Capsule Networks With Capsule-Type Normalization Routing

***Abstract*—Capsule Networks (CapsNets) have been well known for its part-whole relational property, whilst with heavy computation of the capsule routing. The classic Expectation-Maximization (EM) capsule routing first compute the vote matrix by multiplying part-whole pose matrix and learnable weight matrix, and secondly fed vote matrices and activations into EM algorithm for clustering. The heavy computation comes from a large-scale computations of vote matrices and large-scale data EM cluster. To address the challenge of lightweight design of CapsNets, different from the previous EM capsule routing that obeys the first-vote-then-cluster rule, we implement a novel first-cluter-then-vote mechanism. To this end, in this paper, we develop a capsule-type normalization routing algorithm as illustrated in Fig. 1 (b). Specifically, we first normalize the part-level capsules along the type dimension with the aim of transforming all types of capsules into a uniform distribution. Secondly, all part-level capsules and their mixed capsules are voted to the whole-level capsules via a multiplication with a single transformable matrix. In such way, the cluster computation and matrix multiplication computation both get reduced with a large margin. Our capsule-type normalization routing builds a deep CapsNets, which are proved to be promising on multiple datasets, including MNIST, SVHN, SmallNORB, CIFAR-10/100. Notably, our CapsNets can be implemented on the large-scale ImageNet-1K dataset and beats ResNets, which is quite difficult for the previous CapsNets versions.**

***Index Terms*—Capsule Network, Capsule-type normalization routing, Classification**

I. Introduction

Convolutional Neural Networks (CNNs) have been the most popular in the deep learning era, although they still face certain challenges such as information loss in pooling layers, low robustness, and poor spatial feature correlation. More concerning is that natural and non-adversarial pose changes of familiar objects in the real world can easily deceive deep networks [**?**], [**?**], [**?**].

To address this issue, an alternative neural network called Capsule Networks (CapsNets) [**?**], [**?**] was introduced. Different from the activation scalar in CNNs, each capsule in CapsNets learns to recognize a visual entity implicitly defined within a limited range of observation conditions and deformations. By dynamically adjusting the connections between capsule layers, the routing algorithm captures part-whole spatial relationships between different-layer capsules, which helps to handle pose variations and partial occlusions. Such property improves classification accuracy and generalization ability of the network. With capsule units and the dynamic routing algorithm, CapsNets address the static connection limitations of CNNs. Moreover, compared to traditional pooling operations of CNNs that may loss the pose equivariance, the routing algorithm can more effectively retain and transmit pose variations, improving performance for complicated scenes.

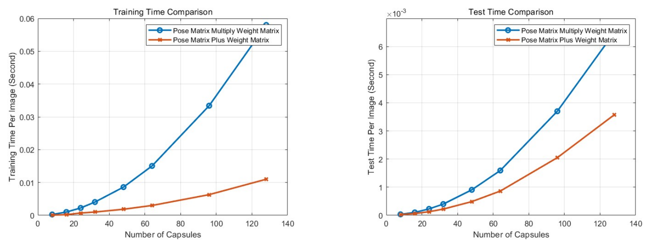

Fig. 1: The left and right figures illustrate the training and testing times for two capsule networks with varying numbers of capsules. The networks use the pose matrix either multiplied by or added to the weight matrix. Experiments on the CIFAR-10 dataset with a batch size of 20 show that during training, the network using matrix multiplication takes five times longer than the one using matrix addition; during testing, it takes twice as long. The computational overhead increases significantly with the number of capsules.

Despite the superiority of CapsNets, their have been hindered due to heavy computation, which makes CapsNets far away from deep layers and further limited in terms of performance on large-scale tasks, e.g., detection, segmentation, and large-scale dataset classification such as ImageNet. The EM routing implements the first-vote-then-cluster strategy, which first computes the vote matrix to project the low-layer capsules to high-layer capsules and then cluster projected data. The heavy computation lies in two folds: i) The vote matrix in CapsNets [**?**] is computed by multiplying pose matrix and a learnable matrix. Large-scale vote matrices lead to heavy computation of multiplications. As shown in Fig. 1, the multiplications for vote matrices significantly worsen the training and testing speed; ii) The EM algorithm implements cluster on ($M \times N \times H \times W \times 17$) data[1], which inevitably generate high cluster computational load due to large-scale data, which will grow exponentially with respect to the increase of M, N, and $H \times W$.

In this paper, to address the aforementioned challenge, we propose a novel and efficient capsule routing algorithm, named Capsule-Type Normalization Routing (CTNR). Different from the previous EM capsule routing [**?**] that implements the first-vote-then-cluster mechanism, which generates large-scale matrix multiplications and cluster data, our CTNR alternatively opt for first-cluster-then-vote mechanism for lightweight routing. as show in Fig2. Specifically, we first activate the low-layer capsule pose matrices by their capsule activation values,

[1] M and N represent the capsule type number of low and high layers, respectively. $H \times W$ is the spatial resolution of capsule features. 17 is the capsule dimension (pose matrix plus activation).

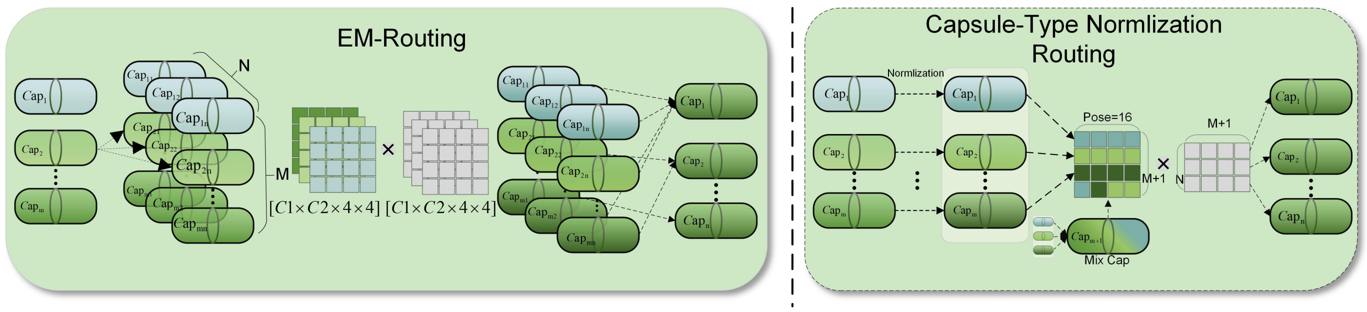
EM-Routing
N
M
$[C1\times C2\times 4\times 4]$ $[C1\times C2\times 4\times 4]$
Capsule-Type Normlization Routing
Normlization
Pose=16
M+1
N
Mix Cap

Fig. 2: The capsule full connectivity adopted by EM routing ensures comprehensive information exchange but imposes a high memory burden. In contrast, our TCNR approach decouples the capsules through the coefficient matrix, which effectively reduces the amount of computation while maintaining the necessary interactions between the capsules, and achieves the double optimization of efficiency and performance.

and carry out clustering on these activated pose matrices using a type-dimensional normalization, which helps to normalize M types of capsules to the uniform distribution. Such cluster mechanism reduce $N\times$ cluster data, greatly speeding up the clustering process. Besides, we mix low-layer activated capsule pose matrices to get their associated type of capsules. Secondly, we vote the uniform-distribution low-layer capsules along with their mixed capsules to the high-layer capsules using a single learnable matrix. Compared with the previous large-scale matrix multiplications, our vote mechanism employs only one transformation matrix, which significantly reduce the consumption for matrix multiplications, parameters, and computing memory burden.

