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.


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, etc. 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 utilised 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 modelling 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 which 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 optimised 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 1a 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.
Table 1. Organization of capsule network literature covered in this survey split into the most prominent research areas.

<table>
<thead>
<tr>
<th>Research Area</th>
<th>Literature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capsule Routing Algorithms (Sections 3 &amp; 4)</td>
<td>[4, 22, 30, 44, 50, 64, 80, 91, 101, 106, 120]</td>
<td>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.</td>
</tr>
<tr>
<td>Video &amp; Motion with Capsules (Section 5.1)</td>
<td>[27, 28, 85, 88, 133]</td>
<td>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.</td>
</tr>
<tr>
<td>Graph/Geometric Capsules (Section 5.2)</td>
<td>[42, 60, 78, 113, 116, 128, 132, 142]</td>
<td>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.</td>
</tr>
<tr>
<td>Generative Capsules (Section 5.3)</td>
<td>[29, 58, 91, 112]</td>
<td>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).</td>
</tr>
<tr>
<td>NLP with Capsules (Section 5.4)</td>
<td>[16, 18, 61, 75, 79, 121, 125, 131, 134, 136, 138, 139, 141, 143]</td>
<td>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.</td>
</tr>
<tr>
<td>Capsules in Medical Imaging (Section 5.5)</td>
<td>[2, 3, 33, 68, 69, 76, 81, 90, 126, 140]</td>
<td>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.</td>
</tr>
<tr>
<td>Other Applications of Capsule Networks (Appendix A)</td>
<td>[24, 43, 84, 92, 94, 98, 127, 145]</td>
<td>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 network architectures to be proposed, but these works do not have sufficient overlap to warrant their categorisation.</td>
</tr>
</tbody>
</table>

Nonetheless, as shown in Figure 1b, 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 organisation 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. [124] was written shortly after capsule networks were first introduced, so it does not cover large milestones achieved recently. Similarly, [110] was written to be brief and therefore covers a very small portion of the literature. [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 organised 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 like vector-valued neural activities (capsules), agreement and capsule routing. In Section 4, we review the most prominent capsule
(a) Translation Invariance

(b) Viewpoint Invariance

(c) Translation Equivariance

(d) Viewpoint Equivariance

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.

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. Lastly, 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’d often like the final prediction of our model to be invarient 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 centre (rotational 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 generalisation and design [9, 54].

**Translation Invariance.** To humans, both images shown in Figure 2a 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:

\[ f(x) = f(T(x)). \]  

(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 2b. 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 very good at extrapolating object appearance to novel viewpoints. However, this is not the case for standard CNNs [51] as they often struggle to generalise 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 modelled 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 generalisation 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 2c, a change in translation of the input leads to equivalent changes in neural activities. More formally, f is equivariant with respect to (w.r.t.) translation T if:

\[ f(T(x)) = T(f(x)). \]  

(2)

Unlike the translation case, the output \([x' \ast f]\) is not simply a rotated version of \([x \ast f]\) as convolution is not rotation equivariant.

---

ACM Comput. Surv.
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 \textit{place-coded} equivariance, since a discrete change in the input \( x \) results in a discrete change in which neurons are used to encode it.

\textbf{Viewpoint Equivariance.} It is more challenging to capture \textit{viewpoint equivariance} in a model (see Figure 2d). 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 \textit{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 \textit{place-coded} to \textit{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.

\section*{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 \textit{equivariance} w.r.t. viewpoints in neural activities, and \textit{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 \textit{non-linear} effects on pixel intensities, but \textit{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, that leverage robust representations to better \textit{generalise} 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

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{picasso.png}
\caption{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 [103, 108].}
\end{figure}
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 \( y \in \mathbb{R} \). (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 \( y \in \mathbb{R}^k \) of neural activities.

an observer and attempt to disentangle salient features of objects into 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 is 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 labelled as a person’s face [108], since all the parts are in agreement w.r.t. the face.

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 generalise to all viewpoints – which is the goal of capsule networks.

Sub-sampling & Convolution. The issue of poor generalisation to novel viewpoints in CNNs is exacerbated by pooling (sub-sampling) operators, which discard potentially pose-aware information in favour of training/inference
Fig. 8. A 2D example of capsule routing. All \( n \) part capsules \( x_i \in \mathbb{R}^2 \) in the lower layer vote for each object capsule \( y_j \in \mathbb{R}^2 \) above by: \( v_{ij} = W_{ij} x_i \). If the votes for an object capsule agree and form a cluster there is iterative feedback to increase the routing weights \( y_{ij} \) between the object capsule and the part capsules’ votes that form the cluster, whilst decreasing them for other object capsules.

