Capsule Networks with Routing Annealing

Capsule Networks with Routing Annealing

Riccardo Renzulli[0000−0003−0532−5966], Enzo Tartaglione[0000−0003−4274−8298], Attilio Fiandrotti[0000−0002−9991−6822], and Marco Grangetto[0000−0002−2709−7864]

Computer Science Department, University of Turin, 10149 Turin, TO, Italy
{name.surname}@unito.it

Abstract. Capsule Networks overcome some shortcomings of convolutional neural networks organizing neurons into groups of *capsules*. Capsule layers are dynamically connected by means of an iterative routing mechanism, which models the connection strengths between capsules from different layers. However, whether routing improves the network performance is still object of debate. This work tackles this issue via Routing Annealing (RA), where the number of routing iterations is annealed at training time. This proposal gives some insights on the effectiveness of the routing for Capsule Networks. Our experiments on different datasets and architectures show that RA yields better performance over a reference setup where the number of routing iterations is fixed (even in the limit case with no routing), especially for architectures with fewer parameters.

Keywords: Capsule Networks · Routing · Annealing.

1 Introduction

Capsule Networks (CapsNets) [8, 9, 16], received lots of attention lately as they tackle several shortcomings of Convolutional Neural Networks (CNNs). The human visual system is known to recognize objects (e.g., faces) decomposing them in hierarchy of parts (e.g., mouth and nose) with poses imposing coordinate frames to represent shapes [7]. While CNNs can detect the presence of objects in an image, they cannot however capture the spatial relationships between its parts mainly due to max pooling layers progressively dropping spatial information. CapsNets attempt to preserve and leverage an image representation as a hierarchy of parts with poses introducing two architectural novelties.
First, neurons are organized in groups called *capsules*, where each capsule accounts for a different *visual entity*, e.g. for a different part of an object. Then, neurons inside each capsule account each for a different property or attribute of the object such as pose (position, size, orientation) and properties (color, deformation, etc.) [16]. The output of each capsule is a vector, where the normalized vector length is the probability that the image contains the object the capsule accounts for [16].
Second, pooling layers are replaced by a *routing algorithm*, which captures the

part-objects spatial relationships between one capsule layer and the following. Unlike conventional neural networks, each capsule chooses to which capsules of the next layer to forward its output. Capsules activations are multiplied by learnable weight matrices in order to cast the votes for how the poses of the capsules of the next layer will be. The routing algorithm iteratively computes the *agreement* between the predictions of a capsule layer for the following layer. The routing algorithm outputs both the poses of the following capsule layer and the probabilities with which parts are assigned to objects. Therefore, the information flow across layers is not given by the network topology anymore, rather it is dynamically controlled by the routing algorithm.
Recently, the contribution of the routing algorithm to CapsNets generalization ability and robustness to affine transformations has been questioned [5, 14]. Typically, the number of routing iterations r is fixed once and for all during training and inference. Dropping the routing procedure is equivalent to run just one iteration (uniform routing in [14], $r = 1$) so that the coupling coefficients are not updated and they are all initialized equally. In [5, 14] it is shown that by simply averaging the predictions instead of finding the coupling coefficients between capsules through the routing algorithm yields better results. To the present date, it is not clear whether the routing algorithm improves the performance of CapsNets and what is the optimal number of iterations.
This work 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 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.
The rest of this work is organized as follows. In Sec. 2 we first provide the background on CapsNets instrumental to understanding this work, then we discuss recent literature on routing. Next, in Sec. 3 we present *Routing Annealing* (RA), our proposed training procedure for CapsNets. Finally, in Sec. 4 we experiment with RA and a reference routing algorithm over multiple datasets, highlighting the benefits of the former. Sec. 5 drawn the conclusions and discusses further developments of this work.

2 Background and Related Works

This section first describes those aspects of CapsNet instrumental to the understanding of this work, namely their architecture and the routing algorithm introduced by Sabour et al. [16]. Then, we review the literature especially related to the routing algorithm and we make some considerations on this procedure.

2.1 CapsNet Architecture

Fig. 1 shows the CapsNet architecture proposed in [16] for MNIST classification, consisting in one convolutional layer and two *capsule layers*. Due to its relevance for our work and for sake of simplicity, our overview on CapsNets will focus on this specific architecture.