On top of the proposed CTNR strategy, we build a deep CapsNets architecture. Specifically, four blocks, each of which contains one primary capsule layer and two CTNR layers. Such deep design of CapsNets are able to tackle complicated image classification due to their ability of high-level spatial relations exploration. Without any backbone networks, our CapsNets achieve promising performance for the fundamental task of image classification on datasets including MNIST [?], CIFAR-10/100 [?], SVHN [?], SmallNORB [?], compared with the previous CapsNets versions. Besides, our CapsNets achieve 78% accuracy, which surpass the well-known CNNs ResNet-50 [?].

To sum up, the contributions of this paper are described as follows:

(1) We propose a novel and efficient Capsule-Type Normalization Routing strategy, which effectively solves the huge computational complexity and memory consumption problems associated with matrix multiplication computation and data cluster.

(2) Inspired by the lightweight CTNR, we build a deep CapsNets architecture composed by four blocks involving 12 layers like classical CNNs such as VGG and ResNet.

(3) Experiments demonstrate our CapsNets achieve promising performance in public datasets, including MNIST, CIFAR-10/100, SVH, and SmallNORB. Besides, our CapsNets perform well on the large-scale ImageNet-1K, compared with the CNNs like VGG and ResNet, which is a great challenge for the existing CapsNets versions.

This paper is organized as follows. Sec. II reviews the related references to our work. Sec. III describes the details of the proposed Capsule-Type Normalization Routing strategy. Sec. IV Introduces our capsule network architecture. Sec. V carries out abundant experiments and analyses to understand our method. Sec. VI concludes the paper.

II. Related Work

In this section, we will review references related to our work, including CapsNets Routing Algorithm and CapsNets Architecture.

A. Capsule Routing Algorithm

The capsule routing algorithm is a vital component within the CapsNets framework, which captures the spatial hierarchical relationships and pose changes of the input data by iteratively assigning weights to enhance the identification and generalization of the model. Dynamic routing and EM routing were the first capsule routing algorithms proposed by Hinton *et al.* [?], [?]. Following that, many attempts have been devoted to the developments of capsule routing mechanisms. For example, Hahn *et al.* [?] proposed a self-routing strategy for capsules allocation. Wang *et al.* [?] proposed a capsule routing algorithm using orthogonal matrices to reduce redundancy between capsules. Choi *et al.* [?] introduced the attention mechanism into capsule routing through a non-iterative feed-forward operation. Ahmed and Torresani [?] utilized straight-through estimators to make binary decisions to either connect or disconnect the routes between capsules. Yao-Hung *et al.* [?] designed routing via inverted a dot-product attention. Liu *et al.* [?] utilized a direct pose mapping using a residual routing. Liu and Zhang *et al.* [?] introduced disentangled capsule routing for fast Part-Object relational saliency.

Different from the above capsule routing algorithm, we propose a first-cluster-then-vote routing algorithm.

B. CapsNets Architecture

The earliest CapsNets architectures employed a Transforming auto-encoders [?] to compute the existence probabilities and spatial locations of entities. Later, vector CapsNets [?] are an evolved version containing a capsule routing layer and a decoder. They further consolidated this concept by proposing

Matrix CapsNets [?], which contained a primary capsule layer, two convolutional capsule layers, and a capsule classification layer. In addition, various capsule network architectures have been designed. For, example, To further add depth to the capsule architecture, Gugglberge *et al.* [?] introduced jump connections to the CapsNets architecture. Instead, Liu *et al.* [?] deepens the network by employing gestalt residuals. Wani *et al.* [?] used an attention mechanism to combine CNNs with the capsule network oh architecture. Wei *et al.* [?] proposes a new architecture by combining Transfomer modules and capsule networks. Framework in [?] consists of three main stages: convolutional stage, capsule network stage, and output layer.

Unlike the above CapsNets structure, we propose a plain-style capsule network structure similar to VGG with only 12 layers.

III. Capsule-Type Normalization Routing

In this section, we illustrate the details of proposed Capsule-Type Normalization Routing layer. It employs the first-cluster-then-vote mechanism, specifically, the NFM strategy.

A. Capsule-Type Normalization Layer

To extract the capsule features and capture the part-whole relationship of the capsule, we propose the Capsule-Type Normalization layer, as shown in Fig 3. which uses a normalization-mix-vote (NFV) strategy.

1) Normalization : The purpose of the normalization operation is to achieve internal clustering along the capsule type dimension, ensuring that each type-capsule possesses its own unique representational information, and constraining each type of capsules into a uniform distribution.

Given the capsule features $CapF^l_{C1} \in \Re^{H\times W\times C1\times D_P}$, which is composed by the pose matrix $\hat{P}^l_{C1}$ and activation A^l_{C1}, from layer l, deformation is first performed as a preprocessing step to optimize the feature representation to improve the efficiency and effectiveness of subsequent operations. By deforming the features, they are made more suitable for subsequent computational tasks:

$$\hat{\mathbf{P}}^l_{C1} \in \Re^{H\times W\times C1\times D_P} \xrightarrow{R} \Re^{HW\times C1\times D_P} \xrightarrow{P} \Re^{C1\times HW\times D_P} \xrightarrow{R} \Re^{C1\times HWD_P}; \boldsymbol{A}^l_{C1} \in \Re^{H\times W\times C1\times D_A}, \tag{1}$$

where $\hat{\mathbf{P}}^l_{C1}$ and $\mathbf{A}^l_{C1}$ represent the pose matrix and activation values of capsules in layer l, respectively, and C_1 denotes the capsule type. R and P refer to the operations of reshape and permute, respectively. The superscript l indicates the layer index. D can be set as $D_P = 16$ and $D_A = 1$ to disentangle the pose matrix $\hat{\mathbf{P}}^l_{C1}$ and activation values $\mathbf{A}^l_{C1}$ of the capsules in layer l, respectively.