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, [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, [50] demonstrated that capsule networks can better generalise to novel viewpoints compared to CNNs of a similar size, and [106] showed that they are considerably better than CNNs at recognising 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 which 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 which 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 parameterised function 

\[ c(X; W) : \mathbb{R}^{N \times D} \rightarrow \mathbb{R}^P, \]

where \( X = \{x_i\}_{i=1}^N \) is a set of \( N \) input signals \( x_i \in \mathbb{R}^D \), and \( W = \{W_i\}_{i=1}^N \) is a set of \( N \) transformation weight matrices \( W_i \in \mathbb{R}^{P \times D} \). A dynamic routing\(^1\) process – akin to clustering – gives rise to routing coefficients: \( y = \{y_i\}_{i=1}^N, y_i \in [0, 1] \), which represent the affinity between each input signal vector \( x_i \) and the 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. Lastly, an activation function \( \varphi(\cdot) \) is

\(^1\)How this routing process is defined is explained in detail later.
applied, and the output capsule $y \in \mathbb{R}^p$ is given by

$$y = \varphi \left( \sum_{i=1}^{N} y_i W_i \cdot x_i \right).$$

(3)

Notice that, unlike the artificial neuron, a capsule outputs a vector $y \in \mathbb{R}^p$ 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, colour, size, texture etc.

**Capsule Layers.** In adjacent capsule layers we have $i = 1, \ldots, N$ lower level part capsules $x_i \in \mathbb{R}^D$ (considered the inputs), and $j = 1, \ldots, M$ higher level object capsules $y_j \in \mathbb{R}^p$ (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 $\ell$ 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 $\ell$ to object capsules $j = 1, \ldots, M$ in layer $\ell + 1$, by iteratively adjusting routing coefficients $y \in \mathbb{R}^{N \times M}$, where $0 \leq y_{ij} \leq 1$. These routing coefficients are like an attention matrix which 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 generalising 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**

The idea of using capsules instead of neurons as the building blocks in a neural network was first introduced by [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
Algorithm 1 Dynamic Routing-by-Agreement

1: function ROUTING(x, W, r)
2: ∀ i, j capsules in layer ℓ and ℓ + 1 : y_ij ← 0
3: ∀ i, j : v_ij ⊲= W_ij · x_i
4: for r iterations do
5: ∀ i ∈ ℓ : y_i ← softmax(y_i)  // routing weights
6: ∀ j ∈ ℓ + 1 : s_j ← ∑_i y_ij v_ij
7: ∀ j ∈ ℓ + 1 : y_j ← squash(s_j)  // Eq. (4)
8: ∀ i, j : a_ij ← v_ij · y_j
9: ∀ i, j : y_ij ← y_ij + a_ij
10: return y_j

The idea of routing-by-agreement was first introduced in the seminal work by [106], and since then many other variants of capsule routing have been proposed. Their routing process is shown in Algorithm 1, whereby vectors are iteratively updated based on the agreement between the output y_j of each higher layer capsule j, and the prediction (votes) v_ij made by each lower layer capsule i. The agreement is measured by the scalar product: α_ij = v_ij · y_j. To gain some intuition about this procedure, imagine y_j and v_ij are both unit vectors, i.e., their magnitudes are: |y_j| = 1 and |v_ij| = 1. Then, the dot product between them is equal to cos θ, where θ is the angle between the two vectors. The resulting agreement α_ij is added to y_ij, updating part-object affinities based on the extent to which the vectors y_j and v_ij point in the same direction.

3.2 Dynamic Routing Between Capsules

The dynamic routing between capsules is the angle between the two vectors. The resulting agreement is measured by the scalar product: α_ij = v_ij · y_j. To gain some intuition about this procedure, imagine y_j and v_ij are both unit vectors, i.e., their magnitudes are: |y_j| = 1 and |v_ij| = 1. Then, the dot product between them is equal to cos θ, where θ is the angle between the two vectors. The resulting agreement α_ij is added to y_ij, updating part-object affinities based on the extent to which the vectors y_j and v_ij point in the same direction.

