Interpretable Capsule Networks

A Project as a Course requirement for

Master of Technology in Computer Science

Ankit Anand

18553

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

(Deemed to be University)

Department of Mathematics and Computer Science

Prasanthi Nilayam Campus

March 2020

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING
(Deemed to be University)

Dept. of **Mathematics & Computer Science**
Prasanthi Nilayam Campus

CERTIFICATE

This is to certify that this Project titled **Interpretable Capsule Networks** submitted by Ankit Anand, 18553, Department of Mathematics and Computer Science, Prasanthi Nilayam Campus is a bonafide record of the original work done under my supervision as a Course requirement for the Degree of Master of Technology in Computer Science.

……..
Dr. S. Balasubramanian
Project Supervisor

Countersigned by

Place: ……...
Dr. R. Raghunatha Sarma
Date: Associate Head, DMACS

DECLARATION

The Project titled **Interpretable Capsule Networks** was carried out by me under the supervision of Dr. S. Balasubramanian, Department of Mathematics and Computer Science, Prasanthi Nilayam as a Course requirement for the Degree of Master of Technology in Computer Science and has not formed the basis for the award of any degree, diploma or any other such title by this or any other University.

……..

Place :

Ankit Anand

18553

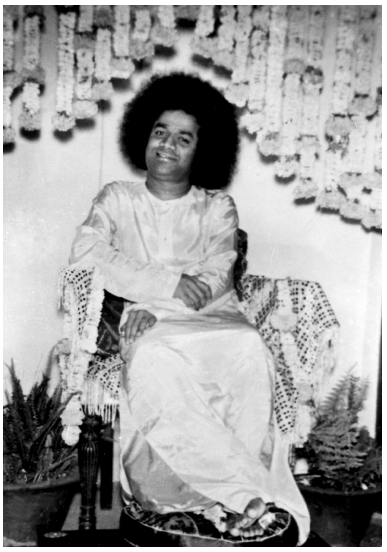
Date :

M.Tech Computer Science

Prasanthi Nilayam

Dedicated at The Lotus Feet of

my Beloved Mother Sai



Acknowledgements

With deepest gratitude in my heart, I express my heartfelt thanks to my beloved Lord, Bhagawan Sri Sathya Sai Baba, who is the source of inspiration throughout my life. This work wouldn't have been possible without His grace. I would like to express my gratitude to my parents and my sister, for their love and care for me.

My heartfelt gratitude to my supervisor, Associate Professor, Dr. S. Balasubramanian, Department of Mathematics and Computer Science, Sri Sathya Sai Institute of Higher Learninig, for his excellent guidance and timely advice. I also thank brother Sai Raam Venkataraman, a research scholar in Department of Mathematics and Computer Science, Sri Sathya Sai Institute of Higher Learninig for his invaluable guidance and support. I also thank Associate Professor, Dr. Vineeth N Balasubramanian

I would like to thank Prof. (Dr.) C B Sanjeevi, the Vice Chancellor, Dr. B Sai Giridhar, the Registrar, Sri Sanjay Sahini, the Controllor of Examinations, Prof. Siva Sankar Sai, the Director, Sri Sathya Sai Institute of higher learning, Prashanthi Nilayam and Dr. R. Raghunatha Sarma, Associate Head, Department of Mathematics and Computer Science, for encouraging me to pursue this project. I am also grateful to my DMACS family who made me feel that I was always at home. I thank Sri. C.Uday Kiran sir, Sri Varun sir for taking their valuable time to set up the lab.

I also extend my deep gratitude to all the professors, Asst. professors, teachers and research scholars of our department, for their guidance.

I would like to thank Raghava, attender in the department, for his support. I thank all my classmates for their encouragement.

I thank all my "Roommates" and "Hostel inmates" for their constant support and encouragement.

I would like to thank Chalam sir and all the hostel workers for preparing delicious food in hostel.

Lastly, I thank all those whom I haven't mentioned above but have directly or indirectly helped in the completion of this work.

SRI SATHYA SAI INSTITUE OF HIGHER LEARNING, PRASANTHI NILAYAM

Abstract

Capsule neural networks [1] improve over many shortcomings of the convolutional neural networks [2]. However, to be able to use capsule networks in various applications, the network should be interpretable. To that end, we propose simple regularization techniques, termed *impurity regularizations*. The idea is to make the network tree-like because a tree-like formation of the routing weights at each layer allows a clear-cut decomposition of the input into its important components thereby helping the user decipher the reason behind the network's output. The experiments show that this also improves learning for the classification tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The common use of artificially intelligent agents interacting with, and aiding humans has been a part of the collective consciousness of society for centuries, usually finding expression in the form of science fiction. In recent decades, however, these notions have moved beyond literature to the more technical field of artificial intelligence, and in the past few years to practically viable products.

While the goal of building devices with human-level intelligence is still far from any notion of achievement, or even concrete definition, agents with limited and specific intelligence have been the part of major research as well as public use for a variety of tasks[4–7].

Much of these advances is made possible due to rapid successes in the so-called field of deep learning. Deep learning involves the use of neural networks with many hidden layers to learn a multi-level representation of an input, and use this to perform tasks such as classification, regression or generation.

While the current revolution of deep learning research has several contributing factors, including both software and hardware advances, a key development is the use of deep convolutional neural networks (CNNs)[8].

Ever since the introduction of deep CNNs (and several variants [9, 10]) to solve learning tasks, few other models of learning have come with comparable performance.

A huge reason for this is the efficiency and intuition that the convolutional layer brings with it. CNNs have a shared-weights architecture for each layer that reduces the number of parameters each layer uses, while allowing for shared detection of patterns across an input. Moreover, the property of translation-equivariance of convolutions [11] allows shifted patterns to be detected without loss of information, mirroring the translation symmetry of the label function - which is often invariant to translations.

The parameter-efficiency of CNNs allows for deeper architectures with a large number of hidden layers. This allows for a representation of an input across multiple semantic resolutions [12], while also improving general performance - as experiments have generally found. Thus, the CNN model and its many variations [9, 10] have generally showcased state-of-the-art results across multiple vision-tasks[13, 14].

While the above is true, and CNNs continue to showcase such results, there are a number of issues that should be overcome before image-understanding is possible. We name a few here.

First, the use of subsampling 'pooling' layers[8, 10] causes a loss of local information about the presence of patterns detected by the previous layer. Thus, pooling allows for relaxations in the representation of compositional structures that do not naturally showcase such relaxations. Specifically, pooling layers do not allow for an accurate learning of inter-component spatial relationships that defines the structure of much of vision. While such subsampling layers contribute to improved performance, they affect the semantic correctness of representations, and therefore, in turn any notion of interpretability.

Second, CNNs detect features based only on the presence of components, ignoring information that the pose of these can offer. This is directly due to the inductive-bias of CNNs that uses scalar neurons - each of which is obtained from a correlation of a filter and a local pool of input neurons. A better model would take into consideration the pose of components, and thus learn their spatial relationships.

These arguments resulted in the neural network architecture termed 'capsule networks' [15][1]. In this model, each 'neuron', termed a 'capsule', is a vector of

coordinates that aims to describe the pose of learned visual-entities at different levels of semantic resolution. Moreover, a deeper capsule is built from a consensus-based summation of predictions made for it by a local pool of capsules from a shallower layer by a process termed 'routing'. Thus, a component is detected based on the spatial alignment of its parts.

Capsule networks, thus aim to build a compositional representation, similar to a parse-tree, of components based on the activation of capsules. In other words, a properly trained capsule network should provide a compositional interpretation of its input, that can be seen by considering the components represented by active capsules. This provides a greater measure of interpretability than most deep neural network models, and also allows for better image understanding.

Interpreting neural networks has been and important area of a research. Many critical fields demand that decisions should be open to explanations and unambiguous intrepretation. However, neural networks do not naturally allow this. Thus, a significant amount of research has focused on interpreting decisions made by deep neural networks[16, 17]. A second, less popular, approach aims at building interpretable neural network models[18, 19]. However, there is little work on interpreting neural networks based on compositional structures - something that is latent in most natural images. Therefore, we aim to study capsule networks in this regard.

1.1 Objectives

1. Evaluate the interpretability of existing capsule networks.
2. Develop interpretable models for capsule networks.

1.2 Contributions

1. Metrics for the compositionality of capsule network models.
2. Learning strategies to optimise the metrics.

1.3 Thesis Outline

* Chapter 2 presents a concise description of some fundamental papers in the field of capsule networks. We explain some of the main ideas used in the capsule networks. We summarize three important papers of the field.

* In chapter 3, we start with motivating our work. We then, propose a regularization term to enforce appropriate tree-structures among capsules. We give full details of the models and other training parameters. We conclude with observations and results.

* Chapter 4 presents conclusions and directions for future work.

Chapter 2

Literature Survey and Prerequisites

In order to learn compositional structures in data, it is appropriate to use a model that is itself compositional. Thus, a model must represent an input as a parse tree of detected parts. Capsule networks are model to do this in a learnable manner. Here, we present some existing work for capsule networks.

2.1 CapsNet

Dynamic routing between capsules[1] is one the first papers on capsule networks. In the model it introduced, each capsule is encoded by a vector whose norm does not exceed 1. The probability of the existence of an object each capsule denotes is encoded by the length of the corresponding vector. In order to ensure the length of each capsule is less than 1, a non-linear "**squashing**" is applied. This is formulated as:

$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1+||\mathbf{s}_j||^2}\frac{\mathbf{s}_j}{||\mathbf{s}_j||} \tag{2.1}$$

where $\mathbf{s}_j$ is the total input and $\mathbf{v}_j$ is the vector output of capsule j.