Subsequently, the pose matrix $\hat{\mathbf{P}}^l_{C1}$ of each type capsule in layer l is statistically analyzed in order to normalize the capsules within each type, in which the purpose is to compute the mean $\mu \in \Re^{C1\times 1}$ and variance $\sigma^2 \in \Re^{C1\times 1}$. To this end, in light of the fact that pose matrix contains the attributes of the entity, we first compute the statistical distribution on pose matrix, as follows:

$$\tilde{\boldsymbol{P}} = \frac{\hat{P} - \mu}{\sqrt{\sigma^2 + eps}}, \tag{2}$$

where the mean μ and variance σ^2 can be computed from the spatial capsules for each type. In Eq. (2), the mean value μ is subtracted from each element of the pose matrix helps to eliminate pose bias for each type. The standard deviation σ ensures a unit standard deviation for each-type capsule.

Thereafter, the pose matrix $\tilde{\boldsymbol{P}}$ is restored to its original scale as in $\hat{\boldsymbol{P}}$:

$$\tilde{\mathbf{P}}^l_{C1} \in \Re^{C1\times HWD_P} \xrightarrow{R} \Re^{C1\times H\times W\times D_P} \xrightarrow{P} \Re^{H\times W\times C1\times D_P}, \tag{3}$$

Besides the pose matrix, the activation contains the existing probability of the entity. To involve activation in the capsules normalization, we reorganize Eq. (2) as follows :

$$\tilde{\boldsymbol{P}} = \frac{\boldsymbol{A}(\hat{P} - \mu)}{\sqrt{\sigma^2 + eps}}. \tag{4}$$

Using Eq. (4), the distributed pose matrix is able to be dynamically activated, enhancing the informative pose representation while diminishing the unimportant version.

To discriminate the contributions of different types of capsules, we learn a weight to attend each type as follows:

$$\tilde{\boldsymbol{P}} = \gamma\frac{\boldsymbol{A}(\hat{P} - \mu)}{\sqrt{\sigma^2 + eps}}, \tag{5}$$

in which the learnable parameter γ will interpret the existing probability of the normalized pose, helping to improving the model expressiveness and generalization. In addition, γ carries the burden of feature selection, and is able to self-tune itself to enhance or suppress specific features during the training process, thus dramatically improving the expressive power and generalization ability of the model.

2) Mix: The purpose of the Mix operation is to integrate multiple normalized types of capsules to form comprehensive capsules.

Based on the learnable contribution parameter γ, we compute a normalized correlation vector M_γ to quantify the relative importance of different types of capsules as follows:

$$M_\gamma = \{m_i\} = \frac{\gamma_i}{\sum_{j=1}^{C1}\gamma_j}\ ,\ i,\ j = 1, 2, \ldots, C1\ , \tag{6}$$

where m_i is the weight of the gamma after weighting, i, j denotes the index of the lth layer capsule.

Immediately thereafter, the updated weights M_γ are applied to the feature representation of each capsule through element-wise multiplication, ensuring that the contribution of each capsule aligns with its importance. Finally, all the weighted capsule features are aggregated via summation to form a unified and comprehensive capsule representation. The mixed capsules can be obtained by :

$$\boldsymbol{P_{weight}} = M_\gamma\tilde{\boldsymbol{P}} = \sum_{i=1}^{C1} m_i\tilde{P} = \sum_{i=1}^{C1}\frac{\gamma_i\tilde{\boldsymbol{P}}}{\sum_{j=1}^{C1}\gamma_j}, \tag{7}$$

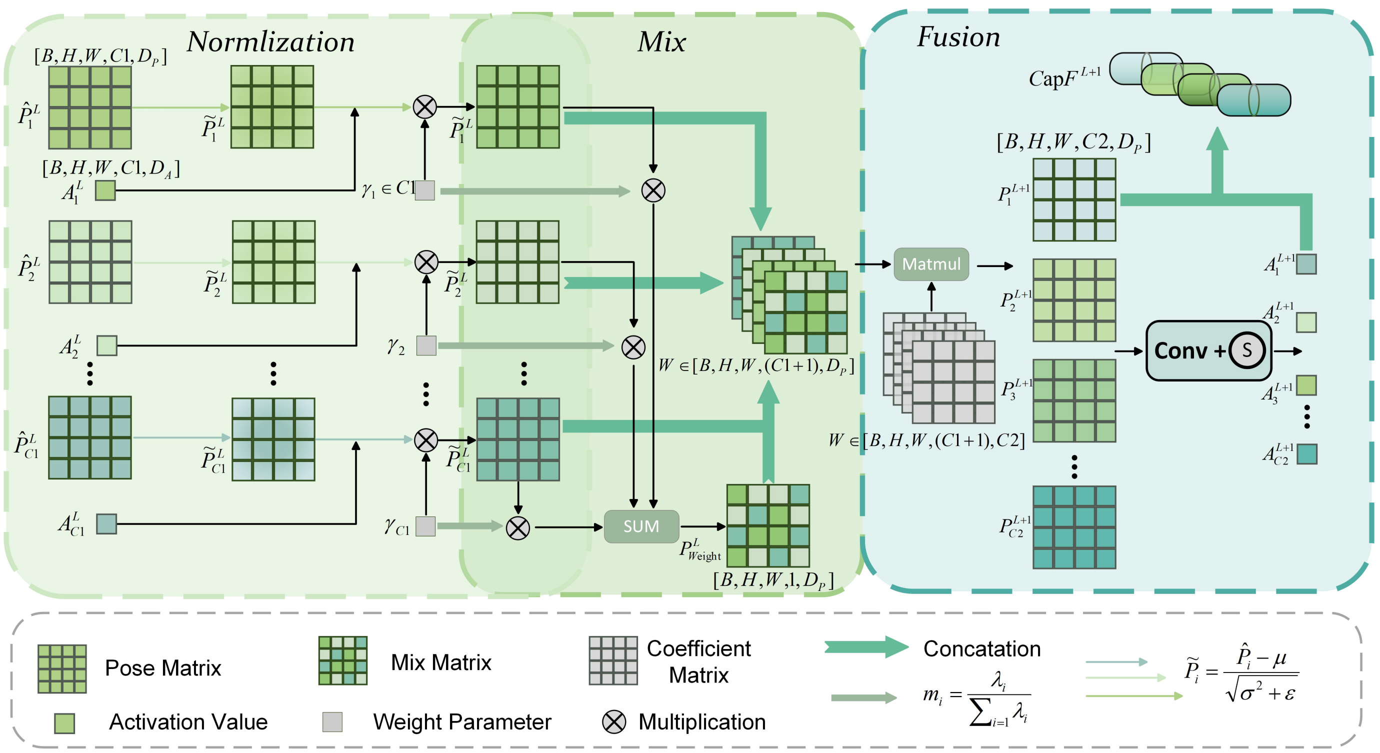

Fig. 3: The core feature of the capsule routing mechanism is the integration of the NFV (Normalization-Mix-Vote) strategy, a strategy that significantly improves the performance and efficiency of capsule networks through a combination of normalization, sample mixing, and feature fusion.