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:

\[ y_j = \frac{\|s_j\|}{1 + \|s_j\|^2} s_j = \sum_i y_ij W_{ij} \cdot x_i. \] (4)

The routing coefficients y_ij are iteratively updated based on the agreement between the output y_j of each higher layer capsule j, and the prediction (votes) v_ij made by each lower layer capsule i. The agreement is measured by the scalar product: α_ij = v_ij · y_j. To gain some intuition about this procedure, imagine y_j and v_ij are both unit vectors, i.e., their magnitudes are: |y_j| = 1 and |v_ij| = 1. Then, the dot product between them is equal to cos θ, where θ is the angle between the two vectors. The resulting agreement α_ij is added to y_ij, updating part-object affinities based on the extent to which the vectors y_j and v_ij point in the same direction.

3.2.2 Margin Loss Function. The objective function presented in [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 \( \|y_k\| \), 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...
Algorithm 2 Expectation-Maximisation Routing

1: function EM-Routing(a, M, W, r)
2: ∀ i, j capsules in layer ℓ and ℓ + 1: γij ← M−1
3: ∀ i, j: Vji ← Mi · Wij > voting
4: for r iterations do
5: ∀ i ∈ ℓ: γij ← γij ∘ a;
6: ∀ j ∈ ℓ + 1: μj, σj ← M-step(γ, V)
7: costh ← (βa + log σj 2) ∑iγij
8: a’j ← sigmoid(λ(βa − ∑h costh)) > activate
9: ∀ i ∈ ℓ: yi ← E-step(μ, σ, a’, V) > update
10: return a, M

loss used is [106]:

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

where T_k = 1 if a (digit of class) k is present, m^+ = 0.9, m^- = 0.1 and λ = 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 x_i ∈ ℝ^D as before, the authors use M_j ∈ ℝ^V x νD capsule pose matrices, and a separate activation probability a ∈ ℝ to represent the presence of the entity modelled 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-Maximisation (EM) algorithm [23].

3.3.1 Matrix Capsule Voting. In [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: V_{ji} = M_i · W_{ij}, where W_{ij} ∈ ℝ^4x4, V_{ji} denotes the vote from the i_th part capsule for a j_th object capsule, and W_{ij} is the transformation weight matrix. Since both the pose matrices and the transformations weights are 4x4, each capsule’s vote is V_{ji} ∈ ℝ^4x4. See Figure 12 for a simple 2D translation example. We can motivate the use of 4x4 matrices through a geometric interpretation: i.e. 4x4 transformations matrices are commonly used in 3D computer graphics under homogeneous coordinates for perspective projection [5]. A 4x4 matrix can represent the following transformations among others: translation, rotation, reflection, glides, scale, contraction, expansion, shear, dilation etc. Assuming a capsule network is able to extract sensible entities from the input, 4x4 pose and transformation weight matrices are theoretically sufficient.
Fig. 12. How capsules deal with viewpoint changes (2D translation example). An object can be detected by looking for agreement between votes $V_j$ for its pose matrix $M_j$. Lower level (part) capsules produce a vote by multiplying their pose matrix $M_i$ by a learned viewpoint-invariant transformation matrix $W_{ij}$. Agreement between multiple parts means that their votes are similar: $V_j \approx V_{jk}$, 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 that any voting agreement will persist [50].

It is worth noting the distinction between the voting procedure in Eq. (4) and this one: i.e in the former the vote is calculated via the matrix-vector product: $v_{ji} = W_{ij}x_i$, whereas in the latter we have a matrix-matrix product: $V_j = M_iW_{ij}$. Once again we can provide a geometric interpretation that justifies the order of the product, since due to the non-commutativity of square matrices: $M_iW_{ij} \neq W_{ij}M_i$. In geometric terms, $M_iW_{ij}$ applies a transformation on the pose matrix $M_i$ defined by matrix $W_{ij}$, whereas $W_{ij}M_i$ would wrongly imply that $W_{ij}$ is the pose matrix and $M_i$ 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 $W_{ij}$ 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 generalise knowledge across all spatial locations in the image like CNNs, whilst replacing basic max/average pooling operations in favour of capsule routing-by-agreement.

### 3.3.3 Expectation-Maximisation Routing

Similar to [106], a non-linear procedure to route between adjacent capsule layers they call EM-Routing (see Algorithm 2) is proposed in [50]. Concretely, the procedure is a version of the Expectation-Maximisation (EM) algorithm [23], that iteratively adjusts the means, variances and activation probabilities of the capsules in layer $\ell + 1$, and the assignment probabilities between all capsules (routing weights $y$). Unlike Dynamic routing [106], EM-Routing fits Gaussian distributions on the votes coming from part capsules $i$ to object capsule $j$ poses: $N(M_j \mid \mu_j, \sigma_j^2) \forall j$, where each capsule $j$ (of $N_j$ total object capsules in layer $\ell + 1$) has a diagonal covariance matrix with $h$ components: $\sigma_j^2 \in \mathbb{R}^h$. Following Algorithm 2 closely, EM-Routing iterates between updating the means $\mu_j$, variances $\sigma_j^2$ and activations $a_j$ of capsules $j$ whilst holding the routing coefficients $y_{ij}$ fixed (M-step), and updating $y_{ij}$ holding $\mu_j$, $\sigma_j^2$ and $a_j$ fixed (E-step). For a more detailed explanation, the reader may refer to [50].
### Algorithm 3 Variational Bayes Routing

