Object-centric Learning with Capsule Networks: A Survey

FABIO DE SOUSA RIBEIRO, Department of Computing, Imperial College London, London, United Kingdom
KEVIN DUARTE, Electrical Engineering & Computer Sciences, University of Central Florida, Orlando, United States
MILES EVERETT, Department of Computing Science, University of Aberdeen, Aberdeen, United Kingdom
GEORGIOS LEONTIDIS, Department of Computing Science, University of Aberdeen, Aberdeen, United Kingdom
MUBARAK SHAH, Electrical Engineering & Computer Sciences, University of Central Florida, Orlando, United States

Capsule networks emerged as a promising alternative to convolutional neural networks for learning object-centric representations. The idea is to explicitly model part-whole hierarchies by using groups of neurons called *capsules* to encode visual entities, then learn the relationships between these entities dynamically from data. However, a major hurdle for capsule network research has been the lack of a reliable point of reference for understanding their foundational ideas and motivations. This survey provides a comprehensive and critical overview of capsule networks, which aims to serve as a main point of reference going forward. To that end, we introduce the fundamental concepts and motivations behind capsule networks, such as *equivariant inference*. We then cover various technical advances in capsule routing algorithms as well as alternative geometric and generative formulations. We provide a detailed explanation of how capsule networks relate to the attention mechanism in Transformers and uncover non-trivial conceptual similarities between them in the context of object-centric representation learning. We also review the extensive applications of capsule networks in computer vision, video and motion, graph representation learning, natural language processing, medical imaging, and many others. To conclude, we provide an in-depth discussion highlighting promising directions for future work.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Computer vision**; **Machine learning**; **Hierarchical representations;**

Additional Key Words and Phrases: Deep learning, capsule networks, deep neural networks, convolutional neural networks, transformers, routing-by-agreement, self-attention, representation learning, object-centric learning, generative models, computer vision

Authors' Contact Information: Fabio De Sousa Ribeiro, Department of Computing, Imperial College London, London, United Kingdom; e-mail: f.de-sousa-ribeiro@imperial.ac.uk; Kevin Duarte, Electrical Engineering & Computer Sciences, University of Central Florida, Orlando, Florida, United States; e-mail: kevin_duarte@knights.ucf.edu; Miles Everett, Department of Computing Science, University of Aberdeen, Aberdeen, Aberdeen, United Kingdom; e-mail: m.everett.20@abdn.ac.uk; Georgios Leontidis, Department of Computing Science, University of Aberdeen, Aberdeen, Aberdeen, United Kingdom; e-mail: georgios.leontidis@abdn.ac.uk; Mubarak Shah, Electrical Engineering & Computer Sciences, University of Central Florida, Orlando, Florida, United States; e-mail: shah@crcv.ucf.edu.

ACM 0360-0300/2024/07-ART291
https://doi.org/10.1145/3674500

ACM Reference Format:
Fabio De Sousa Ribeiro, Kevin Duarte, Miles Everett, Georgios Leontidis, and Mubarak Shah. 2024. Object-centric Learning with Capsule Networks: A Survey. *ACM Comput. Surv.* 56, 11, Article 291 (July 2024), 43 pages. https://doi.org/10.1145/3674500

1 Introduction

The quintessential task of computer vision is to classify an object from a vector of features extracted from an image and to provide fuller descriptions such as its pose, shape, appearance, and so on. For decades, constructing data representations (features) that were suitable for downstream tasks involved extensive hand-engineering and expert knowledge. Representation learning [9] consists of a set of tools that enable a machine to automatically discover useful representations of raw data, which may then be utilized for downstream predictive tasks. The most successful representation learning method in recent years is **Deep Learning (DL)** [70]. Despite the many successes of modern DL-based vision systems [47, 66, 70], a general lack of robustness to distributional shifts remains prevalent [40]. Indeed, unlike current systems, humans can quickly adapt to distributional changes using very few examples to learn from [10, 15, 21]. There is compelling evidence that humans parse visual scenes into part-whole hierarchies, and that we do so by modeling the viewpoint-invariant spatial relationship between a part and a whole as the coordinate transformation between the intrinsic coordinate frames assigned to them [48, 59, 102]. One way to make **Neural Networks (NN)** more transparent and interpretable is to try to make them understand images in the same way humans do. However, this is difficult for standard NNs, because they cannot dynamically represent a different part-whole hierarchy tree structure for each image [49]. This inability was the main motivation behind the development of capsule networks [49–51, 64, 106].

A capsule network is a type of NN that is designed to model part-whole hierarchical relationships more explicitly than **Convolutional Neural Networks (CNNs)** by using groups of neurons to encode entities and learning the relationships between these entities [72]. Like many other developments in machine learning [36, 95], capsule networks are biologically inspired, and their goal is to be able to learn more robust object-centric representations that are pose-aware and interpretable. Evidence from neuroscience suggests that groups of tightly connected nearby neurons (i.e., hypercolumns) could represent a vector-valued unit that is able to transmit not only scalar quantities but a set of coordinated values [10]. This idea of vector-valued units is at the heart of both capsule networks and the attention mechanisms [7, 10, 114] found in Transformers [123]. In capsule networks, these vector-valued units are known as capsules, and in Transformers, they are represented by query, key, and value vectors. Performing operations such as the scalar product between neural activity vectors enables powerful algorithmic concepts such as coincidence filtering and attention to be computed. Despite the promising progress on capsule networks, Barham et al.[8] explained that current DL frameworks have been highly optimized for a small subset of computations used by popular models. Although their capsule model required around 4 times fewer **floating point operations (FLOPS)** with 16 times fewer parameters than their CNN, implementations in both TensorFlow [1] and PyTorch [96] ran significantly slower and ran out of memory with much smaller models. We hope this survey will inspire researchers to develop suitable tools for capsule networks.

To that end, we provide a comprehensive review of representation learning using capsule networks and related attention-based models. Although research on capsules is still at an early stage, relatively speaking, Figure 1(a) shows us that despite an initial rapid growth in popularity, the total number of publications per year has somewhat stagnated. This is possibly due to the high barrier of entry to the field and the lack of a reliable point of reference. Nonetheless, as shown in

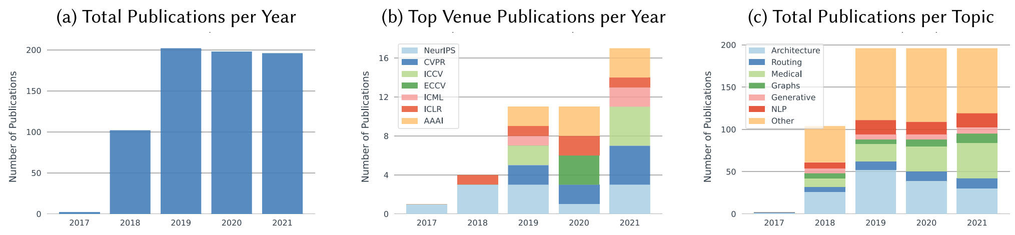

Fig. 1. Statistics of capsule network research activity. (a) Number of capsule network-related papers published in any scientific venue. (b) Number of capsule network-related papers published in top conferences. (c) Main research topics in capsule network literature.

Figure 1(b), the number of capsule network–related publications at the top venues has continued to increase steadily. We believe that there is now sufficient material to warrant a detailed organization of the various techniques and foundational ideas for the benefit of the community. At the time of this writing, there exist only three other capsule network-based surveys. Reference [124] was written shortly after capsule networks were first introduced, so it does not cover large milestones achieved recently. Similarly, Reference [110] was written to be brief and therefore covers a very small portion of the literature. Reference [97] is more recent and covers a larger breadth of research but is still lacking context, as the field has naturally continued to develop (see Table 1 for examples). Moreover, this survey provides a more thorough explanation of the motivating ideas behind capsules and positions them in the context of other seemingly unrelated avenues in the field of deep learning, such as *attention* and slot-based representation learning. In summary, the purpose of this survey is to provide the first highly comprehensive and detailed breakdown of capsule networks and related research. Specifically, we aim to: (a) explain the foundations, motivations, and fundamental concepts; (b) survey the state-of-the-art; (c) relate and compare capsules and routing-by-agreement with Transformers and self-attention; (d) discuss open problems and provide promising future research directions. We anticipate that our survey will serve as the main point of reference on capsule network research going forward and will help contribute towards the advancement of the field.

Overview. This survey is organized as follows: In Section 2, we gently introduce *invariance* and *equivariance* and explain why these concepts are fundamental in object-centric representation learning. In Section 3, we explain the foundational ideas and motivations behind capsule networks, introducing basic concepts such as vector-valued neural activities (*capsules*), *agreement*, and *capsule routing*. In Section 4, we review the most prominent capsule routing algorithms and draw comparisons between them. In Section 5, we uncover the conceptual similarities between capsule routing and the attention mechanism in Transformers. Sections 6 to 11 discuss applications of capsule networks for video, graphs, natural language processing, medical imaging, and many more. Last, in Section 12, we discuss open challenges and shortcomings of capsule networks, along with promising directions for future research.

2 Background & Motivation

2.1 Invariance

Invariance is a useful property to model for a variety of recognition tasks, as we would often like the final prediction of our model to be *invariant* to transformations of the input that preserve intrinsic properties, such as relative positions and symmetries. A *symmetry* of an object is a transformation that leaves it unchanged, e.g., rotating a (perfect) circle about its center (rotational

Table 1. Organization of Capsule Network Literature Covered in This Survey Split into the Most Prominent Research Areas

Research Area	**Literature**	**Description**
Capsule Routing Algorithms (Sections 3 & 4)	[4, 22, 30, 44, 50, 64, 80, 91, 101, 106, 120]	Novel routing algorithms for optimizing the degree of coupling between capsules across capsule network layers. Proposed methods range from dynamic iterative routing to concurrent and non-iterative, inspired by both probabilistic clustering algorithms and attention mechanisms.
Video & Motion with Capsules (Section 5.1)	[27, 28, 85, 88, 133]	Innovative applications of capsule networks in the analysis of video and motion data. This area is distinct from standard capsule networks, which only process images, as there is an extra dimension of *time*. Creative solutions are needed to improve efficiency during training and inference.
Graph/Geometric Capsules (Section 5.2)	[42, 60, 78, 113, 116, 128, 132, 142]	Capsule networks have been adapted to facilitate the processing of graph structures and 3D point clouds. This typically entails significant modifications in both theoretical and architectural aspects of standard capsule networks.
Generative Capsules (Section 5.3)	[29, 58, 91, 112]	These works move away from discriminative learning and pursue a generative perspective to learning with capsule networks. Examples include the integration of capsules into the framework of **Generative Adversarial Networks (GANs)**.
NLP with Capsules (Section 5.4)	[16, 18, 61, 75, 79, 121, 125, 131, 134, 136, 138, 139, 141, 143]	A breadth of work has been done in applying capsule networks to **Natural Language Processing (NLP)** tasks. Such works typically include using modifications of capsule routing instead of, or in combination with, the attention mechanism used in Transformers.
Capsules in Medical Imaging (Section 2.1)	[2, 3, 33, 68, 69, 76, 81, 90, 126, 140]	Many works have proposed applications of capsule networks to high-resolution and/or 3D image data in medical imaging contexts. The focus is typically on proposing modifications to capsule network architectures to improve efficiency and performance on real-world data.
Other Applications of Capsule Networks (Appendix A)	[24, 43, 84, 92, 94, 98, 127, 145]	Research exploring diverse applications of capsule networks across various domains beyond those specified in the other categories. Each of the sub-areas covered in the relevant section required bespoke modifications to capsule networks to be proposed, but these works do not have sufficient overlap to warrant their categorization.

invariance). For example, the intrinsic properties of the "Horse" in Figure 2 remain unchanged, relatively speaking, when translating, scaling, or flipping it, thus our model's prediction of "Horse" should be the same under these transformations. This notion of invariance is also linked to model generalization and design [9, 54].

Translation Invariance. To humans, both images shown in Figure 2(a) should be classified as a "Horse" regardless of where it appears within the boundaries. This is because a positional shift (translation T) of the horse does not change its intrinsic properties. Concretely, *translation invariance* refers to a feature mapping f that produces the same output (e.g., "Horse") regardless of input translation $T(x)$. More formally, f is invariant to input translations of x if:

$$f(x) = f(T(x)). \tag{1}$$

Sub-sampling techniques typically used in CNNs, such as max-pooling, make neural activities of the next layer locally invariant to input translations [9, 138]. That is, the output of a pooling unit is the same irrespective of where a specific feature is located inside its pooling region.

Viewpoint Invariance. A more challenging invariance to model is *viewpoint invariance* as shown in Figure 2(b). By the same logic as before, a feature mapping f is invariant to viewpoint projection V of the input x if: $f(x) = f(V(x))$. To humans, this task is relatively trivial, since we are

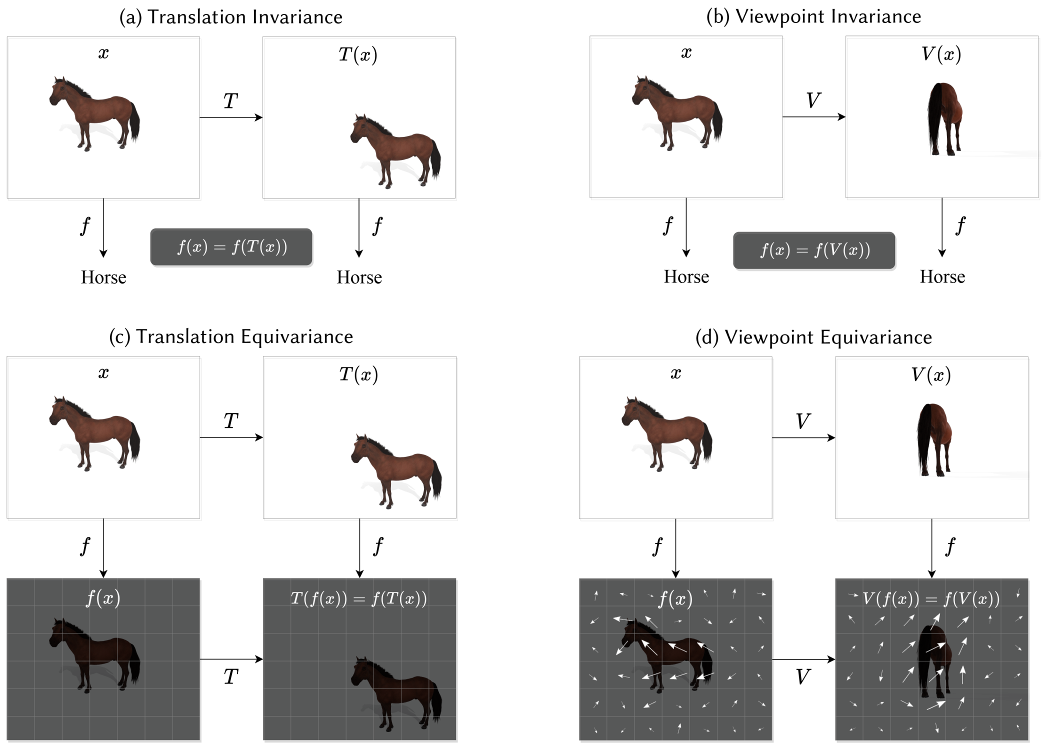

Fig. 2. Depiction of **invariance** and **equivariance** properties in a visual setting. Translation (or shift) is denoted by T, viewpoint (perspective) projection is denoted by V, and $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ denotes a feature mapping. (a) E.g., a CNN with **Global Average Pooling (GAP)**. (b) Capsule networks are *explicitly* wired to try to produce viewpoint invariant predictions. (c) Convolution in CNNs, whereby changes in input translation lead to equivalent (*place-coded*) changes in neural activities. (d) Cartoon example of capsule layers, which look to capture *rate-coded* viewpoint equivariance in neural activity vectors called capsules.

very good at extrapolating object appearance to novel viewpoints. However, this is not the case for standard CNNs [51], as they often struggle to generalize to viewpoints not seen during training. As explained later, unlike CNNs, capsule networks are explicitly wired to try to capture viewpoint invariance in the network's weights to produce viewpoint-invariant predictions. Capsules attempt to encode explicit pose representations of parts and objects hierarchically, as any change in viewpoint can be modeled by a simple *linear* operation on these poses. As explained next, capsule networks look to go beyond invariance to more complex (approximately) equivariant inference.

2.2 Equivariance

The success of CNNs can be largely attributed to their ability to exploit translation *symmetry* to reduce sample complexity. Indeed, the convolution operator and weight sharing provide the useful property of *equivariance* under translation, enabling efficient spatial transfer of knowledge. Naturally, much research in recent years has focused on exploiting other transformations and symmetries such as rotation and scale to improve statistical efficiency [19, 51, 54, 129]. Two fields of particular interest are Group CNNs [19, 20, 25, 103] and capsule networks [22, 50, 64, 99, 101, 106], which are both predicated upon the notion that intermediate neural network layers should *not* be fully invariant. This is primarily because the relative pose of local features ought to be preserved for deeper layers in the network to aid in generalization to new transformations not seen during training [19, 51].

Translation Equivariance. One of the simplest examples of equivariance is *translation equivariance* afforded by the convolution operation in CNNs [71]. As shown in Figure 2(c), a change in translation of the input leads to equivalent changes in neural activities. More formally, f is equivariant w.r.t. translation T if:

$$f(T(x)) = T(f(x)), \tag{2}$$

where f denotes the convolution operation in this example. In other words, we can first translate x then convolve, or first convolve x then translate to obtain the same output. In this case, this is known as *place-coded* equivariance, since a discrete change in the input x results in a discrete change in which neurons are used to encode it.

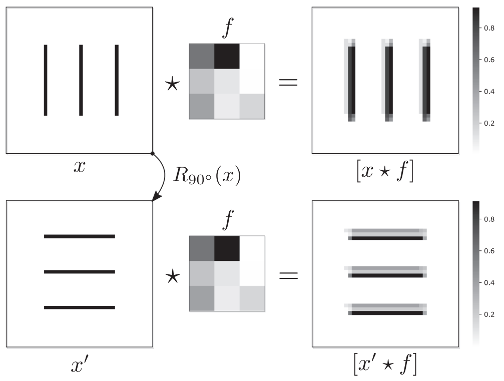

Fig. 3. Example of lack of equivariance in convolution. Each filter weight was sampled as: $f_i \sim \text{Uniform}(0, 1)$. Unlike the translation case, the output $[x' \star f]$ is *not* simply a rotated version of $[x \star f]$, as convolution is *not* rotation-equivariant.

Viewpoint Equivariance. It is more challenging to capture *viewpoint equivariance* in a model (see Figure 2(d)); that is, changes in viewpoint that lead to equivalent changes in neural activities. Like before, f is said to be equivariant w.r.t. viewpoint (perspective) projection V of the input x if: $f(V(x)) = V(f(x))$. Convolution is *not* equivariant to transformations other than translation, which makes it challenging for CNNs to deal with viewpoint changes [19, 51] (see Figure 3). As explained later, capsule networks [50, 64, 106] look to move from *place-coded* to *rate-coded* viewpoint equivariance in the final layers, whereby a real-valued change in the input results in an equivalent real-valued change in neuronal output (capsule pose vectors), but which neurons are used for coding the input stays the same.

2.3 Capsule Network Foundations

Although capsule networks have taken on several different forms since their inception [50, 51, 64, 106], they are generally built upon the following core assumptions and premises [22]:

(i) Capturing **equivariance** w.r.t. viewpoints in neural activities, and **invariance** in the network's weights;
(ii) High-dimensional coincidences are effective feature detectors, e.g., using the dot product to compute the similarity between neural activity vectors;
(iii) Viewpoint changes have **non-linear** effects on pixel intensities but **linear** effects on part-object relationships;
(iv) Object parts belong to a single object, and each location contains at most a single object.

In theory, a perfect instantiation of the above premises could yield more sample-efficient models, which leverage robust representations to better *generalize* to unseen cases. Unlike current methods, humans can often extrapolate object appearance to novel viewpoints even after a single initial observation. Evidence suggests that this is because we impose coordinate frames on objects [48, 102]. Capsules imitate this concept by representing neural activities as poses of objects with respect to a coordinate frame imposed by an observer and attempt to *disentangle* salient features of objects into

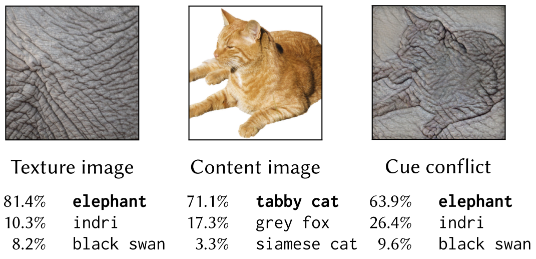

Fig. 4. Evidence of current CNNs (ResNet-50) being biased towards textures rather than using shapes like humans. Applying the "elephant" texture leads to misclassification. Figure from Reference [34].