Deeper capsules are built from a consensus-weighted sum of the predictions $\hat{\mathbf{u}}_{j|i}$ made from each lower-level capsule. These predictions are obtained by multiplying the

lower-level capsules $\mathbf{u}_i$ by a learnable weight matrix $\mathbf{W}_{ij}$. Concisely,

$$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i} \tag{2.2}$$

where

$$\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij} \mathbf{u}_i \tag{2.3}$$

c_{ij}s are determined by an iterative process, termed dynamic routing. In this, the logits b_{ij} are first considered. These are log-prior probabilities that the capsule i should be coupled to the capsule j. c_{ij} are obtained by:

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{2.4}$$

The log-priors do not depend on the current input image but on the type and the location of two capsules. The coefficients for coupling are refined by using an agreement between the output of capsule j, $\mathbf{v}_j$, and the prediction $\hat{\mathbf{u}}_{j|i}$ made by the capsule. The agreement used is the dot product.

$$a_{ij} = \mathbf{v}_j . \hat{\mathbf{u}}_{j|i} \tag{2.5}$$

This agreement is treated as log likelihood and is added to b_{ij} for computing new values of coupling coefficients. Figure 2.1 shows the dynamic routing algorithm.

procedure ROUTING($\hat{\boldsymbol{u}}_{j|i}$, r, l)
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
  for r iterations **do**
    for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
    for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
    for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
    for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} . \mathbf{v}_j$
  return $\mathbf{v}_j$

FIGURE 2.1: Dynamic routing algorithm.

In order to train a model, the margin loss was used. For each output capsule k, the loss is computed as

$$L_k = T_k max(0, m^+ - ||\mathbf{v}_k||)^2 + \lambda (1 - T_k) max(0, ||\mathbf{v}_k|| - m^-)^2 \tag{2.6}$$

where $T_k = 1$ if the input is of class k; $T_k = 0$ for other classes. Thus, the output layer consists of one capsule per class. The values for $m^+ and m$ were fixed as 0.9 and 0.1, respectively. λ was chosen as 0.5. Absent classes' loss is weighted by λ and to stop the initial learning from reducing the lengths of the vectors outputs of all digit capsules. The total loss is sum of the losses of all digit capsules.

The complete architecture for the model in this paper is given in 2.2. The initial convolutional layer is so that appropriate features be used to create capsules. The primary capsules layer groups neurons to build vector capsules. The primary capsules are then used to compute the output capsules. This architecture is termed CapsNet. Also, in order to ensure the capsules learn better features, the architecture uses a decoder which reconstructs the image from a capsule. The decoder network is a fully connected network which takes the masked input from the final capsules. The mean squared error of the reconstruction is used as regularization term in the loss with a weight of 0.0005.

The authors present accuracy results for classification on MNIST, AFFNIST, SVHN and CIFAR10. The ability of CapsNet to handle transformation of the input is seen by the result of 79% on affnist test set (MNIST accuracy 99.23%). A convolutional network with similar number of parameters achieving similar accuracy on expanded MNIST (99.22%) test set acieved 66% on affnist. The authors also experimented with segmenting overlapping digits. The authors argue that since the routing allows each capsule to attend to some of the active capsules at lower level and ignore others, the routing can be viewed as a parallel attention mechanism thereby allowing the model to recognize multiple objects even if they overlap. For this, the authors generate MultiMNIST, a variant of MNIST with overlapping digits. The reconstructions obtained demonstrate that CapsNet is able to segment images into two original digits.

2.1.1 Pros and Cons of CapsNet

There are many advantages of capsule networks over the convolutional neural network model. Some of these are:

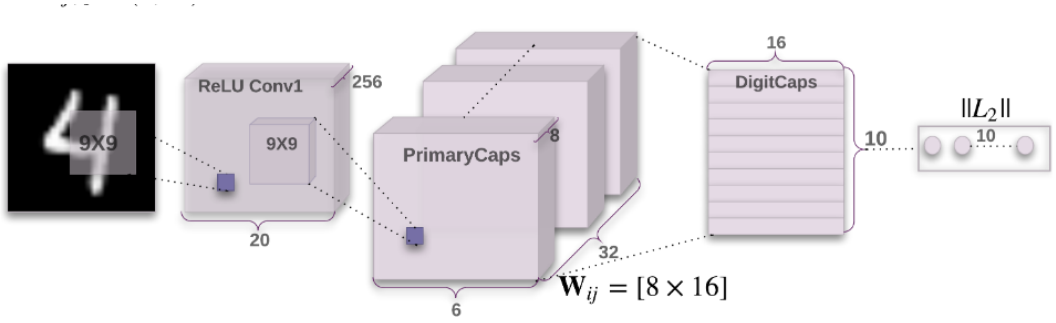


FIGURE 2.2: CapsNet[1]

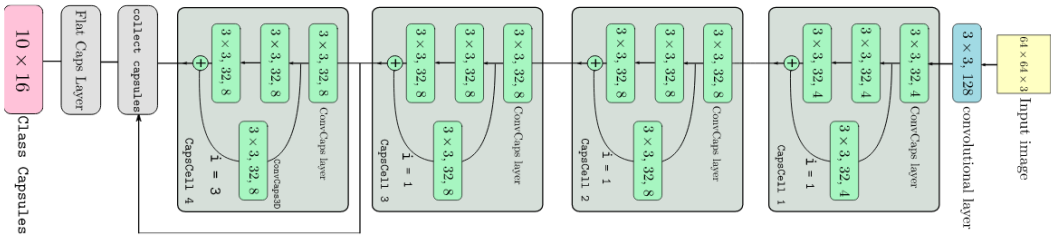


FIGURE 2.3: DeepCaps[3]

(i) It requires lesser training data due to the learning of transformation robust part-whole relations.

(ii) Routing allows CapsNet to work with overlapping objects.

(iii) The activation vectors are found to be quite interpretable, and correspond to features of the input.

Despite the advantages, there are several disadvantages to capsule networks in this paper. Some of the cons of capsule networks are:

(i) Capsule networks do not achieve good results on data with non-uniform backgrounds.

(ii) CapsNet is computationally expensive.

We have implemented this network. Table 2.1 lists the results.

Algorithm 1 Dynamic Routing using 3D convolution

procedure ROUTING
Require: $\mathbf{\Phi}^l \in \mathbb{R}^{(w^l, w^l, c^l, n^l)}$, r and c^{l+1}, n^{l+1}
    $\tilde{\mathbf{\Phi}}^l \leftarrow \texttt{Reshape}(\Phi_l) \in \mathbb{R}^{(w^l, w^l, c^l \times n^l, 1)}$
    $\mathbf{V} \leftarrow \texttt{Conv3D}(\tilde{\mathbf{\Phi}}^l) \in \mathbb{R}^{(w^{l+1}, w^{l+1}, c^l, c^{l+1} \times n^{l+1})}$
    $\tilde{\mathbf{V}} \leftarrow \texttt{Reshape}(\mathbf{V}) \in \mathbb{R}^{(w^{l+1}, w^{l+1}, n^{l+1}, c^{l+1}, c^l)}$
    $\mathbf{B} \leftarrow \mathbf{0} \in \mathbb{R}^{(w^{l+1}, w^{l+1}, c^{l+1}, c^l)}$
    Let $p \in w^{l+1}, q \in w^{l+1}, r \in c^{l+1}$ and $s \in c^l$
    for i iterations **do**
        for all p, q, r, $k_{pqrs} \leftarrow \texttt{softmax\_3D}(b_{pqrs})$
        for all s, $S_{pqr} \leftarrow \sum_s k_{pqrs} \cdot \tilde{V}_{pqrs}$
        for all s, $\hat{S}_{pqr} \leftarrow \texttt{squash\_3D}(S_{pqr})$
        for all s, $b_{pqrs} \leftarrow b_{pqrs} + \hat{S}_{pqr} \cdot \tilde{V}_{pqrs}$
    return $\mathbf{\Phi}^{l+1} = \hat{\mathbf{S}}$

FIGURE 2.4: Dynamic Routing for 3D convolutions[3]

2.2 DeepCaps

Constructing deep models is important to achieving high performance. This is done for capsules in DeepCaps[3]. In the model proposed in this work, intermediate capsule layers are built from reshaped convolutions without routing. In the final two layers, dynamic routing is used. The architecture uses resnet-style connection between capsules to build a deep model with many layers. Further, the last convolutional capsule layer presents a changed dynamic routing algorithm. This is done by using 3D convolutions and a 3D version of the softmax function. The 3D version of the softmax $softmax\_3D$ is given as

$$softmax\_3D(\mathbf{B}_s) = \frac{exp(b_{pqrs})}{\sum_x \sum_y \sum_z exp(b_{xyzs})} \tag{2.7}$$

Paper	Train Data	Test Data	Official	Ours
CapsNet[1]	trans(MNIST)	MNIST	99.75	99.51
CapsNet[1]	trans(FaMNIST)	FaMNIST	-	91.70
DeepCaps[3]	trans(MNIST)	MNIST	99.72	99.53
DeepCaps[3]	trans(FaMNIST)	FaMNIST	94.46	92.12

TABLE 2.1: Implementation Results for CapsNet[1] and DeepCaps[3]

where $\mathbf{B}_s \in \mathbb{R}^{(w^{l+1},w^{l+1},c^{l+1})}$ for each $s \in [c^l]$, c_l is number of capsules in layer l, c_{l+1} is number of capsules in layer $l+1$. Figure 2.3 shows the architecture of DeepCaps. Figure 2.4 shows the proposed dynamic routing for the 3D convolutions. The authors also used a class-independent decoder for image reconstruction as a regularization term in the loss with a weight of 0.4. The authors used margin loss(2.6) for supervised training.

For improved results, the training has two phases. In the first phase, called the soft-training, m^+ is 0.9, m^- is 0.1 and *lambda* is 0.5. The second phase, called the hard-training step, m^+ is 0.95, m^- is 0.05 and *lambda* is 0.8. The two phases separately initialize model parameter optimizers and learning rate schedulers. The decoder network uses deconvolutional layers, unlike CapsNet, to reconstruct the images. The authors argue that this decoder is better than the decoder used by the CapsNet[1] as convolutions capture spatial features better than the fully-connected network. Further, the authors also argue that the class-independence leads to better regularization as the network is forced to learn decoding in a constrained space. The authors also observed that across all of the classes, a specific instantiation parameter captures a specific feature or change i.e., a specific component of any of the 10 class capsules captures the same specific feature or change. They also observed that instantiation parameters with low variance show mixed changes.

Our implementation of this paper has been made public on github, and is in pytorch - the official code is in keras. The results of our model are shown in table 2.1.

2.3 SOVNET

Capsule network models use a large number of parameters due to them attempting to learn all part-whole relationships between capsules. However, many of these capsules do not belong to the same compositional structure, and do not have any part-whole relation. Moreover, most capsule network models are not equivariant - though equivariance is an important property in order to preserve learnt compositionalities. Therefore, in order to correct these, the space-of-variation network (SOVNET) [20] model was proposed.

