Learning with Capsules: A Survey

Fabio De Sousa Ribeiro, Kevin Duarte, Miles Everett, Georgios Leontidis, and Mubarak Shah

Abstract—Capsule networks were proposed as an alternative approach to Convolutional Neural Networks (CNNs) for learning object-centric representations, which can be leveraged for improved generalization and sample complexity. Unlike CNNs, capsule networks are designed to explicitly model part-whole hierarchical relationships by using groups of neurons to encode visual entities, and learn the relationships between these entities. Promising early results achieved by capsule networks have motivated the deep learning community to continue trying to improve their performance and scalability across several application areas. 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. The aim of this survey is to provide a comprehensive overview of the capsule network research landscape, which will serve as a valuable resource for the community going forward. To that end, we start with an introduction to the fundamental concepts and motivations behind capsule networks, such as equivariant inference in computer vision. We then cover the technical advances in the capsule routing mechanisms and the various formulations of capsule networks, e.g. generative and geometric. Additionally, we provide a detailed explanation of how capsule networks relate to the popular attention mechanism in Transformers, and highlight non-trivial conceptual similarities between them in the context of representation learning. Afterwards, we explore the extensive applications of capsule networks in video and motion (e.g. video object segmentation, regression tracking, and action video recognition), natural language processing (e.g. text classification, relation extraction, language and vision, and recommendation systems), medical imaging (e.g. semantic segmentation of lesions, brain tumour classification), fault diagnosis (e.g. bearing fault diagnosis), hyperspectral imaging and forgery detection. To conclude, we provide an in-depth discussion regarding the main hurdles in capsule network research, and highlight promising research directions for future work.

Index Terms—Deep learning, capsule networks, deep neural networks, convolutional neural networks, transformers, routing by agreement, self attention, representation learning, object-centric learning, generative models, clustering, computer vision.

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 [1] consists of a set of tools which 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) [2]. Despite the many successes of modern DL-based vision systems [2], [3], [4], a general lack of robustness to distributional shifts remains prevalent [5]. Indeed, unlike current systems, humans are able to quickly adapt to distributional changes using very few examples to learn from [6], [7], [8]. 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 [9], [10], [11]. 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 [12]. This inability was the motivation for a series of models called capsule networks [12], [13], [14], [15], [16]. 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 [17]. Like many other developments in machine learning [18], [19], 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 [8]. This idea of vector-valued units is at the heart of both capsule networks and soft-attention mechanisms [8], [20], [21], including the transformer [22]. As shown later, 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 works, Barham et al. [23] 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 [24] and PyTorch [25] 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 [26], [27], [28], [29], the underlying problem is not only caused by routing but by the capsule voting procedure as well. In their analysis, [23] 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 [30], 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. We hope this survey will inspire researchers to develop suitable tools for capsule networks.

In this survey, we provide a comprehensive overview 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 lack of a reliable point of reference. Nonetheless, as shown in Figure 1b, the number of capsule network related publications at the top venues has continued to steadily increase. We believe that there is now sufficient material to warrant a detailed organisation of the various concepts, techniques and foundational ideas which would benefit the community and spark research interest in the area. At the time of this writing, there exist only three other Capsule Network based surveys. [31] was written shortly after Capsule Networks were first introduced, so it does not cover large milestones achieved more recently. Similarly [32] was written with the purpose of being brief and therefore covers a very small portion of the relevant literature. [33] is more recent, and covers a larger breadth of papers, but does not extensively survey the field with sufficient detail in the way we feel is necessary. Conversely, the purpose of this survey is to provide the first comprehensive and detailed breakdown of capsule networks and related research on object-centric representation learning. Specifically, we aim to: (a) Explain the foundations, motivations and fundamental concepts behind capsule networks in detail; (b) Survey the state of the art in capsule network research in various application areas; (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 networks going forward, and will help contribute towards the advancement of the field.

This survey is organised as follows. In Section 1, we provide an introductory overview of the ideas behind capsule networks. In Section 2, we begin with a gentle introduction to invariance and equivariance, and explain why these concepts are fundamental in representation learning. In Section 3 we explain the foundational ideas and motivations behind capsule networks, and introduce basic concepts such as agreement and capsule routing. In Section 4, we delve into the most prominent capsule routing algorithms proposed in literature. In section 5, we uncover the conceptual similarities between capsule routing and the popular self-attention mechanism in Transformers. Sections 6 to 10 discuss some major applications of capsule networks for video and motion, graphs, natural language processing, and medical imaging. Section 11 focuses on other applications of capsule networks, such as fault diagnosis, hyperspectral imaging, forgery detection, and adversarial attacks. Lastly, in Section 12 we discuss open challenges and shortcoming of capsule networks, along with what we believe are promising directions for future research.

2 Background & Motivation

To motivate capsule networks, we begin with a gentle introduction to invariance and equivariance, and explain why these concepts are fundamental in representation learning.

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 invariant to transformations of the input
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:

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

Sub-sampling techniques typically used in CNNs, such as max-pooling, make neural activities of the next layer locally invariant to input translations [1], [35]. 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 [13]. 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, as any change in viewpoint can be modeled by a linear operation on these poses.

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 provides 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 [13], [34], [36], [37]. Two fields of particular interest are Group CNNs [36], [38],
and capsule networks [14], [15], [16], [41], which are both predicated upon the notion that intermediate NN layers should not be fully invariant, because the relative pose of local features should be preserved for future layers to aid in generalisation to new transformations [13], [36].

Translation Equivariance. One of the simplest examples of equivariance is translation equivariance afforded by the convolution operation in CNNs [42]. As shown in Figure 2c, a change in translation of the input leads to equivalent changes in neural activities. More formally, is equivariant with respect to (w.r.t.) translation if:

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

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

Viewpoint Equivariance. It is more challenging to capture viewpoint equivariance in a model (see Figure 2d). That is, changes in viewpoint that lead to equivalent changes in neural activities. Like before, is said to be equivariant w.r.t. viewpoint (perspective) projection of the input if:

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

Convolution is not equivariant to transformations other than translation, which makes it challenging for CNNs to deal with viewpoint changes [13], [36] (see Figure 3). As explained later, capsule networks [14], [15], [16] look to move from place-coded to rate-coded viewpoint equivariance in the final layers, whereby a real-valued change in the input results in an equivalent real-valued change in neuronal output (capsule pose vectors).

3 Capsule Network Foundations

Although capsule networks have taken on several different forms since their inception [13], [14], [15], [16], they are generally built upon the following core assumptions and premises [41]:

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

In theory, a perfect instantiation of the above premises could yield more sample-efficient models, that leverage robust representations to better 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 [9], [10]. Capsules imitate this concept by representing neural activities as poses of objects with respect to a coordinate frame imposed by an observer, and attempt to disentangle salient features of objects into their composing parts. This is reminiscent of inverse graphics [43], 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. 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 [8], [44].

3.1 The Picasso Problem

In the ideal case, capsule networks address the ‘Picasso problem’: i.e. images of an object containing all the right parts – but that are not in the correct spatial relationship – are often misclassified as said object by typical DL-based systems. To gain some intuition, consider the example of an image of a person’s face (object), whereby the positions of the various parts of the face (e.g. mouth, eyes and nose) are shuffled, and the image is still (wrongly) classified as a person’s face. As depicted in Figure 4, although every image is composed of the same parts, only the green bordered examples ought to be labelled as a person’s face [45].