In Ea. (7), the mixed capsules not only integrates individual type capsule characteristics but also captures interrelationships and synergies, providing a richer and more comprehensive information base for subsequent high-level feature extraction and classification tasks.

3) Transformation vote: The transformation vote operation is purposed to vote for high-layer capsules from low-layer versions, which explores the part-whole relational property of the entity.

After processing a weighted capsule, it is concatenated with other unweighted capsule feature representations to retain the full information of all capsules.

$$\boldsymbol{P_{mix}} = \mathrm{Cat}(\tilde{\boldsymbol{P}}, \boldsymbol{P_{weight}}) \tag{8}$$

where $P_{mix} \in \Re^{H\times W\times(C1+1)\times D_p}$ represents the mixing capsule. The term $C1+1$ indicates $C1$ capsules for specific local features and one additional capsule for high-level global features. This $C1+1$ structure enhances the model's robustness and generalization in complex visual tasks.

To implement the capsules vote, a transformation matrix $\boldsymbol{M} \in \Re^{H\times W\times C2\times(C1+1)}$ is learned to project low-layer capsules to high-layer capsules via matrix multiplication as follows:

$$\begin{bmatrix} \boldsymbol{P_1^{(l+1)}} \\ \boldsymbol{P_2^{(l+1)}} \\ \vdots \\ \boldsymbol{P_{C2}^{(l+1)}} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,C1+1} \\ w_{2,1} & w_{2,2} & \dots & w_{2,C1+1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{C2,1} & w_{C2,2} & \dots & w_{C2,C1+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{P_1^l} \\ \boldsymbol{P_2^l} \\ \vdots \\ \boldsymbol{P_{C1+1}^l} \end{bmatrix}, \tag{9}$$

where we introduce a learnable coefficient matrix. By multiplying with this learnable coefficient matrix $\boldsymbol{M} \in \Re^{H\times W\times C2\times(C1+1)}$

where P^{l+1} and P^l represent the pose matrices carried by the capsules in the $l+1$ and l layers, respectively, and w is a weight matrix that connects the capsules in the two neighboring layers and is used to transfer.

The pose matrix $\boldsymbol{P}^{l+1}$ of the generated high-level capsule again contains its potential activation probability, which is convolved and activated the pose matrix values $\boldsymbol{A}^{l+1} \in \Re^{H\times W\times\times C2\times D_A}$ of the high-level capsules, after which we splice the activation value of the generated high-level capsule with the pose matrix to generate the high-level capsule.

$$\begin{aligned} \boldsymbol{A}^{l+1} \in \Re^{H\times W\times C2\times D_A} &= Sigmoid(f_{conv}(\boldsymbol{P}^{l+1})) \\ \boldsymbol{CapF}^{l+1} \in \Re^{H\times W\times C2\times D} &= f_{conv}(Cat(\boldsymbol{P}^{l+1}, \boldsymbol{A^{l+1}}))' \end{aligned} \tag{10}$$

where $Sigmoid(\cdot)$ means the sigmoid function, $f_{conv}(\cdot)$ means the pointwise convolution, Cat is a concat operation.

In summary, a Capsule-Type Normalization capsule layer was designed and implemented, incorporating an innovative ”normalization-mixing-fusion” routing algorithm strategy to achieve deeper capsule feature extraction.

To this end, the capsule maps of layer $(l+1)$, *i.e.*, $\mathbf{P}^{l+1}$ and $\mathbf{A}^{l+1}$, can be obtained. Algorithm 1 illustrate the CTNR based CapsNet.

B. Discussion

1) Analysis on Complexities: The Normlized Routing mechanism we devise achieves efficient information decou-

Algorithm 1 CTNR based CapsNet. $\hat{X}$ is the feature maps of the input image. P_*^* and A_*^* are the pose matrices and activation values, respectively.

```
Procedure Capsule-Type Normalization Routing ($\hat{X}$)
    1. Primary capsules generation
    |   $\hat{P}^l, A^l = PrimaryCaps(\hat{X})$
    2. Normalization:
    |   Normalized pose matrix:
    |     $\tilde{\boldsymbol{P}}^{\boldsymbol{l}} = \gamma \frac{\boldsymbol{A}(\hat{P} - \mu)}{\sqrt{\sigma^2 + eps}}$ =Eq. 5($\hat{P}^l, A^l$)
    3. Mix:
    |   Weight matrix:
    |     $M_\gamma = \frac{\gamma_i}{\sum_{j=1}^{C1} \gamma_j}$ = Eq. 6($\gamma$)
    |   Composite capsule pose matrix:
    |     $\boldsymbol{P_{weight}} = \sum_{i=1}^{C1} \frac{\gamma_i \tilde{\boldsymbol{P}}}{\sum_{j=1}^{C1} \gamma_j}$ = Eq. 7($\boldsymbol{P}^l$)
    |   Mix capsule pose matrix:
    |     $\boldsymbol{P_{mix}} = Cat(\tilde{\boldsymbol{P}}, \boldsymbol{P_{weight}})$ =Eq.8($\tilde{\boldsymbol{P}}, \boldsymbol{P_{weight}}$)
    4. Transformation vote:
    |   Capsule Fusion:
    |     $\boldsymbol{P^{l+1}} = \boldsymbol{W} \otimes \boldsymbol{P_{mix}}$ = Eq. ??($\boldsymbol{P_{mix}}, \boldsymbol{W}$)
    |   Heigher Layer Capsule Activition value:
    |   $\mathbf{A}^{l+1} = Sigmoid(f_{conv}(\boldsymbol{P^{l+1}}))$ = Eq. 10
    |   Heigher Layer Capsule Activition value:
    |     $\boldsymbol{CapF^{l+1}} = f_{conv}(Cat(\boldsymbol{P^{l+1}}, \boldsymbol{A^{l+1}}))$ = Eq. 10
```

pling between the bottom and top capsules by introducing a coefficient matrix, an innovation that drastically reduces the memory requirements for network operation. Specifically, the coefficient matrix allows the network to handle inter-capsule connections in a more economical way, avoiding the potential memory consumption problem in the all-connectivity mode, and thus optimizing the efficient use of computational resources while guaranteeing the performance of the model.Assuming that the low-level capsule is C_1 and the high-level capsule is C_2, the memory he occupies in the route is as follows:

$$\boldsymbol{Mem} = B \times H \times W \times C_1 \times C_2 \times D_p \tag{11}$$