In SOVNET, the prediction network learns to projectively encode descriptions of different capsule-types for every layer. This is achieved by associating each capsule-type with some learnable vector-valued function. To guarantee equivariance, SOVNETs use group-equivariant CNNs. Further, a degree-centrality routing algorithm which is provably equivariant was proposed.

The authors of the paper conducted experiments on MNIST[21], FashionMNIST [22] and CIFAR-10[23] for showing transformation robustness at train and test phases. Experiments showcasing the performance of a SOVNET model on KMNIST [24], SVHN [25], CIFAR-100, and AFFNIST are also done. The experiments conducted by the authors show that SOVNET architecture beats many capsule architectures in learning transformation robustness on classification.

The network that we use has a ResNet[9] backbone. So, input to the SOVNET proper is the features learnt by the ResNet layers. Further, SOVNET uses usual convolutions and not the group-equivariant ones mentioned in the paper.

Apart from these papers, several capsule network architectures are proposed in the literature. In [26], Hinton et. al propose a convolutional capsule that uses shared parameters across convolutionally moved patches of a input-grid. In that paper, an EM-routing procedure was proposed. In [27], an SVD-based routing strategy was introduced. In [28], a provably equivariant capsule network was proposed. This network treats each capsule as an element of a group. This formulation allows for a greater transformation-efficiency when compared to equivariant CNN models. Several other

models exist, and propose variations of architecture [29], routing algorithm [30, 31], or combining models [32].

Chapter 3

Interpretable Capsule Networks

3.1 Motivation

Capsule networks allow for a compositional representation of data through the routing mechanism. In other words, shallower capsules strongly routed to deeper capsules capture semantic part-whole relationships. Since these relationships are often in the form a parse-tree, it is expected that a capsule network possesses such a structure in its routing weights.

Thus, a deeper capsule can have many strongly connected shallower capsule; however, a shallower capsule is strongly connected to only one deeper capsule. With this motivation, in order to study if routing enforces this structure, we conducted several experiments on classification tasks. First, we checked whether routing, as in the literature, affects the classification accuracy of capsule networks. We trained and tested two variations of CapsNet models on MNIST[21] and FashionMNIST[22] datasets. Thus, the models either used dynamic routing or gave equal weights to each prediction. The results of this are given in Table 3.1.

We see that the accuracies are comparable. Further, we computed and plotted entropy values for the 1152 capsules that route to the class capsules. We used base 2 logarithm for our computations. So, the highest entropy possible is 3.32192809489. Figures 3.1-3.10 give entropy plots for 1152 capsules for class 0 to class 9 respectively.

Dataset	Dynamic Routing	Equal Weights
MNIST	99.51	99.45
FaMNIST	91.70	90.92

TABLE 3.1: Implementation Results for CapsNet[1]

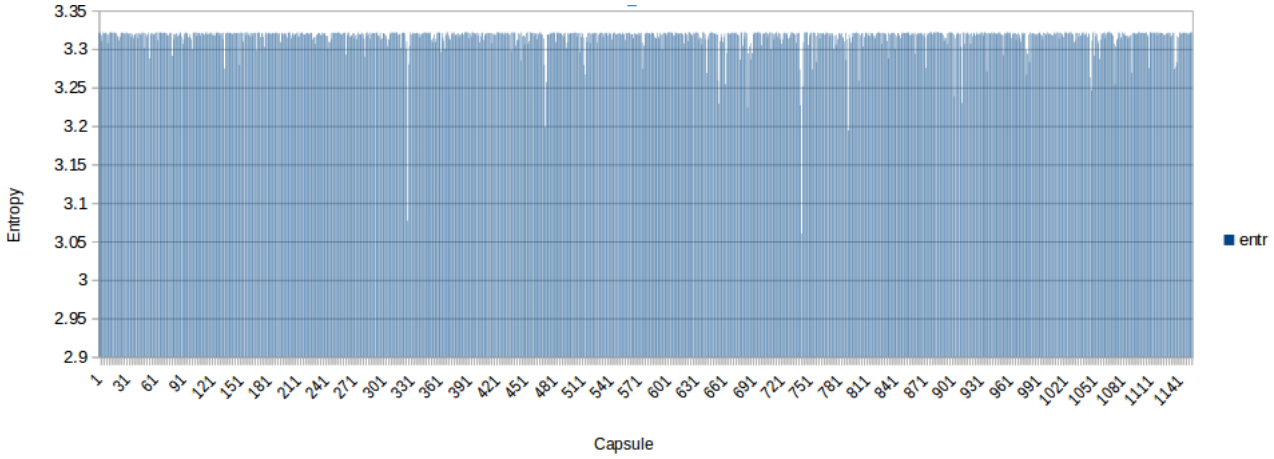


FIGURE 3.1: Entropy of 1152 capsules of CapsNet[1] for class 0

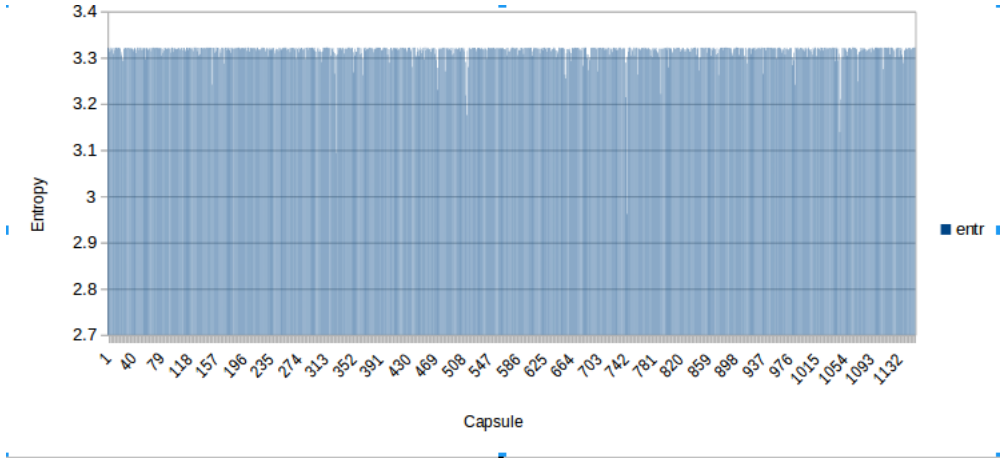


FIGURE 3.2: Entropy of 1152 capsules of CapsNet[1] for class 1

We see that the entropy values are very high, comparable to equal weights-based capsule network (here, all the entropy values will be 3.32192809489) indicating poor parent-child type relationships. Thus in practice, dynamic routing behaves much as if there was no routing. In other words, dynamic routing does not enforce parent-child relationship between the higher level and lower level capsules. We aim to develop a means of rectifying this. The benefits of such a rectification would make capsule networks learn better spatial relationships, and thus become more interpretable. Thus, an interpretation of the output of such a network would arise from considering the activated capsules that

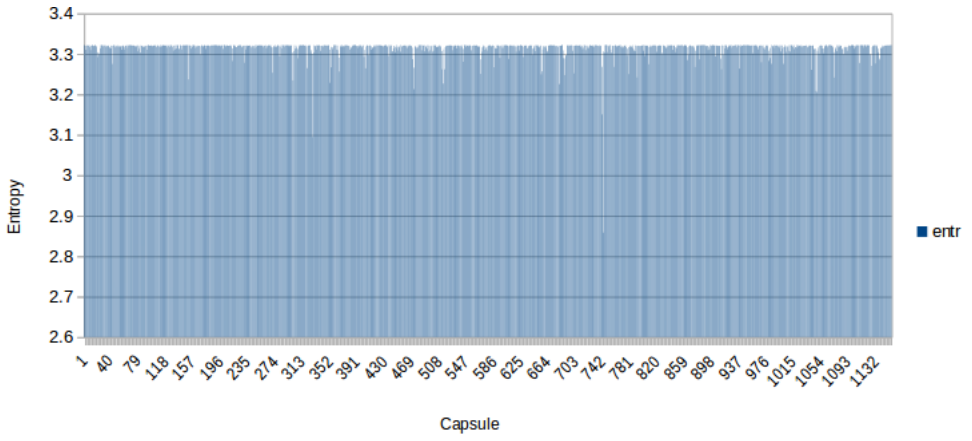


FIGURE 3.3: Entropy of 1152 capsules of CapsNet[1] for class 2

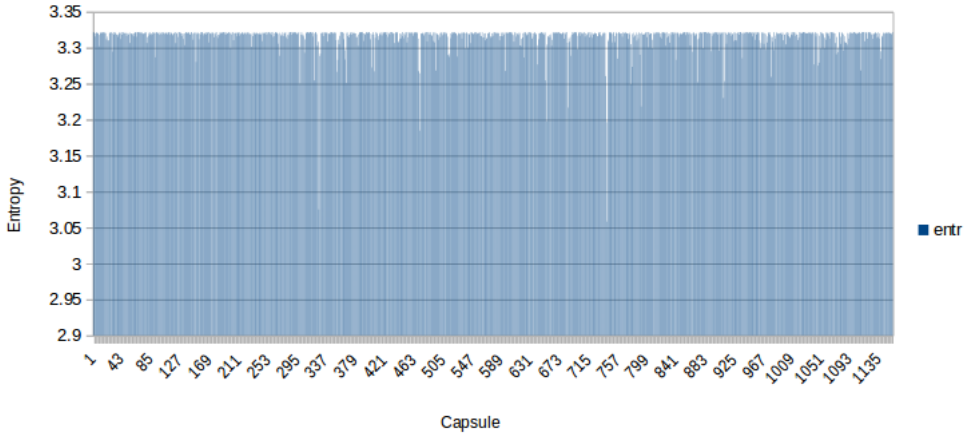


FIGURE 3.4: Entropy of 1152 capsules of CapsNet[1] for class 3

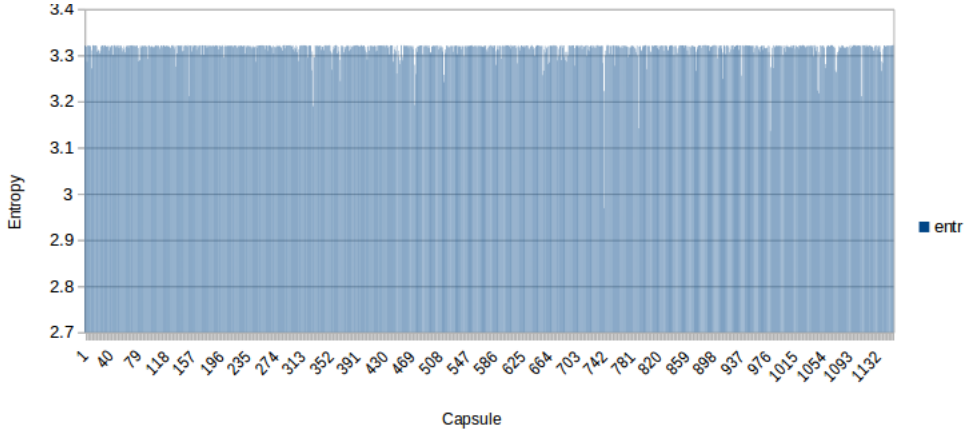


FIGURE 3.5: Entropy of 1152 capsules of CapsNet[1] for class 4

route strongly to deeper capsules. A tree-like formation of the routing weights at each layer allows a clear-cut decomposition of an input into its important components.