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 [13], [46]. In fact, as discussed later, CNNs focus much more on textures than shapes, unlike humans [46] (see Figures 5, 6). 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. simple pose matrix multiplication used in computer graphics) to generalise to all viewpoints at once – which is the goal of capsule networks.

### 3.2 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 speed and performance [14], [15]. 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 [46], [47], [48]. For examples of this phenomenon see Figures 5 and 6. These biases are also corroborated by the fact that adversarial examples are often visually indistinguishable to humans [49], [50]. On that note, [44] showed that capsule networks use features that are more aligned with human perception, and therefore have the potential to address the central issue raised by adversarial examples. Moreover, [15] demonstrated that capsule networks can better generalise to novel viewpoints compared to CNNs of a similar size, and [14] showed that capsule networks 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 early results on pose-aware tasks highlight their potential [14], [15], [41].

### 3.3 Capsule Networks

In order to explain what a capsule network is, we begin by comparing the traditional artificial neuron found in standard NNs, and the capsule as shown in Figure 8.

**Capsule.** A capsule is a biologically inspired structure based on hypercolumns, wherein groups of tightly-connected nearby neurons are thought to represent vector-valued units which are able to transmit not only scalar quantities, but a set of coordinated values [8].

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 process – akin to clustering – gives rise to routing coefficients: \(\gamma = \{\gamma_i\}_{i=1}^N\), \(\gamma_i \in [0, 1]\), which represent the affinity between each input signal vector \(x_i\) and the output capsule in question. As explained later, the process by which these routing coefficients are obtained takes into account the context from other capsules. Lastly, an activation function \(\varphi(\cdot)\) is applied, and the output capsule \(y \in \mathbb{R}^P\) is given by

\[
y = \varphi\left(\sum_{i=1}^N \gamma_i W_i \cdot x_i\right).
\]

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, texture etc.

![Agreement](original.png) ![Disagreement](texturised.png)

**Fig. 7:** Example of agreement and no agreement between part poses (eyes, nose, mouth etc) with respect to the object (person), where each pose is encoded by a vector. Capsule routing aims to detect objects by looking for agreement between its parts, and thereby perform equivariant inference.
Fig. 8: (a) Illustration of the artificial neuron as found in standard neural networks. The artificial neuron receives an input signal vector \( x \in \mathbb{R}^d \) 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 \( x_i \in \mathbb{R}^d \), and outputs a vector \( y \in \mathbb{R}^k \) of neural activities.

Fig. 9: Simple 2D example of capsule routing. Here, all \( n \) capsules from the lower layer \( x_i \in \mathbb{R}^2 \) vote for each higher capsule \( y_j \in \mathbb{R}^2 \), by: \( v_{ij} = W_{ij} \cdot x_i \). If the votes for a higher capsule agree and form a cluster, then there is feedback to increase the routing weights \( \gamma_{ij} \) between the higher capsule and the lower capsules that form the cluster, whilst decreasing them for other higher capsules.

**Capsule Layer.** In a capsule layer we typically 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). As explained in greater detail later, the initial part capsules are extracted from the raw input (e.g. images), and the object capsules of a layer \( \ell \) become the part capsules of the next layer up in a hierarchical fashion until the final layer.

### 3.4 What is Capsule Routing?

Capsule routing is a non-linear, iterative and 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 the routing coefficients \( \gamma \in \mathbb{R}^{N \times M} \), where \( 0 \leq \gamma_{ij} \leq 1 \). 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 9.

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 detector in a local pool in the layer below. As shown in the example Figure 7, 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. 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 [14], [15], [41].

### 4 Capsule Routing Mechanisms

Having provided the basic necessary background and motivation behind capsule networks, in this section we focus on providing an in-depth overview of the most prominent capsule routing algorithms proposed in literature. Our exposition (loosely) follows chronological order starting with the seminal works on capsules and ending on recent work.

Fig. 10: The transforming autoencoder for modelling translations. There are 3 capsules and each one has 3 recognition units (red) and 4 generation units (green). Figure from [13].
Algorithm 1 Dynamic Routing-by-Agreement

1: function ROUTING($\mathbf{x}, \mathbf{W}, r$)
2:     $\forall i,j \in \text{capsules in layer } \ell$ and $\ell + 1 : \gamma_{ij} \leftarrow 0$
3:     $\forall i,j : \mathbf{v}_{ij} \leftarrow \mathbf{W}_{ij} \cdot \mathbf{x}_i$  \hspace{1cm} $\triangleright$ voting
4:     for $r$ iterations do
5:         $\forall i \in \ell : \gamma_i \leftarrow \text{softmax}(\gamma_i)$  \hspace{1cm} $\triangleright$ routing weights
6:         $\forall j \in \ell + 1 : \mathbf{s}_j \leftarrow \sum_i \gamma_{ij} \mathbf{v}_{ij}$
7:         $\forall j \in \ell + 1 : \mathbf{y}_j \leftarrow \text{squash}(\mathbf{s}_j)$  \hspace{1cm} Eq. (4)
8:         $\forall i,j : \alpha_{ij} \leftarrow \mathbf{v}_{ij} \cdot \mathbf{y}_j$  \hspace{1cm} $\triangleright$ agreement
9:         $\forall i,j : \gamma_{ij} \leftarrow \gamma_{ij} + \alpha_{ij}$  \hspace{1cm} $\triangleright$ update
10:    return $\mathbf{y}_j$

4.1 Transforming Autoencoders

The idea of using capsules instead of neurons as the building blocks in a neural network was first introduced by [13]. 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.

As previously mentioned, although several stages of subsampling in a CNN can afford some invariance to pose over a limited range, it ignores precise spatial relationships which are crucial for generalisation [13]. The authors propose the transforming autoencoder as a way to learn the first level of capsules, whereby pixel intensities are converted to pose parameters. As shown in the example in Figure 10, transforming autoencoders can receive the input image and a desired shift, $\Delta x$ and $\Delta y$, and output the shifted input by merging information from generative capsule units. In Figure 10, $p$ is the probability that the visual entity modelled by a particular capsule is present in the input image. The authors also studied the prediction of more complex 2D transformations and changes in 3D viewpoint by using $3 \times 3$ matrix representations of the desired $\Delta$’s, demonstrating the merit of the approach in their preliminary experiments.

4.2 Dynamic Routing Between Capsules

The idea of routing-by-agreement was first introduced in the seminal work by [14], and since then many other variants of capsule routing have been proposed. Their routing process is shown in Algorithm 1, whereby vectors $\mathbf{x}_i \in \mathbb{R}^D$ of lower layer capsules are transformed by weights $\mathbf{W}_{ij} \in \mathbb{R}^{D \times P}$ to make predictions for the vectors $\mathbf{y}_j \in \mathbb{R}^P$ of higher layer capsules. If a lower layer capsule $i$ (e.g. encoding a nose) predicts the properties of a possible parent capsule $j$ (e.g. encoding a face) with high accuracy, there is top-down feedback which increases the affinity (routing coefficient $\gamma_{ij}$) between them. The proposed capsule network architecture in [14] is shown in Figure 11, and the decoder in Figure 12 is used to reconstruct the input.

