Neural Network Encapsulation

Hongyang Li[1]* Xiaoyang Guo[1] Bo Dai[1]
Wanli Ouyang[2] Xiaogang Wang[1]

[1] The Chinese University of Hong Kong
[2] The University of Sydney

Abstract. A capsule is a collection of neurons which represents different variants of a pattern in the network. The routing scheme ensures only certain capsules which resemble lower counterparts in the higher layer should be activated. However, the computational complexity becomes a bottleneck for scaling up to larger networks, as lower capsules need to correspond to each and every higher capsule. To resolve this limitation, we approximate the routing process with two branches: a master branch which collects primary information from its direct contact in the lower layer and an aide branch that replenishes master based on pattern variants encoded in other lower capsules. Compared with previous iterative and unsupervised routing scheme, these two branches are communicated in a fast, supervised and one-time pass fashion. The complexity and runtime of the model are therefore decreased by a large margin. Motivated by the routing to make higher capsule have agreement with lower capsule, we extend the mechanism as a compensation for the rapid loss of information in nearby layers. We devise a feedback agreement unit to send back higher capsules as feedback. It could be regarded as an additional regularization to the network. The feedback agreement is achieved by comparing the optimal transport divergence between two distributions (lower and higher capsules). Such an add-on witnesses a unanimous gain in both capsule and vanilla networks. Our proposed EncapNet performs favorably better against previous state-of-the-arts on CIFAR10/100, SVHN and a subset of ImageNet.

Keywords: Network architecture design; capsule feature learning.

1 Introduction

Convolutional neural networks (CNNs) [1] have been proved to be quite successful in modern deep learning architectures [2,3,4,5] and achieved better performance in various computer vision tasks [6,7,8]. By tying the kernel weights in convolution, CNNs have the translation invariance property that can identify the same pattern irrespective of the spatial location. Each neuron in CNNs is a scalar and can detect different (low-level details or high-level regional semantics)

* Email: `yangli@ee.cuhk.edu.hk`

patterns layer by layer. However, in order to detect the same pattern with various variants in viewpoint, rotation, shape, *etc.*, we need to stack more layers, which tends to "memorize the dataset rather than generalize a solution" [9].

A capsule [10,11] is a group of neurons whose output, in form of a vector instead of a scalar, represents various perspectives of an entity, such as pose, deformation, velocity, texture, object parts or regions, *etc.* It captures the existence of a feature *and* its variant. Not only does a capsule detect a pattern but also it is trained to learn the many variants of the pattern. This is what CNNs are incapable of. The concept of capsule provides a new perspective on feature learning via instance parameterization of entities (known as capsules) to encode different variants within a capsule structure, thus achieving the feature equivariance property[3] and being robust to adversaries. Intuitively, the capsule detects a pattern (say a face) with a certain variant (it rotates 20 degree clockwise) rather than realizes that the pattern matches a variant in the higher layer.

One basic capsule layer consists of two steps: *capsule mapping* and *agreement routing*, which is depicted in Fig. 1(a). The input capsules are first mapped into the space of their higher counterparts via a transform matrix. Then the routing process involves all capsules between adjacent layers to communicate by the routing co-efficients; it ensures only certain lower capsules which resemble higher ones (in terms of cosine similarity) can pass on information and activate the higher counterparts. Such a scheme can be seen as a feature clustering and is optimized by coordinate descent through several iterations. However, the computational complexity in the first mapping step is the main bottleneck to apply the capsule idea in CNNs; lower capsules have to generate correspondence for every higher capsule (*e.g.*, a typical choice [10] is 2048 capsules with 16 dimension, resulting in 8 million parameters in the transform matrix).

To tackle this drawback, we propose an alternative to estimate the original routing summation by introducing two branches: one is the `master` branch that serves as the primary source from the direct contact capsule in the lower layer; another is the `aide` branch that strives for searching other pattern variants along the channel and replenishes side information to `master`. These two branches are intertwined by their co-efficients so that feature patterns encoded in lower capsules could be fully leveraged and exchanged. Such a one-pass approximation is fast, light-weight and supervised, compared to the current iterative, short-lived and unsupervised routing scheme.

Furthermore, the routing effect in making higher capsule have agreement with lower capsule can be extended as a direct loss function. In deep neural networks, information is inevitably lost through stack of layers. To reduce the rapid loss of information in nearby layers, a loss function can be included to enforce that neurons or capsules in the higher layer can be used for reconstructing the counterparts in lower layers. Based on this motivation, we devise an agreement feedback unit which sends back higher capsules as a feedback signal to better supervise feature learning. This could be deemed as a regularization on network. Such a feedback agreement is achieved by measuring the distance between the

[3] Equivariance is the detection of feature patterns that can transform to each other.

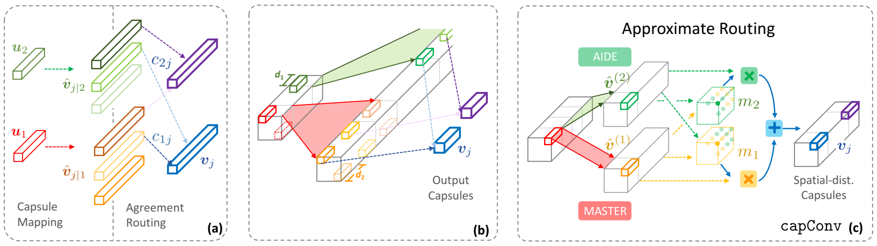

Fig. 1. (a) One capsule operation includes a capsule mapping and an agreement routing. (b) Capsule implemented in a convolutional manner by [10,11] where lower capsules are mapped into the space of *all* higher capsules and then routed to generate the output capsule. (c) Our proposed `capConv` layer: approximate routing with master and aide interaction to ease the computation burden in the current design in (b).

two distributions using optimal transport (OT) divergence, namely the Sinkhorn loss. The OT metric (*e.g.*, Wasserstein loss) is promised to be superior than other options to modeling data on general space. This add-on regularization is inserted during training and disposed of for inference. The agreement enforcement has witnessed a unanimous gain in both capsule and vanilla neural networks.

Altogether, bundled with the two mechanisms aforementioned, we (i) encapsulate the neural network in an approximate routing scheme with master/aide interaction, (ii) enforce the network's regularization by an agreement feedback unit via optimal transport divergence. The proposed capsule network is denoted as EncapNet and performs superior against previous state-of-the-arts for image recognition tasks on CIFAR10/100, SVHN and a subset of ImageNet. The code and dataset are available `https://github.com/hli2020/nn_capsulation`.

