A Light-weight Deep Feature based Capsule Network

Chandan Kumar Singh, Vivek Kumar Gangwar, Anima Majumder, Swagat Kumar,
Prakash Chanderlal Ambwani, Rajesh Sinha

***Abstract*— Capsule Network (CapsNet) has motivated researchers to work on it due to its distinct capability of retaining spatial correlations between image features. However, its applicability is still limited because of its intensive computational cost, memory usage and bandwidth requirement. This paper proposes a computationally efficient, lightweight CapsNet which paves its way forward for deployment in constrained edge devices as well as in web based applications. The proposed framework consists of Capsule layers and a deep feature representation layer as an input for capsules. The deep feature representation layer comprises of a series of feature blocks, containing convolution with a 3×3 kernel followed by batch normalization and convolution with a 1×1 kernel. The deeper or better represented input features help to improve recognition performance even with lesser number of capsules, making the network computationally more efficient. The efficacy of the proposed framework is validated by performing rigorous experimental studies on different datasets, such as CIFAR-10, FMNIST, MNIST and SVHN which include images of object classes as well as text characters. A comparative analysis has also been done with the state-of-the-art technique CapsNet. The comparison with recognition accuracy ensures that, the proposed architecture with deep input features provides more efficient routing between the capsules as compared to CapsNet. The proposed lightweight network has scaled down the number of parameters up to 60% of CapsNet, which is another significant contribution. This is achieved by collaborative effect of deep feature generation module and parametric changes performed in the primary capsule layer.**

I. INTRODUCTION

Development of vision based algorithms for automatic recognition of real-world objects was almost a dream for computer vision researchers until the variants of Convolutional Neural Networks (CNNs), such as LeNet [10] and AlexNet [8] were introduced. The enormous increase in computational efficacy with the advancement of GPU based machines has motivated many computer vision researchers to progress towards data driven learning techniques, which has given birth to CNN architectures, like VGG [18], GoogLeNet [20], ResNet [3], etc. The object recognition results using all these algorithms have shown outstanding performances on large-scale object datasets, like Imagenet and COCO, and have consistently improved the performances in each successive year along with introduction of new architectures.

However, in-spite of all these success stories, one of the major drawbacks with these CNN based techniques, which was left unnoticed is that the CNNs by its nature employ invariance of features against their spatial position [17]. The main reason for such behavior lies in the use of the *pooling* technique. According to Geoffrey Hinton , "*The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster*" [16]. Generally, max pooling (or any kind of pooling) is used to down-sample the feature size to a manageable level. Besides reducing the size of the feature vector, it also maintains translation in-variance. By only considering the maximum value, essentially we are only interested if a feature is present in a certain window. However, we do not really care about their exact locations. If we have a convolution filter, which detects edges, here an edge gives a high response to this filter. The max pooling operation only keeps the information if an edge is present and throws away the rest; which may also include the location and the spatial relationships between certain features. Or in other words, the CNNs with max pooling do not retain any internal representation of the geometrical constraints present in the data. Then one obvious question comes to our mind is that, how these CNNs are managing to perform so well. The answer is, *by training the network with a large amount of labeled data.* For example, in order to be able to efficiently detect a cat at any given viewpoint, we need to have a set of training images containing different instances on that viewpoints of the cat as the network does not encode the prior knowledge of the geometrical relationships.

This issue was first addressed by Hinton et al.[16], where they have introduced a deep network, called *CapsNet* to overcome this limitation of CNNs. The CapsNet is shown to provide good results for image classification, with high recognition accuracy on MNIST dataset. Some of the other advantages of CapsNet are that it needs lesser amount of training data than CNNs and also provides information about pose and related parameters to identify the entities. The capsule network is also shown to be robust to input data transformations and "white box" attacks [17]. However, not much works have been done to explore the full potential of this network. There are some important areas where more attention need to be given. First, state-of-the art results are achieved by using MNIST dataset, which is a low-resolution character recognition image dataset. The network is not yet generalized by validating it with difficult datasets, like CIFAR10[7], SVHN[14] etc. Secondly, the training process of the existing CAPsNet is very slow due to inner loop, which restricts the network from using it for wider and more complex applications.