4.2.1 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 [14]. To that end, the authors proposed the following non-linear “squashing” function to activate every $j^{th}$ capsule:

$$y_j = \frac{\|s_j\|^2 s_j}{1 + \|s_j\|^2 \|s_j\|^2}, \quad s_j = \sum_i \gamma_{ij} \mathbf{W}_{ij} \cdot \mathbf{x}_i. \quad (4)$$

The routing coefficients $\gamma_{ij}$ 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:

$$\alpha_{ij} = v_{ij} \cdot y_j. \quad (5)$$

To gain some intuition, 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 $\cos \theta$, where $\theta$ is the angle between the two vectors. The agreement $\alpha_{ij}$ is added to $\gamma_{ij}$, updating part-object affinities based on how well the vectors $y_j$ and $v_{ij}$ point in the same direction.

4.2.2 Margin Loss Function

The objective function presented in [14] 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 loss used by [14] is defined as follows:

$$L_{\text{margin}} = \sum_k T_k \max(0, m^+ - \|y_k\|)^2$$

$$+ \lambda (1 - T_k) \max(0, \|y_k\| - m^-)^2, \quad (7)$$

where $T_k = 1$ iff a (digit of class) $k$ is present, $m^+ = 0.9$, $m^- = 0.1$ and $\lambda = 0.5$. The $\lambda$ 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 vectors.
4.3 Matrix Capsules with EM-Routing

More recently, a new version of capsules was proposed [15], which overcomes the following deficiencies of [14]:

(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 use of typical objective functions [15]. Instead, they propose to separate 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 \( x_i \in \mathbb{R}^D \) as before, the authors use \( M_i \in \mathbb{R}^{\sqrt{D} \times \sqrt{D}} \) and a separate activation probability \( a_i \in \mathbb{R} \) to represent presence of the entity modelled by each capsule [15]. They also presented a new capsule network architecture featuring convolutional capsules, and proposed an alternative routing-by-agreement procedure based on the Expectation-Maximisation (EM) algorithm [51].

4.3.1 Matrix Capsule Voting

In [15], the voting procedure from capsules \( i \) in a lower layer \( \ell_i \) for the pose matrices of capsules \( j \) in a higher layer \( \ell_j \) is:

\[
V_{j|i} = M_{i} \cdot W_{ij}, \quad W_{ij} \in \mathbb{R}^{4 \times 4}.
\]

where \( V_{j|i} \) denotes the vote from the \( i \)th part capsule for the \( j \)th object capsule, and \( W_{ij} \) is the transformation weight matrix. Since both the pose matrices and the transformations weights are \( 4 \times 4 \), each capsule’s vote is \( V_{j|i} \in \mathbb{R}^{4 \times 4} \). See Figure 14 for a simple 2D translation example.

**Geometric Interpretation.** We can motivate the use of \( 4 \times 4 \) matrices through a geometric interpretation: i.e. \( 4 \times 4 \) transformations matrices are commonly used in 3D computer graphics under homogeneous coordinates for perspective projection [52]. A \( 4 \times 4 \) 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, \( 4 \times 4 \) pose and transformation weight matrices are theoretically sufficient.

Algorithm 2 Expectation-Maximisation Routing

1. function EM-ROUTING(\( a, M, W, r \))
2. \( \forall i, j \) capsules in layer \( \ell \) and \( \ell + 1; \gamma_{ij} \leftarrow M^{-1} \)
3. \( \forall i, j; V_{j|i} \leftarrow M_i \cdot W_{ij} \) \( \triangleright \) voting
4. for \( \ell \) iterations do
5. \( \forall i \in \ell : \gamma_{ij} \leftarrow \gamma_{ij} \odot a_i \) \( \triangleright \) routing weights
6. \( \forall j \in \ell + 1 : \mu_j, \sigma_j \leftarrow M\text{-STEP}(\gamma, V) \)
7. \( \text{cost}^{h} \leftarrow (\beta_a + \log \sigma_j^h) \sum_i \gamma_{ij} \)
8. \( a^j_i \leftarrow \text{sigmoid}(\lambda(\beta_a - \sum_h \text{cost}^{h})) \) \( \triangleright \) activation
9. \( \forall i \in \ell : \gamma_i \leftarrow E\text{-STEP}(\mu, \sigma, a^i, V) \) \( \triangleright \) update
10. return \( a, M \)

It is worth noting the distinction between the voting procedures in eq. (4) and eq. (8): i.e. in the former the vote is calculated via the matrix-vector product: \( v_{j|i} = W_{ij}x_i \), whereas in the latter we have a matrix-matrix product: \( V_{j|i} = M_iW_{ij} \). 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.

4.3.2 Convolutional Capsules

In [15], the authors also introduced the idea of convolutional capsules, whereby the connectivity between capsules in adjacent layers, follows that of a CNN. 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 capsule network (see Figure 13). This extension 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 pooling operations in favour of routing-by-agreement.

4.3.3 Expectation-Maximisation Routing

Similar to [14], a non-linear procedure to route between adjacent capsule layers they call EM-Routing (see Algorithm 2) is proposed in [15]. Concretely, the procedure is a version of the Expectation-Maximisation (EM) algorithm [51], 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 \( \gamma \)).

Unlike Dynamic routing [14], EM-Routing fits Gaussian distributions on the votes coming from part capsules \( i \) to object capsule \( j \) poses: \( \mathcal{N}(M_j | \mu_j, \sigma_j^2) \) \( \forall j \), 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 \( \gamma_{ij} \) fixed (M-step), and updating \( \gamma_{ij} \) holding \( \mu_j, \sigma_j^2 \) and \( a_j \) fixed (E-step). For more detailed explanation please refer to [15].

Fig. 13: The capsule network architecture with 3 convolutional capsule layers (ConvCaps) used in [15]. The number of weights in each layer is shown above.
4.3.4 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: $\sum_{i} \gamma_{ij}$, i.e. amount of part capsules $i$ assigned to object capsule $j$. To set the activation probability $a_j$ for a particular object capsule $j$, [15] compare the description lengths (energies $-\beta_a$ and $-\beta_u$ 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($\rho$), that minimises free-energy when the difference in the energies is its argument (see line 8 in Algorithm 2).

4.3.5 Spread Loss function

The “spread” loss function used in [15] directly maximises the gap between the activation of the (final layer) capsule representing the target class $a_t$, and the other class capsules:

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

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 [15] that EM-Routing matrix capsules outperform [14], and significantly outperform comparable size CNNs on viewpoint-invariance and adversarial robustness tasks.

4.4 Capsule Routing via Variational Bayes

In [29], the authors propose Variational Bayes (VB) routing as a way to address some of the inherent drawbacks of EM-Routing [15] encountered by various other authors such as training instability and reproducibility [28], [53], [54]. In EM, variance-collapse [55] singularities occur when an object capsule (Gaussian cluster) claims sole custody of a part capsule (datapoint), 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 between capsule layers [29]. 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) Known pathological solutions of the EM algorithm (variance-collapse) are addressed in a principled manner.