2 CapNet: Agreement Routing Analysis

2.1 Preliminary: capsule formulation

Let $\boldsymbol{u}_i, \boldsymbol{v}_j$ denote the input and output capsules in a layer, where i, j indicates the index of capsules. The dimension and the number of capsules at input and output are d_1, d_2, n_1, n_2, respectively, *i.e.*, $\{\boldsymbol{u}_i \in \mathbb{R}^{d_1}\}_{i=1}^{n_1}, \{\boldsymbol{v}_j \in \mathbb{R}^{d_2}\}_{j=1}^{n_2}$. The first step is a mapping from lower capsules to higher counterparts: $\hat{\boldsymbol{v}}_{j|i} = \boldsymbol{w}_{ij} \cdot \boldsymbol{u}_i$, where $\boldsymbol{w}_{ij} \in \mathbb{R}^{d_1 \times d_2}$ is a transform matrix and we define the intermediate output $\hat{\boldsymbol{v}}_{j|i} \in \mathbb{R}^{d_2}$ as *mapped activation* (called prediction vector in [10]) from i to j. The second step is an agreement routing process to aggregate all lower capsules into higher ones. The mapped activation is multiplied by a routing coefficient c_{ij} through several iterations in an unsupervised manner: $\boldsymbol{s}_j^{(r)} = \sum_i c_{ij}^{(r)} \hat{\boldsymbol{v}}_{j|i}$.

This is where the highlight of capsule idea resides in. It could be deemed as a voting process: the activation of higher capsules should be entirely dependent on the resemblance from the lower entities. Prevalent routing algorithms include the coordinate descent optimization [10] and the Gaussian mixture clustering via Expectation-Maximum (EM) [11], to which we refer as `dynamic` and `EM` routing, respectively. For `dynamic` routing, given $b_{ij}^{(0)} \leftarrow 0, r \leftarrow 0$, we have:

$$b_{ij}^{(r+1)} \leftarrow b_{ij}^{(r)} + \hat{\boldsymbol{v}}_{j|i} \cdot \boldsymbol{v}_j^{(r)}, \tag{1}$$

where b is the softmax input to obtain c; $\boldsymbol{v}^{(r)}$ is computed from $\boldsymbol{s}^{(r)}$ via $\texttt{squash}(\cdot)$, *i.e.*, $\boldsymbol{v} = \frac{\|\boldsymbol{s}\|^2}{1+\|\boldsymbol{s}\|^2} \frac{\boldsymbol{s}}{\|\boldsymbol{s}\|}$. The update of the routing co-efficient is conducted in a coordinate descent manner which optimizes c and $\boldsymbol{v}$ alternatively. For `EM` routing, given $c_{ij}^{(0)} \leftarrow 1/n_2, r \leftarrow 0$, and the activation response of input capsules a_i, we iteratively aggregate input capsules into d_2 Gaussian clusters:

$$a_j^{(r)}, \boldsymbol{\mu}_j^{(r)}, \boldsymbol{\sigma}_j^{(r)} \leftarrow \texttt{M-step}\big[a_i, c_{ij}^{(r)}, \hat{\boldsymbol{v}}_{j|i}\big], \tag{2}$$

$$c_{ij}^{(r+1)} \leftarrow \texttt{E-step}\big[a_j^{(r)}, p_{j|i}\big(\hat{\boldsymbol{v}}_{j|i}, \boldsymbol{\mu}_j^{(r)}, \boldsymbol{\sigma}_j^{(r)}\big)\big], \tag{3}$$

where the mean of cluster $\boldsymbol{\mu}_j$ is deemed as the output capsule $\boldsymbol{v}_j$. `M-step` generates the activation a_j alongside the mean and std w.r.t. higher capsules; these variables are further fed into `E-step` to update the routing co-efficients c_{ij}. The output from a capsule layer is thereby obtained after iterating R times.

2.2 Agreement routing analysis in CapNet

Effectiveness of the agreement routing. Figure 2 illustrates the training dynamics on routing between adjacent capsules as the network evolves. In essence, the routing process is a weighted average from all lower capsules to the higher entity (Eqn.(4)). Intuitively, given a sample which belongs to the j-th class, the network tries to optimize capsule learning such that the length (existence probability) of $\boldsymbol{v}_j$ in the final capsule layer should be the largest. This requires the magnitude of its lower counterparts who resemble capsule j should occupy a majority and have a higher length compared to others that are dissimilar to j. Take the top row of `Dynamic` case for instance. At the first epoch, the kernel weights $\boldsymbol{w}_{ij}$ are initialized with Gaussian hence most capsules are orthogonal to each other and have the same length. As training goes (epoch 20 and 80), the percentage and length of "blurring" capsules, whose cosine similarity is around zero, goes down and the distribution evolves into a polarization: the most similar and dissimilar capsules gradually take the majority and hold a higher length than other i's. As training approaches termination (epoch 200), such a phenomenon is further polarized and the network is at a stable state where the most resembled and non-resembled capsules have a higher percentage and length than the others. The role of agreement routing is to adjust the magnitude and relevance from lower capsules to higher capsules, such that the activation of relevant higher counterparts could be appropriately turned on and the pattern information from lower capsules be passed on.

The analysis for `EM` routing draws a unanimous conclusion. The polarization phenomenon is further intensified (*c.f.* (h) vs (d) in Fig. (2)). The percentage of dissimilar capsules is lower (20% vs 37%) whilst the length of similar capsules is higher (0.02 vs 0.01): implying that `EM` is potentially a better routing solution than `dynamic`, which is also verified by (a) vs (b) in Table 1.

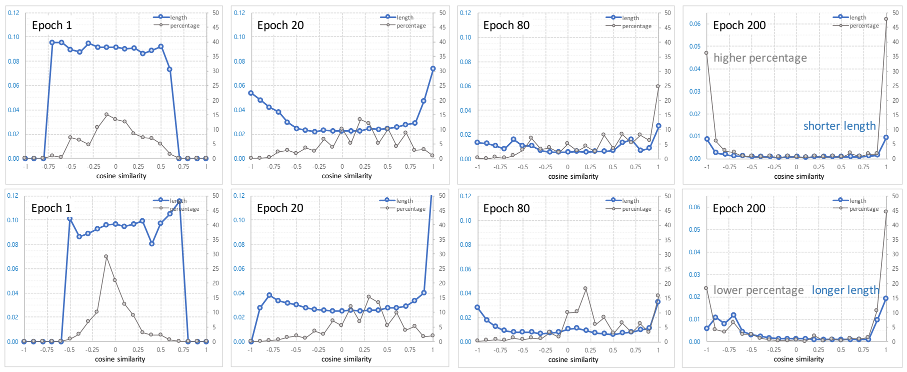

Fig. 2. Training dynamics as network evolves. Routing tends to magnify and pass on pattern variants of lower capsules to higher ones which mostly resemble the lower counterparts. **Top**: `Dynamic` routing. **Bottom**: `EM` routing. We show the cosine similarity between $\boldsymbol{v}_j$ and the mapped lower capsules, *i.e.*, $\texttt{cos\_sim}(\boldsymbol{v}_j, \hat{\boldsymbol{v}}_{j|i})$. Blue line represents the average (across all samples) length $\|\hat{\boldsymbol{v}}_{j|i}\|$ and gray indicates the percentage (%) of how many lower capsules i's agree with j at a given resemblance.

Moreover, it is observed that replacing scalar neurons in traditional CNNs with vector capsules and routing is effective, *c.f.* (a-b) vs (c) in Table 1. We adopt the same blob shape for each layer in vanilla CNNs for fair comparison. However, when we increase the parameters of CNNs to the same amount as that of CapNet, the former performs better in (d). Due to the inherent design, CapNet requires more parameters than the traditional CNNs, *c.f.* (a) vs (c) in Table 1 with around 152 Mb for CapNet vs 24 Mb for vanilla CNNs.