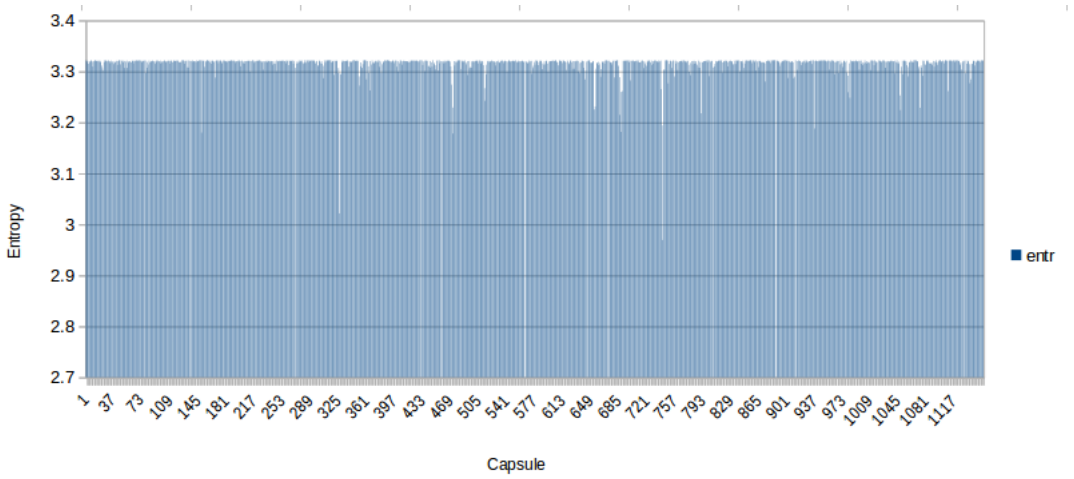


FIGURE 3.6: Entropy of 1152 capsules of CapsNet[1] for class 5

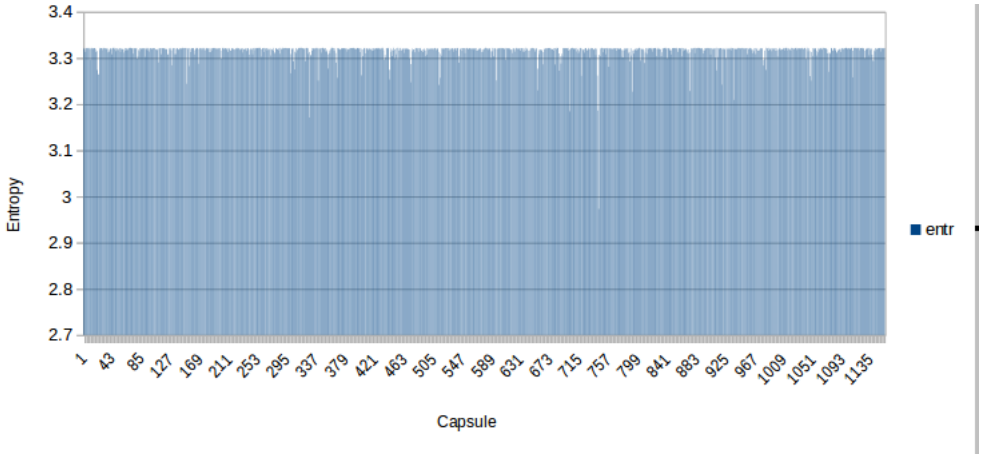


FIGURE 3.7: Entropy of 1152 capsules of CapsNet[1] for class 6

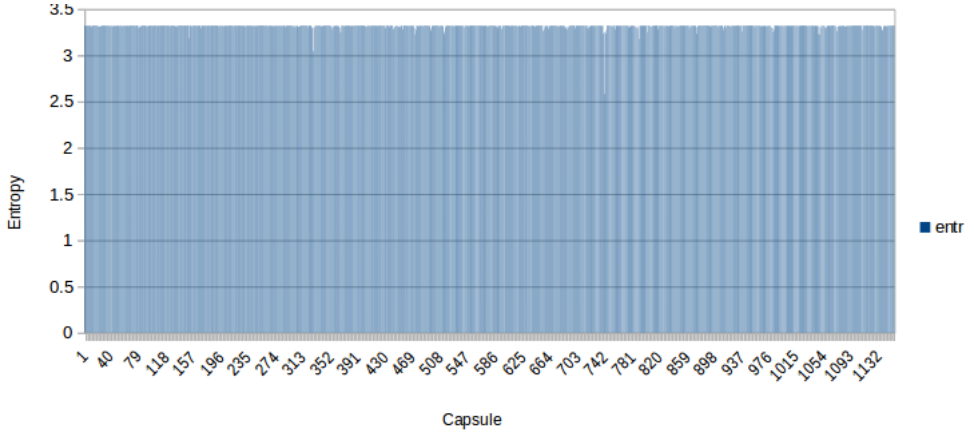


FIGURE 3.8: Entropy of 1152 capsules of CapsNet[1] for class 7

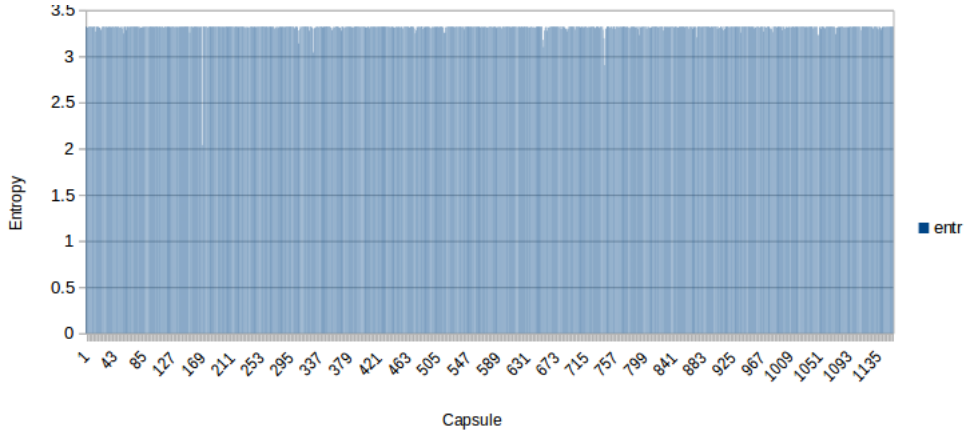


FIGURE 3.9: Entropy of 1152 capsules of CapsNet[1] for class 8

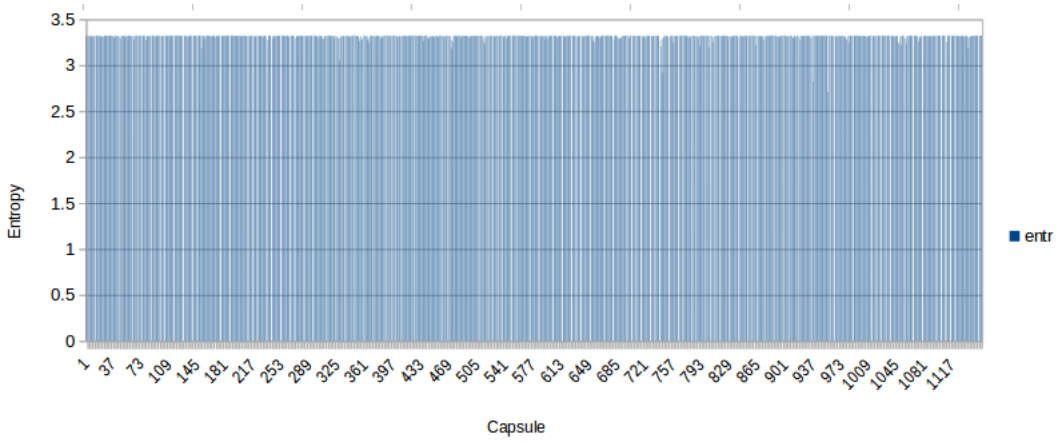


FIGURE 3.10: Entropy of 1152 capsules of CapsNet[1] for class 9

3.2 Impurity Regularization

In order to maintain the parent-child relationship, we introduce a novel regularization loss for training of capsule networks, termed 'impurity regularization'. This regularization is a capsule-capsule layer-wise loss-based regularization. We consider the regularization term as the mean impurity of all the routing coefficients for the lower level and upper level capsules over the training batch. For example, consider the entropy of the routing coefficients

$$reg\_value = \frac{-1}{N*I*H*W} \sum_{N,I,H,W} \sum_{J} c_{ij} log_{c_o}(c_{ij}) \tag{3.1}$$

where c_o is the number of classes, c_{ij} is routing coefficients with dimension $(N \times I \times J \times H \times W)$, N is batch size, I is the number of lower level capsules at each grid position of $H \times W$ grid. The total regularization value is the sum of all the *reg_value*'s for each capsule-capsule layer.

The total loss comprises of a convex combination of the usual loss like the margin loss or cross-entropy loss and the entropy regularization term. For our experiments, we consider the weights 0.2, 0.4 and 0.6 for the regularization term, with the corresponding weights based, on the convex combination, for a capsule network's supervised loss and reconstruction loss.