**Algorithm 3 Variational Bayes Routing**

1: function VB-Routing($a, M, W, r$) 
2: $\forall i, j$ in layers $L_i$ and $L_j$: $\gamma_{ij} \leftarrow N_j^{-1}$ 
3: $\forall i, j$: $V_{ji} \leftarrow M_i \cdot W_{ij}$ $\triangleright$ voting 
4: $\forall j \in L_i$: $a_0, m_0, k_0, \Psi_0, v_0$ $\triangleright$ initialise priors 
5: for $r$ iterations do 
6: $\forall i \in L_i$: $\gamma_{ij} \leftarrow \gamma_{ij} \odot a_i$ $\triangleright$ routing weights 
7: $\forall j \in L_j$: $q^*(\pi, \mu, \Lambda)$ 
8: $\forall i \in L_i$: $q^*(z)$ 
9: $\forall j \in L_j$: $a'_j = \sigma(\beta_a - (\beta_u + \mathbb{E} [\ln \pi_j] + \mathbb{E} [\ln \det(\Lambda_j)]))$ 
10: return $a', M$

![Fig. 14: How capsules deal with viewpoint changes (2D translation example). An object can be detected by looking for agreement between votes $V_{ji}$ for its pose matrix $M_i$. 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: $V_{ji} \approx V_{jk}$, for $i \neq k$, which is unlikely to occur by chance in high dimensions [14], [15]. As the viewpoint changes, all the pose matrices change in a coordinated manner, so that any vote agreement will persist [15].](image)

![Fig. 15: Depiction of VB-Routing between convolutional capsule layers. Each capsule has an activation $a$ and a pose matrix $M \in \mathbb{R}^{4 \times 4}$. Parent capsules $j \in \mathcal{L}_j$ (blue) only receive votes from child capsules $i \in \mathcal{L}_i$ (orange) within their receptive field. $c$ and $d$ denote the number of child and parent capsule types (channels) respectively. Figure from [29].](image)
4.4.1 Variational Bayes Routing

See Figure 15 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 $z = \{z_1, \ldots, z_{|\ell|}\}$ are a set of one-hot vectors describing the cluster assignments of each of the lower capsules’ votes $V_{ji}$ to higher capsules’ Gaussian distributions. Variational inference (VI) [56], [57] of the above latent variables is performed [29] – analogously to Bayesian Gaussian mixture models [58], [59] – between all adjacent 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 [15], [29].

The *generative story* for each part capsule $i$ is that of a mixture model with priors over object capsule $j$ parameters $\pi, \mu, \Lambda$. The joint distribution of the model factors as: $p(V, z, \pi, \mu, \Lambda) = p(V | z, \pi, \mu, \Lambda) p(z | \pi) p(\pi) p(\mu | \Lambda) p(\Lambda)$, and the posterior is approximated with factorised variational distribution over all the latent variables:

$$p(z, \pi, \mu, \Lambda | V) \approx q(z) q(\pi) \prod_{j \in I_i} q(\mu_j, \Lambda_j).$$

Coordinate Ascent Updates. To perform routing between adjacent capsule layers, the authors [29] iteratively optimise parent capsule parameter distributions: $q(\pi, \mu, \Lambda)$, 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 an EM-like algorithm with variational EM updates as outlined in Algorithm 3. For further details on the standard closed-form update equations of Bayesian mixture models refer to [29], [58], [59].

4.4.2 Agreement & Capsule Activation

To measure agreement, [29] propose to use the differential entropy of higher capsule $j$’s Gaussian-Wishart variational posterior distribution $q^*(\mu_j, \Lambda_j)$. 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:

$$\mathbb{H}[q^*(\mu_j, \Lambda_j)] \approx \mathbb{E}[\ln \det(\Lambda_j)].$$

As outlined in Algorithm 3, this agreement measure is then weighted by the amount of support for each object capsule (mixing coefficient) and activated via the logistic function $\sigma(\cdot)$, where $\beta_a, \beta_u$ are (optional) parameters learned discriminatively, similar to EM-Routing [15]. Unlike EM [15] or Dynamic routing [14] however, capsules are only activated after the routing iterations.

4.5 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, [41] 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 (VI) of part-object connections in a 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 direct influence on their end-to-end variational free-energy objective [41].

4.5.1 Inference & Model Assumptions

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 [41]. To circumvent this, the authors [41] use stochastic VI tools to find the best approximation $q^*(z)$ that minimises $D_{KL}(q^*(z) \parallel p(z | D, W))$, where $z$ are global latent part-object connection variables, and $W$ are viewpoint-invariant transformation parameters, in a CapsNet with $L$ layers.

Prior. The authors [41] place a prior distribution $p(q^{(i)})$ 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:

$$p(z) = \prod_{\ell=1}^{L-1} \prod_{i=1}^{N_\ell} p(z^{(i)}_{\ell}),$$

**Algorithm 4 Capsule Layer with Routing Uncertainty**

1: function CONVCAPS2D (a, M, W)
2: \hspace{1em} $\forall i, j$ capsules in $\ell, \ell + 1 : V_{ji} \leftarrow M_{ij} \cdot W_{ij}$ \hspace{1em} $\triangleright$ voting
3: \hspace{1em} $\forall i \in \ell : \pi_i \leftarrow \mathbb{R}^{N_{\ell-i}}$ \hspace{1em} $\triangleright$ set Dirichlet priors
4: \hspace{1em} $\forall i \in \ell : z^{(i)} \sim q_{\phi}(z^{(i)}_{\ell, \ell+1})$ \hspace{1em} $\triangleright$ posterior sample
5: \hspace{1em} $\forall j \in \ell + 1 : \mu_j, \sigma_j \leftarrow \text{ROUTE}(z, a_{\ell+1}, V_{ji})$
6: \hspace{1em} $\forall j \in \ell + 1 : a_j' = \sigma(-\eta \epsilon(S_j)^{-1} H(M_j))$ \hspace{1em} $\triangleright$ activation
7: return $a', M$
the connections vector \( z^{(i)} \in \mathbb{R}^{N_i \to j} \), of each \( c_i \) is:

\[
z^{(i)} = (z_1, \ldots, z_{N_{i \to j}}) \sim p(z^{(i)}), \quad \forall c_i \in \ell,
\]

where \( N_{i \to j} \) denotes the number of object capsules \( j \) that each part capsule \( i \) votes for in a particular layer (see Figure 16 for convolutional capsule voting example). Different choices of prior are considered, and the authors opt for Dirichlet priors due to a reduced parameter count. A mean-field variational approximation \( q_\phi(z_{\ell, \ell+1}) \) to the (intractable) posterior on part-object connection variables is made between all adjacent capsule layers. The model is defined hierarchically where the object capsules in layer \( \ell \) are the part capsules of \( \ell + 1 \) and so on.

**Free Energy Objective.** The model is fit end-to-end by maximising a lower bound on the conditional marginal log likelihood \( \log p(y|x) \). In practice, VI of latent variables \( z \), and maximum a posteriori (MAP) inference of \( W \) is performed [41].

### 4.5.2 Routing & Activating Capsules

As in previous work [15], [29], matrix capsules \( M \in \mathbb{R}^{4 \times 4} \), and convolutional capsule voting are used (see Figure 16). During training, the authors [41] fit multivariate Gaussians \( M_j \sim N(\mu_j, \sigma_j) \), on each object capsule’s \( D=16 \) dimensional poses, and randomly sample part-object connections from the approximate posterior at each capsule layer \( \ell \):

\[
z^{(i)} = (z_1, \ldots, z_{N_{i \to j}}) \sim q_\phi(z_{\ell, \ell+1}), \quad \forall c_i \in \ell,
\]

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: \( q_\phi(z|D) = p(z|D, W) \) is inferred for all layers at once, rather than performing local (iterative) inference of \( z \) in the E-step of EM-Routing [15] between each pair of adjacent capsule layers.