The capsule network is implemented in a group convolution fashion by [10,11], which is depicted in Fig. 1(b). It is assumed that the vector capsules are placed in the same way as the scalar neurons in vanilla CNNs. The *spatial* capsules in a channel share the same transform kernel since they search for the same patterns in different locations. The *channel* capsules own different kernels as they represent various patterns encapsulated in a group of neurons.

Computational complexity in CapNet. From an engineering perspective, the original design for capsules in CNN structure (see Fig. 1(b)) is to save computation cost in the capsule mapping step; otherwise it would take 64$\times$ more kernel parameters (assuming spatial size is 8) to fulfill the mapping step. However, the burden is not eased effectively since step one has to generate a mapping for *each and every* capsule j in the subsequent layer. The output channel size of the transform kernel in Tab. 1(a-b) is 1,048,576 ($16 \times 32 \times 2048$). If we feed the network with a batch size of 128 (even smaller option, *e.g.*, 32), OOM (out-of-memory) occurs due to the super-huge volume of the transform kernel. The subtle difference of parameter size between `dynamic` and `EM` is that additionally the latter has a larger convolutional output before the first capsule operation to generate activations; and it has a set of trainable parameters in the `EM` routing. Another impact to consider is the routing co-efficient matrix of size $n_1 \times n_2$, the computation cost for this part is lightweight though and yet it takes longer

Table 1. Comparison of vanilla CNN, CapNet [10,11] and EncapNet. All models have a depth of six layers and are compared via (i) the number of model parameters (Mb), (ii) memory consumption (MB, at a given batch size), (iii) runtime (second per batch size) and (iv) performance (error rate %). 8 and 4 is the largest batch size that can fit in memory[2] for `dynamic` and `EM` routing. Metric (ii) and (iii) are measured on CIFAR-10.

method	param #	mem. size	runtime	CIFAR-10	MNIST
(a) CapNet, `dynamic`	151.24	3,961 (8)	0.444	14.28	0.37
(b) CapNet, `EM`	152.44	10,078 (4)	0.957	**12.66**	**0.31**
(c) vanilla CNN, same shape	24.44	1,652 (128)	0.026	14.43	0.38
(d) vanilla CNN, similar param	146.88	2,420 (128)	0.146	12.09	0.33
(e) EncapNet, `master`	25.76	1,433 (128)	0.039	13.87	0.31
(f) EncapNet, `master/aide`	60.68	1,755 (128)	0.061	**11.93**	**0.25**

runtime than traditional CNNs due to the routing iteration times R to update c, especially for `EM` method that involves two update alternations.

Inspired by the routing-by-agreement scheme to aggregate feature patterns in the network and bearing in mind that the current solution has a large computation complexity, we resort to some alternative that both inherits the spirit of routing-to-interact among capsules and implements such a process in a fast and accurate fashion. This leads to the proposed scheme stated below.

3 EncapNet: Neural Network Encapsulation

3.1 Approximate routing with master/aide interaction

Recall that higher capsules are generated according to the voting co-efficient c_{ij} across all entities (capsules) in the lower layer:

$$\boldsymbol{s}_j = \sum_{i=1}^{n_1} c_{ij} \cdot \hat{\boldsymbol{v}}_{j|i} = c_{1j}\hat{\boldsymbol{v}}_{j|1} + \cdots + c_{ij}\hat{\boldsymbol{v}}_{j|i} + \cdots + c_{n_1 j}\hat{\boldsymbol{v}}_{j|n_1}, \quad (4)$$

$$= \underbrace{c_{ij}\hat{\boldsymbol{v}}_{j|i}}_{i=j} + \sum_{i \neq j} c_{ij}\hat{\boldsymbol{v}}_{j|i}. \quad (5)$$

Eqn. (4) can be grouped into two parts: one is a main mapping that directly receives knowledge from its lower counterpart i, whose spatial location is the same as j's; another is a side mapping which sums up all the remaining lower capsules, whose spatial location is different from j's. Hence the original unsupervised and short-lived routing process can be approximated in a supervised manner (see Fig. 1(c)):

$$\boldsymbol{s}_j \approx m_1 \hat{\boldsymbol{v}}^{(1)}_{|l(\mathcal{N}_j, k_1)} + m_2 \hat{\boldsymbol{v}}^{(2)}_{|l(\overline{\mathcal{N}}_j, k_2)}, \quad (6)$$

where $\mathcal{N}_j$ is a location set along the channel dimension that *directly* maps lower capsules (there might be more than one) to higher j; $\overline{\mathcal{N}}_j$ is the complimentary set of $\mathcal{N}_j$ that contains the *remaining* locations along the channel; $k_{(*)}$ is the

[4] A single Titan X GPU, which has a 12G memory.

spatial kernel size; altogether $l(\cdot, \cdot)$ indicates the location set of all contributing lower capsules to create a higher capsule. Formally, we define $\hat{\boldsymbol{v}}^{(1)}$ and $\hat{\boldsymbol{v}}^{(2)}$ in Eqn. (6) as the master and aide activation, respectively, with their co-efficients denoted as m_1 and m_2.

The `master` branch looks for the same pattern in two consecutive layers and thus only sees a window from its direct lower capsule. The `aide` branch, on the other hand, serves as a side unit to replenish information from capsules located in other channels. The convolution kernels in both branches use the spatial locality: kernels only attend to a small neighborhood of size $k_1 \times k_1$ and $k_2 \times k_2$ on the input capsule $\boldsymbol{u}$ to generate the intermediate activation $\hat{\boldsymbol{v}}^{(1)}$ and $\hat{\boldsymbol{v}}^{(2)}$. The master and aide activations in these two branches are communicated by their co-efficients in an interactive manner. Co-efficient $m_{(*)}$ is the output of group convolution; the input source is from both $\hat{\boldsymbol{v}}^{(1)}$ and $\hat{\boldsymbol{v}}^{(2)}$, leveraging information encoded in capsules from both the `master` and `aide` branches.

After the interaction as shown in Fig. 1(c), we append the batch normalization [12], rectified non-linearity unit [13] and `squash` operations at the end. These connectivities are not shown in the figure for brevity. To this end, we have encapsulated one layer of the neural network with each neuron being replaced by a capsule, where interaction among them is achieved by the `master/aide` scheme, and denote the whole pipeline as the `capConv` layer. An encapsulated module is illustrated in Fig. 3(a), where several `capConv` layers are cascaded with a skip connection. There are two types of `capConv`. Type I is to increase the dimension of capsules across modules and merge spatially-distributed capsules. The kernel size in the `master` branch is set to be 3 in this type. Type II is to increase the depth of the module for a length of N; the dimension of capsule is unchanged; nor does the number of spatial capsules. The kernel size for the `master` branch in this type is set to be 1. The `capFC` block is a per-capsule-dimension operation of the fully-connected layer as in standard neural network. Table 2 gives an example of the proposed network, called EncapNet.

