UNIVERSITY OF TURIN

DOCTORAL PROGRAM IN COMPUTER SCIENCE
CYCLE XXXV

HIERARCHICAL OBJECT-CENTRIC LEARNING WITH CAPSULE NETWORKS

DOCTORAL DISSERTATION

Candidate:
RICCARDO RENZULLI

Supervisor:
PROF. MARCO GRANGETTO

Reviewers:
PROF. GEORGIOS LEONTIDIS
PROF. PETER SINCAK

NOVEMBER 2023

I hope that people will finally come to realize
that there is only one 'race' - the human race - and
that we are all members of it.

— Margaret Atwood

Dedicated to all the people whose resilience, passion, and unwavering commitment define the collective spirit of pursuing knowledge.

ABSTRACT

Convolutional neural networks have been widely successful in various computer vision tasks. However, they lack an explicit representation of entities, and the loss of spatial information hinders their robustness to distributional shifts, such as object pose variations. Capsule networks were introduced to address these limitations, learning object-centric representations that are more robust, pose-aware, and interpretable. They organize neurons into groups called capsules, where each capsule encodes the instantiation parameters of an object or one of its parts. Moreover, a routing algorithm connects capsules in different layers, thereby capturing hierarchical part-whole relationships in the data.

This thesis investigates the intriguing aspects of capsule networks and focuses on three key questions to unlock their full potential. First, we explore the effectiveness of the routing algorithm, particularly in small-sized networks. We propose a novel method that anneals the number of routing iterations during training, enhancing performance in architectures with fewer parameters.

Secondly, we investigate methods to extract more effective first-layer capsules, also known as primary capsules. By exploiting pruned backbones, we aim to improve computational efficiency by reducing the number of capsules while achieving high generalization. This approach reduces capsule networks memory requirements and computational effort.

Third, we delve into the exploration of part-relationship learning in capsule networks. Through extensive research, we demonstrate that capsules with low entropy can extract more concise and discriminative part-whole relationships compared to traditional capsule networks, even with reasonable network sizes.

Lastly, we showcase how capsule networks can be utilized in real-world applications, including autonomous localization of unmanned aerial vehicles, quaternion-based rotations prediction in synthetic datasets, and lung nodule segmentation in biomedical imaging.

The findings presented in this thesis contribute to a deeper understanding of capsule networks and highlight their potential to address complex computer vision challenges.

PUBLICATIONS

[1] **Renzulli, R.**, Tartaglione, E., Grangetto, M., "Capsule Networks Do Not Need to Model Everything." Submitted to the AAAI Conference on Artificial Intelligence. 2024.

[2] Kinnari, J., **Renzulli, R.**, Verdoja, F., Kyrki, V., "LSVL: Large-scale season-invariant visual localization for UAVs." In: *Robotics and Autonomous Systems* (2023), p. 104497. ISSN: 0921-8890.

[3] **Renzulli, R.**, Grangetto, M., "Towards Efficient Capsule Networks." In: *2022 IEEE International Conference on Image Processing (ICIP)*. 2022, pp. 2801–2805.

[4] **Renzulli, R.**, Tartaglione, E., Fiandrotti, A., Grangetto, M., "Capsule Networks with Routing Annealing." In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Ed. by Igor Farkaš, Paolo Masulli, Sebastian Otte, and Stefan Wermter. 2021, pp. 529–540. ISBN: 978-3-030-86362-3.

[5] Spadaro, G., Tartaglione, E., **Renzulli, R.**, Bragagnolo, A., Giraldo, J. H., Fiandrotti, A., Grangetto, M., "Shannon Strikes Again! Entropy-based Pruning in Deep Neural Networks for Transfer Learning under Extreme Memory and Computation Budgets." Accepted to ICCV 2023 Workshop on Resource Efficient Deep Learning for Computer Vision. 2023.

[6] Di Sario, F., **Renzulli, R.**, Tartaglione, E., Grangetto, M., "Two is Better than One: Achieving High-Quality 3D Scene Modeling with a NeRF Ensemble." Accepted to Image Analysis and Processing (ICIAP). 2023.

[7] Gava, U. A., D'Agata, F., Tartaglione, E., **Renzulli, R.**, Grangetto, M., Bertolino, F., Santonocito, A., Bennink, E., Vaudano, G., Boghi, A., Bergui, M., "Neural network-derived perfusion maps: A model-free approach to computed tomography perfusion in patients with acute ischemic stroke." In: *Frontiers in Neuroinformatics* 17 (2023). ISSN: 1662-5196.

[8] Chaudhry, H. A. H., **Renzulli, R.**, Perlo, D., Santinelli, F., Tibaldi, S., Cristiano, C., Grosso, M., Limerutti, G., Fiandrotti, A., Grangetto, M., Fonio, P., "UniToChest: A Lung Image Dataset for Segmentation of Cancerous Nodules on CT Scans." In: *Image Analysis and Processing – ICIAP 2022*. Ed. by Stan Sclaroff, Cosimo Distante, Marco Leo, Giovanni M. Farinella, and Federico Tombari. Cham: Springer International Publishing, 2022, pp. 185–196. ISBN: 978-3-031-06427-2.

[9] Chaudhry, H. A. H., **Renzulli, R.**, Perlo, D., Santinelli, F., Tibaldi, S., Cristiano, C., Grosso, M., Fiandrotti, A., Lucenteforte, M., Cavagnino, D., "Lung Nodules Segmentation with DeepHealth Toolkit." In: *Image Analysis and Processing. ICIAP 2022 Workshops*. Ed. by Pier Luigi Mazzeo, Emanuele Frontoni, Stan Sclaroff, and Cosimo Distante. Cham: Springer International Publishing, 2022, pp. 487–497. ISBN: 978-3-031-13321-3.

[10] Giordano, L., Gliozzi, V., Pozzato, G. L., **Renzulli, R.,** "RAT-OWL: Reasoning with Rational Closure in Description Logics of Typicality." In: *Joint Proceedings of the 18th Italian Conference on Theoretical Computer Science and the 32nd Italian Conference on Computational Logic*. Ed. by Dario Della Monica, Aniello Murano, Sasha Rubin, and Luigi Sauro. Vol. 1949. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 306–320.

[11] Giordano, L., Gliozzi, V., Pozzato, G. L., **Renzulli, R.,** "An Efficient Reasoner for Description Logics of Typicality and Rational Closure." In: *Proceedings of the 30th International Workshop on Description Logics*. Ed. by Alessandro Artale, Birte Glimm, and Roman Kontchakov. Vol. 1879. CEUR Workshop Proceedings. CEUR-WS.org, 2017.

ACKNOWLEDGMENTS

I want to thank all those who have supported me in the past few years during the journey that led to this dissertation.

I extend my deepest gratitude to my advisor, Marco Grangetto, whose unwavering support, guidance, and expertise have been instrumental throughout this PhD journey. I would like to give a special thanks to Enzo Tartaglione, your mentorship has been invaluable, and I am truly grateful for the countless hours you dedicated to shaping and refining my research.

I would like to thank my defence committee members, Georgios Leontidis, Peter Sincak, Attilio Fiandrotti and Lia Morra, for their insightful feedback and constructive criticism. Your expertise has enriched the quality of this work, and I am honoured to have had your guidance.

I am grateful for the research visiting period at the Intelligent Robotics group at Aalto University, especially for the support of Ville Kyrki, Francesco Verdoja and Jouko Kinnari, who enabled me to enrich my skills and research activities essential for this thesis.

Special thanks to all the EIDOS group members and colleagues of the Department of Computer Science at the University of Turin, in particular to Andrea Bragagnolo, Carlo Alberto Barbano, Giovanni Bonetta, Gabriele Sartor, Mirko Zaffaroni, Daniele Perlo, Alberto Presta, Gabriele Spadaro, Francesco Di Sario and Gianluca Dalmasso for providing a conducive and stimulating academic environment. The exchange of ideas and the camaraderie made the research process both enriching and enjoyable.

Heartfelt thanks to my friends Alvise, Alessio, Clarissa, Dorian, Elena, Joana, Marco and Viola for their unwavering encouragement and understanding during the challenging phases of this academic pursuit.

I extend my deepest thanks to my family, who believed in me from the beginning. Your belief has been a constant source of motivation.

Thanks to Dario, my biggest supporter, my compass, always there to help me even when I was sinking into the deepest part of the ocean.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Convolutional networks lose the spatial relationships between their parts because of max pooling layers, which progressively drop spatial information [1]. Furthermore, convolutional networks do not have an explicit representation of an entity. Capsule networks were introduced by Sabour et al. [1] to overcome the shortcomings of convolutional networks. Capsule networks group neurons into *capsules* that encode objects and objects-parts of the entities in the input images. Therefore, they learn hierarchical structures from data. Thanks to a routing algorithm that connects capsules in different layers, we can carve out a parse tree composed of part-object relationships from the network. They are also more robust to viewpoint changes and affine transformations than traditional convolutional networks.

Despite their potential advantages, capsule networks have not yet seen widespread adoption in the industry or research community compared to traditional convolutional networks or recently introduced vision transformers [2, 3]. The lack of standardized architectures and pre-trained models might be a contributing factor. However, continued research and advancements in capsule networks are essential to harness their potential and address their limitations fully.

In this thesis, we delve into the intriguing aspects of capsule networks, focusing on three key questions that aim to unlock their full potential:

- *Does the routing algorithm really improve the performance of capsule networks?*
- *How can capsule networks learn effective low-level part descriptions?*
- *Can capsules achieve high generalization ability even when not accounting for the clutter?*

The first question addressed in this thesis revolves around the effectiveness of the routing algorithm. While previous research has demonstrated the robustness of capsule networks in various tasks, questions remain regarding the necessity of the routing-by-agreement mechanism, especially in small-sized networks. Through rigorous experimentation, we affirm the importance of the routing algorithm in small-sized networks. We propose a novel method that anneals the number of routing iterations during training, striking better performance, especially for architectures with fewer parameters.

The second question explores methods to extract better first-layer capsules, often referred to as primary capsules. Convolutional layers without pooling have been a common choice in capsule networks, but they might lead to information loss and limited feature representation. To address this concern, we investigate the exploitation of pruned backbones. Promoting sparse topologies, for example, allows the deployment of deep neural networks models on embedded, resource-constrained devices. We explore sparsity besides capsule representations to improve their computational efficiency by reducing the number of capsules. By leveraging this approach, we show how pruning with capsule network achieves high generalization with less memory requirements, computational effort, and inference and training time.

To answer the third question, we delve into the exploration of part-relationship learning in capsule networks. Understanding how capsules capture part-whole relationships is crucial in advancing our comprehension of capsule network dynamics. In this research, we demonstrate that capsules with low entropy hold the potential to extract more succinct and more discriminative part-relationships than traditional capsule networks, even with reasonable network sizes. These findings shed light on the underlying mechanisms of part-aware representations and pave the way for more interpretable and effective capsule networks.

This thesis will focus on answering the previous questions, also showing how we can exploit capsule networks ability to capture spatial relationships between features in applications such as autonomous localization and biomedical imaging.

This thesis is organized as follows:

- in Part i we deepen the limitations of convolutional networks, providing an overview of the fundamentals on capsule networks.

- in Part ii we tackle the above open questions proposing three methods: routing annealing, efficient capsule networks with pruned backbones, and routing entropy minimization.

- in Part iii we show how capsule networks can be applied to more difficult tasks: localization of autonomous unmanned aerial vehicles, quaternion-based rotations prediction in synthetic datasets, and segmentation of lung nodules on medical images.

Part I

LEARNING WITH CAPSULE NETWORKS

This part discusses the drawbacks of convolutional neural networks in image and object recognition, such as the use of pooling layers, which can lead to the loss of spatial relationships between objects that reduce the robustness of a model to novel viewpoints and affine transformations of the input. Then we introduce capsule networks, a novel deep learning architecture proposed by Hinton, which attempts to solve the problems with these state-of-the-art methods grouping neurons into vectors called capsules.

2 THE LIMITATIONS OF CONVOLUTIONAL NETWORKS

CNNs have been remarkably successful in the last few years and they outperform other methods on object recognition tasks. Their main characteristics are:

1. They use many learned feature detectors with different receptive fields.
2. Different types of feature detectors are replicated across space.
3. Higher-level feature detector fields have bigger spatial dimensions.
4. Convolutional layers are interleaved with pooling layers that downsample the input. They combine the outputs of nearby feature detectors neurons of the same type.

Replicated feature detectors, inspired by the human vision system, allow recognition of similar textures, shapes and objects in different locations in the image. Pooling layers give a small amount of translation invariance at each level and reduce the number of neurons to the next feature extraction layer. Therefore, CNNs learn multiple types of features with bigger spatial domains.

However, Hinton et al. [4] argue that pooling is a very inefficient way of learning. The four arguments against pooling are:

1. It does not fit the psychological reality of part-whole hierarchies and coordinate frames in human vision.
2. It is better to ask for invariance in the weights and equivariance in the activities.
3. It fails to use the underlying linear manifold that handles natural image changes.
4. Pooling is a primitive way to route information to the next layers.

We will deepen these arguments in the following sections.

2.1 COORDINATE FRAMES

According to the recognition-by-components theory illustrated by Biederman [5], the "preferred" mode of human objects recognition is to separate them into simple 3D shapes, called geons (blocks, cylinders, wedges, and cones), which are the parts that compose an object.

Biederman [5] suggested that geons can be assembled in various arrangements to form a virtually unlimited number of objects, as shown in Figure 2.1. Additionally, Hinton [6] presented mental imagery tasks

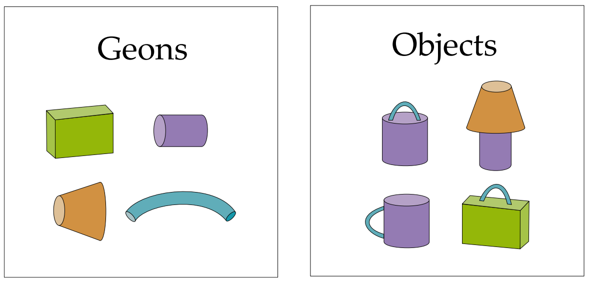


Figure 2.1: Different arrangements of the same components can produce different objects.

and perceptual demonstrations to illustrate the involvement of structural descriptions in the imagination and perception of 3D spatial structures. These descriptions are hierarchically organized: complex objects are assigned hierarchical structural descriptions by parsing into parts, each with a local system of significant directions.

Psychological evidence shows that our visual systems impose coordinate frames for shape recognition [6–9]. Figure 2.2 shows how coordinate frames are a fundamental aspect of human perception. We use as an example the rabbit–duck illusion, the famous ambiguous drawing of the German humor magazine Fliegende Blätter (1892), in which a rabbit or a duck can be seen. The percept of them is utterly different depending on which way they are seen. We can explain how it is possible that people can see both animals using coordinate frames. So, with the additional information that the front of the animal points from the center of the page to the left, the reader can recognize a duck. On the contrary, if the additional information is that the front points from the center of the page to the right, the reader can now see the resemblance with a rabbit instead. Therefore, we get a completely different internal percept depending on what coordinate frame we impose. However, CNNs recognize objects in a very different way from people [4]. Given an input image, they have only one percept for it, and this percept does not depend on imposing different coor-

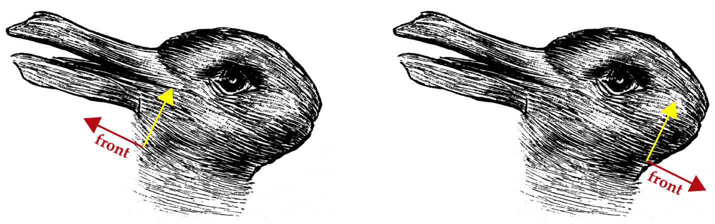


Figure 2.2: The same object will look different depending upon the coordinate frame imposed (a duck if the front points to the left, a rabbit if the front points to the right).

dinate frames. So CNNs can not explain how the same pixels can be perceived differently.

These theories lead to the idea that our brain does a sort of *inverse computer graphics transformation* to perceive shapes [1]. Computer vision is inverse computer graphics, so the higher levels of a vision system should look like the representations used in graphics. Graphics programs use *hierarchical models* in which viewpoint invariant matrices model spatial structure. These matrices represent the transformation from a *coordinate frame* (or coordinate system) embedded in the whole to a coordinate frame embedded in each part. Hierarchical modeling creates complex real-world objects by combining simple primitive shapes into more complex aggregate objects. Adopting this representation, it is easy to compute the relationship between a part and the retina from the relationship between a whole and the retina. Let's see a more formal definition.

We refer to the *position* and *orientation* of an object as its *pose*. Every time we describe the pose of an object, we must specify its relation to some coordinate frame. The pose can be described for example through a rotation and translation transformation, which brings the object from a reference pose to the observed pose. *Affine transformations* are geometric transformations that preserve collinearity; namely, they map points to points, lines to lines and planes to planes, but not the origin of the space. An affine transformation f acting on a vector $\boldsymbol{x}$ can be represented as the composition of a *linear transformation* $\boldsymbol{L}$ (a mapping that preserves the operation of addition and scalar multiplication) and a *translation* by a vector $\boldsymbol{b}$:

$$f(\boldsymbol{x}) = \boldsymbol{L}\boldsymbol{x} + \boldsymbol{b}. \tag{2.1}$$

Examples of linear transformations are reflections, scalings, rotations and shearing. Because matrix multiplication is associative, we can represent the composition of a sequence of linear transformations more efficiently with a single matrix. Therefore, we can encapsulate and store complex transforms in a compact and convenient form. However, since translation is not a linear transformation, we need a different representation. We can represent affine transformations in a fully matrix-multiplication form using *homogeneous coordinates* to determine a frame (4D vector $[p_x, p_y, p_z, p_0]^\top$). With this notation, any point P can be written uniquely as $P = \alpha_x p_x + \alpha_y p_y + \alpha_z p_z + p_0$. The standard convetion sets p_0 to 1. Then any sequence of such operations can be multiplied into a single 4x4 matrix, allowing simple and efficient processing. The general form of a transformation matrix is defined as:

$$\boldsymbol{T} = \begin{bmatrix} n_x & o_x & a_x & v_x \\ n_y & o_y & a_y & v_y \\ n_z & o_z & a_z & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.2}$$

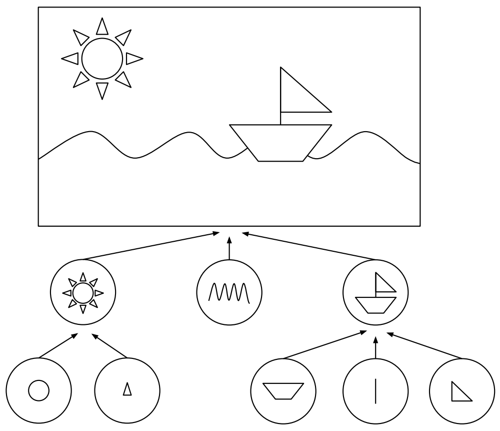


Figure 2.3: Example of a scene graph.

where the upper $3x3$ matrix defines the linear transformation and the last column the translation. Note that the columns n, o and a of the transform matrix specify the coordinates of the x, y and z axes of the new coordinate frame relative to the old frame; namely, they are unit vectors. The third column is the origin of the new coordinate frame concerning the previous frame. For example, in computer graphics terms, we say that $\boldsymbol{T}$ converts points from the *object frame* to the *camera frame*. The inverse of $\boldsymbol{T}$ takes us from the *camera frame* to the *object frame*. These transformations play an important role in hierarchical modeling in computer vision. As mentioned before, complex objects are made up of simpler objects, and simpler objects are made up of geometric primitives that can be drawn directly. We can think of these objects as if each has its coordinate system. The components of a complex scene (for example, an image) can be represented using a tree-like structure called *scene graph* (see Figure 2.3). Each node can contain any number of children and represents either an object or a part of an object. The root represents the entire scene or the world. Each connection represents one occurrence of the object in its parent object, and it can be associated with a transformation.

Therefore, our vision system use rectangular coordinate frames embedded in objects and it has the notion of a scene graph that specifies how each node is related to the viewer, namely an *hierarchy of parts* exists. CNNs do not use coordinate frames. This is why they do not explain why we assign intrinsic coordinate frames to objects and why they have many effects on our perception, namely using a different frame totally changes the percept.

2.2 INVARIANCE AND EQUIVARIANCE

CNNs try to make the neural activities *invariant* to small transformations thanks to pooling layers (see Figure 2.4). This is motivated by the fact that in certain tasks, such as image classification, the final label needs to be viewpoint invariant. As shown in Figure 2.4, CNNs use multiple neurons or feature detectors to detect different variants of the features, namely CNNs realize the face matches a variant that is rotated 45 degrees.

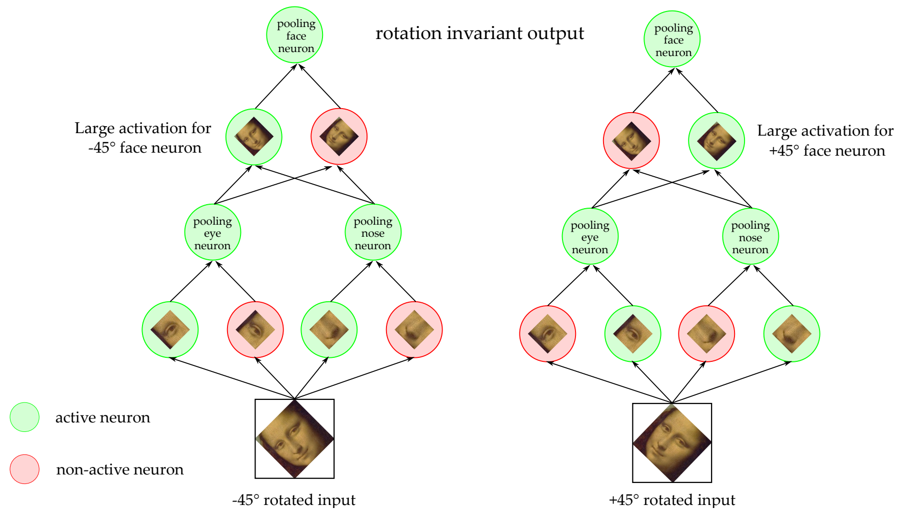


Figure 2.4: Max-pooling over spatial regions produces invariance to translation but not to other transformations, such as rotations. If we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant. Here we show how a CNN can learn rotation invariances thanks to pooling layers applied to many feature detectors.

However, when we look at a face, we do not merely recognize it, but we also know its position, orientation, and scale precisely. With the pooling operation, we loose such information, and it is impossible to compute precise spatial relationships. In other words, we would like a model that detects that the face is rotated 45 degrees thanks to corresponding changes in its activations. This way of learning representations enables the development of deep learning models that require fewer trainable parameters and less training data, also avoiding exhaustive data augmentation with all possible transformations and viewpoints.

Hinton et al. [4] claim that it is better to aim for *equivariant* neural activities: we want representations where changes in viewpoint lead to corresponding (smoothly) changes in neural activities. There are some symmetries in our data, for example, in medical images (Figure 2.5), and building a model that retains the input symmetries in its outputs can be helpful. Therefore, generating equivariant embeddings can help improve the accuracy, robustness, and generalization of the neural network's predictions. This is effective for a broader range of applications, including supervised classification [10, 11] and segmentation [12, 13] tasks, and self-supervised learning pre-training procedures [14–16], where sensitivity to transformations is a desirable property. Furthermore, cognitive scientists discovered that in the in-

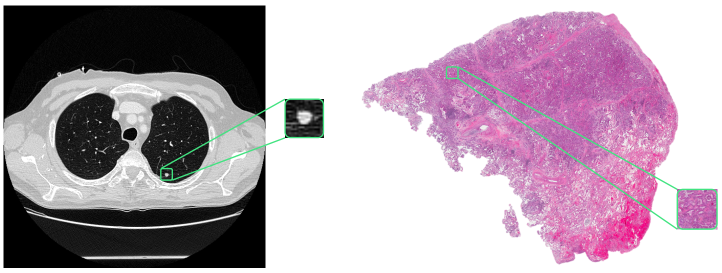


Figure 2.5: Lung nodules in CT scans (left) and histopathology slices (right) have translation, rotation, reflection, and scaling symmetries.

ferior temporal cortex, the same neurons are selective for edges in different orientations, scales, positions, colors, textures, or even for more complex shapes [17, 18]. More formally, we say that a function f is

- *invariant* to a transformation $\boldsymbol{T}$ if $f(\boldsymbol{T}(\boldsymbol{x})) = f(\boldsymbol{x})$. If the input changes, the output does not change.
- *equivariant* to a transformation $\boldsymbol{T}$ if $f(\boldsymbol{T}(\boldsymbol{x})) = \boldsymbol{T}(f(\boldsymbol{x}))$. If the input changes, the output changes in a coordinated way.

Invariance is a special case of equivariance when $\boldsymbol{T}$ is the identity function. Figure 2.6 shows how rotation invariance and equivariance work. CNNs are not inherently designed to recognize the same objects at different rotations without relying on data augmentations and many feature detectors.

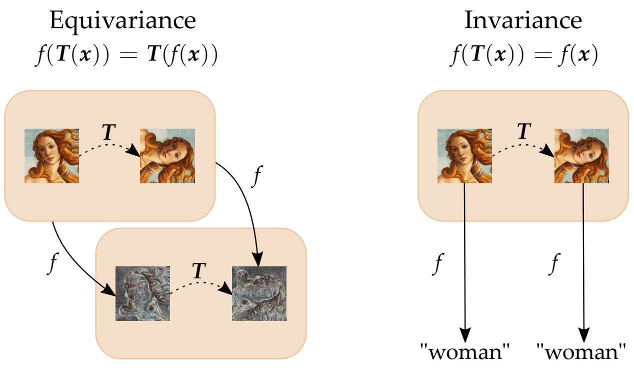

Equivariance
$f(\boldsymbol{T}(\boldsymbol{x})) = \boldsymbol{T}(f(\boldsymbol{x}))$
$\boldsymbol{T}$
f
f
$\boldsymbol{T}$
Invariance
$f(\boldsymbol{T}(\boldsymbol{x})) = f(\boldsymbol{x})$
$\boldsymbol{T}$
f
f
"woman"
"woman"

Figure 2.6: Example of invariance and equivariance when $\boldsymbol{T}$ model a +90° rotation transformation and f is the Sobel filter.

Sabour et al. [1] distinguishes between two types of equivariance.

1. PLACE-CODED: A discrete change in a property of a visual entity leads to a discrete change in which neurons are used for encoding that visual entity.
2. RATE-CODED: A real-valued change in a property of a visual entity leads to a real-valued change in the output of some of

the neurons used for coding that visual entity, but there is no change in which neurons are used.

In the case of convolution, the particular form of parameter sharing causes the layer to be naturally place-coded equivariant only to translation [19]. (see Figure 2.7). A way to achieve rate-coded equivariance

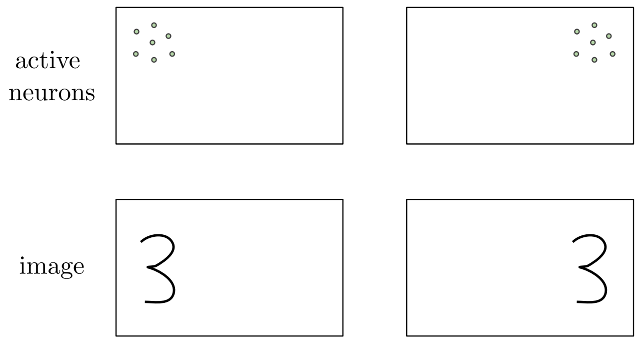


Figure 2.7: Example of place-coded equivariance when translating an image in a CNN.

is to use a group of neurons that do quite a lot of internal computation (using non-linear *recognition units*) and encapsulate the results of this computation into a low dimensional output [4]. At low levels, there are small domains, so if a part moves to a very different position, it will be represented by a different group of neurons (place-coded equivariance). At higher levels, we have bigger domains so if a part only moves a small distance, it will be encoded by the same group, but its activations will change (rate-coded equivariance). Therefore, we would like that in deep learning models, like in the perceptual system, viewpoint-invariant knowledge is embedded in the weights, not in the neural activities, which, in contrast, encode the equivariance property.

2.3 LINEAR MANIFOLD

If humans are presented with an image from a viewpoint different from the usual one, we do not have difficulty recognizing the object. Therefore, unlike CNNs, we do not have to see the same object in a thousand different poses to learn to recognize it. A model is needed that learns the relationships between the parts so that it is viewpoint invariant. CNNs with pooling are not built on the *manifold hypothesis*. A *manifold* is a set a of points associated with a neighborhood around each point and from any given point it locally appears to be an Euclidean space [19]. According to the manifold hypothesis, valid high-dimensional input data (such as images) tend to lie near low-dimensional manifolds. The concept of neighborhood surrounding each point implies the existence of linear transformations that can be applied to move on the manifold from one point to a neighboring one.

For example, in image space we can move on the manifold changing the lights or rotating the objects.

CNNs do not take advantage of this linear natural manifold that handles the largest source of variance in images. In order to achieve high accuracy in shape recognition, they learn different models for different viewpoints and this requires a lot of training data, for example employing data augmentation methods.

We know that the manifold of images of the same rigid shape is highly-non linear in the space of pixel intensities. As shown in Figure 2.8 we want to transform pixel intensities into a space in which the manifold is globally linear, just like the graphical representation that uses explicit pose coordinates. Therefore, if we can get pose coordinates from raw pixels, we can do massive extrapolation of new poses from this linear manifold by applying linear transformations. This lead to better generalization over novel viewpoints, training our network on fewer input data. Generalization is achieved using the

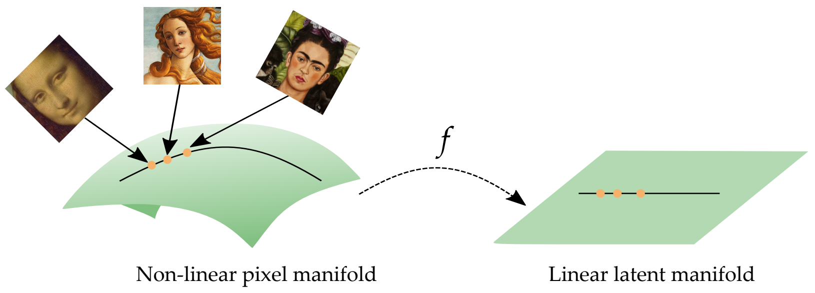
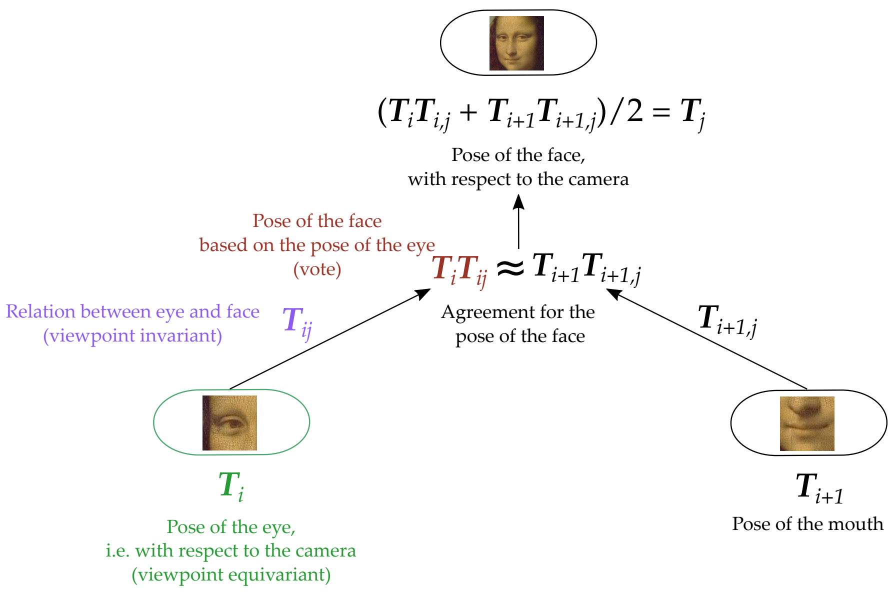


Figure 2.8: Locally near high-dimensional images lies in a non-linear manifold. If we map their parts into low-dimensional embeddings in a linear manifold, we can exploit linear transformations as in computer graphics to generalize on novel viewpoints.

(linear) invariant transformation matrices $\boldsymbol{T}$ described in Section 2.1. It does not matter how much the pose changes, it is the same matrix of weights that takes the pose of the part and gives the pose of the whole, or vice versa. Therefore, with inverse graphics, we mean that from raw pixels we obtain the poses of the parts (encoded in a bunch of neurons), we multiply these poses by $\boldsymbol{T}$ to get the pose of the whole. But can a deep learning model detect objects exploiting this linear manifold? In Figure 2.9 we can see how a coincidence filtering mechanism using the linear manifold can detect familiar objects in a hierarchy of parts. Each part makes a *vote* for the object: how the pose of the object will be based on the pose of that part. Then, an object can be detected by looking for agreement between these votes for its pose matrix, computed for example with the mean of these predictions. Finding these agreements is not easy when there are many parts and objects. We need a more complex operation than pooling over the lower activations.



$(T_i T_{i,j} + T_{i+1} T_{i+1,j})/2 = T_j$
Pose of the face, with respect to the camera
Pose of the face based on the pose of the eye (vote)
$T_i T_{ij} \approx T_{i+1} T_{i+1,j}$
Agreement for the pose of the face
Relation between eye and face (viewpoint invariant)
T_{ij}
$T_{i+1,j}$
T_i
Pose of the eye, i.e. with respect to the camera (viewpoint equivariant)
T_{i+1}
Pose of the mouth

Figure 2.9: Coincidence filtering in a two layers hierarchy of parts. Example of how to compute the pose of a face (i.e., with respect to the camera) from the poses of an eye and a mouth. There is an agreement between the predictions of the poses of the eye and the mouth for the pose of the face, so a face is detected.

2.4 ROUTING

As mentioned in Section 2.1, given an input image, we would like to carve out a scene graph. Therefore, parts of objects in one layer need to choose which object in the layer above they belong to. This process of part-object relationship extraction is called routing. It is based on the coincidence filtering mechanism when there are many parts and objects.

CNNs do a basic primitive routing through max-pooling: the pooling unit selects the most active neuron only based on its magnitude. For complicated shapes, replicating the knowledge across all locations with a small stride is not a good solution. A good routing principle is to send the information about a specific part to the object in the layer above that can account more for it. Routing, as shown in Figure 2.10, naturally resembles clustering: a higher-layer parent object j (cluster) comprises many lower-layer child parts predictions. A higher-level object asks for more feedback from lower-level parts that vote for its cluster and less feedback from lower-level parts that vote for its outliers. Finding the best routing is equivalent to parsing the image to obtain a parse tree where each node has only one parent [1]. The total weight on the bottom-up inputs supplied by one lower-level part is 1 to satisfy this "single parent" constraint. The parse tree representations are similar to scene graphs in computer graphics. So how can we detect objects without pooling? We can think of this process as an

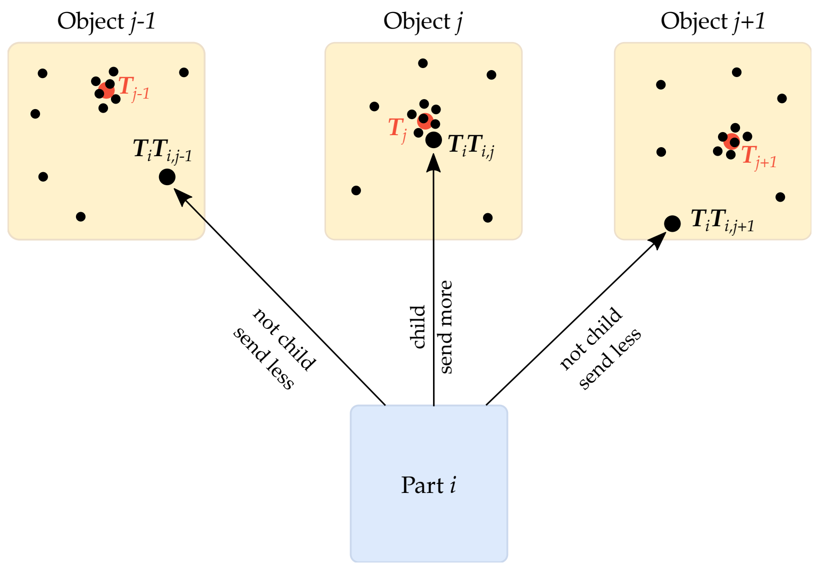


Figure 2.10: Routing naturally resembles clustering. Each dot is a lower-level part prediction for a given higher-layer object. We want to send more feedback to parts that agree more, namely those in the same cluster. The final pose of a high-level object will be the centroid of this cluster.

assembly mechanism that takes as input lower-level parts and outputs higher-level objects. This process will be deepened in Chapter 3.

3 BACKGROUND ON CAPSULE NETWORKS

3.1 NOTATION

This Section reports the notation used in the following Chapters for capsule networks. Superscript (ξ) will denote the ξ^{th} input example while superscript $[l]$ will denote the l^{th} capsule layer.

- Ξ input dataset
- $\boldsymbol{x}^{(\xi)} \in \Xi$ the ξ^{th} input example represented as a feature vector
- Y label dataset
- $y^{(\xi)} \in Y$ the corret label for the ξ^{th} example
- $\hat{y}^{(\xi)} \in Y$ the predicted label for the ξ^{th} example
- Γ label set
- $\gamma \in \Gamma$ generic label
- $\boldsymbol{T} \in \mathbb{R}^{4\times 4}$ generic affine transformation
- $\boldsymbol{T}_{i,j}$ generic affine transformation between part i and object j
- L number of capsule layers in the network
- $\Omega^{[l]}$ set of capsules in layer $l \in \{1, 2, ..., L\}$
- $|\Omega^{[1]}|$ number of primary capsules
- $|\Omega^{[L]}|$ number of class capsules

  When $L = 2$, it is possible to denote $|\Omega^{[1]}| = I$ and $|\Omega^{[L]}| = J$
- $a \in \mathbb{R}$ generic activation of a capsule
- $\boldsymbol{u} \in \mathbb{R}^{D}$ generic D-dimensional feature vector of a capsule
- $a^{[l]}$ generic activation of a capsule in layer l
- $\boldsymbol{u}^{[l]} \in \mathbb{R}^{D^{[l]}}$ generic $D^{[l]}$-dimensional feature vector of a capsule in layer l
- $b_{i,j}^{[l]}$ log prior probability that capsule i in layer l should be coupled to capsule j in layer $l+1$
- $\boldsymbol{U}^{[l]}$ matrix containing all feature vectors of all capsules in layer l

- $\boldsymbol{C}^{[l]}$ matrix containing coupling coefficients between capsules in layer l and capsules in layer $l+1$
- $c_{i,j}^{[l]}$ coupling coefficient between capsule i in layer l and capsule j in layer $l+1$
- $c_{i,j}^{[l](\xi)}$ coupling coefficient between capsule i in layer l and capsule j in layer $l+1$ for the ξ^{th} example
- $\tilde{c}_{i,j}^{[l]}$ quantized coupling coefficient between capsule i in layer l and capsule j in layer $l+1$
- $\boldsymbol{W}_{i,j}^{[l]} \in \mathbb{R}^{D^{[l]} \times D^{[l+1]}}$ trainable weight matrix between a capsule i in layer l and a capsule j in layer $l+1$
- $\hat{\boldsymbol{U}}^{[l+1]} \in \mathbb{R}^{|\Omega^{[l]}| \times |\Omega^{[l+1]}| \times D^{[l+1]}}$ contains all the votes from capsules in layer l for capsules in layer $l+1$
- $\hat{\boldsymbol{u}}_{i,j}^{[l+1]} \in \mathbb{R}^{D^{[l+1]}}$ vote of a capsule i in layer l for a capsule j in layer $l+1$
- $r^{[l]}$ number of routing iterations between capsule layers l and $l+1$

When there are only two capsule layers, we can omit the l notation since i is used for lower-level capsules and j for higher-level capsules. In this scenario, we also use the index j to denote the j-th class label. For abuse of notation, we often suppress the index ξ.

3.2 WHAT ARE CAPSULES

In Chapter 2 we discussed some of the limitations of CNNs, emphasizing which are the main characteristics an artificial neural system should hold to perceive shapes as humans, using less data and parameters. We also introduced the notion of parts/children and objects/-parents when parsing an image and the relationships between them. How can we encode this idea directly into the architecture of an ANN? In standard neural networks, there is no explicit notion of an entity.

In 1990, a first attempt to embed structural relations into a general model of object recognition inspired by inverse computer graphics was pursued by Zemel et al. [20]. The concept of a capsule was introduced later in 2011 [4], where a capsule is defined as a group of neurons within a layer.