The number of parameters of our proposed CTNR is:

$$\begin{aligned} \boldsymbol{Mem_{CTNR}} = & B \times H \times W \times (C_1 + 1) \times D_p \\ & + B \times H \times W \times C_2 \times (C_1 + 1) \end{aligned} \tag{12}$$

$$\begin{aligned} Ratio = & \frac{\boldsymbol{Mem}}{\boldsymbol{Mem_{CTNR}}} \\ = & \frac{B \times H \times W \times C_1 \times C_2 \times D_p}{B \times H \times W \times (C_1 + 1) \times (D_p + C_2)} \\ = & \frac{D_p}{(1 + \frac{1}{C_1}) \times (\frac{D_p}{C_2} + 1))} \end{aligned} \tag{13}$$

Here we give a comparison to show the performance of the proposed CTNR. Assuming $C_1, C_2 \in [2, \infty)$ and calculated from Eq. (13) that $Ratio \in [1.18, 16)$, in our experiment, when $C_1 = C_2 = 32$ and $D_p = 16$, $\frac{\boldsymbol{Mem}}{\boldsymbol{Mem_{GN}}} \approx 10$.

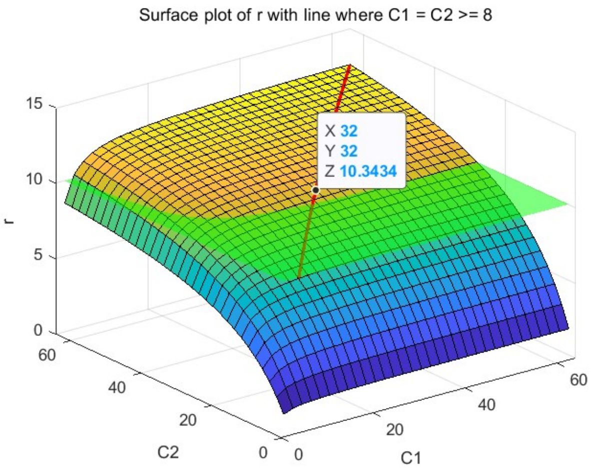

Fig. 4: The green plane corresponds to the coefficient matrix with a capsule fully connected memory occupancy ratio of 10, while the red line indicates the memory ratio trend when the number of capsules is equal and not less than 32, specifically marking the ratio value of 10.34 for a number of 32 capsules. This presents a direct comparison of memory efficiency for different configurations.

2) *Differences from Group Normalization:* The CTNR approach differs from Group Normalization (GN) in several ways: 1) Different application scenarios: GN addresses the problem that BN is ineffective in small batch search, whereas we address the problem of capsule network architecture. 2) Different core concepts: GN reduces the internal covariate bias by partitioning the feature graph into different groups and normalizing independently within each group as a way to reduce internal covariate bias. Whereas we are normalizing within each capsule.3) Implementation details are different: GN is only dealing with specific groups of features. Whereas capsule network is normalized only for the pose matrix of the capsule not for the activation values.

IV. Capsuele Network

A. *Different From Gn*

In this section, based on the proposed normlized routing algorithm, we design a deep CapsNet architecture, named Nomlized CapsNet (NCaps).

B. *Primary Capsule layer*

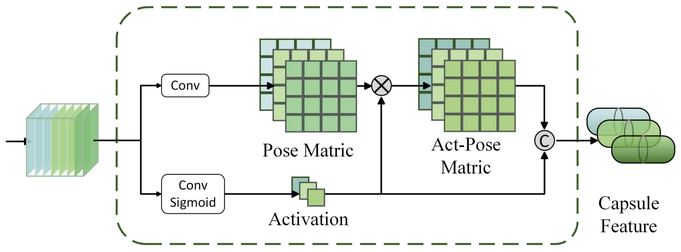

Fig. 6: The architecture of PrimaryCapsule

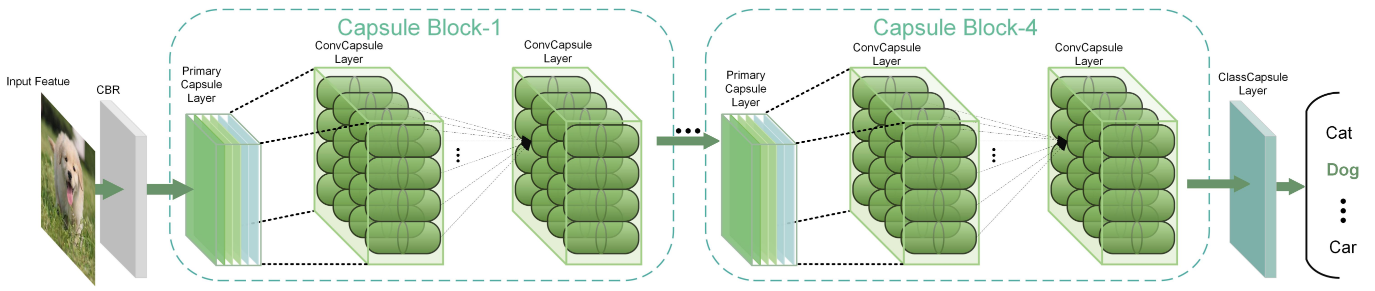

Fig. 5: The capsule network architecture we demonstrate consists of the following key components: a front-loaded convolutional layer for base feature extraction, four series-connected capsule blocks for multi-level feature analysis with inter-capsule interactions, and a final category capsule layer for outputting classification decisions.

In the Normlized Capsule Rooting, capsule formation begins with the primary capsule layer, which is responsible for initializing the capsule units that build the foundation of the entire capsule network.

Each Capsule contains two core components: a Pose Matrix and an Activation Value. The pose matrix encodes the geometric properties of the entity, and the activation value indicates the probability or importance of the entity represented by the capsule being present in the current input. Suppose $\mathbf{P} \in \Re^{W\times H\times C\times D_P}$ and $\mathbf{A} \in \Re^{W\times H\times C\times D_A}$ are the pose matrix and the activation of the capsule maps. $D = \{D_P = 16, D_A = 1\}$ are the dimensions of the pose matrix and activation values, respectively.