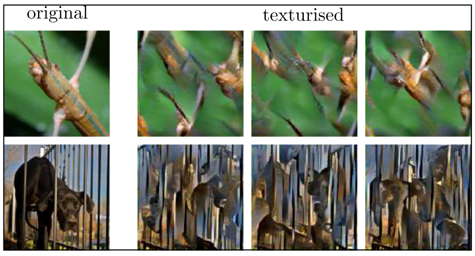

Fig. 5. Example of texture bias in CNNs. CNNs still achieve high accuracy on the texturized images, whereas humans do not, due to the loss of global shape information. Figure from Reference [13].

their composing parts. This is reminiscent of inverse graphics [67] but is not explicitly enforced in capsule formulations, since the learned pose matrices are not constrained to be interpretable geometric forms. Capsule networks can also be viewed as an extension of the successful inductive biases already present in CNNs by wiring in some additional complexity to deal with viewpoint changes using vector-valued neural activities. The desired effect is to produce *viewpoint invariant* predictions and align the learned representations with those perceptually consistent to humans, such that adversarial examples become less effective [10, 98].

2.3.1 *The Picasso Problem.*

In the ideal case, capsule networks address the "Picasso problem": I.e., images of an *object* containing all the right *parts*—but that are not in the correct spatial relationship—are often misclassified as said *object* by typical DL-based systems. To gain some intuition, consider the example of an image of a person's face (*object*), whereby the positions of the various *parts* of the face (e.g., mouth, eyes, and nose) are shuffled, and the image is still (wrongly) classified as a person's face. As depicted in Figure 6, although every image is composed of the same *parts*, only the green bordered examples ought to be labeled as a person's face [108], since all the parts are in agreement w.r.t. the face.

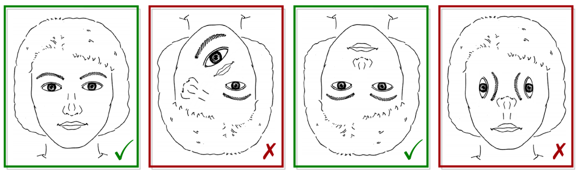

Fig. 6. Example of the "Picasso problem." Relative part-object relationships ought to be preserved if we are to label each image as a person's face. Inspired by Figure 1 from References [103, 108].

Classifying an image of an object viewed from a very different angle than those seen during model training also often leads to misclassification. Intuitively, this occurs because typical modern vision systems such as CNNs are not wired to explicitly model relative positions and spatial relationships between parts and objects. Instead, they tend to focus on detecting the most generally discerning properties of the input in the hope it produces the correct outcome [34, 51]. As discussed later, CNNs focus much more on textures than shapes, unlike humans [34] (see Figures 4, 5). One inefficient way of mitigating the above issue is to use data augmentation to provide the CNN model with examples of objects from all possible angles. However, this approach is not general and can become infeasible in real-world scenarios due to lack of data. A more efficient way to solve this problem would be to decompose the images into their constituent parts and objects and use the linearity of part-object spatial relationships (i.e., pose matrix multiplication used in computer graphics) to generalize to *all* viewpoints—which is the goal of capsule networks.

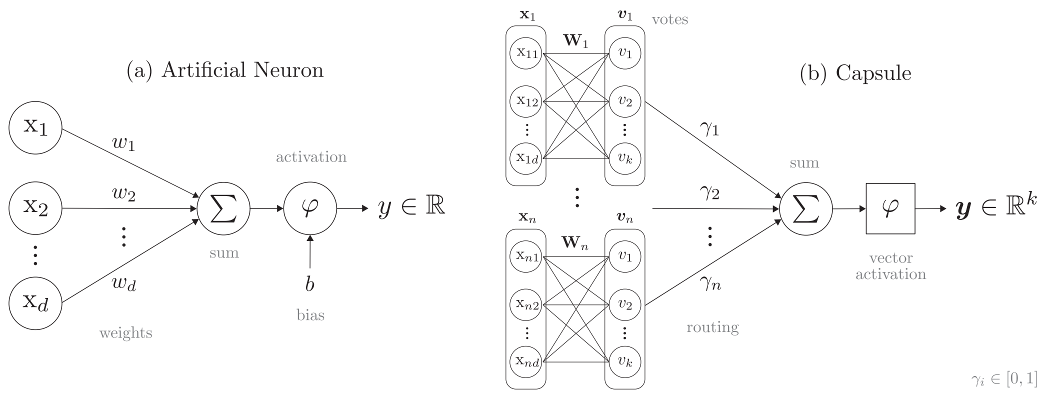

Fig. 7. (a) Illustration of the artificial neuron as found in standard neural networks. The artificial neuron receives an input signal vector $\mathbf{x} \in \mathbb{R}^d$ and outputs a scalar quantity y. (b) Depiction of a capsule as found in capsule networks. Unlike the neuron, a capsule receives n input signal vectors $\mathbf{x}_i \in \mathbb{R}^d$ and outputs a vector $\boldsymbol{y} \in \mathbb{R}^k$ of neural activities.

Sub-sampling & Convolution. The issue of poor generalization to novel viewpoints in CNNs is exacerbated by *pooling* (sub-sampling) operators, which discard potentially pose-aware information in favor of training/inference speed and performance [50, 106]. Indeed, there is strong evidence that modern deep CNNs do not parse images like humans and that they are in fact biased towards textures and other properties rather than shapes [13, 34, 35]. For examples of this phenomenon, see Figures 4 and 5. These biases are also corroborated by the fact that adversarial examples are often visually indistinguishable to humans [38, 118]. On that note, Reference [98] showed that capsule networks use features that are more aligned with human perception, and therefore have the potential to address the central issue of adversarial examples. Moreover, Reference [50] demonstrated that capsule networks can better generalize to novel viewpoints compared to CNNs of a similar size, and Reference [106] showed that they are considerably better than CNNs at recognizing overlapping digits. Although research on capsule networks is still in its infancy, there are representational reasons for believing that it is a better approach to vision, and these early results highlight their potential [50, 106].

2.3.2 *Capsule Networks.*

To explain what a capsule network is in simple terms, we begin by comparing the traditional *artificial neuron* found in standard NNs and the *capsule* as found in a capsule network (shown in Figure 7). Note that, in this section, we focus on providing a brief description of the main sub-components of capsule networks, and we introduce some general notation that will aid in understanding more intricate capsule formulations described later on.

Capsule. A capsule is a group of artificial neurons. It is a biologically inspired structure based on *hypercolumns* in the brain, wherein groups of tightly connected nearby neurons are thought to represent vector-valued units that can transmit not only scalar quantities but a set of coordinated values [10]. The transmission of coordinated values can increase the flexibility of an output signal. More formally, a capsule is a parameterized function $\mathbf{c}(\mathcal{X}; \mathcal{W}) : \mathbb{R}^{N \times D} \to \mathbb{R}^P$, where $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$ is a set of N input signals $\mathbf{x}_i \in \mathbb{R}^D$, and $\mathcal{W} = \{\mathbf{W}_i\}_{i=1}^N$ is a set of N transformation weight matrices $\mathbf{W}_i \in \mathbb{R}^{P \times D}$. A *dynamic routing*[1] process—akin to clustering—gives rise to *routing* coefficients: $\boldsymbol{\gamma} = \{\gamma_i\}_{i=1}^N,\ \gamma_i \in [0, 1]$, which represent the affinity between each input signal vector $\mathbf{x}_i$ and the

[1]How this routing process is defined is explained in detail later.

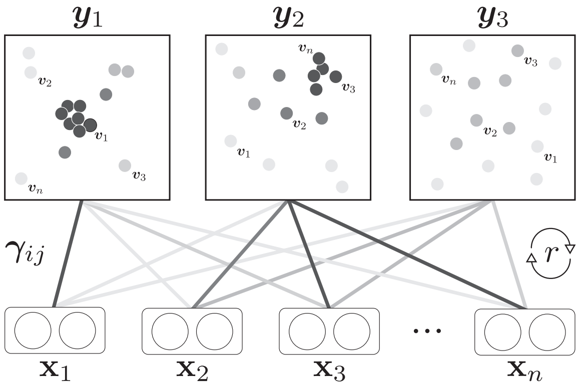

Fig. 8. A 2D example of capsule routing. All n part capsules $\mathbf{x}_i \in \mathbb{R}^2$ in the lower layer *vote* for each object capsule $\boldsymbol{y}_j \in \mathbb{R}^2$ above by: $\boldsymbol{v}_{j|i} = \mathbf{W}_{ij}\mathbf{x}_i$. If the votes for an object capsule agree and form a cluster, then there is iterative feedback to increase the routing weights $\boldsymbol{\gamma}_{ij}$ between the object capsule and the part capsules' votes that form the cluster, while decreasing them for other object capsules.

output capsule in question. As explained later in Section 2.3.3, the process by which these routing coefficients are obtained takes into account the context from other capsules. Last, an activation function $\varphi(\cdot)$ is applied, and the output capsule $\boldsymbol{y} \in \mathbb{R}^P$ is given by

$$\boldsymbol{y} = \varphi\Big(\sum_{i=1}^{N} \gamma_i \mathbf{W}_i \cdot \mathbf{x}_i\Big). \tag{3}$$

Notice that, unlike the artificial neuron, a capsule outputs a vector $\boldsymbol{y} \in \mathbb{R}^P$ of neural activities rather than a scalar $y \in \mathbb{R}$. Indeed, a capsule operates on a set of N input signal vectors and N parameter matrices, rather than a single input signal vector and weight vector. The neural activities within a capsule aim to encode the various properties of the entity it learns to represent, such as its pose, color, size, texture, and so on.

Capsule Layers. In adjacent capsule layers, we have $i = 1, \ldots, N$ lower-level *part* capsules $\mathbf{x}_i \in \mathbb{R}^D$ (considered the inputs), and $j = 1, \ldots, M$ higher-level *object* capsules $\boldsymbol{y}_j \in \mathbb{R}^P$ (considered the outputs). The initial part capsules (primary capsules) are typically extracted from the raw input (e.g., images) via convolution, and the object capsules of a layer ℓ become the part capsules of $\ell + 1$ and so on hierarchically until the final layer in the network.

2.3.3 *What Is Capsule Routing?*

Capsule routing is a non-linear, clustering-like process that occurs between adjacent capsule layers. The goal of capsule routing is to dynamically assign *part* capsules $i = 1, \ldots, N$ in layer ℓ to *object* capsules $j = 1, \ldots, M$ in layer $\ell + 1$ by iteratively adjusting routing coefficients $\boldsymbol{\gamma} \in \mathbb{R}^{N \times M}$, where $0 \leq \gamma_{ij} \leq 1$. These routing coefficients are like an attention matrix that modulates the outputs as a weighted average of the inputs. For a simple example of capsule routing in 2D, see Figure 8. This type of "routing-by-agreement" is a dynamic alternative to the primitive form of routing implemented by max-pooling, whereby neurons in the upper layer ignore all but the most active feature activation in a local pool in the layer below. As shown in the example Figure 6, rather than merely detecting whether certain parts/objects are present anywhere in an input image like pooling CNNs, capsule routing aims to detect objects by looking for coherent agreement between the pose of discovered parts and performing equivariant inference. Dynamic routing has been shown to be an effective way to implement the "explaining away" that is needed for segmenting overlapping objects and generalizing to novel viewpoints [50, 106].

3 Capsule Routing Mechanisms

This section provides an in-depth overview of the most prominent capsule routing algorithms in the literature. Our exposition loosely follows chronological order, starting with the seminal works on capsules and ending with more recent research.

3.1 Transforming Autoencoders

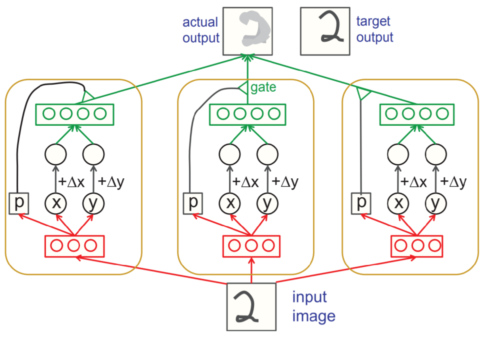

Fig. 9. The transforming autoencoder for modeling translations. There are three capsules and each one has three recognition units (red) and four generation units (green). Figure from Reference [51].

The idea of using capsules instead of neurons as the building blocks in a neural network was first introduced by Reference [51]. All subsequent work on capsules is built upon the premises laid out in this work, such as pose agreement and vector-valued neural activities. The authors showed how a neural network can be used to learn features that output a vector of instantiation parameters (capsule) and argued that this is a better way of dealing with transformations of the input. They proposed the transforming autoencoder as a way to learn the first level of capsules, whereby pixel intensities are converted to pose parameters. As shown in Figure 9, transforming autoencoders can receive the input image and a desired shift, Δx and Δy, and output the shifted input by merging information from generative capsule units. In Figure 9, p is the probability that the visual entity modeled by a particular capsule is present in the image. The authors also studied the prediction of more complex 2D transformations and changes in 3D viewpoint by using 3×3 matrix representations of the desired Δ's, demonstrating the merit of the approach.

3.2 Dynamic Routing between Capsules

The idea of routing-by-agreement was first introduced in the seminal work in Reference [106], and since then many other variants of capsule routing have been proposed. Their routing process is shown in Algorithm 1, whereby vectors $\mathbf{x}_i \in \mathbb{R}^D$ of lower layer capsules are transformed by weights $\mathbf{W}_{ij} \in \mathbb{R}^{D\times P}$ to make predictions for the vectors $\boldsymbol{y}_j \in \mathbb{R}^P$ of higher layer capsules. If a lower layer capsule i (e.g., encoding a nose) predicts the properties of a possible parent capsule j (e.g., encoding a face) with high accuracy, then there is top-down feedback that increases the affinity (routing coefficient γ_{ij}) between them. The proposed capsule network architecture in Reference [106] is shown in Figure 10.

3.2.1 Capsule Vector Activation. This version of capsules uses the length of the output capsule vector to represent the probability that the *entity* encoded by that capsule is present in the input [106]. To that end, the authors proposed the following non-linear "squashing" function to activate every jth capsule:

$$\boldsymbol{y}_j = \frac{\left\|\mathbf{s}_j\right\|^2}{1+\left\|\mathbf{s}_j\right\|^2}\frac{\mathbf{s}_j}{\left\|\mathbf{s}_j\right\|}, \qquad \mathbf{s}_j = \sum_i \gamma_{ij}\mathbf{W}_{ij}\cdot\mathbf{x}_i. \tag{4}$$

The routing coefficients γ_{ij} are iteratively updated based on the *agreement* between the output $\boldsymbol{y}_j$ of each higher layer capsule j and the prediction (votes) $\boldsymbol{v}_{j|i}$ made by each lower layer capsule i. The agreement is measured by the scalar product: $\alpha_{ij} = \boldsymbol{v}_{j|i}\cdot\boldsymbol{y}_j$. To gain some intuition about this procedure, imagine $\boldsymbol{y}_j$ and $\boldsymbol{v}_{j|i}$ are both unit vectors, i.e., their magnitudes are: $|\boldsymbol{y}_j| = 1$ and $|\boldsymbol{v}_{j|i}| = 1$. Then, the dot product between them is equal to $\cos\theta$, where θ is the angle between the

ALGORITHM 1: Dynamic Routing-by-agreement

```
1: function ROUTING(x, W, r)
2:     ∀ $i, j$ capsules in layer $\ell$ and $\ell + 1 : \gamma_{ij} \leftarrow 0$
3:     ∀ $i, j : \boldsymbol{v}_{j|i} \leftarrow \mathbf{W}_{ij} \cdot \mathbf{x}_i$                ▷ voting
4:     for $r$ iterations do
5:         ∀ $i \in \ell : \boldsymbol{\gamma}_i \leftarrow \mathtt{softmax}(\boldsymbol{\gamma}_i)$    ▷ routing weights
6:         ∀ $j \in \ell + 1 : \mathbf{s}_j \leftarrow \sum_i \gamma_{ij} \boldsymbol{v}_{j|i}$
7:         ∀ $j \in \ell + 1 : \boldsymbol{y}_j \leftarrow \mathsf{squash}(\mathbf{s}_j)$     ▷ Equation (4)
8:         ∀ $i, j : \alpha_{ij} \leftarrow \boldsymbol{v}_{j|i} \cdot \boldsymbol{y}_j$           ▷ agreement
9:         ∀ $i, j : \gamma_{ij} \leftarrow \gamma_{ij} + \alpha_{ij}$           ▷ update
10:    return $\boldsymbol{y}_j$
```

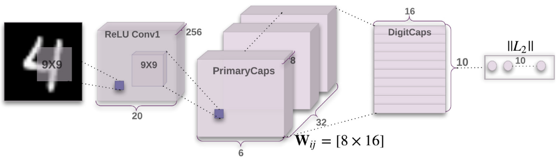

Fig. 10. A simple capsule network with three layers. The `PrimaryCaps` are the lowest level of multi-dimensional neuronal activity (each capsule is an 8D vector). The length of the activity vector of each capsule in the `DigitCaps` layer indicates the presence of each class. Figure from Reference [106].

two vectors. The resulting agreement α_{ij} is added to γ_{ij}, updating part-object affinities based on the extent to which the vectors $\boldsymbol{y}_j$ and $\boldsymbol{v}_{j|i}$ point in the same direction.

3.2.2 *Margin Loss Function.*

The objective function presented in Reference [106] for learning capsule network parameters leverages the length of the capsule vectors to represent the probability that a capsule's entity is present in the input. Note that the norm of the last layer capsule vector $||\boldsymbol{y}_k||$, representing class k must be (long) close to 1, if and only if (iff) an image belonging to class k is present in the input. With that in mind, the (multiple) margin loss used is [106]:

$$\mathcal{L}_{\text{margin}} = \sum_k T_k \max(0, m^+ - ||\boldsymbol{y}_k||)^2 + \lambda(1 - T_k) \max(0, ||\boldsymbol{y}_k|| - m^-)^2, \tag{5}$$

where $T_k = 1$ iff a (digit of class) k is present, $m^+ = 0.9$, $m^- = 0.1$, and $\lambda = 0.5$. The λ down-weighting of the loss for absent (digit) classes stops the initial learning from shrinking the lengths of the activity vectors of all the digit capsules. The authors opt for this multi-margin loss function over standard **cross-entropy (CE)** used in CNNs to more easily accommodate multi-label classification tasks.

3.3 Matrix Capsules with EM-routing

More recently, a new version of capsules was proposed [50], which overcomes the following deficiencies of [106]:

(i) Using the length of the pose vector to represent the probability that an entity is present requires an unprincipled non-linearity ("squashing") that prevents the use of typical objective functions [50]. Instead, they propose to separate the probability of existence from the pose vector.

(ii) Using the cosine of the angle between vectors to measure agreement makes the system insensitive to small differences between good and very good agreements.

(iii) Using a vector of length D to represent poses (unnecessarily) increases the number of transformation weights. They propose to use a matrix with D elements instead, which reduces parameters from D^2 to D.

Rather than using vector capsules $\mathbf{x}_i \in \mathbb{R}^D$ as before, the authors use $\mathbf{M}_i \in \mathbb{R}^{\sqrt{D} \times \sqrt{D}}$ capsule pose matrices and a separate activation probability $a \in \mathbb{R}$ to represent the presence of the entity modeled by each capsule [50]. They also presented a new capsule network architecture (see Figure 11) featuring convolutional capsules and proposed an alternative routing-by-agreement procedure based on the **Expectation-Maximization (EM)** algorithm [23].