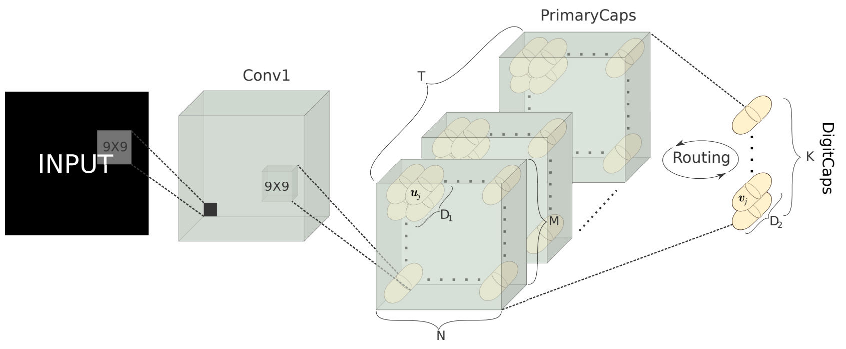


Fig. 1: CapsNet architecture described in [16]. There are one convolution layer (Conv1) and two capsule layers (PrimaryCaps and DigitCaps). The routing algorithm controls the information flow between capsule layers.

The first layer (Conv1) is a 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 a 9×9 filters [16] and $T \times D_1$ channels where T is the number of *primary capsules* types and D_1 is the dimension of a capsule vector. Overall, there are $T \times M \times N$ primary capsules. Let us denote as $\boldsymbol{u}_j$ primary capsules vectors normalized with the squashing function introduced in [16]. Each capsule is composed of a pose vector whose magnitude models the probability that the object that detects is present in the image. The output layer (*DigitCaps*) comprises of K D_2-dimensional *digit capsules* $\boldsymbol{v}_j$, one for each output class.
The information flow between primary and digit capsules is governed by a routing algorithm which aims to organize these capsules into a part-whole hierarchy. One of the most employed is the one introduced in [16]. This is an iterative procedure that computes both the poses of digit capsules from the poses of primary capsules and the strengths of the connections between the latter two. Each primary capsule predicts a pose for each digit capsule: if there are a lot predictions agreeing with each others, this means that they are in correct spatial-relationship to activate a specific digit capsule. The routing algorithm aims to find *clusters* of these agreements. Let $\boldsymbol{W}_{ij} \in \mathbb{R}^{D_1 \times D_2}$ be the matrix which projects the information flow between the i-th primary capsule and the j-th digit capsule. This can be learned through standard error gradient backpropagation. The *prediction*, or *vote*, of a primary capsule i for the digit capsule j is defined

as $\hat{\boldsymbol{u}}_{j|i} = \boldsymbol{W}_{ij}\boldsymbol{u}_i$ which is the input for the routing algorithm described in [16] and shown in Alg. 1. This procedure shows how to dynamically compute the

Algorithm 1 Dynamic routing algorithm

```
 1: procedure ROUTING($\hat{\boldsymbol{u}}_{j|i}$, $r$)
 2:     for each primary capsule $i$ and digit capsule $j$: $b_{ij} \leftarrow 0$
 3:     for $r$ iterations do
 4:         for each primary capsule $i$: $\boldsymbol{c}_i \leftarrow \text{softmax}(\boldsymbol{b}_i)$
 5:         for each digit capsule $j$: $\boldsymbol{s}_j \leftarrow \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i}(\boldsymbol{b}_i)$
 6:         for each digit capsule $j$: $\boldsymbol{v}_j \leftarrow \text{squash}(\boldsymbol{s}_j)$
 7:         for each primary capsule $i$ and digit capsule $j$: $b_{ij} \leftarrow b_{ij} + \boldsymbol{v}_j \cdot \hat{\boldsymbol{u}}_{j|i}$
 8:     end for
 9:     return $\boldsymbol{v}_j$
10: end procedure
```

