MixCaps: Capsules With Iteration Free Routing

Ifty Mohammad Rezwan*, Mirza Belal Ahmed†, Shazzad Sakim Sourav‡,
Ezab Quader§, Arafat Hossain¶, Nabeel Mohammed‖
Department of Electrical and Computer Engineering,
North South University,
Dhaka, Bangladesh
Email: *mohammad.rezwan@northsouth.edu, †mirza.ahmed@northsouth.edu, ‡shazzad.sourav@northsouth.edu,
§ezab.quader@northsouth.edu, ¶hossain.arafat@northsouth.edu, ‖nabeel.mohammed@northsouth.edu

***Abstract*—In this paper, we propose a new variant of Capsule Networks called MixCaps. It is a new architecture that significantly decreases the compute capability required to run capsule networks. Due to the nature of our modules, we propose a new routing algorithm that does not require multiple iterations. All routing models prior to this architecture uses multiple iterations. This decreases our model's memory requirements by a significant margin unlike previous methods. This also provides us with the advantage to use both Matrix and Vector Poses. The model learns better complex representations as an aftereffect. Despite all this, we also show that our model performs on par with all prior capsule architectures on complex datasets such as Cifar-10 and Cifar-100.**

***Index Terms*—capsule neural networks, routing, deep learning, convolutional neural networks, image classification, computer vision.**

I. Introduction

The advent of convolutional neural networks(CNNs) [1] [2] [3] has revolutionized the field of computer vision. Although CNNs are robust feature extractors, they lack transformation equivariance due to the use of pooling layers. As we go deeper within CNNs [4], the extracted features contain more information as the receptive field keeps getting larger. So, pooling from these features might lead to major information loss. To alleviate this, capsules were proposed as a solution. Capsules are basically the concept of grouping a number of neurons. They were first proposed in Transforming Auto-Encoders [5]. Convolutional Capsule layers are a mixture of convolution layers followed by capsule layers. They have gained much attraction ever since Sabour et al. [6] proposed Dynamic Routing.

Routing is an integral part used in all capsule architectures. It is the operation to calculate higher order capsules from lower order capsules, through the use of an agreement function. This agreement might be addressed from the perspective of the lower order capsules to higher order capsules or vice-versa. Due to the nature of this routing operation, it has been designed in multiple ways by different authors.

In this paper, we propose a routing algorithm along with new modules. At first, we tackle the agreement function from the perspective of the lower capsules. This creates robust higher level capsules which guide the lower level capsules. We achieve this by removing the local iteration and creating a mechanism where all the capsules are providing global context to each other in a single feed-forward operation. As our routing scheme proves to be quite inexpensive, we use both Vector and Matrix poses in our architectures leading to very robust capsules with complex learning in the Final Capsule Layer.

II. Related Works

Transforming Auto-Encoders [5] was the first model to tackle the shortcomings of CNNs. Transforming Auto-encoders could calculate the initial pose parameters of an object or object parts, if it was given a transformation matrix along with a affinely translated image pair of the object. Transforming Auto-Encoders provide the idea behind converting simple neurons to capsules. But the concept of learning higher order capsules from lower order capsules still remained.

So, building upon this work, Sabour proposed Dynamic Routing and Vector Capsules. In this work, higher level instantiation parameters are learned when lower level capsules agree on predictions through transformation matrices. This agreement is calculated by the size of output vector of higher level capsules compared to the prediction of the lower level capsule. Edgar et al. [7] discusses the various limitations of capsule networks on complex data. Much work [8], [9], [10], [11] has been done which tried to tackle this specific method of routing as an optimization problem by maximizing the clustering relation between lower level capsules. This helps achieve better higher level capsules.

Hinton et al. [8] first proposed the use of matrix pose capsules. The matrix pose capsules were matrices said to capture the change of viewpoint of an object. This proved to be true as the authors achieved impressive results on a difficult shape changing task. Their routing scheme clusters the capsules into a Gaussian distribution through the use of Expectation Maximization Clustering. Fabio et al. [11] tries to address the routing problem from a bayesian perspective by achieving the fitting of Gaussians. They achieved an admirable result on the shape task.

DenseCaps [12] were the first to combine state of the art backbones called DenseNets [13] with capsule nets. This proved that capsules worked better with higher level features. This paper [14] on the other hand showed that even with better higher level features, we need an improved routing algorithm. Proposed uniform routing is a standard feed forward iteration free routing which is quite similar to ours. So to alleviate this,

2020 Digital Image Computing: Techniques and Applications (DICTA) | 978-1-7281-9108-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/DICTA51227.2020.9363386

Rajasegaran et al. [15] proposed DeepCaps where they stacked capsules and went deeper through the use of 3D convolutions to facilitate skip connections between layers. Also, Choi et al. [16] learned the routing mechanism between lower and higher order capsules through the use of an attention architecture. But due to the lack of an agreement function, their accuracy fell short. And finally Tsai et al. [17] proposed dot inverted routing where they turn the routing process upside down so that, the higher level capsules calculate the highest agreement to choose their lower level capsules unlike the previous methods. This is facilitated through the use of an inverted dot product mechanism. They also propose the concurrent routing mechanism which decreases the computation of routing through parallel computation on multiple capsule layers.