We also consider other measures of impurity. Specifically, we also regularize with gini index and mis-classification error. The gini index based regularization term is

$$gini = \frac{1}{N*I*H*W} \sum_{N,I,H,W} (1 - \sum_{J} c_{ij}^2) \tag{3.2}$$

and the mis-classification error based term is

$$mcls = \frac{1}{N*I*H*W} \sum_{N,I,H,W} (1 - max_J c_{ij}) \tag{3.3}$$

3.3 Experiments

We conducted several experiments on SmallNorb dataset [33]. For the SmallNorb dataset we used two deeper, different architectures. We considered two different network architectures given in (3.3.1). For the first model, we used 20% of the training set for validation. The testing is done for the model with the best validation score. For training, we resized the images to $48 \times 48 \times 1$ and random-cropped to $32 \times 32 \times 1$. We also adjusted brightness and contrast and normalized the images. For validation and testing, the images were resized to $48 \times 48 \times 1$ and center-cropped to $32 \times 32 \times 1$. Then, the images were normalized before validation/testing. For the second model, we choose to work with the model which gives highest accuracy on the test set. The whole

of training set is used. The transformations applied are same as that for the training and testing of the first model.

3.3.1 Model Architecture, Loss Functions and Optimizers

The first model uses a ResNet[9] network as backbone for the capsule network. The capsule network is a SOVNet-type network[20]. Specifically, the network is

```
ResnetCnnsovnetDynamicRouting(
  (resnet_precaps): ResNetPreCapsule(
    (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(16, eps=1e-5, momentum=0.1, affine=True,
                       track_running_stats=True)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),
                        padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(16, eps=1e-5, momentum=0.1, affine=True,
                           track_running_stats=True)
        (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),
                        padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(16, eps=1e-5, momentum=0.1, affine=True,
                           track_running_stats=True)
        (shortcut): Sequential()
      )
      (1): BasicBlock(
        (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),
                        padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(16, eps=1e-5, momentum=0.1, affine=True,
                           track_running_stats=True)
```

```
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),
                      padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-5, momentum=0.1, affine=True,
                         track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2),
                      padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True,
                         track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1),
                      padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True,
                         track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(16, 32, kernel_size=(1, 1), stride=(2, 2),
                    bias=False)
        (1): BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True,
                         track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1),
                      padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True,
                         track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1),
                      padding=(1, 1), bias=False)
```

```
      (bn2): BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True,
                         track_running_stats=True)
      (shortcut): Sequential()
    )
  )
)
(primary_caps): PrimaryCapsules(
  (preds): Sequential(
    (0): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
    (1): SELU()
    (2): LayerNorm(torch.Size([128, 16, 16]), eps=1e-05,
                   elementwise_affine=True)
  )
)
(conv_caps1): ConvCapsule(
  (preds): Sequential(
    (0): Conv2d(8, 512, kernel_size=(3, 3), stride=(2, 2))
    (1): BatchNorm2d(512, eps=1e-5, momentum=0.1, affine=True,
                     track_running_stats=True)
    (2): SELU()
  )
)
(conv_caps2): ConvCapsule(
  (preds): Sequential(
    (0): Conv2d(16, 256, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(256, eps=1e-5, momentum=0.1, affine=True,
                     track_running_stats=True)
    (2): SELU()
  )
)
(class_caps): ConvCapsule(
```

```
    (preds): Sequential(
      (0): Conv2d(16, 80, kernel_size=(5, 5), stride=(2, 2))
      (1): BatchNorm2d(80, eps=1e-5, momentum=0.1, affine=True,
                       track_running_stats=True)
      (2): SELU()
    )
  )
  (linear): Linear(in_features=16, out_features=1, bias=True)
)
```

There are three convolutional-capsule layers with 32, 16 and 5 capsule-types, respectively. The second model is a DeepCaps[3] model. The architecture is as follows:

```
Model(
  (conv2d): Conv2d(1, 128, kernel_size=(3, 3), stride=(1, 1),
                   padding=(1, 1))
  (batchNorm): BatchNorm2d(128, eps=1e-08, momentum=0.99,
                        affine=True, track_running_stats=True)
  (toCaps): ConvertToCaps()
  (conv2dCaps1_nj_4_strd_2): Conv2DCaps(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2),
                    padding=(1, 1))
  )
  (conv2dCaps1_nj_4_strd_1_1): Conv2DCaps(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
  (conv2dCaps1_nj_4_strd_1_2): Conv2DCaps(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
  (conv2dCaps1_nj_4_strd_1_3): Conv2DCaps(
```

```
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
  (conv2dCaps2_nj_8_strd_2): Conv2DCaps(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
                    padding=(1, 1))
  )
  (conv2dCaps2_nj_8_strd_1_1): Conv2DCaps(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
  (conv2dCaps2_nj_8_strd_1_2): Conv2DCaps(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
  (conv2dCaps2_nj_8_strd_1_3): Conv2DCaps(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
  (conv2dCaps3_nj_8_strd_2): Conv2DCaps(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
                    padding=(1, 1))
  )
  (conv2dCaps3_nj_8_strd_1_1): Conv2DCaps(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
  (conv2dCaps3_nj_8_strd_1_2): Conv2DCaps(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                    padding=(1, 1))
  )
```

```
(conv2dCaps3_nj_8_strd_1_3): Conv2DCaps(
  (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                  padding=(1, 1))
)
(conv2dCaps4_nj_8_strd_2): Conv2DCaps(
  (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
                  padding=(1, 1))
)
(conv3dCaps4_nj_8): ConvCapsLayer3D(
  (conv1): Conv3d(1, 256, kernel_size=(3, 3, 3), stride=(8, 1, 1),
                  padding=(0, 1, 1))
)
(conv2dCaps4_nj_8_strd_1_1): Conv2DCaps(
  (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                  padding=(1, 1))
)
(conv2dCaps4_nj_8_strd_1_2): Conv2DCaps(
  (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
                  padding=(1, 1))
)
(decoder): Decoder_smallNorb(
  (dense): Linear(in_features=16, out_features=1024, bias=True)
  (relu): ReLU(inplace)
  (reconst_layers1): Sequential(
    (0): BatchNorm2d(16, eps=1e-05, momentum=0.8, affine=True,
                     track_running_stats=True)
    (1): ConvTranspose2d(16, 64, kernel_size=(3, 3),
                         stride=(1, 1), padding=(1, 1))
  )
  (reconst_layers2): ConvTranspose2d(64, 32, kernel_size=(3, 3),
                                     stride=(2, 2), padding=(1, 1))
```

```
    (reconst_layers3): ConvTranspose2d(32, 16, kernel_size=(3, 3),
                                       stride=(2, 2), padding=(1, 1))
    (reconst_layers4): ConvTranspose2d(16, 1, kernel_size=(3, 3),
                                       stride=(1, 1), padding=(1, 1))
    (reconst_layers5): ReLU()
  )
  (flatCaps): FlattenCaps()
  (digCaps): CapsuleLayer()
  (capsToScalars): CapsToScalars()
  (mask): Mask_CID()
  (mse_loss): MSELoss()
)
```

Note that for regularization, we considered two capsule layers here namely:

i) `conv3dCaps4_nj_8` and

ii) `digCaps`

There are 32 and 15 capsule-types in them respectively. However, in this thesis we tabulate entropy values for `conv3dCaps4_nj_8` layer only.

The transformation applied to the data is same for both the models. For training, we resized each image to (48×48), random-cropped a (32×32) patch, balanced brightness and contrast and normalized to mean 0.0 and standard deviation 0.3081. For validation and testing, we resized each image to (48×48), center-cropped the (32×32) patch and normalized to mean 0.0 and standard deviation 0.3081.

The losses used are the convex combinations of the usual loss and regularization loss. For the first model, the loss function used for the usual loss is cross-entropy loss. The loss function used for the second model is mean of the sum of margin loss and reconstruction loss(with a weight of 0.01) over the batch. The reconstruction loss uses MSE loss between the original and the reconstructed image. The weights considered for the convex combinations are 0.2, 0.4 and 0.6.

Adam optimizer [34] was used for the both the models is with the learning rate initialized as 0.001. We also used multistep LR scheduler with scheduling at epochs 100 and 200 and gamma as 0.1. The training is done for 250 epochs. For the second model, the scheduler used is initial LR times the $initialLR \times 0.5^{\frac{epoch}{10}}$ where the division is integer division.

3.4 Results on SmallNorb

The results for the accuracies are shown in table 3.2. We see that almost always impurity regularization improves the accuracy over the unregularized case. Though we do not intend to compare different impurity regularizations, in the current work, we observe that in case of SOVNet type model, gini index based regularization gives best results while for the DeepCaps model, entropy regularization gives best results. Please note that in the figures below, *regNum* is used to depict that *reg* regularization is used with its weight in the loss function being $Num \times 0.1$. For example, gini4 means the model is trained with gini index as regularization with its weight being 0.4. We also tabulated gini index and misclassification error values for all the models. They show a similar trend when compared with the unregularized version. Figures 3.11 - 3.40 are for the first model. Tables 3.3 - 3.7 are for the ConvCaps3D layer of the DeepCaps model.