Chandan Kumar Singh, Vivek Kumar Gangwar, Anima Majumder, Swagat Kumar, Prakash Chanderlal Ambwani and Rajesh Sinha are researchers at TCS Innovation Labs, India.
ck.singh1@tcs.com, vivek.gangwar@tcs.com, anima.majumder@tcs.com, swagat.kumar@tcs.com, prakash.ambwani@tcs.com, rajesh.sinha@tcs.com

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

In this paper we propose a deep learning framework based on CAPsNet which is more compact (scaled down the number of parameters up to 60% of CapsNet) and yet more efficient in terms of recognition accuracy. The basic intuition behind this approach is that, if more comprehensive and better represented features (deeper features) are given as an input to a capsule, it can create better routing within the capsule network, resulting into improved recognition accuracy. A comprehensive and better represented feature vector is obtained by using a 1×1 convolutional kernel in the feature block. The same concept has also been used in [20] for dimensionality reduction. To further reduce the parameters, we use the kernel size 3×3 instead of 9×9 that was used in [16]. Another significant contribution is made towards feature enhancement by incorporating more number of feature blocks prior to the primary capsule layer. The underlying concept for this approach is that, use of deep features as input to the primary capsule layer results in better routing even with much lesser number of capsules. Similar assumption is also made in CNNs based architectures [18], [20], where deeper and deeper layers are used to enhance the features for achieving better accuracy. The choice of parameters and the aforesaid assumptions are validated through rigorous experimental and ablation studies. Several datasets including CIFAR-10[7], FMNIST[23], SVHN[14] and MNIST[9] are used in our experiments for validating the proficiency of the proposed framework. Our proposed framework is coined as *DeepFeat-Caps* which is abbreviated from **Deep Feat**ure based **Caps**ule Network. The major contributions of this work can be summarized as follows.

1) A light-weight Capsule based deep learning framework, coined as *DeepFeat-Caps*, is proposed for both object and character recognition. This is achieved by incorporating a series of feature blocks, consisting of 3×3 convolutional kernel followed by 1×1 convolutional kernel, applied prior to the primary capsule layer.
2) The light-weight network is framed based on the assumption that, if a comprehensive and a better represented feature vector is given as an input to the primary capsule layer, it will lead to better routing within the capsule network even with lesser number of capsules. This assumption is validated through rigorous experimental studies.
3) Unlike, CapsNet that uses 32 capsules having dimension 8, we use a combination of (20, 8) in our proposed framework. It reduces the total number of parameters by approximately 60% of the parameters used in the original capsule network [16], which is a very significant contribution in the direction of model compression.
4) The efficacy of the proposed framework is validated on four different datasets, that include both object and character images. The demonstrated results show that the proposed model is efficient to provide generic solution for different kinds of application.
5) The performances of the proposed approach are compared with the state-of-the-art CapsNet [16]. The choice of the parameters (including the kernel size and optimum capsule size) and the selection of a feature block are validated though rigorous experimental studies.

Rest of this paper is organized as follows. A brief literature survey on capsule based works is presented in the following section. Section III gives a detailed explanation of the proposed approach. The experimental results, ablation studies and discussions are provided in Section IV. Finally, conclusions are drawn in Section V.

II. Literature Survey

The capabilities in the capsule based network, as popularized by [4] has drawn a large attention for its exploration among the deep learning research communities. Soon after the introduction of CapsNet [16], another work is published by Hinton et al. [5], in which an Expectation-Maximization (EM) is used for establishing the routing within the capsule blocks. The work tries to solve the performance of capsules while dealing with viewpoint variation and use a 4×4 matrix in place of vectors as used in prior work[16], to capture and learn the pose information. This work([5]) demonstrated its superior performance with smallNORB dataset. In another work, Xi et al. [21] explores the impact of performance by stacking more number of capsules or convolutional layers. [12] used modification in the capsule routing method to solve the challenge of Visual tracking where feature experiences drift. In [22] Multi-scale feature extraction with hierarchy of feature is proposed, however, the work doesn't provide compression at parameter level. [1] introduces spectral capsule network for faster conversion in comparison with EM routing capsules. The proficiency of capsules is also explored in GAN based architecture, where Jaiswal et al. [6] applied GAN over CapsNet to have better character recognition results on both MNIST and CIFAR10 datasets. A sparsified form of the last capsule layer in CapsNet architecture is used in [15] for unsupervised learning approach. Few more works in the literature, that use CapsNet in different applications are [2] for video classification,[11] for object localization, and [13] for face verification. In another work, Singh et al. [19] presented an architecturally similar model applied to text recognition for industrial applications. However, the number of parameters used in that architecture was much higher than the proposed approach. Moreover, the approach was focused on a specific industrial application and no experimental or ablation studies were made for the choice of the parameters used in that work. In contrast to all these aforesaid approaches, we introduce a generic deep framework for both object and character recognition using convolutional feature blocks along-with a scaled down version of CapsNet. The model is demonstrated to be performing significantly better than the state-of-the art CapsNet, both in terms of recognition accuracy and computational complexity when applied to different datasets.