**Agreement & Activation.** To activate capsules, [41] follow the general concept of [29], and measure vote agreement via the average negative entropy of each capsule’s Gaussian:

\[
-\mathcal{H}(N(M|\mu_j, \sigma_j)) \triangleq -\frac{1}{D} \sum_{i=1}^{D} \log \left( \sigma_j^{(i)} / \sqrt{2\pi} \right)
\]

The agreement is weighted by the (normalised) support (# parts assigned to an object) for each capsule, and activated using the logistic function \( \sigma(.) \). As shown in Algorithm 4, \( E(S_j) \) is the average support each object capsule receives in a given layer, \( S_j \sim \text{Binomial}(N_i, N_j^{-1}). \)

### 4.5.3 Uncertainty Quantification

With their method, the authors [41] 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 \( \mathcal{H}(\hat{y}|x, z, W) \) 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.

### 4.6 Stacked Capsule Autoencoders

Kosiorek et al. [16] introduced the Stacked Capsule Autoencoder (SCAE), which is a seminal work on formulating CapsNets as an unsupervised capsule autoencoder which explicitly uses geometric relationships to reason about objects. This work combines ideas from Transforming Autoencoders [13] and EM routing capsules [15] – 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 17, and it consists of two main parts: i) the Part Capsule Autoencoder (PCAE); ii) the Object Capsule Autoencoder (OCAE). A more detailed schematic of what each of these parts does can be seen in Figure 18.

In addition to the two stages mentioned above that are presented in the paper as being the main parts, there is also an earlier step that deals with abstracting away pixels and the part-discovery stage, called Constellation Capsule Autoencoder (CCA). 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.

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

**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 form objects. This is achieved by providing concatenated poses $x_m$, special features $z_m$, 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 [60] 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 [16].

**4.7 Inference for Generative Capsule Models**

In most previous works on capsules [14], [15] an inference algorithm is typically presented without specification of a corresponding generative model for the data (with the exception of [16] who use an autoencoder for generation). In [61] 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 capsules model that leads naturally to a variational algorithm for inferring the transformation of each object and the assignments of observed parts to the objects.

**4.8 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 [26] proposed a simpler self-routing strategy inspired by Mixture-of-Experts.

In Self-Routing Capsule Networks (SR-CapsNet) proposed by [26], each capsule independently defines its rout
ing 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 19). In self-routing, computing the routing coefficients \( c_{ij} \) and predictions \( \tilde{u}_{j|i} \) 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 performances on standard capsule network benchmarks, as well as improved robustness to adversarial attacks. Although this method is simple, it does somewhat limit the capsule network’s capability 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 [63].

4.9 Straight-Through Attentive Routing

One of the main drawbacks of CapsNets is computational complexity that stems from the complex mechanisms of the voting and routing processes. Even if the CapsNet architecture has a fixed number of parameters, the number of routing iterations can increase the training and inference time greatly. In [27] a non-recursive attention-based routing mechanism is proposed, inspired by the non-recurrent self-attention approach found in Transformers [22]. The proposed STAR-Caps layer architecture can be seen in Figure 20, and utilizes a straight-through attentive routing mechanism, formulating each capsule as a matrix rather than a vector like in EM routing [15].

STAR-Caps [27] 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 20, 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 they authors employ a straight-through estimator [64], [65], [66].

As usual, the ClassCaps layer outputs the final predictions, where each capsule vector represents a single class. Like [14], 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. Given the activations, they then calculate the spread loss as in EM routing [15] for training.

4.10 Inverted Dot-Product Attention Routing

In [28], the authors proposed a routing algorithm for capsule networks inspired by the attention mechanism commonly found in transformers [22][see Fig. 21]. 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 [14] since the capsule voting scheme and agreement is 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 routed concurrently rather than routing capsules in each layer sequentially.

5 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 [22]. 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 using similar notation. There is also significant conceptual overlap between capsule networks and other object-centric representation learning techniques [5], [67], thus we discuss previous research on using the attention mechanism for object-centric learning, and highlight conceptual similarities to the capsule formulation along the way. When appropriate, we adopt Einstein index notation to make it easier to highlight the similarities between the two methods, and provide respective code examples in Figure 23.
Let's now consider the calculation of the attention weights matrices, and \( W \) where respective query, key and value matrices as follows:

\[
\begin{align*}
W &= \text{torch.randn}(N, M, D, P) \\
V &= \text{torch.einsum}(\text{'ijd,p,dp->ijp'}, X, W) \\
Y &= (1./M) \times \text{torch.sum}(V, \text{dim}=0) \\
\text{agreement} &= \text{torch.einsum}(\text{'jp,ijp->ijp'}, Y, V, \text{transpose}(2,1)) \\
A &= \text{F.softmax}(\text{agreement}, \text{dim}=-1) \\
Y &= \text{torch.einsum}(\text{'ij,ijp->jp'}, A, V) \\
\text{return } Y
\end{align*}
\]

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 \( x_i \in \mathbb{R}^D \). These vector-valued units 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 \( \{x_i\}_{i=1}^N \):}

\[
y_j = \sum_{i=1}^N A_{ij} x_i, \quad (16)
\]

where \( A \in \mathbb{R}^{N \times M} \) is an attention weight matrix between all input/output pairs, 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 (\( M < N \)), making \( A \) no longer a square matrix like in self-attention.

**Self-Attention.** In scaled dot product self-attention [22], the \( N \) input vectors \( X \in \mathbb{R}^{N \times D} \) are transformed into respective query, key and value matrices as follows:

\[
\begin{align*}
Q &= XW^Q, \\
K &= XW^K, \\
V &= XW^V,
\end{align*}
\]

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 corresponding query row vector \( Q_{i,:} \in \mathbb{R}^{1 \times P} \), its attention weights \( A_{i,:} \) are given by the normalised dot product with each of the key vectors:

\[
A_{i,:} = \text{softmax}(\frac{Q_{i,:}K^T}{\sqrt{D}} \text{agreement}). \quad (18)
\]

The dot product \( \text{agreement} \) between token \( x_i \)'s query \( Q_{i,:} \) and all the keys \( K \) dictates how much “value” \( V \in (V_1, \ldots, V_N) \in \mathbb{R}^{N \times P} \) from each other token should be represented in token \( x_i \)'s revised representation. That is, a single output token \( Y_{i,:} \in \mathbb{R}^{1 \times P} \) is simply a weighted average of the input token’s values:

\[
Y_{ip} = (AV)_{ip} = \sum_{j=1}^N A_{ij} V_{jp}, \quad (19)
\]

for \( i = 1, \ldots, N \) and \( p = 1, \ldots, P \). Each output token \( y_i := Y_{i,:} \in \mathbb{R}^{1 \times P} \) constitutes the revised representation for \( x_i \).

