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \tag{22} $$

Given these embeddings, vote calculation with quaternion rotation in (20) can be performed in the form of matrix multiplication as given in (23), where ${}^{L}W^{*}_{i|j}$ represent the quaternion rotor conjugate embedded to product from right matrix, and ${}^{R}W_{i|j}$ represent quaternion rotor embedded to product from left matrix.

$$ v_{i|j} = {}^{L}W^{*}_{i|j} {}^{R}W_{i|j} q_i \tag{23} $$

The residual blocks in the activation and pose branches are initialized with uniform Kaiming distribution [13], while *1x1* convolutional layers that follow the residual

blocks in both branches are initialized with uniform Xavier distribution [73]. Each element of the rotation axes $\bar{w}_{i|j}$ in quaternion capsule layers is initialized from $U(-1, 1)$. Initialization of the transformation matrices to identity with noise on off-diagonals [25] proves to stabilize the training by restricting the amount of transformation in the earlier stages. This is analogous to the idea of restricting the initial rotations in *QCN*s to a certain range. Therefore, $\theta_{i|j}$ is initialized by the uniform distribution in the range of $[-\pi, \pi]$ as in [65]. Another important detail is the m scheduling in Spread loss. As stated in [74], m is updated by (24) until m reaches 0.9, where σ is the sigmoid function. In all experiments of *QCN*, z is set to 50000 as stated by the authors in paper discussions.

$$m = 0.2 + 0.79 * \sigma\left(min\left(10, \frac{step}{z} - 4\right)\right) \tag{24}$$

3.2.5 Experiments

As the main goal of *QCN* is to achieve better generalization to the novel viewpoints, the experiments on smallNORB are conducted on a setup where training is made on a limited range of viewpoints. Additionally, benchmarking experiments are conducted on smallNORB, MNIST, FashionMNIST, SVHN, and CIFAR10 datasets to show that QCN can achieve on-par performances with Matrix Capsules on multiple datasets. These datasets are chosen with respect to the presented Capsule Network results in the literature [24, 1, 25]. Benchmarking results also contain the results of our implementation and the open-source IBM implementation [74]. In our design, the architecture and hyper-parameters (*e.g* optimizer, initial learning rate, a learning rate scheduler, etc.) are kept the same with our implementation of Matrix Capsules for all experiments. The only factor that may change is the batch size depending on the memory requirements of the datasets.

Table 1: Comparison of QCN test error rates with the reported results of Matrix Capsules (EM), IBM's (EM-IBM) and our (EM*) implementations of Matrix Capsules, and Capsule Routing via Variational Bayes (VB). −: Not reported, ‡: We run their open-source code on the corresponding dataset with their default hyper-parameters.

Models	**smallNORB**		**MNIST**		**FashionMNIST**		**SHVN**		**CIFAR-10**	
	Error (%)	# of Params	Error (%)	# of Params	Error (%)	# of Params	Error (%)	# of Params	Error (%)	# of Params
EM [1]	1.8	~310K	0.44	–	–	–	–	–	11.9	–
EM-IBM [74]	4.6	~335K	1.23‡	~337K‡	10.44‡	~337K‡	–	–	–	–
VB [25]	1.6	~169K	–	–	5.2	~172K	3.9	~323K	11.2	~323K
EM*	3.40	~317K	0.89	~319K	9.74	~319K	8.19	~320K	17.76	~460K
QCN	2.29	~188K	0.37	~187K	6.92	~187K	4.63	~189K	13.92	~189K

In this study, even though the experimental results show that QCN achieves state-of-the-art performance in case of generalization to novel viewpoints, the main purpose is to make a proof-of-concept design that quaternions form a compact way of representing the rotations and orientations for capsules rather than achieving the state-of-the-art performances in Capsule Network literature. Therefore, *QCN*s are not fine-tuned for any of the datasets, instead, the configuration that achieves the best performance on generalization to novel viewpoints setup is used.

3.2.5.1 Generalization to Novel Viewpoints

The experiments of generalization to novel viewpoints are conducted on smallNORB dataset with a given setup in [1]. The smallNORB [38] is a 3D object recognition dataset that consists of 96×96 stereo image samples of 5 classes under 6 lighting conditions, 9 elevations (*i.e.* every 5 degrees from 30 to 70 degrees), and 18 azimuths (*i.e.* every 20 degrees from 0 to 340). In this setup, there are two different cases in which the dataset is reserved with respect to azimuth angles and elevations, instead of instances. For azimuth experiments, the azimuth angles of (300, 320, 340, 0, 20, 40) are used for training, and the tests are made on the remaining azimuth angles. Secondly, QCN is trained on 3 smaller elevations (30, 35, and 40 degrees from the horizontal), and tested on the remaining 6 larger elevations (45, 50, 55, 60, 65, and 70 degrees from the horizontal).

Table 2: Test error rate comparison of the reported results on the original paper (EM), Capsule Routing via Variational Bayes (VB), our implementation (EM*) of Matrix Capsules, proposed Quaternion Capsules (QCN) and CNN results in [1] on novel viewpoint setup (*i.e.* azimuth and elevation).

Viewpoints	**Azimuth** (%)				
(Models)	QCN	EM*	VB	EM	CNN
Novel	**7.5**	13.4	11.3	13.5	20.0
Familiar	3.7	3.7	3.7	3.7	3.7
Viewpoints	**Elevation** (%)				
(Models)	QCN	EM*	VB	EM	CNN
Novel	**11.5**	15.8	11.6	12.3	17.8
Familiar	4.4	4.0	4.3	4.3	4.3

For the sake of fair comparison, the test set (novel viewpoints) accuracy is measured once the training set (familiar viewpoints) accuracy is matched with the reported values in [1]. QCN outperforms both Matrix Capsules with EM Routing (EM) and Capsule Routing via Variational Bayes (VB) with a significant performance improvement for both azimuth and elevation setups, when compared to VB and EM, respectively.

*QCN*s outperform both Matrix Capsules with EM Routing (EM) and Capsule Routing via Variational Bayes (VB) in both azimuth and elevation setups. A significant performance improvement of 3.75% and 5.92% in the case of azimuth angles is observed when compared to VB and EM, respectively. On the other hand, in the case of elevation, *QCN*s outperform EM and VB by 0.84% and 0.13%, respectively.

The baseline CNN results reported in [1] is outperformed by all the capsule architectures by a large margin. This baseline architecture contains two convolutional layers with 32 and 64 channels followed by a fully connected layer with 1024 neurons and summing up to 4.2M parameters.

3.2.5.2 Results on Common Datasets

smallNORB: In the default settings of smallNORB [38], training and test sets contain 23,400 samples where each class has 10 different instances. The original format of the dataset has a standard classification setup where test set instances are not seen during training. During training, samples are initially resized to 48×48, and 32×32 random patches of binocular images are fed to the network with a batch size of 64. Center cropping is applied to the test images. QCN achieves a 2.3% test error rate with almost half of the parameters as in Matrix Capsules. Even though QCN surpasses IBM's and our implementations of Matrix Capsules, it falls behind the accuracy reported in the official paper by 0.6%.

MNIST: Experiments on MNIST are conducted on the standard setup without any modification. Even though QCN specializes in rotations, it can also achieve on-par classification performance with Matrix Capsules on MNIST. QCN achieves a 0.37% test error rate surpassing both performances in our implementation and as reported in [1], which are 0.89% and 0.44%. Note that MNIST is a toy problem, rather than being distinctive in terms of test performances. However, the fact that QCN surpasses Matrix Capsules, indicates that quaternions have at least the same capability to achieve such performance with less parameters and a more compact representation.

FashionMNIST: FashionMNIST [75] can be considered as a more difficult dataset than the previous ones whose properties are the same as MNIST, except the number of samples in each split (*i.e.* 60,000 training and 10,000 test samples with the size of 28×28). During the training of QCN and our Matrix Capsules implementation, any augmentation technique is applied to the samples. As a result, QCN, with a relatively shallow network, achieves a 6.92% test error rate, which is slightly worse than VB which achieves 5.2%. On the other hand, our implementation of Matrix Capsules achieves only a 9.74% test error rate falling behind QCN by a wide margin.

Table 3: The effect of branching on the error rates and parameters for both Matrix and Quaternion Capsules. Non-branched versions of EM‡ and QCN have two consecutive residual blocks with 64 and 96 channels until Primary capsules to ensure Primary capsules have the same number of input channels in both for a fair comparison.

Models	**Error Rate (%)**			**~# Parameters**		
	with	without	diff.	with	without	diff.
EM‡	3.66	3.44	0.22	411K	533K	-122K
QCN	2.29	3.72	**-1.43**	188K	298K	-110K

SVHN: SVHN [76] consists of RGB real-world house number images with 10 classes for each digit. Experiments are conducted on 76,257 samples and tested on 26,032 samples. On both sets, the samples are scaled to 32×32, and normalized before feeding to the network. While QCN achieves a 4.63% test error rate, VB and our implementation of Matrix Capsules achieve 3.9% and 8.19% test error rates, respectively.

CIFAR-10: CIFAR-10 [77] consists of 6000 RGB images per each of 10 classes, where training and test sets have 50,000 and 10,000 samples with the size of 32×32. Training samples are zero-padded by 4 pixels and randomly cropped to 32×32 patches. Lastly, horizontal flipping is applied to all training samples before feeding them to the network. Note that test samples are not modified. QCN achieves a 13.9% test error rate while [1] reported an 11.9% test error rate, which is achieved by increasing the number of neurons in the hidden layer to 256. However, in our main design, the architecture is identical for all datasets. Additionally, our implementation of Matrix Capsules with 256 neurons in the hidden layer achieves a 17.76% test error rate.

Effect of branching: Since we have used the complex component extractor as in [62], it is important to determine its effects on the performance of capsules. Therefore, we have conducted additional experiments on smallNORB with branched and non-branched versions of QCN and Matrix Capsules. Note that the branched version

of Matrix Capsules consists of the same capsule extractor layers as in QCN, while non-branched versions have two consecutive residual blocks with 64 and 96 channels, as shown in Figure 18. Therefore, we have conducted additional experiments on smallNORB by employing Matrix Capsules that have the same capsule extractor layers and branching mechanism with *QCN*s with and without branching. The results demonstrate that using separate branches for pose and activation increases the overall performance of QCN, though the number of parameters is reduced. For QCN, this is expected as we use the blocks that are empirically proven to be useful for extracting the complex components [62]. On the other hand, EM[‡] yields a slightly larger error rate when branching is applied.