Comparison to CapNet. Compared to the heavy computation of generating a huge number of mappings for each higher capsule in CapNet, our design only requires two mappings in the `master` and `aide` branch. The computational complexity is reduced by a large margin: the kernel size in the transform matrix in the first step is $\frac{n_2}{2}$ times fewer and the routing scheme in the second step is $\frac{S^4}{d_2}$ times fewer (S being the spatial size of feature map). Take the previous setting in Table 1 for instance, our design leads to 1024 and 256 times fewer parameters than the original 8,388,608 and 4,194,304 parameters in these two steps. To this end, we replace the unsupervised, iterative routing process [10,11] with a supervised, one-pass `master/aide` scheme. Compared with (a-b) in Table 1, our proposed method (e-f) has fewer parameters, less runtime, and better performance. It is also observed that the side information from the `aide` branch is a necessity to replenish the `master` branch, with baseline error 13.87% decreasing to 11.93% on CIFAR-10, *c.f.* (e) vs. (f) in Table 1.

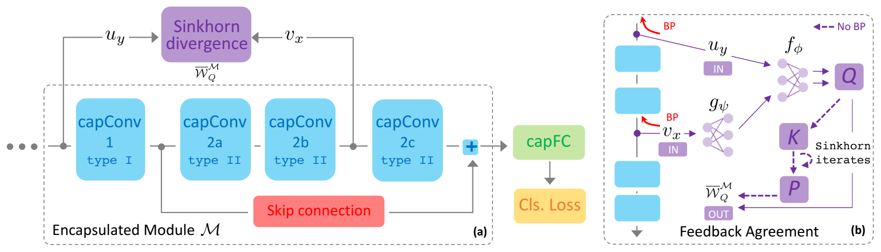

Fig. 3. (a) Connections inside one module of EncapNet, where several `capConv` layers (type I and II) are cascaded with skip connection and regularized by the Sinkhorn divergence. This is one type of design and in Section 5 we report other variants. (b) Pipeline and gradient flow in the Sinkhorn divergence.

3.2 Network regularization by feedback agreement

Motivated by the agreement routing where higher capsules should be activated if there is a good 'agreement' with lower counterparts, we include a loss that requires the higher layer to be able to recover the lower layer. The influence of such a constraint (loss) is used during training and removed during inference.

To put the intuition aforementioned in math notation, let $v_x = \{\boldsymbol{v}_j\}_{j=1}^{n_2}$ and $u_y = \{\boldsymbol{u}_i\}_{i=1}^{n_1}$ be a sample in space $\mathcal{Z}$ and $\mathcal{U}$, respectively, where x, y are sample indices. Consider a set of observations, *e.g.* capsules at lower layer, $\mathcal{S}_1 = (u_1, \dots, u_y, \dots, u_{\mathcal{B}_1}) \in \mathcal{U}^{\mathcal{B}_1}$, we design a loss which enforces samples v on space $\mathcal{Z}$ as input (*e.g.*, capsules at higher layer) can be mapped to u' on space $\mathcal{U}$ through a differentiable function $g_\psi : \mathcal{Z} \to \mathcal{U}$, *i.e.*, $u' = g_\psi(v)$. The data distribution, denoted as $\mathbb{P}_\psi$, for the generated set of samples $\mathcal{S}_2 = (u'_1, \dots, u'_x, \dots, u'_{\mathcal{B}_2}) \in \mathcal{U}^{\mathcal{B}_2}$ should be as much *close* as the distribution $\mathbb{P}_r$ for $\mathcal{S}_1$. In summary, our goal is to find $\psi*$ that minimizes a certain loss or distance between two distributions $\mathbb{P}_\psi, \mathbb{P}_r \in \text{Prob}(\mathcal{U})$[5]: $\arg\min_{\psi*} \mathcal{L}(\mathbb{P}_\psi, \mathbb{P}_r)$.

In this paper, we opt for an optimal transport (OT) metric to measure the distance. The OT metric between two joint probability distributions supported on two metric spaces $(\mathcal{U}, \mathcal{U})$ is defined as the solution of the linear program [16]:

$$\mathcal{W}_Q(\mathbb{P}_\psi, \mathbb{P}_r) = \inf_{\gamma \in \Gamma(\mathbb{P}_\psi, \mathbb{P}_r)} \mathbb{E}\bigg[\int_{\mathcal{U}\times\mathcal{U}} Q(u', u) d\gamma(u', u)\bigg], \tag{7}$$

where γ is a coupling; Γ is the set of couplings that consists of joint distributions over the product space with marginals $(\mathbb{P}_\psi, \mathbb{P}_r)$. Our formulation skips some mathematic notations; details are provided in [15,16]. Intuitively, $\gamma(u', u)$ implies how much "mass" must be transported from u' to u in order to transform the distribution $\mathbb{P}_\psi$ into $\mathbb{P}_r$; Q is the "ground cost" to move a unit mass from u' to u. As is well known, Eqn. (7) becomes the p-Wasserstein distance (or loss,

[5] In some literature, *i.e.*, [14,15], it is called the probability measure and commonly denoted as μ or ν; a coupling is the joint distribution (measure). We use distribution or measure interchangeably in the following context. Prob($\mathcal{U}$) is the set of probability distributions over a metric space $\mathcal{U}$.

divergence) between probability measures when $\mathcal{U}$ is equipped with a distance $\mathcal{D}_{\mathcal{U}}$ and $Q = \mathcal{D}_{\mathcal{U}}(u', u)^p$, for some exponent p.

Note that the expectation $\mathbb{E}(\cdot)$ in Eqn. (7) is used for mini-batches of size $(\mathcal{B}_1, \mathcal{B}_2)$. In our case, $\mathcal{B}_1$ and $\mathcal{B}_2$ are equal to the training batch size. Since both input measures are discrete for the indices x and y (capsules in the network), the coupling γ can be treated as a non-negative matrix P, namely $\gamma = \sum_{x,y} P_{x,y}\delta(v_x, u_y) \in \text{Prob}(\mathcal{Z} \times \mathcal{U})$, where δ represents the Dirac unit mass distribution at point $(v, u) \in (\mathcal{Z} \times \mathcal{U})$. Rephrasing the continuous case of Eqn. (7) into a discrete version, we have the desired OT loss:

$$\mathcal{W}_Q(\mathbb{P}_\psi, \mathbb{P}_r) \xleftarrow{\text{discrete}} \min_{P \in \mathbb{R}_+^{\mathcal{B}_2 \times \mathcal{B}_1}} \langle Q, P \rangle, \tag{8}$$

where P satisfies $P^\intercal \mathbb{1}_{\mathcal{B}_2} = \mathbb{1}_{\mathcal{B}_1}, P\mathbb{1}_{\mathcal{B}_1} = \mathbb{1}_{\mathcal{B}_2}$. $\langle \cdot, \cdot \rangle$ indicates the Frobenius dot-product for two matrices and $\mathbb{1}_m := (1/m, \dots, 1/m) \in \mathbb{R}_+^m$. Now the problem boils down to computing P given some ground cost Q. We adopt the Sinkhorn algorithm [17] in an iterative manner, which is promised to have a differentiable loss function [16]. Starting with $b^{(0)} = \mathbb{1}_{\mathcal{B}_2}, l \leftarrow 0$, `Sinkhorn iterates` read :

$$a^{(l+1)} := \frac{\mathbb{1}_{\mathcal{B}_1}}{K^\intercal b^{(l)}}, \quad b^{(l+1)} := \frac{\mathbb{1}_{\mathcal{B}_2}}{K a^{(l)}}, \tag{9}$$

where the Gibbs kernel $K_{x,y}$ is defined as $\exp(-Q_{x,y}/\varepsilon)$; ε is a control factor. For a given budget of L iterations, we have:

$$P := P^{(L)} = \text{diag}(b^{(L)}) \cdot K \cdot \text{diag}(a^{(L)}), \tag{10}$$

which serves as a proxy for the OT coupling. Equipped with the computation of P and having some form of cost Q in hand, we can minimize the optimal transport divergence along with other loss in the network.