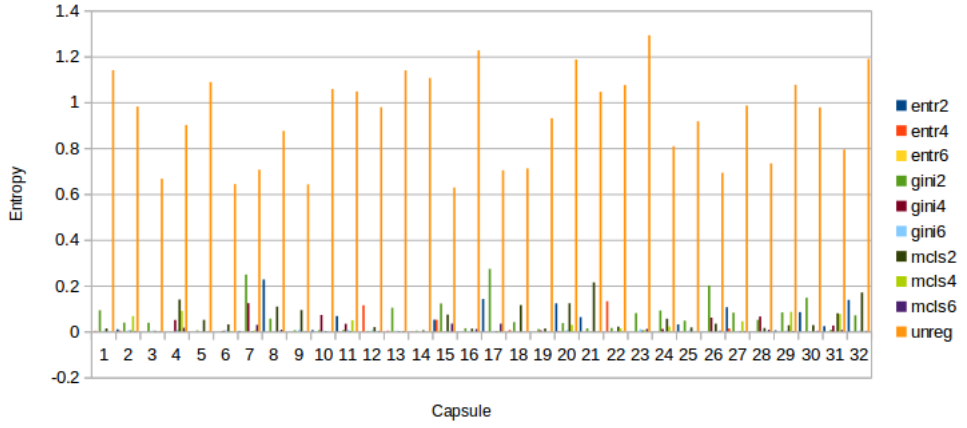


FIGURE 3.11: Average entropy for layer 1 for images of class 0 (SOVNet based model)

Regularization	Model	Accuracy
unreg	ResNet-CNN-SOVNet	88.265
entr_0.2	ResNet-CNN-SOVNet	90.385
entr_0.4	ResNet-CNN-SOVNet	90.177
entr_0.6	ResNet-CNN-SOVNet	90.615
entr_0.8	ResNet-CNN-SOVNet	89.854
gini_0.2	ResNet-CNN-SOVNet	87.477
gini_0.4	ResNet-CNN-SOVNet	90.576
gini_0.6	ResNet-CNN-SOVNet	90.866
gini_0.8	ResNet-CNN-SOVNet	92.914
mcls_0.2	ResNet-CNN-SOVNet	90.340
mcls_0.4	ResNet-CNN-SOVNet	90.023
mcls_0.6	ResNet-CNN-SOVNet	90.444
mcls_0.8	ResNet-CNN-SOVNet	—
unreg	DeepCaps	91.804
entr_0.2	DeepCaps	91.298
entr_0.4	DeepCaps	92.782
entr_0.6	DeepCaps	94.648
entr_0.8	DeepCaps	94.981
gini_0.2	DeepCaps	93.384
gini_0.4	DeepCaps	93.070
gini_0.6	DeepCaps	93.220
gini_0.8	DeepCaps	94.218
mcls_0.2	DeepCaps	93.374
mcls_0.4	DeepCaps	93.031
mcls_0.6	DeepCaps	94.132
mcls_0.8	DeepCaps	—

TABLE 3.2: Implementation Results

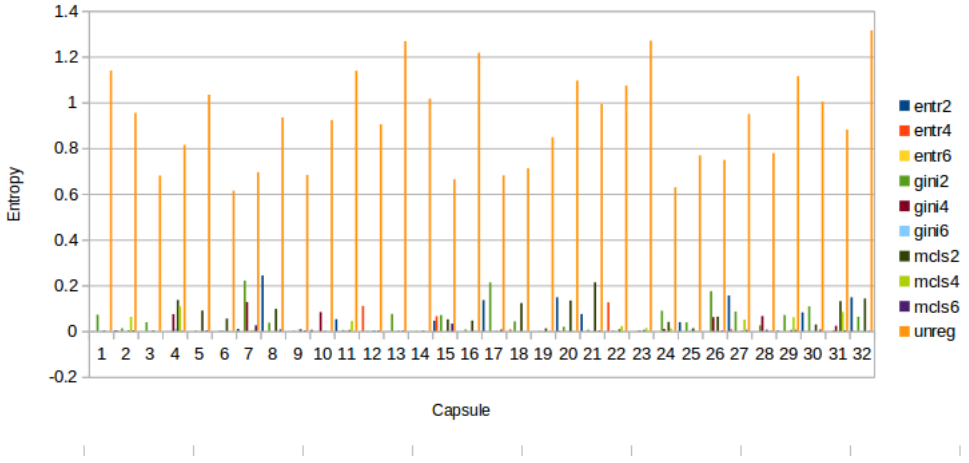


FIGURE 3.12: Average entropy for layer 1 capsules for images of class 1 (SOVNet based model)

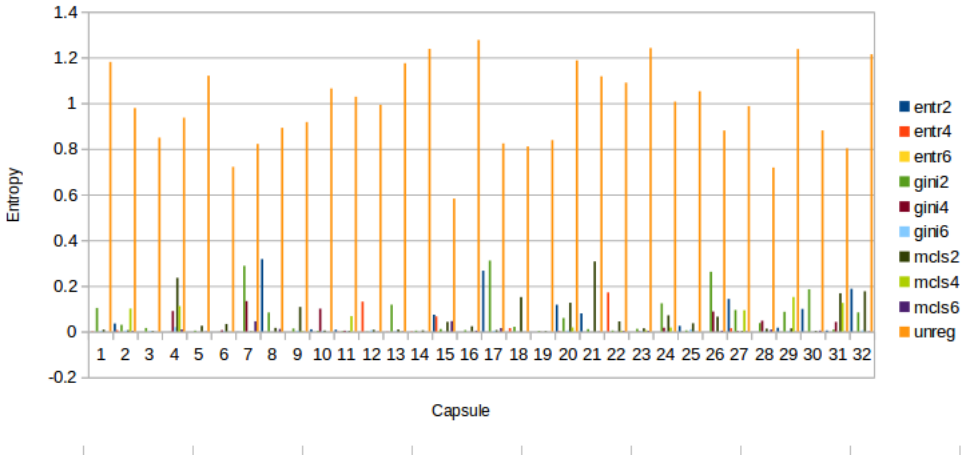


FIGURE 3.13: Average entropy for layer 1 capsules for images of class 2 (SOVNet based model)

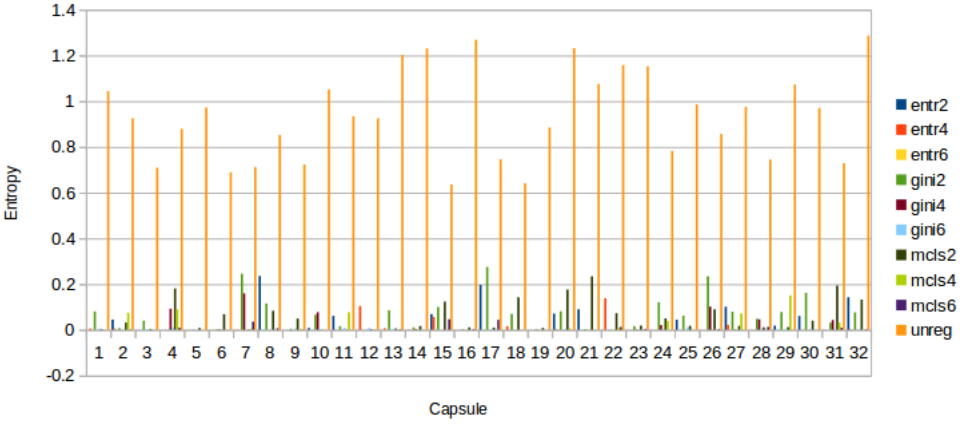


FIGURE 3.14: Average entropy for layer 1 capsules for images of class 3 (SOVNet based model)

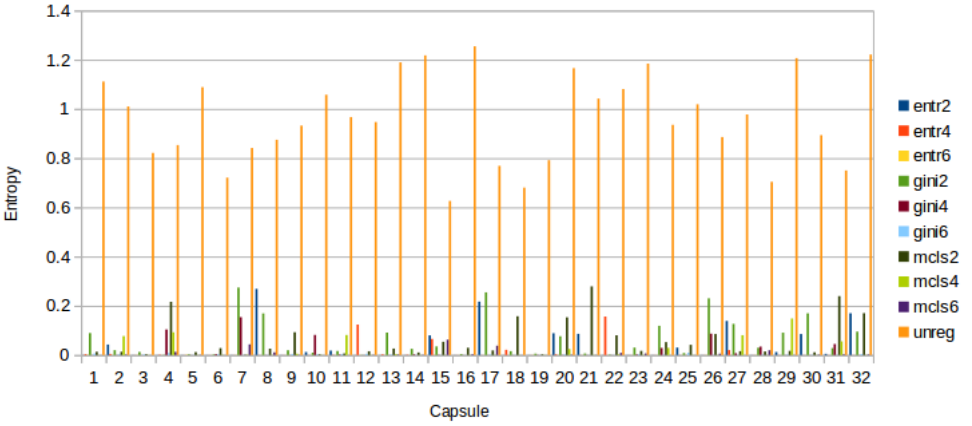


FIGURE 3.15: Average entropy for layer 1 capsules for images of class 4 (SOVNet based model)

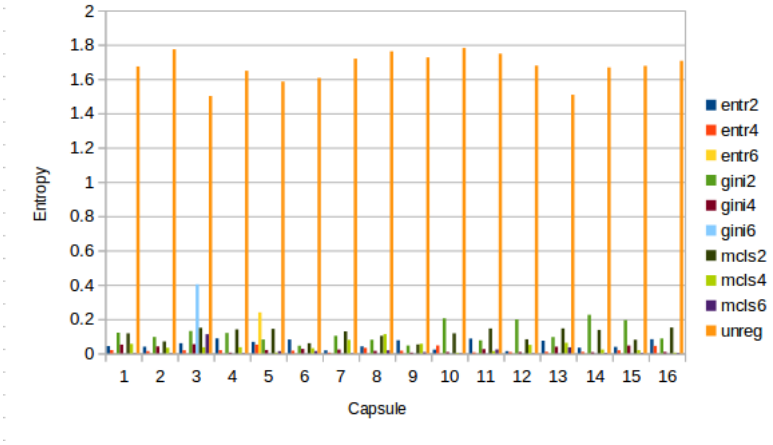


FIGURE 3.16: Average entropy for layer 2 for images of class 0 (SOVNet based model)

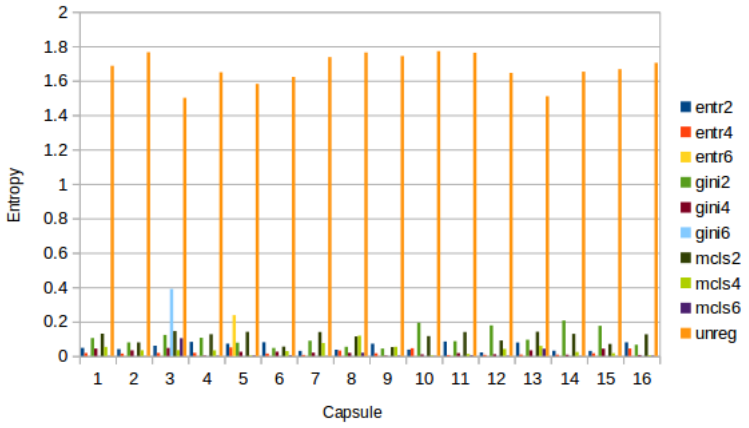


FIGURE 3.17: Average entropy for layer 2 capsules for images of class 1 (SOVNet based model)