3.3 Novel Viewpoint Dataset (NVPD)

Capsule networks are designed for generalization to novel viewpoints and are mainly used for 2D images. However, they are only tested on smallNORB dataset [38] since it was the only dataset at the time that explicitly allows testing novel viewpoint generalization in a controlled setup. Even though being an ideal dataset for developing recognition models due to the lack of texture and background, smallNORB does not provide negative elevation angles and different distances in addition to the small number of classes. Moreover, the novel viewpoint setup in capsule literature provides only azimuth and elevation splits where baseline CNNs cannot match the test set performance of capsule networks [1]. Another recently published controlled dataset iLab2M [2] consists of colored images with object textures and backgrounds. iLab2M is particularly designed to analyze the effect of visual variations with many samples but only has vehicle object classes. In contrast to smallNORB, iLab2M only has a single distance and positive elevation angles. Despite having more samples, iLab2M provides random splits of the object instances rather than focusing on different types of viewpoint splits. In contrast to these datasets, *NVPD* provide 8 meaningful viewpoint

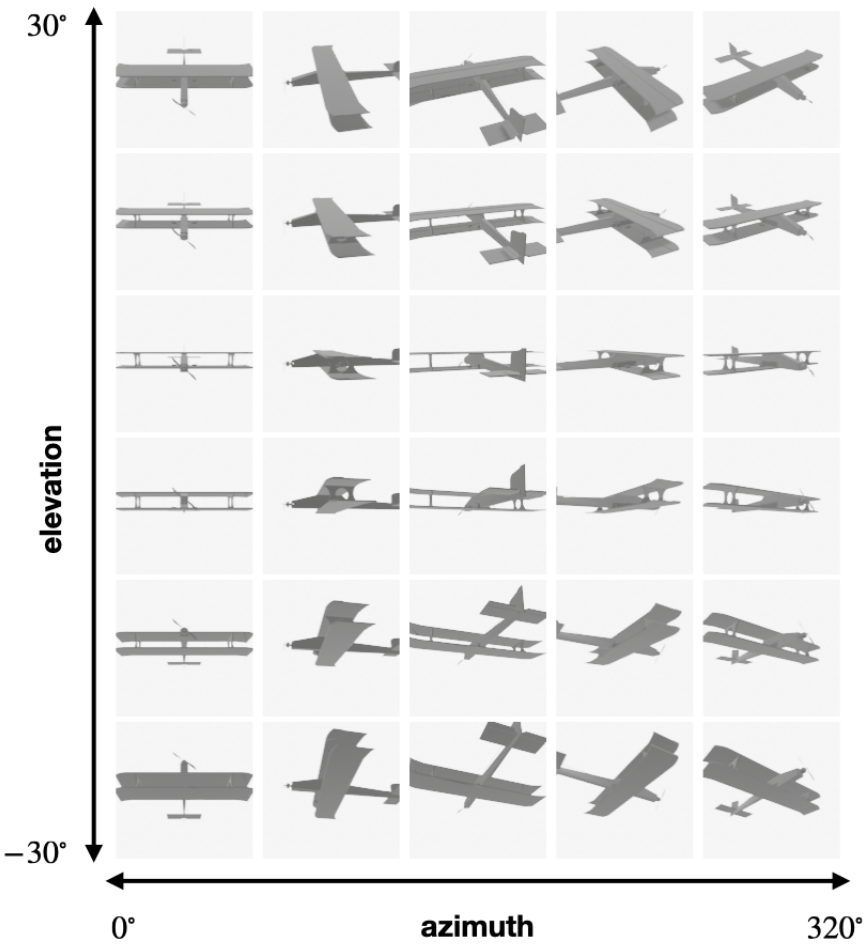


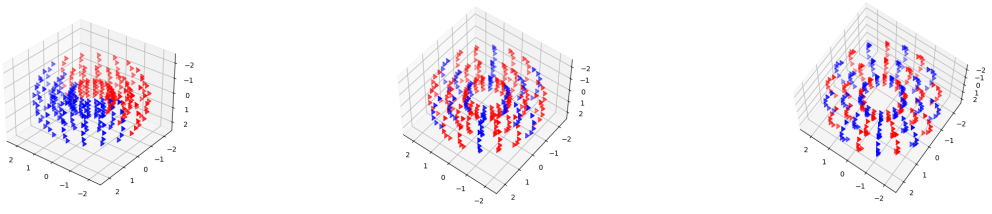
Figure 19: Examples of a single object in novel viewpoint dataset from different elevation and azimuth angles.

splits to particularly test harder setups and try to understand where the algorithms fail. *NVPD* also has a larger variety of object classes with symmetric shapes (*e.g.,* lambs and cups). Lastly, *NVPD* has negative elevation angles and 3 distances that pose a harder problem in the context of viewpoint generalization.

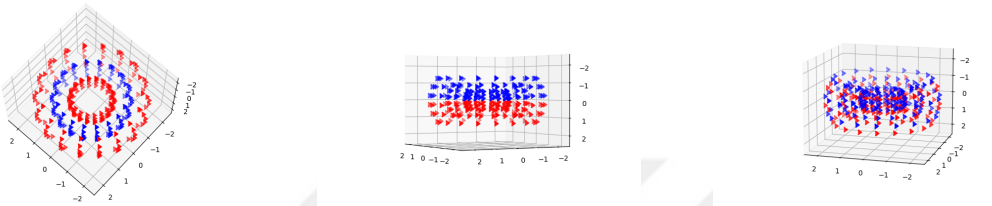
Our experimental design in the remainder of this thesis focuses on testing the generalization capability of a given model to unseen viewpoints of objects in a 2D image recognition task. Unfortunately, there are insufficient publications in the literature that particularly test novel viewpoint generalizations of a model in a controlled setup. Meaning that the training set and test set do not have overlapping viewpoints.

For this purpose, we introduce a controlled dataset, namely *Novel ViewPoint Dataset* (*NVPD*), that allows us to split the dataset in various setups to investigate whether a given method fails in generalizing to distance, azimuth or elevation differences or how the variance of the viewpoints that are available during the training affects the performance. An ideal dataset for development and thorough analysis would require numerous experiments, thus we aimed for a lightweight but also challenging dataset. The accuracy of the state-of-the-art methods on *NVPD* does not saturate at maximum, hence, we can compare the performances. In addition to *NVPD*, we tested on the iLab2M dataset where the training and test samples do not contain the same object instance. The results of the iLab2M dataset allow us to observe the capability of a given model to recognize an object category regardless of its instance properties such as texture or small shape differences.

NVPD consists of 10 classes, 50 instances per class, and 324 different viewpoints per instance. Viewpoints are distributed as 18 azimuth angles, 6 elevations, and 3 distances. Azimuth angles are selected in the range of 0 to 360 degrees, whereas the elevations are picked from -30 to 30 degrees with 12-degree steps, which are illustrated in Figure 19. On the other hand, the distance starts from 1 unit distance to 2.5 unit distance at every 0.75. The dataset is generated from the ShapeNet [78] samples by rendering 2D images from every viewpoint for each object instance. From the 51 classes in ShapeNet, we have picked 10 classes which are *airplane, car, chair, guitar, lamp, motorbike, mug, sofa, table, and train.* These classes are picked by prioritizing the model qualities, and we deliberately included some symmetrical and similarly shaped objects such as a mug and lamb. For each class, 50 instances are sampled to ensure a balanced distribution over the classes. Note that all model sizes are normalized before rendering, meaning that two different models have relatively similar sizes at the same distance levels regardless of their actual size. All objects have no texture, but a default material, plain white background, and the scenes have



(a) [A] Dark side (b) [A] Step-Larger (c) [A] Step-over



(d) [D] Step-over (e) [E] Dark side (f) [E] Step-over

(g) [R] Even (h) [R] One-Third

Figure 20: Training and test set splits for *NVPD*. Each dot represents a viewpoint and the center of the graph represents where the object is stationed. Blue and red viewpoints represent training and test samples, respectively. [A], [E], [D], and [R] refers to azimuth, elevation, distance, and random split axis. Step-larger means skipping two viewpoints in split axis [A] and [E].

exactly the same light sources to ensure each object is rendered in the same conditions.

Lastly, depth maps and camera orientations are also available in *NVPD*. However, all the compared capsule networks did not utilize such information, therefore they are disregarded during our training for a fair comparison.

3.3.1 *NVPD* Settings

There are 8 different settings in our experiments and 6 of them focus on splitting a particular axis (*i.e.* azimuth [A], elevation [E], and distance [D]). There are 4 types of different splits. Dark side split is available for azimuth and elevation ([A] Dark side

Table 4: Novel ViewPoint Dataset (*NVPD*) training and test sample numbers for each setting.

	[A] Dark side	[A] Step-over	[A] Step-larger	[E] Dark side	[E] Step-over	[D] Step-over	[R] Even	[R] One-Third
Train samples	81000	81000	54000	81000	81000	54000	81000	54000
Test samples	81000	81000	108000	81000	81000	108000	81000	108000

and [E] Dark side), where the axis is split from the center to obtain test and training sets. In the azimuth axis, training samples consist of viewpoints that are 180 degrees azimuth angle whereas, in the elevation axis viewpoints with positive angles form the training set. Step-over and Step-larger split contain the training samples by skipping one and two consecutive samples in the given axis. While Step-over split is available for azimuth, elevation and distance axis ([A] Step-over, [E] Step-over and [D] Step-over), Step-larger split is only available for the azimuth axis ([A] Step-larger) since skipping two consecutive elevations and distances were not applicable. Random even ([R] Even) and one-third ([R] One-Third) splits randomly pick half and one-third of the total samples as the training set, respectively. All of these splits are illustrated in Figure 20 where each dot represents a viewpoint that is located according to the center of the graph where the object is stationed. Blue dots correspond to the training samples while red dots represent the test samples. [A] Step-larger, [D] Step-over, and [R] One-Third setups have one-third of the total samples in the dataset for training. On the other hand, all the remaining setups have even splits (see Table 4 for sample numbers for each split).

3.4 Alleviated Pose Attentive Capsule Agreement (ALPACA)

In this section, we propose *Alleviated Pose Attentive Capsule Agreement* (*ALPACA*) that improves the generalization of capsule networks to unseen viewpoints of objects in 2D images. Unlike multi-view object recognition or shape recognition where multiple views or the entire shape model is available to the algorithm, *ALPACA* only uses the 2D input image as every other image recognition CNNs and capsule network. Our contributions in this study are as follows:

i We propose a novel routing scheme called *ALPACA*, which depends on the part-whole pose similarities in terms of the geodesic distance to compute the features of the parent capsules. Also, we employ a single-layered perceptron to calculate feature votes for every parent of a capsule. This module reduces the number of parameters and the computational complexity when compared to learned matrix transformations for each child-parent capsule connection.

ii We propose a novel module for primary capsule extraction with the idea of composing the information from different heads.

iii We propose a novel activation for quaternion capsules where the normalized eigenvalue corresponding to the quaternion average and the total variation over the feature vector is used.

iv We provide the results of both our model and the state-of-the-art capsule networks on iLab2M [2] that is a toy vehicle objects dataset containing 2 million samples with background and different viewpoint variations.

v While outperforming its counterparts in most of the test setups, *ALPACA* increases the average training and test speeds by ~10 fold when compared with the iterative routing-based methods.

Experimental results show that our proposed architecture outperforms its counterparts and CNNs both in *NVPD* and iLab2M. Particularly on the iLab2M dataset, *ALPACA* surpasses its counterparts by at least 4% in terms of accuracy. Likewise, *ALPACA* surpasses, *ResNet50* and *DenseNet* ensemble results which are trained on images with 8 times higher resolution. In addition to the accuracy results, we provide further details about the effects of the reconstruction, loss functions, and proposed activation on the performance of our proposed architecture.

3.4.1 Related Work

In image recognition tasks, it is demonstrated that Quaternion Capsule Networks (QCN) generalize better to novel viewpoints with fewer parameters, even when EM-routing is directly applied. $ALPACA$ architecture builds upon these two papers by adopting quaternion pose representations.

On the other hand, there is a considerable amount of work that introduces some form of attention to the routing procedure because of the resemblance of the routing procedure with attention systems. From a broad perspective, both methods produce output vectors given a set of input vectors where the agreement and attention between the input are considered.

Attention Routing between Capsules [79] for example, uses attention architecture that learns a mapping between features in different levels and a novel capsule activation. This learned mapping is a feed-forward network like Self-routing capsules that determine its own routing coefficients. Inverted Dot-product Routing [80] on the other hand, propose a concurrent routing scheme that involves two stages of agreement computation and pose computation. The competition for attention is inverted when compared to dynamic routing in this approach and pose information is computed via layer normalization.

Lastly, STAR-CAPS [81] consists of two novel mechanisms which are responsible for estimating the attention between connected capsules and a binary differentiable classifier that determines which capsules should be connected to each other during a forward pass. This is referred to as hard attention where either there is full attention or none from the parent capsules to the child capsules.

Despite being an attention-based routing mechanism, our work differs from its counterparts in several ways. First, our capsules are formed by a feature vector, a pose representation, and an activation, in contrast to the previous work where capsules either have a feature vector or a matrix and an activation at most. Consequently,

our routing procedure utilizes the pose agreement as attention to calculate the feature vectors. Additionally, attention is calculated by the geodesic distance between quaternions poses. A closely related work to this thesis is the Quaternion Equivariant Capsules (QEC) which consider each data point in its input 3D point cloud as a quaternion [82]. Particularly, quaternion parts of *ALPACA* utilize the quaternion average that is used in QEC.