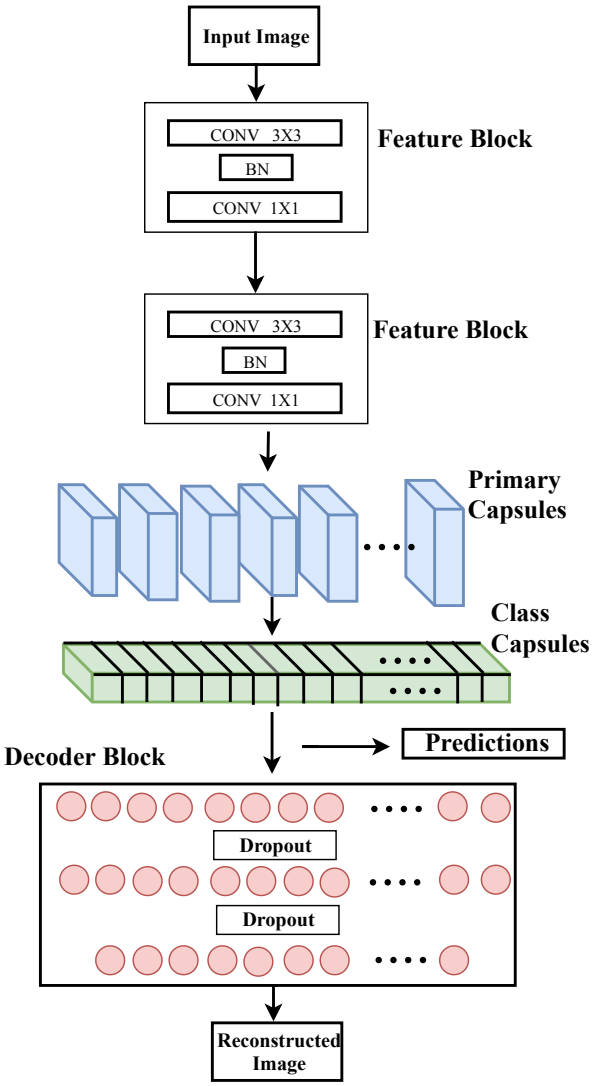


Fig. 1: An architectural overview of DeepFeat-Caps network. It is consisting of three different sub-blocks: Feature Block, Capsule blocks (Primary Capsules and Class Capsules) and Decoder Block. The class capsules are used to predict the classes (as an example: each character in MNIST dataset is a class).

III. Proposed Method

This section provides a detailed explanation of *DeepFeat-Caps*. An architecture diagram of *DeepFeat-Caps* is given in the Figure1. Proposed framework consists of three different sub-blocks: Feature Block, Capsule blocks(Primary Capsules and Class Capsules) and Decoder Block. Each of these blocks are explained below in this section.

A. Feature Blocks

Strength of DeepFeat-Caps lies in the feature generation module present before the primary capsules layer. The framework is developed based on the intuition, that the deeper and more comprehensive features are subjected to learn better routing between the capsules during the training process. In order to reduce the computational complexity

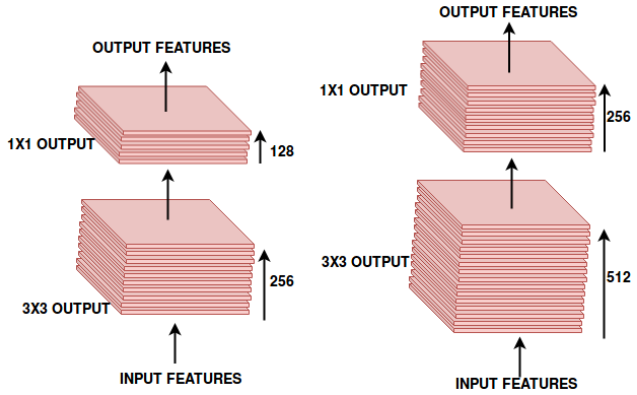


Fig. 2: Left : Feature Block Type III (FB3)
Right: Feature Block Type II (FB2)

of the capsule network, we introduce a convolutional layer within the feature block having kernel size 1×1. Similar concept is used in the inception network [20], mainly for dimensionality reduction. Four different kinds of feature blocks are used in our work, among which the proposed feature block falls under Type-III category. All the four different kinds of feature block used for the experimental purposes are explained below in this section.

1) **Feature Block Type I (FB1):** The input feature is obtained by passing successive layers of convolution. The Convolution kernel size is 9. This is same as used in the original CapsNet.
2) **Feature Block Type II (FB2):** The feature block, as shown in right part of the Fig. 2 is comprised of alternate convolutional layers with 512 filters having kernel size 3×3 and 256 filters with kernel size 1×1 respectively.
3) **Feature Block Type III (FB3):** The FB3, as shown in left part of the Fig. 2, is similar to type FB2, except that it has convolution kernels 3×3 and 1×1 with 256 and 128 filters respectively.
4) **Feature Block Type IV (FB4):** FB4 is shown in the Fig. 3. The stacking of the convolution layers in this feature block is done similar to inception version 1.0, except the max pooling branch as described in [20]. In this feature block, parallel convolution operation with kernel sizes 5X5, 3X3 and 1X1 is applied followed by concatenation of these features. The concatenated features are then applied to the next feature block.