**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 \( X \in \mathbb{R}^{N \times D} \) by a 3D learned parameter matrix (3-tensor) \( W \in \mathbb{R}^{N \times D \times P} \):

\[
V_{ip} = \sum_{d=1}^D W_{idp} X_{id}, \quad (20)
\]

for \( i = 1, \ldots, N \) and \( p = 1, \ldots, P \). The resulting votes are \( V = (V_1, \ldots, V_N) \in \mathbb{R}^{N \times P} \). Note that unlike in self-attention, here we have \( N \) separate weight matrices, one for each input vector-valued unit \( \{x_i\}_{i=1}^N \), i.e. capsule. In the first routing iteration, the output of the \( j^{th} \) capsule is an average of its votes:

\[
y_j = \frac{1}{N} \sum_{i=1}^N V_{ij}, \quad (21)
\]

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:

\[
y_j = \sum_{i=1}^N A_{ij} V_{ij}, \quad (22)
\]

Unfortunately, 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 \( \text{Eq. (18)} \) we can use here without breaking the softmax normalisation. Moreover, since each output capsule receives \( i = 1, \ldots, N \) votes, in order to compute \( j = 1, \ldots, M \) output capsules we require the transformation weights to be a 4-tensor: \( W \in \mathbb{R}^{N \times M \times D \times P} \). Thus there is a learned \( D \times P \) matrix between every input/output capsule pair.
estimates from Eq. (23): the initial uniform attention weights in Eq. (22) with new object (output capsule), and each object must compete with such that each part (input) capsule belongs to a single inductive biases induced by the capsule formulation are cases and reduce sample complexity.

Having established a formal relationship between self-attention and capsule routing, by iteratively refining the initial uniform attention weights in Eq. (22) with new estimates from Eq. (23):

\[ V_{ijp} = \sum_{d=1}^{D} W_{ijdp} X_{id}, \quad \text{(voting)} \tag{21} \]

\[ y_{jp} = \frac{1}{M} \sum_{i=1}^{N} V_{ijp}, \quad \text{(avg. votes)} \tag{22} \]

\[ A_{ij} = \text{softmax} \left( \sum_{p=1}^{P} V_{ijp} y_{jp} \right), \quad \text{(attention)} \tag{23} \]

\[ y_{jp} = \sum_{i=1}^{N} A_{ij} V_{ijp}, \quad \text{(output capsule)} \tag{24} \]

for each \( j = 1, \ldots, M \) output capsule. Notably, Eq. (23) entails a softmax normalisation over outputs, rather than inputs as in self-attention (Eq. (18)). As depicted in Figure 22, 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.

5.2 Comparing Inductive Biases

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.

Capsule. The exposition in Section 5.1 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 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.

Self-Attention. 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 [68], 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.

Common Ground. 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 is then dynamically adjustable based on the input, and the concepts themselves are refined based on global context.

5.3 Slots & Attention

Many object-centric representation learning approaches using neural networks can be categorised as being slot based [5]. As shown in Figure 24, 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 [5], 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 bind to multiple objects rather than a specific category.
Slot-Attention. Recent work by [67] showed how attention can be used to extract object-centric representations that enable generalization to unseen compositions, which is the motivation behind capsule networks. Indeed, the proposed method they call Slot Attention is reminiscent of recent developments in both capsule networks and self-attention. As shown in Figure 25, 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. 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 [12], 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, whilst being more efficient.

More recently, the authors in [69] proposed SCOUTER, a slot-attention based classifier for transparent, explainable and accurate classification (see Figure 26). The main difference between SCOUTER and vanilla slot attention is that the slots are now associated to single categories like in capsule networks. Indeed, the 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 to a certain output category. The authors in [69] 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 are applicable to capsule networks. The authors in [70] 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) (see Figure 27) 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.

5.4 Transformers for Routing & Sets

Sun et al. [71] propose a visual parser that attempt 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 refine the previous whole representation with the global information in this set of parts. Using this iterative encoder-decoder structure, the visual parser learn robust representations which can be applied to several tasks including image classification, object detection, and instance segmentation.

Carion et al. [72] 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 a $N$ inputs. Wu et al. [73] 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). Similarly, Roy et al. [74] 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 [60] 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 [16]: 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. [75] 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 down-stream tasks including classification, clustering, and anomaly detection.

### 5.5 Relational Neural Expectation Maximization

The learning of object-centric representations through Neural Expectation Maximization (N-EM) was studied by [77], [78]. 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. [78] 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 for learning object-centric representations of visual data.

### 6 Capsules for Video and 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 image-based capsule networks to the 3-dimensional video domain is non-trivial. Applying capsule networks to video data raises several questions. 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 the iterative and computational costly routing operations scale to deal with video inputs, which tend to be much larger than images conventionally processed by previous capsule-based approaches? 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.

Similar to how 2D image-based convolutions were generalized to 3D convolutions [80] to process a sequence of video frames, traditional 2D convolutional capsule routing was extended to 3D convolutional routing in [76]. 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, without modification, are unsuitable for 3D capsule networks. To this end, a capsule-pooling procedure is proposed, which averages each capsule types’ 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. [76] 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 spatio-temporally localizes the video by the use of a convolutional decoder.

This idea is extended in [82], 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 the object of interest throughout the input video clip.

A capsule-based approach for regression tracking has been proposed in [83]. 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 cost and obtain more discriminative capsule 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 [79] 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. From the input sentence, a set of sentence capsules are obtained. 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.

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 [81]. This work takes concepts from the stacked capsule autoencoders (see Section 4.6), 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 a approach to RGB videos is an interesting avenue for future work.

### 7 Geometric and Graph Based Capsules

In the past couple of years there have been several studies published that proposed CapsNets variations based on graphs that can better model the topological information of structured data and other types of data, such as social networks. Similarly, CapsNet 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. [84], 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). At the second stage the node embeddings are scaled via an attention module that together with dynamic routing generate the graph capsules (Block 2). At the last stage, graph classification takes place via dynamic routing (Block 3). The whole framework can be seen in Figure 31.

Li et al. [85] 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.

---

**Fig. 30:** The Motion Capsule Autoencoder (MCAE), figure from [81]. The MCAE presents an elegant capsule-based approach to directly model motion in input sequences. “Snippet” (part) and “segment” (object) capsules contain semantic-agnostic information over shorter and longer time frames respectively.

**Fig. 31:** Framework of CapsGNN taken from [84]. GNN is used to extract node embeddings and form primary capsules which are then scaled via an attention module and dynamically routed to generate graph capsules. Lastly, dynamic routing is applied again to perform graph classification.
Fig. 32: GraCapsNets architecture, showing how the extracted primary capsules are transformed and modeled as multiple graphs through attention-based graph pooling. Figure from [87].

Aiming at making CapsNets more interpretable, in a manner similar to Grad-CAM [86] that has been proposed for explaining CNN-based classifications, the method termed GraCapsNets proposed by Gu & Volker [87], modifies CapsNets to have built-in explanations. As shown in Figure 32, the part-part relationship, i.e. the relationship between primary capsules, is modeled with graphs, followed by graph pooling operations that pool relevant object parts from the graphs to make a classification vote. The idea is that since the graph pooling operation reveals which input features are pooled as relevant ones one can create explanations to explain the classification decisions. In addition to interpretability, the proposed model improves object recognition via integrating graph modeling into CapsNets, hence treating capsules as node feature vectors and representing them as graphs so that one can leverage graph structure information.

Srivastava et al. [88] proposed a geometric capsule design (Figure 33), 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 with respect to the viewer. The proposed Geometric Capsule Autoencoder is constructed to group 3D points into parts and these parts into objects in an unsupervised manner.