Our work extends from [17] [16] [14] where we introduce an iteration free routing. Instead of using an attention mechanism, we adopted a matrix multiplication mechanism in multiple steps. This mechanism lets us calculate the agreement function easily by learning from a transformation matrix that resembles the capsules of the next layer. We isolate the routing co-efficients through a softmax function as before from the learned agreement function. After this we perform another matrix multiplication with a transformation matrix that learns the capsules of the next layer. This simple set of operations proved to be robust as this mechanism helps us achieve around the same accuracy and in some cases more than the mechanisms introduced before with less parameters and a fraction of the memory.

III. Contribution

- We introduce a simple routing algorithm, that doesn't require multiple iterations thereby significantly decreasing memory requirements by a significant margin.
- We introduce a Convolutional Capsule layer named $Single\ Matrix$. It uses simple matrix multiplication for transformation matrices. From the simple matrix multiplication mechanism, we also derived two modified $Fully\ Connected\ Capsule\ Layers$, one for Matrix Poses and one for Vector Poses. As these layers are very simple in structure, they help in decreasing parameters and memory significantly.
- Prior capsule architectures either use Vector or Matrix poses in their models. We are the first to introduce an architecture that use both Matrix and Vector poses. We also show that this greatly complements the Final Concatenated Capsule. This process helps it learn better and complex representations.
- We introduce Group Normalization as normalization for some layers and in other cases we use Layer Normalization as normalization.

IV. Proposed Methodology

Our proposed architecture consists of multiple modules. Among these modules reside the *Convolutional Backbone*, *Primary Capsule Layer*, *Convolutional Capsule Layer*, *Fully Connected Capsule Layer*, and *Post Capsule Layer*. In the following section we will be explaining all our modules in detail. Table I contains prior necessary notations which are required to explains for our equations.

TABLE I
Notation Table

Symbol	Description
$\mathbf{O}^{V_{GN}}$	Activation of Vector Pose
$\mathbf{O}^{M_{LN}}$	Activation of Matrix Pose
$\mathbf{C}^{L}$	Capsule Matrix *C* of Layer *L*
h	Height of feature window
w	Width of feature window
N	Number of capsules
a	height of package window
b	width of package window
x	height of convolving window
y	width of convolving window
$\mathbf{P}_V$	dimension of each vector pose capsule
$\mathbf{P}_M$	dimension of each matrix pose capsule
L	Present Layer

A. Convolutional Backbone

At first, we used a convolutional backbone to extract features from the image. One of them was a standard feed-forward convolutional neural network. The other one was a small variant of the *ResNet Model* proposed by Kaiming [4]. The second model was used to show that our proposed model scales well when introduced with a better feature extracting backbone.

1) Standard Feed-Forward CNN Backbone: Our Standard CNN consists of 256 learn-able filters of (3×3) kernel size. This is followed by an activation function called ReLU [18] to introduce non-linearity.

2) ResNet Backbone: Our ResNet Backbone consists of three blocks of 64 learn-able filters of (3×3) kernel size. This is followed by four blocks of 256 filters of (3×3) kernel size. Identity mappings are introduced between layers to retain information. Layer activations are ReLU.

B. Primary Capsule Layer

After the image is passed through the *Convolutional Backbone*, we are greeted to an output of a number of feature maps with a specific kernel size. At first, we performed another convolution of $(\mathbf{N}^{L+1} \times \mathbf{P}_V^{L+1})$ filters with (1×1) kernel size on the feature map we achieved earlier. This is performed to achieve the necessary feature maps for matrix and vector pose matrices. We reshaped the feature maps along the spatial locations to achieve the necessary capsules. This resulted in two matrices.

The shapes of the matrices are $(B, \mathbf{N}^{L+1}, h, w, \mathbf{P}_V^{L+1})$ for the vector pose and $(B, \mathbf{N}^{L+1}, h, w, \mathbf{P}_M^{L+1})$ as the matrix pose. Here, B denotes the batch size.

For convenience, an intuitive difference between the vector pose and matrix pose is as follows. The vector pose matrix is a $1D$ matrix that learns higher level part variations of the whole object whereas the matrix pose being a $2D$ matrix learns much harder narrow parts that might represent the object. Now if,

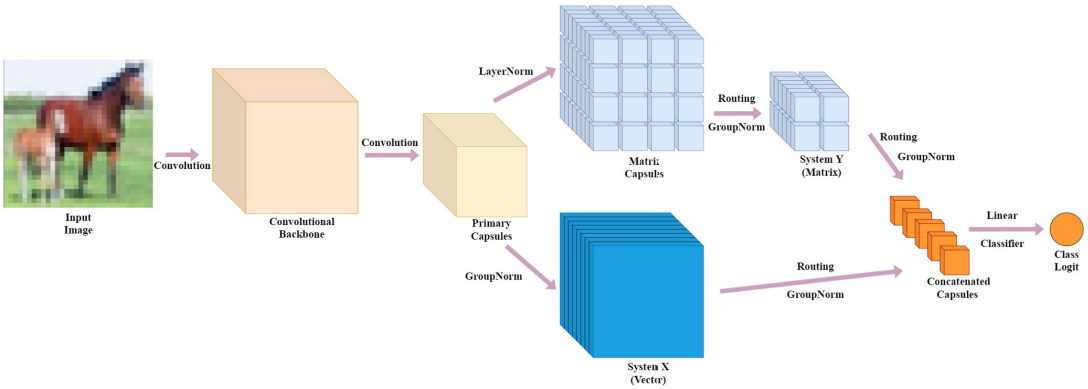

Fig. 1. MixCaps Architecture

$\mathbf{P}_V^L \in \mathbb{R}^{d_L}$ is the dimension of a vector pose of a certain layer L, the dimension of the matrix pose of the layer L would be $\mathbf{P}_M^L \in \mathbb{R}^{\sqrt{d_L} \times \sqrt{d_L}}$.