In practice, we introduce a bias fix to the original OT distance in Eqn. (8), namely the Sinkhorn divergence [15]. Given two sets of samples v_x, u_y and accordingly distributions $\mathbb{P}_\psi, \mathbb{P}_r$, the revision is defined as:

$$\overline{\mathcal{W}}_Q^{\mathcal{M}}(\mathbb{P}_\psi, \mathbb{P}_r) = 2\mathcal{W}_Q(\mathbb{P}_\psi, \mathbb{P}_r) - \mathcal{W}_Q(\mathbb{P}_\psi, \mathbb{P}_\psi) - \mathcal{W}_Q(\mathbb{P}_r, \mathbb{P}_r), \tag{11}$$

where $\mathcal{M}$ is the module index. By tuning ε in K from 0 to ∞, the Sinkhorn divergence has the property of taking the best of both OT (non-flat geometry) and MMD [18] (high-dimensional rigidity) loss, which we find in experiments improves performance.

The overall workflow to calculate a Sinkhorn divergence[6] is depicted in Fig. 3(b). Note that our ultimate goal of applying OT loss is to make feature learning in the *mainstream* (blue blocks) better aligned across capsules in the network. It is added during training and abominated for inference. Therefore the design for Sinkhorn divergence has two principles: light-weighted and capsule-minded.

[6] The term Sinkhorn used in this paper is two-folds: one is to indicate the computation of P via a `Sinkhorn iterates`; another is to imply the revised OT divergence.

Sub-networks g_ψ and f_ϕ should increase as minimal parameters to the model as possible; the generator should be encapsulated to match the data structure. Note that the Sinkhorn divergence is optimized to minimize loss w.r.t. both ϕ, ψ, instead of the practice in [15,19,14] via an adversarial manner.

Discussions. (i) There are alternatives besides the OT metric for $\mathcal{L}(\mathbb{P}_\psi, \mathbb{P}_r)$, *e.g.*, the Kullback-Leibler (KL) divergence, which is defined as $\sum_y \log \frac{\mathrm{d}\mathbb{P}_\psi}{\mathrm{d}u'} u_y$ or Jenson-Shannon (JS) divergence. In [14], it is observed that these distances are not sensible when learning distributions supported by low dimensional manifolds on $\mathcal{Z}$. Often the model manifold and the "true" distribution's support often have a non-negligible intersection, implying that KL and JS are non-existent or infinite in some cases. In comparison, the optimal transport loss is continuous and differentiable on ψ under mild assumptions nonetheless. (ii) Our design of feedback agreement unit is not limited to the capsule framework. Its effectiveness on vanilla CNNs is also verified by experimental results in Section 5.1.

Design choices in OT divergence. We use a deconvolutional version of the `capConv` block as the mapping function g_ψ for reconstructing lower layer neurons from higher layer neurons. Before feeding into the cost function Q, samples from two distributions are passed into a feature extractor f_ϕ. The extractor is modeled by a vanilla neural network and can be regarded as a dimensionality reduction of $\mathcal{U}$ onto a lower-dimension space. There are many options to design the cost function Q, such as cosine distance or l_2 norm. Moreover, it is found in experiments that if the gradient flow in the `Sinkhorn iterates` process is ignored as does in [19], the result gets slightly better. Remind that $Q_{x,y} = \mathcal{D}\big(f_\phi({u'}_x), f_\phi(u_y)\big)$ is dependent on ϕ, ψ (so does P, K, a, b); hence the whole OT unit can be trained in the standard optimizers (such as Adam [20]).

Overall loss function. The final loss of EncapNet is a weighted combination from both the Sinkhorn divergence across modules and the marginal loss [10] for capsule in the classification task: $\mathcal{L}_{\texttt{margin}}(t, \boldsymbol{v}) + \lambda \sum_{\mathcal{M}} \overline{\mathcal{W}}_Q^{\mathcal{M}}$, where $t, \boldsymbol{v}$ is the ground truth and class capsule outputs of the `capFC` layer, respectively; λ is a hyper-parameter to negotiate between these two losses (set to be 10).

4 Related Work

Capsule network. Wang *et al.* [21] formulated the routing process as an optimization problem that minimizes the clustering-like loss and a KL regularization term. They proposed a more general way to regularize the objective function, which shares similar spirit as the agglomerative fuzzy k-means algorithm [22]. Shahroudnejad *et al.* [23] explained the capsule network inherently constructs a relevance path, by way of dynamic routing in an unsupervised way, to eliminate the need for a backward process. When a group of capsules agree for a parent one, they construct a part-whole relationship which can be considered as a relevance path. A variant capsule network [24] is proposed where capsule's activation is obtained based on the eigenvalue of a decomposed voting matrix. Such a spectral perspective witnesses a faster convergence and a better result than the EM routing [11] on a learning-to-diagnose problem.

Attention vs. routing. In [25], Mnih *et al.* proposed a recurrent module to extract information by adaptively selecting a sequence of regions and to only attend the selected locations. DasNet [26] allows the network to iteratively focus attention on convolution filters via feedback connections from higher layer to lower ones. The network generates an observation vector, which is used by a deterministic policy to choose an action, and accordingly changes the weights of feature maps for better classifying objects. Vaswani *et al.* [27] formulated a multi-head attention for machine translation task where attention coefficients are calculated and parameterized by a compatibility function. Attention models aforementioned tries to learn the attended weights from lower neurons to higher ones. The lower activations are weighted by the learned parameters in attention module to generate higher activations. However, the agreement routing scheme [10,11] is a top-down solution: higher capsules should be activated if and only if the most similar lower counterparts have a large response. The routing coefficients is obtained by recursively looking back at lower capsules and updated based on the resemblance. Our approximate routing can be deemed as a bottom-up approach which shares similar spirit as attention models.

5 Experiments

The experiments are conducted on CIFAR-10/100 [28], SVHN [29] and a large-scale dataset called "`h`-ImageNet". We construct the fourth one as a subset of the ILSVRC 2012 classification database [30]. It consists of 200 hard classes whose top-1 accuracy, based on the prediction output of the ResNet-18 [5] model is lower than other classes. The ResNet-18 baseline model on `h`-ImageNet has a 41.83% top-1 accuracy. The dataset has a collection of 255725, 17101 images for training and validation, compared with CIFAR's 50000 for training and 10000 for test. We manually crop the object with some padding for each image (if the bounding box is not provided) since the original image has too much background and might be too large (over 1500 pixels); after the pre-processing, each image size is around 50 to 500, compared with CIFAR's 32 input. "`h`-ImageNet" is proposed for fast verifying ML algorithms on a large-scale dataset which shares similar distribution as ImageNet.