ALPACA differs from *QEC* and *QCN* in a fundamental way since both models only represent pose information and activation in capsules even though it is represented as quaternions. Contrary to *ALPACA* that has capsules with pose information, feature vectors, and an activation. Additionally, both methods use iterative routing schemes. While *QCN* calculates the activation following the procedure in *EM-Routing*, *QEC* calculates activation by the average geodesic distance of the votes to the weighted quaternion average. *ALPACA* calculate capsule activations from a combination of two elements. First the maximum normalized eigenvalue of the covariance matrix of incoming pose votes. Second, the total variance of the feature vector votes.

QEC works on 3D point clouds and therefore assumes each data point is a quaternion thus, do not require a capsule extraction procedure that forms the primary capsules. On the contrary, *QCN* and *ALPACA* employ a primary capsule extraction procedure since they both work on images. *QCN* has a branch for each capsule element (quaternion pose and activation) while *ALPACA* has a backbone-multihead structure to form primary capsules. This structure is entirely similar to the ones in region proposal networks; however, it was not considered for capsule networks previously. Lastly, *ALPACA* has dense connections in capsule layers like[83] in contrast to convolutional connections in *QCN* and the hierarchical structure in *QEC*. An overview of the contrasts between the existing capsule networks and our work is given in Table 5.

Table 5: Comparison of our method with recent capsule networks.

	Iterative Routing	Capsule Structure	Loss function	Reconstruction	Primary Capsule Extraction
ALPACA	No	quaternion, vector, activation	Spread and MSE	Yes	backbone-multihead
QCN [27]	Yes	quaternion, activation	Spread	No	branched
SRCaps [40]	No	vector, activation	Spread	No	Residual Network
InvDotCaps [80]	Concurrent	matrix	Cross Entropy/Binary Cross Entropy	No	Residual Network
VBCaps[25]	Yes	matrix, activation	Negative Log Likelihood and Variational Autoencoder Loss	Yes	Conv. Layers

3.4.2 Capsule Structure

In our proposed architecture, the capsules are composed of an activation (a), a pure quaternion (q), and a feature vector (f). Where, a is a scalar, q is a pure quaternion (3 parameters) and f is a f_{dim} dimensional vector which is constant for each capsule. In terms of capsules with quaternions, we use a similar notation with Ozcan *et al.* [27] to define the interactions between capsule layers. The set of capsules C_L in layer L is defined as $C_L = \left\{ [a_i, q_i, f_i] \right\}_i^{n_L}$ where n_L is the number of capsules in L^{th} layer. Each child capsule in layer L sends the votes to every connected parent capsule in layer $L+1$ about that capsules q and f. Formally, a vote can be defined as in Equation (25) where $i \in n_L$ and $j \in n_{L+1}$. Further details about the calculation of $\hat{q}_{i|j}$ and $\hat{f}_{i|j}$ are explained in the following sections.

$$v_{i|j} = \left[\hat{q}_{i|j}, \hat{f}_{i|j} \right] \tag{25}$$

Votes for the poses $\hat{q}_{i|j}$ are calculated by the same method in QCN given in Eq. 20. However, learned there is an additional learned parameter $\gamma_{i|j}$. The rotation angle $\theta_{i|j}$, the axis $\bar{\psi}_{i|j}$ and $\gamma_{i|j}$ are learned to form the rotation as given in Equation (26) where $\theta_{i|j}$ and $\bar{\psi}_{i|j}$ are the initial values sampled from uniform distributions between $[-\pi, \pi]$ and $[-0, 1]$, respectively. On the other hand, the initial $\gamma_{i|j}$ is sampled from $[-\rho_L, \rho_L]$ where ρ is calculated via Equation (28) to fit the initialization conditions as defined in [65]. As mentioned before, to represent a proper rotation that preserves

the scale of q_i and $w_{i|j}$ must be a unit quaternion. Therefore, the learned rotations are normalized to unit quaternions as in *QCN*.

$$w_{i|j} = \left[\gamma_{i|j} \cos\theta_{i|j}, \quad \gamma_{i|j} \sin\theta_{i|j} \frac{\bar{\psi}_{\mathbf{i|j}}}{\left\| \bar{\psi}_{\mathbf{i|j}} \right\|} \right] \tag{26}$$

$$w^*_{i|j} = \left[\gamma_{i|j} \cos\theta_{i|j}, \quad -\gamma_{i|j} \sin\theta_{i|j} \frac{\bar{\psi}_{\mathbf{i|j}}}{\left\| \bar{\psi}_{\mathbf{i|j}} \right\|} \right] \tag{27}$$

$$\rho_L = \frac{1}{\sqrt{2\left(n_L + n_{L+1}\right)}} \tag{28}$$

3.4.2.1 Quaternion Average and Geodesic Distance

Solution of Equation (29) yields the average of a weighted set of quaternions $\mathcal{A}(\mathbf{q_{i|j}}, \mathbf{a_i})$ [84] where M can be defined as given in Equation (30) with the weights as capsule activations $\mathbf{a_i}$.

$$\mathcal{A}_j(\mathbf{q_{i|j}}, \mathbf{a_i}) = \underset{\mathbf{q}\in\mathbb{S}^3}{\arg\max} \mathbf{q}^\top \mathbf{M_j} \mathbf{q} \tag{29}$$

$$\mathbf{M}_j \triangleq \sum_{i=1}^{n_L} a_i \hat{q}_{i|j} \hat{q}_{i|j}^\top \tag{30}$$

The maximizing quaternion of Equation (29) is found by the eigenvalue decomposition. According to Equation 30, M_j is a symmetric positive semi-definite matrix, therefore eigenvalue decomposition in Equation 31 is valid where Q is the orthogonal matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues λ_i.

$$\mathbf{M_j} = Q \Lambda Q^\top \tag{31}$$

As Q is orthogonal the following statement is valid, for $\hat{\mathbf{q}} = Q^\top \mathbf{q}$.

$$\|\hat{\mathbf{q}}\|^2 = \left\|Q^\top \mathbf{q}\right\|^2 = (Q^\top \mathbf{q})^\top (Q^\top \mathbf{q}) = \mathbf{q}^\top Q Q^\top \mathbf{q} = \mathbf{q}^\top \mathbf{q} = \|\mathbf{q}\|^2 \tag{32}$$

Since q is a unit vector,

$$\|\hat{\mathbf{q}}\|^2 = \|\mathbf{q}\|^2 = \sum_{i=0}^{2} q_i^2 = 1 \tag{33}$$

If M_j in Equation 29 is substituted with the decomposition in Equation 31,

$$\mathbf{q}^\top \mathbf{M_j} \mathbf{q} = \mathbf{q}^\top Q \Lambda Q^\top \mathbf{q} = \hat{\mathbf{q}}^\top \Lambda \hat{\mathbf{q}} = \sum_{i=0}^{2} \lambda_i q_i^2 \tag{34}$$

Boundaries for Equation 34 is easy to find for $\|\mathbf{q}\|^2 = \sum_{i=0}^{3} q_i^2 = 1$ which is as follows,

$$\min(\Lambda) \sum_{i=0}^{2} q_i^2 \leqslant \sum_{i=0}^{2} \lambda_i q_i^2 \leqslant \max(\Lambda) \sum_{i=0}^{2} q_i^2 \tag{35}$$

Where, $\max(\Lambda)$ is the maximum eigenvalue of $\mathbf{M_j}$, and to obtain the maximum value, $\mathbf{q}$ should be the eigenvector that corresponds to the largest eigenvalue according to $\hat{\mathbf{q}} = Q^\top \mathbf{q}$. This operation can be automatically differentiable given that M is a symmetric positive semi-definite matrix [85].

Note that $\mathcal{A}(\mathbf{q_{i|j}}, \mathbf{a_i})$ is a left equivariant and permutation-invariant operation [82] which serves the purpose of capsules being equivariant.

Instead of using the dot product, the Riemannian (geodesic) distance $\delta(\cdot, \cdot)$ in Equation (36) is used to calculate the distances between the rotations, which demonstrates promising results in practical applications [86, 87]. It can be considered as the geodesic curve on the quaternion unit sphere, and can be interpreted as the required energy to rotate a quaternion q_1 to another quaternion q_2. An illustration of the geodesic distance is given in Figure 21.

$$\delta\left(q_1, q_2\right) = 2\cos^{-1}\left(\left|\langle q_1, q_2\rangle\right|\right) \tag{36}$$

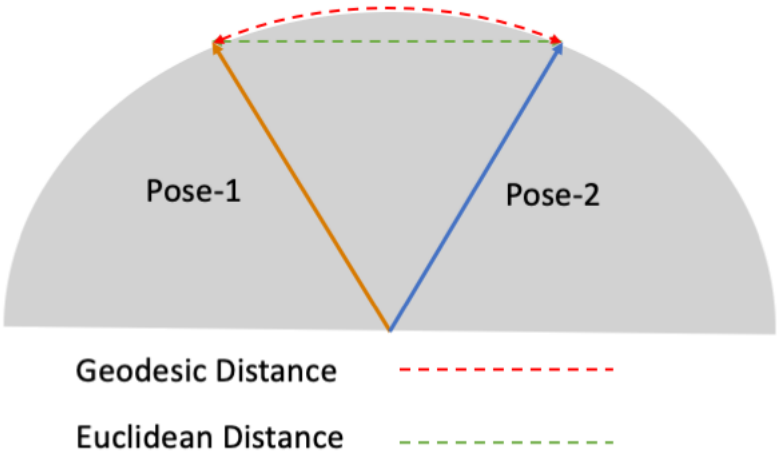


Figure 21: Geodesic vs. Euclidean distance

3.4.3 Vote Calculation and *ALPACA* Routing

In Section, we propose a novel routing scheme, namely *Alleviated Pose Attentive Capsule Agreement* (*ALPACA*) given in Algorithm 3, which mainly depends on the part-whole pose similarities in terms of geodesic distance. In this approach, two consecutive capsule layers are connected via *ALPACA* that calculates the contents for capsules in the deeper layer. For each layer, a single function $\Theta(.)_L$ with a linear layer followed by a ReLU activation [88] generates the feature vote of each child from its feature vector for the corresponding parent capsule, as given in Equation (37). A detailed illustration of how a child capsule calculates its votes is illustrated in Figure 22. Here feature vote calculator $\Theta(.)_L$'s weights are shared among all child capsules and it has an input size of f_{dim} and output size of $f_{dim} \times |C|$ where, $|C|$ is the number of the parent capsules that a child capsule is supposed to send the vote to.

Utilizing a shared fully connected network alleviates the feature vector vote calculation in contrast to previous approaches, where votes are calculated by using linear transformations, which require a learned matrix transformation in between for each

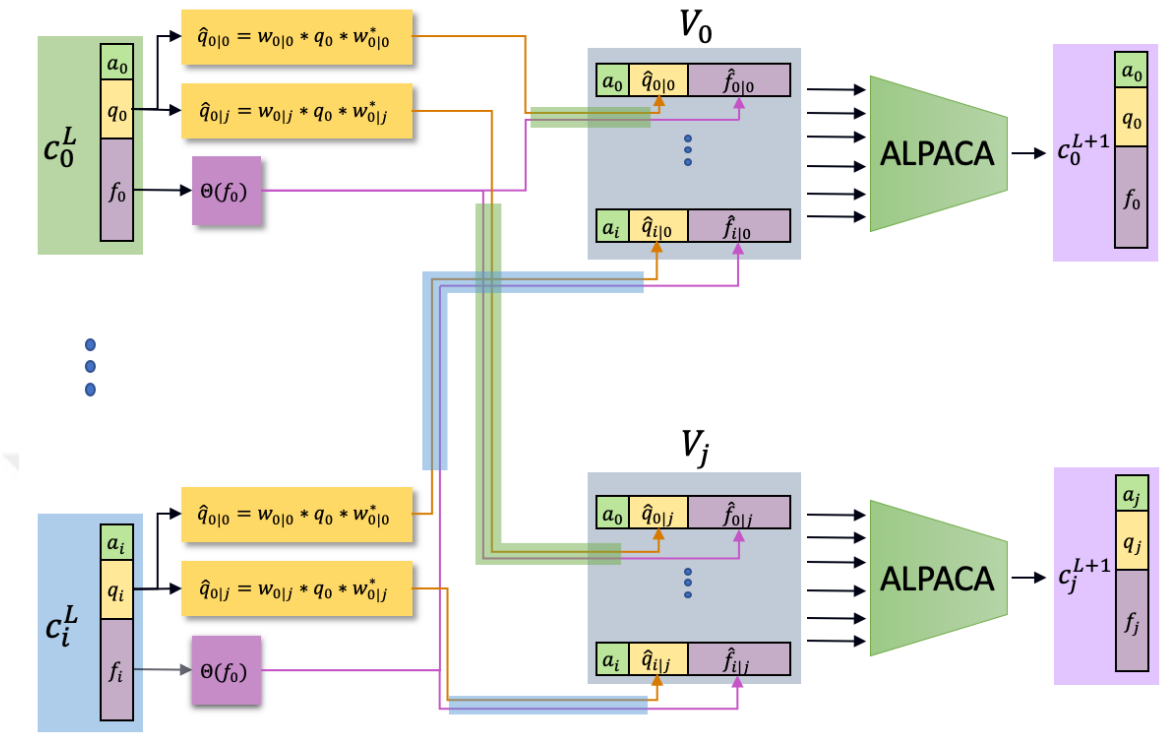


Figure 22: Illustration of how child capsules cast votes to parent capsules. Notice that each child capsule uses the same function Θ to calculate its feature votes. V_0 and V_j stands for the incoming votes to capsules c_0^{L+1} and c_j^{L+1} respectively.