C. Primary Capsule Normalization

The two pose matrices we achieved from the primary capsule are then introduced to a separate normalization function. These normalizations introduce scaling to the network. Layer Normalization [19] is applied to the matrix pose $\mathbf{C}_M$ along the dimension $\mathbf{P}_M$. Group Normalization [20] is applied to the vector pose $\mathbf{C}_V$ along the dimension $\mathbf{P}_V$.

The hyper-parameters in these cases, are the dimensions of the capsules coming in. These are $\mathbf{P}_M$ and $\mathbf{P}_V$ for Layer Normalization and Group Normalization sequentially. Also, in case of Group Normalization, an extra hyper-parameter is the number of groups we want to perform it on.In all cases we define Group, $\mathbf{G} = 2$. Considering all these, the equations for the above operations are as follows,

$$\mathbf{O}^{M_{LN}} = \text{LayerNorm}\left(\mathbf{C}^{V}_{(:,:,:,\mathbf{P}_M)}\right) \tag{1}$$

$$\mathbf{O}^{V_{GN}} = \text{GroupNorm}\left(\mathbf{C}^{M}_{(:,:,:,\mathbf{P}_V)}\right) \tag{2}$$

Layer Normalization normalizes all capsule dimensions together. Group Normalization on the other hand normalizes capsule dimensions based on the number of groups. This normalization is done based on their mean and standard deviation and scales the capsules dimensions to a better representation.

D. Convolutional Capsule Layer(SingleMatrixCaps)

The Convolutional Capsule Layer takes in the output of the Matrix pose activation $\mathbf{O}^{M_{LN}}$. The Convolutional Capsule Layer is a simple elementwise matrix multiplication layer that performs a few transformations with the convenient use of weight matrices. This is followed by normalizations which in this case can be considered as activations.

So, now if we denote the incoming capsules in this layer L as $\mathbf{C}^{L-1}_{\mathbf{N}^{L-1},h,w,\mathbf{P}_M^{L-1}}$. $\mathbf{P}_M^{L-1}$ will denote the dimension of C^{L-1}. As it is a matrix pose, it would be in the form of $p \times p$ where $p \in \mathbb{R}$.After this, we unfolded the input to $\mathbf{C}^{L-1}_{\mathbf{N}^{L-1},x,y,a,b,\mathbf{P}_M^{L-1}}$. This was done to facilitate the matrix multiplication through which the convolving window will convolve over the local package window. An overview of the operation is as follows,

$$\mathbf{C}^{L-1}_{(\mathbf{N}^{L-1},x,y,a,b,\mathbf{P}_M^{L-1})} = \text{Reshape}\left(\mathbf{C}^{L-1}_{(\mathbf{N}^{L-1},h,w,\mathbf{P}_M^{L-1})}\right) \tag{3}$$

We then performed a matrix multiplication between $\mathbf{C}^{L-1}$ and a transformation weight matrix W^1. This produced $R^{(L-1,L),M}$. The operation is as follows,

$$\mathbf{R}^{(L-1,L),M}_{(\mathbf{N}^{L-1},x,y,\mathbf{N}^L)} = \sum_{\text{i=1}}^{\text{a}} \sum_{\text{j=1}}^{\text{b}} \sum_{\mathbf{k}=1}^{\text{L}-1} \left(\left(\mathbf{C}^{L-1}_{(\mathbf{N}^{L-1},x,y,i,j,\mathbf{P}_M^k)} \right) \times \left(\mathbf{W}^1_{(i,j,\mathbf{P}_M^k,\mathbf{N}^L)} \right) \right) \tag{4}$$

One can intuitively think that the model learns an initial agreement of sorts from the random matrix, in this case W^1 which represents the next layer's capsules as the training progresses. This agreement, in this case $\mathbf{R}$ is later strengthened and initialized with the next operation.

After this, we performed a softmax operation on $\mathbf{R}^{(L-1,L),M}$ across the dimension that addressed the number of capsules in the next layer, $\mathbf{N}^L$. This operation is performed to contain the agreement between outgoing and incoming capsules from the perspective of the capsules in the next layer. One can think of the matrix from the output of the softmax operation, $\mathbf{R_S}$ as an initialized agreement matrix between the incoming and outgoing capsules. The operation is as follows,

$$\mathbf{R}^{(L-1,L),M}_{\mathbf{S}(\mathbf{N}^{L-1},x,y,\mathbf{N}^L)} = \text{Softmax}\left(\mathbf{R}^{(L-1,L)}_{(:,:,:,\mathbf{N}^L)}\right) \tag{5}$$

After this, we perform a transpose to the $\mathbf{R_S}$ function to convert it to $\mathbf{R}_{\mathbf{S}}^{(L,L-1),M}$. Following this, we performed a second matrix multiplication with $\mathbf{R}_{\mathbf{S}}^{(L,L-1),M}$ and another transformation matrix which we denote as W^2. This produced C^L. The operation is as follows,

$$\mathbf{C}^{L}_{(\mathbf{N}^{L},a,b,\mathbf{P}_M^{L})} = \sum_{\text{i=1}}^{\text{x}} \sum_{\text{j=1}}^{\text{y}} \sum_{\text{k=1}}^{\text{L}-1} \left(\left(\mathbf{R}^{(L,L-1),M}_{\mathbf{S}(\mathbf{N}^{L},i,j,\mathbf{N}^k)} \right) \times \left(\mathbf{W}^2_{(i,j,\mathbf{N}^k,a,b,\mathbf{P}_M^L)} \right) \right) \tag{6}$$