Capsules are neurobiologically inspired by the structure of *cortical minicolumns* discovered by Mountcastle in 1957, a group of neurons in the cortex of the brain that encode similar features. Cells in 50 μm minicolumn all have the same receptive field; adjacent minicolumns may have different fields. Cortical minicolumns contains from 80 to

100 neurons and they are organized in *macrocolumns*. There are nearly 100 minicolumns in a macrocolumn.

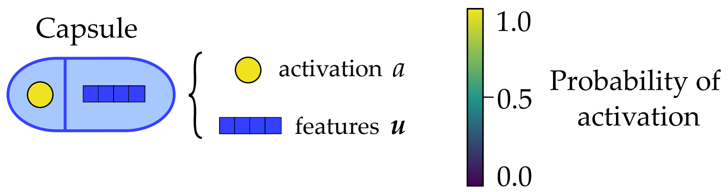


Figure 3.1: Capsule components..

Parts and objects are indeed visual entities. Each capsule learns to recognize an implicitly defined visual entity over a limited domain of viewing conditions and deformations [4]. The neural network will learn what the entities are and how they interact with each other. A Capsule Network (CapsNet) consists of several layers l of capsules, where $l \in \{1, 2, ..., L\}$. The set of capsules in layer l is denoted as $\Omega^{[l]}$. We will refer to parts as lower-layer capsules $i \in \Omega^{[l]}$ and to objects as higher-layer capsules $j \in \Omega^{[l+1]}$. For classification tasks, the number $|\Omega^{[L]}|$ of capsules in the last layer is equal to $|\Gamma|$ output labels.

A capsule in layer l is a set of neurons $\{a^{[l]}, \boldsymbol{u}^{[l]}\}$ where

1. $a^{[l]} \in \mathbb{R}$ is the activation of the capsule. It is the probability that an object of that type is present;

2. $\boldsymbol{u}^{[l]} \in \mathbb{R}^{D^{[l]}}$ is the feature vector of the capsule. It encodes the instantiation properties of a multi-dimensional entity of the type that the capsule detects. These properties can include its pose (position, size, orientation), deformation, velocity, albedo, hue, texture, etc.

The matrix $\boldsymbol{U}^{l} \in \mathbb{R}^{|\Omega^{[l]}| \times D^{[l]}}$ contains all $D^{[l]}$-dimensional capsule vectors in layer l, the vector $\boldsymbol{a}^{[l]}$ holds the corresponding activations.

As mentioned in Section 2.4, assembling parts to build objects can be divided into two phases: a voting procedure and a routing procedure. Similarly, CapsNets use this assemble module to compute higher-layer capsules j from lower-layer capsules i based on coincidence filtering. Figure 3.2 gives an overview of this module. Most capsule layers in the literature share the same voting module and they differ mainly on how the agreements are found in the routing module.

High-dimensional coincidences do not happen by chance: a higher-order capsule receives multi-dimensional predictions from capsules in the layer below on how its pose should be and it looks for a tight cluster of predictions. If it finds such cluster, namely the predictions are in strong agreement, then it outputs:

1. A high probability $a_j^{[l+1]}$ that an entity of its type exists in its domain. We say that the low-level capsules of the cluster are in the right spatial relationship to activate the high-level capsule j.

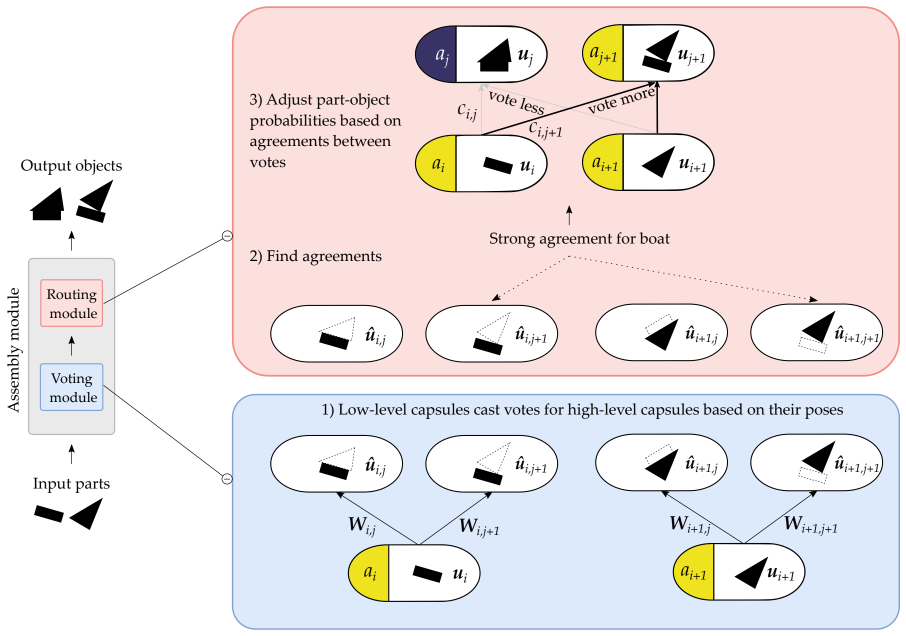

Output objects
Assembly module
Routing module
Voting module
Input parts
3) Adjust part-object probabilities based on agreements between votes
vote less
vote more
$c_{i,j}$
$c_{i,j+1}$
a_j u_j
a_{j+1} u_{j+1}
a_i u_i
a_{i+1} u_{i+1}
Strong agreement for boat
2) Find agreements
$\hat{u}_{i,j}$ $\hat{u}_{i,j+1}$ $\hat{u}_{i+1,j}$ $\hat{u}_{i+1,j+1}$
1) Low-level capsules cast votes for high-level capsules based on their poses
$W_{i,j}$ $W_{i,j+1}$ $W_{i+1,j}$ $W_{i+1,j+1}$

Figure 3.2: The assembly module comprises two sub-modules: 1) the votes module, which computes the object-capsules predictions from the input part-capsules, and 2) the routing module, which adjusts the part-object connections based on the agreements between these votes to compute the output capsules.

2. The center of gravity of the cluster, which represents the instantiation parameters $\boldsymbol{u}_j^{[l+1]} \in \mathbb{R}^{D^{[l+1]}}$ of the high-level capsule j.

The prediction $\hat{\boldsymbol{u}}_{i,j}^{[l+1]}$, or vote, of a capsule i in a layer l for a capsule j in layer $l+1$ is produced by multiplying its features $\boldsymbol{u}_i^{[l]}$ by a weight matrix $\boldsymbol{W}_{i,j}^{[l]} \in \mathbb{R}^{D^{[l]} \times D^{[l+1]}}$:

$$\hat{\boldsymbol{u}}_{i,j}^{[l+1]} = \boldsymbol{W}_{i,j}^{[l]} \boldsymbol{u}_i^{[l]}. \tag{3.1}$$

The matrix $\hat{\boldsymbol{U}}^{[l+1]} \in \mathbb{R}^{|\Omega^{[l]}| \times |\Omega^{[l+1]}| \times D^{[l+1]}}$ contains all the votes of two consecutive capsule layers. In fully-connected or dense capsule layers there is a different weight matrix for each possible pair (i, j). We learn them through backpropagation and we hope that they will be the invariant viewpoint transformation matrices $\boldsymbol{T}_{i,j}$ seen in computer graphics. As the viewpoint changes, the poses of the parts and the whole will change in a coordinated way so that any agreement between votes from different parts will persist [1]. As we mentioned in Section 2.2, a capsule works properly when its activation is locally invariant, and the instantiation parameters are equivariant. The part-whole relations are modeled by the coupling coefficients $c_{i,j}^{[l]} \in \mathbb{R}$, which are dependent on the input and not learned during backpropagation. After the voting procedure, we need to route the votes to

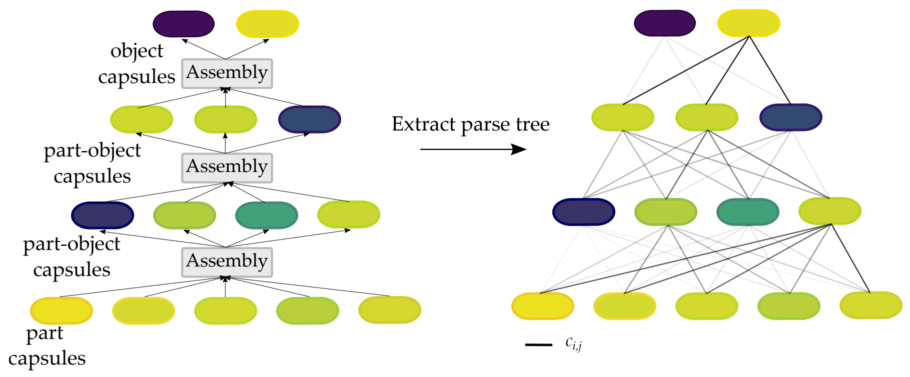


Figure 3.3: Stacking multiple assembly modules on top of each other create a deeper hierarchy of objects. For a given input image, we can extract a parse tree where each node is a part or object capsule, and each connection is the probability of the part-whole (or child-parent) relation.

higher-layers to output object-capsules $\boldsymbol{u}_j^{[l+1]}$ and adjust their connection $c_{i,j}^{[l]}$ based on the agreements. A "routing softmax" determines the coupling coefficients (single-parent constraint), whose initial logits $b_{i,j}^{[l]}$ are the log prior probabilities that capsule i should be coupled to capsule j

$$c_{i,j}^{[l]} = \sigma(\boldsymbol{b}_{i,:}^{[l]})_j = \frac{e^{b_{i,j}^{[l]}}}{\sum_{k=1}^{|\Omega^{[l+1]}|} e^{b_{i,k}^{[l]}}}, \qquad \sum_{j=1}^{|\Omega^{[l+1]}|} c_{i,j}^{[l]} = 1. \tag{3.2}$$

During the routing procedure, each $c_{i,j}^{[l]}$ is refined by measuring the agreement between the output $\boldsymbol{u}_j^{[l]}$ of a capsule j and the prediction $\hat{\boldsymbol{u}}_{i,j}^{[l+1]}$ Therefore, routing finds clusters of agreements between the predictions to solve the problem of assigning parts to wholes [1]. Routing can be viewed as a parallel *attention* mechanism that allows each capsule at one level to attend to some active capsules at the level below and to ignore others [1]. It is an attention mechanism in the opposite direction as the competition is between the higher-level capsules that a lower-level capsule might send its vote to.

At the end of the routing process, we extract a parse tree of the image using the coupling coefficients. Since we are dealing with deep learning models, the parse tree representations approximate the scene graphs in computer graphics. Figure 3.3 shows an example of a parse tree for a given input image.

Capsules can be implemented in many different ways. In the next sections we will explore different implementations of capsules and routing.

3.3 CAPSULE NETWORKS FUNDAMENTALS

This section gives an overview of a general architecture of a CapsNet. Then, we review the first routing algorithm proposed by Sabour et al. [1] and other routing implementations.

3.4 GENERAL ARCHITECTURE

A CapsNet is composed of an encoder and a decoder part. The encoder consists of several capsule layers l stacked on top of a convolutional backbone. There are three groups of capsule layers: primary capsules (PrimaryCaps, the first one, built upon convolutional layers), convolutional capsules (ConvCaps, each capsule has its own receptive field), and fully-connected capsules (FcCaps, each capsule is connected with all the capsules in the previous layer). Typically, a CapsNet comprises at least two capsule layers, PrimaryCaps and FcCaps (also called ClassCaps when it is the last layer, with one output capsule for each object class). The decoder network is used to reconstruct the input image from the capsule of the target class (during training) or with highest activation (during inference) and it is used as a regularization that helps the network to encode better the objects instantiation features. However, many CapsNets implementations rely only on the encoder part since the decoder improves performances mostly on simple datasets. Figure 3.4 shows an example of a general CapsNet architecture. We can see that primary and convolutional capsules are organized in three dimensions and located in a $M^{[l]} \times N^{[l]} \times O^{[l]}$ grid structure, where $O^{[l]}$ is the number of capsule types. Therefore, in this scenario, $|\Omega^{[l]}| = M^{[l]} \times N^{[l]} \times O^{[l]}$. As shown in Table 3.1, the

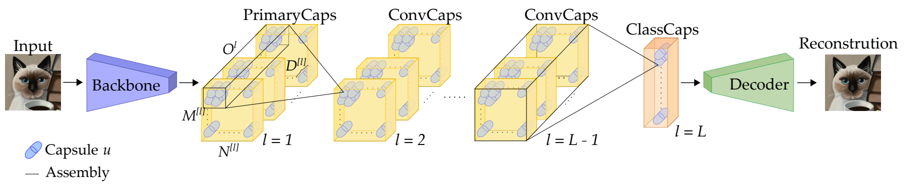


Figure 3.4: An example of a general CapsNet architecture composed by a convolutional backbone, a PrimaryCaps layer (which is a convolutional layer with squash activation and then reshaping), several ConvCaps layers (where a capsule in layer l is computed using only a subset of capsules in layer $l-1$), and a ClassCaps (or FcCaps, where each capsule in layer l is computed using all the capsules in layer $l-1$) layer. A decoder is used to reconstruct the input image.

dimensions of the weight matrix $\boldsymbol{W}^l$ between capsules in layer l and capsules in layer $l+1$ can vary based on the groups of these two layer.

Caps layer l	**Caps layer $l+1$**	**Weight sharing**	**$W^{[l]}$ dimensions**
PrimaryCaps+Flatten, ConvCaps+Flatten, FcCaps	FcCaps	False	$\lvert\Omega^{[l]}\rvert \times \lvert\Omega^{[l+1]}\rvert \times D^{[l]} \times D^{[l+1]}$
PrimaryCaps+Flatten, ConvCaps+Flatten, FcCaps	FcCaps	True	$\lvert\Omega^{[l+1]}\rvert \times D^{[l]} \times D^{[l+1]}$
PrimaryCaps, ConvCaps	FcCaps	False	$M^{[l]} \times N^{[l]} \times O^{[l]} \times \lvert\Omega^{[l+1]}\rvert \times D^{[l]} \times D^{[l+1]}$
PrimaryCaps, ConvCaps	FcCaps	True	$O^{[l]} \times \lvert\Omega^{[l+1]}\rvert \times D^{[l]} \times D^{[l+1]}$
PrimaryCaps, ConvCaps	ConvCaps	False	$K^{[l]} \times K^{[l]} \times O^{[l]} \times O^{[l+1]} \times D^{[l]} \times D^{[l+1]}$
PrimaryCaps, ConvCaps	ConvCaps	True	$K^{[l]} \times K^{[l]} \times O^{[l+1]} \times D^{[l]} \times D^{[l+1]}$

Table 3.1: The dimensions of the weight matrix in two consecutive capsule layers in different configurations, where $K^{[l]}$ denotes the kernel size in convolutional capsule layers. Here, we assume that $\lvert\Omega^{[l]}\rvert \neq \lvert\Omega^{[l+1]}\rvert$.

3.4.1 *Dynamic Routing*

In 2017 Sabour et al. [1], Hinton presented the first CapsNet architecture along with the first description of the routing algorithm based on dynamic agreements (DR-CapsNet) using the scalar product. Due to its relevance for our work and for sake of simplicity, we will go into details only for this routing algorithm.

A capsule is implemented as an activity vector $\boldsymbol{u}$ whose overall length $\|\boldsymbol{u}\|_2$ represents the probability of existence of the entity and its orientation represents its instantiation parameters.

The information flow between two capsules is governed by the iterative routing algorithm described in [1] and shown in Alg. 1. This procedure is used to dynamically compute the poses $\boldsymbol{u}_j^{[l+1]}$ of

Algorithm 1 Dynamic routing algorithm

```
1:  procedure DYNAMIC-ROUTING($\hat{\boldsymbol{U}}^{[l+1]}$, $r^l$)
2:      for each capsule $i \in |\Omega^1|$ and capsule $j \in |\Omega^{1+1}|$: $b_{i,j}^{[l]} \leftarrow 0$
3:      for $r^l$ iterations do
4:          for each capsule $i$: $\boldsymbol{c}_{i,:}^{[l]} \leftarrow \sigma(\boldsymbol{b}_{i,:}^{[l]})$
5:          for each capsule $j$: $\boldsymbol{s}_j^{[l+1]} \leftarrow \sum_i c_{i,j}^{[l]} \hat{\boldsymbol{u}}_{i,j}^{[l+1]}$
6:          for each capsule $j$: $\boldsymbol{u}_j^{[l+1]} \leftarrow g(\boldsymbol{s}_j^{[l+1]})$
7:          for each capsule $i$ and capsule $j$: $b_{i,j}^{[l]} \leftarrow b_{i,j}^{[l]} + \boldsymbol{u}_j^{[l]} \cdot \hat{\boldsymbol{u}}_{i,j}^{[l+1]}$
8:      end for
9:      return $\boldsymbol{U}_j^l$
10: end procedure
```

capsules j in layer $l+1$ given the vote matrix $\hat{\boldsymbol{U}}^{[l+1]}$, which holds the predictions $\hat{\boldsymbol{u}}_{i,j}^{[l+1]}$, computed using Equation 3.1. At the beginning of the routing algorithm (line 2), each logit $b_{i,j}^{[l]}$ is initialized to zero. The core of the routing algorithm is depicted in lines 3-8. At every iteration, the routing softmax (line 4, Equation 3.2) is applied to the logits $\boldsymbol{b}_{i,:}^{[l]}$ to obtain the corresponding coupling coefficient $c_{i,j}^{[l]}$. Then, the total unnormalized input $\boldsymbol{s}_j^{[l+1]}$ of capsule j is computed as the weighted average of the input predictions (line 5) as

$$\boldsymbol{s}_j^{[l+1]} \leftarrow \sum_i c_{i,j}^{[l]} \hat{\boldsymbol{u}}_{i,j}^{[l+1]}. \tag{3.3}$$

Each vote $\hat{\boldsymbol{u}}_{i,j}^{[l+1]}$ is weighted by the corresponding coupling coefficient $c_{i,j}^{[l]}$. The vector $\boldsymbol{u}_j^{[l+1]}$ is the normalized *squashed* $\boldsymbol{s}_j^{[l+1]}$ (line 6), where the squashing function g map the magnitude of a vector into the range [0, 1)

$$\boldsymbol{u} = g(\boldsymbol{s}) = \frac{\|\boldsymbol{s}\|^2}{1+\|\boldsymbol{s}\|^2} \frac{\boldsymbol{s}}{\|\boldsymbol{s}\|}. \tag{3.4}$$

Then, each $b_{i,j}^{[l]}$ is refined by measuring the agreement between the output $\boldsymbol{u}_j^{[l]}$ of a capsule j and the prediction $\hat{\boldsymbol{u}}_{i,j}^{[l+1]}$ (line 7). Therefore, if there is a strong agreement, the corresponding link strength $b_{i,j}^{[l]}$ between capsules i and j is increased, decreased otherwise. The agreement is simply the scalar product $\boldsymbol{u}_j^{[l]} \cdot \hat{\boldsymbol{u}}_{i,j}^{[l+1]}$. Finally, after $r^{[l]}$ iterations of lines 4-7, the routing algorithm output the final pose $\boldsymbol{u}_j^{[l]}$ for each digit capsule.

The activity vector of the correct digit capsule is used to reconstruct the input image. During testing we take the capsule with the highest magnitude. This capsule is fed into the decoder which consists of three traditional fully-connected layers. The decoder encourage the digit capsules to encode the instantiation parameters [1].

Note that this iterative process is executed both on training and testing time only between two consecutive capsule layers.

In order to train the network, two losses are used: the *margin loss* and the *reconstruction loss*.

The margin loss is based on the idea that the class capsule for a label γ should have high magnitude if and only if that the object with label γ is present in the input image. We have a separate margin loss $\mathcal{M}_\gamma$ for each label γ:

$$\mathcal{M}_\gamma = T_\gamma \max(0, m^+ - \|\boldsymbol{u}_\gamma^{[l]}\|_2)^2 + \lambda(1 - T_\gamma)\max(0, \|\boldsymbol{u}_\gamma^{[l]}\|_2 - m^-)^2 \tag{3.5}$$

where $T_\gamma = 1$ if and only if the object of label y is present. The λ term prevents initial learning from shrinking the lengths of the capsules of all the classes.

The reconstruction loss $\mathcal{R}$ is the sum of squared differences between the reconstructed image and the original one. It is used a regularization term during training.

The total margin loss is defined as:

$$\mathcal{L}_\mathcal{M} = \sum_\gamma \mathcal{M}_\gamma + \beta\mathcal{R} \tag{3.6}$$

DR-CapsNets achieved higher classification performance than the convolutional baseline on MNIST [21], Fahion-MNIST [22], CIFAR-10 [23] and SVHN [24] datasets. They also showed more robustness to affine transformations (affNIST) and novel viewpoints (smallNORB). Capsules are also very good for dealing with segmentation highly overlapping digits (MultiMNIST [1]).

3.5 DRAWBACKS

Capsule networks have some limitations.

1. Capsules make a very strong representational assumption [1]: at each location in the image, there is at most one instance of the

type of entity a capsule represents. As a matter of fact, the only thing that binds the instantiation units together is the fact that they are the pose outputs of the same capsule. Therefore, it is not possible for a capsule to represent more than one instance of its visual entity at the same time [1]. This is a serious weakness because it will always be possible to confound the system by putting two instances of the same visual entity with slightly different poses in the same region. This phenomenon is known as *crowding* and it occurs also in human vision.

2. Capsules, like other generative models, account for everything in the image so they have poorer performances when the backgrounds are too much varied, like in CIFAR10, and the size of the network is not big enough to model the clutter.

3. Training capsule networks is computationally expensive because of the large transformation matrices and because of the routing algorithm which can not benefit from the parallel computation of modern GPUs, increasing both training and testing time respect to a standard CNN. This is one of the reason why they have not been trained yet on datasets with high resolution such as ImageNet [25].

4. A capsule network has many hyperparameters like the number of types of capsules, their dimensions in every layer, the shape of the kernels etc. This could make fine-tuning in capsule network more difficult than in standard CNNs.

3.6 CAPSULE NETWORKS FOLLOW-UPS

CAPSULE NETWORKS ARCHITECTURES Capsule networks were first introduced by Sabour et al. [1], and since then, much work has been done to improve the routing stage and build deeper models. Regarding the routing algorithm, Hinton et al. [26] replaces the dynamic routing with Expectation-Maximization, adopting matrix capsules instead of vector capsules. Wang et al. [27] model the routing strategy as an optimization problem. Li et al. [28] use master and aide branches to reduce the complexity of the routing process. Peer et al. [29] use inverse distances instead of the dot product to compute the agreements between capsules to increase their transparency and robustness against adversarial attacks. Hahn et al. [30] incorporates a *self-routing* method such that capsule models no longer require agreements. De Sousa Ribeiro et al. [31] replace the routing algorithm with a variational inference of part-object connections in a probabilistic capsule network, leading to a significant speedup without sacrificing performance. Ribeiro et al. [32] propose a new routing algorithm derived from Variational Bayes for fitting a mixture of transforming Gaussians. Edraki et al. [33] model entities through a group of capsule subspaces without any form of routing. Since the CapsNet model introduced by Sabour et al. [1] is a shallow network, several works attempted to build deep CapsNets. Rajasegaran et al. [34] propose a deep capsule network architecture that uses a novel 3D convolution-based dynamic routing algorithm to improve the performance of CapsNets for more complex image datasets. Gugglberger et al. [35] introduce residual connections to train deeper capsule networks. The above-mentioned CapsNets architectures rely on the fact that capsules model specific location-dependent features of objects in the image. However, [36] claims that it is better to model capsules as *set* of parts or objects. This assumption ensures that if the parts can be detected equivariantly, then the inferences for the objects will also be equivariant [37]. Therefore, Kosiorek et al. [36] propose an unsupervised stacked capsule autoencoder (SCAE) consisting of two stages. First, it segments an image into constituent parts, predicting the presence and poses of part templates. Then, it tries to organize discovered parts and their poses into a smaller set of objects, which are then used to reconstruct part poses. Similarly, Nazabal et al. [37] consider a scene or an image composed of a set of objects in different poses, which are instations of affine-transformed fixed templates. They show that the parts detected in the first stage by SCAE are not equivariant to rotation. Therefore, they propose a generative model and derive a variational algorithm for inferring the transformation of each model object in a scene and the assignments of observed parts to the objects. The model is interpretable and admits prior knowledge about an object. Furthermore, their formulation is also composable in that models for individual objects can be learned separately and then combined together at inference time. Alternatevely to capsules, other object-centric representation learning approaches using neural networks exist in the literature, for example, slot-based models [38]. Slots are "universal" capsules that can bind to any object in the input; they provide a sort of working memory with a fixed capacity which can be used to access independent object representations simultaneously.

Recently, Hinton [39] described how to represent part-whole hierarchies in neural networks, which attempts to merge the advantages of capsules, contrastive learning, distillation and transformers into a single theoretical system (known as GLOM). GLOM aims at mimicking how humans parse visual scenes. Still, these approaches are not tested on more complicated datasets such as ImageNet, which limits their use in downstream tasks, and consequently, capsules have not yet gained widespread adoption in the literature.

SPARSE CAPSULE NETWORKS A naive solution to reduce uncertainty within the routing algorithm is to run more iterations. As shown by Paik et al. [40] and Gu et al. [41], the routing algorithms tend to overly polarize the link strengths, namely a simple route in which each input capsule sends its output to only one capsule and all other routes are suppressed. On the one hand, this behavior is desirable because the routing algorithm computes binary decisions to either connect or disconnect objects and parts. On the other hand, running many iterations is computationally expensive, and it is only helpful in the case of networks with few parameters, as we will see in Chapter 4. Rawlinson et al. [42] trained CapsNets in an unsupervised setting, showing that the routing algorithm no longer discriminates among capsules: the coupling coefficients collapse to the same value. Therefore, they sparsify latent capsule layers activities by masking output capsules according to a custom ranking function. Kosiorek et al. [36] impose sparsity and entropy constraints into capsules, but they do not employ an iterative routing mechanism. Jeong et al. [43] introduced a structured pruning layer called ladder capsule layers, which removes irrelevant capsules, namely capsules with low activities. Kakillioglu et al. [44] solve the task of 3D object classification on point clouds with pruned capsule networks. They aimed to compress robust capsule models to deploy them on resource-constrained devices.

3.7 COMPARISON WITH VISION TRANSFORMERS

In this section, we compare Vision Transformers (ViTs) [2] and CapsNets architectures, focusing on their similarities and differences.

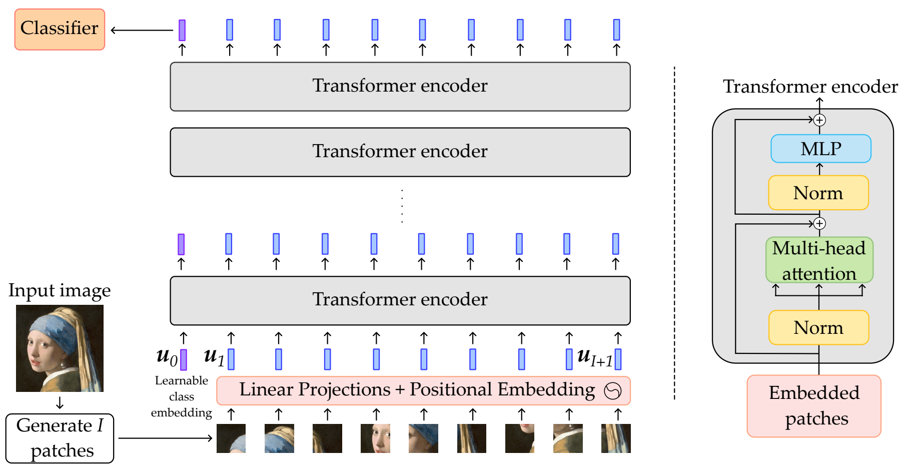


Figure 3.5: Overview of the ViT architecture illustrated in [2]. Fixed-size patches are extracted from an input image; then, they are linearly embedded with the addition of position embeddings to form tokens, which are then fed to a standard transformer encoder. Multiple blocks of transformer encoder can be stacked together for deeper models. An extra learnable classification token is added to the input sequence. The embedding vector of this token in the last layer can be seen as a compact representation of the input image. Therefore, it is fed to a multilayer perceptron network (MLP) to perform classification. An alternative is to have a global average pool over all the output embedding vectors.

Even though capsule networks have been around for a while, they have not received as much attention as transformers. These networks, introduced around the same time as CapsNets, have become quite popular and are widely used in various applications, including large language models and computer vision. ViTs, which are transformers designed for computer vision, generally consist of only the encoder part of a standard transformer. An overview of ViT architecture is depicted in Figure 3.5, while Figur 3.6 depicts the multi-head attention layer that extends the idea of single-head attention by performing self-attention computations multiple times in parallel.

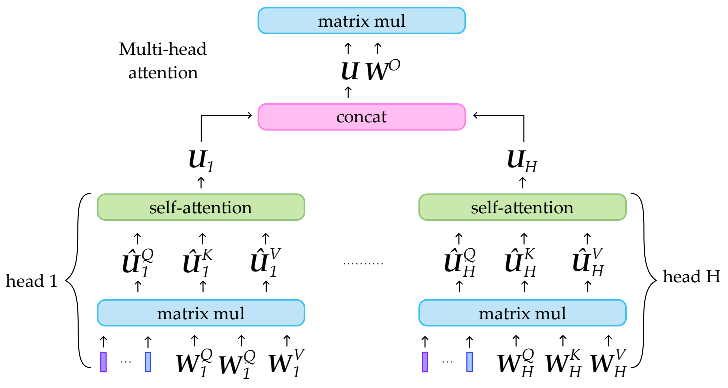


Figure 3.6: Multi-head attention is a module that runs MLPs and self-attentions several times in parallel. The independent attentions computed by different heads are then concatenated and linearly transformed into the expected dimension. Intuitively, multiple attention heads allow for attending to parts of the input differently.

In this Section, we use the same notation we adopted when describing capsule network components. For example, we denote with $\boldsymbol{u}^{[l]} \in \mathbb{R}^{D^{[l]}}$ the D-dimensional embedding vectors in layer l, as we did with a capsule vector. Since we consider only two consecutive layers, let $\boldsymbol{U}^{[l]} = (\boldsymbol{u}_1^{[l]}, ..., \boldsymbol{u}_I^{[l]})$ denote a matrix of input embedding vectors in layer l, and $\boldsymbol{U}^{[l+1]} = (\boldsymbol{u}_1^{[l+1]}, ..., \boldsymbol{u}_J^{[l+1]})$ denote a matrix of output embedding vectors in layer $l+1$ (or capsules), with $\boldsymbol{u}_i^{[l]} = (u_1^{[l]}, ..., u_{D^{[l]}}^{[l]})$ and $\boldsymbol{u}_j^{[l+1]} = (u_1^{[l+1]}, ..., u_{D^{[l+1]}}^{[l+1]})$. We can omit the l notation since $i \in \{1, 2, ..., I\}$ is used for inputs of the single-head attention and $j \in \{1, 2, ..., J\}$ for the outputs. We also use $D = D^{[l]}$ and $P = D^{[l+1]}$.

In scaled dot product self-attention [2] and shown in Figure 3.6, I input embedding vectors $\boldsymbol{U}_i \in \mathbb{R}^{I \times D}$ are transformed into query, key and value matrices for each head $h \in \{1, 2, ..., H\}$ as

$$\hat{\boldsymbol{U}}_h^Q = \boldsymbol{U}_i \boldsymbol{W}_h^Q, \qquad \hat{\boldsymbol{U}}_h^K = \boldsymbol{U}_i \boldsymbol{W}_h^K, \qquad \hat{\boldsymbol{U}}_h^V = \boldsymbol{U}_i \boldsymbol{W}_h^V, \tag{3.7}$$

where $\boldsymbol{W}_h^Q$, $\boldsymbol{W}_h^K$ and $\boldsymbol{W}_h^V$ are the trainable $D \times P$ transformation matrices. Therefore, $\hat{\boldsymbol{U}}_h^Q$, $\hat{\boldsymbol{U}}_h^K$ and $\hat{\boldsymbol{U}}_h^V$ are $I \times P$ dimensional. Here, we assume that the new query, key, and value vectors have the same dimension as the embeddings (usually, they are smaller). Let be $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$ and $\boldsymbol{W}^V$ the weight matrices considering all attention heads, these are $H \times D \times P$ dimensional, where usually $H < J$. Note that typically, between two consecutive fully-connected capsule layers (so capsules in these layers are flattened and not organized in a grid, without weight sharing), as also shown in Table 3.1, the transformation weight matrix $\boldsymbol{W}^Q$ is $I \times J \times D \times P$ dimensional, therefore the vote matrix $\hat{\boldsymbol{U}}$ is $I \times J \times P$ dimensional. The self-attention mechanism computes the attention weights $c_{i,j,h}$ (or coupling coefficients in capsules) between each pair (i, j) of embedding vectors in head h as

$$c_{i,j,h} = \sigma \left(\frac{\hat{\boldsymbol{U}}_h^Q \hat{\boldsymbol{U}}_h^{K^\top}}{\sqrt{D}} \right)_{ij}, \qquad \sum_{i=1}^{I} c_{i,j,h} = 1, \tag{3.8}$$

where σ is the softmax function. Then, a single output embedding (or capsule) $\boldsymbol{u}_{j,h}$ in a head h for token j is computed as a weighted average of the input token vectors

$$\boldsymbol{u}_{j,h} = \sum_{k=1}^{I} c_{j,k,h} \hat{\boldsymbol{U}}_{h,k}^V \tag{3.9}$$

Note that in ViTs, the number of embedding vectors for the input tokens is the same in each layer, namely $I = J$. Therefore we can rewrite Equation 3.9 as

$$\boldsymbol{u}_{i,h} = \sum_{j=1}^{I} c_{i,j,h} \hat{\boldsymbol{U}}_{h,j}^V \tag{3.10}$$

Therefore, a significant distinction between capsules and transformer embeddings lies in how they utilize context. In the case of capsules, to calculate the vector of an output capsule (and attention weights), all other output capsules are required as context (remember that in the vote matrix, there are $I \times J$ predictions of the input capsule vectors for the output capsule vectors). In contrast, in ViTs, the vector of an output embedding is computed by considering all the other input embeddings as context (there are three "vote" matrices, query, key and value, so the total number of votes is $I \times 3$). Note that, in the language of transformers, the output capsules at iteration $r - 1$ act as the query matrix at iteration r, while capsule votes act as the key and value matrices. See Figure 23 in [45] for comparing capsule routing and self-attention in Pytorch code.

So far, we have seen that CapsNets and ViTs have a common ground regarding learned representations. For example, both methodologies organize neuron activities into vectors to represent concepts, and they compute the agreement between these vectors as a filtering mechanism to refine concepts based on global context [32]. Moreover, both models can be stacked on top of a CNN backbone to build hybrid architectures.

Note that typically, a ResNet model is preferred since it has only one max pooling layer, so that the spatial relationships are not completely lost when going deeper in the network. In the case of a ViT, as an alternative to raw image patches, the input sequence can be formed from feature maps of a CNN, and then patches are extracted from a CNN feature map[2]. This hybrid model inherits, for example, the ability to learn mutual regions at different scales in a CNN, thanks to multiple convolutional layers with different kernel sizes. As we will see in Chapter 5, in the case of a CapsNet, the CNN backbone (without pooling layers) can help in extracting effective low-level part descriptions (primary capsules) since CNNs are very good at learning low-level features, such as edges, corners and basic local shape information.

However, CapsNets and ViTs have different *inductive biases*. Recall that inductive biases in deep learning architectures refer to the implicit assumptions or constraints that are built into the design of a neural network model. These biases guide the learning process by favoring certain types of solutions or representations over others.

LOCALITY Primary and convolutional capsule layers are organized in a 3-dimensional grid, and the sliding window approach (as in CNNs) assumes that important features are present in small, local regions of an image and that these features can be detected irrespective of their position within the image. On the contrary, ViTs do not have a built-in notion of spatial locality and layers are not organized in a grid structure. Images are tokenized into smaller patches, which is a different inductive bias than the sliding window approach. They use positional embeddings to encode the spatial information of the input image, and these positional embeddings provide information about where each token (representing a patch of the image) is located in the image.

SINGLE-PARENT ASSUMPTION Both routing and self-attention compute attention weights between capsules or embeddings. However the semantics of this attention weights is different. In fact, in CapsNets, since routing is a clustering mechanism, c_{ij} is an assignment probability, namely to what extent object i is part of object j. This is also known as the single-parent assumption (see the softmax in Equation 3.2), since the goal of a CapsNet is to build part-whole hierarchies. While in ViT, c_{ij} models how to attend to different parts of the input and how information from different parts contributes to the updates of representations. We can see in Figure 3.7 the opposite directions of the connections in capsules (bottom-up) and token embeddings (top-down).

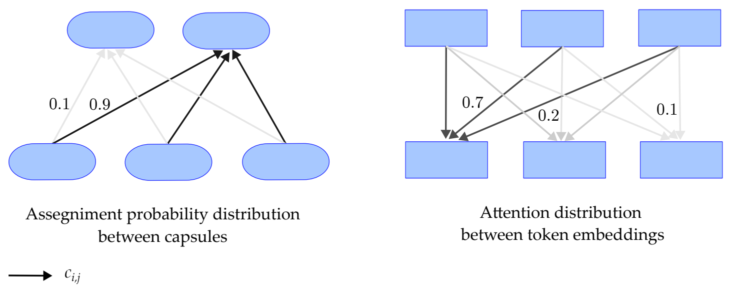

0.1
0.9
0.7
0.2
0.1
Assegniment probability distribution between capsules
Attention distribution between token embeddings
$c_{i,j}$

Figure 3.7: In CapsNets, the competition is between capsules in the higher layer (bottom-up attention), while in ViTs, the competition is between embeddings in the lower one (top-down attention).

NUMBER OF OBJECTS Figure 3.7 also shows that, when assembling capsules, we often have a smaller number of output capsules than input capsules ($J < I$) because we assume that, in an input image, the number of objects is lower than the number of parts, since an object is composed of many parts. This also means that we do not have any prior assumption on the representations of the higher layer capsules, what they are supposed to represent is not known in advance. In contrast, in ViTs, the number of embeddings in each layer is constant ($J = I$), corresponding to the number of patches. This is because an embedding can be considered a contextualized representation of the same patch in each layer.

CAPSULE TYPES AND ATTENTION HEADS Different capsule types in CapsNets and attention heads in ViTs are akin to the diverse kernels in CNNs. They aim to process lower-layer representations from multiple viewpoints when computing higher-layer representations. In CapsNets, different transformation matrices exist between pairs of capsule types from adjacent layers. This way, each higher-layer capsule sees the entity in the lower-layer capsule from a different point of view. Similarly, in ViTs, multiple attention heads employ distinct transformation matrices to compute key, value, and query projections, allowing each head to focus on a different perspective of lower-layer representations. In real implementations[1], each head focuses on a different portion of the input vector, namely D will be split across H heads (i.e. each head will have dimension $D//H$). However, the key difference between the two architectures is that, in the end, in CapsNets, assignment probabilities for a lower-layer capsule are normalized over all higher-layer capsules, irrespective of their types, resulting in one assignment distribution per lower-layer capsule (as seen in Figure 3.7). In contrast, in ViTs, each attention head independently processes its input, yielding separate attention distributions for each position in the higher layer. The attention head outputs are combined in the last step

1 https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html

by concatenating and linearly transforming them to compute the final output of the multi-headed attention block.

INVARIANCE AND EQUIVARIANCE As explained in Section 2.2 and Section 3.2, CapsNets take advantage of the linear manifold that handles the source of variance in images thanks to the transformation matrices $\boldsymbol{W}$s used to compute the votes $\hat{\boldsymbol{u}}$s. Therefore, we say that the vote transformation matrix $\boldsymbol{W}$ is *biased* towards encoding invariance to viewpoint transformations, and capsule vectors are *biased* towards capturing equivariance of neural activities Ribeiro et al. [45]. In ViTs, only MLP layers are local and translationally equivariant [2].

So, we have seen that ViTs inductive biases are more relaxed compared to CapsNets, or even CNNs. However, inductive biases are essential because they help neural networks generalize from limited training data and make predictions in situations they haven't seen before. Thus, ViTs do not generalize well when trained on insufficient amounts of data [2] and must learn all spatial relations between the patches from scratch. However, when trained on large amounts of data, large-scale training trumps inductive bias [2]. Recently, Hinton [39][39] proposed a single theoretical framework (called GLOM), which attempts to merge the advantages of capsules and transformers to parse visual scenes. However, research is still in the early stages, and as of now, a high-performing GLOM network pretrained on ImageNet has not surfaced yet.