ALGORITHM 2: Expectation-maximization Routing

```
1:  function EM-ROUTING($a$, $\mathbf{M}$, $\mathbf{W}$, $r$)
2:      $\forall\ i, j$ capsules in layer $\ell$ and $\ell+1$: $\gamma_{ij} \leftarrow M^{-1}$
3:      $\forall\ i, j : \mathbf{V}_{j|i} \leftarrow \mathbf{M}_i \cdot \mathbf{W}_{ij}$                ▷ voting
4:      for $r$ iterations do
5:          $\forall\ i \in \ell : \gamma_{ij} \leftarrow \gamma_{ij} \odot a_i$                ▷ routing weights
6:          $\forall\ j \in \ell+1 : \mu_j, \sigma_j \leftarrow$ M-STEP($\gamma$, $\mathbf{V}$)
7:              $\text{cost}^h \leftarrow (\beta_u + \log \sigma_j^h) \sum_i \gamma_{ij}$
8:              $a'_j \leftarrow \text{sigmoid}(\lambda(\beta_a - \sum_h \text{cost}^h))$        ▷ activate
9:          $\forall\ i \in \ell : \gamma_i \leftarrow$ E-STEP($\mu$, $\sigma$, $a'$, $\mathbf{V}$)          ▷ update
10:     return $a$, $\mathbf{M}$
```

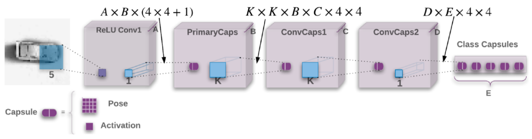
$A \times B \times (4\times4+1)$
$K \times K \times B \times C \times 4 \times 4$
$D \times E \times 4 \times 4$
ReLU Conv1
A
PrimaryCaps
B
ConvCaps1
C
ConvCaps2
D
Class Capsules
5
1
K
K
1
E
Capsule
=
Pose
Activation

Fig. 11. The capsule network architecture with three convolutional capsule layers (ConvCaps) used in Reference [50]. Rather than connecting all part capsules in a lower layer to all object capsules in a higher layer, convolutional object capsules only receive votes from part capsules within their receptive field. The number of weights in each layer is shown above.

3.3.1 Matrix Capsule Voting. In Reference [50], the voting procedure from capsules i in a lower layer ℓ_i for the pose matrices of capsules j in a higher layer ℓ_j is: $\mathbf{V}_{j|i} = \mathbf{M}_i \cdot \mathbf{W}_{ij}$, where $\mathbf{W}_{ij} \in \mathbb{R}^{4\times4}$, $\mathbf{V}_{j|i}$ denotes the vote from the ith part capsule for the jth object capsule, and $\mathbf{W}_{ij}$ is the transformation weight matrix. Since both the pose matrices and the transformations weights are 4×4, each capsule's vote is $\mathbf{V}_{j|i} \in \mathbb{R}^{4\times4}$. See Figure 12 for a simple 2D translation example. We can motivate the use of 4×4 matrices through a geometric interpretation: I.e., 4×4 transformation matrices are commonly used in 3D computer graphics under homogeneous coordinates for perspective projection [5]. A 4×4 matrix can represent the following transformations among others: translation, rotation, reflection, glides, scale, contraction, expansion, shear, dilation, and so on. Assuming a capsule network is able to extract sensible entities from the input, 4×4 pose and transformation weight matrices are theoretically sufficient. It is worth noting the distinction between the voting procedure in Equation (4) and this one: I.e., in the former, the vote is calculated via the matrix-vector product, $\boldsymbol{v}_{j|i} = \mathbf{W}_{ij}\mathbf{x}_i$, whereas, in the latter, we have a matrix-matrix product: $\mathbf{V}_{j|i} = \mathbf{M}_i\mathbf{W}_{ij}$. Once again, we can provide a geometric interpretation that justifies the order of the product, since due to the non-commutativity of square matrices: $\mathbf{M}_i\mathbf{W}_{ij} \neq \mathbf{W}_{ij}\mathbf{M}_i$. In geometric terms, $\mathbf{M}_i\mathbf{W}_{ij}$ applies a transformation on the pose matrix $\mathbf{M}_i$ defined by matrix $\mathbf{W}_{ij}$, whereas $\mathbf{W}_{ij}\mathbf{M}_i$ would wrongly imply that $\mathbf{W}_{ij}$ is the pose matrix and $\mathbf{M}_i$ is the transformation matrix.

3.3.2 Convolutional Capsules. Hinton et al. [50] also introduced the idea of convolutional capsules, whereby the connectivity between capsules in adjacent layers follows that of a convolutional neural network. That is, rather than performing a regular convolution (sharing scalar feature detector kernels across the input), convolutional capsule layers share transformation weight matrices $\mathbf{W}_{ij}$ spatially across input capsules. Multiple convolutional capsule layers are then stacked to build a deep capsule network (see Figure 11). This extension in the EM paper makes intuitive sense, as, ideally, we would like to retain the ability to generalize knowledge across all spatial locations in the image like CNNs while replacing basic max/average pooling operations in favor of capsule routing-by-agreement.

3.3.3 Expectation-maximization Routing. Similar to Reference [106], a non-linear procedure to route between adjacent capsule layers they call EM-routing (see Algorithm 2) is proposed in Reference [50]. Concretely, the procedure is a version of the EM algorithm [23], which iteratively adjusts the means, variances, and activation probabilities of the capsules in layer $\ell+1$ and the assignment probabilities between all capsules (routing weights $\boldsymbol{\gamma}$). Unlike Dynamic routing [106], EM-routing fits Gaussian distributions on the votes coming from part capsules i to object capsule j poses: $\mathcal{N}(\mathbf{M}_j \mid \boldsymbol{\mu}_j, \boldsymbol{\sigma}_j^2)\ \forall j$, where each capsule j (of N_j total object capsules in layer $\ell+1$) has a

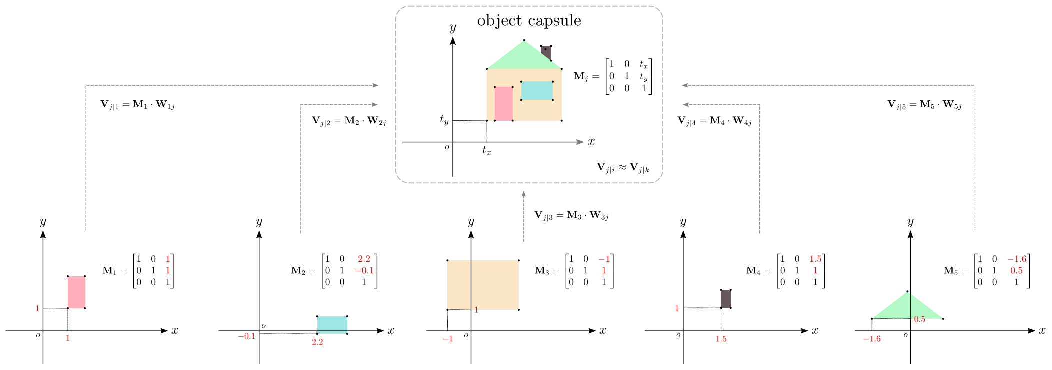
object capsule
$\mathbf{M}_j = \begin{bmatrix}1 & 0 & t_x\\0 & 1 & t_y\\0 & 0 & 1\end{bmatrix}$
t_y
t_x
$\mathbf{V}_{j|i} \approx \mathbf{V}_{j|k}$
$\mathbf{V}_{j|1} = \mathbf{M}_1 \cdot \mathbf{W}_{1j}$
$\mathbf{V}_{j|2} = \mathbf{M}_2 \cdot \mathbf{W}_{2j}$
$\mathbf{V}_{j|3} = \mathbf{M}_3 \cdot \mathbf{W}_{3j}$
$\mathbf{V}_{j|4} = \mathbf{M}_4 \cdot \mathbf{W}_{4j}$
$\mathbf{V}_{j|5} = \mathbf{M}_5 \cdot \mathbf{W}_{5j}$
$\mathbf{M}_1 = \begin{bmatrix}1 & 0 & 1\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_2 = \begin{bmatrix}1 & 0 & 2.2\\0 & 1 & -0.1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_3 = \begin{bmatrix}1 & 0 & -1\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_4 = \begin{bmatrix}1 & 0 & 1.5\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_5 = \begin{bmatrix}1 & 0 & -1.6\\0 & 1 & 0.5\\0 & 0 & 1\end{bmatrix}$

Fig. 12. How capsules deal with viewpoint changes (2D translation example). An object can be detected by looking for agreement between votes $\mathbf{V}_{j|i}$ for its pose matrix $\mathbf{M}_j$. Lower-level (part) capsules produce a vote by multiplying their pose matrix $\mathbf{M}_i$ by a learned viewpoint-invariant transformation matrix $\mathbf{W}_{ij}$. Agreement between multiple parts means that their votes are similar: $\mathbf{V}_{j|i} \approx \mathbf{V}_{j|k}$, for $i \neq k$, which is unlikely to occur by chance in high dimensions [50, 106]. As the viewpoint changes, all the pose matrices change in a coordinated manner, so any voting agreement will persist [50].

diagonal covariance matrix with h components: $\boldsymbol{\sigma}_j^2 \in \mathbb{R}^h$. Following Algorithm 2 closely, EM-routing iterates between updating the means $\boldsymbol{\mu}_j$, variances $\boldsymbol{\sigma}_j^2$, and activations $\boldsymbol{a}_j$ of capsules j while holding the routing coefficients γ_{ij} fixed (M-step) and updating γ_{ij} holding $\boldsymbol{\mu}_j$, $\boldsymbol{\sigma}_j^2$, and $\boldsymbol{a}_j$ fixed (E-step). For a more detailed explanation, the reader may refer to Reference [50].

3.3.4 *Capsule Activation.*

As shown in Algorithm 2, vote agreement is measured using the variance $\boldsymbol{\sigma}_j$ of each object capsule's Gaussian, which is then weighted by its *support*: $\sum_i \gamma_{ij}$, i.e., the number of part capsules i assigned to object capsule j. To set the activation probability a_j for a particular object capsule j, Reference [50] compares the description lengths (energies $-\beta_u$ and $-\beta_a$ that are learned discriminatively) of two different ways of coding the poses of the activated capsules i assigned to j by the routing procedure. The difference in the two energies is put through a logistic function to determine the activation probability of each object capsule j, noting that the logistic function computes the distribution: Bernoulli(p), which minimizes free energy when the difference in the energies is its argument (see line 8 in Algorithm 2).

3.3.5 *Spread Loss Function.*

The "spread" loss function used in Reference [50] directly maximizes the gap between the activation of the (final layer) capsule representing the target class a_t and the other class capsules:

$$\mathcal{L}_{\text{spread}} = \sum_{i \neq t} \mathcal{L}_i, \qquad \mathcal{L}_i = \max(0, m - (a_t - a_i))^2, \tag{6}$$

where the margin m is linearly increased during training from 0.2 to 0.9, avoiding dead capsules in the earlier layers. It is reported in Reference [50] that EM-routing matrix capsules outperform Reference [106] and significantly outperform comparable size CNNs on viewpoint-invariance and adversarial robustness tasks.

3.4 Capsule Routing via Variational Bayes

Ribeiro et al. [101] proposed **Variational Bayes (VB)** routing as a way to address some of the inherent drawbacks of EM-routing [50] encountered by various other authors, such as training instability and reproducibility [41, 45, 120]. A problem with EM-routing is that variance-collapse

ALGORITHM 3: Variational Bayes Routing

```
 1: function VB-ROUTING($\boldsymbol{a}$, M, W, $r$)
 2:     $\forall\ i, j$ capsules in layer $\mathcal{L}_i$ and $\mathcal{L}_j$: $\gamma_{ij} \leftarrow N_j^{-1}$
 3:     $\forall\ i, j : \mathbf{V}_{j|i} \leftarrow \mathbf{M}_i \cdot \mathbf{W}_{ij}$                        ▷ voting
 4:     $\forall\ j \in \mathcal{L}_j : \alpha_0, \mathbf{m}_0, \kappa_0, \mathbf{\Psi}_0, \nu_0$                 ▷ initialize priors
 5:     for $r$ iterations do
 6:         $\forall\ i \in \mathcal{L}_i : \gamma_{ij} \leftarrow \gamma_{ij} \odot \boldsymbol{a}_i$                ▷ routing weights
 7:         $\forall\ j \in \mathcal{L}_j$ : UPDATE $q^{\star}(\boldsymbol{\pi}, \boldsymbol{\mu}, \Lambda)$
 8:         $\forall\ i \in \mathcal{L}_i$ : UPDATE $q^{\star}(\boldsymbol{z})$
 9:     $\forall\ j \in \mathcal{L}_j : a_j' = \sigma(\beta_a - (\beta_u + \mathbb{E}[\ln \boldsymbol{\pi}_j] + \mathbb{E}[\ln \det(\Lambda_j)]))$
10:     return $\boldsymbol{a}'$, M
```

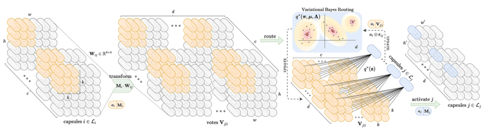
Variational Bayes Routing
$q^*(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})$
w
h
c
k
$\mathbf{W}_{ij} \in \mathbb{R}^{4\times4}$
transform
$\mathbf{M}_i \cdot \mathbf{W}_{ij}$
a_i $\mathbf{M}_i$
capsules $i \in \mathcal{L}_i$
d
votes $\mathbf{V}_{j|i}$
route
UPDATE
a_i $\mathbf{V}_{j|i}$
$a_i \odot \mathbf{r}_{ij}$
$q^*(\mathbf{z})$
capsules $j \in \mathcal{L}_j$
$\mathbf{V}_{j|i}$
activate j
a_j $\mathbf{M}_j$
w'
h'
capsules $j \in \mathcal{L}_j$

Fig. 13. Depiction of VB-routing between convolutional capsule layers. Each capsule has an activation a and a pose matrix $\mathbf{M} \in \mathbb{R}^{4\times4}$. Parent capsules $j \in \mathcal{L}_j$ (blue) only receive votes from child capsules $i \in \mathcal{L}_i$ (orange) within their receptive field. c and d denote the number of child and parent capsule types (channels), respectively. Figure from Reference [101].

singularities can occur when an object capsule claims sole custody of a part capsule, yielding infinite likelihood and zero variance. To address this, Bayesian learning is brought to capsule networks by placing priors and modeling uncertainty over capsule parameters [101]. The advantages of VB-routing include: (i) flexible control over capsule complexity by tuning priors; (ii) number of effective object capsules is determined automatically; (iii) pathological solutions of EM-routing are addressed in a principled manner.

3.4.1 *Variational Bayes Routing.*

See Figure 13 for an illustration of VB-routing. The authors place conjugate priors over $\boldsymbol{\mu}$ and Λ, which are the mean and inverse covariance (precision) matrix of each object capsule's Gaussian distribution, and over $\boldsymbol{\pi}$, which are the mixing coefficients of the mixture model. The latent variables $\boldsymbol{z} = \{\boldsymbol{z}_i, \ldots, \boldsymbol{z}_{|\mathcal{L}_i|}\}$ are a set of one-hot vectors describing the cluster assignments of each of the lower capsules' votes $\mathbf{V}_{j|i}$ to higher capsules' Gaussian distributions. **Variational inference (VI)** [52, 57] of the above latent variables is performed [101]—analogously to Bayesian Gaussian mixture models [6, 12]—between all adjacent capsule layers. Unlike standard mixture models, here, every cluster (object capsule) has its own learnable matrix $\mathbf{W}_{ij}$ with which its datapoints (votes) are transformed, so every cluster sees a different view of the data [50, 101]. The *generative story* for each part capsule i is that of a mixture model with priors over object capsule j parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \Lambda.$, as follows:

(i) Choose an **object** capsule: $z_i \sim \text{Multinomial}(\boldsymbol{\pi})$, where $\boldsymbol{\pi} \sim \text{Dirichlet}(\boldsymbol{\alpha}_0)$;
(ii) Generate **part** capsule from jth **object's** Gaussian:

$$\mathbf{V}_{j|i} \mid z_i = j \sim \mathcal{N}(\boldsymbol{\mu}_j, \Lambda_j^{-1}), \quad \boldsymbol{\mu}_j \mid z_i = j \sim \mathcal{N}(\mathbf{m}_0, (\kappa_0 \Lambda_j)^{-1}), \quad \Lambda_j \mid z_i = j \sim \text{Wishart}(\mathbf{\Psi}_0, \nu_0). \tag{7}$$

The joint distribution of the model is: $p(\mathbf{V}, \boldsymbol{z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \Lambda) = p(\mathbf{V}|\boldsymbol{z}, \boldsymbol{\mu}, \Lambda)p(\boldsymbol{z}|\boldsymbol{\pi})p(\boldsymbol{\pi})p(\boldsymbol{\mu}|\Lambda)p(\Lambda)$, and the posterior is approximated with a factorized variational distribution over all the latent variables: $p(\boldsymbol{z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \Lambda|\mathbf{V}) \approx q(\boldsymbol{z})q(\boldsymbol{\pi}) \prod_{j \in \mathcal{L}_j} q(\boldsymbol{\mu}_j, \Lambda_j)$. To perform capsule routing, the authors iteratively optimize parent capsule parameter distributions, $q(\boldsymbol{\pi}, \boldsymbol{\mu}, \Lambda)$, with the responsibilities $\boldsymbol{z}$ over child capsules fixed, and re-evaluate the new expected responsibilities $q^{\star}(\boldsymbol{z})$ with the distributions over parent capsule parameters fixed. This approach leads to a variational EM algorithm (see Algorithm 3). For further details on the standard closed-form update equations of Bayesian mixture models the reader may refer to References [6, 12, 101].

3.4.2 *Agreement & Capsule Activation.*

To measure agreement, Reference [101] proposes to use the differential entropy of higher capsule j's Gaussian-Wishart variational posterior distribution $q^{\star}(\boldsymbol{\mu}_j, \Lambda_j)$. In simple terms, if the entropy of a capsule j's Gaussian is low, then that means the

ALGORITHM 4: Capsule Layer with Routing Uncertainty

```
1: function ConvCaps2D (a, M, W)
2:     $\forall\ i, j$ capsules in $\ell, \ell+1 : \mathbf{V}_{j|i} \leftarrow \mathbf{M}_i \cdot \mathbf{W}_{ij}$        ▷ voting
3:     $\forall\ i \in \ell : \boldsymbol{\pi}_0^{(i)} \in \mathbb{R}^{N_{i\to j}}$        ▷ set Dirichlet priors
4:     $\forall\ i \in \ell : \mathbf{z}^{(i)} \sim q_\phi(\mathbf{z}_{\ell,\ell+1})$        ▷ posterior sample
5:     $\forall\ j \in \ell+1 : \boldsymbol{\mu}_j, \boldsymbol{\sigma}_j \leftarrow \textsc{Route}\ (\mathbf{z}_{\ell,\ell+1}, \mathbf{V}_{j|i})$
6:     $\forall\ j \in \ell+1 : a'_j = \sigma\big(-\boldsymbol{\eta}_j \mathbb{E}(S_j)^{-1}\mathcal{H}(\mathbf{M}_j)\big)$        ▷ activation
7:     return a′, M
```

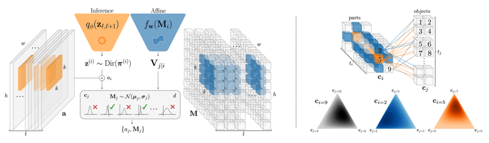
Fig. 14. (Left) Part capsule activations **a** and pose matrices **M** give rise to votes $\mathbf{V}_{j|i}$ and part-object connections **z** that are sampled from Dirichlet distributions during training. (Right) Part-object connectivity (kernel size $k = 2$). Figure from Reference [45].

votes received from capsules i agree (i.e., form a tight cluster) and vice versa. In practice, the authors approximate the entropy up to constant factors with: $\mathbb{H}[q^\star(\boldsymbol{\mu}_j, \boldsymbol{\Lambda}_j)] \approx \mathbb{E}[\ln\det(\boldsymbol{\Lambda}_j)]$. As outlined in Algorithm 3, this agreement measure is then weighted by the amount of *support* for each object capsule (mixing coefficient) and activated via the logistic function $\sigma(\cdot)$, where β_a, β_u are (optional) parameters learned discriminatively, similar to EM-routing [50]. Unlike EM [50] or Dynamic routing [106], however, capsules are only activated after the routing iterations are complete. The authors find that this can improve training stability without degrading performance.