Given the input features $\hat{X}$, the pose matrix $\mathbf{P}$ is first generated by one layer of convolution operation, which contains not only the existence probability of the capsule but also the energy value. Next, we activate the pose matrix $\mathbf{P}$ to obtain the activation values $\mathbf{A}$ of the capsule, a step that usually involves the application of a nonlinear activation function to ensure that the output values better represent the features of the capsule. Finally, the activated pose matrix is spliced with the activation values of the capsule to generate the final output, thus realizing the mapping of the input features $\hat{X}$ to the capsule features $\mathbf{CapF}$ and providing an efficient feature representation for the subsequent convolutional capsule layer. The whole process of acquiring $\mathbf{CapF}$ can be expressed as equation 1:

$$\boldsymbol{P} = f_{conv}(\hat{X}), \boldsymbol{A} = Sigmoid(f_{conv}(\hat{X})) \tag{14}$$

$$\hat{\boldsymbol{P}} = \boldsymbol{P} \times \boldsymbol{A} \tag{15}$$

$$\boldsymbol{CapF} = f_{cat}(\hat{\boldsymbol{P}}, \boldsymbol{A}) \tag{16}$$

where $\boldsymbol{P}$ represents the pose matrix obtained by applying the point convolution function f_{conv} to the input $\hat{\boldsymbol{X}}$, $\boldsymbol{A}$ represents the activation values obtained by applying the sigmoid activation function to the convolution result, $\hat{\boldsymbol{P}}$ represents the enhanced pose matrix obtained by the element-wise multiplication of $\boldsymbol{P}$ and $\boldsymbol{A}$, f_{conv} represents the point convolution operation, Sigmoid represents the sigmoid activation function, $\times$ represents the element-wise multiplication, f_{cat} represents the concatenation operation, and CapF represents the capsule feature obtained by concatenating $\hat{\boldsymbol{P}}$ and $\boldsymbol{A}$.

C. Class Capsule layer

The features generated by the capsule blocks eventually flow into the Class Capsule Layer, which is similar in design to the Convolutional Capsule Layer, but functionally specializes in classification tasks. In the Class Capsule Layer, instead of applying a ”normalization-mix-fusion” strategy, the focus is directly on the final decision - determining the class to which the input data belongs.

D. Overall Architecture

We introduce the NCaps architecture. As shown in the Fig 5, it contains three main parts: a convolutional layer, four capsule blocks, and a class capsule layer. Given input images $x \in \Re^{B\times C\times H\times W}$,where the B-axis represents the batch size; the C-axis represents the feature dimension; and the H- and W-axes represent the height and width of the image, respectively, which are used to describe the spatial resolution of the image. it will first go into the convolutional layer for processing to simply extract its features to get the processed result $\hat{X} \in \Re^{B\times C_0\times H\times W}$. The preliminarily processed feature $\hat{X}$ representations are then fed into four capsule blocks for deeper feature analysis and acquisition. Each capsule block consists of a Primary capsule layer and two Convolutional capsule layers for more accurate capture and representation of capsule features $CapF \in \Re^{B\times H\times W\times N\times D}$ in the image ,where N represents the number of capsules; and where D represents the dimension of each capsule that describes the feature information captured and expressed by that capsule unit.

E. Loss Function

From the activation values of each category capsule output by the class capsule layer, we train the capsule network using the Spread Loss function, a loss function specifically designed for multi-categorization tasks to improve the network's discrimination between the correct categories.

The core idea of Spread Loss is that for each sample, the network should widen the gap between the activation value of the correct category and the next highest activation value above a certain threshold. Specifically, let T_k be the label of the sample belonging to the kth class (1 if it belongs to that class, 0 otherwise), a_k be the capsule activation value of the

TABLE I: Capsule Network Architecture

Capsule Network Architecture	
A	B
Input Image	
small Image(32 × 32)	Big Image(300 × 300)
Conv3(16 × 16) Batch Normalization Relu	Conv(150 × 150) Batch Normalization Relu
Capsule Block-1	
Primary Capsule Layer-64(8 × 8) CTN Layer-64 CTN Layer-64 Batch Normalization Gelu	Primary Capsule Layer-8 (75 × 75) CTN Layer-8 CTN Layer-16 Batch Normalization Gelu
Capsule Block-2	
Primary Capsule Layer-48(4 × 4) CTN Layer-48 CTN Layer-48 Batch Normalization Gelu	Primary Capsule Layer-16 (38 × 38) CTN Layer-8 CTN Layer-16 Batch Normalization Gelu
Capsule Block-3	
Primary Capsule Layer-48(2 × 2) CTN Layer-48 CTN Layer-48 Batch Normalization Gelu	Primary Capsule Layer-16 (19 × 19) CTN Layer-16 CTN Layer-32 Batch Normalization Gelu
Capsule Block-4	
Primary Capsule Layer-48(1 × 1) CTN Layer-48 CTN Layer-48 Batch Normalization Gelu	CTN Layer-32 (10 × 10) CTN Layer-32 CTN Layer-32 Batch Normalization Gelu
Primary Capsule Layer Class Capsule Layer	FC1-1280 FC2-1000

kth class, and m be the gradually increasing minimum interval threshold, then Spread Loss can be defined as:

$$L_i = \sum_k (T_k \max(0, m - (a_{y_i} - a_k)))^2$$

where y_i is the correct category for the sample i and a_{y_i} is the activation value for that category. This function encourages the network to separate the activation values of the correct category by at least m from the activation values of all the incorrect categories.

V. Experiment and Analysis

In this section, we will carry out abundant experiments and analysis to provide a comprehensive understanding of the proposed method.

A. Dataset

We evaluate the proposed salient object detection network on four public benchmarks.

The **MNIST** [?] dataset contains 60,000 training images and 10,000 test images, each of which is a 28x28 pixel handwritten digit categorized into 10 categories from 0 to 9.

The **SVHN** [?] dataset contains house door numbers extracted from Google Street View images and is designed for machine learning and computer vision tasks. The dataset is divided into 10 categories (numbers 0 to 9) and includes 73,257 training images and 26,032 test images, each of which is a 32x32 pixel color image.

The **SmallNORB** [?] dataset is used to evaluate 3D object recognition algorithms and contains five categories of objects: four-wheelers, mannequins, airplanes, trucks, and animals. Each category contains 10 instances, each taken at different azimuths, lighting conditions, and elevation angles, providing a total of 24,300 grayscale images of 96x96 pixels.

CIFAR-10 [?] contains 60,000 color images of 32x32 pixels divided into 10 categories (e.g., airplanes, cars, cats, etc.), and the dataset consists of 50,000 training images and 10,000 test images.The **CIFAR-100** [?]dataset is similar but has 100 categories of 600 images each, and the dataset is also divided into 50,000 training images and 10,000 test images. images and 10,000 test images, and organized in 20 super categories.