Part II

DEMYSTIFYING CAPSULE NETWORKS COMPONENTS

This part delves into the intriguing aspects of capsule networks, focusing on three key aspects that aim to unlock their full potential:

- **Effectiveness of the routing algorithm**: We introduce a novel technique that anneals the number of routing iterations during training, striking better performance, especially for architectures with fewer parameters.
- **Extraction of low-level part descriptors**: We propose a method to extract better first-layer capsules promoting sparse topologies. We improve their computational efficiency by reducing the number of capsules.
- **Cardinality of part-whole relationships**: We show that capsules with low entropy hold the potential to extract more succinct and more discriminative part-relationships than traditional capsule networks. This can be achieved without accounting for everything in the input image, even with reasonable network sizes.

4 CAPSULE NETWORKS WITH ROUTING ANNEALING

The research findings and insights presented in this chapter were originally published in the following work: **Renzulli, R.** "Capsule Networks with Routing Annealing." In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Ed. by Igor Farkaš et al. 2021, pp. 529–540. ISBN: 978-3-030-86362-3.

4.1 INTRODUCTION

Recently, the contribution of the routing algorithm to DR-CapsNets generalization ability and robustness to affine transformations has been questioned [40, 41]. Typically, the number of routing iterations r is fixed once and for all during training and inference. Paik et al. [40] highlight that running just one iteration of the routing algorithm (assigning the connection strengths uniformly or randomly) leads to better results. This is explained as more iterations of the routing algorithms do not change the classification result but polarize the link strengths [40]. Gu et al. [41] mitigate this problem with a simple but effective solution in which the transformation matrices are shared between all capsule types. However, these works do not change the number of iterations during the training process nor the number of capsule types and their dimensions, which, as we will see, they do have a strong impact on the number of iterations of the routing algorithm that achieves better generalization. Therefore, it is unclear whether the routing algorithm improves the performance of DR-CapsNets and the optimal number of iterations.

This chapter provides new evidence on the benefits of routing proposing *Routing Annealing (RA)*, a novel technique where the number of routing iterations is iteratively found at training time. With RA, the number of iterations of the routing algorithms increases whenever the network performance reaches a loss plateau. We observed that, for the same number of routing iterations, a gradual ramp thereof allows to reach better minima of the loss function. Our experiments over multiple datasets show better performance when using RA, especially when the number of capsules in the network is limited, i.e. where DR-CapsNets performance is weaker. We also found that the number of routing iterations depends on the number of capsules, their dimensions and on the dataset itself.

4.2 methodology

This section first describes the standard methodology training algorithm and *Routing Annealing* (RA), the routing training technique we propose in this work, and then discusses its relation with the simulated annealing. For simplicity of notation, we assume there are only two capsule layers (primary capsules and class capsules), and we suppress the index l for capsule vectors $\boldsymbol{u}$ and routing iterations r. However, RA can be applied to any multilayered CapsNet whose connections are governed by any iterative routing algorithm and where the number of routing iterations can vary for different layers.

4.2.1 *Training with Fixed Routing*

As a reference, Alg. 2 shows the standard strategy for training a DR-CapsNet. The network parameters are optimized with standard backpropagation of the error gradients for a number of epochs until some stop criterion is met. For each epoch, the forward pass (line 5-11) is computed, followed by error gradients backpropagation and parameter update (line 12). The training procedure ends when the loss computed over a validation set does not decrease for p epochs in a row (p is usually termed as *patience*). The algorithm returns the network (i.e., the learned parameters set) that yields the lowest loss on the validation set. In this procedure, as can be noted from line 6 which refers to Alg. 1, the number of routing iterations r is fixed once for all (usually, r=3), so we refer to this technique as *Fixed Routing* (FR). Notice that when the trained network is deployed for inference, the routing algorithm is executed for r iterations, as well. A standard procedure towards optimising the iterations number would be to optimize r with a grid-search strategy: one runs as many simulations as r values to test, during which r is kept constant. However, we experimentally show that this approach leads to sub-optimal performance, which motivates the design of our routing technique.

4.2.2 *Training with Routing Annealing*

In this section we propose *Routing Annealing* (RA), an iterative method to jointly optimize the number of routing iterations $r^\star$ and the network parameters. In a nutshell, RA finds $r^\star$ adaptively during training for a given capsule architecture over a given dataset and is described in pseudo-code as Alg. 3. The algorithm takes as input: r_0, the initial value of r; r_T, the maximum value for r; s, the schedule used to increase r; the patience p, in number of epochs. Let us denote as r_k the value of r at step k: we say that every time r increases, an *annealing step* is performed. We denote as $\mathcal{L}_k^\star$ and $e_k^\star$ the lowest losses achieved so far and the corresponding epochs for each r_k. The main difference

Algorithm 2 Training with *Fixed Routing*: learns the network parameters for a fixed number of iterations r.

```
1:  procedure FIXED-ROUTING($r$, $p$)
2:      initialize DR-CapsNet
3:      $e \leftarrow 0$;   $\mathcal{L}^\star \leftarrow 0$; $e^\star \leftarrow 0$
4:      while $e - e^\star < p$ do
5:          compute all primary capsules poses $\boldsymbol{u}_i$ and votes $\hat{\boldsymbol{u}}_{i,j}$
6:          compute all class capsules poses: $\boldsymbol{u}_j \leftarrow$ ROUTING($\hat{\boldsymbol{u}}_{i,j}$, $r$)
7:          evaluate current loss $\mathcal{L}$ on the validation set
8:          if $\mathcal{L} < \mathcal{L}^\star$ then
9:              $\mathcal{L}^\star \leftarrow \mathcal{L}$; $e^\star \leftarrow e$
10:         end if
11:         $e \leftarrow e + 1$
12:         backpropagate error gradients and update parameters
13:     end while
14:     return DR-CapsNet of epoch $e^\star$ with the best loss value
15: end procedure
```

between Alg. 2 and 3 lies in line 3 where we loop over the possible values of r instead over the number of epochs and in line 8 where the number of routing iterations is increased. In line 5, Alg. 1 is used as core routing algorithm. At step k, we increase r by s if the validation loss $\mathcal{L}_k^\star$ does not decrease for p epochs (lines 7-8). Every time r is increased, the training does not start from scratch again. Instead it is resumed with the network weights with the best loss achieved with the previous value of r, namely the network at epoch $e_{k-1}^\star$ (line 9). Here we assume that we save the network weights for each epoch. When r reaches the maximum allowed r_T, the training procedure ends and best network obtained during training along with the corresponding number of routing iterations is returned (lines 16-17). To summarize, RA increases the value of r when the validation loss does not decrease for p epochs in a row. As an upper bound for the number of routing iterations, we stop the training when r reaches its maximum value r_T. When r increases, the training restarts with the weights of the network with the best validation loss obtained with its previous value. By comparison, using the standard training procedure mentioned in Sec. 4.2.1, the weights need to be reinitialize for every simulation with a a different value for r.

4.2.3 *Rationale*

RA takes inspiration from the *simulated annealing* (SA) algorithm, a probabilistic technique used in combinatorial-optimization problems to minimize a cost function.

Algorithm 3 Training with *Routing Annealing*: learns the number of iterations $r^\star$ jointly with the network parameters.

```
1: procedure ROUTING-ANNEALING($r_0$, $r_T$, $s$, $p$)
2:     $r$; $r \leftarrow r_0$; $\mathcal{L}_0^\star \leftarrow +\infty$; $e_0^\star \leftarrow 0$ , $e \leftarrow 0$; $k \leftarrow 0$
3:     while $r \leq r_T$ do
4:         compute all primary capsules poses $\boldsymbol{u}_i$ and votes $\hat{\boldsymbol{u}}_{i,j}$
5:         compute all class capsules poses: $\boldsymbol{u}_j \leftarrow$ ROUTING($\hat{\boldsymbol{u}}_{i,j}$, $r$)
6:         evaluate loss $\mathcal{L}$ on the validation set
7:         if $(\mathcal{L} \geq \mathcal{L}_k^\star)$ and $(e - e_k^\star \geq p)$ then
8:             $k \leftarrow k+1$; $r \leftarrow r+s$; $r_k \leftarrow r$; $\mathcal{L}_k^\star \leftarrow +\infty$; $e_k^\star \leftarrow 0$
9:             load DR-CapsNet network of epoch $e_{k-1}^\star$
10:        else if $\mathcal{L} < \mathcal{L}_k^\star$ then
11:            $\mathcal{L}_k^\star \leftarrow \mathcal{L}$; $e_k^\star \leftarrow e$
12:        end if
13:        $e \leftarrow e+1$
14:        backpropagate error gradients and update parameters
15:    end while
16:    $k^\star \leftarrow \underset{k}{\operatorname{argmin}}\, \mathcal{L}^\star$; $r^\star \leftarrow r_{k^\star}$
17:    return DR-CapsNet network of epoch $e_{k^\star}^\star$ and $r^\star$
18: end procedure
```

In our approach, we relate the temperature of our system being inversely proportional to the number of routing iterations r: the highest r, the highest the agreement between the capsules and the lowest the noise.
The number of routing iterations relates to the distribution of the coupling coefficients $c_{i,j}$. According to Alg. 1, when r is low, the agreement is low as well. When $r = 1$, all the coupling coefficients will have the same value, $\frac{1}{J}$. Increasing the routing iterations, a certain number of coupling coefficients becomes dominant over others, since Alg. 1 looks for capsule's agreement. Considering that $c_{i,j}$ are normalized values, we can say that, for the i-th capsule $\sum_{j \in \mathcal{J}_i} c_{i,j} \rightarrow 1$ and $\sum_{j \in \overline{\mathcal{J}_i}} c_{i,j} \rightarrow 0$, where $\mathcal{J}_i$ is a subset of the J coupling coefficients for the i-th primary capsule and $\overline{\mathcal{J}_i}$ is its complementary set. When $r = 1$, the cardinality of $\mathcal{J}_i$ is exactly J, but increasing r, its cardinality drops to some optimal value $J_i^\star$: this means that the i-th primary capsule will be coupled to $J_i^\star$ digit capsules only, avoiding noisy coupling to the others (which are $J - J_i^\star$). A visual representation of this effect is displayed in Fig. 4.1. As r increases, many coupling coefficients drop to zero, while others converge to higher coupling values. In this way, the routing algorithm learns how to build relationships between primary and digit caps, discarding noisy information, which helps in improving the generalization of the model. In the next section we are going to test on-the-field our RA strategy, observing in particular

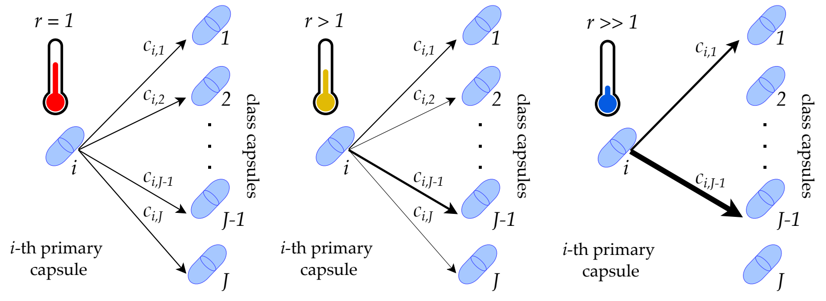


Figure 4.1: Routing coupling coefficients between the i-th primary capsule and J class capsules. The higher the line weight, the higher the corresponding coupling coefficient. When $r = 1$ (left), the coupling coefficients have all the same value, while increasing r (center), some of the coupling parameters survive, while the others are almost zero. When r is high (right), $J^\star = 2$ and $\mathcal{J}_i = \{1, J-1\}$.

the generalization capability of the RA models compared to the other state-of-the-art approaches.

4.3 EXPERIMENTS

In this section we compare our proposed Routing Annealing (RA) method in Alg. 3 against the reference method in Alg. 2. First we show that, with RA, DR-CapsNet performs better as the number of routing iterations r improves, whereas this is not the case with the reference algorithm. Then, we further validate RA on multiple datasets and settings showing that it delivers best gains especially where the number of parameters the network can afford is low, i.e. where DR-CapsNets performance is weaker.

4.4 ARCHITECTURE

Fig. 4.2 shows the DR-CapsNet architecture used in the experiments in this chapter, consisting of one convolutional layer and two capsule layers.

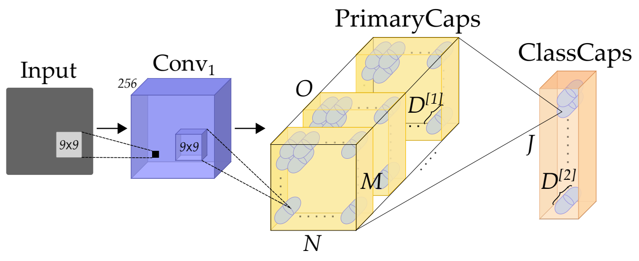


Figure 4.2: DR-CapsNet architecture. There are one convolution layer (Conv1) and two capsule layers (PrimaryCaps and ClassCaps). The dynamic routing algorithm controls the information flow between capsule layers.

The first layer (Conv_1) is a 9×9 convolutional layer that converts pixel intensities to the activities that are given in input to the first capsule layer. The PrimaryCaps layer is implemented as a convolutional layer with 9×9 filters and $O \times D^{[1]}$ channels where O is the number of primary capsules types and $D^{[1]}$ is the dimension of a capsule vector. Overall, there are $O \times M \times N$ primary capsules. Capsule vectors are normalized with the squashing function defined in Equation 3.4. The output layer (ClassCaps) comprises J $D^{[2]}$-dimensional class capsules $\boldsymbol{u}_j$, one for each output class. The information flow between primary and class capsules is governed by the dynamic routing described in Algorithm 1.

4.4.1 *Experimental Setup*

We experiment with the DR-CapsNet in Fig. 4.2 at classifying natural images in a fully supervised scenario. We consider the MNIST, Fashion-MNIST and CIFAR-10 datasets. For all datasets, 5% of the training set samples are reserved for validation. MNIST and Fashion-MNIST are composed of 28x28 images; concerning CIFAR-10, we randomly crop the original 32x32 images into 24x24 patches for training whereas crops from the image center are used for testing as done in [1]. Our experiments consider several flavors of the architecture in Fig. 4.2 with different types $O \in \{1, 2, 4, 8, 16, 32\}$ and dimensions $(D^{[1]}, D^{[2]}) \in \{(2, 4), (4, 8), (8, 16)\}$ of capsules. We train the network minimizing a *margin loss* [1] with the Adam optimizer [47] with a constant learning rate equal to 0.001 and a batch size of 128. No weight decay, dropout or other regularization techniques were used.

Concerning the proposed RA method, we train the network with the procedure in Alg. 3, with the following configuration: $r_0 = 1$, $r_T = 50$, $s = 1$ and $p = 10$. As we discussed in Sec. 4.2.2, RA can be applied to any iterative routing algorithm but this work use as base routing algorithm the one described in Alg. 1.

About the reference method, we use FR which employs the procedure in Alg. 2, i.e. where the number of routing iterations r is fixed (common values in literature are $r = 1$ [41] or $r = 3$ [1]).

4.4.2 *Preliminary analysis on MNIST*

Preliminary, we assess the effect of the number of routing iterations r on MNIST for a minimal capsule network where the PrimaryCaps layer has only $O = 1$ capsule types and vectors have dimension $D^{[1]} = 2$ while the ClassCaps layer vectors have $D^{[2]} = 4$ elements. This network has only 65k parameters, which helps isolating the effect of r, whereas the architecture in [1] has 6M parameters (8.2M with the decoder).

In Fig. 4.3 we report the learning curves for FR and RA. For FR, we train a new DR-CapsNet from scratch for each and every value of r. In the case of RA, instead, we train one model only, where we gradually increase the number of routing iterations (when the network loss reaches a plateau). We plot the best loss and accuracy values for every r. Fig. 4.3 shows that as r increases, the proposed RA enables decreasing loss that reflects into higher classification accuracy. Conversely, with a fixed routing strategy, the loss function diverges as r increases. We explain the gap between the two loss curves with the following hypothesis. Each iteration of the routing algorithm strengthens or weakens the connections between a capsule of the primary layer and all the capsules of the digit layer. Therefore, imposing high r for all the training epochs leads the DR-CapsNet to be overconfident on its predictions on the link strengths, preventing the network form learning the correct connections between the capsules.

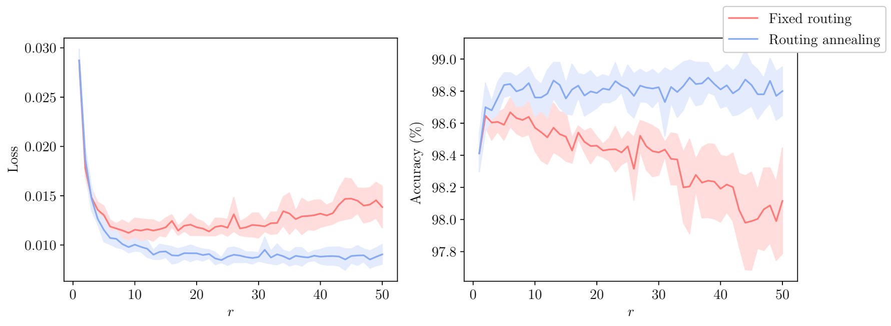
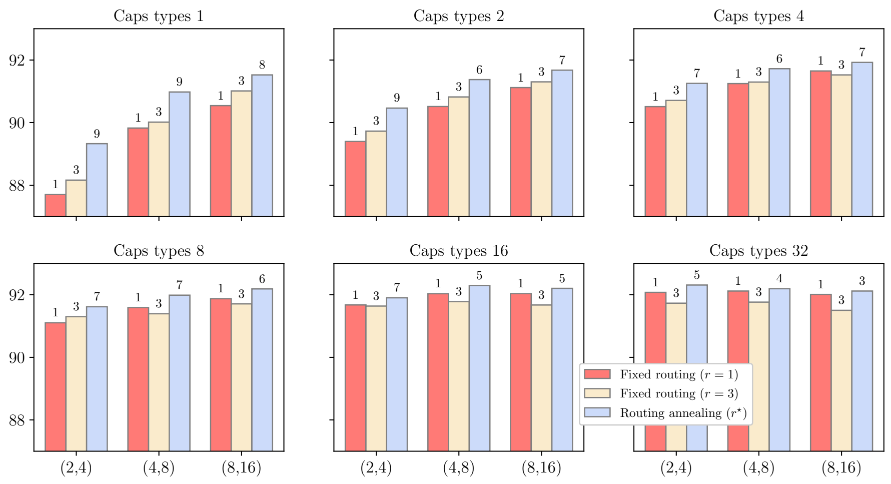


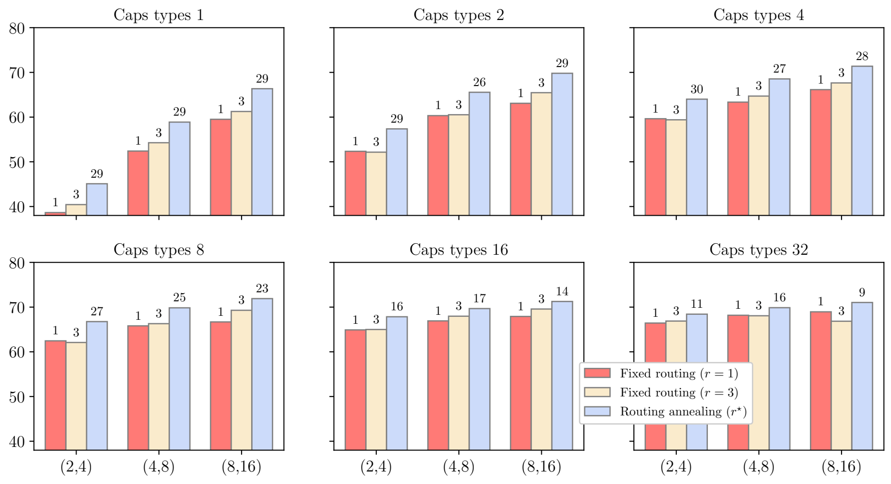
Figure 4.3: Loss function (left) and classification accuracy (right) on MNIST test set for a DR-CapsNet with $O = 1$, $D^{[1]} = 2$, $D^{[2]} = 4$ (means and stds of 5 seeds) as a function of the number of routing iterations r.

4.4.3 *Results*

Next, we experiment with the more complex datasets Fashion-MNIST and CIFAR-10. Fig. 4.3 showed that RA performs better than the fixed routing reference for large r values. For fixed routing experiments we only consider $r = 1$ and $r = 3$, as done in much of the recent literature. Fig. 4.4a and 4.4b show that RA performs better than fixed routing (both $r = 1$ and $r = 3$) in all the settings. Such experiments brought us to the following observations. First, coherently with our previous findings on MNIST, RA delivers the most appreciable gains when the network can afford only few learnable parameters. We recall here that for each capsule we have a matrix of weights $\boldsymbol{W}_{ij}$ and these matrices have shapes $D^{[1]} \times D^{[2]}$, namely the dimensions of the capsule vectors. This means that the number of capsules types T and their dimensions, along with the convolutional layers, drive the number of parameters of the network. This behaviour can be explained observing



(a) Fashion-MNIST



(b) CIFAR10

Figure 4.4: Classification accuracy (%) on Fashion-MNIST (a) and CIFAR-10 (b) test set for different capsule types T and dimensions $(D^{[1]}, D^{[2]})$. On top of each bar it is shown the number of iterations r used during training/inference, for RA it is shown the median value of $r^\star$.

that finding agreements between many high-dimensional capsule is not trivial. Running more iterations of the routing algorithm tends to polarize the coupling coefficients, namely the link strengths between capsules, such that it results in a simple route where each primary capsule sends its output to only one digit capsule. Therefore, when there are a lot of capsules, introducing some level of uncertainty with a low value of r helps the network to not be overconfident on its predictions and to not overfit on the training data. As a matter of fact, Fig. 4.4a shows that with 32 capsule types of 8-dimensional primary capsules and 16-dimensional digit capsules, our proposed method RA finds $r^\star = 3$, namely the value used in the original formulation of CapsNets in [1]. Second, in high-dimensional settings the same conclusions about routing as in [41] and in [40] hold for the fixed routing procedure, which achieves higher accuracy with $r = 1$ than $r = 3$. Nevertheless, RA always achieves better performance in all cases, sometime even with fewer routing iterations. Third, Fig. 4.4b shows that $r^\star$ for CIFAR-10 is not the same as for Fashion-MNIST in Fig. 4.4a for identical network conditions. This means that despite $r^\star$ differs from dataset to dataset, nevertheless our method can find it.

4.5 SUMMARY

In this chapter we presented a novel training technique for iterative routing-based CapsNets where the number of iterations is iteratively found at training time rather than being fixed. We also showed experiments on DR-CapsNets in settings with a different number of capsule types and their dimensions, namely the network capacity in terms of trainable parameters, and on several datasets. We showed that this value depends heavily on the size of the network and the dataset used. Typically, the smaller the network, the higher the number of iterations it requires to improve its generalization capability. Given the potentiality of our technique, in future works, we plan to apply RA on more complex and sophisticated routing algorithms such as EM routing [26].

5 TOWARDS EFFICIENT CAPSULE NETWORKS

The research findings and insights presented in this chapter were originally published in the following work: **Renzulli, R.** "Towards Efficient Capsule Networks." In: *2022 IEEE International Conference on Image Processing (ICIP)*. 2022, pp. 2801–2805.

5.1 INTRODUCTION

From the moment NNs dominated the scene for image processing, the computational complexity needed to solve the targeted tasks skyrocketed: against such an unsustainable trend, many strategies have been developed, ambitiously targeting performance's preservation. Promoting sparse topologies, for example, allows the deployment of deep neural networks models on embedded, resource-constrained devices. As mentioned in previous chapters, CapsNets were proposed to over-

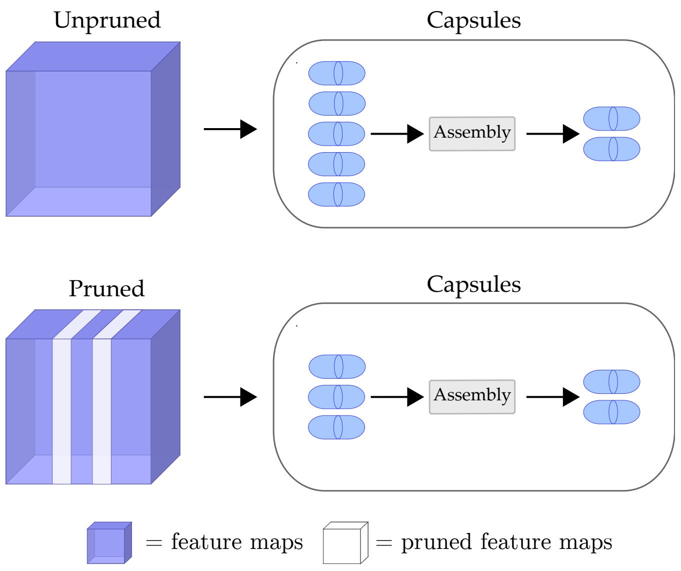


Figure 5.1: A standard CapsNet with many capsules (top). We exploit structured pruned features (bottom) to reduce the number of capsules.

come the shortcomings of CNNs. Despite CapsNets being successfully applied to various tasks [49, 50], the research is still in its early stages and CapsNets are far from replacing CNNs as state-of-the-art neural architectures due to their low scalability. Their applicability is limited because stacking multiple capsule layers increases the computational effort [51, 52]. Applying CapsNets on high-resolution data with complex backgrounds is also critical: stacking many capsule layers or adding more capsules can lead to instability during training because of the delicate routing algorithm [34]. However, the problem of over-parametrization and the difficulty of deploying networks in scenarios with limited memory and resources is not peculiar only to CapsNets, but also to CNNs. To this end, pruning methods (which reduce the size and complexity of a neural network model) have gained more and more attention lately [53]. Learning sparse topologies can effectively reduce the network footprint and speed up execution with negligible performance loss [54]. So, why not exploit these techniques also for CapsNets? Reducing their complexity, especially in early layers, will enable their deployment on complex datasets. In this chapter, we exploit structured pruning approaches to reduce the complexity of CapsNets without performance loss and test them on data with non-straightforward background such as CIFAR-10 and high dimensional data such as Tiny ImageNet. Fig. 5.1 provides an overall idea of the functioning of our approach. We show how employing structurally sparse backbones, extracting features in early layers for CapsNets, affects the number of capsules in the overall network. Our experiments show that reducing the complexity of the backbone is an effective way to achieve high generalization with less memory requirements, energy consumption, training and inference time.

5.2 METHODOLOGY

In this section, we show first how to extract primary capsules with a backbone model, then we propose a method to reduce the complexity of a capsule network with pruned backbones.

5.2.1 *Primary capsules extraction*

Following the work of [30, 40], we extract I primary capsule activity vectors by means of a backbone network. With this approach, we can build deep capsule networks in an efficient way since the input space of the capsule network is significantly reduced. Fig. 5.2 shows the

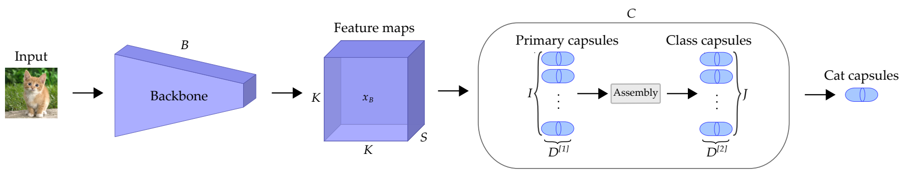


Figure 5.2: The architecture used in this chapter is composed by a backbone part and a capsule part. Primary capsules space has 8 real dimensions while class capsules are 16-dimensional vectors ($D^{[1]} = 8$ and $D^{[2]} = 16$).

architecture used in this chapter. We can build a CapsNet on top of the backbone by replacing the last two layers by a primary capsule (PrimaryCaps) and fully-connected capsule (ClassCaps) layers, respectively. Note that PrimaryCaps is a convolutional capsule layer with I channels of convolutional $D^{[1]}$ capsules, namely each capsule contains $D^{[1]}$ convolutional units with a $K \times K \times S$ kernel. The kernel size used is wide as the spatial dimensions of the output of the backbone network, which means the number of capsules is dynamically reduced. In our experiments we use backbones both as fixed feature extractor, namely we freeze their weights except for the capsule layer, and as trainable feature extractor, where we finetune all layers.

5.2.2 *Effect of pruned backbones to capsule layers*

In order to reduce the computational complexity of CapsNets as in Fig. 5.2, one possible strategy is to deploy pruned backbones. Let us define the backbone network $\mathcal{B}(\boldsymbol{x}_\mathcal{M}, \boldsymbol{w}_\mathcal{B})$ where $\boldsymbol{w}_\mathcal{B}$ are the backbone parameters and $\boldsymbol{x}_\mathcal{M}$ is the input; it produces as output the tensor $\boldsymbol{x}_\mathcal{B} \in \mathbb{R}^{K\times K\times S}$. Then, tensor $\boldsymbol{x}_\mathcal{B}$ is fed into the CapsNet $\mathcal{C}(\boldsymbol{x}_\mathcal{B}, \boldsymbol{w}_\mathcal{C})$, where $\boldsymbol{w}_\mathcal{C}$ are the capsule parameters. We can concatenate the two parts expressing the end-to-end model $\mathcal{N}$ as

$$\mathcal{N}(\boldsymbol{x}_\mathcal{M}, \boldsymbol{w}_\mathcal{B}, \boldsymbol{w}_\mathcal{C}) = \mathcal{C}\left[\mathcal{B}(\boldsymbol{x}_\mathcal{M}, \boldsymbol{w}_\mathcal{B}), \boldsymbol{w}_\mathcal{C})\right]. \tag{5.1}$$

As we saw in Sec. 5.2.1, $D^{[1]}$-dimensional primary capsules poses $\boldsymbol{u}_i$ are built upon convolutional layers. We define the number of primary capsules as

$$I = \left\lfloor \frac{S}{D^{[1]}} \right\rfloor. \tag{5.2}$$

From this, we observe that reducing the complexity of the backbone would result in the overall reduction of the complexity for the entire model $\mathcal{N}$, and towards this end pruning has already proved to be an effective approach [53, 54].
Pruning approaches can be divided into two groups. *Unstructured* pruning methods aim at minimizing the cardinality $\|\boldsymbol{w}\|_0$ of the parameters in the model, regardless the output topology [55–57]. On the other hand, *structured* approaches drop groups of weight connections entirely, such as entire channels or filters, imposing a regular pruned topology [53, 58]. As an effect, they minimize the cardinality of some i-th intermediate layer's output $\|\boldsymbol{x}_i\|_0$. Bragagnolo *et al.* [59] showed that structured sparsity, despite removing significantly less parameters from the model, yields lower model's memory footprint and inference time. When pruning a network in a structured way, a simplification step which practically reduces the rank of the matrices is possible; on the other side, encoding unstructured sparse matrices lead to representation overheads [54].
Considering the structure of the primary capsules layer, we are interested in the reduction of the cardinality for the input of the primary capsules $\boldsymbol{x}_\mathcal{B}$. Indeed, as we can see in equation 5.2, the minimization of $\|\boldsymbol{x}_\mathcal{B}\|_0$ results in a proportional reduction of S, which in turn reduces I,

resulting in a reduction of the votes and the complexity of the routing algorithm.

5.3 EXPERIMENTS

We performed several experiments with different backbones on CIFAR-10 and Tiny ImageNet. We choose ResNets and MobileNets models since they are used as common baselines in traditional computer vision tasks. In fact, they offer a good compromise between both efficiency and performance. As CapsNet architecture, we employed DR-CapsNet introduced in [1]. Furthermore, they are mainly composed of convolutional layers so it is easy to replace the first convolutional layer in a DR-CapsNet with these models.

5.3.1 *Experimental Setup*

We resized CIFAR-10 images to 64×64 resolution so as to add a deeper backbone network as in [34]. We used the 5% of the training set as validation set for hyper-parameters tuning. We also run experiments on Tiny Imagenet (224×224 images, 200 classes). Here we used 10% of the training set as validation set and the original validation set as test set. Our experiments consider several flavors of the architecture in Fig. 5.2 with different number of primary capsules I, backbones (ResNet-50 e MobileNetV1) and training methods (whether the backbone is finetuned or not). We train all the networks minimizing a *margin loss* [1] with the Adam optimizer and a batch size of 128. The experiments were run on a Nvidia Ampere A40 equipped with 48GB RAM, and the code uses PyTorch 1.10. We also used PyNVML library to compute the GPU memory consumption. As pretrained pruned backbones, we have used both ResNet-50 and MobileNetV1 pruned with EagleEye [58]. EagleEye is a state-of-the-art, open-source, structured pruning strategy which identifies relevant features at the level of intermediate outputs $\boldsymbol{x}_i$.

5.3.2 *Results and discussion*

The main results are shown in Fig. 5.3. Each configuration differs from the dataset used, percentage of pruned parameters in the backbone network (which affects the number of primary capsules), and training method (backbone finetuned or not). In Fig. 5.3 we report the accuracy as a function of the total number of FLOPS (of both backbone and capsule part) for training the network. We can see that when the backbone is freezed (in red) the networks achieve poor performances, but as the number of pruned backbone FLOPS decreases, namely the number of primary capsules is higher, the accuracy increases. The poor performance can be explained by the fact that the output features

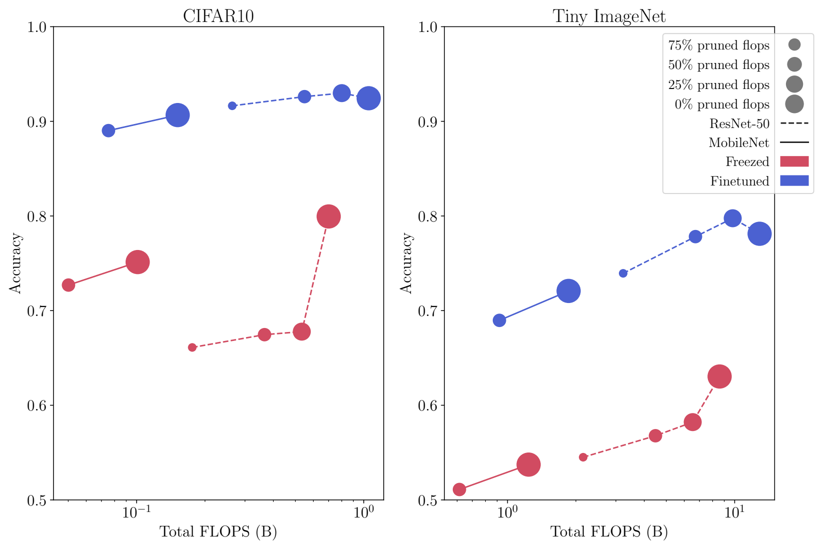


Figure 5.3: Accuracies using ResNet-50 and MobileNetV1 as backbones for CIFAR-10 (left) and Tiny ImageNet (right) with several pruned FLOPS configurations.

$x_{\mathcal{B}}$ of the backbone are not directly optimized to represent objects poses. Therefore, they can not be used directly as input to capsule layers. To overcome such a drawback, we can either add more primary capsules or train the whole network. In fact, when the backbone is finetuned (in blue) the performance is higher since the backbone is able to map more suitable features to the capsules space. We can see in this case that adding more capsules does not always lead to an improvement in the classification accuracy. In Tab. 5.1 the performance for Tiny ImageNet with a pretrained ResNet-50 backbone is reported. Employing a backbone with 50% of pruned FLOPS yields similar accuracy as with a full backbone, with less GPU memory consumption and training time. With 25% of pruned FLOPS we can even improve the accuracy compared to the unpruned one. The partial improving in the performance in a low pruning regime, despite being at a first glance surprising, is not new to the literature and it is twofold. First, Rajasegaran *et al.* [34] shows that building too many capsules inhibits learning because the coupling coefficients are too small, preventing the gradient flow. Furthermore, Han *et al.* observed such a phenomenon for unstructured pruning [55]. Recently, it has been showed that the average entropy in the bottleneck layer for pruned backbones (in our case, $x_{\mathcal{B}}$) is higher than in non-pruned ones [60]: this results in the propagation of less specific and more general information, which prevents features overfit, on top of which capsules layers can extract much more accurate information.

Backbone pruned FLOPS (%)	Bottleneck size	Primary caps	Total FLOPS (B)	GPU memory consumption (GB)	Training time (epoch, s)	Accuracy
0	2048	256	12.9	47.02	462	0.78
25	1970	246	**9.8**	**43.50**	**450**	**0.80**
50	1947	243	**6.7**	**39.42**	**410**	**0.78**
75	944	118	3.2	20.11	214	0.74

Table 5.1: Performances using ResNet-50 as backbone (finetuned, TinyImageNet).

5.4 SUMMARY

In this chapter, we presented a method to improve the scalability and reduce the computational effort of CapsNets on complex datasets deploying backbones with structured sparsity. CapsNets with many capsules are difficult to train: in such a scenario, the routing algorithm, a key mechanism for CapsNets, struggles to find the necessary agreements between capsules. Therefore, employing pruned backbones (so fewer capsules) for high-resolution datasets leads to competitive results with no performance loss. We also showed how extracting features in early layers with sparse networks improves the efficiency, memory consumption, inference, and training time of the overall CapsNet. Therefore, this chapter opens the way to apply CapsNets on resource-constrained devices. Future works involve specific methods concerning pruning capsule layers in a structured way.

6 CAPSULE NETWORKS DO NOT NEED TO MODEL EVERYTHING

The research findings and insights presented in this chapter are currently under review: **Renzulli, R.** "Capsule Networks Do Not Need to Model Everything." Submitted to the AAAI Conference on Artificial Intelligence. 2024.

6.1 INTRODUCTION

As mentioned in the previous chapters, CapsNets can learn hierarchical representations from the input images thanks to a routing algorithm, which carves out a parse tree composed of part-object relationships. They are also more robust to viewpoint changes and affine transformations than traditional CNNs.

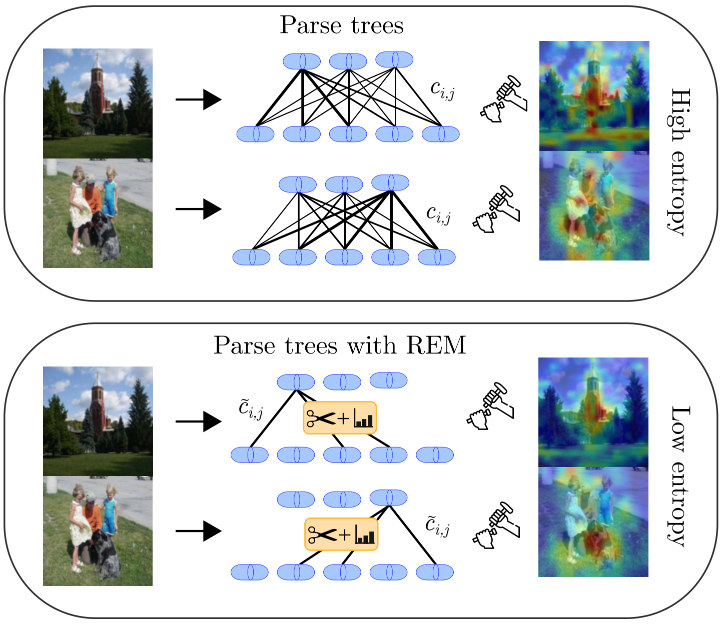


Figure 6.1: A hierarchy of parts (parse tree) is *carved out* of a CapsNet, like a sculpture is carved from a rock. REM enriches a CapsNet with pruning and quantization. REM, with low entropy configurations, reduces modeling noisy backgrounds and extracts more discriminative features.

This chapter explores a drawback of CapsNets identified by Sabour et al. [1]: these networks perform better when they can model all elements present in an image; therefore, larger network sizes are required to handle image clutter, such as intricate backgrounds or complex objects. Therefore, the entropy of the connections of the parse trees is high. Recently, many pruning methods were applied to CNNs to reduce the complexity of the networks, enforcing sparse topologies [56, 62, 63]. These considerations lead us to the following question: *is it possible to tailor pruning and quantization approaches to not only reduce the size of CapsNets but aid the extraction of more discriminative features (namely fewer parse trees)?*

Here we introduce REM (Routing Entropy Minimization), which moves some steps toward extracting more succinct relationships from CapsNets with negligible performance loss and fewer parameters. To encourage this, as shown in Figure 6.1, we impose sparsity and entropy constraints. In low pruning regimes, noisy couplings cause the entropy to increase considerably. By contrast, in high sparsity regimes, pruning can effectively reduce the overall entropy of the connections of the parse tree-like structure encoded in a CapsNet. We collect the coupling coefficients studying their frequency and cardinality, observing lower intra-class conditional entropy: the pruned version *adds a missing explicit prior* in the routing mechanism, grounding the coupling of the unused capsules and disallowing fluctuations under the same baseline performance on the validation/test set. This implies that the parse trees are significantly less, hence more stable for the pruned models, focusing on more relevant features (per class) selected from the routing mechanism. We also introduce two visualization methods of the parse trees based on the type of last capsule layer.