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: <code>function</code> VB-Routing($\alpha$, $\mathbf{M}$, $\mathbf{W}$, $r$)</td>
<td></td>
</tr>
<tr>
<td>2: $\forall i, j$ capsules in layer $L_i$ and $L_j$: $y_{ij}$ $\leftarrow N_j^{-1}$</td>
<td></td>
</tr>
<tr>
<td>3: $\forall i, j$: $V_{j</td>
<td>i} = \mathbf{M}<em>i \cdot W</em>{ij}$ $\triangleright$ voting</td>
</tr>
<tr>
<td>4: $\forall j \in L_j$: $\alpha_0, m_0, \kappa_0, \Psi_0, v_0$ $\triangleright$ initialise priors</td>
<td></td>
</tr>
<tr>
<td>5: <code>for</code> $r$ iterations <code>do</code></td>
<td></td>
</tr>
<tr>
<td>6: $\forall i \in L_i$: $y_{ij} \leftarrow y_{ij} \odot \alpha_i$ $\triangleright$ routing weights</td>
<td></td>
</tr>
<tr>
<td>7: $\forall j \in L_j$: Update $q^*(\pi, \mu, \Lambda)$</td>
<td></td>
</tr>
<tr>
<td>8: $\forall i \in L_i$: Update $q^*(z)$</td>
<td></td>
</tr>
<tr>
<td>9: $\forall j \in L_j$: $a'_j = \sigma(\beta_a - (\beta_u + E[\ln \pi_j] + E[\ln \det(\Lambda_j)])$</td>
<td></td>
</tr>
<tr>
<td>10: <code>return</code> $a'$, $\mathbf{M}$</td>
<td></td>
</tr>
</tbody>
</table>

---

### 3.3.4 Capsule Activation

As shown in Algorithm 2, vote agreement is measured using the variance $\sigma_j$ of each object capsule’s Gaussian, which is then weighted by its support: $\sum_i y_{ij}$, i.e., the number of part capsules $i$ assigned to object capsule $j$. To set the activation probability $a_i$ for a particular object capsule $j$, [50] compare the description lengths (energies $-\beta_u$ and $-\beta_v$ 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: $Bernoulli(p)$, that minimises 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 [50] directly maximises the gap between the activation of the (final layer) capsule representing the target class $a_t$, and the other class capsules:

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

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 [50] that EM-Routing matrix capsules outperform [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 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 modelling 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 $\mu$ and $\Lambda$, which are the mean and inverse covariance (precision) matrix of each object capsule’s Gaussian distribution, and over $\pi$ which are the mixing coefficients of the mixture model. The latent variables $z = \{z_i, \ldots, z_{L_j}\}$ are a set of one-hot vectors describing the cluster assignments of each of the lower capsules’ votes $V_{j|i}$ to higher capsules’ Gaussian distributions. Variational inference (VI) [52, 57] of the above latent
Algorithm 4 Capsule Layer with Routing Uncertainty

1: function ConvCaps2D (a, M, W)
2: ∀ i, j capsules in ℓ, ℓ + 1: Vjj i ← Mj · Wij \rightarrow voting
3: ∀ i ∈ ℓ: x(i) 0 ∈ R^N→i \rightarrow set Dirichlet priors
4: ∀ i ∈ ℓ : x(i) ∼ qθ(x(i+1)) \rightarrow posterior sample
5: ∀ j ∈ ℓ + 1 : μj, σj \rightarrow route (xℓ+1, Vjj i)
6: ∀ j ∈ ℓ + 1 : a'_j = σ(−ηj \mathbb{E}(S_j)^{-1} \mathcal{H}(M_j)) \rightarrow activation
7: return a', M

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 W_{ij}, with which its datapoints (votes) are transformed, so every cluster sees a different view of the data. In practice, the authors approximate the entropy up to constant factors with:

H(M_j) = \mathbb{E}(\ln \det W_{ij}) \approx \mathbb{E}(\ln \det M_j).