poses $\boldsymbol{v}_j$ of the digit capsules given the predictions $\hat{\boldsymbol{u}}_{j|i}$. At the beginning of the routing algorithm (line 2) the logits b_{ij} are initialized equally and they are the log prior probabilities that capsule i should be coupled to capsule j. The core of the routing algorithm is depicted in lines 3-8. At every iteration, a "routing softmax" (line 4) is applied to the logits b_{ij} to obtain the corresponding coupling coefficient c_{ij}. Then, the total input $\boldsymbol{s}_j$ of capsule j of the DigitCaps layer is computed as the weighted average of the input predictions (line 5). Each vote $\hat{\boldsymbol{u}}_{j|i}$ is weighted by the corresponding coupling coefficient c_{ij}. $\boldsymbol{v}_j$ is defined as the normalized "squashed" s_j (line 6). Then each b_{ij} is refined by measuring the agreement between the output $\boldsymbol{v}_j$ of a capsule j and the prediction $\hat{\boldsymbol{u}}_{j|i}$ (line 7). Therefore, if there is a strong agreement, the corresponding link strength b_{ij} between capsules i and j is increased, decreased otherwise. Finally, after r iterations of lines 4-7, the routing algorithm output the final pose $\boldsymbol{v}_j$ for each digit capsule.

2.2 Literature review and considerations on the routing algorithm

Ever since, different routing algorithms and architectures for capsule networks have been proposed and have found applications in various tasks [2, 4, 11, 18]. We refer to routing-based CapsNets as those models that employ a routing algorithm in the architecture of the network. Hinton et al. [9] employ the Expectation-Maximization algorithm for the iterative routing procedure and build a deeper capsule network with convolutional capsule layers. Wang et al. [17] model the routing strategy as an optimization problem that minimizes a clustering-like loss and a KL divergence between the coupling distribution. Li et al. [13] reduce the computational complexity of the routing process using master and aide branches. Hahn et al. [6] describe a *self-routing* method that incorporates mixture-of-experts into capsule network models so as they do not require agreements anymore. De Sousa Ribeiro et al. [3] replace the routing algorithm with

variational inference of part-object connections in a probabilistic capsule network, leading to a significant speedup without sacrificing performance. Furthermore, Ahmed et al. [1] exploit attention modules and differentiable binary router to remove the recurrence of the routing algorithm to estimate the coupling coefficients. Lenssen et al. [12] exploit group convolutions to guarantee equivariance of pose vectors as well as invariance of output activations. Rajasegaran et al. [15] propose a deep capsule network architecture which uses a novel 3D convolution based dynamic routing algorithm aiming at improving the performance of CapsNets for more complex image datasets.
Despite of all the contributions mentioned before it is still no clear if CapsNets really need a routing algorithm. Paik et al. [14] highlight that running just one iteration of the routing algorithm (namely 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 [14]. Gu et al. [5] mitigate this problem with a simple but effective solution in which the transformation matrices are shared between all capsule types. However, in contrast with the present work, they do not change the number of iterations during the training process neither 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.

3 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.

3.1 Training with Fixed Routing

As a reference, Alg. 2 shows the standard strategy for training a CapsNet. The network parameters are optimized with standard backpropagarion 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 note 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.

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

```
procedure FIXED-ROUTING($r$, $p$)
    initialize CapsNet
    $e \leftarrow 0$;   $\mathcal{L}^\star \leftarrow 0$; $e^\star \leftarrow 0$
    while $e - e^\star < p$ do
        compute all primary capsules poses $\boldsymbol{u}_i$ and votes $\hat{\boldsymbol{u}}_{j|i}$
        compute all digit capsules poses: $\boldsymbol{v}_j \leftarrow$ ROUTING($\hat{\boldsymbol{u}}_{j|i}$, $r$)
        evaluate current loss $\mathcal{L}$ on the validation set
        if $\mathcal{L} < \mathcal{L}^\star$ then
            $\mathcal{L}^\star \leftarrow \mathcal{L}$; $e^\star \leftarrow e$
        end if
        $e \leftarrow e + 1$
        backpropagate error gradients and update parameters
    end while
    return CapsNet network of epoch $e^\star$ with the best loss value
end procedure
```