3.5 Uncertainty in Capsule Routing

Sources of uncertainty in assembling objects via a composition of parts can arise from: (i) feature occlusions due to observed viewpoints; (ii) sensory noise in captured data; (iii) object symmetries for which poses may be ambiguous, such as spherical objects and/or parts. Recently, Reference [22] proposed a *global* (locally non-iterative) view of capsule routing based on representing the inherent *uncertainty* in part-object relationships by approximating a posterior distribution over part-object connections. In simple terms, the *local* routing iterations are replaced with VI *probabilistic* capsule network, leading to increased efficiency and improved performance on pose-aware benchmarks. In this way, they encourage global context to be taken into account when routing information by introducing *global* latent variables that have a direct influence on their end-to-end variational free-energy objective [22]. To represent uncertainty about part-object relationships in a capsule network, the posterior distribution $p(\mathbf{z}|\mathcal{D}, \mathbf{W})$ over part-object latent connections **z** given the data $\mathcal{D}$ is needed. However, exact inference is intractable for complex models [22]. To circumvent this, the authors [22] use stochastic VI tools to find the best approximation $q_\phi^\star(\mathbf{z})$ that minimizes $D_{\mathrm{KL}}(q_\phi(\mathbf{z})\ ||\ p(\mathbf{z}|\mathcal{D}, \mathbf{W}))$, where **z** are global latent part-object connection variables, and **W** are viewpoint-invariant transformation parameters, in a probabilistic capsule network with L total layers.

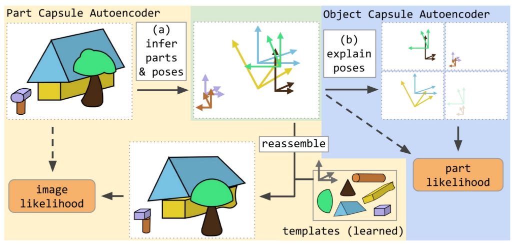

Fig. 15. The SCAE. The part capsule autoencoder segments input images into individual parts and their poses. The object capsule autoencoder rearranges the inferred parts into objects. The SCAE is trained by maximizing image and part log-likelihoods subject to sparsity constraints. Figure from Reference [64].

Priors. The authors [22] place a prior distribution $p(\mathbf{z}^{(i)})$ over each part capsule's $\mathbf{c}_i \in \ell$ connections to the object capsules they vote for $\mathbf{c}_j \in \ell + 1$ and assume fully factorized independence across layers: $p(\mathbf{z}) = \prod_{\ell=1}^{L-1}\prod_{i=1}^{N_i} p(\mathbf{z}_\ell^{(i)})$, with $\mathbf{z}^{(i)} = (z_1, \dots, z_{N_{i\to j}}) \sim p(\mathbf{z}^{(i)})$, where the part-object connections vector of each capsule $\mathbf{c}_i$ is $\mathbf{z}^{(i)} \in \mathbb{R}^{N_{i\to j}}$, with $N_{i\to j}$ denoting the number of object capsules j that each part capsule i votes for in a particular layer (see Figure 14 for convolutional capsule voting example). After considering different choices of prior, the authors opted for Dirichlet priors due to a reduced parameter count. A mean-field variational approximation $q_\phi(\mathbf{z}_{\ell,\ell+1})$ to the (intractable) posterior on part-object connection variables is made between all adjacent capsule layers. The model is defined hierarchically, where the object capsules in layer ℓ are the part capsules of $\ell + 1$ and so on. The model is then fit end-to-end by maximizing a variational lower bound on the conditional marginal log-likelihood $\log p(\mathbf{y}|\mathbf{x})$ (see Reference [22] for further details).

3.5.1 Routing & Activating Capsules. As in previous work [50, 101], matrix capsules $\mathbf{M} \in \mathbb{R}^{4\times 4}$ and convolutional capsule voting are used (see Figure 14). During training, the authors [22] fit multivariate Gaussians $\mathbf{M}_j \sim \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j)$ on each object capsule's D=16 dimensional poses and randomly sample part-object connections from the approximate posterior at each capsule layer ℓ: $\mathbf{z}^{(i)} = (z_1, \ldots, z_{N_{i\to j}}) \sim q_\phi(\mathbf{z}_{\ell,\ell+1}), \forall \mathbf{c}_i \in \ell$, then calculate the parameters of each capsule's Gaussian. The procedure (see Algorithm 4) can be interpreted as *global* (locally non-iterative) routing, since the posterior: $q^\star_\phi(\mathbf{z}|\mathcal{D}) \approx p(\mathbf{z}|\mathcal{D}, \mathbf{W})$ is inferred for all layers at once, rather than performing *local* (iterative) inference of $\mathbf{z}$ in the E-step of EM-routing [50] between each pair of adjacent capsule layers.

To activate capsules, Ribeiro et al. [22] follow the general concept of Reference [101] and measure vote agreement via the average negative entropy of each capsule's Gaussian: $-\mathcal{H}[\mathcal{N}(\mathbf{M}_j \mid \boldsymbol{\mu}_j, \boldsymbol{\sigma}_j)] \triangleq -\frac{1}{D}\sum_{i=1}^{D} \log(\boldsymbol{\sigma}_j^{(i)}\sqrt{2\pi e})$. The *agreement* is weighted by the (normalized) *support* (# parts assigned to an object) for each capsule and activated using the logistic function $\sigma(\cdot)$, where $\mathbb{E}(S_j)$ is the average support each object capsule receives in a given layer, $S_j \sim \text{Binomial}(N_i, {N_j}^{-1})$, as shown in Algorithm 4. For further technical details please refer to Reference [22].

3.5.2 Uncertainty Quantification. With their method, the authors unlock *uncertainty* representation in capsule networks. To that end, they draw T Monte Carlo samples of part-object connections $\mathbf{z}$ from the approximate posterior and calculate the predictive entropy $\mathcal{H}(\widehat{\mathbf{y}}|\mathbf{x}, \mathbf{z}, \mathbf{W})$ of the model's output distribution with sampled $\mathbf{z}^t \sim q^\star_\phi(\mathbf{z}|\mathcal{D})$. Under full posterior learning, $q_\phi(\mathbf{z}, \mathbf{W})$, the pose transformation matrices $\mathbf{W}$ are also randomly sampled.

3.6 Stacked Capsule Autoencoders

Kosiorek et al. [64] introduced the **Stacked Capsule Autoencoder (SCAE)**, a seminal work on formulating CapsNets as an unsupervised capsule autoencoder that explicitly uses geometric relationships to reason about objects. This was the first work that combined ideas from Transforming Autoencoders [51] and EM routing capsules [50]—but unlike previous methods, inference in this model is amortized and performed by off-the-shelf neural encoders. The authors also used discovered objects to predict parts rather than using parts to predict objects as in previous capsule networks. Even though the training objective used in SCAE is not concerned with classification or clustering, it is the only method that achieves competitive results in unsupervised object classification without relying on mutual information. The proposed SCAE architecture can be seen in Figure 15, and it consists of two main parts: (i) the **Part Capsule Autoencoder (PCAE)** ; (ii) the **Object Capsule Autoencoder (OCAE)**.

In addition to the two stages mentioned above, there is also an earlier step that deals with abstracting away pixels and the part-discovery stage, called **Constellation Capsule Autoencoder (CCAE)**. CCAE uses two-dimensional points as parts, whose coordinates are given as input to the system. CCAE then learns to model the sets of points as arrangements of familiar constellations, each of which has been transformed by an independent similarity transform.

3.6.1 Part Capsule Autoencoder (PCAE). Although CCAE considers a part as a 2D point (x and y coordinates), for PCAE, each part capsule m has a six-dimensional pose x_m (two rotations, two translations, scale, and shear), a presence variance $d_m \in [0, 1]$, and a unique identity. Discovering part is formulated as an auto-encoding exercise: The *encoder* learns to infer the poses and presences of different part capsules, while the decoder learns an image template T_m for each part. In the case where a part exists, the corresponding template is affine-transformed with the inferred pose giving $\hat{T}_m$. Finally, all the transformed templates are arranged into the image.

3.6.2 Object Capsule Autoencoder (OCAE). This stage proceeds PCAE and resembles the processes involved in the CCAE. All the parameters that have been extracted and identified from PCAE need to be composed in a way that forms objects. This is achieved by providing concatenated poses $\mathbf{x}_m$, special features $\mathbf{z}_m$, and flattened templates T_m as input to the OCAE. This differs from the CCAE in that the part capsules presence probabilities d_m are fed into the OCAE's encoder to add bias to the attention mechanism of the Set Transformer [73] not to consider absent points. In addition, d_ms are also used to weight the part-capsules' log-likelihood, so we do not take absent points into account. This is achieved by raising the likelihood of the $m^t h$ part capsule to the power of d_m. Parts discovered by the PCAE have independent identities, therefore every part-pose is explained as an independent mixture of predictions from object capsules. The OCAE is trained by maximizing the likelihood of the detected parts, and it learns to discover further structure in previously identified parts, leading to sparsely activated object capsules. For more detailed information on the mathematical formulations of SCAE please refer to Reference [64].

3.7 Self-routing Capsule Networks

The fact that capsules specialize in disjoint regions of the feature space leads to them making multiple predictions based on the information that is made available to them for each region. At a layer level, this means that we have an ensemble of submodules that are activated differently per example—similar to a mixture of experts, where each expert specializes in different regions of input space. Motivated by this observation and the fact that routing-by-agreement is computationally costly, the authors in Reference [44] proposed a simpler self-routing strategy inspired by Mixture-of-experts. In **Self-Routing Capsule Networks (SR-CapsNet)** proposed by Reference [44], each capsule independently defines its routing coefficients without coordinating the agreement with other capsules. Instead, each capsule is empowered by higher modeling capabilities in the form of a subordinate routing network that learns to predict the routing coefficients directly (Figure 16). In self-routing, computing the routing coefficients c_{ij} and predictions $\widehat{\mathbf{u}}_{j|i}$ involves two learnable weight matrices $\mathbf{W}^{route}$ and $\mathbf{W}^{pose}$, respectively. For each layer of the routing network, each pose vector $\mathbf{u}_i$ is multiplied by a trainable weight matrix $\mathbf{W}^{route}$ to output the routing coefficients directly. After softmax normalization, the calculated routing coefficients c_{ij} are then multiplied by the capsule's activation scalar a_i to generate weighted votes. The activation a_j of an upper-layer capsule is the summation of the weighted votes of lower-level capsules over spatial dimensions $H \times W$, or $K \times K$ when using convolutions. The authors observed competitive performance on standard benchmarks, including

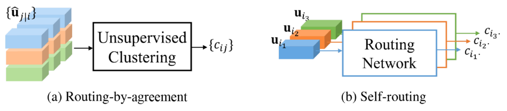

Fig. 16. Self-routing capsule networks. (a) typical routing-by-agreement; (b) the self-routing mechanism proposed by Reference [44]. In self-routing, subordinate routing networks with parameters $\mathbf{W}_i^{route}$ are fed capsule pose vectors $\mathbf{u}_i$ and learn to output the routing coefficients c_{ij} directly. Figure from Reference [44].

robustness to adversarial attacks. Although simple, this method somewhat limits the capsule network's ability to dynamically adjust the routing weights based on the input, since these are now fully determined by the learned parameters of the routing subnetworks. This approach is also reminiscent of the Synthesizer in Transformer literature [119].

3.8 ProtoCaps: Non-iterative Capsule Routing

Everett et al. [30] present a non-iterative approach for capsule routing that primarily aims to improve capsule efficiency via prototype routing. Each ProtoCaps routing layer operates by projecting the pose matrices of the lower-level Capsules into a shared subspace denoted as S. The projection is executed by a multi-layer perceptron following: $\text{pose}_i^{\text{proj}} = \text{MLP}_{\text{proj}}(\text{pose}_i)$, where pose_i corresponds to the ith pose matrix of the lower-level capsules. Within the shared subspace S, one has n learnable prototype vectors $Q = (q_1, q_2, \dots, q_n)$, where each $q_i \in \mathbb{R}^d$. Compared to other non-iterative capsule approaches, such as Reference [45], ProtoCaps uses fewer FLOPS while achieving better performance.

3.9 Straight-through Attentive Routing

One of the main drawbacks of capsule networks, especially the dynamic routing approach, is computational complexity that stems from the complex mechanisms of the voting and routing processes. Even if the capsule network architecture has a fixed number of parameters, the number of routing iterations can increase the training and inference time significantly. With that in mind, Ahmed and Torresani [4] proposed a non-recursive attention-based routing mechanism, inspired by the non-recurrent self-attention approach found in Transformers [123]. The proposed STAR-Caps layer architecture can be seen in Figure 17 and utilizes a straight-through attentive routing mechanism, formulating each capsule as a matrix rather than a vector as proposed in EM routing framework [50].

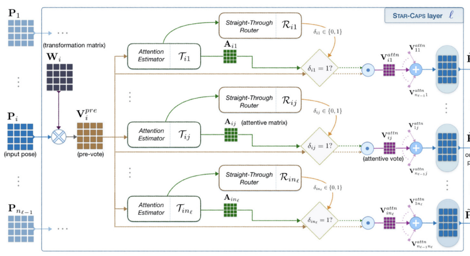
STAR-CAPS layer ℓ
$\mathbf{P}_1$
$\mathbf{P}_i$
(input pose)
$\mathbf{P}_{n_{\ell-1}}$
(transformation matrix)
$\mathbf{W}_i$
$\mathbf{V}_i^{pre}$
(pre-vote)
Attention Estimator $\mathcal{T}_{i1}$
Attention Estimator $\mathcal{T}_{ij}$
Attention Estimator $\mathcal{T}_{in_\ell}$
Straight-Through Router $\mathcal{R}_{i1}$
Straight-Through Router $\mathcal{R}_{ij}$
Straight-Through Router $\mathcal{R}_{in_\ell}$
$\mathbf{A}_{i1}$
$\mathbf{A}_{ij}$ (attentive matrix)
$\mathbf{A}_{in_\ell}$
$\delta_{i1} \in \{0,1\}$
$\delta_{ij} \in \{0,1\}$
$\delta_{in_\ell} \in \{0,1\}$
$\delta_{i1} = 1?$
$\delta_{ij} = 1?$
$\delta_{in_\ell} = 1?$
$\mathbf{V}_{i1}^{attn}$
$\mathbf{V}_{ij}^{attn}$
(attentive vote)
$\mathbf{V}_{in_\ell}^{attn}$
$\hat{\mathbf{P}}_1$
$\hat{\mathbf{P}}_j$
output pose
$\hat{\mathbf{P}}_{n_\ell}$

Fig. 17. STAR-CAPS layer architecture [4]. Given the pose features from the lower-level capsules, the pose is transformed through shared trainable weight matrices (pre-vote). The routing between the lower-level and higher-level capsules takes place through two components: the Attention Estimator and the Straight-through Router. This router estimates a binary signal that decides whether to connect or disconnect the current route between the lower-level capsule and the higher-level capsule. Figure from Reference [4].

STAR-Caps [4] employs the following two-mechanism process for routing capsules: (i) the attention estimator; (ii) the straight-through router. The role of the attention estimator $\mathcal{T}_{ij}$ is to estimate the attention matrix $\mathbf{A}_{ij} \in \mathbb{R}^{p \times p}$ between lower and higher-level capsules. The straight-through router $\mathcal{R}_{ij}$ decides which capsules to connect/disconnect. As shown in Figure 17, given the attentive matrix $\mathbf{A}_{ij}$, the straight-through router $\mathcal{R}_{ij}$ acts as a gate that estimates a binary decision value $\delta_{ij} \in 0, 1$, indicating whether to disconnect ($\delta_{ij} = 0$) or connect ($\delta_{ij} = 1$) the route between capsules i and j. This process is akin to hard attention, where each $\mathcal{R}_{ij}$ sends its hard attention signal to the higher-level capsules. To make this process differentiable, the authors employ a straight-through estimator [11, 56, 86]. As usual, the ClassCaps layer outputs the final predictions, where each capsule vector represents a single class. Like Reference [106], the authors encode activations implicitly in the capsule, and the final probability is given by a global average pooling operation on the poses followed by a logistic transformation. They then calculate the spread loss as in EM routing [50] for training.

3.10 Inverted Dot-product Attention Routing

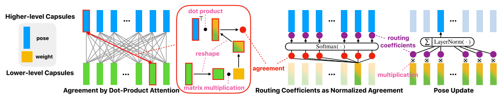

Fig. 18. The Inverted Dot-product Attention routing mechanism, showing the two-step process, i.e., the agreement between lower-level capsules and higher-level capsules, and the update of the pose of the higher-level capsules. Figure from Reference [120].

In Reference [120], the authors proposed a routing algorithm for capsule networks inspired by the attention mechanism commonly found in transformers [123] (see Figure 18). They design their routing algorithm via an inverted dot-product attention mechanism that includes layer normalization when updating the poses of higher-level capsules. This approach is most similar to Dynamic routing [106], since the capsule voting scheme and agreement are computed in much the same way. Unlike dynamic routing, however, inverted dot-product attention routing introduces the concept of *concurrent routing*, whereby multiple layers of capsules are iteratively routed at the same time rather than routing capsules in each each layer sequentially. In their experiments, the authors showed that their method compares favorably to Dynamic and EM routing on standard benchmark tasks.

3.11 Inference for Generative Capsule Models

In most previous works on capsules [50, 106] an inference algorithm is typically presented without specification of a corresponding generative model for the data (with the exception of Reference [64] who use an autoencoder for generation). In Reference [91] the authors argue for a generative approach to model the relationships of objects and their parts in capsule networks. They state that it is more natural to describe the generative process by which an object gives rise to its parts, rather than the other way around, as is typical. To that end, they present a principled generative capsule model that leads naturally to a variational algorithm for inferring the transformation of each object and the assignments of observed parts to the objects. Their work is built on the premise that the input to a capsule should be a set of parts. For example, say we have an object k with instantiation parameters y_k, and each object has parts p_n, where $n = 1, \dots, N_k$. These parts are then matched against observed parts x_m. Under a probabilistic framework, the authors get posterior distributions for both the y_k's and the match variables z_{mnk} that match x_m to part n of object k.

Such a setup leads directly to a principled routing-by-agreement algorithm via variational inference, which can be derived similarly to the classical Gaussian Mixture Model [12]. They therefore avoid having to devise a custom inference algorithm with an ad hoc objective function, as previously proposed in EM-routing capsules [50]. The authors demonstrated that their approach outperforms the CCAE part of the SCAE method [64] in the constellations data generated from multiple geometric objects, e.g., triangles, squares, that they used, as well as data from a parts-based model of faces. They also demonstrated that **random sample consensus (RANSAC)** [32]—where a minimal number of parts are used to instantiate an object—is often an effective alternative to variational inference routing-by-agreement, especially when the basis in RANSAC is highly informative about the object.

4 Understanding Attention & Capsules

As previously alluded to in Section 1, there are notable conceptual similarities between capsule routing and the self-attention mechanism popularized by Transformers [123]. In this section, we first provide a detailed breakdown of the relationship between them and show how we can think of each method from a unified perspective. There is also significant conceptual overlap between capsule networks and other object-centric representation learning techniques involving *slots* [40, 82].

Therefore, we discuss previous research on using the attention mechanism for object-centric learning and highlight conceptual similarities to the capsule formulation along the way.

4.1 A Unifying Perspective: Agreement Machines

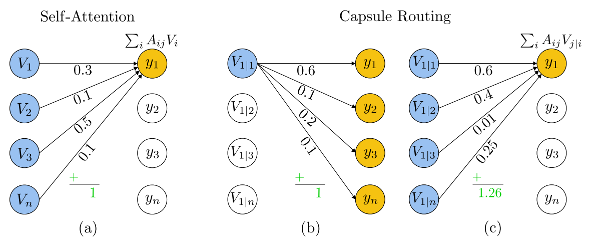

Fig. 19. Comparing self-attention and capsule routing. (a) In self-attention, an output token (e.g., y_1) is a weighted average of the input values, where the weights sum to 1 over $i = 1, \dots, n$. (b) In capsules, the weights of each input sum to 1 over the *outputs*. (c) A single output capsule (e.g., y_1) is a weighted average of the input votes with weights that do *not* necessarily sum to 1 over i.