The joint distribution of the model is: p(V, z, π, μ, Λ) = p(V|z, μ, Λ)p(z|π)p(π)p(μ|Λ)p(Λ), and the posterior is approximated with a factorised variational distribution over all the latent variables: p(z, π, μ, Λ|V) \approx q(z)q(π)q(μ, Λ). To perform capsule routing, the authors iteratively optimise parent capsule parameter distributions: q(π, μ, Λ), with the responsibilities z over child capsules fixed, and re-evaluate the new expected responsibilities \( q^*(z) \) 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 [6, 12, 101].

3.4.2 Agreement & Capsule Activation. To measure agreement, [101] propose to use the differential entropy of higher capsule j's Gaussian-Wishart variational posterior distribution \( q^*(μ_j, Λ_j) \). In simple terms, if the entropy of a capsule j's Gaussian is low that means the 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: \( \mathbb{H}[q^*(μ_j, Λ_j)] \approx \mathbb{E}(\ln \det(Λ_j)) \). 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 \( σ(\cdot) \), where \( β_0, β_a \) 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, [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 Variational Inference

\[ \pi \rightarrow \text{VIM} \]
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 which 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 \( p(z|D, W) \) over part-object latent connections \( z \) given the data \( D \) 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 \( q_\phi(z) \) that minimises \( \text{KL}(q_\phi(z) \| p(z|D, W)) \), 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.

**Priors.** The authors [22] place a prior distribution \( p(z^{(i)}) \) over each part capsule’s \( c_i \in \ell \) connections to the object capsules they vote for \( c_j \in \ell + 1 \), and assume fully factorised independence across layers: \( p(z) = \prod_{\ell=1}^{L-1} \prod_{i=1}^{N} p(z^{(i)}), \) with \( z^{(i)} = (z_1, \ldots, z_{N_{c_j}}) \sim p(z^{(i)}), \) where the part-object connections vector of each capsule \( c_i \) is \( z^{(i)} \in \mathbb{R}^{N_{c_j}}, \) with \( N_{c_j} \) 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 \( q_\phi(z^{(i)}_{\ell+1}) \) 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 \( \ell \) are the part capsules of \( \ell + 1 \) and so on. The model is then fit end-to-end by maximising a variational lower bound on the conditional marginal log-likelihood \( \log p(y|x) \) (see [22] for further details).

3.5.1 Routing & Activating Capsules. As in previous work [50, 101], matrix capsules \( M \in \mathbb{R}^{D \times 4} \) and convolutional capsule voting are used (see Figure 14). During training, the authors [22] fit multivariate Gaussians \( M_j \sim \mathcal{N}(\mu_j, \sigma_j) \), on each object capsule’s \( D=16 \) dimensional poses, and randomly sample part-object connections from the approximate posterior at each capsule layer \( \ell: z^{(i)} = (z_1, \ldots, z_{N_{c_j}}) \sim q_\phi(z^{(i)}_{\ell+1}), \forall c_i \in \ell \), 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: \( q_\phi(z|D) \approx p(z|D, W) \) is inferred for all layers at once, rather than performing local (iterative) inference of \( z \) 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 [101], and measure vote agreement via the average negative entropy of each capsule’s Gaussian: \( -\mathcal{H}[\mathcal{N}(\mu_j, \sigma_j)] = -\frac{1}{2} \sum_{i=1}^{D} \log(\sigma_j^{(i)} \sqrt{2\pi\sigma_j}) \). The agreement is weighted by the (normalised) support (# parts assigned to an object) for each capsule, and activated using the logistic function \( \sigma(\cdot) \), where \( E(S_j) \) 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 [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 \( z \) from the approximate posterior, and calculate the predictive entropy \( \mathcal{H}(\tilde{y}|x, z, W) \) of the model’s output distribution with sampled \( z' \sim q_\phi(z|D) \). Under full posterior learning: \( q_\phi(z, W) \), the pose transformation matrices \( W \) 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 which 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 \( \mathbf{x}_m \) (two rotations, two translations, scale and shear), a presence variance \( \mathbf{d}_m \in [0, 1] \), 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 \( \hat{T}_m \) 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, \( d_m \)s are also used to weight the part-capsules’ log-likelihood, so that we do not take absent points into account. This is achieved by raising the likelihood of the \( m^{th} \) 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 maximising 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 [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 [44] proposed a simpler self-routing strategy.