The rest of the chapter is organized as follows: in Section 6.2 we describe our technique REM, in Section 6.3 we show the effectiveness of our approach on many datasets and finally we discuss the summary of the proposed approach.

6.2 METHODOLOGY

The coupling coefficients computed by the routing mechanism model are the part-whole relationships between capsules of two consecutive capsule layers. Assigning parts to objects (namely, learning how each object is composed) is challenging. One of the main goals of the routing algorithm is to extract a parse tree of these relationships. For example, given the ξ-th input of class j, an ideal parse tree for a capsule i detecting one of the discriminative parts of the entity in the input ξ would ideally lead to

$$\boldsymbol{c}_{i,:}^{[L-1](\xi)} = \mathbb{1}_{\hat{y}^{(\xi)}}, \tag{6.1}$$

where $\mathbb{1}_{\hat{y}^{(\xi)}}$ is the one-hot encoding for the target class $y^{(\xi)}$ of the ξ-th sample. This means that the routing process can carve a parse tree out of the CapsNet which explains perfectly the relationships between parts and wholes. One of the problems of this routing procedure is that there is no constraint on how many parse trees there should be.

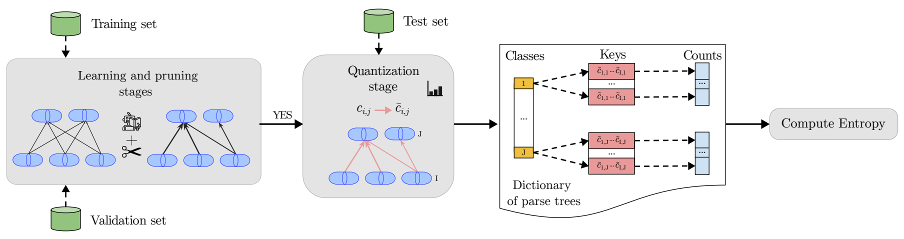

Training set
Learning and pruning stages
Validation set
YES
Test set
Quantization stage
$c_{i,j} \rightarrow \tilde{c}_{i,j}$
Classes
Keys
Counts
Dictionary of parse trees
Compute Entropy

Figure 6.2: Pipeline of REM. After the learning stage, the coupling coefficients of the CapsNet are quantized, and the obtained parse trees are collected in a dictionary.

In this section, we present our technique REM, first showing how to extract a parse tree and then how to extract fewer parse trees. The pipeline of our method is depicted in Figure 6.2.

6.2.1 *Parse Trees Extraction*

Once we have a trained CapsNets model, to analyze the routing mechanism, we extract all the possible routing coupling coefficients and build a parse tree. Towards this end, we want to define a metric that helps us decide if the relationships captured by the routing algorithm resemble a parse tree or not. Therefore, we organize the coupling coefficients into *associative arrays* so that we can compute the number of occurrences of each coupling sequence to measure the entropy of the whole dictionary. We refer to this entropy as the *simplicity* of the parse tree. In other words, we refer to the number of keys in the dictionary as the number of unique parse trees that can be carved out from the input dataset. In the following paragraphs, we explain how to generate these sequences by discretizing the coupling coefficients and how to create the dictionary.

Quantization. During the quantization stage, we first compute the *continuous* coupling coefficients $c_{i,j}^{[l](\xi)}$ for each ξ-th input example. It should be noted that these are the coupling coefficients obtained after the forward pass of the last routing iteration. Then, we quantize them into K discrete levels through the uniform quantizer $q_K(\cdot)$, obtaining

$$\tilde{c}_{i,j}^{[l](\xi)} = q_K(c_{i,j}^{[l](\xi)}). \tag{6.2}$$

We choose the lowest K such that the accuracy does not deteriorate. Here, we will refer to CapsNet+Q as trained CapsNet where the coupling coefficients are quantized.

Entropy. Given the quantized coupling coefficients of a CapsNet+Q, we can extract the parse tree (and create a dictionary of parse trees) for each class j, where each entry is a string composed of the quantization indices of the coupling coefficients $\tilde{\boldsymbol{c}}_{:,j}^{[L-1](\xi)}$. Given a dictionary for the

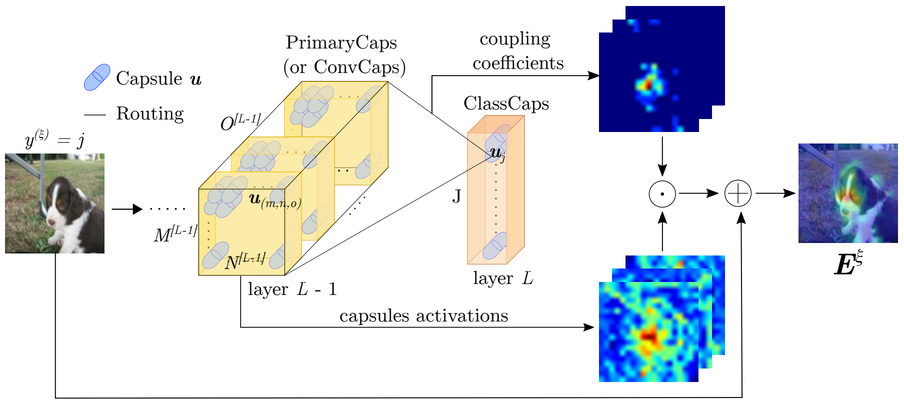


Figure 6.3: Extraction of a saliency map.

coupling coefficients of a CapsNet+Q, we can compute the entropy for each class as

$$\mathbb{H}_j = -\sum_{\xi} \{\mathbb{P}(\tilde{\boldsymbol{c}}_{:,j}^{[L-1](\xi)} \mid y^{(\xi)} = j) \cdot \log_2 [\mathbb{P}(\tilde{\boldsymbol{c}}_{:,j}^{[L-1](\xi)} \mid y^{(\xi)} = j)]\}, \tag{6.3}$$

where $\mathbb{P}(\tilde{\boldsymbol{c}}_{:,j}^{[L-1](\xi)} \mid y^{(\xi)} = j)$ is the frequency of occurrences of a generic string ξ for each *predicted* class $y^{(\xi)}$. When stacking multiple capsule layers, we apply the quantization stage to each layer and compute the entropy values on the last layer L. Finally, the entropy of a dictionary for a CapsNet+Q on a given dataset is the average of the entropies $\mathbb{H}_j$ of each class

$$\mathbb{H} = \frac{1}{J} \sum_j \mathbb{H}_j. \tag{6.4}$$

Intuitively, the lower Equation 6.4, the fewer the number of parse trees carved out from the routing algorithm.

6.2.2 *Parse Trees Visualization*

As mentioned in Section 3.4, there are three types of capsule layers. Primary capsules and convolutional capsules have a spatial connotation, while fully-connected capsules do not. We describe here two different visualization methods of the extracted parse trees based on the type of the capsules in layer $L-1$. In both scenarios, we follow only the connections starting from the capsule representing the predicted object label, we refer to this method as “backtracking”.

Primary and convolutional capsules. In Figure 3.4, we see that primary and convolutional capsules are organized in three dimensions and located in a $M \times N \times O$ grid structure. Therefore, in this scenario, we exploit the coupling coefficients $c_{i,j}^{[L-1]}$ for the predicted class j as visual attention built-in explanation to carve out the part structure discovered by a capsule model. We say built-in explanation because we rely only on the forward pass of the network. While Grad-CAM [64] weighs neurons activations by gradients computed in the backward pass, we weigh each capsule activation by the corresponding coupling coefficient. We refer to $\boldsymbol{u}^{(\xi)}_{(m,n,o)}$ to indicate the pose of the capsule in position (m, n, o) for a given input ξ. With $\tilde{c}^{[L-1](\xi)}_{(m,n,o),j}$, we refer to the quantized coupling coefficient between a capsule in position (m, n, o) of layer $L-1$ and the predicted capsule-class j of layer L for a given input ξ. We denote with $\boldsymbol{E}^{(\xi)}$ the saliency map for a given input ξ. We follow [65], where the coupling coefficients of the predicted class j of a trained model for a given input are used as an attention matrix. Unlike [65], we also weight each coupling coefficient by the activity of capsule i. Therefore, each element of the saliency map is computed as

$$E^{(\xi)}_{(m,n)} = \frac{1}{O}\sum_{o}(\|\boldsymbol{u}^{[L-1](\xi)}_{(m,n,o)}\| \cdot \tilde{c}^{[L-1](\xi)}_{(m,n,o),j}). \tag{6.5}$$

Then we upsampled the saliency map to the input size with the bilinear method. Figure 6.3 depicts our method's visualization of extracting a saliency map from an input image given a CapsNet model.

Fully-connected capsules. CapsNets can also be organized in fully connected capsule layers. The saliency map method can not be employed in this scenario, as capsules in layer $L-1$ do not have a spatial connotation. Therefore, similarly to Mitterreiter et al. [66], we propose a different visualization of the parse tree. Each capsule is a node where the color is its activation probability, and the strength of the connection is depicted with fuzzy edges of different strokes. We also denote with $deg^{-}(\boldsymbol{u}_j)$ the indegree of a capsule j in layer l, which is the number of coupling coefficients starting from capsules i in layer $l-1$ that are above some threshold. We only show the connections from capsule j to capsules k in layer $l+1$ if $deg^{-}(\boldsymbol{u}_j)$ is at least 1.

6.2.3 *Unconstrained Routing Entropy*

In this subsection, we are going to more-formally analyze the distribution of the coupling coefficients

$$c_{i,j} = \frac{e^{b_{i,j}+\sum_{r=1}^{r_T}\boldsymbol{u}_j^{(r)}\boldsymbol{u}_i\boldsymbol{W}_{i,j}}}{\sum_k e^{b_{i,k}+\sum_{r=1}^{r_T}\boldsymbol{u}_k^{(r)}\boldsymbol{u}_i\boldsymbol{W}_{i,k}}}, \tag{6.6}$$

where r_T indicates the target routing iterations. We suppress the indices ξ and l for abuse of notation. Let us evaluate the $c_{i,j}$ over a non-yet trained model: as we also saw in Section 6.2.1, we have

$$c_{i,j} \approx \frac{1}{J}\ \forall i, j. \tag{6.7}$$

When updating the parameters, following [41], we have

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{W}_{i,j}} = \left[\frac{\partial\mathcal{L}}{\partial\boldsymbol{u}_j}\frac{\partial\boldsymbol{u}_j}{\partial\boldsymbol{s}_j}\cdot c_{i,j} + \sum_{m=1}^{M}\left(\frac{\partial\mathcal{L}}{\partial\boldsymbol{u}_m}\frac{\partial\boldsymbol{u}_m}{\partial\boldsymbol{s}_m}\cdot\hat{\boldsymbol{u}}_{i,m}\frac{\partial c_{i,m}}{\hat{\boldsymbol{u}}_{i,m}}\right)\right]\cdot\boldsymbol{u}_i, \tag{6.8}$$

where we can have the gradient for $\boldsymbol{W}_{i,j} \approx 0$ in a potentially-high number of scenarios, despite $c_{i,j} \neq \{0, 1\}$. Let us analyze the simple case in which we have perfect outputs, matching the ground truth. Hence we are close to a local (or potentially the global) minimum of the loss function:

$$\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_m} \right\|_2 \approx 0 \ \forall m. \tag{6.9}$$

Looking at Equation 3.2, we see that the right class is chosen, but given the squashing function, we have as an explicit constraint that, given the j-th class as the target one, we require

$$\left\| \boldsymbol{u}_j \right\|_2 \gg \left\| \boldsymbol{u}_m \right\|_2 \ \forall m \neq j \tag{6.10}$$

on the $\boldsymbol{W}_{i,j}$, which can be accomplished in many ways, including:

- having sparse activation for the primary capsules $\boldsymbol{u}_i$: in this case, we have constant $\boldsymbol{W}_{i,j}$ (typically associated with no-routing based approaches); however, we need heavier deep neural networks as they have to force sparse signals already at the output of the primary capsules. In this case, the coupling coefficients $c_{i,j}$ are also constant by definition;
- having sparse votes $\hat{\boldsymbol{u}}_{i,j}$: this is a combination of having both primary capsules and weights $\boldsymbol{W}_{i,j}$ enforcing sparsity in the votes and the typical scenario with many routing iterations.

Having sparse votes, however, does not necessarily result in having sparse coupling coefficients. According to Equation 3.4.1, the coupling coefficients are multiplied with the votes, obtaining the output capsules. The distribution of the coupling coefficients requires Equation 6.10 to be satisfied only: if $\boldsymbol{W}_{i,j}$ is not sparsely distributed, we can still have sparse votes. However, this is the main reason we observe high entropy in the coupling coefficient distributions: as the votes, $\hat{\boldsymbol{u}}_{i,j}$ are implicitly sparse (yet also disordered, as we are not explicitly imposing any structure in the coupling coefficients distribution), the model is still able to learn, but it finds a typical solution where $c_{i,j}$ are not sparse. However, we would like sparsely distributed, recurrent couplings to the same j-th output caps $\boldsymbol{c}_{:,j}$, establishing stable relationships between the features extracted at the primary capsule layer.

Minimizing explicitly the entropy term in Equation 6.3 is an intractable problem due to the non-differentiability of the entropy term and the quantization step (in our considered setup) and due to the huge computational complexity to be introduced at training time. Hence, we can try to implicitly enforce routing entropy minimization by forcing a sparse and organized structure in the coupling coefficients. Towards this end, one efficient solution is to enforce sparsity in the $\boldsymbol{W}_{i,j}$ representation by compelling a vote between the i-th primary capsule and the j-th output caps to be exactly zero for any input, according to Equation 6.6

$$c_{i,j} = \frac{1}{\sum_k e^{b_{i,k}+\sum_{r=1}^{t} \boldsymbol{u}_k^{(r)} \boldsymbol{u}_i \boldsymbol{W}_{i,k}}}. \tag{6.11}$$

In this way, having a lower variability in the $c_{i,j}$ values (and hence building more stable relationships between primary and output capsules), straightforwardly, we are also explicitly minimizing the entropy of the quantized representations for the coupling coefficients. In the following subsection, we will tailor a sparsity technique to accomplish such a goal.

6.2.4 *Enforcing REM with Pruning*

CapsNets are trained via standard back-propagation learning, minimizing some loss functions like margin loss. Our ultimate goal is to assess to what extent a variation of the value of some parameter θ would affect the error on the network output. In particular, the parameters not affecting the network output can be softly pushed to zero, meaning we can apply an L^2 penalty term. Several approaches have been proposed, especially in recent years [63, 67, 68]. One recent state-of-the-art approach, LOBSTER [57] proposes to penalize the parameters by their gradient-weighted L^2 norm, leading to the update rule

$$\theta^{t+1} = \theta^t - \eta G\left[\frac{\partial \mathcal{L}}{\partial \theta^t}\right] - \lambda \theta^t \mathrm{ReLU}\left[1 - \left|\frac{\partial \mathcal{L}}{\partial \theta^t}\right|\right], \tag{6.12}$$

where $G\left[\frac{\partial \mathcal{L}}{\partial \theta^t}\right]$ is any gradient-based optimization update (for SGD it is the plain gradient, but other optimization strategies like Adam can be plugged) and η, λ are two positive hyper-parameters.

Such a strategy is particularly effective on standard convolutional neural networks and easy to plug into any back-propagation-based learning system. Furthermore, LOBSTER is a regularization strategy that can be plugged at any learning stage, as it self-tunes the penalty introduced according to the learning phase. This non-intrusiveness in the complex and delicate routing mechanism for CapsNets resulted in a suitable choice to enforce REM.

6.3 EXPERIMENTS AND RESULTS

This section reports the experiments and results we performed to test REM. We first show the effects on the MNIST dataset with dynamic routing CapsNets (DR-CapsNets), reporting how the entropy and the accuracy values change during training. Then, we test REM on more complex datasets such as Fashion-MNIST, CIFAR-10, SVHN, Dogs vs.

Model	Accuracy	Entropy
DR-CapsNet$_1$+Q	97.53	1.83
DR-CapsNet$_{190}$+Q	99.55	9.41

Table 6.1: Classification results (%) and entropy of two DR-CapsNets+Q at epochs 1 and 190 on MNIST (test set).

Cats [69] and Imagenette [70]. We used the same architectures configurations and augmentations described in Sabour et al. [1]. We also conducted experiments applying our technique to γ-CapsNets [29], DeepCaps [34], Eff-CapsNets [71] to test the efficacy of REM to some other variants of capsule models, including different architectures, routing algorithms, and the number of trainable parameters. We used a custom architecture called Eff-ConvCapsNet for higher-resolution images such as Dogs vs. Cats and Imagenette. This model has three capsule layers (primary, convolutional and fully-connected) and shares a backbone network similar to Eff-CapsNets. We trained all models with five random seeds. We report the classification accuracy (%) and entropy (averages and standard deviations), the *sparsity* (percentage of pruned parameters, median), and the number of keys in the dictionary (median).

6.3.1 *Ablation Study*

To assess our REM technique, we analyze in-depth the benefits of pruning towards REM on the MNIST dataset. Despite its outdatedness, MNIST remains an omnipresent benchmark for CapsNets [1, 36, 41, 42, 72].

Choice of quantization levels. The routing algorithms used in the models employed in this paper are performed between two consecutive capsule layers. As we can see in Figure 6.4, the choice of the number of quantization levels K for the coupling coefficients computed by a routing algorithm of a DR-CapsNet affects the performance of the network. We select the value for K that achieves the best accuracy value with relatively low entropy. In this case, when K=11, DR-CapsNet+Q achieves 99.47% accuracy and 9.32 entropy, while DR-CapsNet+REM achieves 99.57% accuracy and 4.40 entropy.

Entropy at different epochs. Let us now generate the parse trees on the CapsNet model trained on the MNIST dataset, investigating their evolution during training. To this end, we compute the distribution of the coupling coefficients on the test set for two models: the model trained after the first epoch and the one that achieved the lowest loss value on the validation set. Fig. 6.5 shows the distributions for the two models and Table 6.1 reports the corresponding accuracy and entropy values. We can observe that after the first epoch, DR-CapsNet is far from optimality, both in terms of performance (accuracy of 97.4%) and

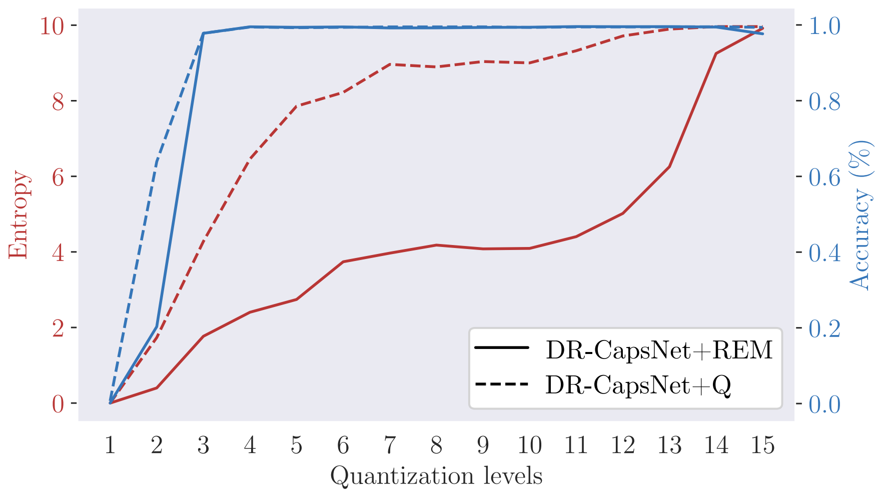


Figure 6.4: Entropy and accuracy for DR-CapsNet+Q and DR-CapsNet+Q with different quantization levels on MNIST (test set).

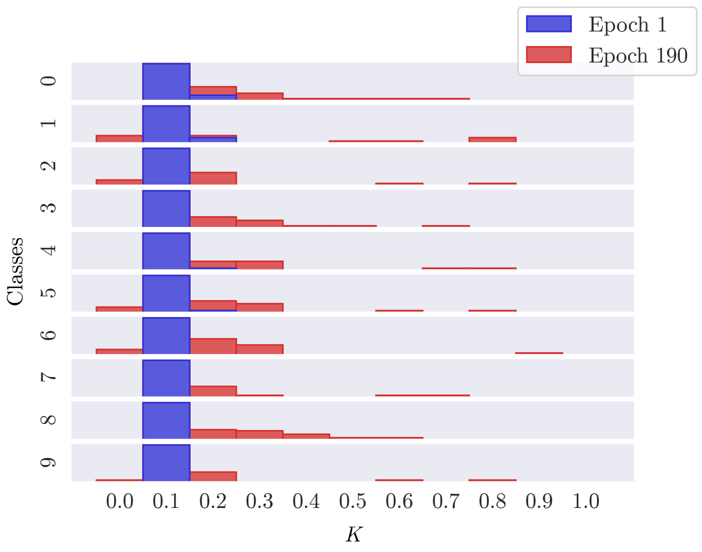


Figure 6.5: Coupling coefficients distributions for each class of two DR-CapsNets+Q at epochs 1 and 190 on MNIST (test set).

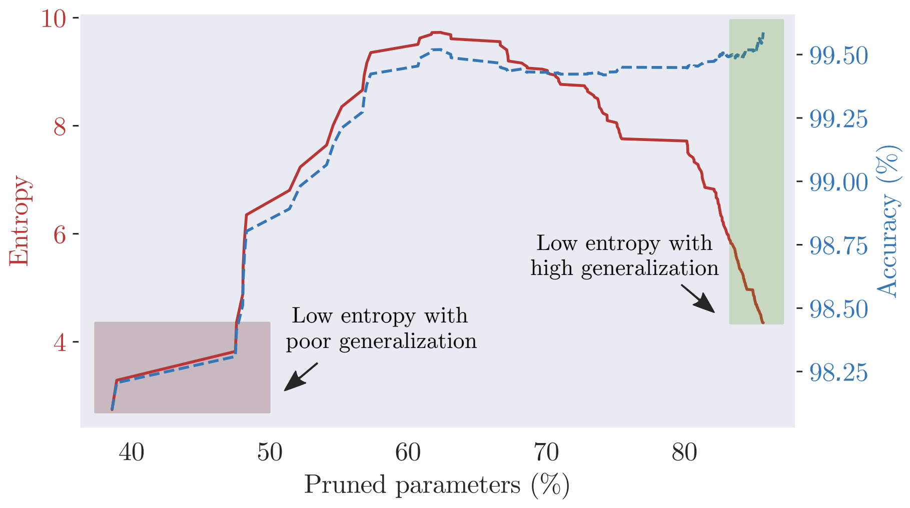


Figure 6.6: Accuracy and entropy vs. pruned parameters on MNIST (test set).

parse tree discriminability: indeed, all coupling coefficients are almost equal to the value selected for initialization, i.e., $1/J$, where J is the number of output capsules. On a given dataset, we target a model with high generalization but low entropy, namely a low number of extracted parse trees. Figure 6.6 shows how the entropy (red line) and classification accuracy (blue dotted line) change as the sparsity increases during training. At the beginning of the training stage, we can see that the entropy is low (1.83) because the routing algorithm has not yet learned to correctly discriminate the relationships between the capsules (97.53% accuracy). This effect is almost the same when we train a DR-CapsNet with $t = 1$ as Gu et al. [41], where its entropy is zero, but capsules are uniformly coupled. However, at the end of the training process, we can get a model trained with REM with higher performances (99.60% accuracy) and still low entropy (4.31). Figure 6.7 illustrates the distributions of the coupling coefficients for a DR-CapsNet+Q and a DR-CapsNet+REM following the method described in Section 6.2.1. We can see that the distributions of the DR-CapsNet+REM model are sparser than those for the DR-CapsNet+Q model.

Number of parse trees. Figure 6.8 shows the number of intra-class parse trees (collected in a dictionary) for DR-CapsNets+REM and a DR-CapsNets+Q, namely a DR-CapsNet where the quantization is applied without pruning the network during training. We can see that the number of keys in the dictionary for DR-CapsNets+REM is lower than the one for DR-CapsNets+Q for each class. Also, the entropy measure for DR-CapsNets+REM is lower than DR-CapsNets+Q; namely, REM has successfully extracted fewer parse trees on the MNIST test set.

Decoder. A CapsNet is typically composed of an encoder and a decoder part, where the latter is a reconstruction network with three fully connected layers [1]. In the previously-discussed experiments,

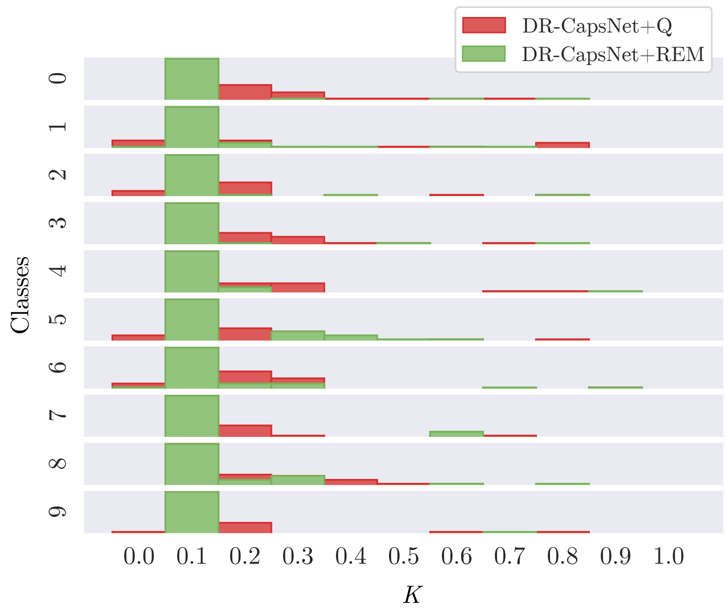


Figure 6.7: Coupling coefficients distributions on MNIST (test set).

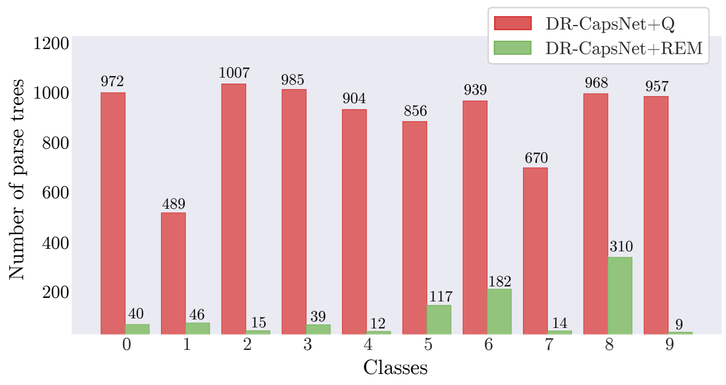


Figure 6.8: Number of keys for each class on MNIST (test set).

we have removed the decoder. One limitation of our work arises when computing the entropy of CapsNets trained with the decoder. Tables 6.2 and 6.3 report the classification results and entropies values when we trained the encoder and the decoder part together. We observed that the entropy of a DR-CapsNets+REM is almost the same as that of a DR-CapsNet+Q. Indeed, when the decoder is used, the activity vector of an output capsule encodes richer representations of the input. Sabour et al. [1] introduced the decoder to boost the routing performance on MNIST by enforcing the pose encoding a capsule. They also show that such perturbation affects the reconstruction when a perturbed activity vector is fed to the decoder. So capsule representations are *approximately equivariant*, meaning that even if they do not come with guaranteed equivariances, transformations applied to the input can still be described by continuous changes in the output vector. To verify if output capsules of a trained DR-CapsNet+REM without the decoder (so with low entropy) are still approximately equivariant, we stacked on top of it the reconstruction network, without training the encoder. The MNIST dataset's decoder comprises three fully connected layers of 512, 1024, and 784 neurons, respectively, with two RELU and a final sigmoid activation function. This network is trained to minimize the Euclidean distance between the image and the output of the sigmoid layer. We can see in Figure 6.9 that DR-CapsNets+REM with low entropy are still approximately equivariant to many transformations.

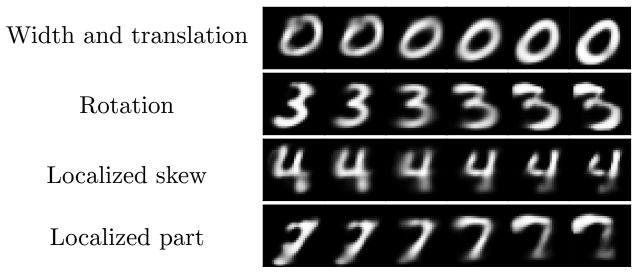


Figure 6.9: MNIST perturbation reconstructions of a frozen DR-CapsNet+REM.

Model	MNIST	F-MNIST	CIFAR-10
DR-CapsNet+Q	$99.58 \pm_{0.03}$	$92.57 \pm_{0.39}$	$72.40 \pm_{0.54}$
DR-CapsNet+REM	$99.63 \pm_{0.02}$	$92.76 \pm_{0.38}$	$76.00 \pm_{0.63}$

Table 6.2: Classification results with the decoder on MNIST, Fashion-MNIST, CIFAR-10 (test set).

Model	MNIST	F-MNIST	CIFAR-10
DR-CapsNet+Q	$9.88 \pm_{0.06}$	$8.49 \pm_{1.50}$	$4.55 \pm_{1.13}$
DR-CapsNets+REM	$9.40 \pm_{0.55}$	$6.15 \pm_{2.32}$	$3.85 \pm_{0.54}$

Table 6.3: Entropies for models trained with the decoder on Fashion-MNIST and CIFAR-10 (test set).

Other networks Table 6.4 shows the performances on MNIST of DR-CapsNets, γ-CapsNets, DeepCaps and Eff-CapsNets. We notice that γ-CapsNet and γ-CapsNet+REM have the lowest entropy values since γ-CapsNets employ a scaled-distance-agreement routing algorithm that enforces the single parent constraint. With our technique REM we can successfully lower the entropy even more.

6.3.2 *Experiments*

Considering the broad heterogeneity of proposed architectures and the adaptability of REM to other architectures and datasets, we have

Model	Parameters	Accuracy (%)	Sparsity (%)	Entropy
DR-CapsNet+Q	6.8M	$99.56 \pm_{0.03}$	–	$9.53 \pm_{0.54}$
DR-CapsNet+REM	0.9M	$99.56 \pm_{0.02}$	85.53	$4.16 \pm_{1.59}$
γ-CapsNet+Q	7.7M	$99.50 \pm_{0.07}$	–	$1.87 \pm_{1.38}$
γ-CapsNet+REM	0.8M	$99.50 \pm_{0.05}$	89.71	$1.34 \pm_{1.09}$
DeepCaps+Q	8.4M	$99.51 \pm_{0.24}$	–	$5.26 \pm_{2.00}$
DeepCaps+REM	2.4M	$99.61 \pm_{0.23}$	71.73	$3.10 \pm_{1.07}$
Eff-CapsNets+Q	161k	$99.55 \pm_{0.31}$	–	$4.38 \pm_{1.59}$
Eff-CapsNets+REM	43k	$99.58 \pm_{0.64}$	73.15	$2.60 \pm_{1.72}$

Table 6.4: Results for DR-CapsNets, γ-CapsNets, DeepCaps and Eff-CapsNets on MNIST (test set).

chosen to test DR-CapsNets γ-CapsNets, DeepCaps, Eff-CapsNets and Eff-ConvCapsNets to other datasets other than MNIST.

Datasets. We trained and tested CapsNets on: i) Fashion-MNIST, 28×28 grayscale images (10 classes); (ii) SVHN, 32×32 RGB images (10 classes); iii) CIFAR-10, 32×32 RGB images (10 classes); (iv) Dogs vs. Cats (images resized to 128×128); (v) Imagenette, a subset of 10 classes from the Imagenet dataset resized to 128×128. For MNIST, Fashion-MNIST, and CIFAR-10, we used 5% of the training set as a validation set. For Dogs vs. Cats, Imagenette and Tiny Imagenet, we used 10% of the training set as validation set. To test the robustness of novel azimuths on smallNORB, we train all models on 1/3 of the training data with azimuths of 0, 20, 40, 300, 320, and 340 degrees and test them on 2/3 of test data with remaining azimuths never seen during training. To test the robustness of our technique on novel elevations, we trained models on 1/3 of the training data with elevations of 30, 35, and 40 degrees from the horizontal. We tested on 2/3 of the test data with the remaining elevations. Finally, to test the robustness to affine transformations, we used expanded MNIST training and validation sets (40×40 padded and translated MNIST images) and the affNIST test set, in which each example is an MNIST digit with a random small affine transformation.

Model architectures. All models employed in this work were tested using the same architectures (number of layers, capsule dimensions, number of routing iterations, etc.) presented in the original papers. Therefore, for DR-CapsNets, γ-CapsNets, DeepCaps and Eff-CapsNets we used the same architectures configurations as in Sabour et al. [1], Peer et al. [29], Rajasegaran et al. [34], and Mazzia et al. [71] respectively. Eff-ConvCapsNet consists of three capsule layers stacked on top of a backbone network similar to Eff-CapsNets. We use the notation Conv(*output-channels*, *kernel*, *stride*) for standard convolutional layers. For convolutional capsule layers, the notation is ConvCaps(*output-capsule-dimension*, *output-capsule-types*, *kernel*, *stride*). For fully-connected capsule layers, we use FcCaps(*output-capsule-dimension*, *output-capsule-types*). The backbone network utilizes four standard convolutional layers Conv(32,5,2), Conv(64,3,1), Conv(64,3,1), and Conv(256,3,2), followed by a fifth depthwise convolutional layer Conv(256,3,1) to extract 16 primary capsule types of dimension 16. Then, a convolutional capsule layer ConvCaps(16,16,3,2) is stacked on top of primary capsules. Finally, a fully-connected capsule layer FcCaps(16, 10) (FcCaps(16, 2) for Dogs vs. Cats) with shared transformation matrices is stacked to output the class capsules. We used the dynamic routing method introduced by Sabour et al. [1].

To visualize the parse trees in multilayer DR-CapsNets, we stacked five capsule layers, 16 capsules for the primary capsule layer, 16 capsules for the three hidden layers and ten capsules for the last layer. Each capsule is composed by 8 neurons.

Training. For models without REM, we take the checkpoint that achieved the lowest loss on the validation set, while for models with REM we take the model on the last epoch. We checked the loss on the validation set and used an early-stop of 200 epochs. The models were trained on batches of size 128 using Adam optimizer with its PyTorch 1.12 default parameters, including an exponentially decaying learning rate factor of 0.99.

Quantization levels We used γ-CapsNets with 3 capsule layers as in Peer et al. [29]. For γ-CapsNets+Q and γ-CapsNets+REM we found $K = 11$ for the first two capsule layers and $K = 6$ for the last two. For DeepCaps+Q and DeepCaps+REM we used $K = 11$ for all the capsule layers where the number of routing iterations is greater than one. For Eff-CapsNets+Q and Eff-CapsNets+REM we also used $K = 11$. For Eff-ConvCapsNets+Q and Eff-ConvCapsNets+REM we used $K = 21$ on Dogs vs. Cats and Imagenette.

Generalization ability. As shown in Tables 6.5, a CapsNet+REM has a high percentage of pruned parameters with a minimal performance loss. So this confirms our hypothesis that CapsNets are over-parametrized. We also report the entropy of the dictionary of the last routing layer for the quantized models. The entropy is successfully lower for all datasets when REM is applied to all architectures, even with fewer parameters than CapsNets, such as Eff-CapsNets. Compared to other models, Eff-CapsNets have lower entropy since, in the original implementation, the kernel size for the PrimaryCaps has the same dimension as the backbone output, namely PrimaryCaps, in this case, has no spatial resolution. Furthermore, we noticed that apart from simple datasets such as MNIST or Fashion-MNIST, the coupling coefficients distributions collapse to similar values, namely the non-iterative self-attention mechanism does not build a proper parse tree.

Robustness to affine transformations. To test the robustness to affine transformations of DR-CapsNets+REM, we used expanded

Dataset	Architecture	Vanilla	Quantized (Q)		REM		
		accuracy	entropy	accuracy	entropy	accuracy	sparsity
Fashion-MNIST	DR-CapsNet	$92.76{\pm}_{0.2}$	$8.64{\pm}_{1.2}$	$92.46{\pm}_{0.2}$	$4.80{\pm}_{1.7}$	$92.62{\pm}_{0.1}$	80.71
	γ-CapsNet	$92.59{\pm}_{1.1}$	$3.98{\pm}_{0.8}$	$92.43{\pm}_{1.1}$	$1.45{\pm}_{0.7}$	$93.01{\pm}_{1.1}$	87.07
	DeepCaps	$92.36{\pm}_{0.2}$	$7.15{\pm}_{1.3}$	$92.33{\pm}_{0.2}$	$6.08{\pm}_{1.3}$	$94.61{\pm}_{0.1}$	83.29
	Eff-CapsNet	$93.31{\pm}_{0.2}$	$3.88{\pm}_{1.1}$	$93.22{\pm}_{0.2}$	$1.10{\pm}_{0.5}$	$92.98{\pm}_{0.4}$	63.29
SVHN	DR-CapsNet	$93.30{\pm}_{0.2}$	$7.13{\pm}_{1.2}$	$92.20{\pm}_{0.2}$	$5.23{\pm}_{0.7}$	$91.71{\pm}_{0.5}$	74.40
	γ-CapsNet	$89.02{\pm}_{0.1}$	$7.15{\pm}_{0.9}$	$87.42{\pm}_{1.2}$	$5.65{\pm}_{1.2}$	$88.36{\pm}_{0.2}$	73.89
	DeepCaps	$93.32{\pm}_{0.3}$	$11.06{\pm}_{0.6}$	$93.20{\pm}_{0.4}$	$3.97{\pm}_{1.5}$	$93.06{\pm}_{0.2}$	80.50
	Eff-CapsNet	$93.64{\pm}_{0.1}$	$0.53{\pm}_{0.6}$	$93.62{\pm}_{0.1}$	$0.24{\pm}_{0.4}$	$93.12{\pm}_{0.1}$	47.80
CIFAR-10	DR-CapsNet	$79.93{\pm}_{0.2}$	$6.26{\pm}_{0.6}$	$78.42{\pm}_{2.1}$	$4.15{\pm}_{0.6}$	$79.25{\pm}_{0.6}$	81.17
	γ-CapsNet	$74.02{\pm}_{0.3}$	$3.67{\pm}_{0.70}$	$73.08{\pm}_{0.48}$	$3.22{\pm}_{0.7}$	$74.89{\pm}_{0.2}$	74.89
	DeepCaps	$90.80{\pm}_{0.1}$	$8.99{\pm}_{0.5}$	$90.47{\pm}_{0.1}$	$7.07{\pm}_{1.1}$	$90.35{\pm}_{0.1}$	46.83
	Eff-CapsNet	$81.53{\pm}_{0.6}$	$0.25{\pm}_{0.3}$	$81.51{\pm}_{0.5}$	$0.005{\pm}_{0.1}$	$81.49{\pm}_{0.5}$	53.79
Dogs vs. Cats	Eff-ConvCapsNet	$97.74{\pm}_{0.3}$	$4.05{\pm}_{0.3}$	$99.2{\pm}_{0.3}$	$2.03{\pm}_{0.7}$	$99.1{\pm}_{0.6}$	71.92
Imagenette	Eff-ConvCapsNet	$83.11{\pm}_{0.1}$	$7.63{\pm}_{0.5}$	$82.98{\pm}_{0.6}$	$5.85{\pm}_{0.3}$	$83.02{\pm}_{0.8}$	63.52

Table 6.5: Accuracy (%), entropy and sparsity on Fashion-MNIST, SVHN, CIFAR-10, Dogs vs. Cats and Imagenette (test set).

MNIST: a dataset composed of padded and translated MNIST, in which each example is an MNIST digit placed randomly on a black background of 40×40 pixels. We used the affNIST dataset as a test set, in which each example is an MNIST digit with a random small affine transformation. We tested an under-trained DR-CapsNet with early stopping which achieved 99.22% accuracy on the expanded MNIST test set as in Sabour et al. [1] and Gu et al. [41]. We also trained these models until convergence. We can see in Table 6.6 that the under-trained network entropies are high. Instead, a well-trained DR-CapsNet+REM can be robust to affine transformations and have low entropy.