In this section, we introduce both self-attention and capsule routing as *agreement machines*, consisting of dynamic weighted averaging layers that operate on D dimensional vector-valued units $\mathbf{x}_i \in \mathbb{R}^D$. When convenient, we adopt Einstein index notation to highlight conceptual similarities between the two methods more clearly (see the respective code in Figure 20). These vector-valued units $\mathbf{x}_i$ are known as token embedding vectors in transformers and capsules in capsule networks. To illustrate this, let $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N) \in \mathbb{R}^{N \times D}$ denote a matrix of input token embedding vectors or capsules $\mathbf{x}_i = (x_1, \dots, x_D)$. Consider the computation of a single output token $\boldsymbol{y}_j \in \mathbb{R}^P$ in simple self-attention, given a sequence of input tokens $\{\mathbf{x}_i\}_{i=1}^N$, where $A \in \mathbb{R}^{N \times M}$ is an attention weight matrix between all input/output pairs:

$$\boldsymbol{y}_j = \sum_{i=1}^{N} A_{ij} \mathbf{x}_i, \tag{8}$$

with $N = M$ and $0 \leq A_{ij} \leq 1$. Similarly, to compute a single output capsule $\boldsymbol{y}_j \in \mathbb{R}^P$, given input capsules $\{\mathbf{x}_i\}_{i=1}^N$, we have the same expression: $\boldsymbol{y}_j = \sum_{i=1}^N A_{ij}\mathbf{x}_i$. As we describe in greater detail next, the main differences lie in the precise introduction of parameters and how the attention weights are normalized. Moreover, we often have fewer outputs than input capsules ($M < N$), making A no longer a square matrix like in self-attention.

Self-attention. In scaled dot product self-attention [123], the N input vectors $\mathbf{X} \in \mathbb{R}^{N \times D}$ are transformed into respective query, key, and value matrices as follows:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \qquad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \qquad \mathbf{V} = \mathbf{X}\mathbf{W}^V, \tag{9}$$

where $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$ are $D \times P$ dimensional parameter matrices, and $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are therefore $N \times P$ dimensional. Let us now consider the calculation of the attention weights for a single input token $\mathbf{x}_i$. Given $\mathbf{x}_i$'s query row vector $\mathbf{Q}_{i,:} \in \mathbb{R}^{1 \times P}$, its attention weights $A_{i,:} \in \mathbb{R}^{1 \times N}$ are given by the normalized dot product with each of the key vectors:

$$A_{i,:} = \operatorname{softmax}\Bigg(\underbrace{\frac{\mathbf{Q}_{i,:}\mathbf{K}^\top}{\sqrt{P}}}_{\text{agreement}} \Bigg). \tag{10}$$

The dot product *agreement* between token $\mathbf{x}_i$'s query $\mathbf{Q}_{i,:}$ and all the keys $\mathbf{K}$ dictates how much "value" $\mathbf{V} = (V_1, \dots, V_N) \in \mathbb{R}^{N \times P}$ from each other token should be represented in token $\mathbf{x}_i$'s revised representation. That is, a single output token $\mathbf{Y}_{i,:} \in \mathbb{R}^{1 \times P}$ is simply a weighted average of the input

```python
def capsule_routing(X: torch.Tensor, iters: int = 3):
    N, D = X.shape  # N-by-D inputs
    M, P = 3, 4     # M-by-P outputs

    W = nn.Parameter(torch.randn(N, M, D, P))

    V = torch.einsum('id,ijdp->ijp', X, W)
    Y = (1./M) * torch.sum(V, dim=0)  # init capsules

    agreement = torch.zeros(N, M)

    for i in range(iters):

        agreement += torch.einsum('jp,ijp->ij', Y, V)
        A = agreement.softmax(dim=1)
        Y = torch.einsum('ij,ijp->jp', A, V)
    return Y
```

```python
def self_attention(X: torch.Tensor):
    N, D = X.shape  # N-by-D inputs
    M, P = N, 4     # M-by-P outputs, M = N

    Wq = nn.Parameter(torch.randn(D, P))
    Wk = nn.Parameter(torch.randn(D, P))
    Wv = nn.Parameter(torch.randn(D, P))

    Q = torch.einsum('id,dp->ip', X, Wq)
    K = torch.einsum('id,dp->ip', X, Wk)
    V = torch.einsum('id,dp->ip', X, Wv)

    agreement = torch.einsum(
        'ip,jp->ij', Q, K) / np.sqrt(P)
    A = agreement.softmax(dim=1)
    Y = torch.einsum('ij,jp->ip', A, V)
    return Y
```

Fig. 20. Comparing capsule routing and self-attention in PyTorch code [96]. Both operations take $i = 1, \ldots, N$ input vectors $\mathbf{x}_i \in \mathbb{R}^D$ and produce $j = 1, \ldots, M$ output vectors $\boldsymbol{y}_j \in \mathbb{R}^P$, where $M = N$ in self-attention. The outputs are attention-weighted averages of the inputs, with attention weights $A \in \mathbb{R}^{N \times M}$ computed using the *agreement* (similarity) between neural activity vectors, which is a form of *coincidence filtering*. Notice how the final steps are similar, and that in capsule routing the output capsules $\mathbf{Y}$ act as the queries $\mathbf{Q}$ in self-attention, and the capsule votes $\mathbf{V}_{j|i}$ play the role of both the keys $\mathbf{K}$ and the values $\mathbf{V}$. A notable difference is that self-attention can handle variable-length (N) sequences naturally, whereas capsule routing has a more rigid structure defined by the weights $\mathbf{W} \in \mathbb{R}^{N \times M \times D \times P}$. This can be relaxed via weight sharing across capsules, e.g., using $\mathbf{W} \in \mathbb{R}^{M \times D \times P}$. Self-attention is more parallelizable and better suited for current hardware accelerators compared to the sequential and iterative nature of capsule routing.

token's values:

$$\mathbf{Y}_{ip} = (A\mathbf{V})_{ip} = \sum_{j=1}^{N} A_{ij} V_{jp}, \tag{11}$$

for $i = 1, \ldots, N$ and $p = 1, \ldots, P$. Each output token $\boldsymbol{y}_i := \mathbf{Y}_{i,:} \in \mathbb{R}^{1 \times P}$ constitutes the revised representation for $\mathbf{x}_i$.

Capsule Routing. Comparatively, a single output capsule is given by first calculating the capsule votes, which are each input capsule's prediction of what the output capsule should be. Using tensor contraction notation as above, we start by multiplying input capsules $\mathbf{X} \in \mathbb{R}^{N \times D}$ by a 3D learned parameter matrix (3-tensor) $\mathbf{W} \in \mathbb{R}^{N \times D \times P}$:

$$\mathbf{V}_{ip} = \sum_{d=1}^{D} W_{idp} X_{id}, \tag{12}$$

for $i = 1, \ldots, N$ and $p = 1, \ldots, P$. The resulting votes are $\mathbf{V} = (V_1, \ldots, V_N) \in \mathbb{R}^{N \times P}$. Note that, unlike in self-attention, here, we have N separate weight matrices, one for each input vector-valued unit $\{\mathbf{x}_i\}_{i=1}^N$, i.e., capsule. In the first routing iteration, the output of the jth capsule is an average of its votes: $\boldsymbol{y}_j = \frac{1}{M} \sum_{i=1}^N V_i$, since the attention weights are uniform over outputs. The *agreement* (also measured by the dot product) between the votes and the output is then used to iteratively revise both the attention weights and the output: $\boldsymbol{y}_j = \sum_{i=1}^N A_{ij} V_i$.

However, unlike in self-attention, to compute the attention weights for a single output capsule, we require the context of *all* the other output capsules in the same layer. Indeed, there is no direct equivalent equation to Equation (10) we can use here without breaking the softmax normalization. Moreover, since each output capsule receives $i = 1, \ldots, N$ votes, to compute $j = 1, \ldots, M$ output

```python
def slot_attention(X: torch.Tensor, iters: int = 3):
    N, D = X.shape  # N-by-D inputs
    M, P = 3, 4     # M-by-P outputs

    Wq = nn.Parameter(torch.randn(P, P))
    Wk = nn.Parameter(torch.randn(D, P))
    Wv = nn.Parameter(torch.randn(D, P))

    K = torch.einsum('id,dp->ip', X, Wk)
    V = torch.einsum('id,dp->ip', X, Wv)
    Y = torch.randn(M, P)  # init slots

    for i in range(iters):
        Q = torch.einsum('jp,pp->jp', Y, Wq) / np.sqrt(P)
        agreement = torch.einsum('jp,ip->ij', Q, K)
        A = agreement.softmax(dim=1)
        Y = torch.einsum('ij,ip->jp', A, V)
    return Y
```

```python
def cross_attention(
    X: torch.Tensor, Y: torch.Tensor
):
    N, D = X.shape  # N-by-D inputs for keys/values
    M, P = Y.shape  # M-by-P inputs for queries

    Wq = nn.Parameter(torch.randn(P, P))
    Wk = nn.Parameter(torch.randn(D, P))
    Wv = nn.Parameter(torch.randn(D, P))

    K = torch.einsum('id,dp->ip', X, Wk)
    V = torch.einsum('id,dp->ip', X, Wv)

    Q = torch.einsum('jd,pp->jp', Y, Wq) / np.sqrt(P)
    agreement = torch.einsum('jp,ip->ij', Q, K)
    A = agreement.softmax(dim=1)
    Y = torch.einsum('ij,ip->jp', A, V)
    return Y
```

Fig. 21. Comparing slot attention and cross-attention in PyTorch code [96]. As shown, slot attention can be understood as an iterative cross-attention procedure, where the second input sequence is randomly initialized rather than being given: $\mathbf{Y} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_p) \in \mathbb{R}^{M \times P}$. This $\mathbf{Y}$ sequence plays the role of the slots/capsules, which we infer given $\mathbf{X} \in \mathbb{R}^{N \times D}$ via an iterative attention procedure similar to capsule routing. In contrast, the initial slots/capsules $\mathbf{Y}$ in capsule routing are typically a function of $\mathbf{X}$, not randomly initialized.

capsules, we require the transformation weights to be a 4-tensor: $\mathbf{W} \in \mathbb{R}^{N \times M \times D \times P}$. Thus, there is a learned $D \times P$ matrix between every input/output capsule pair.

With that in mind, carefully consider the full procedure for computing a single output capsule in a capsule layer, in conjunction with the accompanying code implementation in Figure 20. The following steps can be repeated to constitute *routing* by iteratively refining the initial uniform attention weights in Equation (13) with new estimates from Equation (14):

$$\text{(1. Voting)} \qquad \mathbf{V}_{ijp} = \sum_{d=1}^{D} W_{ijdp} X_{id}, \qquad \text{(2. Average votes)} \qquad \boldsymbol{y}_{jp} = \frac{1}{M} \sum_{i=1}^{N} V_{ijp}, \tag{13}$$

$$\text{(3. Attention)} \qquad A_{ij} = \text{softmax}\left(\sum_{p=1}^{P} V_{ijp} \boldsymbol{y}_{jp} \right)_j, \qquad \text{(4. Output capsule)} \qquad \boldsymbol{y}_{jp} = \sum_{i=1}^{N} A_{ij} V_{ijp}, \tag{14}$$

for each $j = 1, \ldots, M$ output capsule. Notably, Equation (14) entails a softmax normalization over outputs, rather than inputs as in self-attention (Equation (10)). As depicted in Figure 19, the input capsules spread their value among output capsules, causing output capsules to compete with each other for input capsule's values—whereas in self-attention the competition is between the input tokens instead. In the language of transformers, we can think of the output capsules as the query, and the capsule votes act as both the keys and the values in self-attention. To calculate the attention weights in capsule routing, we compute the *agreement* between outputs and votes via the dot product, just like we would in self-attention.

4.2 Inductive Biases: Transformers vs. Capsules

Having established a formal relationship between self-attention and capsule routing, in this section, we compare and contrast the inductive biases inherent to both methodologies. Recall that an inductive bias of a learning algorithm is a modeling assumption that induces a preference for certain solutions. Inductive biases often consist of encoding useful prior assumptions about the target function mapping inputs to outputs, which can aid in generalization to unseen cases and reduce sample complexity. The exposition in Section 4.1 highlights the inductive biases induced by the capsule formulation are such that each part (input) capsule belongs to a single object (output capsule), and each object must compete with other objects for parts. This is also known as the "single parent" assumption, commonly found in mixture models and clustering algorithms. Moreover, as previously outlined in Section 2.3, the (per-capsule) vote transformation matrix $\mathbf{W}$ is biased towards encoding invariance to viewpoint transformations, and the capsule vectors are biased towards capturing equivariance of neural activities. However, the self-attention mechanism induces weaker inductive biases, since there are no equivariance or single-parent assumptions, such as in CNNs or capsule networks, for example. This relaxed inductive bias makes transformers with self-attention very flexible models, but it also means that more data is typically required to match the performance of models with more explicit inductive biases [26], since a portion of the modeling capacity has to be spent on learning to encode any useful biases. With that said, one inductive bias we can interpret from self-attention is that each output token is best explained by a single input token (due to the softmax normalization over inputs), which can be thought of as a "single child" assumption. Subsequently, input tokens compete with each other to be included in each output token's revised representation through pairwise interactions.

Last, it is important to note that both capsules and attention share some key inductive biases, such as: using vectors of neural activity to represent a collection/hierarchy of concepts and taking the *agreement* between these high dimensional vectors as a feature detection mechanism. The relationships between these concepts are then dynamically adjustable based on the input, and the concepts themselves are refined based on global context. Self-attention can be considered a more efficient way to perform *coincidence filtering* with fewer inductive biases than capsule routing.

4.3 Slot Representations & Attention

Many object-centric representation learning approaches using neural networks can be categorized as being *slot* based [40]. As shown in Figure 22, slots constitute a general representational format used for separating object-based representations. They provide a sort of working memory with fixed capacity that can be used to access independent object representations simultaneously. There are four main types of slots, as outlined by Reference [40], but to remain within the scope of this survey, we focus mainly on: (i) *category slots*, which are the most commonly used representational format in capsule networks; (ii) *instance slots*, which can be classed as "universal" capsules that can dynamically bind to multiple objects rather than a specific pre-specified category.

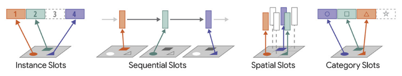

Fig. 22. Depicting the four different types of slot-based representations formats (the figure is from Reference [40]). Final layer capsules are typically instantiated as *category slots*, since they *bind* to objects in the input based on some categorical criteria like class identity. However, convolutional capsules [50] can be thought of as *spatial slots*, since they only receive votes from parts within their receptive field.

Locatello et al. [82] showed how attention can be used to extract object-centric representations that enable generalization to unseen compositions. Indeed, the proposed method they call *Slot Attention* is reminiscent of recent developments in both capsule networks and self-attention. As shown in Figure 23, the authors introduce the slot attention module, a differentiable interface between the outputs of a CNN and a set of variables they call *slots*. They employ an iterative attention mechanism, much like capsule routing, wherein the slots play the role of the capsules (see Figure 21). However, unlike capsules, the slots produced by slot attention do not specialize to one particular type of class or object, instead they can store/bind to any object in the input, making them more flexible. Because of this, slots have been referred to as "universal" capsules [49], as they can contain enough knowledge to model more than one type of object/part. Nonetheless, the *slots* still compete with each other at each iteration for explaining parts of the input via a softmax-based attention mechanism, just like in capsule networks and transformers. In fact, the slot attention iterations in their method can be thought of as equivalent to unrolled transformer layers that share parameters. This is reminiscent of capsule routing, whereby the routing iterations can be thought of as being equivalent to unrolled layers of attention that share parameters. In their experiments, the authors demonstrate that slot attention is competitive with previous approaches on unsupervised visual scene decomposition tasks like object discovery, while being more efficient.

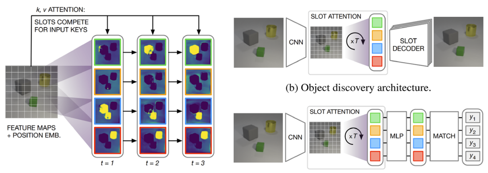

Fig. 23. Slot attention module (left), object discovery/set prediction (bottom right). Slots are "universal" capsules that can bind to any object in the input. Figure from Reference [82].

More recently, Li et al. [77] proposed SCOUTER, a slot-attention-based classifier for transparent, explainable, and accurate classification. The main difference between SCOUTER and vanilla slot attention is that the slots are now associated with single categories like in capsule networks. Indeed, the so-called *evidence* for a certain category in SCOUTER can be thought of as its *support* in capsule networks, i.e., using an attention mechanism to find support in the image that directly correlates with a certain output category. Li et al. [77] also employ an iterative attention mechanism to update the slots, where the number of iterations T=3 just like in capsule routing and vanilla slot attention. We can think of SCOUTER as a capsule network with restricted inductive biases and fewer parameters, thus the explainability insights from SCOUTER apply to capsule networks. Zhou et al. [144] bring the slot attention ideas to real-world data and achieve state-of-the-art performance on video panoptic segmentation tasks. The proposed **Video Panoptic Retriever (VPR)** retrieves and encodes all panoptic entities in a video, including both foreground instances and background semantics, with a unified object-centric representation called panoptic slots. The output panoptic slots can be directly converted into the class, mask, and object ID of panoptic objects in videos.

4.4 Transformers for Routing & Sets

Sun et al. [115] propose a visual parser that attempts to learn part-whole hierarchies through attention operations. The visual parser learns a two-level hierarchy iteratively refining the part and whole representations. At each iteration, the part encoder uses a set of learned part prototypes and performs an attention operation on the previous whole representations to obtain a set of N part representations. Then, the whole decoder refines the previous whole representation with the global information in this set of parts. Using this iterative encoder-decoder structure, the visual parser learns robust representations that can be applied to several tasks, including image classification, object detection, and instance segmentation. Carion et al. [17] present a transformer-based framework for object detection. After extracting a spatio-temporal grid of features from an image, the **DEtection TRansformer (DETR)** uses an encoder-decoder architecture to generate a set of N object predictions. This is accomplished by using N learned object queries in the transformer decoder whose output features are given to a feed-forward network to generate the class and bounding box dimensions. The network is trained end-to-end with a bipartite graph matching loss that attempts to minimize the difference between the predicted and ground-truth objects.

Self-attention has a quadratic $O(N^2)$ space and time complexity for N inputs. Wu et al. [130] alleviate this by proposing the centroid transformer, which clusters the N inputs into a set of M centroids that are then passed to the self-attention operation resulting in an $O(NM)$ complexity. This clustering operation can be viewed as a means to "route" the information from N inputs (parts) to M higher-level outputs (wholes) like in capsules. Similarly, Roy et al.[105] reduce the computational cost of the self-attention operation by only computing attention between a subset of keys and queries. Given an input sequence of length N, a clustering operation (k-means) is performed on the keys and queries to obtain k centroids. Then, for each of the N inputs, attention is computed on the set of keys that belongs to the same centroid as its given query. This proposed "Routing Transformer" reduces the computational cost from $O(N^2)$ to $O(N^{1.5})$ and outputs a sequence length of N.

Contrary to standard self-attention, the set transformer [73] performs the attention operation on a fixed set of m learned query vectors (also known as inducing points). This allows for reduced computational cost when the number of input vectors (n) becomes large (i.e., operation becomes $O(mn)$ as opposed to $O(n^2)$ in self-attention). Furthermore, set transformers have been utilized for the capsule routing procedure in the Stacked Capsule Auto-encoder [64]: The part capsules are passed through multiple set transformer layers to obtain a set of object capsules. Although the inducing points are learned in the set transformer, they are static and do not change based on the given input. Zare et al.[135] remedy this by introducing a "PICASO" block to update the learned inducing points based on information from a given input. By passing the inducing points through multi-head attention blocks, PICASO leads to improved representations for downstream tasks including classification, clustering, and anomaly detection.

The learning of object-centric representations through **Neural Expectation Maximization (N-EM)** was studied by References [39, 122]. N-EM is a probabilistic model that attempts to group pixels within an image into K entities whose properties are described by a vector θ_k. A differentiable EM algorithm is used to find these groupings by computing the Maximum Likelihood Estimate for each θ_k. Van et al.[122] extended this work by proposing a **Relational N-EM (R-NEM)** approach to learn interactions between different entities (parts/objects) over time. By replacing the M-step of the EM algorithm with a recurrent neural network, R-NEM is able to model the temporal dynamics of a given scene. Although these methods were primarily evaluated on primitive objects and shapes (e.g., triangles, squares, and circles), they have similar goals to capsule networks and constitute a promising probabilistic alternative.

5 Applications of Capsule Networks

Capsules for Video & Motion. Although the majority of foundational capsule approaches tend to be applied to image data, there have been several works that focus on the video domain. Generalizing 2-dimensional capsule networks to the 3-dimensional video domain is non-trivial. First, with the addition of a temporal dimension, how can capsule networks successfully capture the motion information from multiple frames, or timesteps, in a video sequence? Second, how can computationally costly routing operations scale to deal with video inputs? There is no current capsule work that completely answers these questions, but there have been several works that apply capsule networks to various video and motion problems.