The **ImageNet-1k** [?] dataset is a widely used image classification dataset containing about 1.2 million images categorized into 1000 classes. It is divided into training, validation and test sets for training and evaluating image classification models. Due to its large scale and diversity, many state-of-the-art image classification models are trained and tested on this dataset.

B. Implementation Detail

For the MNIST, FashionMNIST, smallNORB, and SVHN datasets, the GNCaps model was developed and run using Python 3.9 in the PyTorch version 1.12.1 environment. For the training configuration, we chose the Adam optimizer with an initialized learning rate of 0.001 and a batch size of 128, as well as a configuration of the number of capsules in the network architecture of [64, 64, 48, 48, 48, 48, 48, 48], a configuration that ensures a reasonable number of capsule units in each layer to optimize the parameter learning process. To accelerate the training process, we utilize 1 NVIDIA GeForce RTX 3090 GPU equipped with 24 GB of video memory.

For the CIFAR-10 and CIFAR-100 datasets, we trained the models on two different GPU platforms: the NVIDIA GeForce RTX 3090 vs. the NVIDIA A100. on the RTX 3090, we followed most of the training configurations similar to the previous dataset, but with the batch size adjusted to 80. on the A100 GPU, the model was developed and run in PyTorch version 1.9.1 and Python 3.8, with an initial learning rate set to 0.001 and a batch size of 128. The network architecture, with the number of capsules, was configured in this A100 as [128, 64, 128, 64, 64, 64, 64, 64].

For the ImageNet-1k dataset, we used parallel training on two A100 GPUs, using the same strategy as EfficientNetV2, including the MixUp method. The training environment is PyTorch 1.9.1 and Python 3.8, the batch size is set to 32, the number of capsules is [32,32,32,32,32,32,32], and the optimizer is chosen as SGD.

1) Evaluation on MNIST and FashionMinst: Table I provides a detailed comparative analysis, clearly indicating that the GNCaps model demonstrates superior performance in tests on the MNIST dataset. Specifically, GNCaps achieves a low test error rate of 0.62%, which corresponds to an accuracy rate of up to 99.54 %. This result not only outperforms the performance of traditional CNNs, but also stands out in

TABLE II: Mean error (%) and accuracy (%) on MNIST. - represents no result released from the original paper or related papers. The best method is marked by **bold.**

Method	Mean error MNIST	Accuracy MNIST
GNCaps	**0.69**	**99.54**
ResCaps [?]	0.72	99.45
Baseline CNN [?]	0.8	99.22
BCN [?]	2.5	97.50
Dynamic Rooting [?]	0.77	99.23
G-Caps [?]	1.58	98.42
Matrix-CapsNet with EM routing [?]	0.8	99.20
Aff-Caps [?]	0.77	99.23
U-Routing [?]	0.8	-
DA-CapsNet [?]	-	99.53
AA-CapsNet [?]	-	99.34
CapProNet [?]	-	94.98

comparison with similar capsule network models, highlighting the significant advantages of GNCaps in handling handwritten digit recognition tasks.

On the FashionMNIST dataset, our model also demonstrates excellent performance, achieving 92.34% accuracy with a reduced mean error of 0.69%, which outperforms classical convolutional neural network models, such as ResNet-18 and VGG16, demonstrating the effectiveness of our approach in dealing with the more complex and diverse task of apparel image classification and the Superiority.

The data in Table II clearly show that our method significantly outperforms the other technical solutions compared on the SVHN dataset. Specifically, our model achieves 96.01% in terms of accuracy and demonstrates a significant advantage in terms of the mean error metric, which highlights the strong performance and robustness of our method in handling the task of street view house digit recognition.

TABLE III: Mean error (%) and accuracy (%) on SVHN. NCaps means our entire framework consisting of 4 blocks. - represents no result released from the original paper or related papers. The best method is marked by **bold.**

Method	Mean error SVHN	Accuracy SVHN
NCaps	**0.72**	**96.01**
Baseline CNN [?]	0.8	91.28
Efficient-Caps [?]	2.5	93.12
EM-Caps [?]	-	87.42
DA-Caps [?]	-	94.82
CapProNet [?]	0.77	93.41
AA-CapsNet [?]	-	92.23
AR-CapsNet [?]	-	85.98

2) Evaluation on SVHN and SmallNORB: In Table III, the performance comparison for the SmallNORB dataset is particularly outstanding, and the GNCaps model exhibits an average error rate of 0.89%, which not only breaks the existing record, but also ranks the lowest among all the reference methods, which is a strong proof of the leading edge of GNCaps in the task of processing three-dimensional object recognition and classification.The SmallNORB dataset is well known for the complexity and diversity of its three-dimensional images. The SmallNORB dataset is well known for the complexity and diversity of its stereo images, and the excellent performance of GNCaps on this dataset further validates its power in capturing and resolving spatial features of objects.

TABLE IV: Mean error (%) on smallNORB. The superior approach is highlighted in **bold.**

Method	Mean error
NCaps	**0.89**
ResCaps [?]	0.91
Baseline CNN [?]	5.2
FREM [?]	2.20
Dynamic Rooting [9]	2.70
STAR-Caps [?]	1.80
Matrix-CapsNet with EM routing [?]	1.8
VB-Rooting [?]	1.60
U-Routing [?]	2.20
Efcient-CapsNet [?]	2.54

TABLE V: Mean error (%) and accuracy (%) on CIFAR-10. - represents no result released from the original paper or related papers. The best method is marked by **bold.**

Method	Mean error cifar-10	Accuracy cifar-10
NCaps	**0.74**	**92.83**
ResCaps [?]	1.14	92.38
OrthCaps-D [?]	-	86.84
OrthCaps-S [?]	-	90.56
Baseline CNN [?]	19.2	-
EM-Caps [?]	11.6	82.20
DA-Caps [?]	-	85.47
CapProNet [?]	-	80.84
Self-Routing [?]	7.86	92.14
AA-CapsNet [?]	-	71.60
AR-CapsNet [?]	-	85.98
MobileNetV2 G32 [?]	-	91.87
Radix VGG20 [?]	-	92.20
kMobileNet V3 Large 16ch [?]	-	92.80

3) Evaluation on Cifar-10/100: In the comparative analysis in Table IV, our GNCaps model achieves 92.83% accuracy on the CIFAR-10 dataset, a result that establishes our new benchmark in the field (state-of-the-art, SOTA). This not only highlights the significant advantages of GNCaps in handling difficult image classification tasks, but also further validates its superior performance in parsing the rich and diverse image features in the CIFAR-10 dataset, which makes it a leader among similar capsule network models.