Model	expanded MNIST Accuracy (%)	affNIST Accuracy (%)	affNIST Sparsity (%)	affNIST Entropy
DR-CapsNet+Q	99.22	$77.93{\pm}_{0.55}$	0	$8.64{\pm}_{1.15}$
DR-CapsNet+REM	99.22	$81.81{\pm}_{0.81}$	71.26	$8.45{\pm}_{1.10}$
DR-CapsNet+Q	$99.36{\pm}_{0.05}$	$83.14{\pm}_{0.24}$	0	$8.45{\pm}_{0.99}$
DR-CapsNet+REM	$99.48{\pm}_{0.02}$	$85.23{\pm}_{0.11}$	87.32	$5.93{\pm}_{1.39}$

Table 6.6: Results on affNIST test set for under-trained and well-trained models.

Robustness to novel viewpoints. CapsNets are well known for their generalization ability to novel viewpoints [1, 26]. We conducted further experiments on the smallNORB dataset to test the robustness of novel viewpoints of our technique. We employed Eff-CapsNets, as they are the state-of-the-art models on this dataset with few trainable parameters. We used $K = 11$ quantization levels for Eff-CapsNets+Q and

Eff-CapsNets+REM. In Table 6.7, we can see that Eff-CapsNets+REM are indeed robust to novel viewpoints with low entropy. Notice that even if our work does not target state-of-the-art generalization, with REM, we manage to achieve a maximum value of accuracy of 86.99% with only 27k trainable parameters on novel elevations (98.43% for familiar elevations) and 90.37% with only 37k trainable parameters on novel azimuths (97.53% for familiar azimuths).

Model	Familiar		Novel		Sparsity
	Accuracy	Entropy	Accuracy	Entropy	
DR-CapsNet+Q (ϕ)	$90.51 \pm_{0.29}$	$6.25 \pm_{1.15}$	$77.40 \pm_{0.64}$	$5.01 \pm_{1.45}$	0
DR-CapsNet+REM (ϕ)	$90.38 \pm_{0.53}$	$3.35 \pm_{1.18}$	$76.98 \pm_{0.45}$	$2.47 \pm_{0.96}$	50.13
DR-CapsNet+Q (ψ)	$87.44 \pm_{0.51}$	$5.42 \pm_{1.34}$	$72.29 \pm_{0.57}$	$5.02 \pm_{1.07}$	0
DR-CapsNet+REM (ψ)	$87.35 \pm_{0.58}$	$3.38 \pm_{1.35}$	$71.89 \pm_{0.65}$	$2.75 \pm_{1.44}$	58.61
γ-CapsNet+Q (ϕ)	$89.62 \pm_{0.51}$	$1.78 \pm_{0.82}$	$75.54 \pm_{0.52}$	$2.72 \pm_{1.08}$	0
γ-CapsNet+REM (ϕ)	$89.34 \pm_{1.53}$	$1.46 \pm_{0.75}$	$74.40 \pm_{0.62}$	$2.05 \pm_{0.80}$	47.50
γ-CapsNet+Q (ψ)	$85.98 \pm_{0.43}$	$1.87 \pm_{0.58}$	$71.33 \pm_{1.22}$	$2.55 \pm_{0.89}$	0
γ-CapsNet+REM (ψ)	$85.26 \pm_{1.24}$	$1.52 \pm_{0.42}$	$71.12 \pm_{0.025}$	$2.17 \pm_{1.61}$	49.61
DeepCaps+Q (ϕ)	$95.01 \pm_{0.58}$	$7.32 \pm_{1.18}$	$83.18 \pm_{1.61}$	$7.28 \pm_{1.66}$	0
DeepCaps+REM (ϕ)	$94.62 \pm_{0.52}$	$6.75 \pm_{1.41}$	$82.49 \pm_{1.53}$	$6.12 \pm_{1.91}$	34.45
DeepCaps+Q (ψ)	$90.16 \pm_{0.27}$	$6.53 \pm_{1.77}$	$79.36 \pm_{0.76}$	$5.14 \pm_{1.45}$	0
DeepCaps+REM (ψ)	$90.13 \pm_{0.13}$	$5.55 \pm_{1.49}$	$78.66 \pm_{1.21}$	$3.92 \pm_{1.36}$	36.06
Eff-CapsNet+Q (ϕ)	$97.81 \pm_{0.45}$	$1.10 \pm_{0.42}$	$85.03 \pm_{1.01}$	$1.33 \pm_{0.53}$	0
Eff-CapsNet+REM (ϕ)	$98.08 \pm_{0.25}$	$0.26 \pm_{0.14}$	$86.50 \pm_{0.31}$	$0.25 \pm_{0.11}$	75.09
Eff-CapsNet+Q (ψ)	$97.07 \pm_{0.52}$	$1.69 \pm_{0.28}$	$87.95 \pm_{1.67}$	$1.71 \pm_{0.34}$	0
Eff-CapsNet+REM (ψ)	$97.31 \pm_{0.31}$	$0.38 \pm_{0.15}$	$88.48 \pm_{1.38}$	$0.33 \pm_{0.12}$	68.85

Table 6.7: Accuracy (%) and entropy values on the smallNORB test set on familiar and novel viewpoints (elevations ϕ and azimuths ψ) seen and unseen during training respectively.

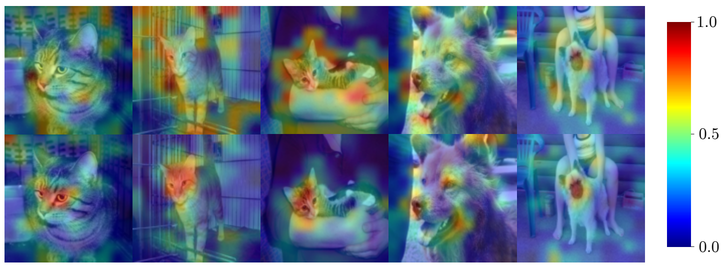
We also show in Table 6.8 the performances of these networks without quantization. All the models are trained with our implementations when the source code is unavailable.

Model	Familiar	Novel	Sparsity
DR-CapsNet (ϕ)	90.62$\pm_{0.21}$	77.51$\pm_{0.43}$	0
DR-CapsNet (ϕ)	90.51$\pm_{0.45}$	77.03$\pm_{0.38}$	50.13
DR-CapsNet (ψ)	87.90$\pm_{0.50}$	72.37$\pm_{0.43}$	0
DR-CapsNet (ψ)	86.81$\pm_{0.61}$	71.99$\pm_{0.66}$	58.61
γ-CapsNet (ϕ)	90.15$\pm_{0.39}$	75.89$\pm_{0.53}$	0
γ-CapsNet (ϕ)	89.92$\pm_{0.91}$	74.96$\pm_{0.71}$	47.50
γ-CapsNet (ψ)	86.11$\pm_{0.63}$	72.55$\pm_{0.83}$	0
γ-CapsNet (ψ)	85.35$\pm_{0.61}$	71.35$\pm_{1.31}$	49.61
DeepCaps (ϕ)	95.32$\pm_{0.48}$	83.13$\pm_{0.91}$	0
DeepCaps (ϕ)	94.48$\pm_{0.31}$	82.42$\pm_{1.53}$	34.45
DeepCaps (ψ)	91.11$\pm_{0.27}$	79.53$\pm_{0.78}$	0
DeepCaps (ψ)	90.15$\pm_{0.91}$	78.83$\pm_{1.12}$	36.06
Eff-CapsNet (ϕ)	97.83$\pm_{0.41}$	85.04$\pm_{1.06}$	0
Eff-CapsNet (ϕ)	98.10$\pm_{0.33}$	86.50$\pm_{0.35}$	75.09
Eff-CapsNet (ψ)	97.07$\pm_{0.52}$	87.98$\pm_{1.64}$	0
Eff-CapsNet (ψ)	97.29$\pm_{0.34}$	88.54$\pm_{1.35}$	68.85

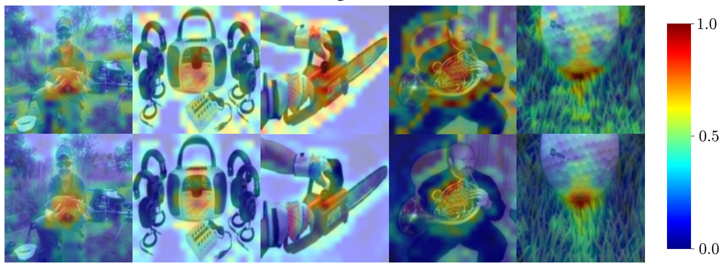
Table 6.8: Accuracy (%) on the smallNORB test set on familiar and novel viewpoints (elevations ϕ and azimuths ψ) for DR-CapsNets, γ-CapsNets, DeepCaps and Eff-CapsNets without quantization.

Improved visualizations with REM. Figures 6.10a and 6.10b show the saliency maps overlayed on Dogs vs. Cats and Imagenette images. We chose these datasets for the saliency maps visualization method since the images come with higher resolution and more variation than the other datasets. When REM is applied (second rows), we can see that the network predominantly concentrates on the object of interest, dismissing extraneous background noise or irrelevant objects. This characteristic of REM leads to the extraction of more concise part-whole hierarchies. As illustrated in Figure 6.10a, only the discriminative features specific to a cat or a dog are prominently highlighted in red. In contrast, features corresponding to people or cages play a diminished role in the classification process. Similarly, in Figure 6.10b, the model focuses mostly on the tench, the cassette player, the chain saw, the French horn, and the golf ball, disregarding trees, headphones, hands, the man, and grass.

In Figures 6.11 and 6.12 we show the saliency maps for smallNORB when tested on novel azimuths and elevations, respectively. We used the method described in Appendix 6.2.2. We show one object instance for each category. We can see from these visualizations on smallNORB that the differences between saliency maps of REM and without REM are less prominent. This is because smallNORB is less complex than



(a) Dogs vs. Cats.



(b) Imagenette.

Figure 6.10: Saliency maps for Eff-ConvCapsNets without and with REM, first and second rows, respectively.

Fashion-MNIST or CIFAR-10. However, we can still notice that some of the saliency maps extracted with REM are more prominent and consistent between each category.

Figure 6.13 shows the parse trees extracted from a fully-connected DR-CapsNet for some CIFAR-10 images of the test set. We can see that fewer capsules are part of the parse tree extracted with REM: the network relies on fewer object parts to detect the objects.

6.4 SUMMARY

This paper introduced REM, a technique that drives the model parameters distribution toward low entropy configurations. With REM, capsule networks do not need to model irrelevant objects or noise in the images to achieve high generalization. Therefore, even a reasonable sized capsule network can now learn more discriminative features. A novel visualization method for capsule networks is employed to show which features play a major role in the classification process. Extensive experiments on different datasets and architectures confirm the effectiveness of the proposed approach. Recent works show that capsule networks do not scale [66] and suffer vanishing activations when adding many capsule layers [73], leading to poor performance. Therefore, carving out an ideal parse tree from complex images with perfectly interpretable capsules is not possible with capsule networks, yet. However, REM opens research pathways toward the distillation of parse trees and model interpretability, including the design of a pruning technique specifically designed for capsules.

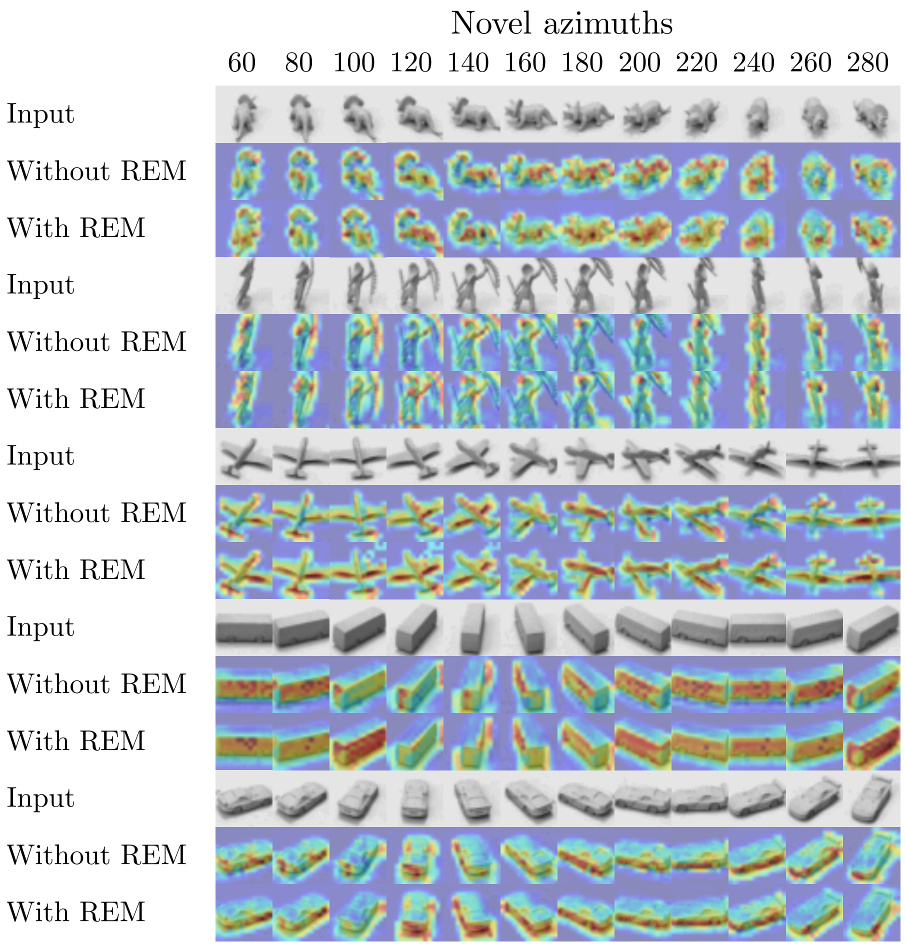


Figure 6.11: Saliency maps for smallNORB for DR-CapsNet+Q and DR-CapsNet+REM (novel azimuths).

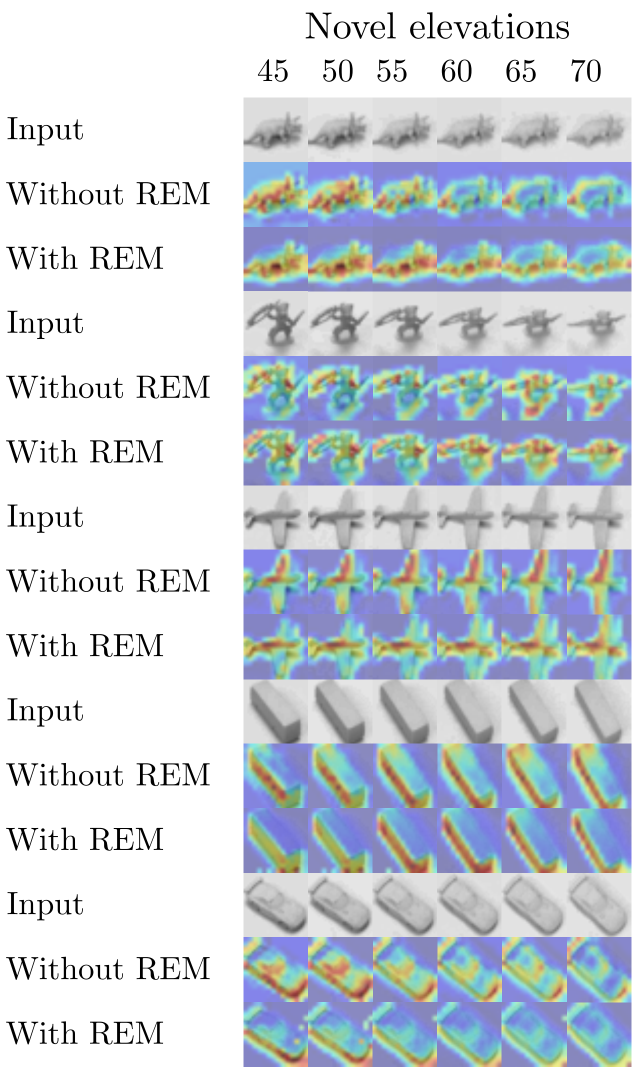


Figure 6.12: Saliency maps for smallNORB for DR-CapsNet+Q and DR-CapsNet+REM (novel elevations).

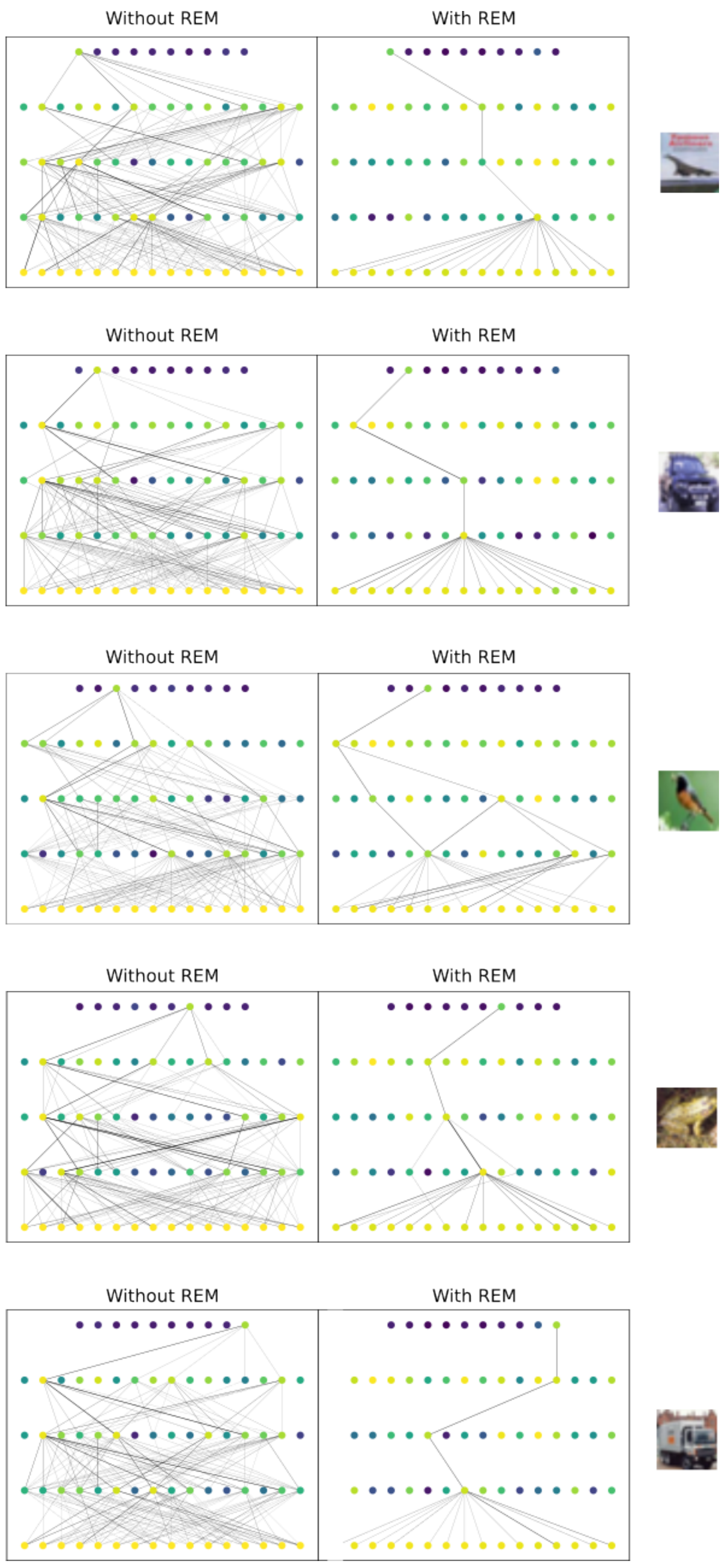


Figure 6.13: Parse trees for some CIFAR-10 test images for multilayered DR-CapsNet and DR-CapsNet+REM with backtracking.

Part III

CAPSULE NETWORKS APPLICATIONS

In the previous parts, we presented the fundamentals of capsule networks along with their extensions, primarily focusing on image classification. However, it's important to note that the utility of capsule networks extends well beyond image classification. Indeed, their versatile architectures can be adapted to tackle various tasks, such as image similarity matching and segmentation. This part delves into the applications of capsule networks beyond the realm of image recognition. Specifically, we will delve into the following intriguing applications of capsule networks:

1. **Localization of unmanned aerial vehicles without GPS reliance**: By harnessing satellite and drone images, capsule networks can play a pivotal role in extracting feature vectors that facilitate the verification of whether a pair of satellite and drone images share the same geographical region. This innovation offers a novel unmanned aerial vehicle localization approach that doesn't rely on traditional GPS systems.

2. **Encoding rotations of synthetic datasets with quaternions**: Many applications such as robot grasping need to know the exact position and orientation of an object. However, real-world datasets do not always include these informations. We generate Blender images of various meshes displayed from different angles, labeled with object classes and rotations represented using quaternions, which offer advantages over rotation matrices. We explore the idea of using capsule networks to encode quaternions, building on previous work that combined quaternions and capsules to capture object poses effectively. The ultimate goal is to leverage synthetic datasets annotated with quaternion labels to generalize to real-world datasets.

3. **Lung nodules segmentation**: Capitalizing on the inherent strengths of capsule architectures, it becomes feasible to expand them into both encoder and decoder components. This extension proves invaluable in the task of lung nodule segmentation. The architecture can classify each pixel within an image, differentiating between regions belonging to a nodule and those not. This advancement holds substantial promise for enhancing medical imaging diagnostics and analysis.

Incorporating capsule networks into these diverse applications underscores their adaptability and potential to revolutionize various fields beyond the confines of image classification.

7 LSVL: LARGE-SCALE SEASON-INVARIANT VISUAL LOCALIZATION FOR UAVS

The research findings and insights presented in this chapter are the results of a visiting period at Aalto University (Helsinki, Finland), supervised by Prof. Ville Kyrki. The collaboration led to the following publication: Kinnari, J. "LSVL: Large-scale season-invariant visual localization for UAVs." In: *Robotics and Autonomous Systems* (2023), p. 104497. ISSN: 0921-8890.

7.1 INTRODUCTION

The ability of a Unmanned Aerial Vehicle (UAV) to robustly estimate its position is one of the basic requirements of autonomous flight. In missions in which the UAV needs to collaborate with other agents operating in the same environment, information of position in a shared, global frame of reference is needed.

There are several possible ways to estimate position. A localization solution can rely on infrastructure built for this purpose. In outdoor UAV operations, by far the most common is to rely on Global Navigation Satellite Systems (GNSS). In ideal conditions, GNSS receivers provide measurements of position. However, GNSS, as well as other radio beacon systems, are susceptible to spoofing and jamming attacks [75, 76], which may be used for denying operation of UAVs in an area.

A sensor system commonly carried by UAVs is the combination of a camera and an Inertial Measurement Unit IMU. Several works have focused on using this sensor combination for resolving the position of the UAV by tracking the difference to a known starting position by methods of VIO [77]. As VIO implementations integrate noisy signals, they suffer from drift over long flights [78]. Simultaneous Localization and Mapping (SLAM) methods [79] can help in partially compensating for this drift by detecting loop closures, but only if the mission contains reentry to a previously visited area. In addition, visual-inertial odometry VIO and SLAM approaches are not robust to random failures in position tracking; temporary failures may lead to loss of position information in the global frame with only a small chance of recovery.

If a map is available, matching camera images acquired by a UAV to the map allows compensating for drift induced by odometry methods and provides an estimate of position with respect to the map. This approach is called visual localization [80, 81]. Using a georeferenced map allows, in principle, finding the position of a UAV with respect to the map, even in the case of no prior information on position at start of mission or after random failures in positioning.

There are, however, multiple challenges in visual localization. Appearance difference between the UAV image and the map may be significant due to changes in season — an image acquired by a UAV in winter looks very different from a satellite image acquired in summer. Moreover, the environment in which the UAV is operating may be naturally ambiguous, *e.g.*, when flying over vast areas of forests. In order for the localization solution to support recovery from random localization failures, the size of the map must correspond to the operating area of the UAV. This means that the localization solution must be computationally efficient enough to run on an onboard computer even on large maps, and it must tolerate the natural ambiguity.

In this work we propose LSVL, which addresses all of the above mentioned challenges of robust visual localization at large scale. The main contributions of this work are:

1. We propose an approach for localizing a UAV which is able to handle natural ambiguities and determine the location of the UAV even when no prior information is available about its starting pose over very large areas. Our extensive testing demonstrates that, starting from an uncertainty corresponding to 100 km^2 area, our approach can converge to an average translation error of 12.6–18.7 m after no more than 23.2–44.4 updates using UAV camera observations, under significant seasonal appearance difference between UAV observation and map, when flying over terrains containing natural ambiguity. Importantly, our approach does not rely on a digital elevation model and can operate effectively using only 2D maps.

2. We propose a UAV image to map matching solution based on compact descriptors learned in a manner that provides invariance to seasonal appearance difference. We train the descriptor networks using only satellite images, without requiring any labeled data such as semantic classification of the terrain. The use of compact descriptors enables fast computation of a large number of hypotheses.

3. We present the problem of UAV visual localization as state estimation utilizing a point mass filter, and integrate it to our map matching approach. This combination allows running our algorithm in real time, with constant time and memory consumption onboard a UAV at sufficiently large scale for UAV missions.

4. We explore different architecture choices for image description, vector dimensionality, and likelihood computation, and evaluate their impact on probability to convergence, time to convergence and positioning error after convergence. We compare our solution to two state-of-the-art approaches for UAV localization, and

demonstrate the operation of our localization solution in real time onboard a commercial UAV.

5. We present a simple method allowing online assessment of integrity of pose estimate, enabling self-diagnostics of the localization solution, which is a key component in the robust global localization problem.

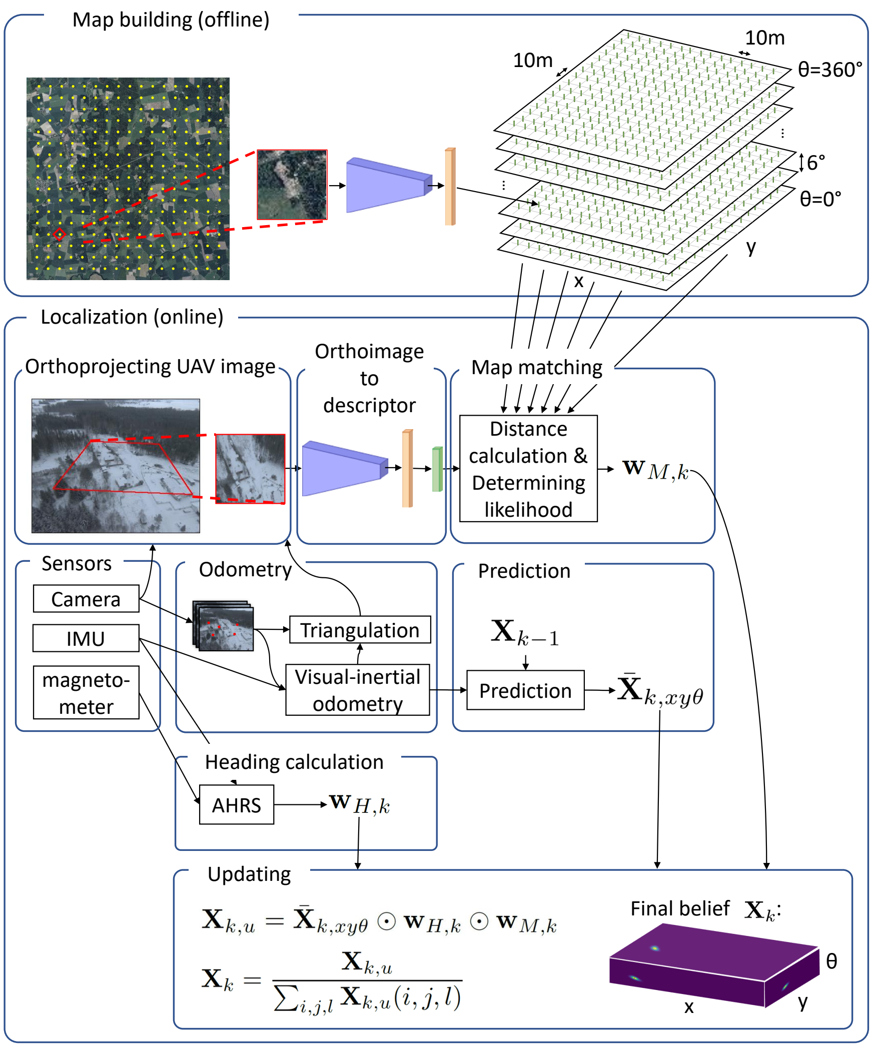
We believe this is the first work in the visual localization area that is able to find true position of the UAV, starting from a scale of uncertainty of 100 km^2, over ambiguous terrains, under significant seasonal appearance change. A block diagram of our algorithm is shown in Figure 7.1.

The paper is structured as follows. Section 7.2 goes over related work. Section 7.3 defines preliminaries of the localization problem. We introduce our method in Section 7.4 and detail a vital part of the solution, map matching, in Section 7.5. Section 7.6 describes our localization experiments and we conclude the paper with discussion and conclusions in Section 7.7 and Section 7.8, respectively.

7.2 RELATED WORK

Visual localization of UAVs is a topic which has attracted interest especially over past few years. The state-of-the-art is covered by relatively recent surveys [80, 81]. Within the area of visual localization, most works are limited by one or more of the following assumptions: accurate initialization is required [82–85], the size of the operating area is limited [86, 87], the movements of the UAV are constrained to specific paths [84], the operation takes place in conditions that are very close to map in terms of appearance [82, 86], the map resolution and detail requirement is significant, such as requiring a topography map [87], or a very high flying altitude is required for successful georeferencing [88]. The following overview of related work focuses on the works that—similarly to LSVL—do not require of knowledge of initial pose and that use an easily obtainable planar 2D map for localization.

A common choice in UAV localization is to detect semantic features such as roads and intersections [88–91] or buildings [92]. Choi *et al.*[92] proposed a method where the UAV image is semantically segmented to find buildings in the camera view. Based on detected buildings, a rotation invariant descriptor, building ratio map (BRM), is computed from the proportion of building pixels visible in camera view, and a precomputed map with similar descriptors is used for localization. The authors demonstrated convergence of position estimate on a 6.17 km^2 map after 27 updates with 25 meters of translation between each update, with 12.01 m root-mean-square (RMS) error after convergence. The demonstration flight takes place over a residential area. The main drawback of this approach is the requirement that features of a specific



Map building (offline)
10m
10m
θ=360°
6°
θ=0°
x
y
Localization (online)
Orthoprojecting UAV image
Orthoimage to descriptor
Map matching
Distance calculation & Determining likelihood
$\mathbf{w}_{M,k}$
Sensors
Camera
IMU
magneto-meter
Odometry
Triangulation
Visual-inertial odometry
Prediction
$\mathbf{X}_{k-1}$
Prediction
$\bar{\mathbf{X}}_{k,xy\theta}$
Heading calculation
AHRS
$\mathbf{w}_{H,k}$
Updating
$\mathbf{X}_{k,u} = \bar{\mathbf{X}}_{k,xy\theta} \odot \mathbf{w}_{H,k} \odot \mathbf{w}_{M,k}$
$\mathbf{X}_k = \frac{\mathbf{X}_{k,u}}{\sum_{i,j,l} \mathbf{X}_{k,u}(i,j,l)}$
Final belief $\mathbf{X}_k$:
θ
x
y

Figure 7.1: Block diagram of proposed localization solution. We train a neural network that computes embedding vectors. We precompute a map of embeddings offline for a grid of position and orientation hypotheses that cover the full considered state space. During flight, we run a localization algorithm that tracks UAV egomotion by VIO and computes an embedding of an orthoprojected section of the UAV image. We predict belief over time steps in accordance with VIO, and update belief based on likelihood computed from distance in embedding space between image observation and map hypotheses. We optionally include heading measurements.

semantic class (*i.e.*, buildings) are required for successful localization, which may be unavailable when flying over natural environments.

Mantelli et al. [93] demonstrate localizing a UAV on a map of size 1.34 km^2 using a particle filter, where particle likelihood is determined based on a handcrafted descriptor called abBRIEF. The authors initialize the particle filter with 50 000 particles and show robustness of their localization solution in comparison to BRIEF [94] descriptors on trajectory lengths up to 2.4 km, showing convergence in less than 50 m to an average translation error of 17.78 m. The descriptor is developed such that it is tolerant to illumination changes and allows fast computation of particle likelihood over a large number of pose hypotheses.

We evaluate against BRM and abBRIEF and show superior performance.

We build on the idea of compressing UAV camera observation into a compact embedding space, to allow fast testing of pose hypotheses on a large map. This has been proposed recently by *e.g.*, Bianchi *et al.* [84] who use a bottleneck autoencoder approach to compress visual observations to descriptor vectors of dimension 1000. Also Samano *et al.* [95] and Couturier *et al.* [96] train Resnet models [97] to project map tiles and UAV images to a low-dimensional (16D) embedding space. Only Samano *et al.*[95] allow movements of the UAV outside precomputed paths by precomputing a grid of embedding vectors, on which hypothesis testing is performed by interpolating a vector from this grid by each pose hypothesis and measuring distance of observed image descriptor to interpolated descriptor vector in embedding space. Authors of [95] demonstrate with simulated flight experiment the convergence of pose estimate to less than 95 m translation error in 78.2% of simulated flights by 200 updates. We take a similar approach but forgo interpolation in embedding space, to avoid unjustified implicit assumption of smoothness of embedding space, and explore other learned descriptor architectures. Compared to [95], we show significantly faster convergence, tolerance to seasonal variation and we demonstrate performance with real experiments.

The topic of localization over large areas under visual appearance change occurs also under the topics of visual place recognition (VPR) [98] and SLAM (*e.g.*, Naseer et al. [99]). In VPR, the problem of localization is commonly solved as an image retrieval task (*e.g.*, Hausler et al. [100]) or as refinement with respect to a preconstructed Structure-from-Motion (SfM) map of an area [101]; in SLAM, the sensor system of the robot is used for building the map. In contrast, we propose a localization solution that does not require having previously visited or acquired imagery in the area of operation. Instead, we focus on the problem of localization with respect to an orthophoto map. In this way, using only low-cost orthoimage maps, the size of the region of operations is not constrained by cost of data acquisition for map building.

Some variations due to weather can be removed by denoising [102, 103]. Our work does not target removal of effects of adverse weather on the imaging process, but methods of denoising could be combined as a preprocessing step to our work.

7.3 preliminaries

We want to resolve the pose of an UAV using measurements available at the UAV during flight, with no dependency on localization or communication infrastructures. Formally, we define the pose of the UAV pose in a common, global reference frame as the state

$$\mathcal{X}_k = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^T \tag{7.1}$$

where x, y, and θ represent longitude, latitude, and heading, respectively, in a Cartesian coordinate system and k is index for time. We consider localization of the UAV in a limited region in longitude and latitude: we assume $x_k \in [x_{min}, x_{max}]$, $y_k \in [y_{min}, y_{max}]$, $\theta_k \in [0, 2\pi)$. The localization problem is to compute the marginal posterior distribution $p(\mathcal{X}_k|\mathbf{Y}_{1:k}, \mathcal{M})$ of state $\mathcal{X}_k$, given history of measurements $\mathbf{Y}_{1:k}$ and map $\mathcal{M}$.

In this work, we concentrate on the *wake-up robot problem*, *i.e.*, at the start of a flight, we assume an uniform prior distribution $p(\mathcal{X}_0)$ over all values of x_k and y_k, across all values of heading. This represents a situation where the only initial information about the UAV position is that it is located within the area of a map defined over a rectangular area in latitude and longitude. Our formulation allows inclusion of more informed initialization.

We take the typical approach of considering localization on preacquired map as a Bayesian filtering problem (see *e.g.*, [104]). This amounts to maintaining a representation of belief of current state and updating that representation when new measurements are available. At each time step k, we obtain three types of measurements from the onboard sensors of the UAV: an odometry measurement $\mathbf{u}_k$, an heading measurement v_k, and an image from the UAV camera $\mathcal{I}_k$ to be used for map matching. Given the measurment $\mathbf{Y}_k = \left\{\mathbf{u}_k, v_k, \mathcal{I}_k\right\}$, at each sampling time k, we first perform a prediction step based on the odometry measurement:

$$p(\mathcal{X}_k|\mathbf{u}_k, \mathbf{Y}_{1:k-1}) = \int p(\mathcal{X}_k|\mathbf{u}_k, \mathcal{X}_{k-1}) p(\mathcal{X}_{k-1}|\mathbf{Y}_{1:k-1}) d\mathcal{X}_{k-1} \tag{7.2}$$

We then use the heading and map matching measurements as our observation model:

$$p(\mathcal{X}_k|\mathbf{Y}_{1:k}, \mathcal{M}) = \frac{1}{\eta_k} p(\mathcal{X}_k|\mathbf{u}_k, \mathbf{Y}_{1:k-1}) p(v_k, \mathcal{I}_k|\mathcal{X}_k, \mathcal{M}) \tag{7.3}$$

where η_k is a normalizing constant. The heading and map matching measurements are considered independent, and likelihood of heading measurement is not conditional on map:

$$p(v_k, \mathcal{I}_k | \mathcal{X}_k, \mathcal{M}) = p(v_k|\mathcal{X}_k) p(\mathcal{I}_k|\mathcal{X}_k, \mathcal{M}) \tag{7.4}$$

7.4 METHOD

In order to find the pose on a large map in presence of matching ambiguities presented in the introduction, a method is needed for utilizing a sequence of as many UAV images as needed in order to converge to a single, correct pose estimate. We propose a recursive *localization method* consisting of the following components, illustrated in Figure 7.1. We use an *odometry measurement* for predicting belief of state $\bar{\mathbf{X}}_{k,xy\theta}$ at time k based on state $\mathbf{X}_{k-1}$ at previous time instant $k-1$. We use a *map matching measurement* for computing likelihood of pose hypotheses $\mathbf{W}_{M,k}$ based on a single UAV image and, optionally, a *heading measurement* for computing the likelihood of pose hypotheses $\mathbf{W}_{H,k}$ based on a compass heading measurement. The localization method updates belief of state using all the measurements, providing $\mathbf{X}_k$, belief of state at time k.

In this section we describe each of these components in detail, putting particular attention in describing how the proposed solutions target the complexity that arises from both dealing with a large map and with considerable appearance change, central limitations of current methods that we address in this work. Section 7.4.1 lists all measurements we use, and Section 7.4.2 specifies our localization method.

7.4.1 *Measurements*

7.4.1.1 *Odometry measurement*

We assume that the UAV is running a VIO algorithm which provides a measurement of incremental motion in six degrees of freedom. We marginalize over roll and pitch, which are directly observable in the global frame [105], and altitude, which we abstract with the map matching approach presented in Section 7.5. The odometry algorithm provides, at time k, the measurement $\mathbf{u}_k = \begin{bmatrix} u_{k,x} & u_{k,y} & u_{k,\theta} & u_{k,o} \end{bmatrix}^T$, a measurement of translation, rotation and distance traveled since time $k-1$. $u_{k,x}$ and $u_{k,y}$ are translation since time instant $k-1$ in x and y coordinates, respectively, with respect to pose at time $k-1$. Similarly, $u_{k,\theta}$ is rotation around vertical axis. $u_{k,o}$ is the integral of distance traveled since instant $k-1$ according to odometry. The measurements are visualized in Figure 7.2a.

We approximate the odometry pose uncertainty with a multivariate normal distribution. We further assume the covariance of pose is isotropic in x, y directions and that rotation noise is independent from translation noise. More formally, we approximate the posterior density with a multivariate normal distribution with covariance $\mathbf{\Sigma}_u$. This gives the motion model

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} cos\theta_k & -sin\theta_k & 0 \\ sin\theta_k & cos\theta_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{u}}_k + \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_u(u_{k,o})) \tag{7.5}$$

where the mean of the measurement $\bar{\mathbf{u}}_k = \begin{bmatrix} \bar{u}_x & \bar{u}_y & \bar{u}_\theta \end{bmatrix}^T$ is specified in the odometry frame at time k and $\mathbf{\Sigma}_u(u_{k,o})$ specifies the covariance of the measurement, and $\mathbf{\Sigma}_u(t)$ is a diagonal matrix:

$$\mathbf{\Sigma}_u(t) = diag(\sigma_{u,xy}(t)^2, \sigma_{u,xy}(t)^2, \sigma_{u,\theta}(t)^2). \tag{7.6}$$

While the isotropic noise for translation and independent noise in heading are simplified approximations of the true distribution [106], we consider this a sufficient upper bound approximation to the odometry noise. This approximation enables a computationally fast method of prediction as elaborated in Section 7.4.2.3.