B. Capsule Blocks

Architecturally, this block is same as that used in CapNet[16], except some modifications in the parameters, discussed later in this section. It consists of two sub blocks: primary capsule layer and digit capsule layer. In this paper, digit capsule layer of CapsNet is renamed as class capsule for better generalization as the main purpose of the last block is to predict a class.

Primary Capsules: This block performs convolution and bundles input features into fixed number of capsule. In

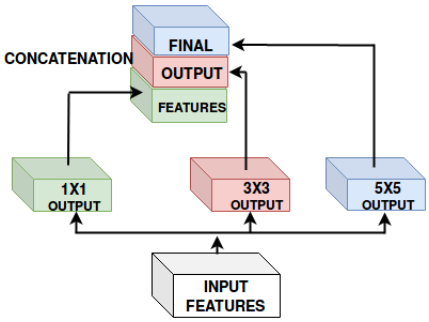


Fig. 3: Feature Block Type IV (FB4)

this block we have made few modifications in the number of capsule as well as in the dimension of each capsule. In CapsNet[16], a combination of number of capsule and dimension (32, 8) is used. Since, the primary capsule defines the representation of initial features in the form of vectors, which are trained to learn the geometric transformations in the routing by agreement training process, it has been found that deeper convolution feature helps in lowering the number of capsules for optimum learning. The choice of deeper features, ease the requirement of large number of initial capsules which leads to the drastic decrease in the total number of parameters. With deeper input feature, the performance is retained while decreasing the primary capsules shape from (32, 8) to (20,16) and finally to (20,8). Moreover, these combinations have reduced the number of parameters up to 40% from original CapsNet presented in [16].

Class Capsules: This block is the final layer of DeepFeatCaps, which is kept similar to that present in CapsNet[16]. The number of capsules in this block is equal to the total number of classes present in the network. Each capsule in this block is responsible for the prediction of a single class and gets activated only for those primary block capsules which together agrees for the required class during training by dynamic routing method. The dimension of each capsule in this layer is kept as 16 (which is same as used in [16]). Each capsule in this block gives an 1-D array output of length 16, i.e, a scalar outcome for each dimension.

C. Decoder block

The main objective of this block is to reconstruct an image and use it to find the mean squared error by comparing it with the input image. The reconstruction is done by taking input from the class capsule block which is then passed through 3 fully connected (FC) layers. The first two layers of the decoder block has 1024 and 512 nodes with Relu activation function. However, the last FC layer consists of nodes equal to the total pixels in the input image. The sigmoid activation function is used for the last FC layer and the output is reshaped to input image dimension.

D. Training using Dynamic routing

Each capsule is comprised of many neurons and deals with data in vector form. The reason as why the capsule is able to preserve spatial relationships between features is it's ability to forward pass information between capsules using dynamic routing. As the output of each capsule layer is in vector form, use of a sigmoid, softmax or other activation function doesn't work. Hence, a non-linear activation function called *squashing* is introduced in [16]. For an input vector s_j to the capsule j, the output vector v_j is given as,

$$v_j = \frac{\|s_j\|^2}{(1+\|s_j\|^2)} \frac{s_j}{\|s_j\|} \tag{1}$$

where, s_j denotes input vector to the capsule j in the class block. The output vector v_j decides the probability of that special feature learned by the capsule. Considering that the i_{th} capsule of the primary capsule block is connected to the j_{th} capsule in the class capsule block, s_j is given as

$$s_j = \sum_i c_{ij} \hat{u}_{j|i}, \qquad \hat{u}_{j|i} = w_{ij} u_i \tag{2}$$

where, w_{ij} is the weight matrix that introduces geometric transformations between capsules of the two blocks and u_i is the input vector of the i_{th} capsule from the primary capsule block. c_{ij} is the coupling coefficient which is determined by the iterative dynamic routing algorithm [16].

The coupling coefficient c_{ij} is a routing softmax value obtained from all the originating link value of this capsule i to all the capsules in the class capsule block. The originating link values kept on updating with each iteration during the dynamic routing process. The total number of iterations in the routing process is used as 2, which is less than that is used in [16]. The information in the primary capsule passes to the class capsule only for those features, where both contributed to the final prediction of the class in the past, otherwise it does not pass. This is known as routing by agreement which has been discussed in [16].

E. Loss calculation

Two different types of losses are used during the training process: Margin loss and reconstruction or decoder loss. Both of these losses are described below in this section.