connection. By employing *ALPACA*, we considerably reduce the number of parameters from $(n_{L+1} \times n_L \times f_{dim} \times f_{dim})$ to $(n_{L+1} \times f_{dim} \times f_{dim})$ for a single fully connected layer.

$$\hat{F}_i = \Theta_L(f_i) \quad \hat{F}_i = \{f_{i|j} \mid j \in n_{L+1}\} \tag{37}$$

Given all votes, *ALPACA* computes the a_j, q_j, and f_j of the parent capsules. Pose q_j is calculated by the weighted quaternion average of the incoming votes where the weights are the child capsule activations. As explained previously, average quaternion obtained via 29 is the eigenvector that corresponds to the largest eigenvalue of M. Formal definition is given in Equation (38)

Algorithm 3: $ALPACA(a_i, q_{i|j}, f_{i|j})$

for *all* $j \in n_L + 1$ **do**

$$q_j = \mathcal{A}_j(\mathbf{q_{i|j}}, \mathbf{a_i}) \; ; \tag{29}$$

$$f_j = \frac{1}{n_L} \sum_i^{n_L} softmax(-\delta(\hat{q}_{i|j}, q_j)) f_{i|j}; \tag{40}$$

$$\beta_j = \sum_h^H \sigma_j^h \; ; \tag{41}$$

$$\alpha_j = \frac{\arg\max \lambda_k}{\sum \lambda_k} \quad k \in \{1, \ldots, K\} \; ; \tag{38}$$

$$a_j = sigmoid\left(\xi_j^\alpha \alpha_j - \xi_j^\beta \beta_j\right) \; ; \tag{43}$$

end

return $C_{L+1} = \left\{ \; [\, a_j, q_j, f_j \,] \; \right\}_i^{n_{L+1}}$

$$\alpha_j = \frac{\arg\max \lambda_k}{\sum \lambda_k} \quad k \in \{1, \ldots, K\} \tag{38}$$

3.4.3.1 Using eigenvalue computation in the activation

In Equation 29, M_j is a non-zero mean, the weighted covariance matrix of the incoming votes $\hat{q}_{i|j}$ where the weights are the respective existence probabilities a_i of the incoming votes. Since the covariance matrix is bilinear, a non-zero mean does not affect this matrix. For a set of random quaternion variables Q and its mean arithmetic $\bar{Q}$ the following statement holds.

$$Cov(Q - \bar{Q}, Q - \bar{Q}) = Cov(Q, Q) \tag{39}$$

Therefore, the normalized value of the maximum eigenvalue remains unaffected. From principal component analysis, we know that normalized eigenvalues of the covariance matrix represent the portion of the information its corresponding eigenvector can encode. Therefore, when eigenvalues are normalized and sum up to 1, the largest

eigenvalue α_j represents the percentage of the total information in M that is represented by the average quaternion. Which is calculated while solving Equation 29 where K refers to the number of eigenvalues of $\mathbf{M}_j$, which is 4 due to quaternion dimensionality. This also avoids the extra computation since the eigenvalue is calculated during the quaternion average. Numerically, while the variance of the incoming votes goes to 0, the maximum normalized eigenvalue converges to 1. This is also empirically tested and illustrated in Figure 23, where the total variance is the sum of the variances in each dimension.

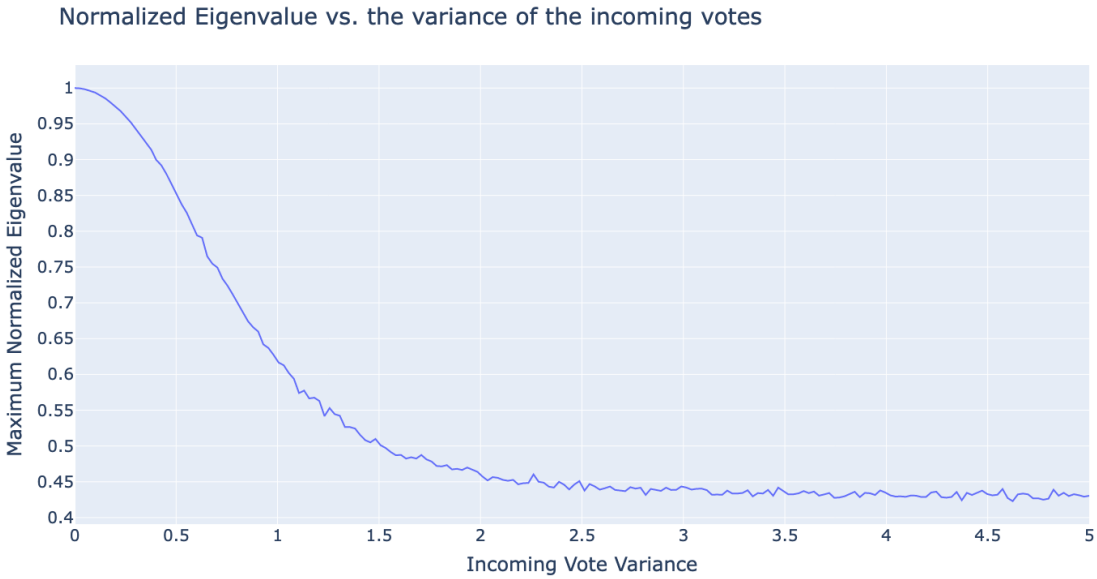


Figure 23: The change in the maximum normalized eigenvalue with respect to the total variance of the incoming votes.

The feature vector of a parent capsule f_j is calculated by attending to the geodesic distance between the parent pose q_j and the incoming pose vote $q_{i|j}$. In this way, child capsules that give their votes similar to the pose of parent capsules are likely to contribute more to the feature vector of parents f_j. This is similar to a single head of multi-head attention mechanism [89] with the fact that the "key" and "query" are in the quaternion domain, and with different dimensionality than the "value". In contrast to vanilla attention systems, the geodesic distance is used instead of the dot

product to calculate the attention weights. The formal definition for the calculation of a capsule feature vector f_j is given in Equation (40).

$$f_j = \frac{1}{n_L} \sum_{i}^{n_L} softmax(-\delta(\hat{q}_{i|j}, q_j)) f_{i|j} \tag{40}$$

Activation a_j of a parent capsule depends on how well a parent capsule represents the incoming feature and pose votes. This is measured by the weighted sum of the largest normalized eigenvalue α_j and the total variance β_j of incoming feature votes. While α_j represents how well a parent capsule represents the incoming pose votes, β_j represents the total agreement on the feature vector. β_j calculation is given in Equation (41) where h denotes the feature vector index and σ_j^h is the variance on h^{th} index over the incoming votes $f_{i|j}$, as formulated in Equation (42). The formal definition of activation a_j is given in Equation (43), where the weights ξ_j^α and ξ_j^β are learned parameters. The negative total variance is used since lower β_j means the incoming votes agree more on the feature vector of the parents. Even though all layers are fully connected in our architecture, *ALPACA* is independent of the connection types, and thus can also work with convolutionally-connected layers. Additionally, *ALPACA* can be calculated in parallel over the parent capsules as the calculations are all independent and its non-iterative nature increases the run-time speed by a large margin. Flowchart of *ALPACA* for the parent capsule c_0^{L+1} is illustrated in Figure 24, where η refers to the attention weights (output of $softmax(-\delta(\hat{q}_{i|j}, q_j)) f_{i|j}$ for $j = 0$).

$$\beta_j = \sum_{h}^{H} \sigma_j^h \tag{41}$$

$$\sigma_j^h = \frac{\sum_i^{n_L} \left(f_j^h - f_{i|j}^h \right)^2}{n_L} \tag{42}$$

$$a_j = sigmoid\left(\xi_j^{\alpha}\alpha_j - \xi_j^{\beta}\beta_j\right) \tag{43}$$

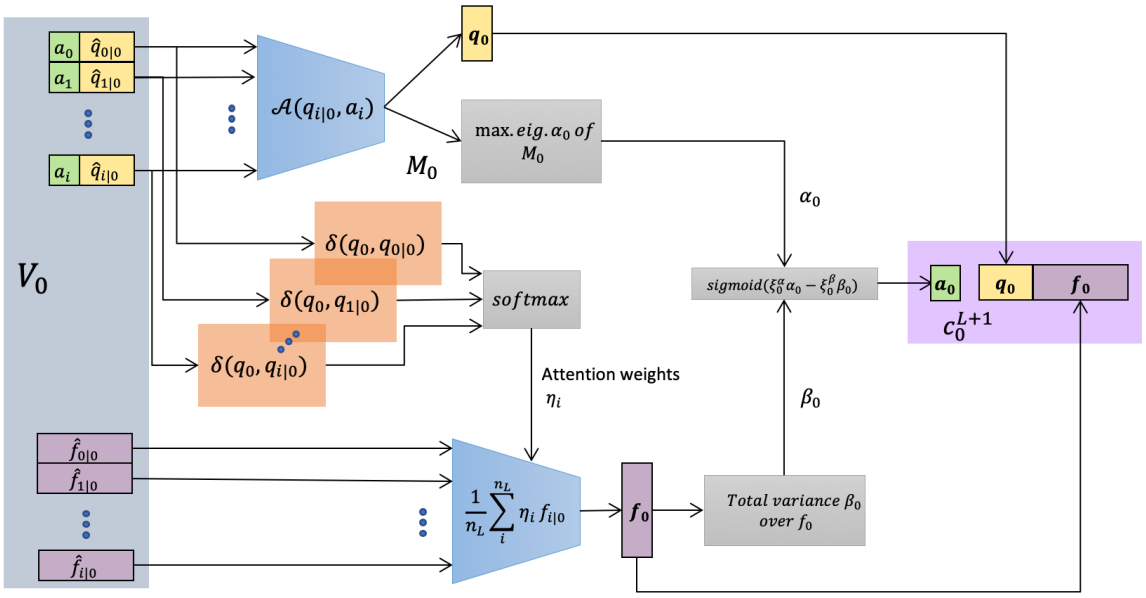


Figure 24: Flowchart of *ALPACA* for a parent capsule c_0^{L+1}.

3.4.4 Model Architecture

The proposed architecture consists of three main parts, a backbone-multihead structure that extracts the primary capsules, the composition of capsule layers with *ALPACA* $\mathcal{C}(\cdot)$, and lastly the reconstruction network $\mathcal{R}(\cdot)$. The proposed architecture differs from the previous models in the literature in the case of primary capsule extraction. While a common approach is to stack some convolutional layers and to group the outputs to form the primary capsules, we have adopted a backbone-multihead type of structure, as illustrated in Figure 25. This allows to the substitution of any type of network compatible with backpropagation in any of the elements in the architecture.