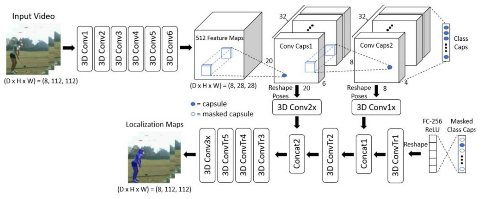

Fig. 24. The VideoCapsuleNet architecture proposed by Reference [27], which includes 3D convolutions and capsule-level pooling for processing video inputs efficiently.

Traditional 2D convolutional capsule routing was extended to 3D convolutional routing in Reference [27] (see Figure 24). In 3D convolutional routing, capsules that are both spatially and temporally nearby are routed together to obtain the higher layers' capsule outputs. Since the number of capsules being routed increases drastically as the size of the receptive field increases, conventional iterative routing operations are unsuited for 3D capsule networks. To this end, a *capsule-pooling* procedure was proposed, which averages each capsule type's poses and activations within the receptive field. Capsule-pooling ensures the number of capsules routed is only proportional to the number of capsule types within each layer, rather than the size of the receptive field. Duarte et al.[27] present a video capsule network, VideoCapsuleNet, which performs end-to-end action detection. The network consists of 3D convolutional layers to transform the input RGB video sequence (eight frames) into the initial video capsules. Then, this is followed by a 3D convolutional capsule layer with capsule pooling, followed by a fully connected capsule layer to produce class capsules. VideoCapsuleNet is not only able to classify the action being performed within the video but also spatiotemporally localizes the video by the use of a convolutional decoder. This idea is extended in Reference [28], where the authors propose CapsuleVOS, a network that can perform video object segmentation. Given a video clip and the segmentation of the object of interest in the first frame, CapsuleVOS propagates the segmentation through all frames of the video. The network consists of two branches that generate capsules for the video clip (video capsules) and capsules for the first frame and segmentation (frame capsules). Then, an attention routing algorithm is proposed to condition the video capsules based on their agreement with the frame capsules. This routing procedure first performs EM-routing on the frame capsules to obtain some higher-level capsule representation for the object in the first frame. Then, the routing coefficients are obtained for the video capsules by measuring their similarity to the higher-level frame capsules. The resultant video capsule representations are used by a convolutional decoder to segment objects.

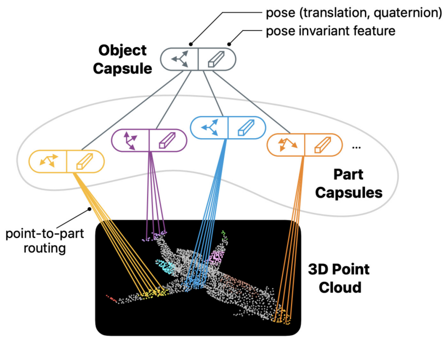

Fig. 25. Geometric capsule model for processing point clouds. The part capsules activate the object capsule to describe the entire object and its pose [113].

Previous video capsule networks implicitly learn temporal and motion information through 3D convolutions and routing. Recently, a capsule autoencoder architecture has been proposed to explicitly learn robust motion representations [133]. This work takes concepts from the stacked capsule autoencoders (see Section 3.6), but the **Motion Capsule Autoencoder (MCAE)** replaces the part and object capsules with "snippet" and "segment" capsules. Here, a snippet capsule contains a semantic-agnostic representation for a short time frame, and a segment capsule contains a semantic-aware representation for a longer time frame. Segment capsules are obtained by aggregating the snippet capsules and reconstructing their parameters. Although this work shows impressive results in unsupervised motion representation learning, the MCAE operates on individual points (2-dimensional coordinates) and not directly on video pixels. Nonetheless, MCAE presents an elegant capsule-based approach to directly model motion in input sequences, and extending such an approach to RGB videos is an interesting avenue for future work. A capsule-based approach for regression tracking has been proposed in Reference [85]. Instead of obtaining a single set of video capsules from 3D convolutions, two sets of capsules S-Caps and T-Caps are obtained that learn the spatial and temporal relationships within the video. These two sets of capsules are then combined and passed through a series of convolutional routing layers to obtain **regression capsules (RegCaps)**, which classify the target and background. Finally, the pose matrices of the RegCaps are compressed using knowledge distillation to reduce computational costs and obtain more discriminative representations. Video capsule networks have also been applied to the multimodal domain of actor and action video segmentation from a sentence. Given a video and a natural language description, Reference [88] proposes an end-to-end capsule network that segments the object/actor described by the description. The network first extracts a spatial grid of video capsules that represent the various entities or objects within the video. Then, a visual-textual routing algorithm combines both capsule modalities at each location on the spatial grid. These capsules are then sent through a convolutional decoder network to obtain the output segmentation mask for the actor described in the sentence.

5.1 Geometric & Graph-based Capsules

Capsule network models have been proposed for processing 3D point clouds that are equivariant to 3D rotations and translations, as well as invariant to permutations of the input points. One of the earliest works is by Xinyi et al. [132], who proposed CapsGNN, which is a framework that combines **graph neural networks (GNN)** and capsules. GNN is used to extract node embeddings, which are fed onto primary capsules (Block 1). In the second stage, the node embeddings are scaled via an attention module that together with dynamic routing generates the graph capsules (Block 2). In the last stage, graph classification takes place via dynamic routing (Block 3). Srivastava et al. [113] proposed a geometric capsule design (Figure 25), in which every visual entity—part or whole object—is encoded using two components: a pose and a feature. The *pose* represents the transformation between a global frame and the entity's canonical frame in a geometrically interpretable manner as a six-degree-of-freedom coordinate transformation. Conversely, the *feature* is represented as a real-valued vector, which encodes all non-pose attributes and is invariant to the object's pose w.r.t. the viewer. The proposed *Geometric Capsule Autoencoder* is constructed to group 3D points into parts and these parts into objects in an unsupervised manner.

Li et al. [78] proposed a graph-based capsule routing mechanism that focuses on learning intra-relationships between capsules in each layer, which is relevant to text classification problems. Intra-relationships that are found in text data need to be taken into account, along with hierarchical relationships, to improve sentiment analysis. The proposed method treats capsules in each layer as nodes in a graph and applies a new routing mechanism that combines bottom-up routing and top-down attention to learn hierarchical- and intra-relationships. Finally, the relationship between different capsules is evaluated by the Wasserstein distance, and a normalization trick is used to approximate the adjacency matrix. Aiming at making CapsNets more interpretable—like Grad-CAM [109], which has been proposed for explaining CNN-based classifications—the method termed *GraCapsNets,* proposed by Gu & Volker [42], modifies CapsNets to have built-in explanations. The part-part relationship, i.e., the relationship between primary capsules, is modeled with graphs, followed by graph pooling operations that pool relevant object parts from the graphs to make a classification vote. The idea is that, since the graph pooling operation reveals which input features are pooled as relevant ones, one can create explanations to explain the classification decisions. In addition to interpretability, the proposed model improves object recognition via integrating graph modeling into CapsNets, hence treating capsules as node feature vectors and representing them as graphs so one can leverage graph structure information. As highlighted by Zhao et al. [142], processing 3D point clouds is a challenging

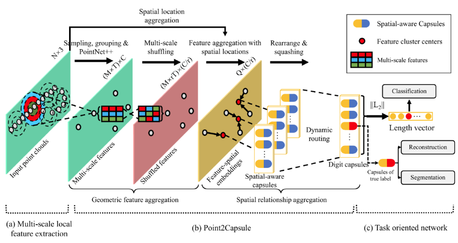

Fig. 26. Point2SpatialCapsule architecture. Given input point clouds, the three steps involved are: (i) extract multi-scale features from multi-scale areas; (ii) feature-spatial embeddings are created and merged via a spatial relationship aggregation step; (iii) task-specific network performs a downstream task. Figure from Reference [128].

problem due to two main reasons: (a) point clouds are irregular and unorganized, and (b) the group of transformations that one has to deal with is more complex, given that 3D data are often observed under arbitrary non-communicative $SO(3)$ rotations. Consequently, extracting and learning relevant embeddings requires 3D point networks to be equivariant to these transformations, while maintaining invariance properties to point permutations. The quaternion equivariant capsule module presented in Reference [142] extends the work presented in Reference [74] and is able to process point clouds while maintaining equivariance to $SO(3)$ rotations and preserving translation and permutation invariance. This work achieves $SO(3)$ by restricting the model to a sparse set of **local reference frames (LRFs)** that collectively determine the object orientation. In addition, the authors proposed a variation of dynamic routing, termed *Weiszfeld dynamic routing*, that uses inlier scores as activations and which together with LRFs, form part of the quaternion equivariant capsule module. This process allows for equivariant latent representations to be extracted that point to local orientations and activations, while also disentangling orientation from evidence of object existence. Another challenge when dealing with 3D point clouds concerns adequately capturing spatial relationships between local regions, e.g., the relative locations to other regions, to learn discriminative shape representation. Pooling-based feature aggregation methods struggle to achieve this satisfactorily.

A way to overcome is presented in Reference [128], wherein a new architecture called "Point2SpatialCapsule" is proposed that consists of two parts: (a) a module named *geometric feature aggregation* is designed to aggregate the local region features into learnable cluster centers, which manages to encode the spatial locations from the original 3D space, and (b) a module named *spatial relationship aggregation* is also proposed that further aggregates the clustered features and the spatial relationships among them in the feature space using a new capsule layer termed *spatial-aware capsules*. The complete architecture can be seen in Figure 26. A self-supervised 3D point cloud capsule architecture was proposed by Sun et al.[116] termed *canonical capsules*. The algorithm computes K-part capsule decompositions of 3D point-cloud objects through permutation-equivariant attention while self-supervising the process by training with pairs of randomly rotated objects (i.e., siamese training). This process removes the need to pre-align training datasets. The decomposition of the point cloud consists of assigning each point to one of the K parts via attention, which is then integrated into K key points. To ascertain equivariance, the two keypoint sets are set to differ only by the known relative transformation; regarding invariance, this takes place naturally by asking the descriptors of each keypoint of the two instances to match.

It is said that many parts of our brain are organized topographically, such as the ocular dominance maps. Keller & Welling [60] built upon this concept and proposed the topographic variational autoencoder: a novel method for efficiently training deep generative models with topographically organized latent variables (Figure 27). In their paper, they refer to capsules as "learning sets of approximately equivariant features or subspaces." The model they proposed tries to bridge two different classes of models, i.e., topographic generative models and equivariance neural networks. The proposed model is built upon the notion that inducing topographic organization can be leveraged to learn a basis of approximately equivariant capsules for observed transformation sequences. The resulting representation consists of a large set of capsules where the dimensions inside the capsule are topographically structured, but

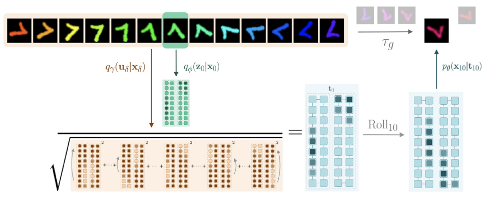
τ_g
$q_\gamma(\mathbf{u}_\delta|\mathbf{x}_\delta)$
$q_\phi(\mathbf{z}_0|\mathbf{x}_0)$
$p_\theta(\mathbf{x}_{10}|\mathbf{t}_{10})$
$\mathbf{t}_0$
Roll_{10}

Fig. 27. The Topographic Variational Autoencoder with shifting temporal coherence. The combined color/rotation transformation in the input space τ_g becomes encoded as a Roll within the capsule dimension. Figure from Reference [60].

between the capsules there is independence. The algorithm allows for sequences of input to be introduced to the model via encouraging topographic structure over time between sequentially permuted activations within a capsule, a property they call *shifting temporal coherence*. For further technical details, the reader may refer to Reference [60].

5.2 Generative Capsule Networks

There exist several studies that propose implementations of **Generative Adversarial Networks (GANs)** [37] using capsule networks. One of the earliest works was proposed by Jaiswal et al. [55], where a CapsNet was used as the discriminator and a deep CNN as the generator. However, the authors did not propose a new routing mechanism and instead, the model used was the one proposed in Reference [106]. However, in the work presented in Reference [29], a new CapsNet model was proposed, called **Subspace Capsule Network (SCN)**, which is built upon the idea of modeling the properties of an entity through a group of capsule subspaces instead of simply grouping neurons to create capsules. Using a learnable transformation, a capsule is then created by projecting an input feature vector from a lower layer onto the capsule subspace. This transformation finds the degree of alignment of the input with the properties modeled by the capsule subspace. A purportedly interpretable variant of capsule networks termed iCaps was proposed in Reference [58] using class-supervised disentanglement learning. This approach aims at disentangling the latent feature of x into two complementary subspaces, i.e., class-relevant and class-irrelevant subspaces, in a setting where the class label for images in the training set is provided. The iCaps architecture consists of six different parts: (i) a capsule network (classifier); (ii) an encoder that represents the class-irrelevant (residual) latent space; (iii) a generator that creates synthetic images; (iv) a discriminator that distinguishes whether an observation is from the dataset; (v) a classifier for image generation; (vi) a discriminator for contrastive regularization that maximizes the distance between the concepts represented.

A probabilistic generative version of capsule networks was proposed by Smith et al.in Reference [112], which aims to encode the assumptions under which capsules are built. This work is similar in spirit to the more recent work on inference in generative capsule models by Reference [91], as discussed previously in Section 4. Smith et al.introduced a variational bound that allowed them to explore the properties of their generative capsule model independently of the approximate inference scheme. In doing so, the authors gained insights into the failures of the capsule assumptions and inference amortisation. Concretely, the authors expressed the modeling assumptions of capsules as a probabilistic model with joint distribution over all latent and observed random variables. They then derived a routing algorithm directly from variational inference principles, leading to an amortized method similar to variational autoencoders [62]. The approach they introduced for routing phrases the problem as approximate inference in a graphical model, hence allowing for further future improvements by leveraging advancements in inference in graphical models. Their model performs comparably with previous works on capsules, showing that their probabilistic interpretation is a close approximation to capsule network assumptions. Their results also suggest that generative capsule formulations such as the one proposed may help enforce desirable equivariance properties, but this is

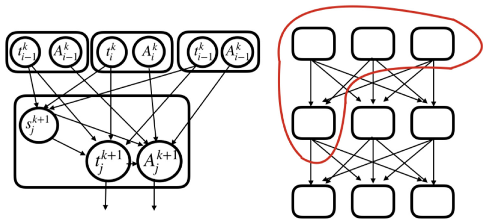

Fig. 28. A diagrammatic overview of the generative model for a capsule network. In the left sketch, one can see the detailed connectivity between the random variables that correspond to the red-circled region of the overall graph that can be seen on the right. Figure from Reference [112].

far from sufficient, as these models typically come without theoretical guarantees. They elaborate that, while promising, this type of capsule-based formulation is still somewhat underdetermined. Specifically, there are issues relating to the identifiability guarantees of obtaining *objects* from purely observed data, which suggests changes to the generative model may be necessary going forward.

5.3 Capsules for Natural Language Processing

Capsule networks have gained popularity in the field of natural language processing due to their ability to model part-whole relationships. Here, the sentence *parts* are individual words and the routing procedure learns the semantic relationships between words. Capsule-based architectures have been applied to a variety of natural language tasks, including text classification, relation extraction, search personalization, and recommender systems.

Text Classification. Text classification spans several tasks, including sentiment classification, question categorization, news categorization, and intent detection. Yang et al.[134] first explored the use of capsule networks in natural language processing for the problem of text classification (see Figure 29). Here, the dynamic routing algorithm [106] is augmented to deal with noisy capsules in three ways: the addition of orphan (i.e., background) categories, the use of leaky-softmax instead of the standard softmax operation to obtain the routing coefficients, and the multiplication of routing coefficients by the probability of existence of child capsules (denoted coefficient amendment). This work shows that a capsule network can achieve strong performance across six datasets when compared with standard neural network methods such as CNNs and LSTMs. Kim et al.[61] also propose a capsule network for text classification. To circumvent the need for max-pooling the text sequence, this work makes use of an ELU-gate unit that does not lose spatial information. Following the gate unit, the primary capsules are generated and passed through a "static routing" procedure that consists of a single forward pass of the dynamic routing algorithm. Another work [141] attempts to adapt capsule networks to be more successfully applied to NLP applications. First, a capsule compression operation is performed that merges similar capsules to reduce the number of primary capsules. Then, for routing, an adaptive optimizer is introduced, which allows for a variable number of routing iterations for a given sample. Last, for final classification, a partial routing procedure allows for a reduced number of output capsules to be produced, leading to a large reduction in computational cost. Another work [131] utilizes capsule networks for the task of zero-shot user intent detection. The proposed capsule network can detect intents unseen at training time by modifying the dynamic routing algorithm with self-attention and allowing for the generation of capsules for emerging intents (i.e., intents not used during training). Furthermore, analysis of the routing coefficients illustrates the capsule networks' ability to model the relationships between parts (words) and their corresponding whole (intents). Chen et al.[18] propose a **Transfer Capsule Network (TransCap)** for the problem of aspect-level sentiment classification (i.e., classifying the sentiment of a specific aspect occurring in a sentence). Given a sentence and the given aspect, TransCap generates a set of feature capsules from the input words and then performs "aspect routing," which gates the sentence (context) capsules using the aspect features to generate **semantic capsules (SemanCaps)**. These SemanCaps are then passed through the dynamic routing algorithm to obtain the final classification capsule

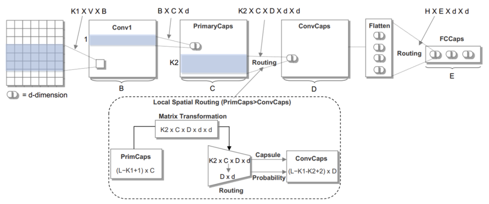

Fig. 29. Architecture of a capsule network–based system for text classification, as proposed by Yang et al. [134]. The *adjusted local spatial routing* between the primary and convolutional capsules is shown in detail.

layer. Capsules have also been applied to the task of slot filling and intent detection [136]. For a given sentence, this task involves a two-step classification problem: first assigning words to a specific slot class (e.g., *artist*, *playlist*, *movie_type*) and then classifying the intent of the overall query (e.g., *change_playlist*, *play_music*). Zhang et al.[136] propose a capsule network that generates a capsule for each word in the sentence and applies dynamic routing to create a set of slot capsules. The routing coefficients represent the assignment of each word in the sentence to a slot class. Then, another routing operation is used to obtain the final intent capsules and the final intent classification.

Relation Extraction. Relation extraction is a problem involving finding the relationships between different entities (i.e., words) within a sentence. Zhang et al.[137] illustrate capsule networks' ability to learn these relationships. The proposed capsule network makes use of a bi-directional LSTM to generate the initial capsule layer. Then, dynamic routing with coefficient amendment is used to generate a set of parent capsules whose activations are the probabilities of different relations. The entities by which these relations are represented are determined through another pretrained method [46]. Another work [139] proposes the architecture Att-CapNet, which also uses a bi-LSTM to generate the initial capsule layer. Here, however, the hidden states of the recurrent model are also used in the generation of attention coefficients for improved capsule routing.

Search Personalization and Recommendation Systems. Capsule networks have been applied to knowledge graph completion for the task of user search personalization [125]. For this problem, the goal is to generate a rating for a tuple (*query*, *user*, *document*) such that the output document is personalized for a given user's query. Vu et al.use a capsule-based approach that first encodes all elements of the tuple into vectors and uses convolution operations to generate a set of primary capsules. Then, the dynamic routing algorithm is used to generate a single two-dimensional capsule whose magnitude determines the score for the given tuple. A higher score denotes that there was agreement between the input capsules, leading to a higher ranking for the document when given the user's query. Capsules have also been used in recommendation systems. Given a set of reviews generated by a user and reviews for various items, recommendation systems attempt to suggest which items the user would like the most. Li et al.[75] present a capsule-based architecture, which takes embeddings for user and item reviews and generates a set of positive and negative capsules to represent the sentiment of various aspects of the reviews. Then, a routing by bi-agreement algorithm is proposed that attempts to find agreement not only between different capsules but also within dimensions of the same capsules (both inter-capsule and intra-capsule agreement). From the output of the routing algorithm, a recommendation score (i.e., rating) is generated for the given item.