Margin loss:

Margin loss is given as

$$L_k = T_k max(0, m^+ - \|v\|)^2 + \lambda(1-T_k) max(0, \|v\| - m^-)^2 \tag{3}$$

where, $T_k = \begin{cases} 1, & \text{For class k} \\ 0, & \text{For other classes} \end{cases}$

$m^+ = 0.9$, $m^- = 0.1$, $\lambda = 0.5$ and $\|v\|$ is output of the k^{th} capsule.

The parameter λ is used to decrease the weightage of other classes during margin loss calculation. The total loss is calculated by the sum of all the class capsules.

Reconstruction or Decoder Loss:

We mask out all, but the activity vector of the correct class capsule during training process. The reconstruction loss is mean squared error of the pixel-wise difference between

TABLE I: An ablation study for performance (both in terms of recognition accuracy and number of parameter used) analysis of the DeepFeat-Caps when different types of feature blocks. In all these cases the DeepFeat-Caps architecture with primary capsule block parameters having 20 capsules and dimension of each as 16 are used.

SI No.	Network Name	Feature Block Type	Number of Feature Blocks	Accuracy				Total Parameters	
				Object Dataset		Character Dataset		RGB Image	Gray Image
				CIFAR-10	FMNIST	SVHN	MNIST		
1	CapsNet	-	0	0.676	0.912	0.932	0.996	11,749,120	8,215,568
2	DeepFeat-Caps	I	2	0.618	0.908	0.936	0.995	12,406,272	9,548,816
3	DeepFeat-Caps	II	4	0.745	0.924	0.961	0.996	18,154,304	14,575,696
4			3	0.784	0.926	0.962	0.996	17,610,816	13,827,408
5			2	0.805	0.932	0.963	0.997	17,169,728	13,181,520
6	DeepFeat-Caps	III	4	**0.767**	**0.922**	**0.961**	**0.996**	**10,899,264**	**7,322,960**
7			3	**0.786**	**0.925**	**0.961**	**0.996**	**11,339,200**	**7,558,096**
8			2	**0.815**	**0.927**	**0.958**	**0.996**	**11,881,536**	**7,895,632**
9	DeepFeat-Caps	IV	1	0.770	0.930	0.934	0.994	16,070,464	10,853,584

TABLE II: An ablation study to demonstrate the network parameter variation with different shape of initial capsules. The feature block type FB3 is found to be outperforming all other configurations as shown in Table I.

SI No.	Network Name	Feature Block Type	Number of Feature Blocks	Accuracy				Total Parameters	
				Object Dataset		Character Dataset		RGB Image	Gray Image
				CIFAR-10	FMNIST	SVHN	MNIST		
Primarycapsule Block Parameters: No. of Capsule = 20 ; Dimension = 16									
1	DeepFeat-Caps	III	4	0.767	0.922	0.961	0.996	10,899,264	7,322,960
2			3	0.786	0.925	0.961	0.996	11,339,200	7,558,096
3			2	0.815	0.927	0.958	0.996	11,881,536	7,895,632
Primarycapsule Block Parameters: No. of Capsule = 20 ; Dimension = 8									
4	DeepFeat-Caps	III	4	0.765	0.923	0.96	0.996	7,985,824	5,023,920
5			3	0.79	0.925	0.959	0.996	8,041,760	4,977,456
6			2	**0.807**	**0.93**	**0.96**	**0.996**	**8,148,896**	**4,982,192**
Primarycapsule Block Parameters: No. of Capsule = 32 ; Dimension = 8									
7	DeepFeat-Caps	III	4	0.774	0.923	0.959	0.996	9,733,888	6,403,344
8			3	0.796	0.927	0.963	0.996	10,020,224	6,525,840
9			2	0.803	0.926	0.959	0.996	10,388,480	6,730,256

the reconstructed image and the input image and provides necessary regularization during training.

In order to train the network, the loss minimization is done using a weighted sum of margin loss and decoder loss. Lower weightage is given to the reconstruction loss in shaping the learning with a value of 0.392 against unit weightage to margin loss.

IV. Experimental Results and Discussions

This section demonstrates experimental results of the proposed deep feature based capsule framework and also performs various ablation studies to validate the proficiency of the DeepFeat-Caps network. Datasets of two different categories are used in our experiments. These include, object image datasets (FMNIST and CIFAR 10) and character image datasets (MNIST and SVHN). These datasets have different formats (RGB and Gray Scale) and have sufficient variations in order to be able to generalize the proposed architecture. The detailed description for each of these datasets are given below:

Object image dataset

1) **FMNIST dataset[23]:** This dataset has gray-scale images of size 28 X 28 pixels for 10 different types of fashion accessories ranging from T-shirt to Ankle-boots. This dataset has 60,000 training images and 10,000 testing images.