Implementation details. The general settings are the same across datasets if not specified afterwards. Initial learning rate is set to 0.0001 and reduced by 90% with a schedule $[200, 300, 400]$ in epoch unit. Maximum epoch is 600. Adam [20] is used with momentum 0.9 and weight decay 5×10^{-4}. Batch size is 128.

5.1 Ablative analysis

In this subsection we analyze the connectivity design in the encapsulated module and the many choices in the OT divergence unit. The depth of EncapNet and ResNet are the same 18 layers ($N = 3, n = 2$) for fair comparison. Their structures are depicted in Table 2. Remind that the comparison of `capConv` block with CapNet is reported in Table 1 and analyzed in Section 3.1.

Table 2. Network architecture of EncapNet and ResNet. The compared ResNet variant has the same input and output shape as EncapNet. '$x \to y$' indicates channel dimension from input to output. `capConv`(k, s, p) means the `master` capsule has a convolution of kernel size k, stride s and padding p. Similarly for the standard convolution `conv`() and residual block `res`(). The depth of the EncapNet and ResNet is $2 + \sum_i (N_i + 1)$ and $2 + \sum_i 2n_i$, respectively. Connection of OT divergence is omitted for brevity.

module		output size	cap dim.	EncapNet_v1	ResNet
$\mathcal{M}_0$		32×32	-	$3 \to 32$, `conv`$(3,1,1)$	$3 \to 32$, `conv`$(3,1,1)$
$\mathcal{M}_1$	I	32×32	$1 \to 2$	$32 \to 32$, `capConv`$(3,1,1)$	$32 \to 64$, `res`$(3,1,1)$
	II		2	$[32 \to 32$, `capConv`$(1,1,0)] \times N_1$	$[64 \to 64$, `res`$(3,1,1)] \times (n_1 - 1)$
$\mathcal{M}_2$	I	16×16	$2 \to 4$	$32 \to 32$, `capConv`$(3,2,1)$	$64 \to 128$, `res`$(3,2,1)$
	II		4	$[32 \to 32$, `capConv`$(1,1,0)] \times N_2$	$[128 \to 128$, `res`$(3,1,1)] \times (n_2 - 1)$
$\mathcal{M}_3$	I	8×8	$4 \to 8$	$32 \to 32$, `capConv`$(3,2,1)$	$128 \to 256$, `res`$(3,2,1)$
	II		8	$[32 \to 32$, `capConv`$(1,1,0)] \times N_3$	$[256 \to 256$, `res`$(3,1,1)] \times (n_3 - 1)$
$\mathcal{M}_4$	I	4×4	$8 \to 16$	$32 \to 32$, `capConv`$(3,2,1)$	$256 \to 512$, `res`$(3,2,1)$
	II		16	$[32 \to 32$, `capConv`$(1,1,0)] \times N_4$	$[512 \to 512$, `res`$(3,1,1)] \times (n_4 - 1)$
$\mathcal{M}_5$		10/100/200	16	`capFC`	`avgPool`, `FC`

Design in `capConv` block. Table 3 (1-4) reports the different incoming sources of the co-efficients m in the `master` and `aide` branches. Without using `aide`, case (1) serves as baseline where higher capsules are only generated from the master activation. Note that the 9.83% result is already superior than all cases in Table 1, due to the increase of network depth. Result show that obtaining m_x from the activation $\hat{\boldsymbol{v}}^{(x)}$ in its own branch is better than obtaining from the other activations, *c.f.*, cases (2) and (3). When the incoming source of co-efficient is from both branches, denoted as "`maser/aide_v3`" in (4), the pattern information from lower capsules is fully interacted by both `master` and `aide` branches; hence we achieve the best result of 7.41% when compared with cases (2) and (3). Table 3 (5-7) reports the result of adding skip connection based on case (4). It is observed that the skip connection used in both types of the `capConv` block make the network converge faster and get better performance (5.82%). Our final candidate model employs an additional OT unit with two Sinkhorn losses imposed on each module. One is the connectivity as shown in Fig. 3(a) where v_x is half the size of v_y; another connectivity is the same as the skip connection path shown in the figure, where v_x shares the same size with v_y; the "deconvolutional" generator in this connectivity has a stride of 1. It performances better (4.55%) than using one OT divergence alone (4.58%).

Network regularization design. Fig. 4 illustrates the training loss curve with and without OT (Sinkhorn) divergence. It is found that the performance gain is more evident for EncapNet than ResNet (21% vs 4% increase on two networks, respectively). Moreover, we testify the KL divergence option as a distance measurement to substitute the Sinkhorn divergence, shown as case (b) in Table 3. The error rate decreases for both model, suggesting that the idea of imposing regularization on the network training is effective; such an add-on is to keep feature patterns better aligned across layers. The subtlety is that the gain clearly differs when we replace Sinkhorn with KL in EncapNet while these two options barely matter in ResNet.

Table 3. Ablative analysis on (**left**) the design in the `capConv` layer and (**right**) network regularization design. EncapNet and ResNet have the same 18 layers. "two OTs" indicates each module has two OT divergences coming from different sources. Experiments in series (d-*) are based on case (c) and conducted by removing or substituting each component in the OT unit while keeping the rest factors fixed.

`capConv` **Design**	error (%)
(1) `master` (baseline)	9.83
(2) `maser/aide_v1`	8.05
(3) `maser/aide_v2`	9.11
(4) `maser/aide_v3`	7.41
(5) skip_on_Type_I	6.81
(6) skip_on_Type_II	6.75
(7) skip_both	5.82
(8) two OTs	**4.55**

Network Regularization	EncapNet	ResNet
(a) `capConv` block (baseline)	5.82	8.03[7]
(b) KL_loss	5.31	7.72
(c) OT_loss	**4.58**	7.67
(d1) remove bias fix	4.71	-
(d2) do BP in P_L	4.77	-
(d3) no extractor f_ψ	5.79	-
(d4) use vanilla g_ϕ	5.01	-
(d5) use l_2 in Q	4.90	-

Furthermore, we conduct a series of experiments (d-*) to prove the rationale of the Sinkhorn divergence design in Section 3.2. Without the bias fix, the result is inferior since it does not leverage both OT and MMD divergences (case d1); if we back-propagate the gradient in the P_L path, the error rate slightly increases; the role of feature extractor f_ψ is to down-sample both inputs to the same shape on a lower dimension for the subsequent pipeline to process. If we remove this functionality and directly compare the raw inputs (u, u') using cosine distance, the error increases by a large margin to 5.79%, compared with baseline 5.82%; if we adopt l_2 norm to measure the distance between raw inputs, loss will not converge (not shown in Table 3). This verifies the urgent necessity of having a feature extractor; if the generator recovering u' from v employs a standard CNN, the performance is inferior (5.01%) than the capsule version of the generator since data flows in form of capsules in the network; finally if we adopt l_2 norm to calculate P after the feature extractor, the performance degrades as well.