In addition, we assume the movements of the UAV produce sufficient IMU excitation to make scale observable or that the UAV is equipped with additional sensors with which scale can be resolved.

7.4.1.2 *Map matching measurement*

The purpose of map matching is to provide a means for verifying or disputing pose hypotheses, given camera image $\mathcal{I}_k$ and map $\mathcal{M}$. A method for computing the likelihood $p(\mathcal{I}_k|\mathcal{X}_k, \mathcal{M})$ is thus needed. Our work focuses especially on finding means for computing the likelihood that work well over ambiguous terrains under significant seasonal appearance difference, in a computationally efficient way. Since the method used for map matching and likelihood computation based on a UAV image is a considerable part of our contribution, a separate section (Section 7.5) has been given to the detailed description of this component.

7.4.1.3 *Heading measurement*

We assume the UAV is equipped with an heading ref- erence system (AHRS), relying on fusion of a compass and an IMU, which provides a measurement of heading, θ, with respect to map East, corrupted by Gaussian noise $n_\theta \sim \mathcal{N}(0, \sigma_v^2)$:

$$v_k = v(t_k) = \theta(t_k) + n_\theta \tag{7.7}$$

7.4.2 *Localization method*

We have to consider how to formulate the pose estimation problem in a computationally feasible way. In choosing the estimation approach, we need to consider characteristics of the problem. Specifically, as we start from a very uninformed state and expect natural ambiguities in the environment we are operating in, it is expected that we need to be able to track a large number of multiple hypotheses before converging to the correct pose, and before convergence, the state is expected to not conform to a parametric probability distribution model. This leads us to consider nonparametric Bayesian filtering approaches, and in the localization context, two commonly used solutions are particle filtering and point mass filtering [104].

7.4.2.1 *Choosing representation for state*

The use of a particle filter in UAV localization is a common choice [86, 93, 95]. However, in cases of very uninformed initialization, a risk exists that a particle is initially not placed in vicinity of the true pose, leaving the probability of converging on the correct pose over time to chance. Furthermore, during a flight, it is possible that the close proximity of true state is left without sufficient particle density in cases such as when flying for long periods of time over ambiguous areas, or in case of large but local map inconsistencies (e.g. forest clearcutting having taken place between map acquisition and flight), leading to divergence in a poorly predictable way. Since the particle filter is stochastic, different instantiations of the filter using the same data may provide different results and, depending on selected resampling scheme, computation time may vary.

Instead of representing belief through the use of particles, we choose to use a point mass filter and compute the belief on a discrete grid. This selection ensures coverage of full state space throughout the mission, at the resolution specified by our choice of grid, and offers deterministic performance and constant computational time and memory requirement. Early adaptations of point mass filtering-based approaches on 2D robot localization include the work by Burgard et al. [107] and in terrain navigation of flying platforms by Bergman [108].

7.4.2.2 *Definition of state using point mass filter*

We approximate belief $p(\mathcal{X})$ on the continuous state space of $\mathcal{X}$ by decomposing it into a grid of regions of equal size with resolution

$r_x = r_y = r_{xy}$ in translation and r_θ in heading. Each region is a voxel $s(i,j,l) = \left[s_x(i), s_y(j), s_\theta(l)\right]$ in the state space with bounds

$$\begin{aligned} s_{xv}(i) &\leq s_x(i) < s_{xu}(i) \\ s_{yv}(j) &\leq s_y(j) < s_{yu}(j) \\ s_{\theta v}(l) &\leq s_\theta(l) < s_{\theta u}(l) \end{aligned} \tag{7.8}$$

with lower bound $s_{bv}(i) = b_{min} + i \times r_b$ and upper bound $s_{bu}(i) = s_{bv}(i) + r_b$ for each axis $b \in \{x, y, \theta\}$ and $i, j, l \in \mathbb{N}$ such that the whole state space is covered.

We approximate belief over state at time instant k as a piecewise constant probability matrix $\mathbf{X}_k \approx p(\mathcal{X}_k)$ where each element of matrix $\mathbf{X}_k[i,j,l]$ assigns a probability to each voxel $s(i,j,l)$ in state space.

7.4.2.3 *Using odometry measurement for prediction*

To approximate the prediction step equation 7.2 with our state representation, using the VIO measurement model presented in Section 7.4.1.1, we formulate a method in which the probability mass contained in a voxel in $\mathbf{X}_{k-1}$ is projected to other voxels in the belief grid at time k as dictated by the odometry measurement. We repeat this operation for each voxel in the belief grid. An example visualization is shown in Figure 7.2. Isotropic (x, y) odometry noise and independent θ odometry noise enable us to run the prediction computationally efficiently as three consecutive 1D convolutions.

We compute offsets $o_x(\alpha)$, $o_y(\alpha)$, o_θ and kernel vectors $\check{}_\theta$, $\check{}_x(\alpha)$ and $\check{}_y(\alpha)$ such that for a given initial heading α in the global frame, the kernels span a region of at least four standard deviations around mean. We then perform prediction by running 1D convolutions in sequence:

$$\bar{\mathbf{X}}_{k,x}[i,j,l] = \sum_{h=1}^{q_x} \mathbf{X}_{k-1}[i - o_x(\bar{s}_\theta(l)) - h, j, l]\check{}_x(\bar{s}_\theta(l))[h] \tag{7.9a}$$

$$\bar{\mathbf{X}}_{k,xy}[i,j,l] = \sum_{h=1}^{q_y} \bar{\mathbf{X}}_{k,x}[i, j - o_y(\bar{s}_\theta(l)) - h, l]\check{}_y(\bar{s}_\theta(l))[h] \tag{7.9b}$$

$$\bar{\mathbf{X}}_{k,xy\theta}[i,j,l] = \sum_{h=1}^{q_\theta} \bar{\mathbf{X}}_{k,xy}[i, j, l - o_\theta - h]\check{}_\theta[h] \tag{7.9c}$$

Here, $\bar{s}_\theta(l)$ returns the centerpoint of the angle corresponding with index l in $\mathbf{X}$. Note that the offsets and convolution kernel vectors in x and y direction depend on the value of α, *i.e.*, probability mass is shifted in the direction defined by the heading value, using the nominal value of that cell. At the edges of state space, equation 7.9a and equation 7.9b are filled with zero for values outside state space and equation 7.9c is wrapped around to the opposite edge.

The end result of this step is $\bar{\mathbf{X}}_{k,xy\theta}$, which states how $\mathbf{X}_{k-1}$ shifted and spread according to odometry measurement and odometry noise from time $k-1$ to k. A clarifying visualization can be found in Figure 7.2.

7.4.2.4 *Weighing belief with heading measurement*

We approximate the circular Gaussian presented in Section 7.4.1.3 by von Mises distribution and we compute a weight matrix $\mathbf{W}_{H,k}$ for all grid indices:

$$\mathbf{W}_{H,k}(i,j,l) = \Phi_{\mathcal{V}}(s_{\theta u}(l), v_k, 1/\sigma_v^2) - \Phi_{\mathcal{V}}(s_{\theta v}(l), v_k, 1/\sigma_v^2) \tag{7.10}$$

where $\Phi_{\mathcal{V}}(t,\theta,1/\sigma^2)$ is the cumulative density function of von Mises distribution with parameters θ, $1/\sigma^2$ evaluated at t.

7.4.2.5 *Weighing belief with map matching measurement*

For all grid indices, we compute a weight matrix $\mathbf{W}_{M,k}$, from likelihood of the observation $\mathcal{I}_k$ representing the voxel $s_x(i), s_y(j), s_\theta(l)$, given map $\mathcal{M}$:

$$\mathbf{W}_{M,k}(i,j,l) = p(\mathcal{I}_k | s_x(i), s_y(j), s_\theta(l), \mathcal{M}) \tag{7.11}$$

The likelihood computation methods are described in detail in Section 7.5.4.

7.4.2.6 *Updating with all measurements*

Our updated state estimate is computed as

$$\mathbf{X}_{k,u} = \bar{\mathbf{X}}_{k,xy\theta} \odot \mathbf{W}_{H,k} \odot \mathbf{W}_{M,k}, \tag{7.12}$$

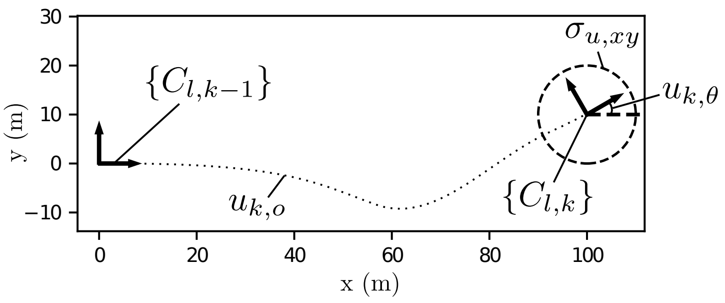
where $\odot$ is elementwise multiplication, and finally normalized:

$$\mathbf{X}_k = \frac{\mathbf{X}_{k,u}}{\sum_{i,j,l} \mathbf{X}_{k,u}(i,j,l)} \tag{7.13}$$

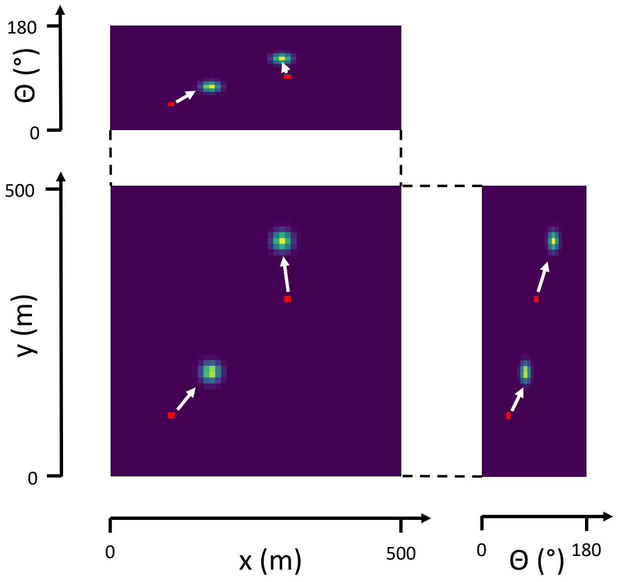
The end result is a recursive discrete approximation of equation equation 7.3.

7.4.2.7 *Interval for running algorithm*

Our localization algorithm is run at fixed intervals of travel, when the UAV has traveled more than a specified distance u_l since the latest update. The amount of travel since latest update is approximated by odometry. The value for u_l can be chosen to balance computational load, ensure enough movement with respect to to selected grid size, and have independence between UAV images used in map matching. In general, smaller u_l is preferred. The lower limit for u_l comes from the requirement of independence of consecutive observations in the Bayesian filter and thus u_l should be approximately equal to the dimension of the image used in map matching, which in our experiments is 100 m and 40 m for convergence experiments and real-time



(a) Odometry measurements are stated with respect to frame at previous update, $\{C_{l,k-1}\}$. $u_{k,o}$ states distance traveled since previous update, according to odometry.



(b) Part of $\mathbf{X}$ after marginalizing different axes individually. Two voxels at $(100, 100, 45°)$ and $(300, 300, 90°)$ representing belief before prediction are shown in red. Voxels highlighted in green and yellow represent belief after prediction. White arrows show how prediction using these odometry measurements with the assumed noise shifts and smooths $\mathbf{X}$ according to the measurement, and that shifts are performed in direction determined by odometry measurement and voxel's θ value.

Figure 7.2: Example of prediction based on odometry. In this example, $u_{k,x} = 100$ m, $u_{k,y} = 10$ m, $u_{k,\theta} = 30°$, $\sigma_{u,xy} = 10$ m, $\sigma_{u,\theta} = 5°$. Figure 7.2a visualizes odometry measurements between times $k-1$ and k. Figure 7.2b visualizes how odometry measurements are used in prediction.

experiments, respectively. Increasing u_l leads to less updates per distance traveled and therefore slower convergence, but allows allocating more time for running the localization algorithm. Between updates, a location estimate in navigation frame is tracked using odometry.

7.5 MAP MATCHING

Our formulation this far has considered how to fuse odometry, heading and map matching measurements using a point mass filter. We have still to define how to compute the likelihood $p(\mathcal{I}_k|\mathcal{X}_k, \mathcal{M})$, *i.e.*, assess how well the observed UAV image $\mathcal{I}_k$ supports each possible value of state $\mathcal{X}_k$, when using map $\mathcal{M}$.

Instead of detecting image features and using them as landmarks as done in *e.g.*, [109], we prefer an area-based approach, *i.e.*, using a large part of the observed image for discriminating between plausible and incorrect poses. The main motivation for this choice is that feature-based approaches require detection of spatially local features, which may be very sparsely detectable when flying over ambiguous terrains, especially across significant seasonal change (*e.g.*, after accumulation of snowfall) and when image footprint on ground is small. Using a large section of UAV image allows us to assess plausibility of matching with respect to a map even if locally distinct features cannot be detected.

Besides the targeted robustness when flying in ambiguous terrains, there are a number of other desirable characteristics for a map matching method. The method should work in matching patches of ground that are of a reasonable size; we should not develop a solution which requires a very large observed ground footprint and thus a high flying altitude in order for the solution to work. As we don't want to restrict the UAV trajectories, it is desirable to have a map matching method that is tolerant to viewpoint change (especially camera pitch and roll) and works at different flight altitudes and across a range of camera intrinsics. In addition, in developing a map matching approach trained with data samples, we cannot assume that a large amount of UAV imagery containing all types of expected variability (seasonal apperance change, camera angle with respect to ground and camera intrinsics, flight altitudes) is available for this purpose. Finally, the approach should be such that it works in the targeted scale and is computationally feasible for an onboard deployment. As a synthesis of all these needs, we propose an approach that splits the problem in two.

We first perform a rudimentary orthoprojection of the observed UAV image such that a section of the image that corresponds to a patch on ground of a specific size, at specific ground sampling distance, is generated (see Figure 7.3). This acts as a means for abstracting away the flight altitude, camera intrinsic parameters and camera orientation with respect to ground and renders the problem of likelihood computation into that of finding likelihood between patches of orthoimages. This is beneficial also in training deep learning-based matching methods that are robust against seasonal variance as we demonstrated in earlier work [110], since our method abstracts away parameters that are irrelevant for the map matching problem (*i.e.*, camera intrinsics and altitude differences) and, unlike UAV image datasets, satellite images containing seasonal variation are plentiful.

As a second step, we compute a compact descriptor vector from the UAV image patch. We compute likelihood of each pose hypothesis by comparing that descriptor vector to a set of descriptor vectors that have been precomputed from map $\mathcal{M}$. Likelihood estimation is done by computing distance in the embedding space spun by the descriptor vectors. The choice of operating on compact descriptor vectors instead of template matching individual pose hypotheses as *e.g.*, in previous work [110] is a key enabler for fast likelihood estimation over a large map, but raises the question of how to engineer a descriptor vector computation method in a way that allows as small vector size as possible while enabling robust localization.

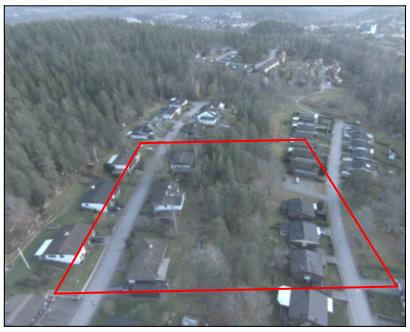
In this section, we first introduce the orthoprojection method, after which we describe the methods of computing descriptor vectors, precomputing a map and finally computing the matching likelihood.

7.5.1 *Orthoprojecting UAV image*

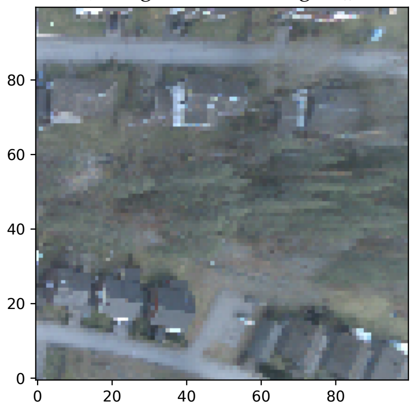
The map matching measurement is generated based on the view from a single UAV image, which may be tilted from nadir and thus contains perspective difference with respect to top-down view. To reduce the impact of this perspective change, we orthoproject the UAV image and make the assumption that the ground beneath the UAV is planar. Our approach resembles earlier work [111] with a few key differences and we report an overview of the full approach for completeness.

Based on direction of gravity estimated by AHRS, we first define local frame $\{C_{l,k}\}$ for image sampled at time t_k whose origin is at the origin of the UAV camera frame, z axis points in opposite direction to gravity, and the component of camera image plane horizontal axis perpendicular to z is aligned with y axis.

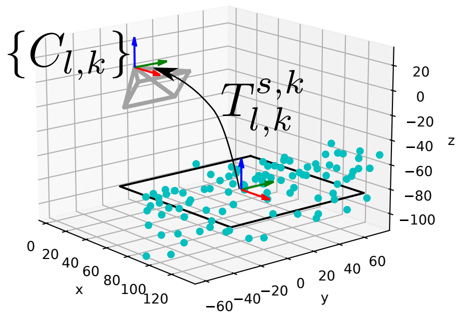
Our localization approach is based on using an orthoprojection of the camera view of the UAV. We detect image feature points using Shi-Tomasi detector [112] and track movement of the features across a batch of ten consecutive camera image frames around keyframe sampled at t_k using a pyramidal Lucas-Kanade tracker [113]. Besides image sampled at t_k, the batch consists of four images prior to the image corresponding with time t_k and five after. We estimate the 3D locations of tracked feature points in frame $\{C_{l,k}\}$ using the linear triangulation method in [114], using relative pose transformations between frames in batch that we compute from ground truth data.



(a) Original UAV image $\mathcal{I}_k$



(b) Orthoprojected image I_k



(c) Coordinate frame $\{C_{l,k}\}$, points tracked in VIO (cyan dots), square lying on plane fit to VIO landmarks and transformation $T^{s,k}_{l,k}$ to $\{C_{s,k}\}$

Figure 7.3: Example of UAV image (100m by 100m area used for orthoprojection highlighted in red), its orthoprojection, and a visualization of the coordinate frames.

Exploration of VIO frontends is beyond the focus of this work and we believe the use of noiseless relative transformations is a sufficient approximation of a generic VIO algorithm over the sequence of ten frames.

We then find the best-fitting plane whose normal is aligned with z axis; *i.e.*, we assume the ground below the UAV is planar and horizontal. We find a square with size 100m by 100m lying on the best-fitting plane that is closest to nadir and fully visible in the camera image $\mathcal{I}_k$. We project the cornerpoints of this square into the image plane of $\mathcal{I}_k$ and, by homograpy, transform the pixels corresponding with the square in $\mathcal{I}_k$ to an 100px by 100px image at 1 m/px resolution which we call I_k. I_k is the observation we use in map matching. We also compute $T^{s,k}_{l,k}$, a transformation from frame $\{C_{l,k}\}$ to a frame centered in the middle of the square, which we label $\{C_{s,k}\}$. There is no rotation between $\{C_{l,k}\}$ and $\{C_{s,k}\}$ and translation is defined such that $\{C_{s,k}\}$ is at the center of the square. An example of one orthoprojected UAV image and a visualization of the coordinate frames is shown in Figure 7.3.

The presented approach enables us to consider the motion of a UAV as a sequence of 2D translations and rotations while allowing extraction of observation I_k at correct scale independent of fight altitude, while $T^{s,k}_{l,k}$ contains information about position of its centerpoint relative to camera, including estimated ground plane altitude.

7.5.2 *From orthoimage patch to descriptor vector*

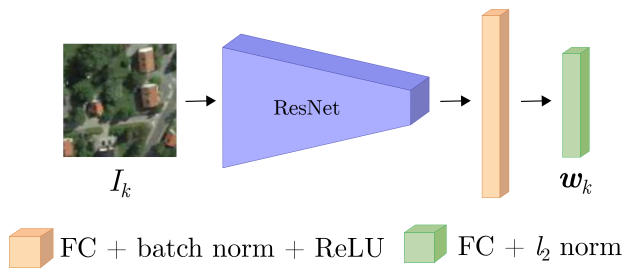
Inspired by other works projecting the UAV observation into a single compact descriptor vector [84, 95, 96], to provide a fast way to compare an observation to a large number of pose hypotheses, our map matching method is based on transforming both UAV observations and a reference map into a suitable descriptor space, and computing probability of pose hypotheses using descriptor vector values. Based on image I_k, we compute a descriptor vector $\mathbf{w}_k$ using function f_i: $\mathbf{w}_k = f_i(I_k)$. This function f_i is composed of a deep neural network backbone b such as Resnet [115] followed by a projection module m. Therefore $f_i(I_k) = m_i(b_i(I_k))$. The projection module is stacked on top the ResNet model where the the last average pooling and final fully connected layer specified in [115] are removed. The last layer of the projection module has D neurons in order to compute D-dimensional l_2-normalized descriptor vectors. The projection model thus produces vectors in unit D-sphere. Therefore, $f : \mathbb{R}^{h \times w \times c} \rightarrow \mathbb{R}^D$, where h, w and c are the height, width and channels dimensions of the input image I_k.

7.5.2.1 *Projection modules*

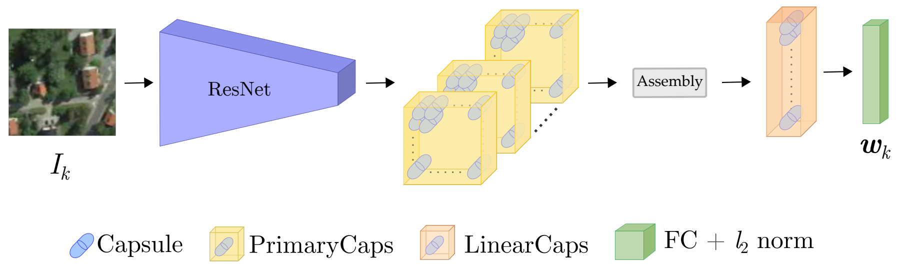
In order to extract D-dimensional embedding vectors, we used two different types of projection modules: one composed only of fully connected layers, m_{FCN}, and one composed by a Capsule Network (CapsNets) model [1], m_{CAP}. We refer to f_{FCN} and f_{CAP} as the models where the ResNet backbone is followed by m_{FCN} and m_{CAP}, respectively.

Inspired by the architecture choices proposed by [95, 96], the m_{FCN} module is composed by two fully connected layers, one with N neurons and one with D outputs. A visualization of the network structures is shown in Figure 7.4a.

CapsNets have gained great attention recently since they form more stable and robust representations with respect to input perturbations [4]. Compared to convolutional neural networks (CNNs), CapsNets also achieve greater generalization with fewer trainable parameters [1, 26]. Their main innovation lies in two major distinctions from CNNs: (i) the encoding of object poses (position, size, orientation) and visual attributes (*e.g.*, color, texture, deformation, hue) into groups of neurons called *capsules* (ii) the routing-by-agreement mechanism, which models the connections between capsules of different layers. Namely, it models the part-whole relationships among objects without losing spatial information.



(a) f_{FCN}



(b) f_{CAP}

Figure 7.4: Architectures for fully connected and capsule networks

Before this work, the seasonal invariance property of CapsNets had not been tested. But since CapsNets are well known for being robust to affine transformations, novel viewpoints, and lighting conditions [4], we investigated the potentials of CapsNets to learn season invariant features. Furthermore, in the literature [30, 65], capsule layers have been widely stacked on top of ResNet backbones instead of traditional fully connected layers to achieve better performance. Owing to these properties, extracting descriptor vectors using f_{CAP} can help to solve the UAV localization task described in this work. A visualization of f_{CAP} is shown in Figure 7.4b.

7.5.3 *Precomputing descriptors for map*

We precompute offline a map around the expected operating region. The map holds descriptor vector values that have been computed from a georeferenced orthophoto RGB bitmap $\mathcal{M}$. For each map coordinate $X_h = (x_h, y_h, \theta_h)$, we crop a w m by w m square image $I_{\mathcal{M},h}$, translated from origin by (x_h, y_h) in map coordinates and rotated by θ_h, from the map image $\mathcal{M}$. We then compute an embedding vector $\mathbf{w}_h = f_i(I_{\mathcal{M},h})$. We compute these embedding vectors using the centerpoint coordinate of each voxel in $\mathbf{X}$. This process yields a precomputed map $\mathbf{M}(i, j, l) \in \mathbb{R}^D$ where indices i, j, l correspond with indices of grid cells in our belief representation $\mathbf{X}$. We compute a separate map for each tested network architecture. Dimensions of the image in pixels, as well as chosen resolution of images, are a design parameter that can be selected based on expected use case. In the experiments section in this work we use either w =100 m or w=40 m, as reported in each experiment, and scale each image to 1 m/px resolution.

7.5.4 *Determining map matching likelihood*

To assess whether a UAV observation corresponds with a state hypothesis, we approximate

$$p(\mathcal{I}_k | s_x(i), s_y(j), s_\theta(l), \mathcal{M}) \approx p(\mathbf{w}_k | i, j, l, \mathbf{M}) \tag{7.14}$$

and compute $\mathbf{W}_{M,k}$, which contains the weight for each state space element on the chosen voxel grid.

$$\mathbf{W}_{M,k}(i,j,l) = p(\mathbf{w}_k | i, j, l, \mathbf{M}) \tag{7.15}$$

We compare two solutions for this. The first one, labeled *linear*, is similar to the choice in [95]. It assumes that Euclidean distance between the embedding obtained from the UAV and from the map $c_k(i,j,l) = \|\mathbf{w}_k - \mathbf{M}(i,j,l)\|_2$ is inversely proportional to the probability of correct pose:

$$\mathbf{W}_{M1,k}(i,j,l) = \frac{2 - c_k(i,j,l)}{2} \tag{7.16}$$

The second one is the method for computing importance factor presented in earlier work [110], where we estimate the probability density of distances in Euclidean space for true and false matches from satellite image data and compute the probability that the observation is from "match" class, for each element in $\mathbf{X}_k$ individually. We label this weighing method *bayesian* and we name weight matrix $\mathbf{W}_{M2,k}$.

7.6 EXPERIMENTS

7.6.1 *Overview of experiments*

We experiment the performance of our solution with respect to baseline methods on real-world datasets. We evaluate probabillity of convergence, time to convergence and localization error after convergence with flights taking place in two areas in Sweden. In both areas, we experiment with the problem of localization starting from 100 km^2 uncertainty. In addition, we experiment with real-time implementation on a UAV onboard computer.

For evaluating the critical design choices in our map matching approach, we vary the projection module type (fully connected or capsule network), likelihood vector dimensionality D (8, 16, 32 or 128) and likelihood conversion method (linear or bayesian), evaluate the impact of these choices on probability of convergence, time to convergence and mean localization error after convergence.

We evaluate localization performance by the criteria defined in Section 7.6.2. The datasets we use for experimentation and model training are described in Section 7.6.3. Training methods are outlined in Section 7.6.4.1, baseline methods are described in Section 7.6.5, followed by evaluation of localization performance in Section 7.6.6 and learnings from real-time experiments in Section 7.6.7.

7.6.2 *Evaluation criteria in localization experiments*

7.6.2.1 *Translation error*

We compute the estimated (x, y)-coordinates and heading using equation 7.17a and equation 7.17b, respectively.

$$\widehat{X}_{s,k}^{xy} = \sum_{i,j,l} \mathbf{X}(i,j,l) \begin{bmatrix} \bar{s}_x(i) & \bar{s}_y(j) \end{bmatrix}^T \tag{7.17a}$$

$$\widehat{X}_k^{\theta} = atan2(\sum_{i,j,l} \mathbf{X}(i,j,l) sin(\bar{s}_\theta(l)), \sum_{i,j,l} \mathbf{X}(i,j,l) cos(\bar{s}_\theta(l))) \tag{7.17b}$$

Here, $\bar{s}_x(i)$, $\bar{s}_y(j)$ and $\bar{s}_\theta(l)$ are the centerpoint coordinates of voxel corresponding to indices (i, j, l) and $atan2$ is the 2-argument arctangent. We then transform these estimates from $\widehat{X}_{s,k}^{xy}$ to drone-centric coordinates $\widehat{X}_k^{xy}$.

We compute the Euclidean distance to ground-truth (x, y)-coordinates $X_{k,gt}^{xy}$:

$$\widetilde{X}_k^{xy} = \|\widehat{X}_k^{xy} - X_{k,gt}^{xy}\|_2 \tag{7.18}$$

We also compute $\sigma_{\widehat{xy}}$, the weighted standard deviation of Euclidean distances to $\widehat{X}_k^{xy}$ in (x, y) plane, weighing with $\mathbf{X}_k$. We claim that $\sigma_{\widehat{xy}}$ is a suitable measure of convergence of the localization solution, and a large spread of uncertainty signals the need for re-initialization of the estimator.

7.6.3 *Datasets*

7.6.3.1 *Satellite images*

We trained our networks on 100m by 100m samples at 1 m/pixel randomly drawn from Google Earth satellite images of size 4800 by 2987 meters collected from 9 regions, in arbitrarily selected places in southern Finland that cover urban and non-urban areas. For each region, 4 to 15 satellite images collected from the same area at different times, containing seasonal variation, are used. The Google Earth datasets are the same as in an earlier work [110].

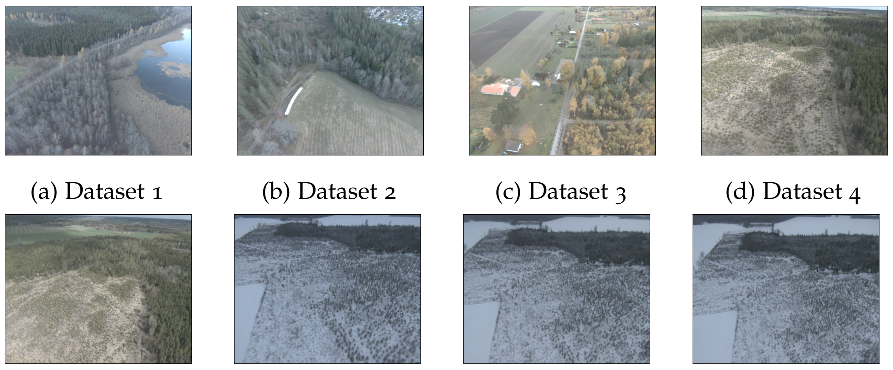

(a) Dataset 1 (b) Dataset 2 (c) Dataset 3 (d) Dataset 4
(e) Dataset 5 (f) Dataset 6 (g) Dataset 7 (h) Dataset 8

Figure 7.5: Example images from datasets. Note difference in seasonal appearance.

7.6.3.2 *UAV images*

Our experiments are run on datasets that have been collected with a UAV in two locations in Sweden at different times[1]. The dataset contains a sequence of posed images sampled at 10 Hz. Ground-truth camera trajectory is collected with a real-time kinematic (RTK)-corrected global positioning system (GPS) in uninterfered conditions. The use of RTK-GNSS ensures position precision in the centimeter range. A listing of the flight experiments is given in Table 7.1. A representative image of each dataset is shown in Figure 7.5 and flight trajectories over an orthophoto map are shown in Figure 7.6. The flights take place over terrains with forest areas, a lake, agricultural fields and some residential areas. At the time of running this experiment, IMU data was not available.

7.6.3.3 *Maps used for localization*

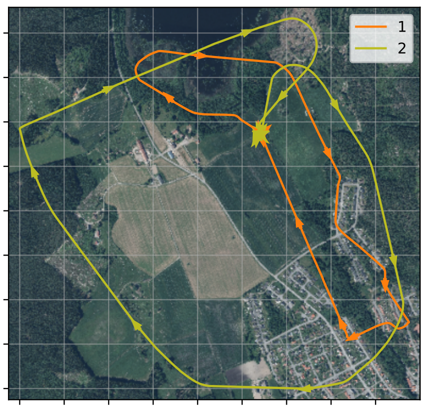
Each original map $\mathcal{M}$ is an orthophoto bitmap constructed from aerial images taken over the operating area in summer 2021. We use orthophoto bitmaps with an original ground sampling distance of 0.16 m/px provided by a local map information supplier[2].

7.6.4 *Training descriptor generator networks*

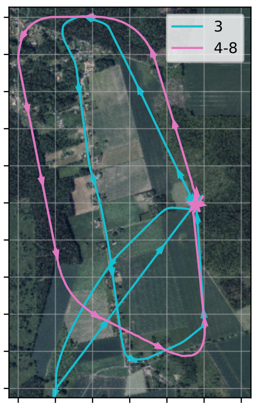
7.6.4.1 *Training details*

We trained the models f_{FCN} and f_{CAP} only on satellite images.[3] At each epoch, 45k training satellite samples are generated from the 9 areas. Each sample represents a location and orientation on a map. We

1 Data provided by Saab Dynamics Ab.
2 © Lantmäteriet, `https://www.lantmateriet.se/`.
3 We acknowledge the computational resources provided by the Aalto Science-IT project.



(a) Kisa



(b) Klockrike

Figure 7.6: Flight trajectories of each dataset. Grid spacing 200 m. Datasets 4-8 have same trajectory with varying altitude and season.

Table 7.1: Characteristics of flight datasets. Trajectory lengths are computed along (x, y) plane, and camera angles between nadir and camera principal axis. Altitude is with respect to starting position.

#	Area	Date	Traj. length (km)	Alt. (m)	Median camera angle [range] (°)
1	Kisa	2019-11-07	4.1	91	59.0 [56.9 . . . 67.7]
2	Kisa	2019-11-07	6.1	92	52.1 [49.8 . . . 56.0]
3	Klockrike	2019-10-18	6.8	92	51.2 [49.9 . . . 53.9]
4	Klockrike	2020-04-29	4.8	53	59.3 [57.8 . . . 60.5]
5	Klockrike	2020-04-29	4.8	92	59.9 [58.4 . . . 61.3]
6	Klockrike	2021-01-19	4.9	54	58.9 [58.0 . . . 60.4]
7	Klockrike	2021-01-19	4.7	71	58.7 [57.7 . . . 60.0]
8	Klockrike	2021-01-19	4.9	84	58.4 [56.9 . . . 60.9]

draw 5 images corresponding with each location and orientation. For each batch, we randomly select 10 locations. As deep metric learning for encoding the images into low dimensional vectors, we employed the triplet loss [116] with batch-all strategy and margin 0.2, using vectors representing the same location and orientation as positive samples, and vectors representing other locations and orientations in the batch as negative samples. In addition to random selection of locations and orientations, we add a random translation offset of 0 to 35 meters with uniform distribution to each location, with the intent to add robustness for observations that do not align perfectly on the map grid. With similar motivation, we add normally distributed random rotations with standard deviation of 6 degrees to the orientation of each sample before extracting the image patch from the satellite image. In addition to these, we augment the samples with Gaussian noise, motion blur, random brightness and contrast changes, and hue and saturation changes to add tolerance for changes in imaging conditions. We used Adam optimizer [47] with learning rate 10^{-6}. We found empirically that low triplet loss on satellite data leads to low loss on UAVs images and better localization performance. Therefore, we exploit the loss of the network as a proxy for localization performance during training. We let the network train for 500 epochs and employed descriptors of the model that reached the lowest loss on the validation satellite images.

7.6.4.2 *Architecture details of neural networks*

We employed a Resnet-50 model implemented in PyTorch 1.8.0 as backbone b. This network is pretrained on Places365 dataset [117]. As regards the projection module m_{FCN}, we used a fully connected layer of sizes 1024 and D, respectively. We experiment with vector length $D \in \{8, 16, 32, 128\}$. On the other hand, with regards to the module m_{CAP}, the PrimaryCaps layer is a capsule layer with 64 types of 16-dimensional capsules, which are obtained from a convolutional layer of 1024 1×1 kernel size filters. The LinearCaps layer consists of 32 32-dimensional capsules extracted running 3 iterations of the routing algorithm. We stacked on top of the LinearCaps layer a fully connected layer of size D. With the capsule projection module, we explore $D \in \{8, 16, 32\}$. For both projection modules, the D-dimensional output of the last layer is used to produce the l_2-normalized embedding vectors.

7.6.5 *Comparison methods*

We implement two methods to work as comparison methods of map matching for our approach. To provide comparable results, we utilize the same point mass filter implementation, same odometry measurements, prediction method, heading weighing method and same measurement images I_k for all methods and only replace the map matching solution with their approach.

As baseline methods for map matching, we use the solution by Mantelli et al. [93], whose formulation is scalable to large maps. To provide comparable results, we compute the descriptor vectors $\mathbf{w}_k = f_i(I_k)$ such that $\mathbf{w}_k$ is the abBRIEF descriptor. In a similar manner, we precompute a grid of abBRIEF descriptors from $\mathcal{M}_b$, using equal grid spacing as with our methods, and with abBRIEF, we compute similarity as specified in Mantelli et al. [93] and label this similarity computation method *mantelli*.

To provide a reference to a method that operates on semantic maps and allows very compact map representation over large areas, we implement the map matching solution proposed by Choi et al. [92]. We trained a U-Net [118] network on the Massachusetts Buildings Dataset [119] to segment buildings in the input images. We employ this network to extract the invariant feature descriptors introduced by Choi *et al.* [92] on satellite and UAVs images: in this setting the feature vectors $\mathbf{w}_k$ are built upon building ratio information.

7.6.6 *Localization performance*

7.6.6.1 *Experimental setting*

We perform all localization experiments on datasets that contain images collected with a UAV. In all experiments, we update our belief after at least $u_l = 50$ m of travel have occurred since the previous update, and the amount of travel after most recent update is estimated from odometry. In all experiments, we use a map grid with resolution $r_{xy} = 10$ m, $r_\theta = 6°$.

7.6.6.2 *AHRS and VIO measurements*

The datasets used in experiments do not contain IMU or magnetometer measurements and thus we have to simulate them. We simulate odometry measurements by random sampling from distribution equation 7.5. The translation standard deviation $\sigma_{u,xy}$ is approximated as 0.05 m per meter of travel and heading standard deviation $\sigma_{u,\theta}$ as $0.15°$ per meter of travel, which approximately correspond with the performance of VIO algorithms reported in literature [78].

For heading, we sample from the distribution specified in equation 7.7. Manufacturers of compact commercial AHRS sensors typically report RMS error of $2°$ in heading [120, 121]. In all experiments with AHRS, we simulate the heading measurement from ground-truth orientation data and assume $\sigma_v = 3°$.

Table 7.2: Probability of convergence p_c, time to convergence $\bar{k}_c$ and mean localization error after convergence $\bar{\bar{X}}_c^{xy}$ when using our methods with various design choices and when comparing to reference method.

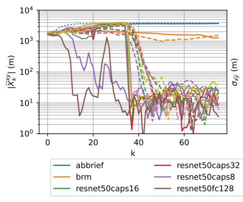
Model type	Likelihood conversion	p_c	$\bar{k}_c$	$\bar{\bar{X}}_c^{xy}(m)$
Ours, Caps, D=8	Linear	0.875	60.9	15.7
Ours, Caps, D=16	Linear	0.875	56.3	13.3
Ours, Caps, D=32	Linear	0.875	58.9	13.9
Ours, FCN, D=8	Linear	0.75	65.2	21.1
Ours, FCN, D=16	Linear	0.875	61.1	13.9
Ours, FCN, D=32	Linear	0.875	63.1	12.9
Ours, FCN, D=128	Linear	0.875	62.3	11.2
Ours, Caps, D=8	Bayesian	1.0	44.4	18.3
Ours, Caps, D=16	Bayesian	1.0	35.2	12.8
Ours, Caps, D=32	Bayesian	1.0	30.8	15.0
Ours, FCN, D=8	Bayesian	0.875	43.0	18.7
Ours, FCN, D=16	Bayesian	1.0	36.1	14.6
Ours, FCN, D=32	Bayesian	1.0	33.2	15.7
Ours, FCN, D=128	Bayesian	1.0	23.2	12.6
BRM	Linear	0.0	N/A	N/A
BRM	Bayesian	0.0	N/A	N/A
abBRIEF	Mantelli	0.625	63.8	4112.2
abBRIEF	Bayesian	0.0	N/A	N/A

7.6.6.3 *Evaluating localization performance*

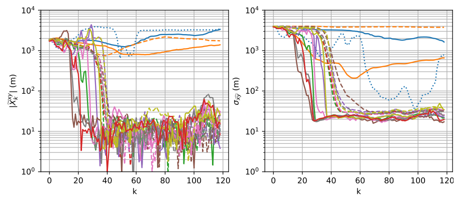
We evaluate localization performance for each step k after completing prediction, map matching and AHRS update at that step. We define that a localization solution has converged when the translation standard deviation $\sigma_{\widehat{xy}}$ is less than 100 m. We compute the mean number of updates to convergence $\bar{k}_c$ and mean translation error in converged state, $\bar{\bar{X}}_c^{xy}$, for each tested model and likelihood conversion method. We compute the proportion of flights where each compared solution converges, p_c, and tabulate results in Table 7.2. In addition, we visualize the translation error and standard deviation of (x, y) translation in each case in Figure 7.7.