From the last operation, we can hypothesize that the neural network is learning the capsules of the next layer $\mathbf{C}^L$ through the agreement function $\mathbf{R_S}$ and a transformation matrix W^2 based on which pose matrices of the last layer it must focus on. W^2 also achieves a better representation as the training progresses. This is the underlying ideology behind our convolutional capsule layer.

1) Convolutional Capsule Activation Layer: The Matrix capsule layers are sent to this layer for scaling after the Convolutional Capsule layers. The activation function is a Group Normalization function. The hyper-parameters in this case,are the number of capsules coming in which is $\mathbf{N}_M^L$ and $\mathbf{G}$ as before. Considering all these, the equations for the above operation is as follows,

$$\mathbf{C}^L_{(\mathbf{N}^L,a,b,\mathbf{P}^L_M)} = \text{GroupNorm}\left(\mathbf{C}^M_{(:,:,:,\mathbf{P}^L_M)}\right) \tag{7}$$

Performing Group Normalization across the number of capsules is built upon the intuition that this operation helps scale similar capsules beside each other by learning the mean and standard deviation between themselves by their features. Therein, providing much more robust capsules as they will be normalized only across similar features. This might technically provide a better distribution of data points for the neural network and can be thought of as a forced prior. But this methodology worked quite robustly for us. Bear in mind, the explanation behind this operation is still a hypothesis and remains to be tested upon.

E. Fully Connected Capsule Layer

Our Fully Connected Layer is quite similar to our Convolutional Capsule Layer. So, there is an impressive decrease in parameters, memory and complexity compared to previously proposed fully connected capsule layers.

We introduce two variants of our Fully Connected Capsule Layer. One of them is used exactly after the Primary Capsule Layer and the other for the Convolutional Capsule Layer. We will call them $SystemX$ and $SystemY$. $SystemX$ is primarily used to handle Vector Poses and $SystemY$, Matrix Poses. The underlying operation between both these variants remain the same. They are explained in detail in the following sections.

1) System X: System X is used exactly after the Primary Capsule Layer to calculate the class capsules in vector form. It takes in a Vector pose O^V_{LN} from the Primary Capsule Layer. We denote the incoming capsule as $\mathbf{C}^{L-1}_{\mathbf{N}^{L-1},h,w,\mathbf{P}^{L-1}_V}$. As this is a fully connected Vector Pose layer, we collapse all the capsules along with their spatial packages h & w along the $\mathbf{N}^{L-1}$ dimension. The operation is as follows,

$$C^{L-1}_{\mathbf{N}^{L-1}\times h\times w,\mathbf{P}^{L-1}_V} = \text{Collapse}\left(\mathbf{C}^{L-1}_{\mathbf{N}^{L-1},h,w,\mathbf{P}^{L-1}_V}\right) \tag{8}$$

After this, the operations are the same as the Convolutional Capsule layer after the $Reshape$ operation, with the relevant shapes being replaced with the ones in this layer to facilitate all operations. The output of this layer is, $\mathbf{C}^L_{(\mathbf{N}^L,\mathbf{P}^L_V)}$ which is sent to the Capsule Activation Layer.

2) System Y: System Y is a very similar fully connected capsule layer with a few modifications. This fully connected capsule layer is put on top of the Convolutional Capsule Layer. This layer has been modified to take in and process Matrix pose capsules. So in this case, the input is $\mathbf{C}^{L-1}_{\mathbf{N}^{L-1},h,w,\mathbf{P}^{L-1}_M}$. As this is a fully connected capsule layer, collapsing the Matrix pose provided us with a matrix shape of $\mathbf{C}^{L-1}_{\mathbf{N}^{L-1}\times h\times w,\mathbf{P}^{L-1}_M}$. After this, the operation is exactly the same as $ConvolutionalCapsule$ and $SystemX$ with it's own shapes being facilitated for the operations.

Then, we sent the output capsules $\mathbf{C}^L_{(\mathbf{N}^L,\mathbf{P}^L_V)}$ to the activation layer.

3) Fully Connected Capsule Normalization: The Fully Connected Capsule normalizations are the same as the normalization after the Convolutional Capsules. This applies to both Vector pose and Matrix pose. However, we flattened the Matrix poses together so we could concatenate them with the Vector Poses in the Post Capsule Layers. The operations are as follows,

$$\mathbf{C}^{L(M)}_{(\mathbf{N}^L,\mathbf{P}^L_{M(v)})} = \text{Flatten}\left(\mathbf{C}^L_{(:,\mathbf{P}^L_M)}\right) \tag{9}$$

$$\mathbf{C}^{L(M)}_{(\mathbf{N}^L,\mathbf{P}^L_{M(v)})} = \text{GroupNorm}\left(\mathbf{C}^M_{(:,\mathbf{P}_M)}\right) \tag{10}$$

$$\mathbf{C}^{L(V)}_{(\mathbf{N}^L,\mathbf{P}^L_V)} = \text{GroupNorm}\left(\mathbf{C}^V_{(:,\mathbf{P}_V)}\right) \tag{11}$$

The intuition and hypothesis also remains the same as Convolutional Capsule activations.