Language and Vision. Several applications require the use of both textual and visual data. Capsule networks have shown promise in these multi-modal tasks. One such problem is that of visual question-answering, where a natural language question is given for an image, and the goal is to select a multiple-choice answer or generate a natural language answer. One of the first capsule-based approaches for this task is Reference [143]. Here, routing-by-agreement is used as an attention mechanism between the visual and textual features to improve learned representations. Urooj et al.[121] propose a capsule framework for visual question-answering grounding systems. This work shows that including capsules with EM-routing [50] in the generation of visual features leads to a drastic improvement in grounding accuracy; the capsule architecture uses relevant visual information in intermediate reasoning steps. Recently, Cao et al.[16] presented a routing algorithm that adjusts the capsule routing weights based on the parse tree generated from the given question. The proposed linguistically routed capsule network is shown to achieve strong visual questioning-answering performance, even on out-of-distribution data. Capsule networks have also been applied to multimodal machine translation [79]. The goal of this problem is to improve the translation of a natural language sentence with visual features. Lin et al.[79] propose a context-guided dynamic routing procedure to update the routing coefficients between visual capsules using cross-modal correlations. They show that this context-guided routing outperforms the standard attention and dynamic routing mechanisms when applied to their approach.

5.4 Capsules for Medical Image Analysis

Another field that Capsules have been applied fairly extensively to is that of medical imaging [3, 33, 81, 126, 140]. This is largely part due to a Capsule Network's ability to generalize to new variations of the learned classes in unseen data that were not captured in the training data. Additionally, this property is achieved after being trained on the small amount of data, which is typical for medical datasets due to the expertise required in labeling. SegCaps [68] replaces the convolutional blocks of a U-Net [104] with convolutional capsule blocks and modifies dynamic routing to be locally connected by only routing capsules in layer L to parent capsules in layer L+1 within a $k_h \times k_w$ kernel to semantically segment 2D slices of **computed tomography (CT)** scans showing irregular lesions and nodules. The modified Dynamic Routing algorithm is shown below. This method achieved state-of-the-art DICE Coefficient on the LUNA16 dataset with 98.479% accuracy while also reducing the parameters by approximately 95%.

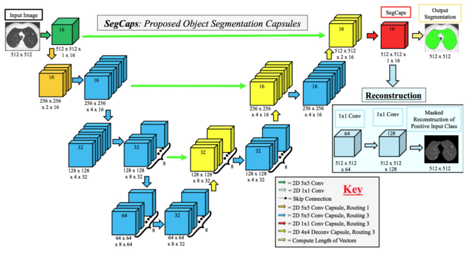
Input Image
SegCaps: Proposed Object Segmentation Capsules
SegCaps
Output Segmentation
Reconstruction
1x1 Conv
1x1 Conv
Masked Reconstruction of Positive Input Class
Key

Fig. 30. The architecture for SegCaps [68], which uses Capsule Networks in a U-Net–like architecture combining the information learned at different levels of downscaling via Capsule Network convolutional layers.

Building upon their previous work in References [68, 69] extends their method to five new datasets as well as introduces the condition that child capsules are only routed within a spatially located window. This change allows their network to scale to images of 512×512 size while remaining at around 1.5 million parameters. For reference, a standard dynamically routed capsule network would require approximately 2 quadrillion parameters to scale to images of this size. Again, their method outperformed the state-of-the-art on five datasets in terms of DICE coefficient and outperformed on four of the five in terms of Hausdorff Distance score. Additionally, it should be noted that their method has a significantly lower standard deviation between runs on different random seeds. Again, applied to the detection of nodules in CT scans, Fast CapsNet [90] modified dynamic routing CapsNet [106] to scale to 3D data. Their contributions are to add a constraint that allows only one Capsule per pixel location in the Primary Capsules. By doing this, they reduce the computational overhead of routing by agreement by 32×, resulting in a 3× overall increase of network training per epoch while retaining approximately the same accuracy as base CapsNets on 2D images that were slices of a 3D volume of the full CT scan. While this change may be small for 2D data, when they attempted to apply base CapsNets to 3D scans, the network was unable to be trained stably, while their modified CapsNet was able to achieve better accuracy on 3D than 2D data. Additionally, they replace the fully connected layers in the decoder section of the CapsNet with deconvolutional layers. Overall, they achieve 91.84% accuracy, which is greater than the 91.05% accuracy provided by the best CNN-based architectures and greatly improved upon the 73.65% provided by non–deep learning approaches. In the realm of Brain Tumor classification, CNNs have been shown to fail to utilize the spatial relationships between brain perturbations, which causes misdiagnosis of tumors. As a result of this, the authors in Reference [3] propose using a locally routed Capsule Network similar to Reference [90]. Additionally, rough boundaries of where the tumors are located are given to the network to ensure that it ignores irrelevant areas of the image. When trained with the entire image without rough tumor boundaries, the network achieves 78% accuracy on their dataset of MRI scans. This is then improved to 90.89% when additionally given the rough tumor boundaries, slightly improving upon the state-of-the-art CNNs, which are able to achieve 88.33% accuracy when given the rough tumor boundaries.

Covid-19 created a demand for fast and accurate diagnosis of patients. As a result, multiple deep learning techniques were tested upon CT scan and x-rays of the lungs. Motivated by Capsules Networks' ability to achieve strong results with low amounts of data, COVID-CAPS [2] was devised. Utilizing a standard dynamic routing [106] architecture to classify either positive or negative for COVID-19, COVID-CAPS is able to achieve an accuracy of 95.7%. When pre-trained using a large dataset of other x-ray images, COVID-CAPS is able to achieve 98.3% accuracy. Similarly, Reference [76] proposes MHA-CoroCapsule, a novel capsule architecture that builds upon the dynamic routing capsules [106] by replacing the dynamic routing element with a non-iterative multi-head attention routing process. By using a dataset of under 300 lung x-rays of COVID-19 patients, they were able to achieve nearly state-of-the-art results of 97.28% accuracy compared to 98.30% achieved by COVID-CAPS [2], however, COVID-CAPS is pre-trained on over 100K images of other lung diseases compared to the 1K total images used in MHA-CoroCapsules.

6 Discussion & Future Directions

In this article, we provided an extensive breakdown of the literature on object-centric learning using capsules and related attention-based methods. In doing so, we remark that capsule networks do not yet work as well as they might, which can in part be attributed to their lack of efficiency in enforcing the aforementioned modeling premises (see Section 2.3). The additional complexity induced by vector-valued neural activities, along with the high-dimensional coincidence filtering algorithm to detect capsule level features (capsule routing), leads to inefficient models that are often difficult to train. In the following sections, we provide an in-depth discussion of what we believe are the main conceptual considerations for future research in the field.

The Hardware Lottery. In their detailed analysis, Reference [8] explained that although their convolutional capsule model required around 4 times fewer **floating point operations (FLOPS)** with 16 times fewer parameters than their CNN, implementations in both TensorFlow [1] and PyTorch [96] ran significantly slower and ran out of memory with much smaller models. Although several more efficient versions of capsule routing have since then been proposed [4, 44, 101, 120], the underlying problem is not only caused by routing but by the capsule voting procedure as well. In their analysis, Reference [8] concludes that current frameworks have been highly optimized for a small subset of computations used by a popular family of models and that these frameworks have become poorly suited to research, since there is a huge discrepancy in performance between standard and non-standard compute workloads. As a result, non-standard workloads such as those induced by the routing and voting procedures in capsule networks are a lot slower than they could be. As pointed out by Reference [53], while capsule network's operations can be implemented reasonably well on CPUs, performance drops drastically on accelerators such as GPUs and TPUs, since they have been heavily optimized for standard workloads using the building blocks found in common architectures. This phenomenon begs the question of how much the tools researchers have readily available can predetermine the success of certain ideas based on how well they can be operationalized. Conceptual changes to capsule networks that can capture their inherent inductive biases while improving their operationalization using current hardware/frameworks would constitute a significant breakthrough. The development of more flexible tools that enable research using non-standard workloads is also of paramount importance going forward if we are to avoid future hardware lotteries.

The Binding Problem. The motivation behind capsule networks is part of an overarching narrative around the shortcomings of neural networks at human-level generalization. In most works, this problem is tackled from a computer vision perspective, where the goal is to be able to extract sensible object-centric representations from raw visual input to endow neural networks with reasoning and compositionality capabilities. The main assumption behind this is that *objects* play a fundamental role in systematic generalization. Greff et al. [40] argue that current limitations of neural networks are due to the *binding problem*, which prevents them from incorporating sensible object-centric representations. As quoted in Reference [31] "The Binding Problem concerns how items that are encoded by distinct brain circuits can be combined for perception, decision, and action." Biological neural networks in human brains are said to overcome the binding problem by enabling flexible and dynamic binding of information belonging to separate entities [83]. However, even the most advanced DL systems today struggle with compositionality [93, 100], so there is a need for neural network–based systems that attempt to tackle the binding problem more explicitly like capsule networks. With that in mind, Greff et al. [40] propose a functional division of the binding problem into three subproblems, as depicted in Figure 31. The *segregation* problem refers to the ability to form modular object representations from raw inputs. The *representation* problem relates to separately representing multiple object representations in a common format without interference between them. Last, the *composition* problem involves the capacity to dynamically relate and compose object representations to build structured object-centric models. Indeed, we believe that overcoming each of these open subproblems may give rise to more robust neural systems that generalize more like humans do. Capsule networks are one potential approach for tackling some aspects of the binding problem, but thus far they have mostly been applied in constrained supervised settings and lack the capabilities to integrate segregation, representation, and composition into a single system. Since most previous works on capsule networks have only considered the category slot format, there is also an opportunity to extend them to different types of slot representations as shown in Figure 22.

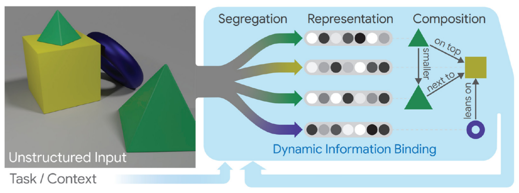

Fig. 31. The binding problem in neural networks. It can be understood as a collection of three subproblems, namely: *segregation*, *representation,* and *composition*. Figure from Reference [40].

Limitations & Open Challenges. Capsule networks have the potential to be a disruptive technology, but until the framework and hardware limitations are overcome, it will be quite difficult to truly test them at scale. Indeed, Mitterreiter et al. [89] corroborate this by providing some experimental insights as to why capsule neural networks do not scale in their current formulations, and we deem this to be fertile ground for further investigation. On that note, we believe that developing more flexible frameworks that are better suited to innovative research is a really important avenue for future work. As stressed by Reference [8], there are concerns regarding the lack of flexibility of languages and framework backends putting a brake on innovative research. From a technical perspective, we believe the main takeaways of capsule network models are the crucial ideas of *high-dimensional coincidence filtering* and *agreement*. In fact, many parallels can be drawn between capsules and the attention mechanism in Transformers [123], as both methods measure high-dimensional coincidences using the *agreement* between neural activity vectors to detect features. This is generally a good idea, since random vectors tend towards orthogonality as the number of dimensions grows large. Embedding vectors in Transformers can also be seen as capsules with a much greater number of dimensions. It would be interesting to see if a Transformer-like visual model with the correct part-whole inductive wiring can deal with viewpoint changes like capsule networks attempt to do, while being more efficient. There is also an opportunity to develop **Vision Transformer (ViT)**-style [26] capsule architectures, which do away with the expensive convolutional capsules formulation in favor of patch-based processing. This setup would likely entail using a fewer number of much higher dimensional capsules and a potential relaxation of the inductive biases incurred by the computationally costly capsule voting procedure. The extraction of more faithful primary capsules is also a promising research direction, since capsule networks remain hindered by the inability to learn effective low-level part descriptions (i.e., inverse rendering). Initial steps in this direction have recently been taken in Reference [107], where visual motion is used as a cue for part definition. One of the central ideas behind this is that a *part* or an *object* can be thought of as an entity that is perceptually consistent across time. There is an opportunity to extend these ideas to larger video datasets and use self-supervised learning to uncover 3D parts in much more complex scenes, as done in Reference [63]. Improved primary capsule representations could also lead to more effective iterative refinement routing algorithms, since the latter is predicated upon adequate low-level part descriptions to work as intended. Another aspect of interest for future work is carving out the role of approximately equivariant models like capsule networks in geometric deep learning [14, 19, 60]. Specifically, further study and comparison of approximately equivariant models to their counterparts in terms of equivariance metrics, sample/runtime complexity, generative modeling, and semi-supervised learning performance in 2D/3D complex tasks could be particularly important going forward. Moreover, given that capsule networks are only approximately equivariant, there is an opportunity to develop/uncover formal equivariance guarantees to expand the applicability of the model class.

Appendix

A Additional Applications of Capsule Networks

Capsule Networks have been applied to many different fields. The majority of these applications only use the original dynamically routed capsules [106] in domains where either data is difficult to obtain or label or domains where a network needs to be able to generalize to different conditions.

Adversarial Attacks. The properties of Capsule Networks are desirable for several reasons, including how they are resistant to affine transformation and single-pixel attacks that affect CNNs. In Reference [98], the authors showed that the reconstruction element of capsule networks improves the robustness to standard CNN adversarial attacks. In Reference [43], it is shown that dynamically routed capsule networks [106] are able to maintain a 17.3% accuracy on the CIFAR-10 dataset [65] against PGD attacks [87] compared to CNNs with 0% accuracy. Also, since the capsule voting mechanism is slow, it naturally takes longer to generate adversarial images. However, the authors then refocus their attacks to target the voting mechanism directly. With their proposed voting attack inside the PGD framework, the authors are able to decrease the performance of the capsule network to 4.83%. The authors then proceeded to train a capsule network that is specifically able to resist their proposed attack within the [98] framework, thereby creating a capsule network able to maintain 55.3% accuracy of its 94.7% accuracy when tested against adversarial PGD attacks.

Fault Diagnosis. In the field of bearing fault diagnosis, it is not possible to easily obtain images to visually inspect whether certain elements of machinery are failing during operation. In Reference [145], the authors convert 1D signals of vibration and electrical current to time-frequency graphs via Fourier transforms that can then be fed into for analysis. Additionally, these signals often contain a lot of noise and variations between every machine, depending on the state of the sensors and the speed and load that the machine is operating at. The only modification that the authors make to standard dynamically routed capsules [106] is the replacement of the initial convolutional layers with inception blocks from GoogLeNet [117] to create **Inception Capsule Networks (ICN)**. Over six different fault diagnosis tasks, ICNs are consistently either state-of-the-art or comparative to non-capsule methods—this is in stark contrast to the other methods, which generally perform very well in one task but underperform in others. Overall, ICNs are able to achieve 97.15% accuracy on average across the six tasks, compared to 94.58% of the best-competing method. Additionally, ICNs achieve 82.05% when tested on data for six new tasks from machines operating at different loads and speeds. Additionally, the authors propose that the output of the class capsules can be used for regression, achieving 94.04% accuracy in determining the severity of the faults detected. Building upon the work in Reference [145], the authors of Reference [127] propose an extension that proposes the **Xception module Capsule Network (XCN)**. This method follows the previous methodology of converting 1D signals to time-frequency graphs but uses wavelet time-frequency analysis rather than Fourier transforms. XCNs are able to achieve 98.4% accuracy across the three tasks of inner ring faults, outer ring faults, and ball faults. This is a fairly large improvement compared to the 97.6% that the best ensemble method is able to achieve. Additionally, XCNs perform more consistently well with 99.2%, 99.7%, and 96.3%, compared to 96.4%, 100%, and 96.4% from the ensemble approach. Additionally, when compared to live data from a working machine, XCNs achieve 97.2%, 98.7%, and 94.5% accuracy, a significantly smaller drop-off than all the other approaches, which shows the Capsule Networks' superior generalization ability.

Hyperspectral Images. Another application of Capsule Networks is in hyperspectral Image classification. Hyperspectral images are traditionally badly classified by current deep learning approaches due to their lack of ability to exploit the spatial relationships in the spectral spacial domain, which is a key factor in dealing with extremely high dimensional data. Additionally, CNNs are known to require a large amount of data, which given the high dimensionality and complexity of hyperspectral imaging is not possible. In Reference [94] the authors propose using capsule networks named **Spectral-Spatial Capsule Networks (SSCN)** to classify these images. The authors achieve a large amount of success in this field with an unmodified dynamically routed Capsule Network. Over five different random seeds and trained on only 15% of the available data, the authors achieve state-of-the-art segmentation in every class of two different datasets with 99.45% and 99.95% accuracy. In the third dataset, SSCNs are able to achieve state-of-the-art in 56 out of 58 classes with an average accuracy of 98.25% on this most complex dataset. However, it should be noted that the SSCN approach was significantly slower in terms of time per training epoch than all but one of the other approaches on all three of the datasets. Additionally in Reference [24] the authors build upon the work in Reference [94], showing further how standard dynamically routed Capsule Networks are able to outperform CNNs, achieving 96.27% accuracy on a difficult hyperspectral image dataset compared to the state-of-the-art CNN, which achieves 95.63%.

Forgery Detection. Detection of artificially generated forgery is a task that previously would require specific models for each variation of an attack. However, in Reference [92] the authors propose a unified Capsule-powered framework named **CAPSULE-FORENSICS (CF)**, where one network is able to detect forgeries of different types in both images and video. CFs use a pipeline of pre-processing where images are normalized and a video is split into individual frames. These images are then fed through a VGG-19 [111] network to extract the latent features rather than the traditional convolutional layers before the Primary Capsules. The vector output from the two class capsules of either real or fake is then used in a traditional forgery detection framework to detect forgeries. By applying noise to transformation matrix weights during the routing process, CFs are able to achieve state-of-the-art or comparable results in six different datasets. Building upon the work in Reference [92], the authors of Reference [84] improve the forgery detection framework to include the ability to detect AI-generated audio forgeries; they achieve 98.93% and 97.95% accuracy on the PA and LA subsets of the ASVspoof2019 dataset, while the previous best techniques are only able to achieve 98.16% and 96.22% accuracy.

Acknowledgments

The authors would like to thank all reviewers, and especially Professor Chris Williams from the School of Informatics of the University of Edinburgh, who provided constructive feedback and ideas on how to improve this work.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. USENIX, 265–283.

[2] Parnian Afshar, Shahin Heidarian, Farnoosh Naderkhani, Anastasia Oikonomou, Konstantinos N. Plataniotis, and Arash Mohammadi. 2020. COVID-CAPS: A capsule network-based framework for identification of COVID-19 cases from X-ray images. *Pattern Recogn. Lett.* 138 (2020), 638–643. DOI: https://doi.org/10.1016/j.patrec.2020.09.010

[3] Parnian Afshar, Konstantinos N. Plataniotis, and Arash Mohammadi. 2019. Capsule networks for brain tumor classification based on MRI images and coarse tumor boundaries. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'19)*. IEEE, 1368–1372.

[4] Karim Ahmed and Lorenzo Torresani. 2019. STAR-caps: Capsule networks with straight-through attentive routing. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 9098–9107.

[5] Alex M. Andrew. 2004. Multiple View geometry in computer vision. *Kybernetes* 30 (2004), 1333–1341.

[6] Hagai Attias. 1999. Inferring parameters and structure of latent variable models by variational Bayes. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 21–30.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473* (2014).

[8] Paul Barham and Michael Isard. 2019. Machine learning systems are stuck in a rut. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 177–183.

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 8 (2013), 1798–1828.

[10] Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. 2021. Deep learning for AI. *Commun. ACM* 64, 7 (2021), 58–65.

[11] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or Propagating Gradients through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432* (2013).

[12] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

[13] Wieland Brendel and Matthias Bethge. 2019. Approximating CNNs with Bag-of-local-features Models Works Surprisingly Well on ImageNet. *arXiv preprint arXiv:1904.00760* (2019).

[14] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: Going beyond Euclidean data. *IEEE Sig. Process. Mag.* 34, 4 (2017), 18–42.

[15] Vicki Bruce and Glyn W. Humphreys. 1994. Recognizing objects and faces. *Visual Cogn.* 1, 2–3 (1994), 141–180.

[16] Qingxing Cao, Wentao Wan, Keze Wang, Xiaodan Liang, and Liang Lin. 2021. Linguistically routing capsule network for out-of-distribution visual question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1614–1623.

[17] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *Proceedings of the European Conference on Computer Vision*. Springer, 213–229.

[18] Zhuang Chen and Tieyun Qian. 2019. Transfer capsule network for aspect level sentiment classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 547–556.

[19] Taco Cohen and Max Welling. 2016. Group equivariant convolutional networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2990–2999.