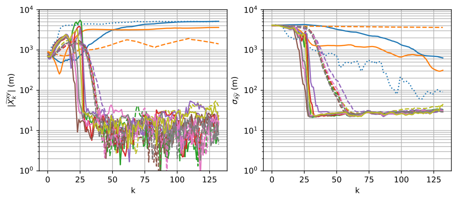
7.6.6.4 *Statistical significance testing*

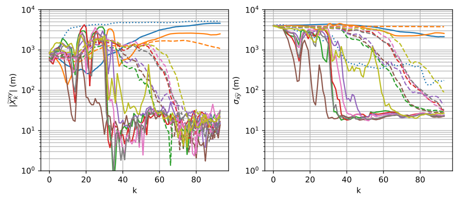
To explore what significance our results show in the various parameters we have used in the experiment configurations, we run a type II analysis of variance test on time to convergence and on localization error after convergence. We consider the embedding dimension (8, 16, 32, 128), projection module type (fully connected or capsules) and

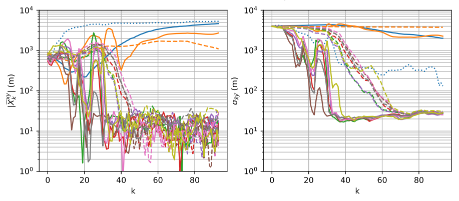


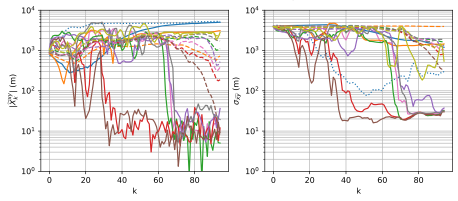
(a) Dataset 1

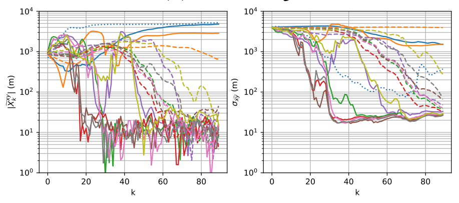

(b) Dataset 2


(c) Dataset 3


(d) Dataset 4


(e) Dataset 5


(f) Dataset 6


(g) Dataset 7

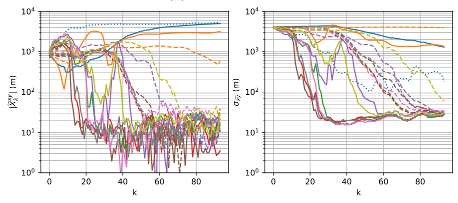

(h) Dataset 8

Figure 7.7: Translation error $|\widetilde{X}_k^{xy}|$ and standard deviation of translation $\sigma_{\widehat{xy}}$ as function of update index k in different datasets using all methods. Line color represents descriptor computation method, line style (dashed, dotted, solid) represents likelihood computation method. Logarithmic scale. Updates are made approximately every 50 meters of travel.

likelihood computation method (linear or bayesian) and combinations of these parameters. We use $p = 0.05$ as limit for significance.

For time to convergence, we reject null hypothesis that chosen likelihood method is not significant ($p = 4.2 * 10^{-16}$). For localization error after convergence, we reject null hypothesis that the parameter is not significant for embedding dimension ($p = 2.5 * 10^{-14}$), projection module type ($p = 2.1 * 10^{-2}$), likelihood method ($p = 3.7 * 10^{-4}$) and combination of embedding dimension and projection module type ($p = 5.0 * 10^{-5}$) and combination of embedding dimension and likelihood method ($p = 1.3 * 10^{-2}$).

7.6.7 *Real-time experiment*

To understand the applicability of our solution to an embedded system, we implemented a version of our algorithm running in real time on an embedded computer on an example UAV. We use the Nokia Drone Networks drone (see Figure 7.8) carrying a camera gimbal. The drone was equipped with an NVidia Jetson Nano computer on which the localization algorithm was run. The algorithm was implemented in Python and we used ROS [122] for inter-process communication. To accommodate the smaller observable ground footprint due to the narrower field of view of the drone camera at the selected flight altitude and observation parameters (50 m altitude at 45 degree camera pitch), we retrained a model with Resnet50 + FCN, D=16 architecture to work on 40m by 40m images at 1 m/px resolution. The size of the operating region and map was 1.62 km by 3.82 km. We used VINS-Mono [123] as the VIO algorithm. We used the output of a non-GNSS-corrected AHRS algorithm implemented on the drone flight controller for heading updates.

The mean network inference time (run on CPU) was 1.02 s, mean prediction time was 0.83 s, map matching took 1.18 s and AHRS update took on average 0.12 s, and all steps took on average 3.15 s. The algorithm was configured to perform an update every 40 meters of travel, while the drone was flying at 5 m/s. There is room for speedup by proper parallelization of the algorithm. The algorithm was run fully onboard the UAV.

Some localization errors appeared over terrain patches with very uniform colouring due to scale drift and rotation estimation errors of the VIO algorithm which are not considered in our odometry noise model. In addition, the AHRS heading estimate from the flight controller appeared to contain non-Gaussian heading errors up to 15 degrees; to accommodate this, we used $\sigma_v = 60°$ in this experiment.

The memory footprint of the precomputed map in this experiment was 460.8 MB. On a 100 km^2 map, the precomputed map memory requirement for D=8, D=16, D=32 and D=128 are 3.5 GB, 7.0 GB, 14.0 GB and 55.9 GB, respectively, setting further practical constraints for embedded implementation at very large scale.

This experiment shows that the algorithm can be run in real time on an embedded computer carried by a drone in an operating region of typical size for drone operations.

7.7 DISCUSSION

To solve the wake-up robot problem at the presented scale, we derived a solution based on a recursive point mass filter instead of the more common particle filter approach. We believe this avoids issues with particle depletion, which is important in cases of significant initial

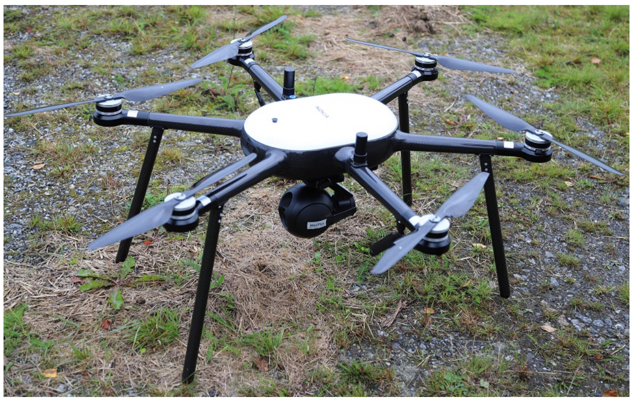


Figure 7.8: UAV used in real time experiment

uncertainty, ambiguity of terrain and potential intermittent but not random correspondence mismatches between observations and maps.

An architecture where an image observation is projected into an embedding space, with a major reduction in data dimensionality, is a key enabler for fast and memory-efficient similarity comparisons over a large number of hypotheses. Our approach for projecting data from different source domains (*i.e.*, UAV camera images and orthophoto maps or satellite images) into a common domain enables the use of learned descriptors. With our approach, extensive training data covering all expected variation is required in only one source domain without need for labeling of data. Our approach of using a learnable embedding appears to be efficient for localization over large areas containing natural and built environments, in scenarios where also significant seasonal appearance change occurs between flights. Other existing methods, *e.g.*, a handcrafted descriptor approach (abBRIEF) or learned description trained to detect pre-specified semantics (BRM), do not converge at this scale.

Bayesian likelihood conversion showed the greatest effect in time to convergence, in comparison to the more common linear approximation. The results in Table 7.2 appear to hint at the possibility that increasing embedding dimensionality leads to faster convergence, but further experimentation would be required to verify statistical significance. For localization error afer convergence, embedding dimensionality was found to be one of the statistically significant parameters. Results tabulated on Table 7.2 and error plots in Figure 7.7 seem to suggest that there may be a lower bound on localization error that appears independent from dimensionality and is most likely a result of other design choices and the characteristics of the operating environment. In other words, it appears that low-dimensionality descriptors are an efficient way of expressing what is important for localization and dimensionality of description is not the main hindering factor, what comes to localization error.

In order to extract low-dimensionality descriptors, we tested both traditional fully connected and capsule layers. Our hypothesis was initially that capsule layers would improve localization performance, extracting better encodings thanks to their ability to model part-whole relationships and robustness to novel viewpoints. Employing capsule layers lead to a slight improvement in error after convergence in comparison to fully connected layers while using less trainable parameters, as also stated in [1]. In fact, for 16-dimensional embeddings, with fully connected layer, the network has 57M trainable parameters, while with capsule layers, it has 42M trainable parameters. Since this improvement is only marginal, we did not conduct experiments with 128-dimensional capsule embeddings, as CapsNets are also well known to be computationally very demanding in terms of memory consumption, training and inference times.

Experiments demonstrate that the proposed approach is suitable for real time implementation on a flying platform at a typical scale of UAV operations. Tailoring design parameters allows the implementation of the solution on a very resource-constrained platform and running it in real time together with an odometry system. Future work for reductions in computational requirements may include *e.g.*, North-aligning UAV observations before map matching and prediction, thereby removing the need for heading estimation from the filtering problem, which would significantly (60-fold) reduce computational and memory requirements of map matching and prediction. However, in this approach, a dependable heading estimation method would be required at all times. For reducing running time of forward pass of the neural network during flight, pruning and quantization strategies can be considered.

Our work focuses on the UAV application context. In considering applicability of the proposed approach to ground vehicular applications, it is likely that a more elaborate means for orthoprojecting the camera observations into a top-down view is required, since the simple assumption of planar ground will lead to severe slanting of objects protruding from terrain, if the camera is moving very close to ground. However, we postulate that the concept of separating the problem of geometric appearance change due to difference in viewpoint and visual appearance change due to natural changes in environment apperance, and solving those problems in sequence, enables projecting the latter of the two problems into a domain where training data for learning-based approaches are plentiful across natural apperance change. In the traditional VPR problem setting, current solutions focus on solving both of these subproblems simultaneously, inescapably leading to the costly need to collect extensive datasets that contain both these variation types. We postulate our two-step approach, given a more roubst novel view synthesis method, would be an efficient method in extreme cross-view localization cases in ground vehicular applications, and consider this a potential future research direction.

To enable error recovery, the solution provides a measure of position uncertainty by computation of standard deviation. Upon detection of prolonged high standard deviation, this enables the triggering of reinitialization of the pose estimator. The ability to detect localization failures and recover from the loss of location information on a large scale paves way for a failure-aware, failsafe UAV localization system.

7.8 summary

We have shown that the approach utilizing our map matching method together with a point mass filter is able to resolve UAV pose even in the case of highly uninformed initialization, in conditions of significant seasonal appearance change between UAV image and map, even when flying over areas with natural ambiguity. We experiment with a map size of 100 km^2 where the proposed solution converges to a localization error of 12.6–18.7 m on average in 23.2–44.4 updates, depending on the chosen architecture, while reference methods are not able to converge to the correct pose under the same circumstances. All of these contributions show that real-time localization is possible on a large scale. Going beyond the demonstrated 100 km^2 will require being able to represent even larger hypothesis spaces. Addressing this challenge will potentially require future work in hierarchical models in order to retain the favorable characteristics of high spatial accuracy, extreme initial uncertainty, and complete coverage of the hypothesis space.

8 ENCODING ROTATION REPRESENTATIONS OF SYNTHETIC DATASETS IN QUATERNIONS-BASED DEEP LEARNING MODELS

This chapter represents the initial stages of our research, and it has been a collaborative effort with Alessandro Grassi, an undergraduate student. This chapter marks the beginning of our pursuit to achieve more robust and generalized results, bridging the gap between synthetic and real-world datasets.

8.1 INTRODUCTION

Over recent years, although artificial intelligence research has been propelled by the continuous development of novel models and advanced algorithms, one challenge remains: data. In fact, neural networks still need a massive amount of training data to generalize to new patterns, and finding suitable datasets is not trivial, assuming they are available at all. To address this issue, we leverage computer graphics techniques to generate synthetic datasets containing images of different meshes displayed in thousands of views. Each image is labeled with the object class and rotation using quaternions.

When representing rotations, employing quaternions offers several advantages over using rotation matrices. Firstly, unlike rotation matrices, Euler angles, and exponential maps, quaternions do not suffer from gimbal lock [124, 125]. Secondly, to properly represent a rotation, rotation matrices must be orthogonal, and quaternions must be unit vectors. However, neither of these properties can be guaranteed to be maintained after weight updates. At this juncture, maintaining orthogonality in rotation matrices becomes more challenging, while normalizing quaternions proves to be a better approach. Furthermore, previous research has demonstrated the success of quaternions in restoring spatial relationships [126] and extracting them from images [127, 128]. Lastly, using quaternions reduces the number of parameters required to represent a 3D rotation from 9 to 4.

Since we are dealing with rotation representations and capsule networks can capture spatial relationships and poses of objects, one question arises: *can we exploit capsule vectors to encode quaternions?* Özcan et al. [129] incorporated the aforementioned properties of quaternions and capsules, constraining the pose representation to 3D rotations between capsules since convolutional connections are translation invariant. In the object coordinate system, the intrinsic rotations between parts and whole are constant from any viewpoint. Thus, they exploit this rotation to achieve better generalization to novel viewpoints. Furhermore, Zhao et al. [130] built a capsule network that disentangles geometry from pose, paving the way for more informative descriptors and a structured latent space. They architecture allows joint object classification and orientation estimation without explicit supervision of rotations. In contrast with these works, our final goal is to exploit synthetic datasets annotated with quaternion labels to generalize to real-world datasets. This chapter is the first preliminary step towards achieving this goal, where we present a method to modify network architectures to predict rotations encoded as quaternions.

This chapter compares the performances and the limitations of several neural network architectures, including multilayer perceptrons, convolutional and capsule networks, in predicting both the orientation of rendered objects and their labels. This is achieved by grouping output neurons to encode the probability of existence for each class and a valid quaternion (unit length quaternion so that it represents a valid rotation).

8.2 BACKGROUND ON QUATERNIONS

Quaternions were developed in 1843 by William Rowan Hamilton and used to represent rotations and orientations. Even if Euler angles are more human-understandable, quaternions are computer-efficient and resolve several problems like Gimbal lock and ambiguity. Hamilton's quaternions have many applications besides computer graphics. For example, they are widely used in spacecraft and computer graphics.

Quaternions use complex numbers that can be interpreted as unit-vectors. A quaternion $\boldsymbol{q}$ is a 4-dimensional vector that can be written as

$$\boldsymbol{q} = xi + yj + zk + w \tag{8.1}$$

where x, y, z, and w are real numbers, and i, j, and k are three imaginary axes.

According to Hamilton's famous expression:

$$i^2 = j^2 = k^2 = -1, \tag{8.2}$$

$$ij = -ji = k, k = -kj = i, ki = -ik = j. \tag{8.3}$$

Quaternions are also represented as an ordered pair

$$\boldsymbol{q} = [s, \boldsymbol{v}] \tag{8.4}$$

where s is the scalar part and $\boldsymbol{v}$ is the vector part.

8.3 METHODOLOGY

8.3.1 *Datasets*

Each dataset was generated with Blender through a Python script that takes as input a configuration file that determines the composition of the scene. The configuration file is responsible for defining parameters such as:

- **Background** — Color, transparency
- **Render** — Engine, CUDA, and resolutions
- **Camera** — Name, location, and rotation
- **Lights** — Energy, location, rotation, type, energy, and color
- **Meshes** — Start location, start rotation, and object directory

In this work, each 3D model is rotated on each axis of 20 degrees, for a total of 6859 images on each object. Each image has a resolution of 128×128 pixel with a black background, and it is paired with a file that contains the rotation of the object expressed in quaternions.

Blender's fixed scene for generating datasets includes the following details: a camera in position $(x = 4, y = 0, z = 0)$ pointing to $(x = 0, y = 0, z = 0)$; a first point light in position $(x = 6, y = 6, z = 4)$ with 1000 energy; a second light point in position $(x = 0, y = 0, z = 0)$ with 1000 energy. All renderings are made with the EEVEE rendering engine integrated into Blender to significantly increase performance at the expense of a slight loss of detail, such as the absence of advanced ray tracing techniques.

8.4 CUBES AND TETRAHEDRONS

To verify our method, we first built a dataset with simple shapes, such as the cube and the tetrahedron. Table 8.1 shows some details of this dataset.

CubeTetrahedron	
Resolution	$128 \times 128 \times 3$
Classes	2
Objects per class	1
Images per objects	6859
Total images	13718

Table 8.1: Details of the CubeTetrahedron dataset.

However, predicting rotations on the x-axis when the model shows to the camera only one face (Figure 8.1) is impossible because multiple solutions can exist. For this reason, we enriched the model faces with visual information about the adjacent faces, as shown in Figure 8.2.

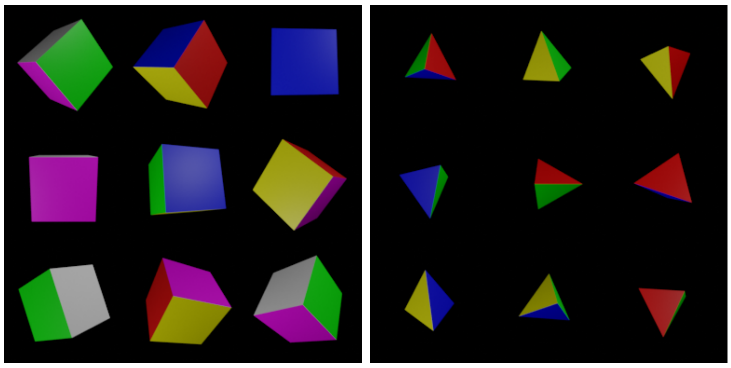


Figure 8.1: Some images of the CubeTetrahedron dataset without borders, on the left the cube, and on the right the tetrahedron.

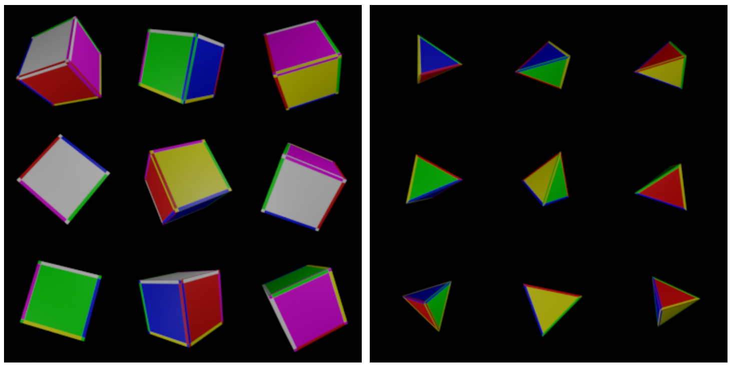


Figure 8.2: Some images of the CubeTetrahedron dataset with borders.

With this solution, a neural network can predict the rotation of a single face in a deterministic way.

8.5 TINYSHAPENET

The main dataset of this chapter is partially based on the ShapeNet [131] dataset. It is a richly-annotated, large-scale dataset of 3D objects developed by researchers from Stanford University. The repository contains over 300 million models, with 220,000 classified into 3,135 classes. However, we used a small subset of ShapeNet with 60 objects divided into 6 classes: airplanes, cars, chairs, guitars, and laptops. Table 8.2 shows some details of this dataset, while Figure 8.3 shows some samples.

The models used were adjusted with Blender to achieve a more or less uniform size across all models, large enough not to lose detail and to fit the entire object into the scene. Also, each model was modified to get the same default pose across all meshes of the same class. However, no objective definition of "default pose" applies to every object, so a subjective interpretation was given based on the class of the objects in question.

TinyShapeNet	
Resolution	128 x 128 x 3
Classes	6
Objects per class	10
Images per objects	6859
Total images	411540

Table 8.2: Details of the TinyShapeNet dataset.

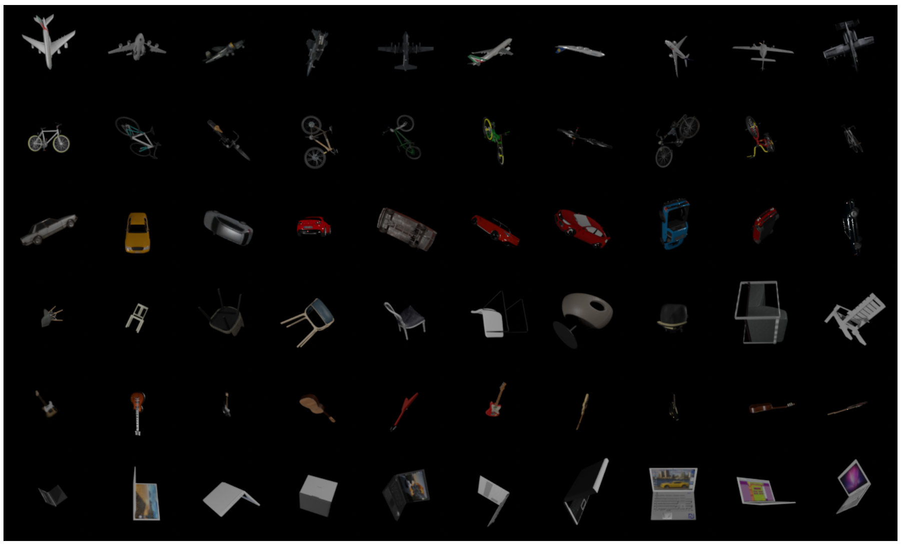


Figure 8.3: Some samples taken from the TinyShapeNet dataset.

For this work, the dataset was also generated at $1024 \times 1024 \times 4$ resolution, which is a high-resolution version with an alpha channel (transparency) to take advantage of data augmentation.

8.5.1 *Training*

Neural networks employed in this chapter are trained by means of two losses: spread loss and render loss. The spread loss is the classification loss, used to predict the object class. The render loss is used to compute the similarity between the target and predicted quaternions. In fact, each network outputs J 4-dimensional vectors that represent the predicted quaternions (one for each object label j) along with the class probabilities.

The total spread loss S is computed as the sum of the spread loss S_j for the j-th class

$$\mathcal{S} = \sum_j \mathcal{S}_j \quad \text{where} \quad \mathcal{S}_j = (\max(0, m - (a_t - a_j)))^2, \tag{8.5}$$

a_j represents the activation of the capsule of class j and a_t the activation of the target class. If the margin between the true label and the wrong class is smaller than m, we penalize it by the $m - (a_t - a_j)$ square.

The render loss is used to compare quaternions. To check if two quaternions represent the same orientation (namely, the object would be facing the same way) we define the render loss

$$\mathcal{R} = 1 - |\langle \boldsymbol{u}_j, \boldsymbol{q} \rangle|, \tag{8.6}$$

where $\boldsymbol{u}_j$ is the prediction for the class capsule j and $\boldsymbol{q}$ the target quaternion. Since q and $-q$ represent the same orientation and they are unit vectors, we compute the absolute value of their product to make sure they are the same rotation.

The total loss is defined as

$$\mathcal{L} = \mathcal{S} + \lambda\mathcal{R}. \tag{8.7}$$

8.5.2 *Architectures*

Here, we describe the neural network architectures used in this chapter, namely, multilayer perceptron (MLP), convolutional neural network (CNN), and capsule network with Expectation-Maximization routing [26] (EM-CapsNet).

The EM routing algorithm computes the activation and pose of the capsules in layer $l+1$, finding a cluster of similar votes of capsules in layer l using the Expectation-Maximization algorithm. In our case, we apply a l_2 normalization to the class capsules vectors to compute the predicted quaternions. We use directly the activations computed by the EM routing as class probabilities. For a more detailed explanation of using EM routing instead of other routing algorithms, see Section 8.5.5. Besides, for the MLP and ConvNet architectures, given the embedding of the last layer, we apply l_2 normalization to determine the quaternion and the squash function described in Equation 3.4 to shrink the embedding so that its norm determines the class activation.

8.5.3 *Multilayer perceptron*

The multilayer perceptron is the simplest network of the three used in the experiments. As shown in Table 8.3, our implementation for the TinyShapeNet dataset involves an input of $28 \times 28 \times 3$, a first hidden layer of 1024 neurons connected to a second layer of 512 neurons, which is in turn connected to a third layer of 256 neurons. Then, ending with 24 (4×6 classes) outputs.

For the CubeTetrahedron dataset test, we used an MLP with a simpler structure, with 300 neurons at the first hidden layer, and 100 neurons at the second.

Layer		Size	Act./Norm.
Input	Image	28×28×3	-
1	Hidden	1024	ReLU
2	Hidden	512	ReLU
3	Hidden	256	ReLU
Output	FC	4 × 6 classes	squash & l_2

Table 8.3: The multilayer perceptron architecture (TinyShapeNet) with corresponding layer type, tensor size, activation and normalization functions.

8.5.4 *Convolutional network*

As shown in Table 8.4, the architecture of the convolutional network is composed of a convolutional layer of size 20×20, with 256 channels, a 9×9 filter, and a stride of 1. Then, the result of the first layer passes to a second convolutional layer of size 6×6, consisting of 128 channels, with a kernel size of 9×9 and a stride of 2.

Layer		Size	Ch.	Kernel	Stride	Act./Norm.
Input	Image	28×28 × 3	-	-	-	-
1	Convolution	20×20	256	9×9	1	ReLU
2	Convolution	6×6	128	9×9	2	ReLU
3	Avg. Pooling	1×1	-	-	-	-
Output	FC	4×6 classes	-	-	-	squash & l_2

Table 8.4: The architecture of the convolutional network (TinyShapeNet) with corresponding layer type, tensor size, activation and normalization functions.

8.5.5 *Capsule network*

Table 8.5 depicts the architecture used in this work for the CapsNet model. In this model, a capsule is implemented as a 4-dimensional vector to match the quaternion dimension. The first layer is a traditional ReLU convolutional layer with a size of 20×20, a kernel of 9×9 and stride of 1. Then, the result of the kernel is passed to the PrimaryCaps layer, which is responsible for computing both the poses and the activations of the primary capsules. This layer applies two convolutional layers (one for the poses and one for the activations) and a reshape operation to convert neurons into capsule-quaternions. Then, the primary capsules are given as input to the CapsClass layer, where, thanks to the EM routing algorithm, it outputs the quaternions predictions for each class and the corresponding probability values. We chose the EM routing instead of the dynamic routing (Algorithm 1) because, in the latter one, the activation of a capsule is computed using the squash function, which normalizes the vector to a length between 0 and 1. However, in our case, since we are dealing with quaternions that are unit vectors, a separation between class prediction and quaternion prediction is needed to model the additional complexity of the proposed dataset. A visual representation of the architecture is shown in Figure 8.4

Layer		Size	Ch.	Kernel	Stride	Act.
Input	Image	28×28×3	-	-	-	-
1	Convolution	20×20	256	9×9	1	ReLU
2	PrimaryCaps (activations)	32×6×6	32	9×9	2	sigmoid
3	PrimaryCaps (poses)	4×32×6× 6	128	9×9	2	l_2
Output	CapsClass (activations)	6 classes	-	-	-	EM
Output	CapsClass (poses)	4×6 classes	-	-	-	EM

Table 8.5: The architecture of the capsule network with corresponding layer type, tensor size, activation and normalization functions.

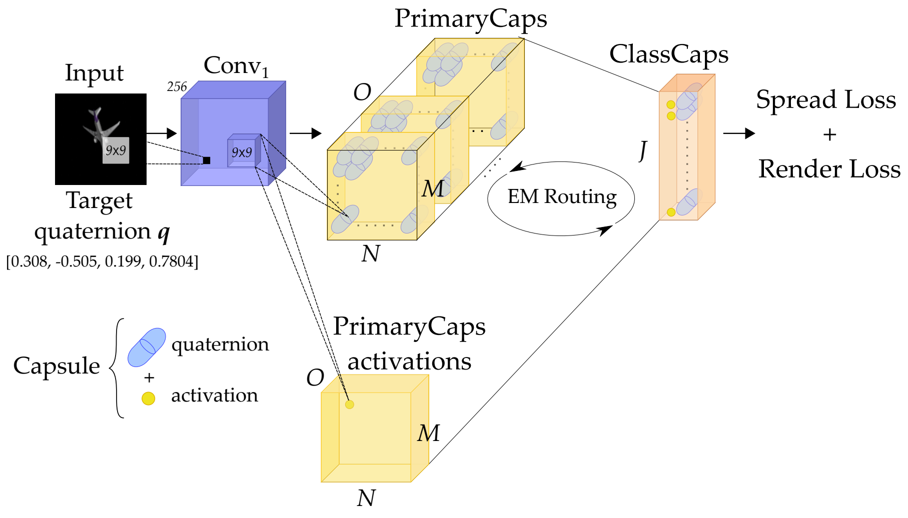


Figure 8.4: Visual representation of capsule network architecture, where a capsule encodes a quaternion representation. This model comprises a convolutional layer, a PrimaryCaps layer, which maps neurons to quaternions, and a ClassCaps layer, which outputs a capsule-quaternion for each class. The spread loss is computed using all the class capsule activations, while the render loss only considers the capsule with the highest activation, whose vector represents the predicted quaternion.

8.6 EXPERIMENTS

This section shows the experimental results achieved on CubeTetrahedron and TinyShapeNet datasets. Each network uses Adam as an optimization algorithm, with a learning rate of 0.001. The hyperparameter m of the spread loss starts as 0.2 and linearly increases by 0.1 after each epoch training. It stops growing after reaching the maximum of 0.9. Starting at a lower margin helps the training to avoid too many dead capsules during the early phase [26]. We weight the render loss by $\lambda = 0.1$. We rescaled the images to a $28 \times 28 \times 3$ resolution to dramatically decrease training times at the expense of a small loss of detail.

The training was executed on an Nvidia GTX 1080 GPU.

8.6.1 *Cube and tetrahedron*

We used CubeTetrahedron as a toy dataset to evaluate the feasibility of our method. We trained an MLP model on the dataset without and with faces. Each face enriches the cube or tetrahedron with visual information about the adjacent face. In Figure 8.5 we can see that the network, when given as input the face additional information, is able to match the ground truth quaternion better.

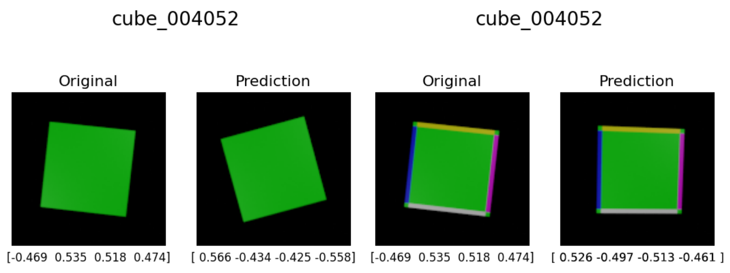


Figure 8.5: Visual differences in predicted quaternions between the two versions of the CubeTetrahedron dataset.

8.6.2 *TinyShapeNet*

As mentioned above, no data augmentation was used for this work, but each image in the dataset was rescaled to decrease network training times. All models were trained for a maximum of 1000 epochs using Early Stopping with a patience of 50, monitoring the validation score. Experimental results are shown in Table 8.6. We can see that the MLP network achieves the best performance in terms of test loss, while the ConvNet and CapsNet achieve the best accuracy values. Figure 8.6 shows some visual comparisons of the networks obtained feeding to Blender as input the predicted quaternions.

8.7 SUMMARY

In this chapter, we showed how to generate a simple synthetic dataset with quaternions and how to modify the architecture of neural networks to handle rotation predictions.

	MLP	ConvNet	CapsNet
Accuracy	0.99984	1.0	1.0
Loss	0.00182	0.00384	0.00416
Training time	4d 5h	2d 1h	9d 5h

Table 8.6: Experimental results on TinyShapeNet test set for the multilayer, convolutional and capsule networks.

Despite the limited complexity of the architecture used, the convolutional network still proved to be state-of-the-art regarding image recognition, both for accuracy and for very low training times.

Instead, the multilayer perceptron showed several limitations due to too high training times and a training subject to many fluctuations. Despite this, the accuracy does not differ much from the other results, and render loss is better minimized than other networks. Capsule networks appear promising but still immature architecture for object recognition. Its training was very time-consuming, especially when compared to the convolutional model. The slowness of capsule networks is mainly due to the iterative nature of the dynamic routing algorithm and the high number of hyperparameters. Further research needs to be done to optimize hyperparameters to improve render loss. However, despite all these problems, capsules provide a different and successful approach for dealing with generalization on new viewpoints compared to other networks.

In the future, it would be interesting to continue this research by testing more complex architectures. Then, to evaluate the goodness of the networks more objectively, it is necessary to increase the dataset's size, also through data augmentation. In addition, another good option to apply is to use uniformly distributed random rotations, rather than 20-degree steps, for the generation of the object pose. Finally, splitting the data into mutually exclusive object classes for training and testing could help evaluate the architectures.

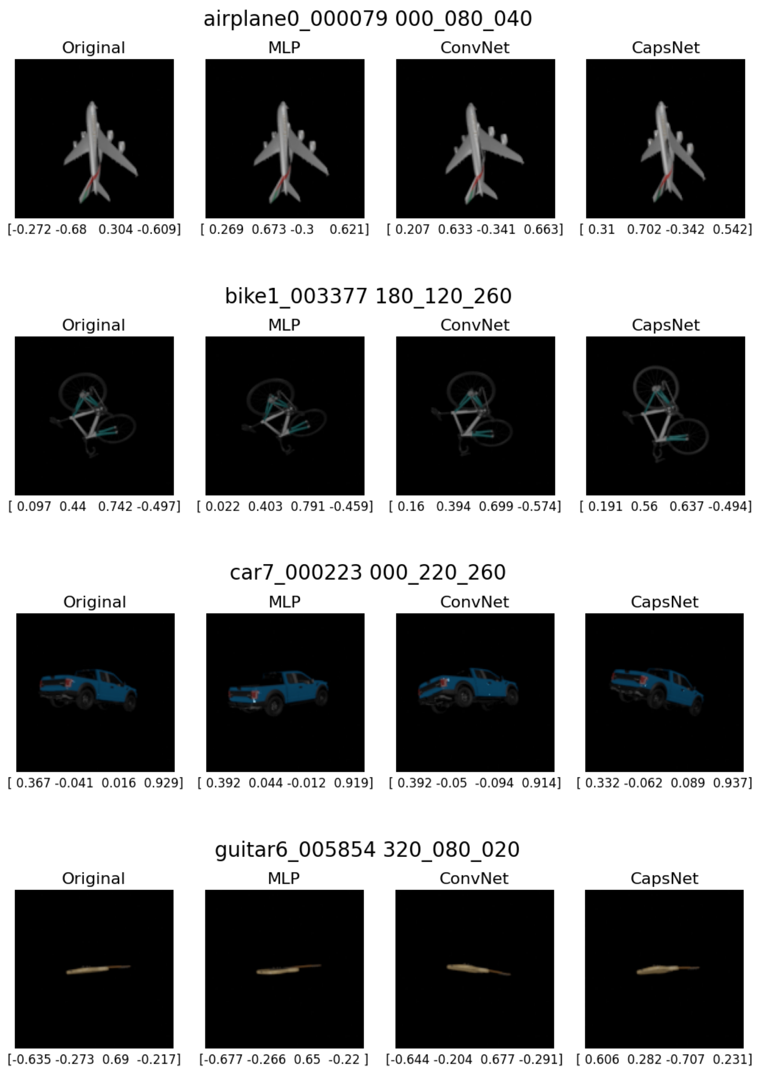


Figure 8.6: Visual comparisons of network predictions (MLP, ConvNet, and CapsNet) on TinyShapeNet.

9 LUNG NODULES SEGMENTATION WITH CAPSULE NETWORKS

During my Ph.D. studies, I worked on the EU-funded DeepHealth project[1], which focuses on applying deep learning and high-performance computing to biomedical image processing. In this context, we released the UniToChest dataset[2], a collection of anonymized chest CT scan slices coupled with the proper lung nodule segmentation map. This chapter marks the beginning of our pursuit to improve lung nodule segmentation, requiring less data using capsule networks. It represents the initial stages of our research, and it has been a collaborative effort with Paolo Peretti, a graduate student.

9.1 INTRODUCTION

In recent years, medical imaging has undergone a revolutionary transformation, primarily attributed to the remarkable progress of deep learning techniques. Deep learning has demonstrated exceptional capabilities in various domains, and its application to medical imaging has opened up new avenues for accurate and efficient diagnosis, prognosis, and treatment planning. With the ever-increasing complexity and volume of medical data, traditional image analysis methods have faced limitations in extracting meaningful insights. In contrast, deep learning models, fueled by their ability to automatically learn intricate features from data, have shown unparalleled potential in enhancing the accuracy, speed, and precision of medical image interpretation.

Lung nodules are an early indicator of lung cancer, which remains one of the most prevalent and deadly forms of cancer worldwide. With such a low survival rate of 14-15% at late stages of lung cancer, early detection and accurate diagnosis are crucial for improving patient outcomes and survival rates [132]. Deep learning models such as convolutional networks have shown tremendous promise in enhancing the accuracy and efficiency of lung cancer detection and diagnosis. Most common tasks include nodule detection [133–135], segmentation [136–139] and malignancy prediction [140–142] in chest radiographs and computed tomography (CT) scans. Despite the remarkable achievements, the application of deep learning in lung cancer imaging presents certain challenges. Images and relative annotations in fact often lack in terms of quality or quantity or both, due to the cost of acquiring and annotating the images by a radiologist. Therefore,

1 https://deephealth-project.eu/
2 https://zenodo.org/record/5797912

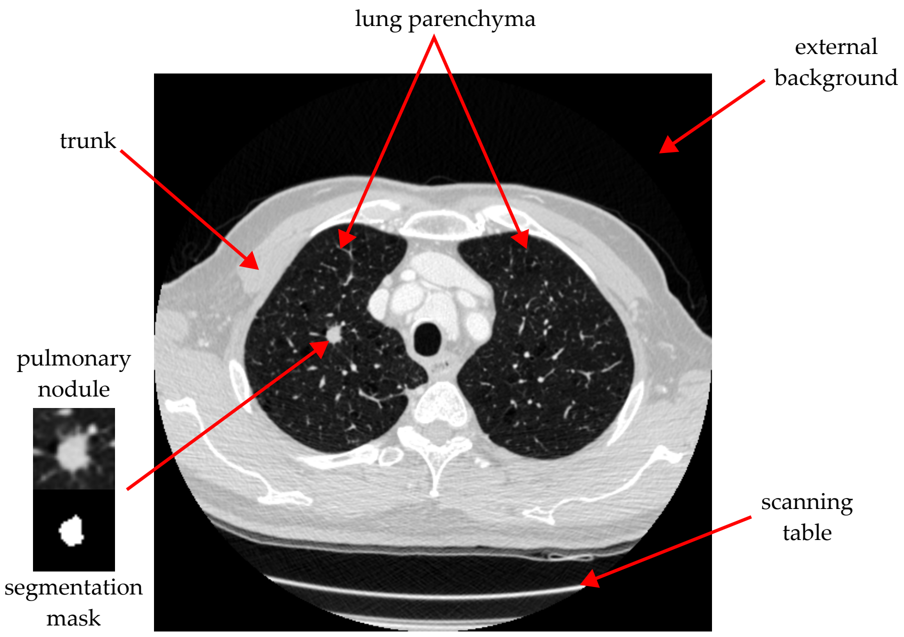


Figure 9.1: CT image of lungs with an annotated nodule segmentation mask.

industry, academia, and health organizations work together to collect and publicly release new datasets with marked-up annotated lesions such as LIDC-IDRI [143] or UniToChest [144] datasets.

Encoder-decoder models based on CNNs such as U-Net [118] are the most commonly applied to the medical image segmentation task, where the goal is to identify the pixels of organs or lesions. Though CNNs are popularly used, they have limitations due to poor robustness to affine transformations and lack of instantiation parameters (e.g., precise location information, pose, deformation, etc.). Therefore, many capsule-based models were introduced, such as SegCaps [145], Matwo-CapsNet [146], 3D-UCaps [147] and OnlyCaps-Net [148]. Capsules should be able to generalize to unseen images even when the number of images with different views used to train the networks is very low. However, to the best of our knowledge, no study has shown this property for lung nodules segmentation. To overcome this issue, this chapter aims to present a preliminary investigation on whether capsule-based methods can perform better compared to U-Net-based models in fewer data scenarios for segmenting lung nodules.