F. Post Capsule Layer

In the post capsule layer, we got our outputs from both System X and System Y which we denote as $\mathbf{S}^X_{(\mathbf{N}^L,\mathbf{P}^L_V)}$ and $\mathbf{S}^Y_{(\mathbf{N}^L,\mathbf{P}^L_{M(v)})}$. We concatenated both the matrices to get a single matrix $\mathbf{S}^F_{(\mathbf{N}^L,\mathbf{P}^L)}$. One can think of this intuitively as a concatenation of both layers which holds both Vector & Matrix pose traits. After this, we introduced a fixed linear classifier across all capsules. This provided us with the output class capsules.

$$S^F_{(\mathbf{N}^L,\mathbf{P}^L)} = \text{Concatenate}\left(\mathbf{S}^X_{(\mathbf{N}^L,\mathbf{P}^L_V)} + \mathbf{S}^Y_{(\mathbf{N}^L,\mathbf{P}^L_{M(v)})}\right) \tag{12}$$

$$FinalOutput = \text{LinearClassifier}\left(\mathbf{S}^F_{(:,\mathbf{P}^L)}\right) \tag{13}$$

1) Iteration-Free Sequential Routing: Our routing algorithm is a modified version of the routing introduced in [16] [17]. Previous routing algorithms conformed to a local connection between capsule layer to capsule layer by repeated iterations. This locally repeated iterative routing algorithms updated the capsules of the later layer by updating the transformation matrices in a local window based on the capsules of the present layer. We assume that by performing multiple iterations in a local window the previous routing algorithms are not taking every small change on what should be accepted into account. As we know, higher order poses are actually in reality, concatenated form of robust, agreeing small order poses. So, we remove this process of local multiple iterations.

The intention behind this is to bring about a global context to the transformation matrices. The hypothesis behind this, is that this leads to a fully feed forward capsule neural network where every small change in the topmost higher order capsule will lead to changes in the lowest small order capsules and vice-versa. So, in our case each epoch can be considered an iteration. We used both Matrix Pose and Vector Pose

Algorithm 1: Iteration-Free Sequential Routing

input : This Layer Poses $\mathbf{C}^{L}$, Next Layer Blueprint Poses $\mathbf{W}^{L+1}$, Link Strength Dimension Features $\mathbf{LS}^{L+1}$
output: Next Layer Pose $\mathbf{C}^{L+1}$

Agreement Between Next Layer & This Layer Pose;
$\mathbf{A}^{(L+1,L)} \leftarrow$ *Agreement Function(*$\mathbf{C}^{L}$, $\mathbf{W}^{L+1}$*)*;
Routing co-efficient which ensures maximized agreement between all incoming and outgoing capsules;
$\mathbf{R}^{(L+1,L)} \leftarrow$ *Softmax(*$\mathbf{A}^{(L+1,L)}$*)* ;
Network learns the Next Layer Capsules from a Random Next Layer Blueprint Capsule ;
$\mathbf{C}^{L+1} \leftarrow$ *Matrix Multiplication(*$\mathbf{R}^{(L+1,L)}$, $\mathbf{LS}^{L+1}$*)*;
Ensuring Specific Activation For Specific Pose;
$\mathbf{C}^{L+1} \leftarrow$ *GroupNorm(*$\mathbf{C}^{L+1}$*)* ;

Algorithm 2: Forward-Pass or Inference

input : Images $\mathbf{I}$ of Shape $h \times w \times c$
output: Class Capsules Logits for each Class

Feature Extraction from Convolutional Backbone;
$\boldsymbol{F} \leftarrow$ *Covnvolutional Backbone(*$\boldsymbol{I}$*)*;
Primary Capsule Layers, Initialization;
if $\mathbf{C}^{L+1}$ == ***Matrix Pose*** **then**
  $\mathbf{C}^{L+1} \leftarrow$ LayerNorm(Convolution($\mathbf{F}$))
else
  $\mathbf{C}^{L+1} \leftarrow$ GroupNorm(Convolution($\mathbf{F}$))
end
Capsule Layers, First to Finish(Parallely Done);
for $L \leftarrow 1^{L^M}$ **to** L^M **do**
  $\mathbf{C}^{M,(L+1)} \leftarrow$ Routing($\mathbf{C}^{L}$,$\mathbf{W}^{L+1}$,$\mathbf{LS}^{L+1}$)
end
for $L \leftarrow 1^{L^V}$ **to** L^V **do**
  $\mathbf{C}^{V,(L+1)} \leftarrow$ Routing($\mathbf{C}^{L}$,$\mathbf{W}^{L+1}$,$\mathbf{LS}^{L+1}$)
end
Concatenation of Capsules Final Matrix & Vector Layer;
$\mathbf{F}^{L} \leftarrow \mathbf{C}^{M,(L+1)} + \mathbf{C}^{V,(L+1)}$
Linear Classification and Logits for each class;
Class Capsule Logits = Linear Classifier($\mathbf{F}^{L}$***)***