[20] Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. 2018. Spherical CNNs. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=Hkbd5xZRb

[21] Taco S. Cohen and Max Welling. 2017. Steerable CNNs. In *Proceedings of the 5th International Conference on Learning Representations (ICLR'17)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=rJQKYt5ll

[22] Fabio De Sousa Ribeiro, Georgios Leontidis, and Stefanos Kollias. 2020. Introducing routing uncertainty in capsule networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, Vol. 33. 6490–6502.

[23] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Societ.: Series B (Methodol.)* 39, 1 (1977), 1–22.

[24] Fei Deng, Shengliang Pu, Xuehong Chen, Yusheng Shi, Ting Yuan, and Shengyan Pu. 2018. Hyperspectral image classification with capsule network using limited training samples. *Sensors* 18, 9 (2018), 3153.

[25] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. 2016. Exploiting cyclic symmetry in convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1889–1898.

[26] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakub Uszkoreit, and Neil Houlsby. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations*.

[27] Kevin Duarte, Yogesh S. Rawat, and Mubarak Shah. 2018. VideoCapsuleNet: A simplified network for action detection. *Advan. Neural Inf. Process. Syst.* 31 (2018).

[28] Kevin Duarte, Yogesh S. Rawat, and Mubarak Shah. 2019. CapsuleVOS: Semi-supervised video object segmentation using capsule routing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 8480–8489.

[29] Marzieh Edraki, Nazanin Rahnavard, and Mubarak Shah. 2020. Subspace capsule network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 10745–10753.

[30] Miles Everett, Mingjun Zhong, and Georgios Leontidis. 2023. ProtoCaps: A fast and non-iterative capsule network routing method. *Trans. Mach. Learn. Res.* (2023), 2835–8856.

[31] Jerome Feldman. 2013. The neural binding problem(s). *Cogn. Neurodyn.* 7, 1 (2013), 1–11.

[32] Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.

[33] Divya Jyothi Gaddipati, Alakh Desai, Jayanthi Sivaswamy, and Koenraad A. Vermeer. 2019. Glaucoma assessment from OCT images using capsule network. In *Proceedings of the 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'19)*. IEEE, 5581–5584.

[34] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. 2018. ImageNet-trained CNNs Are Biased towards Texture; Increasing Shape Bias Improves Accuracy and Robustness. *arXiv preprint arXiv:1811.12231* (2018).

[35] Robert Geirhos, Carlos R. Medina Temme, Jonas Rauber, Heiko H. Schütt, Matthias Bethge, and Felix A. Wichmann. 2018. Generalisation in Humans and Deep Neural Networks. *arXiv preprint arXiv:1808.08750* (2018).

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advan. Neural Inf. Process. Syst.* 27 (2014).

[38] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572* (2014).

[39] Klaus Greff, Sjoerd Van Steenkiste, and Jürgen Schmidhuber. 2017. Neural expectation maximization. *Advan. Neural Inf. Process. Syst.* 30 (2017).

[40] Klaus Greff, Sjoerd Van Steenkiste, and Jürgen Schmidhuber. 2020. On the Binding Problem in Artificial Neural Networks. *arXiv preprint arXiv:2012.05208* (2020).

[41] Ashley Daniel Gritzman. 2019. Avoiding implementation pitfalls of "matrix capsules with em routing" by Hinton et al. In *Proceedings of the International Workshop on Human Brain and Artificial Intelligence*. Springer, 224–234.

[42] Jindong Gu and Volker Tresp. 2020. Interpretable graph capsule networks for object recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'20)*.

[43] Jindong Gu, Baoyuan Wu, and Volker Tresp. 2021. Effective and Efficient Vote Attack on Capsule Networks. *arXiv preprint arXiv:2102.10055* (2021).

[44] Taeyoung Hahn, Myeongjang Pyeon, and Gunhee Kim. 2019. Self-routing capsule networks. In *Proceedings of the Conference on 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. Retrieved from https://proceedings.neurips.cc/paper/2019/file/e46bc064f8e92ac2c404b9871b2a4ef2-Paper.pdf

[45] Taeyoung Hahn, Myeongjang Pyeon, and Gunhee Kim. 2019. Self-routing capsule networks. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 7658–7667. Retrieved from http://papers.nips.cc/paper/8982-self-routing-capsule-networks.pdf

[46] Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. OpenKE: An open toolkit for knowledge embedding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[48] Geoffrey Hinton. 1979. Some demonstrations of the effects of structural descriptions in mental imagery. *Cogn. Sci.* 3, 3 (1979), 231–250.

[49] Geoffrey Hinton. 2021. How to Represent Part-whole Hierarchies in a Neural Network. *arXiv preprint arXiv:2102.12627* (2021).

[50] Geoffrey Hinton, Sara Sabour, and Nicholas Frosst. 2018. Matrix capsules with EM routing. In *Proceedings of the 6th International Conference on Learning Representations (ICLR'18).* 1–15.

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

[52] Geoffrey E. Hinton and Drew Van Camp. 1993. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the 6th Annual Conference on Computational Learning Theory.* 5–13.

[53] Sara Hooker. 2020. The Hardware Lottery. *arXiv preprint arXiv:2009.06489* (2020).

[54] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. 2015. Spatial transformer networks. *Advan. Neural Inf. Process. Syst.* 28 (2015), 2017–2025.

[55] Ayush Jaiswal, Wael AbdAlmageed, Yue Wu, and Premkumar Natarajan. 2018. CapsuleGAN: Generative adversarial capsule network. In *Proceedings of the European Conference on Computer Vision (ECCV'18) Workshops.*

[56] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical Reparameterization with Gumbel-Softmax. *arXiv preprint arXiv:1611.01144* (2016).

[57] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. 1999. An introduction to variational methods for graphical models. *Mach. Learn.* 37, 2 (1999), 183–233.

[58] Dahuin Jung, Jonghyun Lee, Jihun Yi, and Sungroh Yoon. 2020. iCaps: An interpretable classifier via disentangled capsule networks. In *Proceedings of the European Conference on Computer Vision.* Springer, 314–330.

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

[60] Thomas Keller and Max Welling. 2021. Topographic VAEs learn equivariant capsules. *Advan. Neural Inf. Process. Syst.* 34 (2021).

[61] Jaeyoung Kim, Sion Jang, Eunjeong Park, and Sungchul Choi. 2020. Text classification using capsules. *Neurocomputing* 376 (2020), 214–221.

[62] Diederik P. Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114* (2013).

[63] Thomas Kipf, Gamaleldin F. Elsayed, Aravindh Mahendran, Austin Stone, Sara Sabour, Georg Heigold, Rico Jonschkowski, Alexey Dosovitskiy, and Klaus Greff. 2021. Conditional Object-centric Learning from Video. *arXiv preprint arXiv:2111.12594* (2021).

[64] Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E. Hinton. 2019. Stacked capsule autoencoders. In *Proceedings of the Conference on Advances in Neural Information Processing Systems.* 15486–15496.

[65] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009). http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf

[66] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems.* 1097–1105.

[67] Tejas D. Kulkarni, William F. Whitney, Pushmeet Kohli, and Josh Tenenbaum. 2015. Deep convolutional inverse graphics network. In *Proceedings of the Conference on Advances in Neural Information Processing Systems.* 2539–2547.

[68] Rodney LaLonde and Ulas Bagci. 2018. Capsules for Object Segmentation. *arXiv:stat.ML/1804.04241* (2018).

[69] Rodney LaLonde, Ziyue Xu, Ismail Irmakci, Sanjay Jain, and Ulas Bagci. 2021. Capsules for biomedical image segmentation. *Med. Image Anal.* 68 (2021), 101889.

[70] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[71] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. 1989. Handwritten digit recognition with a back-propagation network. *Advan. Neural Inf. Process. Syst.* 2 (1989).

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

[73] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the International Conference on Machine Learning.* PMLR, 3744–3753.

[74] Jan Eric Lenssen, Matthias Fey, and Pascal Libuschewski. 2018. Group equivariant capsule networks. *Advan. Neural Inf. Process. Syst.* 31 (2018).

[75] Chenliang Li, Cong Quan, Li Peng, Yunwei Qi, Yuming Deng, and Libing Wu. 2019. A capsule network for recommendation and explaining what you like and dislike. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval.* 275–284.

[76] Fudong Li, Xingyu Lu, and Jianjun Yuan. 2021. MHA-CoroCapsule: Multi-head Attention Routing-based capsule network for COVID-19 chest X-ray image classification. *IEEE Trans. Med. Imag.* 41, 5 (2021), 1208–1218.

[77] Liangzhi Li, Bowen Wang, Manisha Verma, Yuta Nakashima, Ryo Kawasaki, and Hajime Nagahara. 2021. SCOUTER: Slot attention-based classifier for explainable image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 1046–1055.

[78] Yang Li, Wei Zhao, Erik Cambria, Suhang Wang, and Steffen Eger. 2021. Graph routing between capsules. *Neural Netw.* 143 (2021), 345–354.

[79] Huan Lin, Fandong Meng, Jinsong Su, Yongjing Yin, Zhengyuan Yang, Yubin Ge, Jie Zhou, and Jiebo Luo. 2020. Dynamic context-guided capsule network for multimodal machine translation. In *Proceedings of the 28th ACM International Conference on Multimedia.* 1320–1329.

[80] Yi Liu, De Cheng, Dingwen Zhang, Shoukun Xu, and Jungong Han. 2024. Capsule networks with residual pose routing. *IEEE Trans. Neural Netw. Learn. Syst.* (2024), 2162–237X.

[81] Yan Liu, Ying Fu, and Pu Chen. 2019. WBCaps: A capsule architecture-based classification model designed for white blood cells identification. In *Proceedings of the 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'19).* IEEE, 7027–7030.

[82] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. 2020. Object-centric learning with slot attention. *Advan. Neural Inf. Process. Syst.* 33 (2020), 11525–11538.

[83] Sindy Löwe, Phillip Lippe, Maja Rudolph, and Max Welling. 2022. Complex-valued Autoencoders for Object Discovery. *arXiv preprint arXiv:2204.02075* (2022).

[84] Anwei Luo, Enlei Li, Yongliang Liu, Xiangui Kang, and Z. Jane Wang. 2021. A capsule network based approach for detection of audio spoofing attacks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'21).* IEEE, 6359–6363.

[85] Ding Ma and Xiangqian Wu. 2021. CapsuleRRT: Relationships-aware regression tracking via capsules. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 10948–10957.

[86] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *arXiv preprint arXiv:1611.00712* (2016).

[87] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv preprint arXiv:1706.06083* (2017).

[88] Bruce McIntosh, Kevin Duarte, Yogesh S. Rawat, and Mubarak Shah. 2020. Visual-textual capsule routing for text-based video segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 9942–9951.

[89] Matthias Mitterreiter, Marcel Koch, Joachim Giesen, and Sören Laue. 2023. Why capsule neural networks do not scale: Challenging the dynamic parse-tree assumption. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37.

[90] Aryan Mobiny and Hien Van Nguyen. 2018. Fast CapsNet for lung cancer screening. In *Proceedings of the International Conference on Medical Image Computing and Computer-assisted Intervention.* Springer, 741–749.

[91] Alfredo Nazabal and Christopher K. I. Williams. 2021. Inference for Generative Capsule Models. *arXiv preprint arXiv:2103.06676* (2021).

[92] Huy H. Nguyen, Junichi Yamagishi, and Isao Echizen. 2019. Capsule-forensics: Using capsule networks to detect forged images and videos. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'19).* IEEE, 2307–2311.

[93] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. 2021. GLIDE: Towards Photorealistic Image Generation and Editing with Text-guided Diffusion Models. *arXiv preprint arXiv:2112.10741* (2021).

[94] Mercedes E. Paoletti, Juan Mario Haut, Ruben Fernandez-Beltran, Javier Plaza, Antonio Plaza, Jun Li, and Filiberto Pla. 2018. Capsule networks for hyperspectral image classification. *IEEE Trans. Geosci. Remote Sens.* 57, 4 (2018), 2145–2160.

[95] Harold Pashler. 2016. *Attention.* Psychology Press.

[96] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the Conference on Advances in Neural Information Processing Systems.* 8024–8035.

[97] Mensah Kwabena Patrick, Adebayo Felix Adekoya, Ayidzoe Abra Mighty, and Baagyire Y. Edward. 2022. Capsule networks–A survey. *J. King Saud Univ.-Comput. Inf. Sci.* 34, 1 (2022), 1295–1310.

[98] Yao Qin, Nicholas Frosst, Sara Sabour, Colin Raffel, Garrison Cottrell, and Geoffrey Hinton. 2020. Detecting and diagnosing adversarial images with class-conditional capsule reconstructions. In *Proceedings of the International Conference on Learning Representations.* Retrieved from https://openreview.net/forum?id=Skgy464Kvr

[99] Yan Qin, Yong Liang Guan, and Chau Yuen. 2023. Spatiotemporal capsule neural network for vehicle trajectory prediction. *IEEE Trans. Vehic. Technol.* 72 (2023), 9746–9756.

[100] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *Proceedings of the International Conference on Machine Learning*. PMLR, 8821–8831.

[101] Fabio De Sousa Ribeiro, Georgios Leontidis, and Stefanos Kollias. 2020. Capsule routing via variational Bayes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3749–3756.

[102] I. Rock. 1973. *Orientation and Form.* Academic Press. Retrieved from https://books.google.co.uk/books?id=hgQEAQAAIAAJ

[103] David Romero, Erik Bekkers, Jakub Tomczak, and Mark Hoogendoorn. 2020. Attentive group equivariant convolutional networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 8188–8199.

[104] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:cs.CV/1505.04597* (2015).

[105] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Trans. Assoc. Computat. Ling.* 9 (2021), 53–68.

[106] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. 2017. Dynamic routing between capsules. In *Proceedings of the Conference on Advances in Neural Information Processing Systems.* 3856–3866.

[107] Sara Sabour, Andrea Tagliasacchi, Soroosh Yazdani, Geoffrey Hinton, and David J. Fleet. 2021. Unsupervised part representation by flow capsules. In *Proceedings of the International Conference on Machine Learning*. PMLR, 9213–9223.

[108] Gudrun Schwarzer. 2000. Development of face processing: The effect of face inversion. *Child Devel.* 71, 2 (2000), 391–401.

[109] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision.* 618–626.

[110] Ruiyang Shi and Lingfeng Niu. 2020. A brief survey on capsule network. In *Proceedings of the IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'20).* IEEE, 682–686.

[111] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

[112] Lewis Smith, Lisa Schut, Yarin Gal, and Mark van der Wilk. 2020. Capsule Networks—A Probabilistic Perspective. *arXiv preprint arXiv:2004.03553* (2020).

[113] Nitish Srivastava, Hanlin Goh, and Ruslan Salakhutdinov. 2019. Geometric Capsule Autoencoders for 3D Point Clouds. *arXiv preprint arXiv:1912.03310* (2019).

[114] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end Memory Networks. *arXiv preprint arXiv:1503.08895* (2015).

[115] Shuyang Sun, Xiaoyu Yue, Song Bai, and Philip Torr. 2021. Visual Parser: Representing Part-whole Hierarchies with Transformers. *arXiv preprint arXiv:2107.05790* (2021).

[116] Weiwei Sun, Andrea Tagliasacchi, Boyang Deng, Sara Sabour, Soroosh Yazdani, Geoffrey E. Hinton, and Kwang Moo Yi. 2021. Canonical capsules: Self-supervised capsules in canonical pose. *Advan. Neural Inf. Process. Syst.* 34 (2021).

[117] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1–9.

[118] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing Properties of Neural Networks. *arXiv preprint arXiv:1312.6199* (2013).

[119] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2021. Synthesizer: Rethinking self-attention for transformer models. In *Proceedings of the International Conference on Machine Learning*. PMLR, 10183–10192.

[120] Yao-Hung Hubert Tsai, Nitish Srivastava, Hanlin Goh, and Ruslan Salakhutdinov. 2020. Capsules with inverted dot-product attention routing. In *Proceedings of the International Conference on Learning Representations.* Retrieved from https://openreview.net/forum?id=HJe6uANtwH

[121] Aisha Urooj, Hilde Kuehne, Kevin Duarte, Chuang Gan, Niels Lobo, and Mubarak Shah. 2021. Found a reason for me? Weakly-supervised grounded visual question answering using capsules. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8465–8474.

[122] Sjoerd Van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. 2018. Relational Neural Expectation Maximization: Unsupervised Discovery of Objects and Their Interactions. *arXiv preprint arXiv:1802.10353* (2018).

[123] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 5998–6008.

[124] T. Vijayakumar. 2019. Comparative study of capsule neural network in various applications. *J. Artif. Intell.* 1, 01 (2019), 19–27.

[125] Dai Quoc Nguyen, Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2019. A capsule network-based embedding model for knowledge graph completion and search personalization. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2180–2189.

[126] Tian Wang, Anastasios Bezerianos, Andrzej Cichocki, and Junhua Li. 2020. Multikernel capsule network for schizophrenia identification. *IEEE Trans. Cybern.* 52 (2020), 4741–4750.

[127] Zhijian Wang, Likang Zheng, Wenhua Du, Wenan Cai, Jie Zhou, Jingtai Wang, Xiaofeng Han, and Gaofeng He. 2019. A novel method for intelligent fault diagnosis of bearing based on capsule neural network. *Complexity* June (2019) 1–17.

[128] Xin Wen, Zhizhong Han, Xinhai Liu, and Yu-Shen Liu. 2020. Point2SpatialCapsule: Aggregating features and spatial relationships of local regions on point clouds using spatial-aware capsules. *IEEE Trans. Image Process.* 29 (2020), 8855–8869.

[129] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. 2017. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5028–5037.

[130] Lemeng Wu, Xingchao Liu, and Qiang Liu. 2021. Centroid Transformers: Learning to Abstract with Attention. *arXiv preprint arXiv:2102.08606* (2021).

[131] Congying Xia, Chenwei Zhang, Xiaohui Yan, Yi Chang, and Philip S. Yu. 2018. Zero-shot user intent detection via capsule neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

[132] Zhang Xinyi and Lihui Chen. 2018. Capsule graph neural network. In *Proceedings of the International Conference on Learning Representations*.

[133] Ziwei Xu, Xudong Shen, Yongkang Wong, and Mohan Kankanhalli. 2021. Unsupervised motion representation learning with capsule autoencoders. In *Proceedings of the 35th Conference on Neural Information Processing Systems*.

[134] Min Yang, Wei Zhao, Jianbo Ye, Zeyang Lei, Zhou Zhao, and Soufei Zhang. 2018. Investigating capsule networks with dynamic routing for text classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 3110–3119.

[135] Samira Zare and Hien Van Nguyen. 2021. PICASO: Permutation-Invariant Cascaded Attentional Set Operator. *arXiv preprint arXiv:2107.08305* (2021).

[136] Chenwei Zhang, Yaliang Li, Nan Du, Wei Fan, and Philip S. Yu. 2018. Joint slot filling and intent detection via capsule neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 5259–5267.

[137] Ningyu Zhang, Shumin Deng, Zhanlin Sun, Xi Chen, Wei Zhang, and Huajun Chen. 2018. Attention-based capsule networks with dynamic routing for relation extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

[138] Richard Zhang. 2019. Making convolutional networks shift-invariant again. In *Proceedings of the International Conference on Machine Learning*. PMLR, 7324–7334.

[139] Xinsong Zhang, Pengshuai Li, Weijia Jia, and Hai Zhao. 2019. Multi-labeled relation extraction with attentive capsule network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7484–7491.

[140] XiaoQing Zhang and Shu-Guang Zhao. 2019. Cervical image classification based on image segmentation preprocessing and a CapsNet network model. *Int. J. Imag. Syst. Technol.* 29, 1 (2019), 19–28.

[141] Wei Zhao, Haiyun Peng, Steffen Eger, Erik Cambria, and Min Yang. 2019. Towards scalable and reliable capsule networks for challenging NLP applications. *arXiv preprint arXiv:1906.02829* (2019).

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

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

[144] Yi Zhou, Hui Zhang, Hana Lee, Shuyang Sun, Pingjun Li, Yangguang Zhu, ByungIn Yoo, Xiaojuan Qi, and Jae-Joon Han. 2021. Slot-VPS: Object-centric Representation Learning for Video Panoptic Segmentation. *arXiv preprint arXiv:2112.08949* (2021).

[145] Zhiyu Zhu, Gaoliang Peng, Yuanhang Chen, and Huijun Gao. 2019. A convolutional neural network based on a capsule network with strong generalization for bearing fault diagnosis. *Neurocomputing* 323 (2019), 62–75.

Received 27 June 2022; revised 7 February 2024; accepted 15 June 2024