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

ACM Comput. Surv.
inspired by Mixture-of-Experts. In Self-Routing Capsule Networks (SR-CapsNet) proposed by [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 $\hat{u}_{ij}$ involves two learnable weight matrices $W_{{}^{\text{route}}}$ and $W_{{}^{\text{pose}}}$ respectively. For each layer of the routing network, each pose vector $u_i$ is multiplied by a trainable weight matrix $W_{{}^{\text{route}}}$ 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 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: $\text{pose}^{\text{proj}}_i = \text{MLP}^{\text{proj}}(\text{pose}_i)$, where $\text{pose}_i$ corresponds to the $i^{\text{th}}$ pose matrix of the lower-level capsules. Within the shared subspace $S$, one has $n$ learnable prototype vectors $Q = (q_1, q_2, \ldots, q_n)$, where each $q_i \in \mathbb{R}^d$. Compared to other non-iterative capsule approaches, such as [45], ProtoCaps uses fewer FLOPS whilst 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 [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 $T_{ij}$, is to estimate the attention matrix $A_{ij} \in \mathbb{R}^{p \times p}$ between lower and higher level capsules. The straight-through router $R_{ij}$ decides which capsules to connect/disconnect. As shown in Figure 17, given the attentive matrix $A_{ij}$, the straight-through router $R_{ij}$ 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 $R_{ij}$ 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 [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

In [120], the authors proposed a routing algorithm for capsule networks inspired by the attention mechanism commonly found in transformers [123] (see Fig. 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 layer sequentially. In their experiments, the authors showed that their method compares favourably 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 [64] who use an autoencoder for generation). In [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 $n = 1, \ldots, N_k$. 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, etc. 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 in order 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.

ACM Comput. Surv.
4 Understanding Attention & Capsules

As previously alluded to in Section 1, there are notable conceptual similarities between capsule routing and the self-attention mechanism popularised 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

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 \( x_i \in \mathbb{R}^D \). 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 \( x_i \) are known as token embedding vectors in transformers and capsules in capsule networks. To illustrate this, let \( X = (x_1, \ldots, x_N) \in \mathbb{R}^{N \times D} \) denote a matrix of input token embedding vectors or capsules \( x_i = (x_{i1}, \ldots, x_{id}) \). Consider the computation of a single output token \( y_j \in \mathbb{R}^P \) in simple self-attention, given a sequence of input tokens \( \{x_i\}_{i=1}^N \), where \( A \in \mathbb{R}^{N \times M} \) is an attention weight matrix between all input/output pairs:

\[
y_j = \sum_{i=1}^N A_{ij} x_i,
\]

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

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

\[
Q = X W_Q, \quad K = X W_K, \quad V = X W_V,
\]

where \( W_Q, W_K \) and \( W_V \) are \( D \times P \) dimensional parameter matrices, and \( Q, K \) and \( V \) are therefore \( N \times P \) dimensional. Let’s now consider the calculation of the attention weights for a single input token \( x_i \). Given \( x_i \)'s query row vector \( Q_{i:} \in \mathbb{R}^{1 \times P} \), its attention weights \( A_{i:} \in \mathbb{R}^{1 \times N} \) are given by the normalised dot product with each of the key vectors:

\[
A_{i:} = \text{softmax} \left( \frac{Q_{i:} K^T}{\sqrt{P}} \right)
\]

The dot product agreement between token \( x_i \)'s query \( Q_{i:} \) and all the keys \( K \) dictates how much "value" \( V = (V_1, \ldots, V_N) \in \mathbb{R}^{N \times P} \) from each other token should be represented in token \( x_i \)'s revised representation. That is, a
Comparing capsule routing and self-attention in Pytorch code [96]. Both operations take $\mathbf{x}_i \in \mathbb{R}^D$, and produce $\mathbf{y}_j \in \mathbb{R}^P$, 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 $Q$ in self-attention, and the capsule votes $\mathbf{V}$ play the role of both the keys $K$ and the values $V$. A notable difference is that self-attention can handle variable-length ($\mathbf{v}$) sequences naturally, whereas capsule routing has a more rigid structure defined by the weights $W \in \mathbb{R}^{N \times D \times P}$. This can be relaxed via weight sharing across capsules, e.g. using $W \in \mathbb{R}^{M \times D \times P}$. Self-attention is more parallelizable and better suited for current hardware accelerators compared to the sequential and iterative nature of capsule routing.