3.2 Training with Routing Annealing

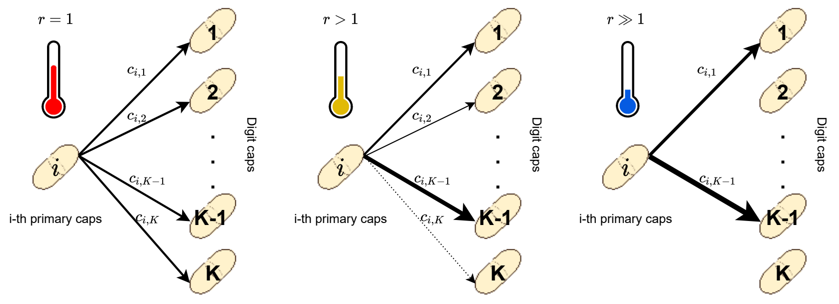
In this section we propose *Routing Annealing* (RA), an iterative method to jointly optimize the number of routing iterations $r^\star$ and the network parameters. In a nutshell, RA finds $r^\star$ adaptively during training for a given capsule architecture over a given dataset and is described in pseudo-code 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 $\mathcal{L}^\star_k$ and $e^\star_k$ the lowest losses achieved so far and the corresponding epochs for each r_k. The main difference 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 $\mathcal{L}^\star_k$ 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 $e^\star_{k-1}$ (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. 3.1, the weights need to be reinitialize for every simulation with a a different value for r.