in our architecture. After the Primary Capsule Layer, we introduced a split, that sent the $Matrixpose$ output to the $ConvolutionalCapsuleLayer$ followed by $SystemY$. We also send the $Vectorpose$ output to $SystemX$. The outputs from $SystemX$ and $SystemY$ were then concatenated and sent to a classifier to detect class capsules. This happened in a feed-forward manner with no multiple iterations as stated earlier. Although our Convolutional and Fully connected capsule modules are flexible and can be adopted to any routing scheme explored so far. A step by step guide to our iteration-free routing algorithm is shown in Algorithms 1 and 2. In algorithms 1 and 2, $\mathbf{W}^{L+1}$ and $\mathbf{LS}^{L+1}$ start out as random matrices. But later, $\mathbf{W}^{L+1}$ captures the essence of the higher order capsules and converts the lower order capsules accordingly. $\mathbf{LS}^{L+1}$ on the other hand calculates the dimensions and package window for the next layer capsules based on the connection between the present layer capsules and next layer capsules through the agreement function. An outline of our architecture is provided in figure 1

V. Experimental Setup

A. Experiments on Cifar10 & Cifar-100

As we were thinking about data complexity and model scaling, we performed our experiments on the Cifar-10 and Cifar-100 dataset [21] which contains natural images. Natural images are inherently more complex than images of numbers. Cifar contains 50000 training images and 10000 test images of size 32×32. Cifar-10 has 10 classes and Cifar-100 has 100 classes.

For our models, we used the SGD optimizer with a learning rate of 0.01 along with $L2 - Normalization$ based weight decay of $5e-4$. We used Xavier Initialization to initialize all our weight matrices in our custom layers. For our loss function we used Binary Cross Entropy. We trained our models for 350 epochs. We introduced learning rate decay of 10 times at 150 epochs and 250 epochs. Before training the model, we introduced a number of generic data augmentations to the dataset. The data augmentations are Random Crop of 32 with 4 padding, Random Horizontal Flipping of probability 0.5. During evaluation, we did not perform any Data Augmentation. We trained and tested our models on an Nvidia GTX1070ti with a batch size of 16.

TABLE II
Evaluation on Cifar-10 (Simple Backbone)

Name	Backbone	Accuracy(Parameters)(Routing)
Dynamic Routing	simple	84.08%(7.99M)(R=1)
Em Routing	simple	82.19% (0.45M)(R=2)
Attention Routing	simple	82.83% (3.3M)
Dot Inverted Routing	simple	85.17% (0.56M)(R=4)
IterationFree Routing (SingleMatrix)	simple	83.76% (0.55M)
IterationFree Routing (CapsuleConv + System(X+Y))	simple	85.51% (0.31M)

B. Evaluation on Cifar-10 and Cifar-100

For our evaluation on Cifar-10 and Cifar-100, we took two procedures. One with the simple backbone and the other with a ResNet backbone as a robust feature extractor. The backbones were similar to the ones used by [17] to keep a fair comparison with other models.

TABLE III
EVALUATION ON CIFAR-10 (RESNET BACKBONE)(MORE IS BETTER)

Name	Backbone	Accuracy(Parameters)	Memory
Dynamic Routing	ResNet-7	92.65%(12.45M)	(3.7GB)
Em Routing	ResNet-7	92.15% (1.71M)	
Attention Routing	Custom	88.61% (9.6M)	
DeepCaps	Custom	91.01% (13.4M)	(6.8GB)
Dot Inverted Routing (In their Paper)	ResNet-7	94.73% (1.83M)(R=1) 94.85% (1.83M)(R=2) (Highest)=95.14% (R=N/A)	N/A (R=1) N/A (R=2)
Dot Inverted Routing (In Our Env)	ResNet-7	94.39% (1.83M)(R=1) 94.52% (1.83M)(R=2) (Highest)=94.73% (R=2)	(1.8GB) (R=1) (3.6GB) (R=2)
IterationFree Routing (SingleMatrix)	ResNet-7	94.72% (1.81M) (Highest)= 94.86%	(0.8GB)
ResNet-32		94.16% (21.3M)	

TABLE IV
EVALUATION ON CIFAR-100 (SIMPLE BACKBONE)(MORE IS BETTER)

Name	Backbone	Accuracy(Parameters)(Routing)
Dynamic Routing	simple	56.96% (31.59M)(R=1)
Em Routing	simple	37.73% (0.50M)(R=2)
Dot Inverted Routing	simple	57.32% (1.46M)(R=4)
IterationFree Routing (SingleMatrix)	simple	54.22% (1.07M)
IterationFree Routing (CapsuleConv + System(X+Y))	simple	60.22% (0.69M)

C. Comparisons with other Capsule Networks and Convolutional Networks based on experiments

As one can observe from the tables II, III, IV, V, our architectures require less parameters and memory to run. But the accuracy's are better, on par or close to previous state of the art architectures. At first from table II and IV, one can observe that our architectures perform better than Dynamic Routing and EM routing but are a few points behind Dot Inverted Routing. So we modified our architectures where we replaced all our Capsule Convolutional layers with layers from Dot Inverted Routing. We kept the rest of our layers and routing methodology same as before. Applying this methodology, we managed to get around 0.35× parameters decrease for Cifar-10 and 0.48× parameters decrease for Cifar-100 from the previously optimized architecture for capsules. These tests proved the robustness of our Fully Connected Layers as well as our own routing scheme. The effect of robust feature extractors on capsule networks are undeniable so we placed a better feature extractor representing the first 7 layers of ResNet-34 model and performed experiments. We replaced the