Single output token $\mathbf{Y}_{i,:} \in \mathbb{R}^{1 \times P}$ is simply a weighted average of the input token’s values:

$$ Y_{i,:} = (AV)_{i,:} = \sum_{j=1}^{N} A_{ij} V_{j,:}, $$

for $i = 1, \ldots, N$ and $\rho = 1, \ldots, P$. Each output token $\mathbf{y}_i := \mathbf{Y}_{i,:} \in \mathbb{R}^{1 \times P}$ 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) $\mathbf{W} \in \mathbb{R}^{N \times D \times P}$:

$$ V_{i,:} = \sum_{d=1}^{D} W_{idp} X_{id}, $$

for $i = 1, \ldots, N$ and $\rho = 1, \ldots, P$. 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 $\{\mathbf{x}_i\}_{j=1}^N$, i.e. capsule. In the first routing iteration, the output of the $j$th capsule is an average of its votes: $\mathbf{y}_j = \frac{1}{M} \sum_{i=1}^{N} V_{i,j}$, 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: $\mathbf{y}_j = \sum_{i=1}^{N} A_{ij} V_{i,j}$.

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 Eq. (10) we can use here without breaking the softmax normalisation. Moreover, since each output capsule receives
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: $Y \sim N(0, \sigma^2) \in \mathbb{R}^{\mathcal{N} \times \mathcal{P}}$. This $Y$ sequence plays the role of the slots/capsules, which we infer given $X \in \mathbb{R}^{\mathcal{N} \times \mathcal{D}}$ via an iterative attention procedure similar to capsule routing. In contrast, the initial slots/capsules $Y$ in capsule routing are typically a function of $X$, not randomly initialized.

$i = 1, \ldots, \mathcal{N}$ votes, in order to compute $j = 1, \ldots, \mathcal{M}$ output capsules we require the transformation weights to be a 4-tensor: $W \in \mathbb{R}^{\mathcal{N} \times \mathcal{M} \times \mathcal{D} \times \mathcal{P}}$. Thus there is a learned $\mathcal{D} \times \mathcal{P}$ 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 Eq. (13) with new estimates from Eq. (14):

\begin{align*}
(1. \text{Voting}) \quad V_{ijp} &= \sum_{d=1}^{\mathcal{D}} W_{ijdp} X_{id}, \\
(2. \text{Average votes}) \quad y_{jp} &= \frac{1}{\mathcal{M}} \sum_{i=1}^{\mathcal{N}} V_{ijp}, \quad (13) \\
(3. \text{Attention}) \quad A_{ij} &= \text{softmax}\left(\sum_{p=1}^{\mathcal{P}} V_{ijp} y_{jp}\right), \\
(4. \text{Output capsule}) \quad y_{ij} &= \sum_{i=1}^{\mathcal{N}} A_{ij} V_{ijp}, \quad (14)
\end{align*}

for each $j = 1, \ldots, \mathcal{M}$ output capsule. Notably, Eq. (14) entails a softmax normalisation over outputs, rather than inputs as in self-attention (Eq. (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 modelling assumption which induces a preference for certain solutions. Inductive biases often
consist of encoding useful prior assumptions about the target function mapping inputs to outputs, that can aid in generalisation 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 $W$ is biased towards encoding invariance to viewpoint transformations, and the capsule vectors are biased towards capturing equivariance of neural activities. On the other hand, the self-attention mechanism induces weaker inductive biases since there are no equivariance or single-parent assumptions like 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 modelling 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 normalisation 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.

Lastly, 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 categorised 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 which can be used to access independent object representations simultaneously. There are four main types of slots as outlined by [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.

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 specialise 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, whilst being more efficient.

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 which 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 which attempts to minimise 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 which 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 which 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 utilised in as a 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 [39, 122]. N-EM is a probabilistic model which attempts to group pixels within an image into \( K \) entities whose properties are described by a vector \( \theta_k \). A differentiable Expectation Maximization (EM) algorithm is used to find these groupings by computing the Maximum Likelihood Estimate for each \( \theta_k \). 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 time steps, in a video sequence? Second, how can computationally costly routing operations scale to deal with video inputs? There is no current capsule work which completely answers these questions, but there have been several works which apply capsule networks to various video and motion problems.

Traditional 2D convolutional capsule routing was extended to 3D convolutional routing in [27] (see Figure 24). In 3D convolutional routing, capsules which 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 (8 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 [28], where the authors propose CapsuleVOS, a network which 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 which 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.

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 [85]. Instead of obtaining a single set of video capsules from 3D convolutions, two sets of capsules S-Caps and T-Caps are obtained which 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 [88] propose 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 which 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, in order 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] that 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 modelled 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 modelling into CapsNets, hence treating capsules as node feature vectors and representing them as graphs so that one can leverage graph structure information. As highlighted by Zhao et al. [142], processing 3D point clouds is a challenging 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 [142] extends the work presented in [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 \textit{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 in order to learn discriminative shape representation. Pooling-based feature aggregation methods struggle to achieve this satisfactorily.