Backbone-multihead structure has a backbone network $\Phi(\cdot)$ that extracts the features and different heads for activation $\vartheta_a(\cdot)$, quaternions $\vartheta_q(\cdot)$ and feature vectors $\vartheta_f(\cdot)$ for primary capsules. Backbone network is composed of 5 consecutive residual blocks. These blocks have 3×3 kernels along with the number of channels for each

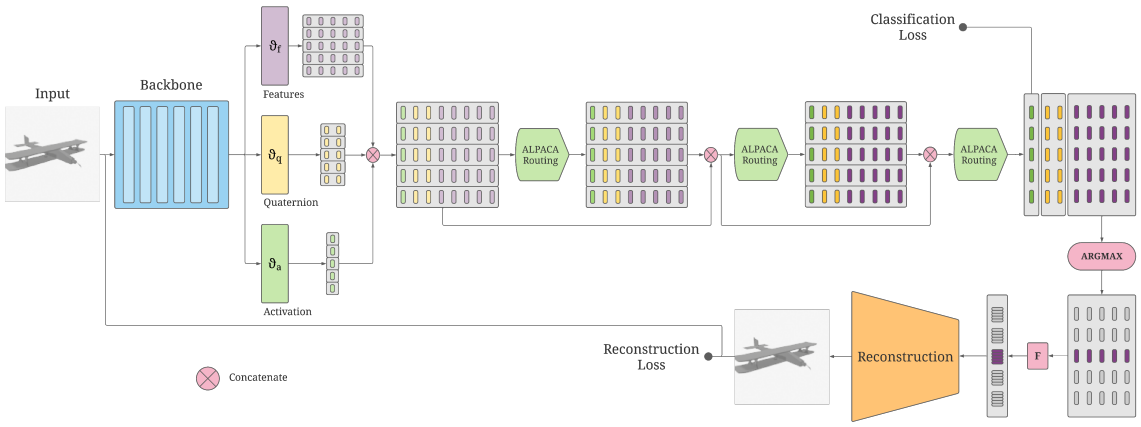


Figure 25: Detailed architecture of the proposed model. Where, $\vartheta_f(\cdot)$, $\vartheta_q(\cdot)$ and $\vartheta_a(\cdot)$ represent feature vector, quaternion and activation heads respectively. Classification loss and Reconstruction loss refer to L_c and L_{mse}. Lastly, *ALPACA* routing is the Algorithm 3.

block $C_i \in [16, 16, 32, 32, 64]$, and strides $S_i \in [1, 2, 2, 2, 2]$. We have used the same residual block mechanism in *QCN*, which is referred as pre-activation residual block [90], as illustrated in Figure 18.

Output feature of the backbone f_b is fed into the heads which calculate the activation, pose, and feature vector of primary capsules. $\vartheta_a(.)$, $\vartheta_q(.)$ and $\vartheta_f(.)$ represent the different heads containing fully connected layers with 1, 3 and f_{dim} channels, respectively. The activation head $\vartheta_a(.)$ and the feature vector head $\vartheta_f(.)$ have sigmoid activation function, whereas the quaternion head $\vartheta_q(.)$ utilizes the normalization as an activation. Note that pose is represented by a pure quaternion where s_q equals to 0, thus 3 channels in the output of $\vartheta_q(.)$ are concatenated with zeroes to represent a pure quaternion. The outputs of different heads for all primary capsules (*i.e.*, 32 capsules) in our design is formally defined in Equations (44), (45), (46). A_i, Q_i and F_i are then packed together with respect to i to form primary capsules C_0.

$$A_i = \vartheta_a(f_b) \quad A_i = \{a_i \mid i \in n_0\} \tag{44}$$

$$Q_i = \vartheta_q(f_b) \quad Q_i = \{q_i \mid i \in n_0\} \tag{45}$$

$$F_i = \vartheta_a(f_b) \quad F_i = \{f_i \mid i \in n_0\} \tag{46}$$

Capsule composition module $\mathcal{C}(\cdot)$ takes primary capsules C_0, which is formed in the previous module with the help of different heads. It outputs 16 intermediate capsules at each layer. In our design, there are 3 consecutive fully and densely connected layers where the input of each fully connected layer is concatenated with the output of the *ALPACA* until the class capsule layer. This allows better gradient flow and has a valid intuitive explanation for the part-whole relationships. A primary capsule may be a part of the whole at different levels or even be a class capsule itself and should be re-used since some objects contain multiples of the same parts. Therefore, dense connections allow these entities to be re-used in deeper layers. *ALPACA* calculates capsules in the next layer given the incoming votes from the capsules in the previous layer. Lastly, the class capsule layer gives the final output of the proposed architecture.

The main purpose of the reconstruction network $\mathcal{R}(\cdot)$ is to regularize the training process rather than visual plausibility. Therefore, simple neural networks are used to avoid further computational burden during the training. $\mathcal{R}(\cdot)$ reconstructs the input image from the concatenated feature vectors of class capsules F_c. Note that every feature vector apart from the ground truth class capsule is masked in F_c. The reconstruction network in [24] is used in *NVPD* training since it is previously tested on grayscale images. On the other hand, in iLab2M reconstruction network is substituted with the one in [91] mainly because of the colored images in the dataset.

3.4.5 Loss functions

ALPACA is mainly trained to minimize the standard classification losses used in the previous Capsule-related studies. In our study, we have compared well-known cross-entropy loss L_{ce} and spread loss L_{spread}, which is used in Matrix Capsules [1]. The formulas for both loss functions are defined in Equations (47) and (48).

$$L_{ce} = -\sum_{i}^{c} t_i \log(a_i) \tag{47}$$

$$L_i = \sum_{i \neq t} (max(0, m - (a_t - a_i)))^2, \quad L_{spread} = \sum_{i \neq t} L_i \tag{48}$$

where, for both equations, a_i represents the activations of the class capsules, and t_i is the ground truth indicator for each class capsule in Equation (47). Moreover, a_t is the ground truth, and m is the margin parameter which is gradually increased during the training from 0.2 up to 0.9 to avoid overfitting in the initial stages of training. The scheduling formula for m is given in Equation (24). Note that the division parameter z determines how fast m increases during training depending on the current iteration. We have empirically found that 8500 leads to the best stable training, and it gives the best results on our dataset and hyper-parameter settings.

Following the same practice in [24], we have used the reconstruction mechanism as a regularizer for capsules during training. This mechanism forces the capsule to capture the properties which is important for reconstructing the input image. At this point, we have used mean-squared error as the reconstruction loss, as shown in Equation (49) where $\hat{I}$ and I are the reconstruction and the input images, respectively.

$$L_{mse} = \frac{1}{I_w I_h I_c} \sum_{w=1}^{I_w} \sum_{h=1}^{I_h} \sum_{c=1}^{I_c} [\hat{I}(w,h,c) - I(w,h,c)]^2 \tag{49}$$

Finally, the total loss function can be represented as the weighted sum of the classification loss and the reconstruction loss. The common practice is to keep the weight of the reconstruction loss λ_r small, and thus it is set to 10^{-3}. The formal definition of the total loss L_{total} is given in Equation (50) where L_c refers to the classification loss, which is either the spread or cross-entropy loss during our experiments.

$$L_{total} = L_c + \lambda_r L_{mse} \tag{50}$$

3.4.6 Experiments

We evaluate our method on iLab2M and our *NVPD* datasets. While iLab2M has colored images with texture and background, *NVPD* has gray-scale images with no background and texture. However, *NVPD* has more challenging split scenarios in terms of viewpoint generalization in contrast to iLab2M where test and training sets have non-overlapping object instances rather than viewpoints. Please note that we refer to the spread loss variant of our model with *ALPACA* since it is the best-performing variant, while *ALPACA-CE* represents the model trained with cross-entropy.

We have compared our model with Matrix Capsules [1] which can be considered as a baseline. As a quaternion using a capsule network, Quaternion Capsules [27] are included in the experiments. Additionally, the state-of-the-art Capsule variants [25, 40, 80] are compared methods from the capsule network literature. Comparison to CNNs is important since capsule networks claim to achieve better generalization to unseen viewpoints when compared to CNNs. Therefore, we created custom and shallow CNNs as a baseline and included widely used state-of-the-art deep CNNs [92, 93, 94] to provide empirical evidence of the capsule literature's claims. All the previously mentioned methods are tested with both on *NVPD* and iLab2m datasets. Hyperparameters (*e.g.*, number of layers and capsules or routing iterations if applicable) of each model are set according to their respective papers. Reported results are

the average results and the variance of the 10 and 5 best-performing experiments of each model in *NVPD* and iLab2m respectively. Results show that our method outperforms its counterparts in most of the settings and there are no major differences in accuracy variances. Another important outcome is that deeper CNNs generalize better to novel viewpoints when compared to shallow CNNs. Lastly, employing sparse samples from unseen viewpoints for training dramatically affects the generalization performance of both types of networks. Therefore, CNNs and Capsule Networks can interpolate unseen viewpoints better from closer viewpoints.

NVPD is previously explained in Section 3.3. On the other hand, the iLab2M dataset, proposed in [2], is a dataset with real shots of toy objects. The iLab2M dataset consists of vehicle object-colored images under variations of viewpoint and background. All parameters of this dataset such as rotation angles, background images, or azimuth angles are controlled similarly to *NVPD*. There are $1.7M$ samples that are partitioned into 70% training, 15% validation, and 15% test splits. These splits are random splits in terms of viewpoints where an object instance in the dataset is not available in the training set. The iLab2M dataset contains five cameras and six rotations and the rotations refer to the turntable rotation where the objects are stationed on. Cameras are stationed on an arc over the object which corresponds to elevation on *NVPD*. However, all these cameras are stationed above the object in contrast to *NVPD* where half of the viewpoints are positioned below. There are 15 object categories and all are different types of vehicles, the dataset is almost balanced in terms of the number of images per class. Some examples of the iLab2m dataset are presented in Figure 26.

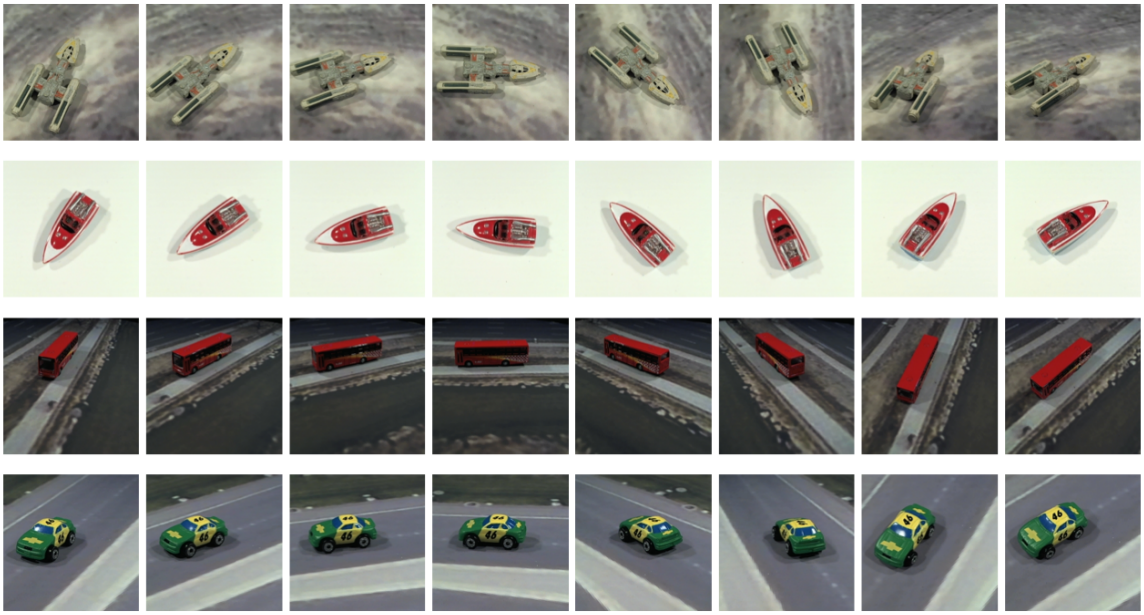


Figure 26: Examples of different objects and viewpoints from iLab2m dataset.