TABLE V
EVALUATION ON CIFAR-100 (RESNET BACKBONE) (MORE IS BETTER)

Name	Backbone	Accuracy(Parameters)	Memory
Dynamic Routing	ResNet-7	71.70%(36.04M)	(5.8GB)
Em Routing	ResNet-7	58.08% (1.76M)	
DeepCaps	Custom	63.03% (72.4M)	(6.9GB)
Dot Inverted Routing (In their Paper)	ResNet-7	76.02% (2.8M)(R=1) 76.27%(2.8M)(R=2) (Highest)=78.02% (R=N/A)	N/A (R=1) N/A (R=2)
Dot Inverted Routing (In Our Env)	ResNet-7	75.69% (2.8M)(R=1) (N/A)(2.8M)(R=2) (Highest)=75.89% (R=1)	(3.6GB) (R=1) (7.5GB) (R=2)
IterationFree Routing (SingleMatrix)	ResNet-7	75.85% (2.41M) (Highest)= 76.14%	(1.19GB)
ResNet-32		75.23% (21.3M)	

TABLE VI
ARCHITECTURE MEMORY CONSUMPTION WHEN INTRODUCED TO SPECIFIC CAPS LAYER (LESS IS BETTER)

Name(Paper)	2	4	8	16	32 (B)
CapsuleConv (Dot Inverted)	0.87	1.2	1.87	3.72	6.91GB
IterationFree Routing (SingleMatrix)	0.58	0.69	0.86	1.19	1.85GB

robust Convolutional Capsules from [17] because they require more memory. Table VI tries to bring this point to light. The settings were set for Cifar-100. So, we perform the rest of the experiments with our own Convolutional Capsule Layers. Results from these experiments are provided in tables III and V respectively. As one can observe from these tables, when added with a robust feature extractor, our model performs better than previous architectures, requiring less parameters and memory. Test accuracy can be observed in figure 2 for Cifar-10 and figure 3 for Cifar-100. We trained all models from [17] in our environment to provide a fair comparison. A table detailing the parameters and FLOPS compared to

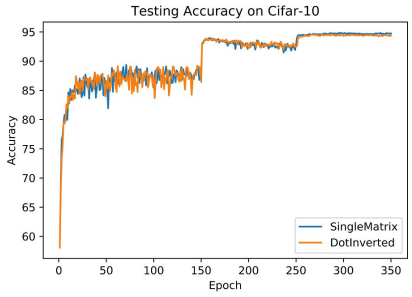

Fig. 2. Test Accuracy on Cifar-10

the most recent optimized architecture, Dot inverted routing is given in Table VII. The parameter counts in these tables are derived when our architecture is trained on Cifar-100. We also set the batch size to 1 as to avoid any unfair comparison. An important observation from this table shows us that in all cases our parameters and FLOP counts are less except

TABLE VII
THROUGHPUT OF EACH LAYER IN TERMS OF PARAMETERS AND FLOPS
(LESSER IS BETTER)

Name	Parameters	FLOPs
CapsuleConv (Dot Inverted)	3.317×10^5	5.852×10^8
IterationFree Routing (SingleMatrix)	5.645×10^5	6.502×10^7
CapsuleFC (Dot Inverted)	5.760×10^5	4.064×10^7
IterationFree Routing (SystemY)(Matrix)	6.024×10^4	2.258×10^7
IterationFree Routing (SystemX)(Vector)	2.987×10^5	1.179×10^8

for the parameters for $SingleMatrix$ Layer. But in the case of $SystemX$ our FLOP count is much less than the Dot-Inverted Capsule Layer. Observations show us that our model takes half and in case of Cifar-100 one-third the time of Dot-Inverted Routing even when Routing is set at a minimum of 1. This flexibility provided us the possibility to use both Matrix and Vector Poses. We also provide comparisons with

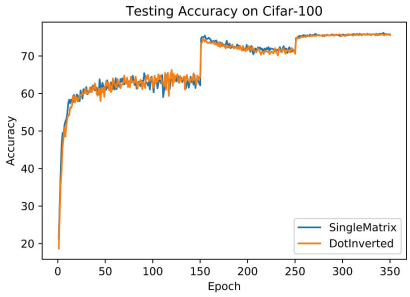

Fig. 3. Test Accuracy on Cifar-100

ResNet-34. ResNet-34 is a good representative for CNNs and our feature extractors are based on the first layers of ResNet architectures. Observations show that, training ResNet-34 from scratch for Cifar-10 and Cifar-100 falls behind the Capsule Networks. This might be due to the decreased batch size in our environment and no transfer learning from Imagenet [22] weights which seems to adversely affect all CNNs.

D. Capsule Characteristics

We perform some necessary tests, to observe if our architecture exhibits behavior akin to a capsule network. We took our architecture and added a linear decoder at the end of the Matrix Pose layer, the Vector Pose layer and the Final Concatenated layer. We trained for 700 epochs on the MNIST dataset.

After training, to observe if our model exhibits transformation equivariance, we performed dimension perturbation. We perturbed the scalar element of the Capsule outputs of each dimension by 0.0 to 2.0 over 10 values which is 0.2.