A way to overcome is presented in [128] whereby a new architecture called "Point2SpatialCapsule" is proposed that consists of two parts: a) a module named \textit{geometric feature aggregation} is designed to aggregate the local region features into learnable cluster centres, which manages to encode the spatial locations from the original 3D space, and b) a module named \textit{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 \textit{spatial-aware capsules}. The complete architecture can be seen in Figure 26. A self-supervised 3D point cloud capsule architecture was proposed by Sun \textit{et al.} [116] termed \textit{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 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 \textit{shifting temporal coherence}. For further technical details, the reader may refer to [60].

5.2 Generative Capsule Networks

There exist several studies which propose implementations of Generative Adversarial Networks (GANs) [37] using capsule networks. One of the earliest works was proposed by Jaiswal \textit{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 [106]. On the other hand, in the work presented...
in [29], a new CapsNet model was proposed, called Subspace Capsule Network (SCN), which is built upon the idea of modelling 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 modelled by the capsule subspace. A purportedly interpretable variant of capsule networks termed iCaps was proposed in [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 [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 [91] as discussed previously in Section 4. Smith et al. introduced a variational bound which 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 modelling 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 amortised 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 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 categorisation, news categorisation, 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 6 datasets when compared with standard neural network methods like 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 which 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 which 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. Lastly, 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 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 which 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 which 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 which 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 \((i.e. \text{ rating})\) 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 [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 generalise to new variations of the learnt classes in unseen data which 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 labelling. 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%.

Building upon their previous work in [68], [69] extends their method to five new datasets as well as introducing the condition that child capsules are only routed within a spatially located window. This change allows their
network to scale to images of 512x512 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 is able to outperform state-of-the-art networks in five datasets DICE coefficient and outperforms in four of the five datasets in 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] in order to scale to 3D data. Their contributions are to add a constraint which allows only one Capsule per pixel location in the Primary Capsules. By doing this they reduce the computational overhead of routing by agreement by 32x resulting in a 3x overall increase of network training per epoch while retaining approximately the same accuracy as base CapsNets on 2D images which 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 whilst 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 Tumour classification, CNNs have been shown to fail to utilise the spatial relationships between brain perturbations which causes misdiagnosis of tumours. As a result of this, the authors in [3] propose using a locally routed Capsule Network similar to [90]. Additionally, rough boundaries of where the tumours 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 tumour boundaries, the network achieves 78% accuracy on their dataset of MRI scans. This is then improved to 90.89% when additionally given the rough tumour boundaries, slightly improving upon the state-of-the-art CNNs which are able to achieve 88.33% accuracy when given the rough tumour 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. Utilising 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, [76] propose MHA-CoroCapsule a novel capsule architecture which 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 100 thousand images of other lung diseases compared to the 1 thousand total images used in MHA-CoroCapsules.

6 Discussion & Future Directions

In this paper, 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
modelling 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, [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, [8] conclude that current frameworks have been highly optimised 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 like those induced by the routing and voting procedures in capsule networks are a lot slower than they could be. As pointed out by [53], while capsule network’s operations can be implemented reasonably well on CPUs, performance drops drastically on accelerators like 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 which can capture their inherent inductive biases whilst 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 in order 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 [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]. On the other hand, 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. Lastly, 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 work 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.

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] corroborates 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 [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, whilst being more efficient. There is also an opportunity to develop Vision Transformer (ViT) [26] style capsule architectures which do away with the expensive convolutional capsules formulation in favour 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 [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 [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 modelling 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.

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

ACM Comput. Surv.
Visual cognition


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 generalise 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 which affect CNNs. In [98], the authors showed that the reconstruction element of capsule networks improves the robustness to standard CNN adversarial attacks. In [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 which 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 [145] the authors convert 1D signals of vibration and electrical current to time-frequency graphs via Fourier transforms which 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 which 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 6 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 6 tasks, compared to 94.58% of the best-competing method. Additionally, ICNs achieve 82.05% when tested on data for 6 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 [145], the authors of [127] propose an extension which 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 over approaches, showing clearly the strength of Capsule Networks ability to generalise.

**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 [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 5 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 [24] the authors build upon the work in [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 [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 normalised 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 in order 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 6 different datasets. Building upon the work in [92], the authors of [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.

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