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:     $\boldsymbol{r}$; $r \leftarrow r_0$; $\mathcal{L}_0^\star \leftarrow +\infty$; $e_0^\star \leftarrow 0$ , $e \leftarrow 0$; $k \leftarrow 0$
 3:     while $r \leq r_T$ do
 4:         compute all primary capsules poses $\boldsymbol{u}_i$ and votes $\hat{\boldsymbol{u}}_{j|i}$
 5:         compute all digit capsules poses: $\boldsymbol{v}_j \leftarrow$ Routing($\hat{\boldsymbol{u}}_{j|i}$, $r$)
 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 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:     $k^\star \leftarrow \underset{k}{\operatorname{argmin}}\, \boldsymbol{\mathcal{L}}^\star$; $r^\star \leftarrow r_{k^\star}$
17:     return CapsNet network of epoch $e_{k^\star}^\star$ and $r^\star$
18: end procedure
```

3.3 Rationale

RA takes inspiration from the *simulated annealing* (SA) algorithm, a probabilistic technique used in combinatorial-optimization problems to minimize a cost function. 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_{ij}. 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, $\frac{1}{K}$. 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_{ij} are normalized values, we can say that, for the i-th capsule $\sum_{j\in\mathcal{K}_i} c_{ij} \rightarrow 1$ and $\sum_{j\in\overline{\mathcal{K}_i}} c_{ij} \rightarrow 0$, where $\mathcal{K}_i$ is a subset of the K coupling coefficients for the i-th primary capsule and $\overline{\mathcal{K}_i}$ is its complementary set. When $r = 1$, the cardinality of $\mathcal{J}_i$ is exactly K, but increasing r, its cardinality drops to some optimal value $K_i^\star$: this means that the i-th primary capsule will be coupled to $K_i^\star$ digit capsules only, avoiding noisy coupling to the others (which are $K - K_i^\star$). A visual representation of this effect is displayed in Fig. 2. 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



(a) (b) (c)

Fig. 2: Routing coupling coefficients between the i-th primary capsule and K digit capsules. The highest the line weight, the highest the corresponding coupling coefficient. When $r = 1$ (a) the coupling coefficients have all the same value, while increasing r (b, c) a portion of the coupling parameters survives, while the others drop to zero. In the case of (c), $K^\star = 2$ and $\mathcal{K}_i = \{1, K-1\}$.

our RA strategy, observing in particular the generalization capability of the RA models compared to the other state-of-the-art approaches.

4 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, the network 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 CapsNets performance is weaker.

4.1 Experimental Setup

We experiment with the CapsNet in Fig. 1 at classifying natural images in a fully supervised scenario. We consider the MNIST, Fashion-MNIST and CIFAR10 datasets. For all datasets, 5% of the training set samples are reserved for validation. MNIST and Fashion-MNIST are composed of 28x28 images; concerning CIFAR10, 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 [16]. Our experiments consider several flavors of the architecture in Fig. 1 with different types $T \in \{1, 2, 4, 8, 16, 32\}$ and dimensions $(D_1, D_2) \in \{(2, 4), (4, 8), (8, 16)\}$ of capsules. We train the network minimizing a *margin loss* [16] with the Adam optimizer [10] 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. 3.2, 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 $r = 1$ [5] or $r = 3$ [16]).

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 $T = 1$ capsule types and vectors have dimension $D_1 = 2$ while the DigitCaps layer vectors have $D_2 = 4$ elements. This network has only 65k parameters, which helps isolating the effect of r, whereas the architecture in [16] has 6M parameters (8.2M with the decoder).
In Fig. 3 we report the learning curves for FR and RA. For FR, we train a new 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. 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 CapsNet to be overconfident on its predictions on the link strengths, preventing the network form learning the correct connections between the capsules.

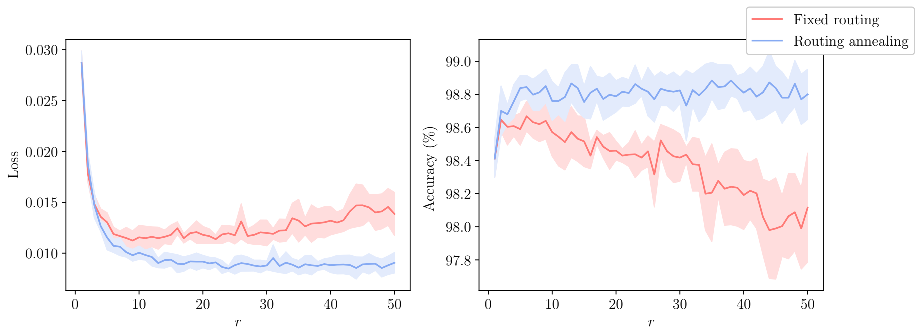
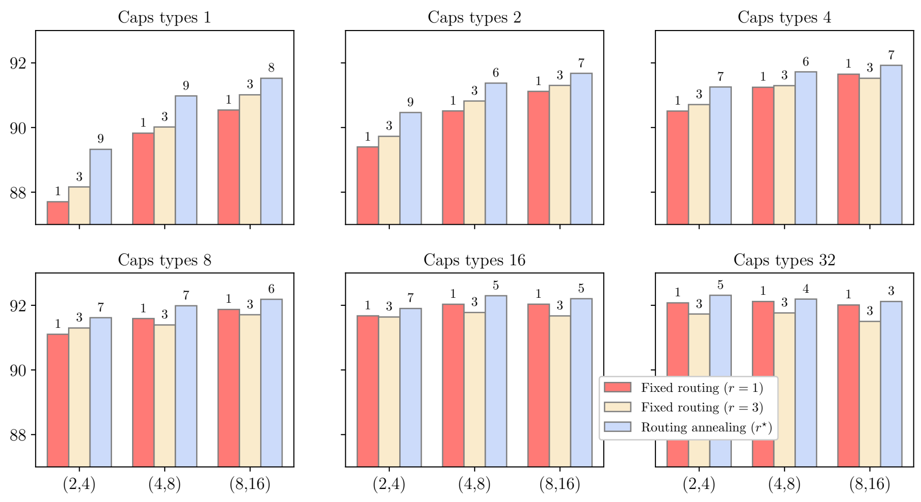


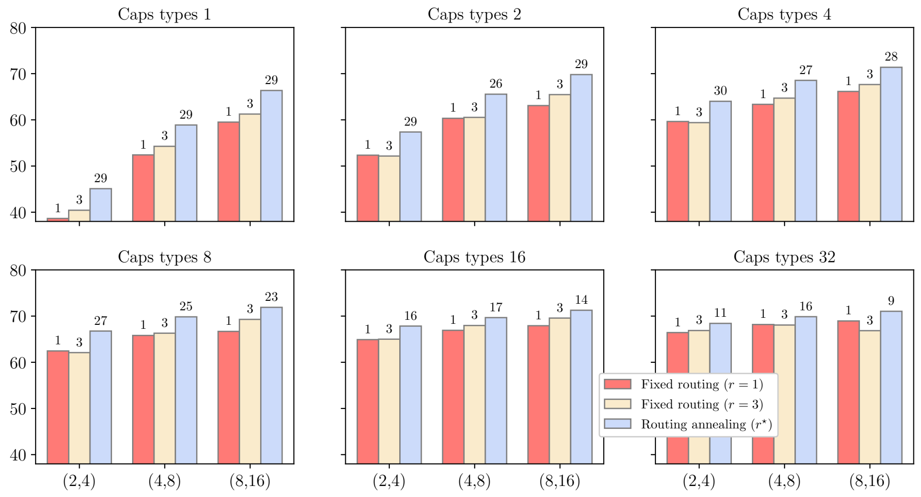
Fig. 3: Loss function (left) and classification accuracy (right) on MNIST test set for a CapsNet with $T = 1$, $D_1 = 2$, $D_2 = 4$ (means and stds of 5 seeds) as a function of the number of routing iterations r.