3.4.6.1 Experimental Setup

We have compared our proposed model with Quaternion Capsule Networks (*QCN*) [27], Matrix Capsules (*MatCaps*) [1], Capsule Routing via Variational Bayes (*VB-Caps*) [25], Capsules with Inverted Dot-Product Attention Routing (*InvDotCaps*) [80], Self-routing Capsules (*SRCaps*) [40]. In addition to Capsule variants, we have tested well-known CNN architectures like *DenseNet* [93], *SqueezeNet* [94] and *ResNet50* [92], a custom deep CNN with dropout (*LargeCNN*) and an equivalent simple CNN (*BaseCNN*). For *NVPD*, 10 independent experiments with each model's default initialization are made for every setup, and 5 experiments are made on iLab2m following [2].

LargeCNN has $7M$ parameters exceeding Capsule variants by a large margin. *LargeCNN* consists of 3 convolutional layers with 32, 64, and 64 channels with 11, 11, and 5 kernel sizes with the stride of 1. These layers are followed by 3 fully connected layers with 1024, 512, and 10 output neurons. Its all layers are activated by ReLU function [88]. The first convolutional layer has a dropout rate of 0.25 while

the rate is set to 0.5 for the second one. Convolutional layers reduce the size of the receptive fields by using the max-pooling operations. On the other hand, *BaseCNN* has 4 convolutional layers where the first layer has a kernel size of 11 while the rest has 5. From the first convolutional layer to the last one, the number of channels is set to 16, 32, 64, and 64, and the only first convolutional layer has a dropout with a rate of 0.25. Convolutional layers are followed by 2 fully connected layers with 128 and 10 output neurons. *BaseCNN* has $0.5M$ parameters as we have tried to build a base model with the same number of parameters as possible with $ALPACA$.

For a fair comparison in both datasets, the batch size, the initial learning rate, the hyper-parameters of the optimizer, and the input size are kept the same for each model. Batch size and initial learning rate are picked as 32 and 0.001. Adam optimizer is employed during training with its default parameters. The input size is set to 32×32 for all models. In addition to the hyper-parameter fixed comparisons, we have also compared our proposed model with the ensemble and mean results of *ResNet50* and *DenseNet* with input size of 256×256 [2].

Table 6: Experimental results on different splits of *NVPD* dataset. [A], [E], [D] and [R] refers to azimuth, elevation, distance and random, respectively.

Model / Splits	[A] Dark side	[A] Step-over	[A] Step-larger	[E] Dark side	[E] Step-over	[D] Step-over	[R] Even	[R] One-Third	# Iterations (it/s)	Million Parameters
BaseCNN	57.58 ∓ 0.54	88.51 ∓ 0.28	74.10 ∓ 0.43	75.52 ∓ 0.38	86.86 ∓ 0.35	44.02 ∓ 0.57	90.13 ∓ 0.20	89.18 ∓ 0.28	18.25	0.57
LargeCNN	73.03 ∓ 0.29	94.85 ∓ 0.13	78.90 ∓ 0.26	83.34 ∓ 0.18	93.75 ∓ 0.13	50.56 ∓ 0.44	93.05 ∓ 0.09	94.53 ∓ 0.12	5.49	7.44
MatCaps	69.01 ∓ 0.45	90.79 ∓ 0.38	73.42 ∓ 0.52	79.04 ∓ 0.29	88.46 ∓ 0.25	47.81 ∓ 0.49	93.45 ∓ 0.17	89.58 ∓ 0.11	3.25	0.32
ResNet-50	59.20 ∓ 0.44	88.40 ∓ 0.40	72.92 ∓ 0.35	73.61 ∓ 0.37	86.52 ∓ 0.33	41.60 ∓ 0.47	89.71 ∓ 0.21	90.23 ∓ 0.13	5.77	23.53
DenseNet	69.03 ∓ 0.41	96.25 ∓ 0.12	79.79 ∓ 0.15	87.14 ∓ 0.22	94.30 ∓ 0.14	42.80 ∓ 0.23	97.18 ∓ 0.1	96.03 ∓ 0.07	2.3	6,97
SqueezeNet	54.21 ∓ 0.47	77.70 ∓ 0.31	69.42 ∓ 0. 37	69.18 ∓ 0. 43	75.80 ∓ 0.38	35.22 ∓ 0.52	78.97 ∓ 0.22	79.41 ∓ 0.29	34.3	0.74
VBCaps	65.88 ∓ 0.32	90.24 ∓ 0.18	73.47 ∓ 0.19	78.45 ∓ 0.23	87.85 ∓ 0.19	47.55 ∓ 0.38	91.36 ∓ 0.15	89.62 ∓ 0.22	4.42	0.10
SRCaps	72.73 ∓ 0.24	93.46 ∓ 0.13	78.70 ∓ 0.33	82.66 ∓ 0.28	92.75 ∓ 0.15	56.86 ∓ 0.47	92.44 ∓ 0.12	92.10 ∓ 0.16	13.31	2.49
InvDotCaps	58.30 ∓ 0.53	92.11 ∓ 0.19	75.20 ∓ 0.24	77.71 ∓ 0.22	90.26 ∓ 0.20	48.49 ∓ 0.45	93.04 ∓ 0.21	91.86 ∓ 0.19	15.88	0.49
QCN	74.78 ∓ 0. 21	96.59 ∓ 0.12	**81.43** ∓ 0.25	87.42 ∓ 0. 21	94.62 ∓ 0.17	**57.19** ∓ 0.43	97.20 ∓ 0.17	94.90 ∓ 0.24	1.74	0.20
ALPACA	**75.25** ∓ 0.18	**96.64** ∓ 0.09	80.70 ∓ 0.19	**88.54** ∓ 0.21	**94.83** ∓ 0.1	53.00 ∓ 0.34	**97.59** ∓ 0.12	**96.10** ∓ 0.07	12.08	0.45

3.4.6.2 NVPD Results

Experimental results on *NVPD* are presented in Table 6 where [A], [E], [D], and [R] refer to azimuth, elevation, distance and random, respectively. *BaseCNN* and *LargeCNN* are the variant of CNNs with Dropout. Additionally, the overall performance of deep state-of-the-art CNN architectures (*i.e., ResNet50*, *SqueezeNet*, *DenseNet*) and Capsule Networks (*i.e., MatCaps*, *VBCaps*, *InvDotCaps*, *SRCaps* and *QCN*) on our proposed dataset are also presented in this table. Lastly, ALPACA refers to our proposed architecture. Visualization for dataset splits can be found in Figure 20. It shows that *ALPACA* outperforms its counterparts in 6 settings out of 8. Moreover, *ALPACA* runs approximately 10 times faster than models with EM-Routing based algorithms (*i.e.* 12.08 *it/s*), whereas *QCN*, *MatCaps* and *VBCaps* complete a single training step in 1.74 *it/s*, 3.25 *it/s* and 4.42 *it/s*, respectively. On the other hand, the methods using different attention-based routing mechanisms *InvDotCaps* and *SRCaps* are slightly faster than *ALPACA* which also deals with quaternion operations. Note that all experiments have been conducted on the same machine with 2x NVIDIA RTX 2080Ti GPU for a fair comparison. *ALPACA* outperforms other capsule networks and CNNs in every setup by a considerable margin. Only *QCN* outperforms in [A] Step-larger and [D] Step-over splits, but we have to point out that there is a huge difference between the speeds of *ALPACA* and *QCN*. On the other hand, networks with attention-based routing mechanisms *InvDotCaps* and *SRCaps* fall behind by a large margin in almost every split.

Experimental results of CNNs in *NVPD* show that CNNs fail in unseen viewpoint generalization even with state-of-the-art architectures and huge models when compared to Capsule Networks. Well-known state-of-the-art architectures, *DenseNet*, *SqueezeNet*, and *ResNet50* fall behind Capsule architectures in all splits. One of the highlights is that *ResNet50* falls behind in accuracy even with 23 million parameters. We must note that only *SqueezeNet* has, to some extent, a comparable number of

parameters to our architecture by having $300k$ more parameters, however, it is still outperformed by a large margin. On the other hand, baseline CNNs, *LargeCNN* and *BaseCNN*, also fail to reach the performance of capsules. In the case of comparing the performance of Capsule Networks, the results indicate that quaternions allow capsules to generalize better on novel viewpoints when compared to other architectures. Note that when capsules use the EM routing, *QCN* outperforms *MatCaps* in all settings.

3.4.6.3 iLab2M Results

Results with input size 32×32 are illustrated in Table 7 where *ALPACA* excels by a larger margin in contrast to NVPD. One of the consensuses on capsule architectures is that their accuracy falls off on datasets with backgrounds. The results of *ALPACA* and *InvDotCaps* indicate that having a slightly deeper backbone network architecture to create primary capsules boosts the performance. Another indication is that *ALPACA* is able to learn the general shape of an object type since train and test splits have different object instances in the iLab2M dataset.

Table 7: Performance on iLab2M Dataset. Models and training schemes are identical to the *NVPD* experiments with the exception of reconstruction network of *ALPACA*.

Model	**Accuracy**
BaseCNN	76.78 $\mp$ 0.36
LargeCNN	82.42 $\mp$ 0.32
MatCaps	68.66 $\mp$ 0.45
ResNet-50	81.74 $\mp$ 0.58
DenseNet	79.40 $\mp$ 0.35
SqueezeNet	62.66 $\mp$ 0.36
SRCaps	82.98 $\mp$ 0.3
InvDotCaps	82.12 $\mp$ 0.46
VBCaps	75.16 $\mp$ 0.41
QCN	75.30 $\mp$ 0.55
ALPACA	**87.00** $\mp$ 0.33

ALPACA also outperforms state-of-the-art architectures trained on images on 8 times larger images that have more details in the input and more importantly their ensemble results of 5 different seeds [2] given in Table 8. While, *ALPACA*, infer on 32×32 images, compared methods from [2] infer on 256×256 images which contain much more detailed information. Achieving better performance with smaller images have advantages in terms of the required hardware and training speed with respect to the batch size as it requires less memory space. Moreover, these results prove that generalization to novel viewpoints on basic objects is more reliant on the model rather than the input size. Naturally, *ALPACA* is outperformed by the methods that utilize viewpoint information in each sample in [2].

Table 8: *ResNet50* and *DenseNet* are trained on 256×256 images where ALPACA is trained on 32×32. *ResNet50* and *DenseNet* refer to the average where *ResNet50** and *DenseNet** stands for the ensemble accuracy of 5 independent initializations for each model [2].

Model	**Accuracy**
ResNet50	84.35 ∓ 0.72
DenseNet	84.08 ∓ 0.52
ResNet50*	86.14
DenseNet*	85.56
ALPACA	**87.00** ∓ 0.33

3.4.6.4 Ablation Study

There are several contributing factors to the performance of *ALPACA*, to pinpoint their effect and prove that they are not redundant, the performance of *ALPACA* without each of these factors is compared with the original. First, we have experimentally tested the effect of the activation in Equation (43) on the performance by removing the total variation β from the activation and only using α for the activation. Another important aspect of *ALPACA* is the reconstruction from feature vectors since it is one of the fundamental properties of the proposed architecture. Therefore, we have

conducted experiments by removing the feature vector from the capsules (*ALPACA-NoFeat*). Lastly, we have trained our model with cross-entropy loss (*ALPACA-CE*) for a comparison to spread loss since it was a design choice. For an easier comparison of the effects, the accuracy of each variant is reported in Table 9. Lastly, we have compared the results when only the maximum eigenvalue (*ALPACA-OnlyEig*) is used as the capsule activation. Instead of calculating the activation by the weighted sum of the total variance over the feature vector and the maximum eigenvalue, only the maximum eigenvalue is fed to the sigmoid function after the learned scaling parameter ξ_j^α in this version which is as follows:

$$a_j = sigmoid\left(\xi_j^\alpha \alpha_j\right) \tag{51}$$

Table 9: Test-set accuracy of *ALPACA* without certain properties. *ALPACA-NoFeat* is the variant of ALPACA where the feature vectors are excluded from the architecture. The second variant is *ALPACA-OnlyEig* which calculates the activation by only including the eigenvalue of the attitude matrix. Lastly, *ALPACA-CE* is the results with cross-entropy loss.