As highlighted by Zhao et al. [89], 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 [89] (Figure 34) extends the work presented in [90] and is able to process point clouds while maintaining equivariance to $SO(3)$ rotations and preserving translation and permutation invariance. This work achieves $SO(3)$ by restricting the model to a sparse set of local reference frames (LRFs) that collectively determine the object orientation. In addition, the authors proposed a variation of dynamic routing, termed Weiszfeld dynamic routing that uses inlier scores as activations, and which together with LRFs, form part of the quaternion equivariant capsule module. This process allows for equivariant latent representations to be extracted that point to local orientations and activations, while also disentangling orientation from evidence of object existence.

Another challenge when dealing with 3D point clouds concern 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 [91] whereby a new architecture called “Point2SpatialCapsule” is proposed that consists of two parts: a) a module named geometric feature aggregation is designed to aggregate the local region features into learnable cluster centers, which manages to encode the spatial locations from the original 3D space, and b) a module named spatial relationship aggregation is also proposed that further aggregates the clustered features and the spatial relationships among them in the feature space using a new capsule layer termed spatial-aware capsules. The complete architecture can be seen in Figure 35.

Recently, a self-supervised capsule architecture for 3D point clouds was proposed by Sun et al. [92] termed canonical capsules. The algorithm computes K-part capsule decom-

Fig. 33: Geometric Capsule model overview showing insight into how different part capsules represent different areas of the image along with details of the parts pose. These part capsules then activate the object capsule to describe the entire object and its pose. Figure from [88].

Fig. 34: Entire capsule network architecture based on a hierarchy of quaternion equivariant capsule modules. The input is a 3D point set $X$, where all the local patches are sent to the quaternion equivariant capsule (QEC) modules. At each level the points are pooled in order to increase the receptive field, gradually reducing the local reference frames into a single capsule per class. The classification and rotation estimation are used as supervision cues to train the network.

Fig. 35: GraCapsNets architecture, showing how the extracted primary capsules are transformed and modeled as multiple graphs through attention-based graph pooling. Figure from [87].
Fig. 35: Point2SpatialCapsule architecture. Assuming point clouds as input, the three steps involved are as follows: a) multi-scale features are extracted from multi-scale areas; b) feature-spatial embeddings are created and are aggregated via a spatial relationship aggregation step that considers both the embeddings and their spatial relationship; c) task-specific network to perform the downstream task. Figure from [91].

Fig. 36: Canonical Capsules framework: the model learns a capsule encoder for 3D point clouds by relating the decomposition result of two random rigid transformations \( T^a \) and \( T^b \), of a given point cloud. Figure from [92].

positions 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 takes place by assigning each point into one of the K parts via attention, which is then integrated into K keypoints. 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. The whole framework can be seen in detail in Figure 36.

It is well established that developing better inductive biases can lead to better deep neural network architectures. Many parts of our brain are organized topographically, such as the ocular dominance maps. Keller and Welling [93] 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 37). In their paper they refer to capsules as “learning sets of approximately equivariant features or subspaces”, and in fact 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 that the authors refer to as shifting temporal coherence. The mathematical background and further details can be found in [93].

8 GENERATIVE ADVERSARIAL CAPSULE NETWORKS

Since the inception of CapsNets, there have been a few studies that proposed implementations and variations of Generative Adversarial Networks (GANs) with CapsNets. One of the earliest work was proposed by Jaiswal et al. [94], where a CapsNets was used as the discriminator and a deep CNN as the generator. However, the authors did not propose a new routing mechanism but instead the model used was the one proposed in [14].

On the other hand, in the work presented in [95], a new CapsNet model (Figure 38) was proposed, called Subspace Capsule Network (SCN), which are built upon the idea of modeling the properties of an entity through a group of capsule subspaces instead of simply grouping neurons to create capsules. Using a learnable transformation, a capsule is then created by projecting an input feature vector from a lower layer onto the capsule subspace. This transformation finds the degree of alignment of the input with the properties modeled by the capsule subspace.

An interpretable variation of CapsNets termed iCaps was proposed in [96], 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, as can be seen in Figure 39, consists of six different parts:

- \( C_c \): a capsule network (classifier) that represents the class-relevant latent space.
- \( E \): an encoder that represents the class-irrelevant (residual) latent space.
- \( G \): a generator that creates synthetic images using \( C_c(x) \oplus E(x) \).
- \( D_G \): a discriminator for image generation, that distinguishes whether an observation is from the dataset or from \( G \).
Fig. 38: Overview of the Subspace Capsule Network (SCN) architecture. At the top (a), we can see the building blocks of SCN for the generative part of the GAN model, whereas at the bottom (b), a similar architecture is used for the discriminator component. Figure from [95].

- $C_G$: a classifier for image generation, that estimates class labels.
- $D_{CR}$: a discriminator for contrastive regularization that maximizes the distance between the concepts represented by $C_C$.

A probabilistic generative version of capsule networks (Figure 40) was proposed by Smith et al. in [97], 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 [61] 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 failures of the capsule assumptions and inference amortisation. Concretely, the authors expressed the modeling assumptions of capsules as a probabilistic model with joint distribution over all latent and observed random variables. They then derived a routing algorithm directly from variational inference principles, leading to an amortised method similar to variational autoencoders [98]. 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 on 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 be helpful for enforcing desirable equivariance properties, but that this is far from sufficient, as these models typically come without theoretical guarantees. They elaborate that, while promising, this type of formulation is still somewhat underdetermined. Specifically, there are issues relating to the identifiability of objects, which suggests changes to the generative model may be necessary going forward.

Fig. 39: iCaps architecture, showing all the building blocks that correspond to the Capsule Network, the Encoder, the Generator, the Discriminator for contrastive regularization, the Discriminator for real/fake, and the classifier for synthetic image. Figure from [96].

Fig. 40: A diagrammatic overview of the generative model for a capsule network. On the left sketch one can see the detailed connectivity between the random variables that correspond to the red circled region of the overall graph that can be seen on the right. Figure from [97].

9 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 spatial and semantic relationships between the sentences components. 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. [99] first explored the use of capsule networks in natural language processing for the problem of text classification. Here, the dynamic routing algorithm [14] 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 capsule network can achieve strong performance across 6 datasets when compared with standard neural network methods like CNNs and LSTMs. Kim et al. [100] 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 [101] 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 and leading to a large reduction on computational cost.

Another work [102] 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 illustrate the capsule networks’ ability to model the relationships between parts (words) and their corresponding whole (intents). Chen et al. [103] 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 [104]. 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. [104] 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. [105] 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 [106]. Another work [107] 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 [108]. 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 use 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. [109] 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. rating) is generated for the given item.

Language and Vision. Several applications require the use of both textual and visual data. Capsules 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 natural language answer. One of the first capsule-based approaches for this task is [110]. Here, routing-by-agreement is used as an attention mechanism between the visual and textual features to improved learned representations. Urooj et al. [111] propose a capsule framework for visual question answering grounding systems. This work shows that including capsules with EM-routing [15] in the generation of visual features leads to drastic improvement in grounding accuracy; the capsule architecture uses relevant visual information in intermediate reasoning steps. Recently, Cao et al. [112] present 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.

Fig. 41: Architecture of a Capsule Network for text classification in [99] The adjusted local spatial routing between the Primary Capsules and the Convolutional Capsules is shown in detail.