4.3 Results

Next, we experiment with the more complex datasets Fashion-MNIST and CIFAR10. Fig. 3 showed that RA performs better than the fixed routing reference for large r values.



(a) Fashion-MNIST



(b) CIFAR10

Fig. 4: Classification accuracy (%) on Fashion-MNIST (a) and CIFAR10 (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 $r^\star$.

For fixed routing experiments we only consider $r = 1$ and $r = 3$, as done in much of the recent literature. Fig. 4a and 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 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. 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 [16]. Second, in high-dimensional settings the same conclusions about routing as in [5] and in [14] 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. 4b shows that $r^{\star}$ for CIFAR10 is not the same as for Fashion-MNIST in Fig. 4a for identical network conditions. This means that despite $r^{\star}$ differs from dataset to dataset, nevertheless our method can find it.

5 Conclusion

In this work we proposed a novel training technique for routing-based CapsNets where the number of iterations is iteratively found at training time rather than being fixed. This work also shows experiments 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 show that this value depends heavily on the size of the network and on the dataset used. Typically, the smaller the network, the higher the number of iterations the network 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 [9].

References

1. Ahmed, K., Torresani, L.: Star-caps: Capsule networks with straight-through attentive routing. In: Advances in Neural Information Processing Systems. vol. 32,

pp. 9101–9110. Curran Associates, Inc. (2019)

2. Algamdi, A.M., Sanchez, V., Li, C.T.: Dronecaps: Recognition of human actions in drone videos using capsule networks with binary volume comparisons. In: 2020 IEEE International Conference on Image Processing (ICIP). pp. 3174–3178 (2020)
3. De Sousa Ribeiro, F., Leontidis, G., Kollias, S.: Introducing routing uncertainty in capsule networks. In: Advances in Neural Information Processing Systems. vol. 33, pp. 6490–6502. Curran Associates, Inc. (2020)
4. Duarte, K., Rawat, Y., Shah, M.: Videocapsulenet: A simplified network for action detection. In: Advances in Neural Information Processing Systems. vol. 31, pp. 7610–7619. Curran Associates, Inc. (2018)
5. 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) (June 2020)
6. Hahn, T., Pyeon, M., Kim, G.: Self-routing capsule networks. In: Advances in Neural Information Processing Systems. vol. 32, pp. 7658–7667. Curran Associates, Inc. (2019)
7. Hinton, G.: Some demonstrations of the effects of structural descriptions in mental imagery. Cognitive Science pp. 231–250 (1979)
8. Hinton, G.E., Krizhevsky, A., Wang, S.D.: Transforming auto-encoders. In: Artificial Neural Networks and Machine Learning – ICANN 2011. pp. 44–51. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
9. Hinton, G.E., Sabour, S., Frosst, N.: Matrix capsules with EM routing. In: International Conference on Learning Representations (2018)
10. 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)
11. LaLonde, R., Xu, Z., Irmakci, I., Jain, S., Bagci, U.: Capsules for biomedical image segmentation. Medical Image Analysis **68**, 101889 (2021)
12. Lenssen, J.E., Fey, M., Libuschewski, P.: Group equivariant capsule networks. In: Advances in Neural Information Processing Systems. vol. 31, pp. 8844–8853. Curran Associates, Inc. (2018)
13. Li, H., Guo, X., Dai, B., Ouyang, W., Wang, X.: Neural network encapsulation. In: Computer Vision – ECCV 2018. pp. 266–282. Springer International Publishing, Cham (2018)
14. Paik, I., Kwak, T., Kim, I.: Capsule networks need an improved routing algorithm. In: Proceedings of The Eleventh Asian Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 101, pp. 489–502. PMLR, Nagoya, Japan (17–19 Nov 2019)
15. 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). pp. 10717–10725 (2019)
16. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: Advances in Neural Information Processing Systems. vol. 30, pp. 3856–3866. Curran Associates, Inc. (2017)
17. Wang, D., Liu, Q.: An optimization view on dynamic routing between capsules. In: ICLR (2018)
18. 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 (CVPR) (June 2019)