Splits / Model	**ALPACA-NoFeat**	**ALPACA-OnlyEig**	**ALPACA-CE**	**ALPACA**
Azimuth Darkside	74.08 ∓ 0.25	73.97 ∓ 0.21	71.80 ∓ 0.18	**75.25** ∓ 0.17
Azimuth Stepover	86.40 ∓ 0.19	95.38 ∓ 0.13	93.59 ∓ 0.11	**96.64** ∓ 0.09
Azimuth Stepover Larger	78.90 ∓ 0.27	78.02 ∓ 0.23	77.25 ∓ 0.18	**80.70** ∓ 0.19
Elevation Darkside	86.03 ∓ 0.26	86.99 ∓ 0.29	85.66 ∓ 0.20	**88.54** ∓ 0.21
Elevation Stepover	90.00 ∓ 0.17	93.94 ∓ 0.2	93.7 ∓ 0.11	**94.83** ∓ 0.1
Distance Stepover	47.00 ∓ 0.47	52.30 ∓ 0.67	48.87 ∓ 0.48	**53.00** ∓ 0.34
Random Even	95.29 ∓ 0.19	95.48 ∓ 0.16	94.57 ∓ 0.1	**97.59** ∓ 0.12
Random One-Third	94.22 ∓ 0.25	94.43 ∓ 0.24	94.66 ∓ 0.13	**96.10** ∓ 0.07
# Parameters	139,554	449,078	449,078	449,078

The results show that the feature vectors and reconstruction help to increase the novel viewpoint generalization performance of the model. This effect was excepted as discussed in [24] since reconstruction acts as a regularizer. Reconstructing purely from quaternions was impossible as you input only 3-dimensional pose information to the reconstruction network. Therefore, to have valid reconstructions, a feature vector that contains shape information is necessary for the architecture. However,

as can be seen in Figure 27 and 28, the reconstruction outputs are not in a realistic form, but for most cases, they resemble the given image and even show signs of translation and color variations in the input image. With that being said, the reconstruction network's aim is to regularize the features in capsules, rather than having realistic reconstructions. Therefore, we did not employ a more powerful reconstruction network or adversarial training strategy. Although as mentioned in future work, a viewpoint-aware reconstruction may be beneficial.

Removing the total variance of the feature vectors from the activation calculation also negatively affects the results as far as the *NVPD* is concerned. This is arguably due to the gradient flow in the feature representations of the capsules and its effect on the capsule existence probability. When total variance over the feature votes of a capsule is included in the capsule activation, that capsule's existence probability is affected by the agreement over the feature vector and the pose agreement of the incoming votes. On the other hand, using only the eigenvalue disregards the feature vector. Moreover, fully connected networks that calculate feature votes $\Theta(.)_L$ only receive gradients from the reconstruction loss in this case.

We have found out that using cross-entropy reduced the generalization performance which is aligned with the results from [1]. We believe this is because spread loss slowly forces the increase in discrepancy between the existence probability of ground truth and other capsules. This allows the training the focus on reconstruction early on and acts as an initialization learning before reducing the classification loss as during the early stages spread loss is very relaxed due to low m value.

Another important factor in the quality of the training is the dense connections. Training without dense connections also destabilizes the learning process at the earlier stages of training where the model fails to reduce the loss therefore not included in the table as the results fall behind the original by a large margin. We have found out that, without dense connections, there is a very weak gradient flow in the earlier

layers as the network gets deeper. Also, a more thorough parameter search may be required for a network without dense connections since the complexity of the trained model directly affects the training process which may require an entirely different set of hyper-parameters.

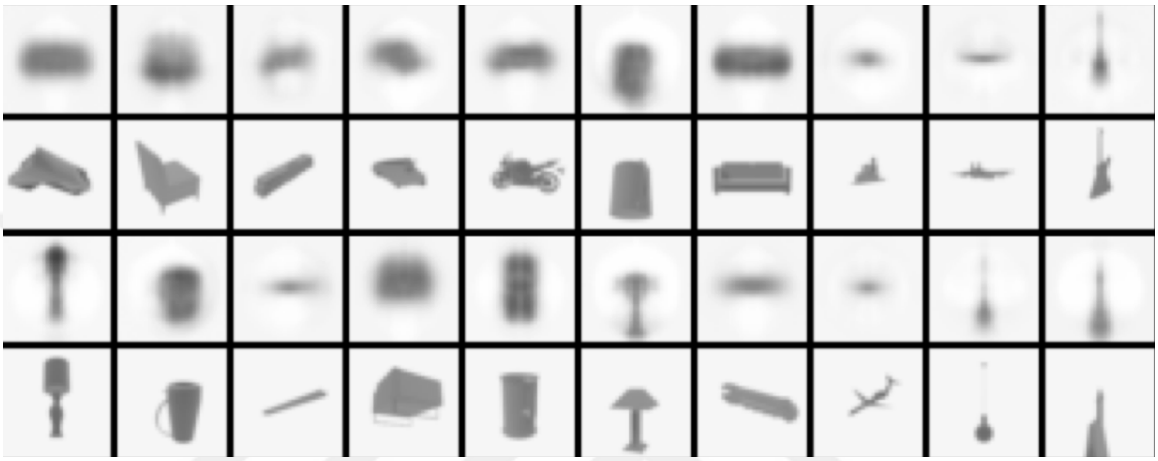


Figure 27: Reconstructions of *NVPD* (the first and third row) vs. input images (the second and last row).

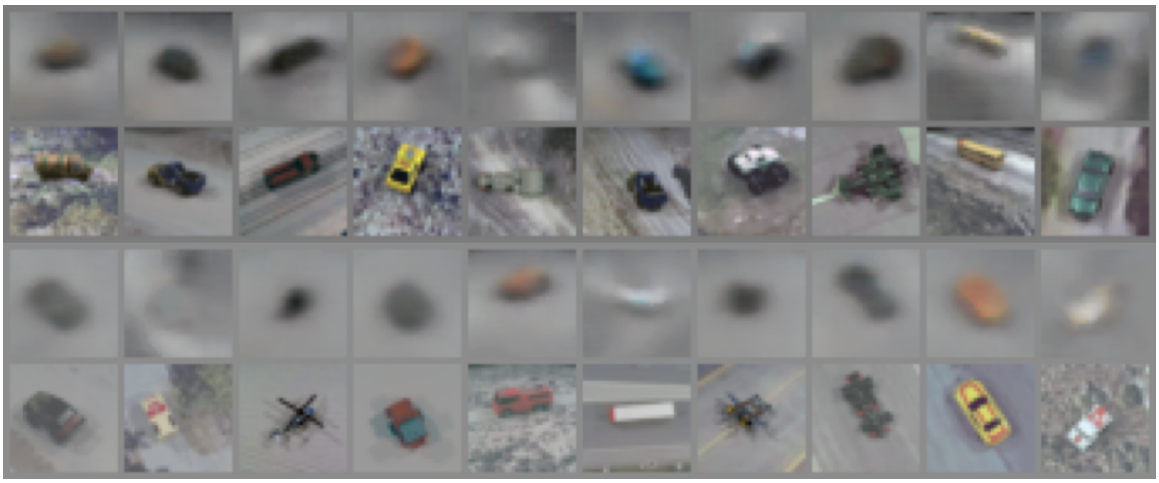


Figure 28: Reconstructions of iLab2m (the first and third row) vs. input images (the second and last row).

CHAPTER IV

CONCLUSION

Achieving viewpoint equivariance is vital for the generalization of visual recognition methods. CNNs are partially equivariant to transforms which led to huge improvements in visual tasks. However, CNNs are not equivariant to any other transformations such as rotation or viewpoints. This leads to storing a corresponding filter for every possible transform, which requires huge amounts of parameters. To learn these weights that are specialized for different transformations, naturally they require enough samples that contain every possible transformation. Another important bottleneck for current deep learning-based algorithms is the pooling operation of neuron activations that disregard the relative consistency of the visual entities which is referred to as the "picasso problem". Capsule networks aim to provide a solution to viewpoint equivariance by encoding each visual entities pose and aim to learn the underlying linear transform between the parts and the whole while substituting any pooling operation with routing by agreement that forces capsule networks to assert relative consistency of parts of any object rather than focusing only on their existence.

Despite its promising initial results capsule networks have a large computational burden due to the routing algorithm and transformation calculations between the layers, in addition to its training stability problems. In this thesis, we propose several improvements to Capsule Networks to achieve better generalization to novel viewpoints. First, we propose to represent pose information in quaternions and utilize quaternion rotation instead of matrix transformations. This alone improves the generalization performance of capsule networks and reduces the number of parameters in

a capsule to a quarter. Next, we have improved on quaternion representations by introducing a novel routing algorithm for capsules, namely *ALPACA*, which is designed to manage capsules that contain pose, feature, and activation information. *ALPACA* utilize quaternions for pose information and by exploiting quaternion average, ensure equivariance and allow us to introduce a novel eigenvalue activation calculation for capsules. Moreover, *ALPACA* has backbone-multihead structure to compute primary capsules, and it leverages dense connections in capsule layers, inspired from [83].

To observe the novel viewpoint generalization performance of Capsule Network variants and CNNs better, we introduce a novel viewpoint dataset tailored for viewpoint experiments. *NVPD* has 8 different setups that split the viewpoints from the different axis or randomly. Proposed architecture is compared with its counterparts in *NVPD* and iLab2m dataset. All the compared models are trained and tested 10 and 5 times for *NVPD* and iLab2m respectively. Our proposed architecture has better average accuracy on *NVPD* and iLab2m datasets. Experiments show that *ALPACA* architecture is 10 times faster from iterative routing methods where the average runtime speed is calculated over approximately 1,500,000 iterations for each model. Additional experiments also indicate that CNNs are not suitable architectures for generalizing novel viewpoints, while capsules can achieve better performance with significantly less number of parameters even when compared to state-of-the-art architectures.

4.1 Future Work

In the current version, *ALPACA* only utilizes a minimal reconstruction method and a related loss function. Particularly, the loss function is the mean squared error, unaware of the structure in the image. Exploring more intelligent reconstruction methods such as GANs or normalizing flows is a possible research direction. In addition to more sophisticated reconstruction methods, exploring canonical forms

for reconstruction rather than the input image is an important track as it simplifies reconstruction tasks.

Now, feature vectors and pose responses of capsules to the same class is not considered in *ALPACA*. While pose from two different viewpoints of the same class should be different, feature vectors must be equal since the attributes such as texture do not change with respect to the viewpoint. This is not enforced in the proposed architecture, introducing a layer-wise loss like what is provided in [95] may boost the performance.

Currently, a typical capsule network requires a capsule for every class in the dataset in its output layer which is a huge bottleneck for larger datasets with many classes. The computational burden this final layer brings makes capsule networks inapplicable in these datasets. Embedding the classification in a single capsule at the final layer by a modification of class estimation to *ALPACA* is an interesting track.

Ensuring primary capsules are primitive parts in the input image is yet to be solved, to attack this unsupervised semantic segmentation models may be utilized to extract primary capsules. Another interesting topic to better understand neural networks-based image recognition models is the experimental setup. Our experimental setup does not include unseen object instances from unseen viewpoints both in iLab2M and *NVPD*. This is a very challenging task and worth exploring either with capsule networks or CNNs as it will give an indication of the capability of models to recognize the basic properties of an object from different viewpoints, regardless of its instance.

REFERENCES

[1] G. E. Hinton, N. Frosst, and S. Sabour, "Matrix capsules with EM routing," in *International Conference on Learning Representations (ICLR)*, 2018.

[2] J. T. Leksut, J. Zhao, and L. Itti, "Learning visual variation for object recognition," *Image and Vision Computing*, vol. 98, p. 103912, 2020.

[3] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness," *arXiv preprint arXiv:1811.12231*, 2018.

[4] F. D. S. Ribeiro, K. Duarte, M. Everett, G. Leontidis, and M. Shah, "Learning with capsules: A survey," *arXiv preprint arXiv:2206.02664*, 2022.

[5] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," *CoRR, abs/1703.06211*, vol. 1, no. 2, p. 3, 2017.

[6] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming Auto-encoders," in *Proceedings of the 21th International Conference on Artificial Neural Networks*, ICANN'11, pp. 44–51, 2011.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[8] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European Conference on Computer Vision*, pp. 404–417, Springer, 2006.

[9] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, pp. 2564–2571, Ieee, 2011.