[Image 48x643 to 300x748]
Capsule networks have also been applied to multimodal machine translation [113]. The goal of this problem is to improve the translation of a natural language sentence with visual features. Lin et al. [113] 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.

10 Capsules for Medical Image Analysis

Another field that Capsules have been applied fairly extensively to is that of medical imaging [114], [115], [116], [117], [118]. This is largely in part due to a Capsule Networks 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 [119] replaces the convolutional blocks of a U-Net [120] 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 95.4%. Again, applied to detection of nodules in CT scans, Fast CapsNet [121] modified dynamic routing CapsNet [14] 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 [114] propose using a locally routed Capsule Network similar to [121]. 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.

Building upon their previous work in [119], [122] extends their work 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.

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 lungs. Motivated by Capsule Networks ability to achieve strong results with low amounts of data, COVID-CAPS [123] was devised. Utilising a standard dynamic routing 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, [124] propose MHA-CoroCapsule a novel capsule architecture which builds upon the dynamic routing capsules [14] 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 [123], 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.

11 Capsules in Other Applications

Capsule Networks have been applied to many different fields. The majority of these applications only use the original Dynamically Routed Capsules 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 than the training data.

11.1 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 [125] 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 [14] is the replacement of the initial convolutional layers with inception blocks from GoogLeNet [126] 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 [125], the authors of [127] propose an extension which proposes the Inception 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.

11.2 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 the 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 [128] 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 author 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 [129] the authors build upon the work in [128] 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%.

11.3 Forgery Detection
Detection of artificially generated forgery is a task where previously would require specific models for each variation of an attack. However in [130] 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 up into individual frames. These images are then fed through a VGG-19 [131] network to extract the latent features rather than the traditional convolutional layers prior to 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 [130], the authors of [132] 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.

11.4 Adverserial Attacks
The properties of Capsule Networks are desirable for a number of reasons, including how they are resistant to affine transformation and single pixel attacks which effect CNNs. In [44] authors show that the reconstruction element of Capsule Networks improves the robustness of the network against standard CNN attacks.

In [133] is shown that Dynamically Routed Capsule Networks [14] are able to maintain a 17.3% accuracy on the CIFAR-10 dataset [134] against PGD attacks [135] compared to CNNs with 0% accuracy. Also, since the voting mechanism is so slow, it takes longer to iteratively generate adversarial images. However, they then refocus their attacks to specifically attack the voting mechanism by maximising Eq. (25) below. With this attack inside the PGD framework, the authors are able to decrease the performance of the network to 4.83%.

$$
\delta^* = \arg \max_{\delta \in \mathcal{N}} \mathcal{H} \left[ \log g \left( \frac{1}{N} \sum_{i=1}^{N} f_i (x + \delta) \right), y \right].
$$
Using this adversarial attack, the authors proceed to train a Capsule Network which is specifically able to resist this kind of attack within the [44] framework which creates a Capsule Network able to maintain 55.3% accuracy of it’s 94.7% accuracy when tested against adversarial PGD attacks.

12 Discussion & Future Directions

Despite the many successes of modern DL-based vision systems [2], [3], [4], a general lack of robustness to distributional shifts remains prevalent [5]. Indeed, unlike current systems, humans are able to quickly adapt to distributional changes using very few examples to learn from [6], [7], [8]. 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 [9], [10], [11]. 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. This inability was one the main motivations behind capsule networks [12]. 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 premises (see Section 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 very inefficient models that are often difficult to train. In the following sections, we provide an in depth discussion on what we believe are the main conceptual considerations for future research in the field.

12.1 The Hardware Lottery

In their detailed analysis, [23] 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 [24] and PyTorch [25] 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 [26], [27], [28], [29], the underlying problem is not only caused by routing but by the capsule voting procedure as well. In their analysis, [23] 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 [30], 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.

To conclude, any 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.

12.2 The Binding Problem

The motivation behind capsule networks is part of an overarching narrative around addressing 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. [5] 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 [136] “In its most general form, 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 [137]. On the other hand, even the most advanced DL systems today struggle with compositionality [138], [139], 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. [5] propose a functional division of the binding problem into three subproblems as depicted in Figure 43. 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 for predictive tasks. Indeed, we believe that overcoming each of these open subproblems may give rise to more robust neural systems which can learn to dynamically represent structured models that generalize more like humans do. Capsule networks are one potential approach for tackling some aspects of the binding problem from a neurosymbolic perspective, 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 an opportunity to extend them to different types of slot representations as shown in Figure 24.

12.3 Limitations & Open Challenges

It remains to be seen whether capsule networks will become the next big thing. They 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. On that note, we believe that developing more flexible frameworks that are better suited to research is a really important avenue for future work. As stressed by [23], although the current machine learning tool chains are extremely powerful and useful to many, there are still concerns regarding the lack of flexibility of languages and backends putting a brake on innovative research in our fast developing field.

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 very successful attention mechanism in Transformers [22], as both methods measure high dimensional coincidences using the agreement between neural activity vectors as a feature detection mechanism. This is generally a good idea since random vectors tend towards orthogonality as the number of dimensions increases. 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 [16]. There is also an opportunity to develop Vision Transformer (ViT) [68] 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 [140] where visual motion is used a 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 using self-supervised learning to uncover 3D parts in much more complex scenes as done in [141]. Improved primary capsule representations could also lead to more effective iterative refinement routing algorithms, since the latter are predicated upon adequate low level part descriptions to work as intended.

To conclude, another aspect of interest for future work is carving out the role of approximately equivariant models like capsule networks in geometric deep learning [36, 93, 142]. 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 known to be only approximately equivariant, there is an opportunity to develop/uncover formal equivariance guarantees which may encourage the usage of these types of models, and mitigate risks of over-reliance on approximate equivariance properties.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 8, pp. 1798–1828, 2013.
[2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” nature, vol. 521, no. 7553, pp. 436–444, 2015.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.
[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[5] K. Greff, S. Van Steenkiste, and J. Schmidhuber, “On the binding problem in artificial neural networks,” arXiv preprint arXiv:2012.05208, 2020.
[6] V. Bruce and G. W. Humphreys, “Recognizing objects and faces,” Visual cognition, vol. 1, no. 2-3, pp. 141–180, 1994.
[7] T. S. Cohen and M. Welling, “Steerable CNNs,” in 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017. [Online]. Available: https://openreview. net/forum?id=rJKY5il
[8] Y. Bengio, Y. Lecun, and G. Hinton, “Deep learning for ai,” Communications of the ACM, vol. 64, no. 7, pp. 58–65, 2021.
[9] I. Rock, Orientation and form. Academic Press, 1973. [Online]. Available: https://books.google.co.uk/books?id=hgQEAQAAIAAJ
[10] G. Hinton, “Some demonstrations of the effects of structural descriptions in mental imagery,” Cognitive Science, vol. 3, no. 3, pp. 231–250, 1979.
[11] D. Kahneman, A. Treisman, and B. J. Gibbs, “The reviewing of object files: Object-specific integration of information,” Cognitive psychology, vol. 24, no. 2, pp. 175–219, 1992.
[12] G. Hinton, “How to represent part-whole hierarchies in a neural network,” arXiv preprint arXiv:2102.12627, 2021.
[13] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming autoencoders,” in International conference on artificial neural networks. Springer, 2011, pp. 44–51.