2) **CIFAR 10 dataset [7]:** This dataset contains 10 real world object images of animals (bird, cat, deer, dog, frog, and horse) and vehicles (airplane, automobile, ship, and truck). The image format is RGB with size 32 X 32 Pixels. It consists of 50,000 training and 10,000 testing images.

Character image dataset

1) **MNIST dataset[9]:** This is a standard dataset for handwritten digits(0-9) and contains gray-scale image of size 28 X 28 pixels. There are 60,000 images for training and 10,000 images for testing in this dataset.

2) **SVHN dataset[14]:** SVHN is a big dataset for digits(0-9), and consists of images taken from Google street view. The dataset poses challenge of recognizing digits in natural scene image. The number of training images in this dataset is 73,257 while for testing it has 26,032 images. The image is in RGB format as evident for a natural scene image and has a size of 32 X 32 pixels.

TABLE III: Effect of performances in the capsule network for variations in the sizes of kernel used in convolution layer.

Network Architecture	FMNIST Accuracy	FMNIST Parameters	SVHN Accuracy	SVHN Parameters
CapsNet	0.912	8,215,568	0.932	11,749,120
Network 1	0.92	10,040,336	0.919	14,028,544
Network 2	0.908	9,548,816	0.936	12,406,272
Network 3	**0.92**	9,637,904	**0.945**	13,459,968

Analysis about the network parameters is done in two categories which is RGB Image and Gray Image. RGB Image category represents 32X32 image with 3 channels as in SVHN and CIFAR-10 dataset. Similarly, Gray Image category represents image of 28X28 resolution with 1 channel as in FMNIST and MNIST dataset.

In the following section ablation study is performed for selection of lower kernel size, feature block architecture and optimum dimension of primary capsule layer.

A. Experimental results obtained by varying kernel size of convolutional layers

Following variations are made in the convolution kernels within the CapsNet as well as addition of extra convolution layers with 3 or 9 kernel size.

1) Network 1 : Kernel size 9 in CapsNet is replaced with kernel size 3.
2) Network 2 : 1 extra convolution of Kernel size 9 is added in CapsNet architecture. The number of filters used is 128.
3) Network 3 : Kernel size 3 in-place of 9, is used in Network 2.

It can be observed from the results as shown in the Table III, that if more convolutional layer with kernel size 3 is stacked, then the performance is improved. However, this also results in increase of total number of parameters. The reason for increase in parameter is not due to the parameters of convolution layer, rather the parameters used in the class capsule layer due to higher dimension output(receptive field) after a lower kernel size convolution operation. This has motivated us to design a deeper architecture with 3X3 convolution, so that receptive field is reduced before capsule block. Hence, we come up with the DeepFeat-Caps architecture.

B. Ablation Study for Selection of Feature Blocks

The Table I gives a comparative analysis of performance for different types of feature blocks when evaluated on the selected datasets. Both recognition accuracy and well as total number of parameters are considered while making a comparative study. It can be observed from the analysis that, the proposed deep feature framework using type-III feature block outperforms all other frameworks, both in terms of recognition accuracy and number of parameters used. The observations show that best performance is obtained using proposed deep features when the dimension of capsules in the primary block is set to 8. It must be noted that, the number of capsules in the primary capsule block is set to 20 for the ablation study to select best feature block as depicted in Figure I. Based on studies performed as shown in Table I and Table II, we can conclude that, the proposed DeepFeat-Caps network is significantly better than CapsNet, both in terms of recognition accuracy as well as number of parameters used. From this ablation study, the best architecture we propose here is DeepFeat-Caps network of 2 FB3 feature blocks.
Figure 4 shows a performance comparison (in terms of precision and recall) between the DeepFeat-Caps and CapsNet for 10 different classes of images for object image dataset and character image dataset. Again, the DeepFeat-Caps outperforms the original CapsNet.

TABLE IV: Performance comparison for optimum number of routing. Table shows Routing results in Proposed DeepFeat-Caps (FB Type: III, No of FB: 2, Primary Caps:20, Dimension:8).

SI No.	Number of Routing	Dataset Performance(Accuracy)			
		CIFAR10	FMNIST	SVHN	MNIST
1	2	0.803	0.93	0.956	0.996
2	3	0.807	0.93	0.96	0.996
3	4	0.809	0.932	0.953	0.996
4	5	0.818	0.932	0.956	0.996

To compress the network, dimension of the capsules in the primary capsule block is reduced. Table II shows this ablation study, where performance is tabulate with change in capsule dimension. Here, it has been shown that the parameter is reduced for a RGB image from 11,749,120 to 8,148,896 while for gray scale image it is reduced from 8,215,568 to 4,982,192. So, DeepFeat-Caps architecture is compressed to 69% and 60% of CapsNet architecture for RGB and Gray scale images respectively. It is to note here that in Table II, row 5,6,7 the number of parameter is decreased with increase in feature blocks, the reason for this is, the most number of parameter is used in the capsule block in comparison with feature blocks. Since, increasing feature block, the receptive fields decreases as a result parameters in the capsule layer decreases accordingly. Architecture with FB3, for 3 and 4 Feature Blocks parameter is low still the performance is better with 2 Feature Block so we choose it as our proposed network architecture.