[10] I. Steinwart and A. Christmann, *Support vector machines*. Springer Science & Business Media, 2008.

[11] W. A. Qader, M. M. Ameen, and B. I. Ahmed, "An overview of bag of words; importance, implementation, applications, and challenges," in *International Engineering Conference (IEC)*, pp. 200–204, IEEE, 2019.

[12] C. Keskin, F. Kirac, Y. E. Kara, and L. Akarun, "Randomized decision forests for static and dynamic hand shape classification," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 31–36, IEEE, 2012.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

[14] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR)*, pp. 3642–3649, IEEE, 2012.

[15] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Computer Vision and Pattern Recognition (CVPR)*, pp. 1701–1708, 2014.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[17] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[18] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," *Journal of Statistical Mechanics: Theory and Experiment*, no. 12, p. 124003, 2021.

[19] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems*, pp. 91–99, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[23] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. 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*, pp. 4845–4854, 2019.

[24] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," in *Advances in Neural Information Processing Systems*, pp. 3856–3866, 2017.

[25] F. D. S. Ribeiro, G. Leontidis, and S. Kollias, "Capsule routing via variational bayes," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3749–3756, 2020.

[26] D. Peer, S. Stabinger, and A. Rodríguez-Sánchez, "Limitation of capsule networks," *Pattern Recognition Letters*, vol. 144, pp. 68–74, 2021.

[27] B. Özcan, F. Kinli, and F. Kiraç, "Quaternion capsule networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 6858–6865, IEEE, 2021.

[28] B. Özcan, F. Kınlı, and F. Kıraç, "Generalization to unseen viewpoint images of objects via alleviated pose attentive capsule agreement," *Neural Computing and Applications*, pp. 1–16, 2022.

[29] C. Olah, N. Cammarata, C. Voss, L. Schubert, and G. Goh, "Naturally occurring equivariance in neural networks," *Distill*, 2020. https://distill.pub/2020/circuits/equivariance.

[30] N. Cammarata, G. Goh, S. Carter, L. Schubert, M. Petrov, and C. Olah, "Curve detectors," *Distill*, 2020. https://distill.pub/2020/circuits/curve-detectors.

[31] G. Hinton, "Some demonstrations of the effects of structural descriptions in mental imagery," *Cognitive Science*, vol. 3, no. 3, pp. 231–250, 1979.

[32] F. D. S. Ribeiro, G. Leontidis, and S. D. Kollias, "Introducing routing uncertainty in capsule networks.," in *NeurIPS*, 2020.

[33] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," *Advances in Neural Information processing systems*, vol. 28, 2015.

[34] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool, "Dynamic filter networks," in *Advances in Neural Information Processing Systems*, pp. 667–675, 2016.

[35] T. S. Cohen and M. Welling, "Group Equivariant Convolutional Networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 2990–2999, 2016.

[36] T. S. Cohen and M. Welling, "Steerable cnns," *arXiv preprint arXiv:1612.08498*, 2016.

[37] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, "Harmonic networks: Deep translation and rotation equivariance," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5028–5037, 2017.

[38] Y. Lecun, F. Huang, and L. Bottou, "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. II–97, 01 2004.

[39] H. Li, X. Guo, B. Dai, O. Wanli, and X. Wang, "Neural Network Encapsulation," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[40] T. Hahn, M. Pyeon, and G. Kim, "Self-routing capsule networks," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[41] S. Zhang, Q. Zhou, and X. Wu, "Fast dynamic routing based on weighted kernel density estimation," in *International Symposium on Artificial Intelligence and Robotics*, pp. 301–309, Springer, 2018.

[42] L. Zhang, M. Edraki, and G.-J. Qi, "Cappronet: deep feature learning via orthogonal projections onto capsule subspaces," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 5819–5828, 2018.

[43] M. Edraki, N. Rahnavard, and M. Shah, "Subspace capsule network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 10745–10753, 2020.

[44] J. E. Lenssen, M. Fey, and P. Libuschewski, "Group Equivariant Capsule Networks," in *Advances in Neural Information Processing Systems*, pp. 8844–8853, 2018.

[45] A. R. Kosiorek, S. Sabour, Y. W. Teh, and G. E. Hinton, "Stacked Capsule Autoencoders," in *Advances in Neural Information Processing Systems*, 2019.

[46] R. LaLonde and U. Bagci, "Capsules for Object Segmentation," *arXiv preprint arXiv:1804.04241*, 2018.

[47] H. J. D. Koresh, S. Chacko, and M. Periyanayagi, "A modified capsule network algorithm for oct corneal image segmentation," *Pattern Recognition Letters*, vol. 143, pp. 104–112, 2021.

[48] K. Duarte, Y. Rawat, and M. Shah, "VideoCapsuleNet: A Simplified Network for Action Detection," in *Advances in Neural Information Processing Systems 31*, pp. 7610–7619, 2018.

[49] F. Kinli, B. Ozcan, and F. Kirac, "Fashion Image Retrieval with Capsule Networks," in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[50] F. Kınlı and F. Kıraç, "Fashioncapsnet: Clothing classification with capsule networks," *Bilişim Teknolojileri Dergisi*, vol. 13, pp. 87 – 96, 2020.

[51] H. H. Nguyen, J. Yamagishi, and I. Echizen, "Capsule-forensics: Using capsule networks to detect forged images and videos," in *ICASSP IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2307–2311, IEEE, 2019.

[52] M. E. Paoletti, J. M. Haut, R. Fernandez-Beltran, J. Plaza, A. Plaza, J. Li, and F. Pla, "Capsule networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 4, pp. 2145–2160, 2018.

[53] M. Yang, W. Zhao, J. Ye, Z. Lei, Z. Zhao, and S. Zhang, "Investigating capsule networks with dynamic routing for text classification," in *Conference on Empirical Methods in Natural Language Processing*, pp. 3110–3119, 2018.

[54] M. Wang, J. Xie, Z. Tan, J. Su, D. Xiong, and L. Li, "Towards linear time neural machine translation with capsule networks," *arXiv preprint arXiv:1811.00287*, 2018.

[55] W. Zhao, H. Peng, S. Eger, E. Cambria, and M. Yang, "Towards scalable and reliable capsule networks for challenging nlp applications," *arXiv preprint arXiv:1906.02829*, 2019.

[56] A. Hirose, "Complex-Valued Neural Networks: Theories and Applications (Series on Innovative Intelligence, 5)," 2004.

[57] H. G. Zimmermann, A. Minin, and V. Kusherbaeva, "Comparison of the Complex Valued and Real Valued Neural Networks Trained with Gradient Descent and Random Search Algorithms," in *Proc. of ESANN 2011*, 2011.

[58] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, "Associative Long Short-Term Memory," in *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48, pp. 1986–1994, 2016.

[59] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary Evolution Recurrent Neural Networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1120–1128, 2016.

[60] T. Nitta, "On the critical points of the complex-valued neural network," in *Proceedings of the 9th International Conference on Neural Information Processing, ICONIP.*, vol. 3, pp. 1099–1103, IEEE, 2002.

[61] A. Hirose and S. Yoshida, "Generalization Characteristics of Complex-Valued Feedforward Neural Networks in Relation to Signal Coherence," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, pp. 541–551, 2012.

[62] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep Complex Networks," *arXiv preprint arXiv:1705.09792*, 2017.

[63] C. J. Gaudet and A. S. Maida, "Deep Quaternion Networks," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2018.

[64] X. Zhu, Y. Xu, H. Xu, and C. Chen, "Quaternion Convolutional Neural Networks," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[65] T. Parcollet, M. Ravanelli, M. Morchid, G. Linarès, C. Trabelsi, R. De Mori, and Y. Bengio, "Quaternion Recurrent Neural Networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[66] T. Parcollet, Y. Zhang, M. Morchid, C. Trabelsi, G. Linarès, R. De Mori, and Y. Bengio, "Quaternion Convolutional Neural Networks for End-to-End Automatic Speech Recognition," 06 2018.

[67] D. Pletincks, "The use of quaternions for animation, modelling and rendering," in *New Trends in Computer Graphics*, pp. 44–53, Springer, 1988.

[68] F. S. Grassia, "Practical parameterization of rotations using the exponential map," *Journal of Graphics Tools*, vol. 3, no. 3, pp. 29–48, 1998.

[69] D. Pavllo, D. Grangier, and M. Auli, "Quaternet: A quaternion-based recurrent model for human motion," *arXiv preprint arXiv:1805.06485*, 2018.

[70] N. Matsui, T. Isokawa, H. Kusamichi, F. Peper, and H. Nishimura, "Quaternion neural network with geometrical operators," *Journal of Intelligent & Fuzzy Systems*, vol. 15, no. 3, 4, pp. 149–164, 2004.

[71] T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," in *Advances in Neural Information Processing Systems 29*, pp. 901–909, 2016.

[72] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, pp. 448–456, 07–09 Jul 2015.

[73] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256, 13–15 May 2010.

[74] A. D. Gritzman, "Avoiding Implementation Pitfalls of "Matrix Capsules with EM Routing" by Hinton et al.," in *Human Brain and Artificial Intelligence*, (Singapore), pp. 224–234, Springer Singapore, 2019.

[75] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[76] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[77] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[78] A. X. Chang *et al.*, "ShapeNet: An Information-Rich 3D Model Repository," Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[79] J. Choi, H. Seo, S. Im, and M. Kang, "Attention routing between capsules," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[80] Y.-H. H. Tsai, N. Srivastava, H. Goh, and R. Salakhutdinov, "Capsules with inverted dot-product attention routing," *arXiv preprint arXiv:2002.04764*, 2020.

[81] K. Ahmed and L. Torresani, "Star-caps: Capsule networks with straight-through attentive routing.," in *NeurIPS*, pp. 9098–9107, 2019.

[82] Y. Zhao, T. Birdal, J. E. Lenssen, E. Menegatti, L. Guibas, and F. Tombari, "Quaternion equivariant capsule networks for 3d point clouds," in *European Conference on Computer Vision*, pp. 1–19, Springer, 2020.

[83] Z.-X. Yu, Y. He, C. Zhu, S. Tian, and X.-C. Yin, "Carnet: Densely connected capsules with capsule-wise attention routing," in *Cyberspace Data and Intelligence, and Cyber-Living, Syndrome, and Health*, pp. 309–320, Springer, 2019.

[84] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman, "Quaternion averaging," *NASA Goddard Space Flight Center*, pp. 1–10, 2007.

[85] S. Laue, M. Mitterreiter, and J. Giesen, "Computing higher order derivatives of matrix and tensor expressions.," in *NeurIPS*, pp. 2755–2764, 2018.

[86] B. Jablonski, "Anisotropic filtering of multidimensional rotational trajectories as a generalization of 2d diffusion process," *Multidimensional Systems and Signal Processing*, vol. 19, no. 3-4, pp. 379–399, 2008.

[87] B. Jabłoński, "Application of quaternion scale space approach for motion processing," in *Image Processing and Communications Challenges 3*, pp. 141–148, Springer, 2011.

[88] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines.," in *ICML*, pp. 807–814, Omnipress, 2010.

[89] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[90] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*, pp. 630–645, Springer, 2016.

[91] F. O. KINLI and F. M. KIRAÇ, "Fashioncapsnet: clothing classification with capsule networks," *Bilişim Teknolojileri Dergisi*, vol. 13, no. 1, pp. 87–96, 2020.

[92] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[93] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[94] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[95] W. Sun, A. Tagliasacchi, B. Deng, S. Sabour, S. Yazdani, G. Hinton, and K. M. Yi, "Canonical capsules: Unsupervised capsules in canonical pose," *arXiv preprint arXiv:2012.04718*, 2020.

VITA

Barış Özcan received his BS degree in Mechatronics Engineering from Bahçeşehir University in 2011 and M.Sc. degree in Mechatronics Engineering from Istanbul Technical University in 2014. He has been involved in several government-funded projects such as a "multi-sensor lane departure warnıng system", "spatio-temporal mapping based post-occupancy evaluation for outdoor spaces" and "underwater sea lice detection". He is currently working at VCTEK as a research engineer. His research focuses on computer vision and machine learning.