It can be observed from figure 4 that the model is learning a variety of transformations. We can assert from the figure that these transformations include partial strokes, scaling, localized transformations and in the last case rotations of localized parts along with scale. This might be due to the learning of complex

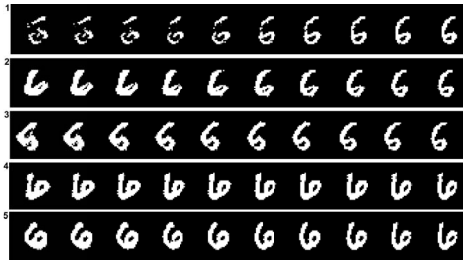

Fig. 4. MNIST Image Perturbation

poses in the fully concatenated capsules. For the third case, our architecture exhibits complex learning of digits to digits.

Then, with the following experiments we observe, if using both Matrix Poses and Capsule Poses in our architecture complemented the Final Concatenated Capsules. We compare reconstructions for the same numbers, for all three capsule variants in our architectures. From figure 5 we observe that,

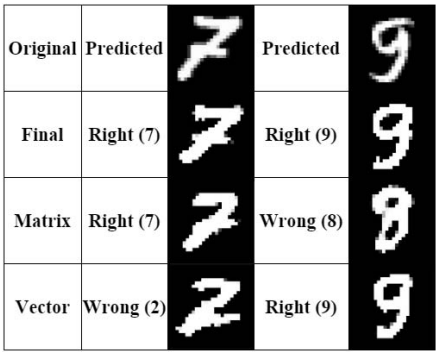

Fig. 5. Complement of Capsules

when the vector capsule failed to recognize a digit but the matrix capsule did, the final capsule detected and reconstructed the digit correctly. The opposite occurred too. We also performed the perturbation experiments on Cifar-10 with values from 0.0 to 2.0. As, we used a linear decoder, it had difficulty in picking up the feature aspects from the matrices, especially the matrix and vector poses. We assume that using a transposed convolution based decoder might provide better results. From figure 6 we observe the reconstruction of an image. Then, from figure 7 we observe that the model is learning the rotation of the horse, the outline of the horse, the shape of the horse without the legs etc.

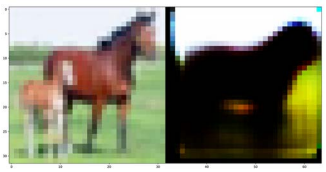

Fig. 6. MixCaps Reconstruction(Cifar-10)

We believe the experiments above prove the robustness of our model in terms of translation in-variance. This was

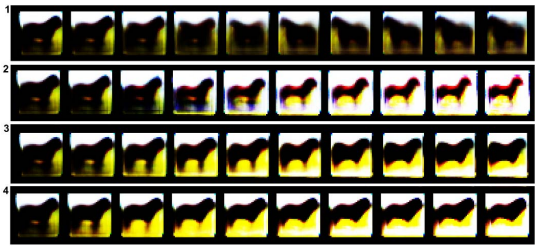

Fig. 7. Cifar-10 Image Perturbation

explained as a robust way in [16] and [6] to prove the nature of capsules.

VI. Conclusion

In this work, we proposed a new variant of capsule networks that uses iteration free routing to learn higher order capsules. We name it MixCaps as different types of capsules are used throughout the model. The model has a few new modular components that can be extended. They have contributed positively to the tests we performed along the way. Among these are Convolutional Capsule layer and Fully Connected capsules. These modules helped us to create a new iteration free routing algorithm which performs well without multiple iterations. This helped us achieve a robust architecture that requires less parameters and memory. This model builds upon the work of previous robust Convolutional Capsule models. Our model scales well when used with state of the art CNNs as backbone feature extractors such as ResNet.

Future directions can be taken from this work. As it seems, the performance of a capsule layer can be optimized to learn more from less. The higher level features from the feature extractors plays a vital role when it comes to capsule layers. Also, another observation might be how the number of capsules and dimension of a capsule contribute to accuracy of an architecture. Finally, to understand the full capability of Convolutional Capsules, it must be scaled to larger datasets if possible.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

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

[4] 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*, 2016, pp. 770–778.

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

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

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

[8] G. E. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with em routing," in *International conference on learning representations*, 2018.

[9] D. Wang and Q. Liu, "An optimization view on dynamic routing between capsules," 2018.

[10] S. Zhang, Q. Zhou, and X. Wu, "Fast dynamic routing based on weighted kernel density estimation," in *International Symposium on Artificial Intelligence and Robotics*. Springer, 2018, pp. 301–309.

[11] F. D. S. Ribeiro, G. Leontidis, and S. D. Kollias, "Capsule routing via variational bayes." in *AAAI*, 2020, pp. 3749–3756.

[12] S. S. R. Phaye, A. Sikka, A. Dhall, and D. Bathula, "Dense and diverse capsule networks: Making the capsules learn better," *arXiv preprint arXiv:1805.04001*, 2018.

[13] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[14] I. Paik, T. Kwak, and I. Kim, "Capsule networks need an improved routing algorithm," *arXiv preprint arXiv:1907.13327*, 2019.

[15] J. Rajasegaran, V. Jayasundara, S. Jayasekara, H. Jayasekara, S. Seneviratne, and R. Rodrigo, "Deepcaps: Going deeper with capsule networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 725–10 733.

[16] J. Choi, H. Seo, S. Im, and M. Kang, "Attention routing between capsules," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[17] Y.-H. H. Tsai, N. Srivastava, H. Goh, and R. Salakhutdinov, "Capsules with inverted dot-product attention routing," *arXiv preprint arXiv:2002.04764*, 2020.

[18] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, 2000.

[19] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[20] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

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

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