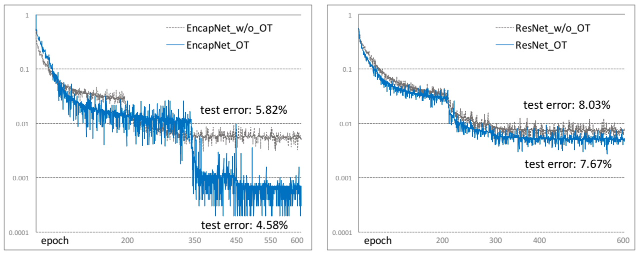

Fig. 4. Training losses with embedded optimal transport divergence for EncapNet and ResNet (*_OT). One OT unit is adopted as depicted in Fig. 3(a) for each module in the network.

5.2 Comparison to state-of-the-arts

As shown in Table 4, (a) on CIFAR-10/100 and SVHN, we achieve a better performance of 3.10%, 24.01% and 1.52 % compared to previous entires. The

[7] ResNet-20 reported in [5] has a 8.75% error rate; some online third party implementation (link anonymised for submission) obtains 6.98%; we run the 18-layer model in PyTorch with settings stated in the context.

multi-crop test is a key factor to further enhance the result, which is widely used by other methods as well. (b) on `h`-ImageNet, v1 is the 18-layer structure and has a reasonable top-1 accuracy of 51.77%. We further increase the depth of EncapNet (known as v2) by stacking more `capConv` blocks, making a depth of 101 to compare with the ResNet-101 model. To ease runtime complexity due to the `master/aide` intertwined communication, we replace some blocks in the shallow layers with `master` alone. v3 has a larger input size. Moreover, we have the ultimate version of EncapNet with data augmentation (v3^{++}) and obtain an error rate of 40.05%, compared with the runner-up WRN [31] 42.51%. Training on `h`-ImageNet roughly takes 2.9 days with 8 GPUs and batch size 256. (c) we have some preliminary results on the ILSVRC-CLS (`complete`-ImageNet) dataset, which are reported in terms of the top-5 error in Table 4.

Table 4. Classification errors (%) compared to state-of-the-arts. For state-of-the-arts, we show the best results available in their papers. $^{+}$ means mild augmentation while $^{++}$ stands for strong augmentation. For `h`-ImageNet, we train models and report results of other networks based on the same setting as EncapNet_v3^{++}.

method	CIFAR-10	CIFAR-100	SVHN
EncapNet	4.55	26.77	2.01
EncapNet^{+}	3.13	**24.01** (24.85 ±0.11)	1.64
EncapNet^{++}	**3.10** (3.56 ±0.12)	24.18	**1.52** (1.87 ±0.11)
GoodInit [32]	5.84	27.66	-
BayesNet [33]	6.37	27.40	-
ResNet [5]	6.43	-	-
ELU [34]	6.55	24.28	-
Batch NIN [35]	6.75	28.86	1.81
Rec-CNN [36]	7.09	31.75	1.77
Piecewise [37]	7.51	30.83	-
DSN [38]	8.22	34.57	1.92
NIN [39]	8.80	35.68	2.35
dasNet [26]	9.22	33.78	-
Maxout [40]	9.35	38.57	2.47
AlexNet [2]	11.00	-	-

`h`-ImageNet	
EncapNet_v1	48.23
EncapNet_v2	43.15
EncapNet_v3	42.76
EncapNet_v3^{+}	40.18
EncapNet_v3^{++}	**40.05**
WRN [31]	42.51
ResNet-101 [5]	44.13
VGG [3]	55.76
GoogleNet [4]	60.18
`complete`-ImageNet(top-5)	
EncapNet-18	7.51
GoogleNet [4]	7.89
VGG [3]	8.43
ResNet-101 [5]	6.21

6 Conclusions

In this paper, we analyze the role of routing-by-agreement to aggregate feature clusters in the capsule network. To lighten the computational load in the original framework, we devise an approximate routing scheme with master-aide interaction. The proposed alternative is light-weight, supervised and one-time pass during training. The twisted interaction ensures that the approximation can make best out of lower capsules to activate higher capsules. Motivated by the routing process to make capsules better aligned across layers, we send back higher capsules as feedback signal to better supervise the learning across capsules. Such a network regularization is achieved by minimizing the distance of two distributions using optimal transport divergence during training. This regularization is also found to be effective for vanilla CNNs.

Acknowledgment. We would like to thank Jonathan Hui for the wonderful blog on capsule research, Gabriel Peyré and Yu Liu for helpful discussions in early stage. H. Li and X. Guo are funded by Hong Kong Ph.D. Fellowship Scheme.

References

1. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. Volume 86. (1998) 2278–2324
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. (2012) 1106–1114
3. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR. (2015)
4. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR. (2015)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
6. Li, H., Liu, Y., Ouyang, W., Wang, X.: Zoom out-and-in network with map attention decision for region proposal and object detection. In: International Journal of Computer Vision (IJCV). (2018)
7. Li, H., Liu, Y., Zhang, X., An, Z., Wang, J., Chen, Y., Tong, J.: Do we really need more training data for object localization. In: IEEE International Conference on Image Processing. (2017)
8. Chi, Z., Li, H., Huchuan, Yang, M.H.: Dual deep network for visual tracking. IEEE Trans. on Image Processing (2017)
9. Hui, J.: Understanding Matrix capsules with EM Routing. https://jhui.github.io/2017/11/14/Matrix-Capsules-with-EM-routing-Capsule-Network (2017) Accessed: 2018-03-10.
10. Sabour, S., Frosst, N., Hinton, G.: Dynamic routing between capsules. In: NIPS. (2017)
11. Hinton, G.E., Sabour, S., Frosst, N.: Matrix capsules with EM routing. In: ICLR. (2018)
12. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML. (2015)
13. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML. (2010) 807–814
14. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. arXiv preprint: 1701.07875 (2017)
15. Genevay, A., Peyré, G., Cuturi, M.: Learning generative models with sinkhorn divergences. arXiv preprint: 1706.00292 (2017)
16. Cuturi, M.: Sinkhorn distances: Lightspeed computation of optimal transport. NIPS (2013)
17. Sinkhorn, R.: A relationship between arbitrary positive matrices and doubly stochastic matrices. Ann. Math. Statist. (2) (06) 876–879
18. Gretton, A., Borgwardt, K., Rasch, M.J., Scholkopf, B., Smola, A.J.: A kernel method for the two-sample problem. NIPS (2007)
19. Salimans, T., Zhang, H., Radford, A., Metaxas, D.: Improving GANs using optimal transport. In: International Conference on Learning Representations. (2018)
20. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR. (2015)
21. Wang, D., Liu, Q.: An optimization view on dynamic routing between capsules. In: Submit to ICLR workshop. (2018)
22. Li, M.J., Ng, M.K., ming Cheung, Y., Huang, J.Z.: Agglomerative fuzzy k-means clustering algorithm with selection of number of clusters. IEEE Transactions on Knowledge and Data Engineering **20** (2008) 1519–1534