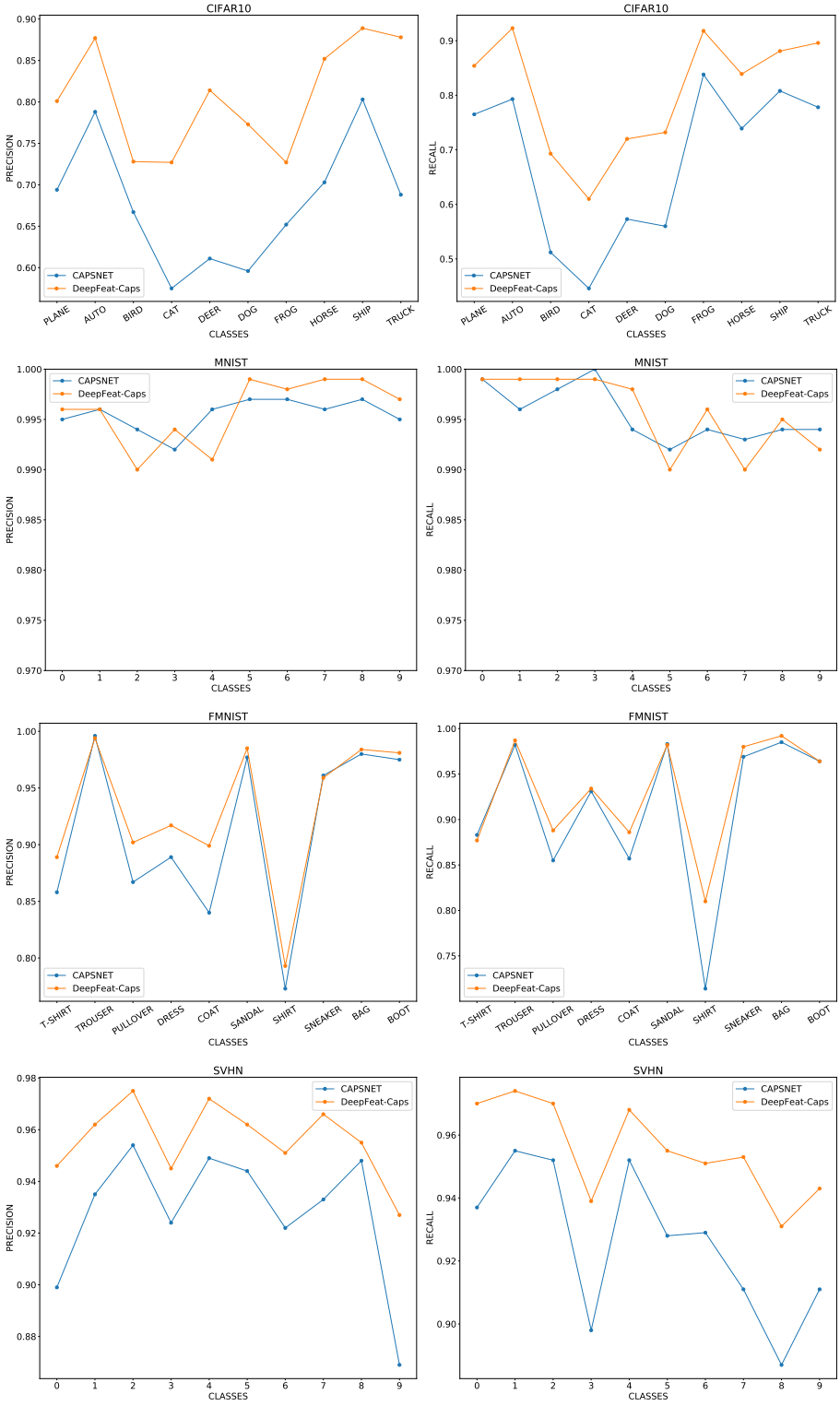
C. Performance analysis after applying Batch Normalization

Table V shows the results obtained after using batch normalization layer in the DeepFeat-Caps architecture of 2 FB3 blocks (which is also chosen as the best architecture based on the analysis made in the Table II). Results clearly show that use of a Batch Normalization layer allows the Capsule Network to train faster and also ensures increase in recognition accuracy.