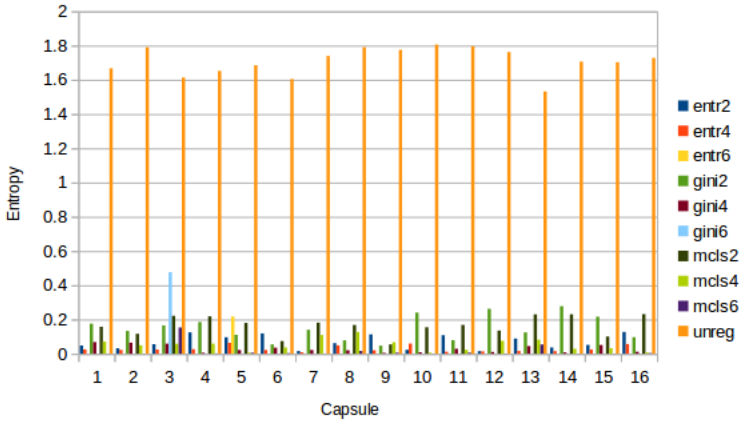


FIGURE 3.18: Average entropy for layer 2 capsules for images of class 2 (SOVNet based model)

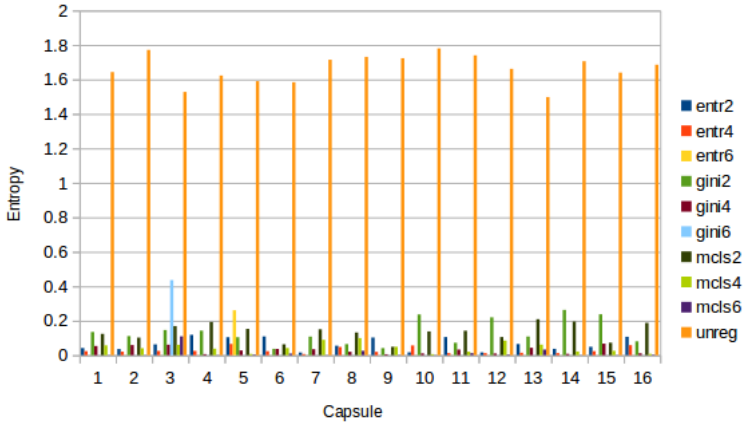


FIGURE 3.19: Average entropy for layer 2 capsules for images of class 3 (SOVNet based model)

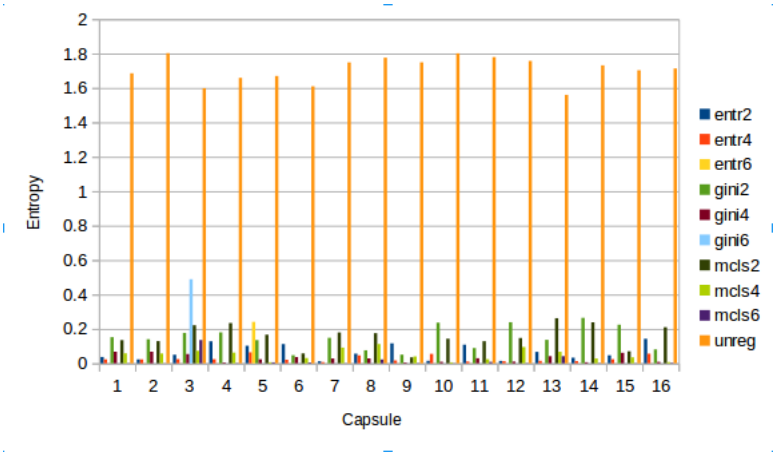


FIGURE 3.20: Average entropy for layer 2 capsules for images of class 4 (SOVNet based model)

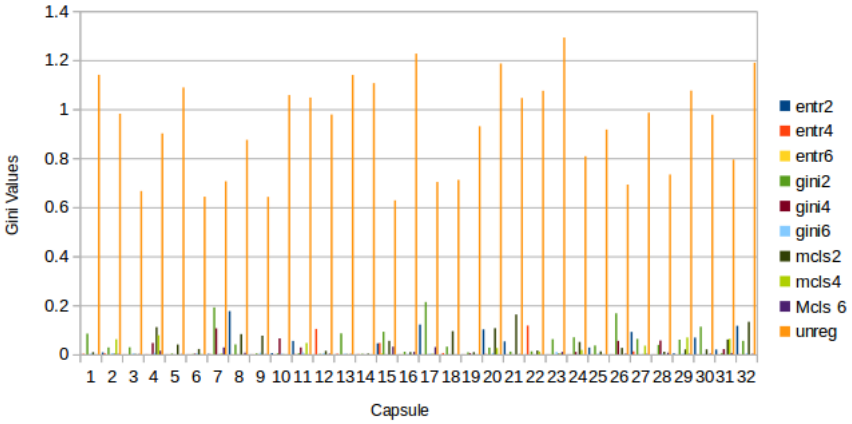


FIGURE 3.21: Average gini for layer 1 for images of class 0 (SOVNet based model)

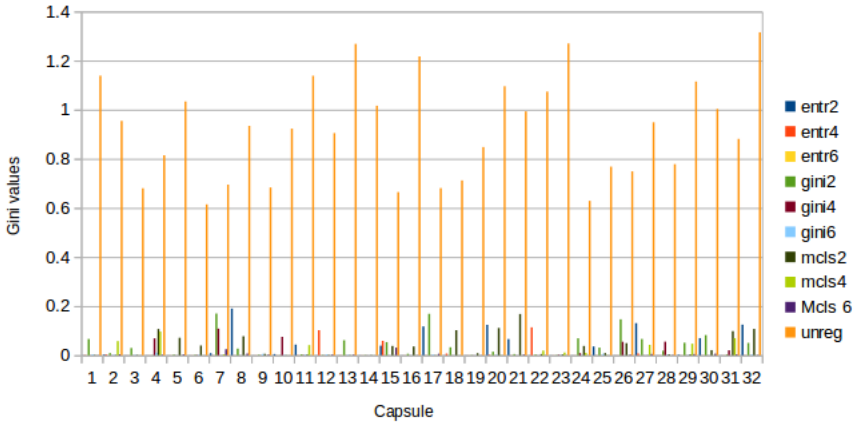


FIGURE 3.22: Average gini for layer 1 capsules for images of class 1 (SOVNet based model)

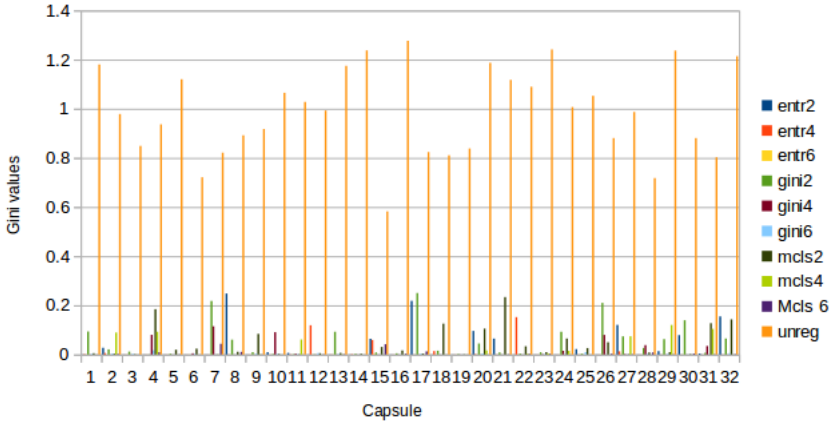


FIGURE 3.23: Average gini for layer 1 capsules for images of class 2 (SOVNet based model)

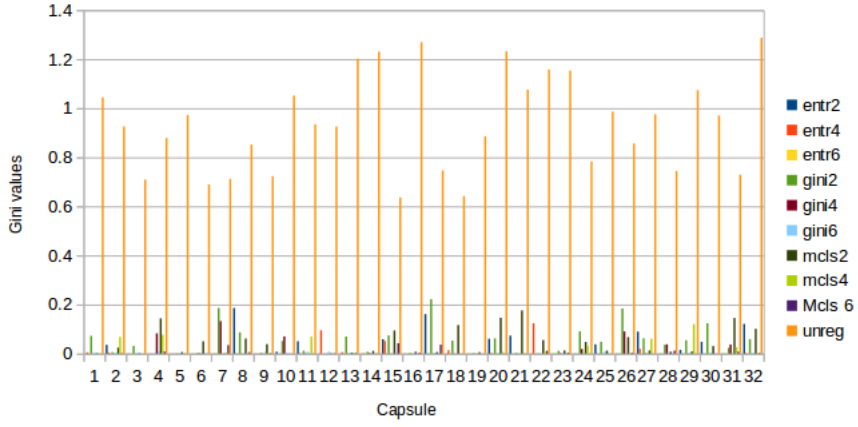


FIGURE 3.24: Average gini for layer 1 capsules for images of class 3 (SOVNet based model)

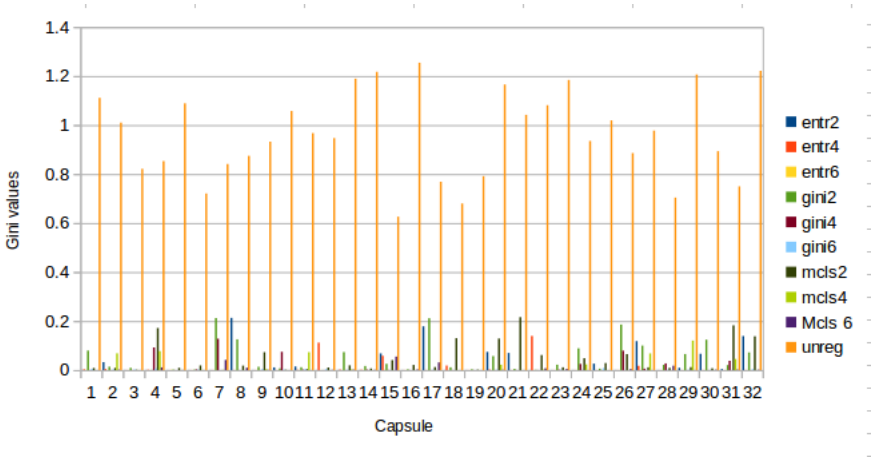


FIGURE 3.25: Average gini for layer 1 capsules for images of class 4 (SOVNet based model)