23. Shahroudnejad, A., Mohammadi, A., Plataniotis, K.N.: Improved explainability of capsule networks: Relevance path by agreement. In: arXiv preprint:1802.10204. (2018)
24. Bahadori, M.T.: Spectral capsule networks. In: ICLR workshop. (2018)
25. Mnih, V., Heess, N., Graves, A., Kavukcuoglu, K.: Recurrent models of visual attention. In: NIPS. (2014)
26. Stollenga, M.F., Masci, J., Gomez, F., Schmidhuber, J.: Deep networks with internal selective attention through feedback connections. In: NIPS. (2014) 3545–3553
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint: 1706.03762 (2017)
28. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. In: Technical Report. (2009)
29. 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. (2011)
30. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3) (2015) 211–252
31. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: BMVC. (2016)
32. Mishkin, D., Matas, J.: All you need is a good init. arXiv preprint:1511.06422 (2015)
33. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable bayesian optimization using deep neural networks. In: ICML. (2015)
34. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units. arXiv preprint: 1511.07289 (2015)
35. Chang, J.R., Chen, Y.S.: Batch-normalized maxout network in network. In: arXiv preprint:1511.02583. (2015)
36. Liang, M., Hu, X.: Recurrent convolutional neural network for object recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (June 2015)
37. Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P.: Learning activation functions to improve deep neural networks. In: ICLR workshop. (2015)
38. Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. arXiv preprint: 1409.5185 (2014)
39. Lin, M., Chen, Q., Yan, S.: Network in network. In: ICLR. (2014)
40. Goodfellow, I.J., Warde-farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: ICML. (2013)
41. Frogner, C., Zhang, C., Mobahi, H., Araya-Polo, M., Poggio, T.: Learning with a wasserstein loss. NIPS (2015)

A More Details and Results in CapNet

A.i Configurations in Section 2.2

All the experiments in Table 1 and Fig. 2 employ a shallow network of six layers. The first four layers across vanilla CNNs, CapNet and EncapNet use the same recipe: one standard convolution plus a BN-ReLU follow-up. The stride and padding in convolution is 3 and 1, respectively; these four modules are for increasing the channel number and down-sampling the feature map. After these modules, the activation map is of size $256 \times 8 \times 8$.

For CapNet (case (a) and (b)), both the fifth and sixth layer are capsule-minded and implemented by group convolution, as shown in Fig. 1(b). The transform matrix (kernel in convolution) in the fifth layer is super huge (of size $8, 1, 1, 32 \times 16 \times 2048$) since the mapping step involves transforming capsule i to each and every j in the subsequent layer. After routing (dynamic or EM), the output has a shape of 2048×16 and is rearranged into the blob shape in a spatial-channel distributed manner ($512 \times 8 \times 8$) for the next layer. The budget of routing iteration is set to $R = 3$.

There are two subtleties when we implement the routing scheme. For `dynamic` routing, we find the loss does not converge if we conduct the softmax operation of co-efficients b along the j dimension (as suggested in [10]); *instead we operate softmax along the i dimension.* For `EM` routing, it is observed that *adding a normalization operation* before data is fed into the sigmoid function to generate activation a_j is crucial to make the network stable and faster towards convergence, since the std acquired after the M-step is small. Having them through a log function results in most values being into the range, say over 100; this will further generate all 1's after sigmoid. If we add normalization to set the range of input data before sigmoid around zero, the output a_j will be diversified in $[0, 1]$.

For the compared vanilla CNNs (case (c) and (d)), the fifth module is a `conv`-BN-ReLU cascade with average pooling to make spatial size of feature maps to be 1×1; the last module is a fully connected layer. The network for case (d), "similar param", is achieved by increasing the channel number such that the model size amounts to a similar level as case (a) and (b).

For the compared EncapNet (case (e) and (f)), the fifth layer is a `capConv` layer with approximate routing and master/aide interaction, which is proposed in the main paper. The sixth layer is a `capFC` layer with outputs being the same as CapNet's (10×16).

A.ii Ablative study on CapNet

In preliminary experiments, we conduct some ablative analysis on the routing scheme and different loss choices in CapNet, which is shown in Table 5.

Table 5. The agreement routing and loss choices in CapNet [10]. The chosen setting in a certain aspect is marked with gray background. We use dynamic routing and the network structure employs the same 6-layer recipe as stated in Section A.i.

routing iter.	CIFAR-10
1	14.79
2	14.41
3	14.28
4	14.25

routing co-efficient	CIFAR-10
random	14.35
zero	14.28
learnable	14.11

loss/optimizer	RMSProp	SGD	Adam
spread, fix_m	15.61	15.38	14.97
spread	15.02	14.87	14.50
margin	14.30	15.18	14.28
CE_loss	14.76	fail	15.31

First, the number of routing iteration. We find the error would get lower if we conduct the routing algorithm via coordinate descent ($R >= 2$) than a simple averaging across all lower capsules ($R = 1$, implying there is no routing). As the number of iteration goes up, the performance gets slightly better; since it would cost more runtime, we fix R to be 3 considering both efficiency and accuracy. Second, the initialization of the routing co-efficients. The zero initialization is suggested in [10]; we also investigate other options. The performance difference between random and learnable initialization of the co-efficients does not vary much. Hence we adopt the original zero manner of initialization. At last, we investigate different loss choices under various optimizers. In general, RMSProp and Adam are better in sense of training capsule networks than does SGD. The popular cross entropy loss is inferior than the marginal [10] and spread [11] loss.

B More Details in EncapNet

The capsule generator is composed of a `convTranspose2d` operation with grouping, followed by a BN-ReLU unit; the value of grouping is the dimension of input capsules. The kernel size in the convolution is 3; the stride could be 2 or 1, depending on whether the size of output activations matches the size of input. The generator appends a squash operation at the end. The feature extractor (also known as "critic") consists of two consecutive convolutions with each followed by a BN-ReLU sequel. The kernel size, padding and stride of these convolutions are (3,1,2); the number of output channel in the first convolution is one forth of the input's while the second convolution only has one output channel.

The control factor ϵ to compute the kernel K is set to 0.1. As suggested in [41], a stronger regularization (larger ε) leads to stable result and faster convergence; hence L can be set small. The length of `Sinkhorn iterates` is set to 10.

C Comparison to State-of-the-arts

The mild data augmentation includes the multi-crop test and random cropping; and the strong augmentation additionally includes random horizontal flipping, color jittering and longer training. For random cropping, a patch of size 32 is randomly cropped out of a resized image of size 34 on CIFAR and SVHN; a patch of size 128 (224) is randomly cropped out of a resized image of size 156 (256) on `h`-ImageNet across all versions. The number in parentheses indicates the setting in `v3` version. For color jittering, the input color channels are randomly disturbed in brightness, contrast, saturation and hue with $\sigma = 0.2$.

The `h`-ImageNet database[8] is proposed to train an ImageNet-like dataset in a manageable time. EncapNet_v3 takes around 2.5 days on 4 GPUs, *c.f.* the full ImageNet database trainig [5] being one week on 8 GPUs. However, the correlation between `h`-ImageNet and the original database is not investigated by time of submission; whether the effectiveness on `h`-ImageNet *still* holds on ILSVRC for most algorithms will be left as future work.

[8] For detailed statistics, please refer to the GitHub repo mentioned earlier.