D. Hyper-parameter tuning for DeepFeat-Caps

We have performed various hyper-parameter tuning for the training of DeepFeat-Caps. Firstly, the performances are checked for different routing values (2,3,4 and 5) for the dynamic routing process. It is observed that changes in performance are negligible for the routing number variations.

Fig. 4: A comparative analysis of the DeepFeat-Caps over the state-of-the-art CapsNet. The top row in the Figure shows class-wise precision and recall comparison for object dataset(CIFAR-10[7], FMNIST[23]) and character dataset(MNIST[9], SVHN[14]).



However, unlike [16] where 3 routing is used, similar performance is achieved with routing value 2 as shown in Table IV. We have thus set the value as 2 for our experiments. Secondly, we have varied the weightage of the regularization parameter(reconstruction loss) in total loss. The variations are made within the range of 0.2 to 0.5. It has been observed,

TABLE V: Performance comparison after using Batch Normalization

SI No.	Network Name	SVHN		MNIST		CIFAR-10		FMNIST	
		Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch
1	CapsNet	0.932	50	0.996	27	0.676	61	0.912	50
2	DeepFeat-Caps	0.96	31	0.996	14	0.807	39	0.93	28
3	**DeepFeat-Caps With BN**	**0.96**	**30**	**0.996**	**13**	**0.830**	**30**	**0.928**	**21**

that the best performance is obtained when it is set to 0.392. To perform above experiments the hardware used is DELL Rack Server 7920 with 192 GB RAM along with 3 NVIDIA P6000 GPU each having 24GB Memory. Training for all the datasets is done with a batch size of 100 images.

V. Conclusions

We propose a deep framework for capsules, named *DeepFeat-Caps*, that can potentially outperform the state-of-the-art CapsNet when validated using two different kinds of datasets containing object images and character images. Four different widely used datasets, such as CIFAR-10, FMNIST, MNIST and SVHN are used for performance evaluation purpose. The ablation studies performed on three different types of feature blocks with different combinations of convolutional layer, consisting of 3×3 kernels, followed by batch normalization and convolution with a 1×1 kernel. It has been demonstrated that the feature block FB3 with 256 convolutional filters having kernel size 3×3 and 128 convolutional filters with kernel size 1×1 makes the best architecture for the capsule based recognition model. Moreover, we have also performed another ablation study, later in this work by reducing the number of capsules and their dimensions in the primary capsule block, only to demonstrate that, the total number of network parameters for the proposed DeepFeat-Capsule can be scaled downed to 60% of the total parameters used in the state-of-the-art CapsNet, while ensuring a significant increase in recognition accuracy.

References

[1] M. T. Bahadori. Spectral capsule networks. 2018.

[2] K. Duarte, Y. Rawat, and M. Shah. Videocapsulenet: A simplified network for action detection. In *Advances in Neural Information Processing Systems*, pages 7621–7630, 2018.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[5] G. E. Hinton, S. Sabour, and N. Frosst. Matrix capsules with em routing. 2018.

[6] A. Jaiswal, W. AbdAlmageed, Y. Wu, and P. Natarajan. Capsulegan: Generative adversarial capsule network. In *European Conference on Computer Vision*, pages 526–535. Springer, 2018.

[7] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[10] Y. LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[11] W. Liu, E. Barsoum, and J. D. Owens. Object localization and motion transfer learning with capsules. *arXiv preprint arXiv:1805.07706*, 2018.

[12] D. Ma and X. Wu. Tcdcaps: Visual tracking via cascaded dense capsules. *arXiv preprint arXiv:1902.10054*, 2019.

[13] J. O. Neill. Siamese capsule networks. *arXiv preprint arXiv:1805.07242*, 2018.

[14] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[15] D. Rawlinson, A. Ahmed, and G. Kowadlo. Sparse unsupervised capsules generalize better. *arXiv preprint arXiv:1804.06094*, 2018.

[16] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[17] S. Sabour, N. Frosst, and G. E. Hinton. Matrix capsules with em routing. In *6th International Conference on Learning Representations, ICLR*, 2018.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] C. K. Singh, V. K. Gangwar, H. V. Singh, K. Narain, A. Majumder, and S. Kumar. Deep capsule network based automatic batch code identification pipeline for a real-life industrial application. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2019.

[20] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[21] E. Xi, S. Bing, and Y. Jin. Capsule network performance on complex data. *arXiv preprint arXiv:1712.03480*, 2017.

[22] C. Xiang, L. Zhang, Y. Tang, W. Zou, and C. Xu. Ms-capsnet: A novel multi-scale capsule network. *IEEE Signal Processing Letters*, 25(12):1850–1854, 2018.

[23] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.