On the CIFAR-100 dataset, our model achieves an accuracy of 63.38% while keeping the average error at 15.8%. This result reflects the effectiveness of our approach in handling this more challenging multi-category classification task, where the fine-grained categories contained in the CIFAR-100 dataset place higher demands on the model's recognition ability and generalization performance.

4) Evaluation on ImageNet: On the ImageNet dataset, our model demonstrates excellent performance with a Top-1 accuracy of 71% and a Top-5 accuracy of 91% .The training loss is 2.9 while the testing loss is 1.69 as shown in the figure8. This achievement not only highlights the power of our model in handling large-scale image classification tasks, but also demonstrates the robust performance in complex visual recognition challenges. As shown in the Fig 7, these results visualize the effectiveness and competitiveness of our approach.

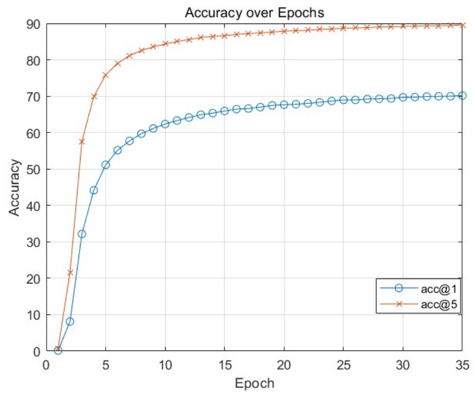

Fig. 7: NFMCaps in ImageNet -1k per Epoch top1and top5 accuracy.

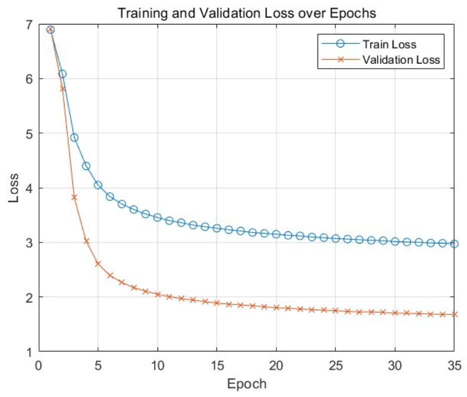

Fig. 8: NFMCaps in ImageNet -1k Losses per Epoch training and testing.

C. Ablation Study

In this section, we will conduct several experiment to analyze the role of each component in our proposed framework.

1) Number of capsules: In order to gain insight into how the number of capsules affects the model performance, we executed an exhaustive sequence of experiments on the CIFAR-10 dataset, please refer to Table 5 for the detailed results.By comparing and analyzing the performance on CIFAR-10, we found that the model demonstrated optimal performance when the capsule network architecture of the GNCaps model was set to 128 and 64 capsules for the first and second block settings, respectively, and 32 capsules for each of the third and fourth blocks set to 64 capsules, the model exhibits optimal performance.

2) Routing Strategy: Exploring the proposed Normalization-Mix-Fusion (NMF) strategy on the CIFAR-10 dataset. In the ablation experiments conducted on the NMF (Normalization-Mix-Fusion) strategy, we separately examined the impact of each component on the model performance. It was found that the model achieved 91.92% accuracy on the CIFAR-10 dataset when only the normalization strategy was used, while when the normalization strategy was used in conjunction with the fusion (NF) strategy, the accuracy slightly decreased to 91.83%, and if the normalization strategy was paired with the mixing (NM) strategy, the model accuracy was improved to 92.29%. These results emphasize the important role of hybrid strategies in enhancing model performance and generalization. In subsequent experiments, we delved into the effect of the timing of the execution of the Fusion strategy in the NMF strategy on the performance of the model. When the Fusion strategy was placed before the Normalization and Mix strategies, i.e., in the order of FMN, we observed a significant decrease in the model performance, with an accuracy of only 87.68%. This result mainly stems from the failure of the capsules to be in a uniform distribution during fusion, coupled with the secondary activation effect of the capsules, both of which together lead to a significant degradation in model performance. This suggests that the execution order of the strategy is crucial to the effectiveness of the capsule network, and an improper order will interfere with the coordinated work among the capsules, which in turn affects the overall performance of the model.

TABLE VI: Experimental results with different Number of capsules on CIFAR-10 dataset.

Method	cifar-10	
	Mean error (%)	Accuracy (%)
[64, 64, 48, 48, 48, 48, 32, 32]	1.33	92.37
[64, 64, 48, 48, 48, 48, 48, 48]	1.22	92.42
[128, 64, 128, 64, 64, 64, 64, 64]	1.18	92.71
[128, 64, 128, 64, 128, 64, 64, 64]	1.16	92.74
[64, 128, 64, 128, 64, 64, 64, 64]	**1.03**	**92.83**

TABLE VII: Experimental results of different NFM strategies on the CIFAR-10 dataset.

Normlization	*Mix*	*Fusion*	Mean error ↓	Accuracy ↑
✓			1.33	92.28
✓	✓		1.30	92.30
✓		✓	1.55	91.83
✓	✓	✓	**1.22**	**92.42**

TABLE VIII: Experimental results with different combination methods of NFM on CIFAR-10 dataset.

Description	Mean error ↓	Accuracy ↑
Fusion +*Mix* +*Normlization*	3.3	87.68
Normlization + *Mix* + *Fusion*	**1.03**	**92.83**

D. Comparison with EM Routing

We set the Block number to 1 constituting NFMCaps-2B with EM-Routing on the CIFAR-10 dataset for comparison. NFMCaps have the best inference speed and accuracy under the equivalent condition of having only one Block.

VI. Conclusions

In this study, an innovative N-Capsule Routing (N-Caps) method is proposed to address the computationally intensive

Model	Memory (G)	Parameter (M)	Time (ms)	Error (%)	Accuracy (%)
NFMCaps-2B	5.4	16.4	**3.2**	**0.91**	**88.52**
ResCaps-2B [?]	**4.8**	16.4	4.2	1.14	88.28
EM-Caps [?]	23.70	**0.02**	5.3	11.6	82.20
ResNet-101 [?]	9.84	42.51	12.1	6.43	86.65

TABLE IX: Comparison of different models based on Memory, Parameters, Time, Error, and Accuracy.

problem of capsule networks, which significantly reduces the computational burden by introducing capsule fusion matrices instead of the traditional weight matrices, while maintaining the part-whole relationship modeling capability unique to capsule networks. Experimental results on several datasets (e.g., SVHN, SmallNORB, CIFAR-10/100, ImageNet) validate the efficiency and performance advantages of the GN-Caps method.The proposed GN-Caps method provides a strong support for the wide application and efficiency optimization of capsule networks, and demonstrates the capsule network's wider use in the field of computer vision The potential of the GN-Caps method.

Acknowledgment