9.2 METHODOLOGY

This section describes our method to segment lung nodules, including the preprocessing stage and the architecture of CNN-based (U-Net) and capsule-based (SegCaps) networks we rely upon.

9.2.1 *Dataset and preprocessing*

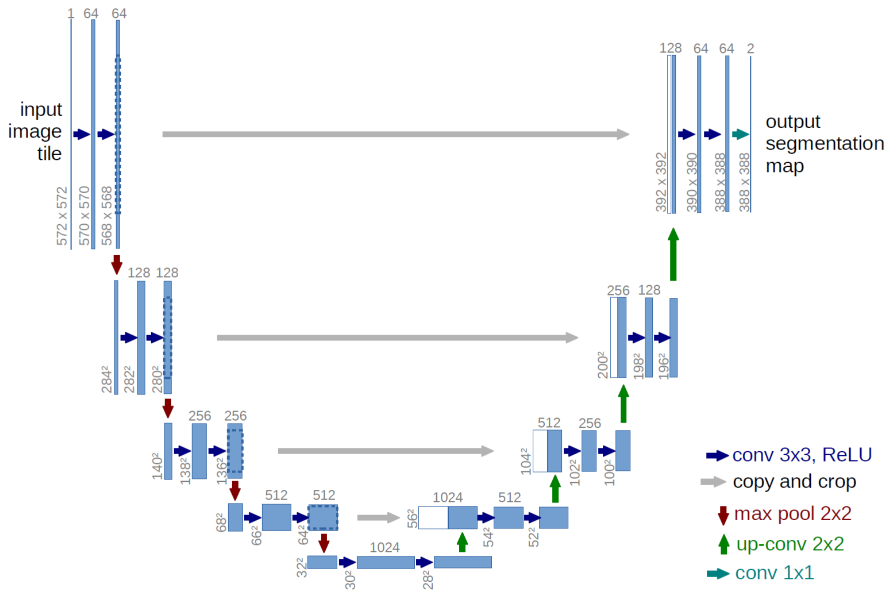
The Lung Image Database Consortium image collection [143] (LIDC-IDRI) consists of diagnostic and lung cancer screening thoracic CT scans with marked-up annotated lesions. This dataset contains more than 1000 cases (1018 cases). Each subject includes many images from a clinical thoracic CT scan and an associated XML (eXtensible Markup Language) file that records the results of the image annotation process performed by experienced radiologists, particularly by a maximum of four experienced radiologists.

Since original CT scans are in DICOM format, we preprocessed each slice, converting pixel intensity values in Hounsfield Units (HU) and then applying a windowing operation. Since this is preliminary work, we decided not to consider all the lungs but only a portion of the image of interest. Specifically, we extract several regions of interest (ROIs) of size 128×128, including a nodule, from each scan. We applied random data rotations and horizontal flips during training.

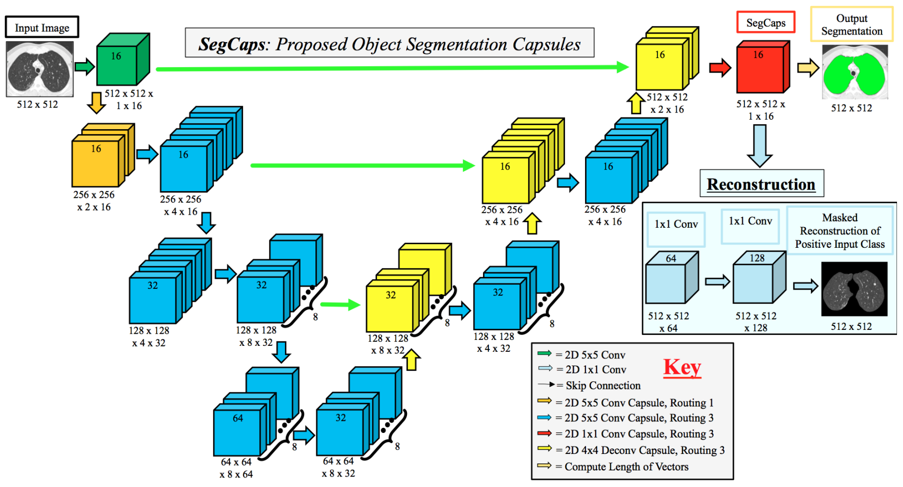
9.2.2 *Architectures*

In our experiments, we tested U-Net and SegCaps architectures, shown in Figure 9.2.

U-Net consists of a contracting path (downsampling) to capture context and a symmetric expansive path (upsampling) to generate accurate segmentation maps. U-Net is known for its "U" shape and skip connections, aiding in preserving spatial information and enabling precise object boundary localization. We chose this network because it is still widely used in biomedical image analysis and has been influential in various computer vision applications. However, the number of trainable parameters of U-Net is very high (31M). Therefore, [145] introduced SegCaps, the first neural network architecture with fewer parameters (1.4M) that employs capsule networks for image segmentation. Capsules capture spatial hierarchies and object poses, enabling a better understanding of object structures and context and potentially improving segmentation accuracy and generalization. Within SegCaps, a distinct collection of capsules is established for every pixel (x, y) present in either the image or intermediate layers. The projected segmentation label for a given location (x, y) corresponds to the identifier of the capsule from the final layer that exhibits the highest activation. SegCaps architecture is inspired by the U-Net architecture, therefore it comprises an encoder-decoder structure and skips connections. In the encoder part, convolutional capsule layers with kernel size 5×5 are stacked on top of a standard 5×5 convolutional layer to reduce the size of the input. SegCaps employ a novel dynamic routing algorithm, called locally-connected routing, where (i) children are only routed to parents within a defined spatially-local window and (ii) transfor-



(a) U-Net architecture.


Input Image
SegCaps: Proposed Object Segmentation Capsules
SegCaps
Output Segmentation
Reconstruction
1x1 Conv
1x1 Conv
Masked Reconstruction of Positive Input Class
Key
= 2D 5x5 Conv
= 2D 1x1 Conv
= Skip Connection
= 2D 5x5 Conv Capsule, Routing 1
= 2D 5x5 Conv Capsule, Routing 3
= 2D 1x1 Conv Capsule, Routing 3
= 2D 4x4 Deconv Capsule, Routing 3
= Compute Length of Vectors

(b) SegCaps architecture.

Figure 9.2: U-Net and SegCaps are the architectures used to segment lung nodules.

mation matrices are shared for each member of the grid within a capsule type. In the decoder part, SegCaps introduces the concept of deconvolutional capsules (with kernel size 4×4), which operate using transposed convolutions, routed by the locally-connected routing. The network outputs both the segmentation mask of the object and the masked reconstruction of the positive input class. In our implementation, we removed the latter because we focus only on the lung nodule and not on the lung itself.

For a fair comparison between U-Net and SegCaps regarding the number of trainable parameters, we also tested smaller versions of U-Net with four and three layers in the encoder part. Note that the original U-Net has five layers.

9.3 RESULTS

We report in Table 9.1 the results on the LIDC-IDRI test set for U-Net and SegCaps with different numbers of layers and percentages of training samples. We can see that the highest dice score is obtained by the original U-Net model with five layers when trained with 100% of the training data. As mentioned before, since the number of trainable parameters of this network is very high, we choose to compare SegCaps with shallower U-Net models. We can see that in these cases, SegCaps performs better than U-Net models with the same number of layers or lower, when trained with the full training set and even on subsets containing 10% or 50% of the training data.

Model	**Layers**	**Training samples (%)**	**Dice score**	**Parameters (M)**
U-Net		100	0.81	
U-Net	5	50	0.79	31
U-Net		10	0.78	
U-Net		100	0.74	
U-Net	4	50	0.71	7.6
U-Net		10	0.68	
U-Net		100	0.72	
U-Net	3	50	0.69	1.8
U-Net		10	0.66	
SegCaps		100	0.79	
SegCaps	4	50	0.77	1.4
SegCaps		10	0.76	

Table 9.1: Dice score results for U-Net and SegCaps with different numbers of layers and percentages of training samples (LIDC-IDRI test set).

9.4 SUMMARY

This chapter briefly investigated whether capsule-based networks can achieve better results for the lung nodules segmentation task. We showed that, despite the model with the highest performance being the original U-Net model with 31M of parameters, a SegCaps model with only 1.4M parameters shows promising results compared to smaller U-Nets with 7.6M and 1.8M parameters. These results are also confirmed in low training data scenarios. This suggests that capsule-based networks have the potential to better segment lung nodules with fewer parameters and training samples. In the future, we will also test other 2D and 3D capsule-based networks, such as 3D-UCaps and OnlyCaps.

10 CONCLUSION

This thesis has delved into the intriguing theory and applications of capsule networks, aiming to unlock their full potential in computer vision tasks. This thesis has been organized into three parts: Part i provided an overview of capsule network fundamentals, Part ii addressed some of capsule networks limitations with novel proposed methods, and Part iii demonstrated capsule network applications in real-world scenarios. We began with Chapter 2 by highlighting the limitations of traditional convolutional networks, which often lose spatial relationships due to max pooling layers and lack of an explicit entity representation. Chapter 3 offered an overview of capsule networks, deep learning models emerged as a promising solution, offering hierarchical object and part encoding, robustness to viewpoint changes, and improved feature preservation.

Despite their promise, capsule networks have not seen widespread adoption compared to traditional convolutional networks or vision transformers, possibly due to the lack of standardized architectures and pre-trained models. However, this work underscores the importance of continued research and advancements in capsule networks to fully harness their potential and address their limitations.

Throughout the thesis, we focused on three key questions to enhance our understanding and utilization of capsule networks:

1. **The routing algorithm:** In Chapter 4, we investigated the effectiveness of the routing algorithm and its necessity, especially in small-sized networks. Our experiments confirmed the importance of the routing algorithm. We introduced a novel method for annealing the number of routing iterations during training, yielding improved performance in architectures with fewer parameters.

2. **Learning effective part descriptions:** In Chapter 5, to enhance primary capsule representations, we explored the use of pruned backbones, promoting sparsity to improve computational efficiency and reduce memory and training time requirements. This approach demonstrated high generalization ability with resource savings.

3. **Part-whole relationships learning:** Understanding how capsules capture part-whole relationships is crucial. In Chapter 6, we revealed that capsules with low entropy can extract more concise and discriminative part-relationships, shedding light on

the mechanisms of part-aware representations and paving the way for more interpretable and effective capsule networks.

Additionally, we showcased the practical applications of capsule networks in autonomous localization (Chapter 7), quaternion-based rotation prediction in synthetic datasets (Chapter 8), and lung nodule segmentation in medical images ((Chapter 9), highlighting their ability to capture spatial relationships in various domains.

Therefore, this work contributes to advancing our understanding of capsule networks and their potential for revolutionizing computer vision tasks across diverse domains.

In closing, Jaini et al. [149] recently showed intriguing emergent properties of generative classifiers such as human-like shape bias in object recognition. They also understand certain perceptual illusions like those presented in Figure 2.2. It will be interesting to study how, and to what degree, diffusion and large language vision models exhibit object-centric hierarchical representations in zero-shot recognition.

IMAGES ATTRIBUTION

- Portrait of woman in d'hermine pass (Olga), 1923, Pablo Picasso
- Rabbit–duck illusion, unattributed drawing from the 23 October 1892 issue of Fliegende Blätter
- The Birth of Venus, mid 1480s, Sandro Botticelli
- Mona Lisa, 1503-1517, Leonardo da Vinci
- Self-Portrait with Thorn Necklace and Hummingbird, 1940, Frida Kahlo

BIBLIOGRAPHY

[1] Sabour, S., Frosst, N., Hinton, G. E., "Dynamic Routing Between Capsules." In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017, pp. 3856–3866.

[2] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." In: *International Conference on Learning Representations*. 2021.

[3] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021.

[4] Hinton, G. E., Krizhevsky, A., Wang, S. D., "Transforming Auto-Encoders." In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 44–51. ISBN: 978-3-642-21735-7.

[5] Biederman, I. "Recognition-by-components: A theory of human image understanding." In: *Psychological Review* 94 (1987), pp. 115–147.

[6] Hinton, G. "Some demonstrations of the effects of structural descriptions in mental imagery." In: *Cognitive Science* 3.3 (1979), pp. 231–250. ISSN: 0364-0213.

[7] Rock, I. *Orientation and Form*. Academic Press, 1973. ISBN: 9780125912501.

[8] Rock, I. "Inference in Perception." In: *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association* 1982 (1982), pp. 525–540. ISSN: 02708647.

[9] Hinton, G. E., Parsons, L. M., "Frames of reference and mental imagery." In: *Attention and performance IX* (1981), pp. 261–277.

[10] Cohen, T., Welling, M., "Group Equivariant Convolutional Networks." In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 2990–2999.

[11] Winkels, M., Cohen, T. S., "3D G-CNNs for Pulmonary Nodule Detection." In: *Medical Imaging with Deep Learning*. 2018.

[12] Veeling, B. S., Linmans, J., Winkens, J., Cohen, T., Welling, M., "Rotation Equivariant CNNs for Digital Pathology." In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*. Ed. by Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger. Cham: Springer International Publishing, 2018, pp. 210–218.

[13] Han, S., Prince, J. L., Carass, A., "Reflection-equivariant convolutional neural networks improve segmentation over reflection augmentation." In: *Medical Imaging 2020: Image Processing*. Ed. by Ivana Išgum and Bennett A. Landman. Vol. 11313. International Society for Optics and Photonics. SPIE, 2020, p. 1131337.

[14] Dangovski, R., Jing, L., Loh, C., Han, S., Srivastava, A., Cheung, B., Agrawal, P., Soljacic, M., "Equivariant Self-Supervised Learning: Encouraging Equivariance in Representations." In: *International Conference on Learning Representations*. 2022.

[15] Xie, Y., Wen, J., Lau, K. W., Ur Rehman, Y. A., Shen, J., "What Should Be Equivariant In Self-Supervised Learning." In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2022, pp. 4110–4119.

[16] Garrido, Q., Najman, L., Lecun, Y., *Self-supervised learning of Split Invariant Equivariant representations*. 2023. arXiv: 2302.10283 [cs.CV].

[17] Schwartz, E. L., Desimone, R., Albright, T. D., Gross, C. G., "Shape Recognition and Inferior Temporal Neurons." In: *Proceedings of the National Academy of Sciences of the United States of America* 80.18 (1983), pp. 5776–5778. ISSN: 00278424.

[18] Kandel, E., Schwartz, J., Jessell, T., Siegelbaum, S., Hudspeth, A., *Principles of Neural Science, Fifth Edition*. McGraw Hill LLC, 2012. ISBN: 9780071810012.

[19] Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[20] Zemel, R., Mozer, M. C., Hinton, G. E., "TRAFFIC: Recognizing Objects Using Hierarchical Reference Frame Transformations." In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989.

[21] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[22] Xiao, H., Rasul, K., Vollgraf, R., "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms." In: *CoRR* abs/1708.07747 (2017).

[23] Krizhevsky, A. *Learning multiple layers of features from tiny images*. Tech. rep. Canadian Institute for Advanced Research, 2009.

[24] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A. Y., "Reading Digits in Natural Images with Unsupervised Feature Learning." In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. 2011.

[25] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[26] Hinton, G. E., Sabour, S., Frosst, N., "Matrix capsules with EM routing." In: *International Conference on Learning Representations*. 2018.

[27] Wang, D., Liu, Q., "An Optimization View on Dynamic Routing Between Capsules." In: *ICLR*. 2018.

[28] Li, H., Guo, X., Dai, B., Ouyang, W., Wang, X., "Neural Network Encapsulation." In: *Computer Vision – ECCV 2018*. Cham: Springer International Publishing, 2018, pp. 266–282.

[29] Peer, D., Stabinger, S., Rodriguez-Sanchez, A., *Increasing the adversarial robustness and explainability of capsule networks with γ-capsules*. 2018. arXiv: 1812.09707 [cs.LG].

[30] Hahn, T., Pyeon, M., Kim, G., "Self-Routing Capsule Networks." In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019, pp. 7658–7667.

[31] De Sousa Ribeiro, F., Leontidis, G., Kollias, S., "Introducing Routing Uncertainty in Capsule Networks." In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 6490–6502.

[32] Ribeiro, F. D. S., Leontidis, G., Kollias, S. D., "Capsule Routing via Variational Bayes." In: *AAAI*. 2020, pp. 3749–3756.

[33] Edraki, M., Rahnavard, N., Shah, M., "SubSpace Capsule Network." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (2020), pp. 10745–10753.

[34] Rajasegaran, J., Jayasundara, V., Jayasekara, S., Jayasekara, H., Seneviratne, S., Rodrigo, R., "DeepCaps: Going Deeper With Capsule Networks." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 10717–10725.

[35] Gugglberger, J., Peer, D., Rodríguez-Sánchez, A., "Training Deep Capsule Networks with Residual Connections." In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Ed. by Igor Farkaš, Paolo Masulli, Sebastian Otte, and Stefan Wermter. Cham: Springer International Publishing, 2021, pp. 541–552. ISBN: 978-3-030-86362-3.

[36] Kosiorek, A., Sabour, S., Teh, Y. W., Hinton, G. E., "Stacked Capsule Autoencoders." In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019, pp. 15512–15522.

[37] Nazabal, A., Tsagkas, N., Williams, C. K. I., "Inference and Learning for Generative Capsule Models." In: *Neural Computation* 35.4 (Mar. 2023), pp. 727–761. ISSN: 0899-7667.

[38] Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., Kipf, T., "Object-Centric Learning with Slot Attention." In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 11525–11538.

[39] Hinton, G. "How to Represent Part-Whole Hierarchies in a Neural Network." In: *Neural Computation* 35.3 (Feb. 2023), pp. 413–452. ISSN: 0899-7667.

[40] Paik, I., Kwak, T., Kim, I., "Capsule Networks Need an Improved Routing Algorithm." In: *Proceedings of The Eleventh Asian Conference on Machine Learning*. Vol. 101. Proceedings of Machine Learning Research. Nagoya, Japan: PMLR, 2019, pp. 489–502.

[41] Gu, J., Tresp, V., "Improving the Robustness of Capsule Networks to Image Affine Transformations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 7283–7291.

[42] Rawlinson, D., Ahmed, A., Kowadlo, G., "Sparse Unsupervised Capsules Generalize Better." In: *CoRR* abs/1804.06094 (2018).

[43] Jeong, T., Lee, Y., Kim, H., "Ladder Capsule Network." In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3071–3079.

[44] Kakillioglu, B., Ren, A., Wang, Y., Velipasalar, S., "3D Capsule Networks for Object Classification With Weight Pruning." In: *IEEE Access* 8 (2020), pp. 27393–27405.

[45] Ribeiro, F. D. S., Duarte, K., Everett, M., Leontidis, G., Shah, M., *Learning with Capsules: A Survey*. 2022. arXiv: 2206.02664 [cs.CV].

[46] **Renzulli, R.**, Tartaglione, E., Fiandrotti, A., Grangetto, M., "Capsule Networks with Routing Annealing." In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Ed. by Igor Farkaš, Paolo Masulli, Sebastian Otte, and Stefan Wermter. 2021, pp. 529–540. ISBN: 978-3-030-86362-3.

[47] Kingma, D. P., Ba, J., "Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[48] **Renzulli, R.**, Grangetto, M., "Towards Efficient Capsule Networks." In: *2022 IEEE International Conference on Image Processing (ICIP)*. 2022, pp. 2801–2805.

[49] Zhao, Y., Birdal, T., Deng, H., Tombari, F., "3D point capsule networks." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1009–1018.

[50] Afshar, P., Mohammadi, A., Plataniotis, K. N., "Brain tumor type classification via capsule networks." In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 3129–3133.

[51] Singh, C. K., Kumar Gangwar, V., Majumder, A., Kumar, S., Ambwani, P. C., Sinha, R., "A Light-weight Deep Feature based Capsule Network." In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–8.

[52] Marchisio, A., Bussolino, B., Colucci, A., Martina, M., Masera, G., Shafique, M., "Q-CapsNets: A Specialized Framework for Quantizing Capsule Networks." In: *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*. DAC '20. Virtual Event, USA: IEEE Press, 2020. ISBN: 9781450367257.

[53] Tartaglione, E., Bragagnolo, A., Odierna, F., Fiandrotti, A., Grangetto, M., "SeReNe: Sensitivity-Based Regularization of Neurons for Structured Sparsity in Neural Networks." In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–14.

[54] Bragagnolo, A., Barbano, C. A., "Simplify: A Python library for optimizing pruned neural networks." In: *SoftwareX* 17 (2022), p. 100907. ISSN: 2352-7110.

[55] Han, S., Pool, J., Tran, J., Dally, W., "Learning both weights and connections for efficient neural network." In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[56] Tartaglione, E., Lepsøy, S., Fiandrotti, A., Francini, G., "Learning sparse neural networks via sensitivity-driven regularization." In: *Advances in neural information processing systems*. 2018, pp. 3878–3888.

[57] Tartaglione, E., Bragagnolo, A., Fiandrotti, A., Grangetto, M., "LOss-Based SensiTivity rEgulaRization: Towards deep sparse neural networks." In: *Neural Networks* 146 (2022), pp. 230–237. ISSN: 0893-6080.

[58] Li, B., Wu, B., Su, J., Wang, G., *EagleEye: Fast Sub-net Evaluation for Efficient Neural Network Pruning*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Cham, 2020.

[59] Bragagnolo, A., Tartaglione, E., Fiandrotti, A., Grangetto, M., "On the Role of Structured Pruning for Neural Network Compression." In: *2021 IEEE International Conference on Image Processing (ICIP)*. 2021, pp. 3527–3531.

[60] Tartaglione, E., Bragagnolo, A., Grangetto, M., "Pruning Artificial Neural Networks: A Way to Find Well-Generalizing, High-Entropy Sharp Minima." In: *Artificial Neural Networks and Machine Learning – ICANN 2020*. Cham: Springer International Publishing, 2020, pp. 67–78. ISBN: 978-3-030-61616-8.

[61] **Renzulli, R.**, Tartaglione, E., Grangetto, M., "Capsule Networks Do Not Need to Model Everything." Submitted to the AAAI Conference on Artificial Intelligence. 2024.

[62] Molchanov, D., Ashukha, A., Vetrov, D., "Variational Dropout Sparsifies Deep Neural Networks." In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, 2498–2507.

[63] Louizos, C., Welling, M., Kingma, D. P., "Learning sparse neural networks through L_0 regularization." In: *arXiv preprint arXiv:1712.01312* (2017).

[64] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D., "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization." In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 618–626.

[65] Gu, J. "Interpretable Graph Capsule Networks for Object Recognition." In: *Thirty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press, 2021, pp. 1469–1477.

[66] Mitterreiter, M., Koch, M., Giesen, J., Laue, S., "Why Capsule Neural Networks Do Not Scale: Challenging the Dynamic Parse-Tree Assumption." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 8. 2023, pp. 9209–9216.

[67] Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J., "Importance estimation for neural network pruning." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11264–11272.

[68] Lee, N., Ajanthan, T., Torr, P. H., "Snip: Single-shot network pruning based on connection sensitivity." In: *arXiv preprint arXiv:1810.02340* (2018).

[69] Cukierski, W. *Dogs vs. Cats*. 2013. URL: https://kaggle.com/competitions/dogs-vs-cats.

[70] Howard, J. *Imagewang*. 2019. URL: https://github.com/fastai/imagenette/.

[71] Mazzia, V., Salvetti, F., Chiaberge, M., "Efficient-CapsNet: capsule network with self-attention routing." In: *Scientific reports* 11 (2021).

[72] Keller, T. A., Welling, M., "Topographic VAEs learn Equivariant Capsules." In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 28585–28597.

[73] Everett, M., Zhong, M., Leontidis, G., *Vanishing Activations: A Symptom of Deep Capsule Networks*. 2023. arXiv: 2305.11178 [cs.CV].

[74] Kinnari, J., **Renzulli, R.**, Verdoja, F., Kyrki, V., "LSVL: Large-scale season-invariant visual localization for UAVs." In: *Robotics and Autonomous Systems* (2023), p. 104497. ISSN: 0921-8890.

[75] Psiaki, M. L., Humphreys, T. E., "GNSS Spoofing and Detection." In: *Proceedings of the IEEE* 104.6 (2016), pp. 1258–1270.

[76] Mason, R., Bonomo, J., Conley, T., Consaul, R., Frelinger, D. R., Galvan, D. A., Goldfeld, D. A., Grossman, S. A., Jackson, B. A., Kennedy, M., Koym, V. R., Mastbaum, J., Nguyen, T. L., Oberholtzer, J., Pint, E. M., Rockstroh, P., Shostak, M., Stanley, K. D., Stickells, A., Vermeer, M. J. D., Worman, S. M., *Analyzing a More Resilient National Positioning, Navigation, and Timing Capability*. Santa Monica, CA: RAND Corporation, 2021.

[77] Scaramuzza, D., Zhang, Z., "Aerial Robots, Visual-Inertial Odometry of." In: *Encyclopedia of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–9. ISBN: 978-3-642-41610-1.

[78] Delmerico, J., Scaramuzza, D., "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 2502–2509.

[79] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., Leonard, J. J., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age." In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[80] Xu, Y., Pan, L., Du, C., Li, J., Jing, N., Wu, J., "Vision-Based UAVs Aerial Image Localization: A Survey." In: *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery*. GeoAI'18. Association for Computing Machinery, 2018, 9–18. ISBN: 9781450360364.

[81] Couturier, A., Akhloufi, M. A., "A review on absolute visual localization for UAV." In: *Robotics and Autonomous Systems* 135 (2021), p. 103666. ISSN: 0921-8890.

[82] Chiu, H.-P., Das, A., Miller, P., Samarasekera, S., Kumar, R., "Precise vision-aided aerial navigation." In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 688–695.

[83] Patel, B., Barfoot, T. D., Schoellig, A. P., "Visual Localization with Google Earth Images for Robust Global Pose Estimation of UAVs." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 6491–6497.

[84] Bianchi, M., Barfoot, T. D., "UAV Localization Using Autoencoded Satellite Images." In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1761–1768.

[85] Goforth, H., Lucey, S., "GPS-Denied UAV Localization using Pre-existing Satellite Imagery." In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 2974–2980.

[86] Shan, M., Wang, F., Lin, F., Gao, Z., Tang, Y. Z., Chen, B. M., "Google map aided visual navigation for UAVs in GPS-denied environment." In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2015, pp. 114–119.

[87] Chen, S., Wu, X., Mueller, M. W., Sreenath, K., "Real-time Geo-localization Using Satellite Imagery and Topography for Unmanned Aerial Vehicles." In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 2275–2281.

[88] Li, Y., Yang, D., Wang, S., He, H., Hu, J., Liu, H., "Road-Network-Based Fast Geolocalization." In: *IEEE Transactions on Geoscience and Remote Sensing* 59.7 (2021), pp. 6065–6076.

[89] Dumble, S. J., Gibbens, P., "Airborne Vision-Aided Navigation Using Road Intersection Features." In: *Journal of Intelligent & Robotic Systems* 78 (2015), pp. 185–204.

[90] Volkova, A., Gibbens, P. W., "More robust features for adaptive visual navigation of UAVs in mixed environments." In: *Journal of intelligent & robotic systems* 90.1 (2018), pp. 171–187.

[91] Wang, T., Zhao, Y., Wang, J., Somani, A. K., Sun, C., "Attention-Based Road Registration for GPS-Denied UAS Navigation." In: *IEEE Transactions on Neural Networks and Learning Systems* 32.4 (2021), pp. 1788–1800.

[92] Choi, J., Myung, H., "BRM Localization: UAV Localization in GNSS-Denied Environments Based on Matching of Numerical Map and UAV Images." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 4537–4544.

[93] Mantelli, M., Pittol, D., Neuland, R., Ribacki, A., Maffei, R., Jorge, V., Prestes, E., Kolberg, M., "A novel measurement model based on abBRIEF for global localization of a UAV over satellite images." In: *Robotics and Autonomous Systems* 112 (2019), pp. 304–319. ISSN: 0921-8890.

[94] Calonder, M., Lepetit, V., Strecha, C., Fua, P., "BRIEF: Binary Robust Independent Elementary Features." In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792. ISBN: 978-3-642-15561-1.

[95] Samano, N., Zhou, M., Calway, A., "Global Aerial Localisation Using Image and Map Embeddings." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5788–5794.

[96] Couturier, A., Akhloufi, M. A., "Convolutional neural networks and particle filter for UAV localization." In: *Unmanned Systems Technology XXIII*. Ed. by Hoa G. Nguyen, Paul L. Muench, and Brian K. Skibba. Vol. 11758. International Society for Optics and Photonics. SPIE, 2021, pp. 108 –120.

[97] He, K., Zhang, X., Ren, S., Sun, J., "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[98] Lowry, S., Sünderhauf, N., Newman, P., Leonard, J. J., Cox, D., Corke, P., Milford, M. J., "Visual Place Recognition: A Survey." In: *IEEE Transactions on Robotics* 32.1 (2016), pp. 1–19.

[99] Naseer, T., Ruhnke, M., Stachniss, C., Spinello, L., Burgard, W., "Robust visual SLAM across seasons." In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 2529–2535.

[100] Hausler, S., Garg, S., Xu, M., Milford, M., Fischer, T., "Patch-NetVLAD: Multi-Scale Fusion of Locally-Global Descriptors for Place Recognition." In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 14136–14147.

[101] Sarlin, P.-E., Cadena, C., Siegwart, R., Dymczyk, M., "From Coarse to Fine: Robust Hierarchical Localization at Large Scale." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[102] Hamzeh, Y., Rawashdeh, S. A., "A Review of Detection and Removal of Raindrops in Automotive Vision Systems." In: *Journal of Imaging* 7.3 (2021). ISSN: 2313-433X.

[103] Tang, X., Li, Y., Wei, X., "Environmental Perception for Intelligent Vehicles." In: *AI-enabled Technologies for Autonomous and Connected Vehicles*. Ed. by Yi Lu Murphey, Ilya Kolmanovsky, and Paul Watta. Cham: Springer International Publishing, 2023, pp. 61–106. ISBN: 978-3-031-06780-8.

[104] Thrun, S. "Probabilistic robotics." In: *Communications of the ACM* 45.3 (2002), pp. 52–57.

[105] Martinelli, A. "Vision and IMU Data Fusion: Closed-Form Solutions for Attitude, Speed, Absolute Scale, and Bias Determination." In: *IEEE Transactions on Robotics* 28.1 (2012), pp. 44–60.

[106] Long, A. W., Wolfe, K. C., Mashner, M. J., Chirikjian, G. S., "The banana distribution is Gaussian: A localization study with exponential coordinates." In: *Robotics: Science and Systems VIII* 265 (2013).

[107] Burgard, W., Fox, D., Hennig, D., Schmidt, T., "Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids." In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. Portland, Oregon: AAAI Press, 1996, 896–901. ISBN: 026251091X.

[108] Bergman, N. "Recursive Bayesian estimation: Navigation and tracking applications." PhD thesis. Linköping University, 1999.

[109] Hou, H., Xu, Q., Lan, C., Lu, W., Zhang, Y., Cui, Z., Qin, J., "UAV Pose Estimation in GNSS-Denied Environment Assisted by Satellite Imagery Deep Learning Features." In: *IEEE Access* 9 (2021), pp. 6358–6367.

[110] Kinnari, J., Verdoja, F., Kyrki, V., "Season-invariant GNSS-denied visual localization for UAVs." In: *IEEE Robotics and Automation Letters* (2022), pp. 1–8.

[111] Kinnari, J., Verdoja, F., Kyrki, V., "GNSS-denied geolocalization of UAVs by visual matching of onboard camera images with orthophotos." In: *2021 20th International Conference on Advanced Robotics (ICAR)*. 2021, pp. 555–562.

[112] Shi, J., Tomasi, "Good features to track." In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600.

[113] Bouguet, J.-Y. "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm." In: *Intel corporation* 5.1-10 (2001), p. 4.

[114] Hartley, R. I., Sturm, P., "Triangulation." In: *Computer Vision and Image Understanding* 68.2 (1997), pp. 146 –157. ISSN: 1077-3142.

[115] He, K., Zhang, X., Ren, S., Sun, J., "Deep Residual Learning for Image Recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[116] Schroff, F., Kalenichenko, D., Philbin, J., "FaceNet: A unified embedding for face recognition and clustering." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 815–823.

[117] Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A., "Places: A 10 million Image Database for Scene Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[118] Ronneberger, O., Fischer, P., Brox, T., "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[119] Mnih, V. "Machine Learning for Aerial Image Labeling." PhD thesis. University of Toronto, 2013.

[120] XSens, *XSens MTi-3 Attitude, Heading and Reference System*. MTi-3. Accessed 2023-03-31. 2022. URL: https://www.xsens.com/mti-3.

[121] Vectornav, *Vectornav VN-100 IMU/AHRS*. VN-100. Accessed 2023-03-31. 2022. URL: https://www.vectornav.com/resources/datasheets/vn-100-imu-ahrs.

[122] Open Source Robotics Foundation, *Robotic Operating System*. Version ROS Noetic Ninjemys. 2020. URL: https://www.ros.org.

[123] Qin, T., Li, P., Shen, S., "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator." In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020.

[124] Grassia, F. S. "Practical Parameterization of Rotations Using the Exponential Map." In: *Journal of Graphics Tools* 3.3 (1998), pp. 29–48.

[125] Pavllo, D., Grangier, D., Auli, M., "QuaterNet: A Quaternion-based Recurrent Model for Human Motion." In: *British Machine Vision Conference (BMVC)*. 2018.

[126] Matsui, N., Isokawa, T., Kusamichi, H., Peper, F., Nishimura, H., "Quaternion neural network with geometrical operators." In: *Journal of Intelligent & Fuzzy Systems* 15 (2004). 3-4, pp. 149–164.

[127] Trabelsi, C., Bilaniuk, O., Zhang, Y., Serdyuk, D., Subramanian, S., Santos, J. F., Mehri, S., Rostamzadeh, N., Bengio, Y., Pal, C. J., "Deep Complex Networks." In: *International Conference on Learning Representations*. 2018.

[128] Gaudet, C., Maida, A., "Deep Quaternion Networks." In: *arXiv preprint arXiv:1712.04604* (2017).

[129] Özcan, B., Kinli, F., Kiraç, F., "Quaternion Capsule Networks." In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 6858–6865.

[130] Zhao, Y., Birdal, T., Lenssen, J. E., Menegatti, E., Guibas, L., Tombari, F., "Quaternion Equivariant Capsule Networks for 3D Point Clouds." In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Cham: Springer International Publishing, 2020, pp. 1–19.

[131] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F., *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[132] Knight, S. B., Crosbie, P. A., Balata, H., Chudziak, J., Hussell, T., Dive, C., "Progress and prospects of early detection in lung cancer." In: *Open Biology* 7.9 (Sept. 2017), p. 170070.

[133] Khan, S. A., Hussain, S., Yang, S., Iqbal, K., "Effective and Reliable Framework for Lung Nodules Detection from CT Scan Images." In: *Scientific Reports* 9.1 (2019), p. 4989.

[134] Zhang, H., Peng, Y., Guo, Y., "Pulmonary nodules detection based on multi-scale attention networks." In: *Scientific Reports* 12.1 (2022), p. 1466. ISSN: 2045-2322.

[135] Niu, C., Wang, G., "Unsupervised contrastive learning based transformer for lung nodule detection." In: *Physics in Medicine and Biology* 67.20 (2022), p. 204001.

[136] Liu, M., Dong, J., Dong, X., Yu, H., Qi, L., "Segmentation of Lung Nodule in CT Images Based on Mask R-CNN." In: *2018 9th International Conference on Awareness Science and Technology (iCAST)*. 2018, pp. 1–6.

[137] Yang, H., Shen, L., Zhang, M., Wang, Q., "Uncertainty-Guided Lung Nodule Segmentation with Feature-Aware Attention." In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*. Ed. by Linwei Wang, Qi Dou, P. Thomas Fletcher, Stefanie Speidel, and Shuo Li. Cham: Springer Nature Switzerland, 2022, pp. 44–54. ISBN: 978-3-031-16443-9.

[138] Qin, Y., Zheng, H., Huang, X., Yang, J., Zhu, Y.-M., "Pulmonary nodule segmentation with CT sample synthesis using adversarial networks." In: *Medical Physics* 46.3 (2019), pp. 1218–1229.

[139] Savic, M., Ma, Y., Ramponi, G., Du, W., Peng, Y., "Lung Nodule Segmentation with a Region-Based Fast Marching Method." In: *Sensors* 21.5 (2021). ISSN: 1424-8220.

[140] Balagurunathan, Y., Beers, A., Mcnitt-Gray, M., Hadjiiski, L., Napel, S., Goldgof, D., Perez, G., Arbelaez, P., Mehrtash, A., Kapur, T., Yang, E., Moon, J. W., Perez, G. B., Delgado-Gonzalo, R., Farhangi, M. M., Amini, A. A., Ni, R., Feng, X., Bagari, A., Vaidhya, K., Veasey, B., Safta, W., Frigui, H., Enguehard, J., Gholipour, A., Castillo, L. S., Daza, L. A., Pinsky, P., Kalpathy-Cramer, J., Farahani, K., "Lung Nodule Malignancy Prediction in Sequential CT Scans: Summary of ISBI 2018 Challenge." In: *IEEE Transactions on Medical Imaging* 40.12 (2021), pp. 3748–3761.

[141] Causey, J. L., Zhang, J., Ma, S., Jiang, B., Qualls, J. A., Politte, D. G., Prior, F., Zhang, S., Huang, X., "Highly accurate model for prediction of lung nodule malignancy with CT scans." In: *Scientific Reports* 8.1 (2018), p. 9286. ISSN: 2045-2322.

[142] Lei, Y., Li, Z., Shen, Y., Zhang, J., Shan, H., *CLIP-Lung: Textual Knowledge-Guided Lung Nodule Malignancy Prediction*. 2023. arXiv: 2304.08013 [cs.CV].

[143] Armato, S. G. "The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): A Completed Reference Database of Lung Nodules on CT Scans." In: *Medical Physics* 38.2 (2011), pp. 915–931.

[144] Chaudhry, H. A. H., **Renzulli, R.**, Perlo, D., Santinelli, F., Tibaldi, S., Cristiano, C., Grosso, M., Limerutti, G., Fiandrotti, A., Grangetto, M., Fonio, P., "UniToChest: A Lung Image Dataset for Segmentation of Cancerous Nodules on CT Scans." In: *Image Analysis and Processing – ICIAP 2022*. Ed. by Stan Sclaroff, Cosimo Distante, Marco Leo, Giovanni M. Farinella, and Federico Tombari. Cham: Springer International Publishing, 2022, pp. 185–196. ISBN: 978-3-031-06427-2.

[145] LaLonde, R., Xu, Z., Irmakci, I., Jain, S., Bagci, U., "Capsules for biomedical image segmentation." In: *Medical Image Analysis* 68 (2021), p. 101889. ISSN: 1361-8415.

[146] Bonheur, S., Štern, D., Payer, C., Pienn, M., Olschewski, H., Urschler, M., "Matwo-CapsNet: A Multi-label Semantic Segmentation Capsules Network." In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*. Ed. by Dinggang Shen, Tianming Liu, Terry M. Peters, Lawrence H. Staib, Caroline Essert, Sean Zhou, Pew-Thian Yap, and Ali Khan. Cham: Springer International Publishing, 2019, pp. 664–672. ISBN: 978-3-030-32254-0.

[147] Nguyen, T., Hua, B.-S., Le, N., “3D-UCaps: 3D Capsules Unet for Volumetric Image Segmentation.” In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*. Ed. by Marleen de Bruijne, Philippe C. Cattin, Stéphane Cotin, Nicolas Padoy, Stefanie Speidel, Yefeng Zheng, and Caroline Essert. Cham: Springer International Publishing, 2021, pp. 548–558. ISBN: 978-3-030-87193-2.

[148] Bonheur, S., Thaler, F., Pienn, M., Olschewski, H., Bischof, H., Urschler, M., “OnlyCaps-Net, a Capsule only Based Neural Network for 2D and 3D Semantic Segmentation.” In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*. Ed. by Linwei Wang, Qi Dou, P. Thomas Fletcher, Stefanie Speidel, and Shuo Li. Cham: Springer Nature Switzerland, 2022, pp. 340–349. ISBN: 978-3-031-16443-9.

[149] Jaini, P., Clark, K., Geirhos, R., *Intriguing properties of generative classifiers*. 2023. arXiv: 2309.16779 [cs.CV].