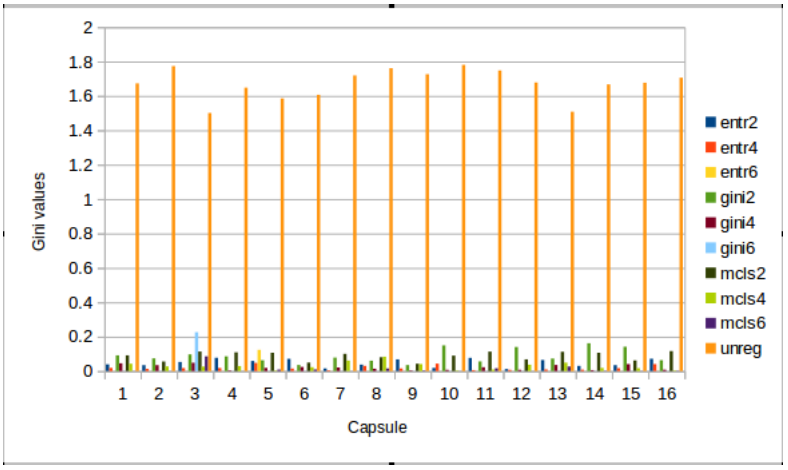


FIGURE 3.26: Average gini for layer 2 for images of class 0 (SOVNet based model)

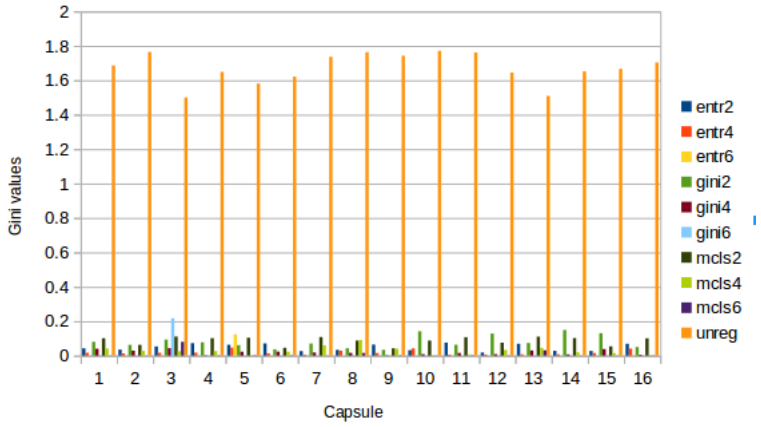


FIGURE 3.27: Average gini for layer 2 capsules for images of class 1 (SOVNet based model)

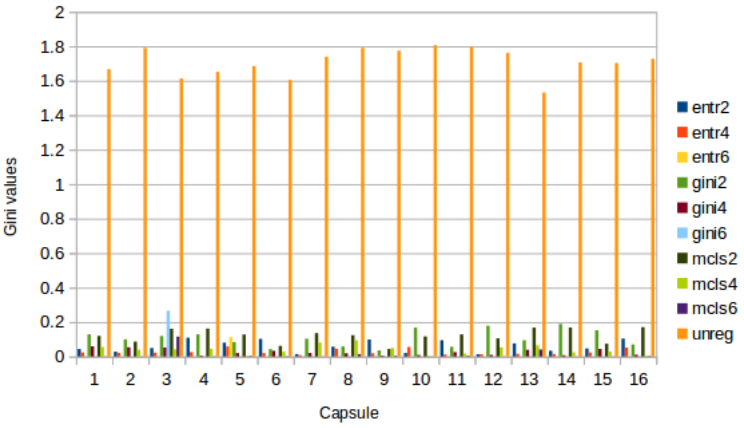


FIGURE 3.28: Average gini for layer 2 capsules for images of class 2 (SOVNet based model)

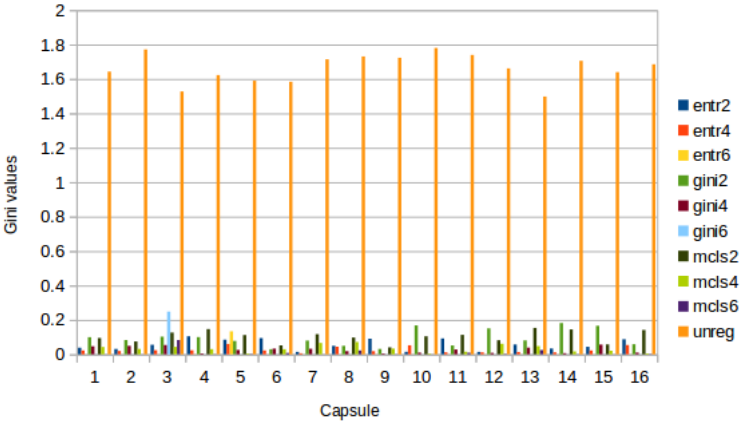


FIGURE 3.29: Average gini for layer 2 capsules for images of class 3 (SOVNet based model)

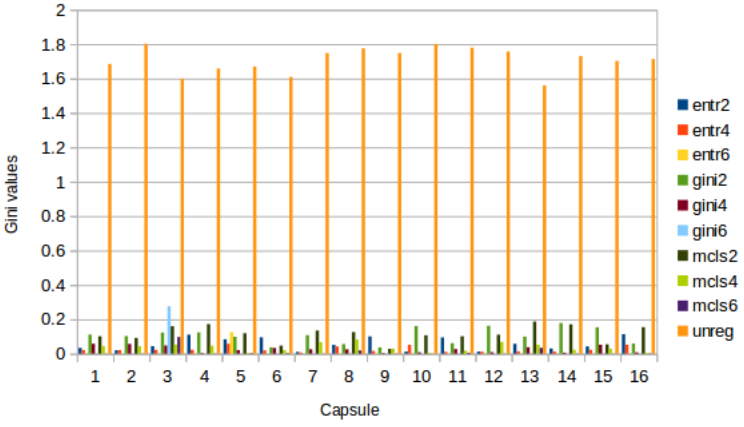


FIGURE 3.30: Average gini for layer 2 capsules for images of class 4 (SOVNet based model)

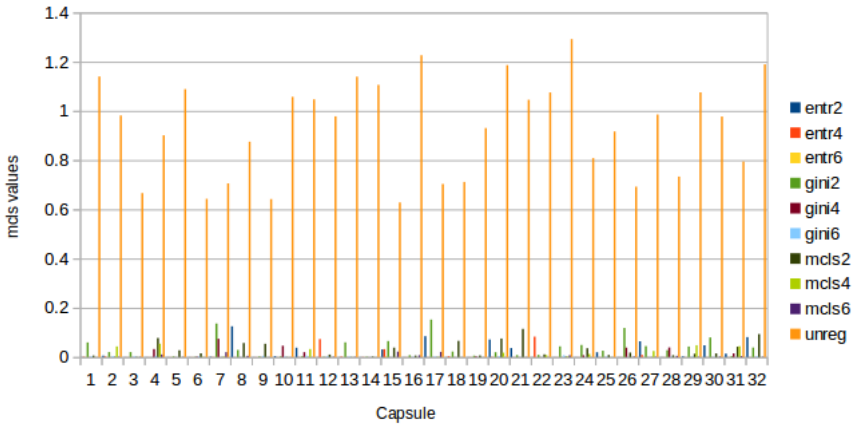


FIGURE 3.31: Average mcls for layer 1 for images of class 0 (SOVNet based model)

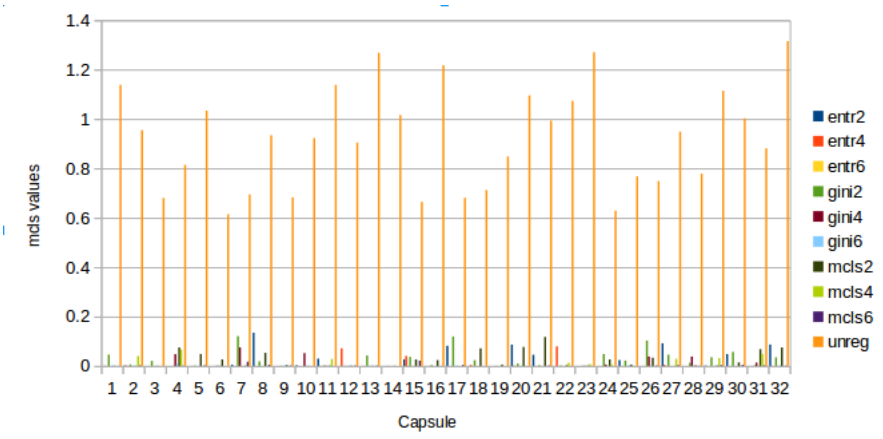


FIGURE 3.32: Average mcls for layer 1 capsules for images of class 1 (SOVNet based model)

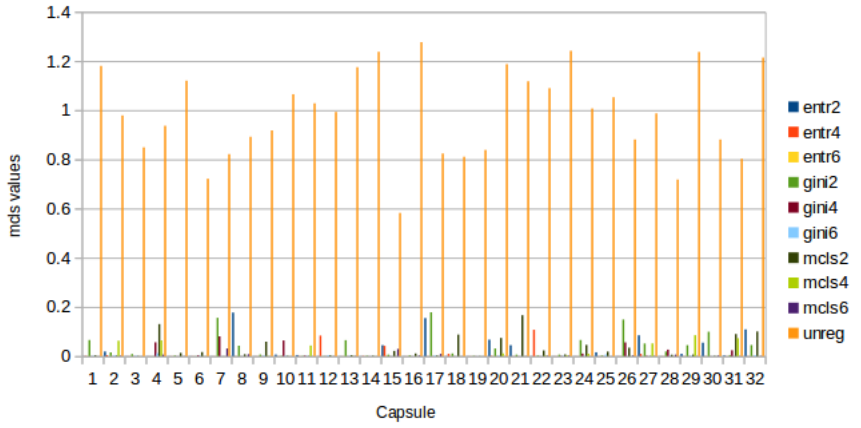


FIGURE 3.33: Average mcls for layer 1 capsules for images of class 2 (SOVNet based model)

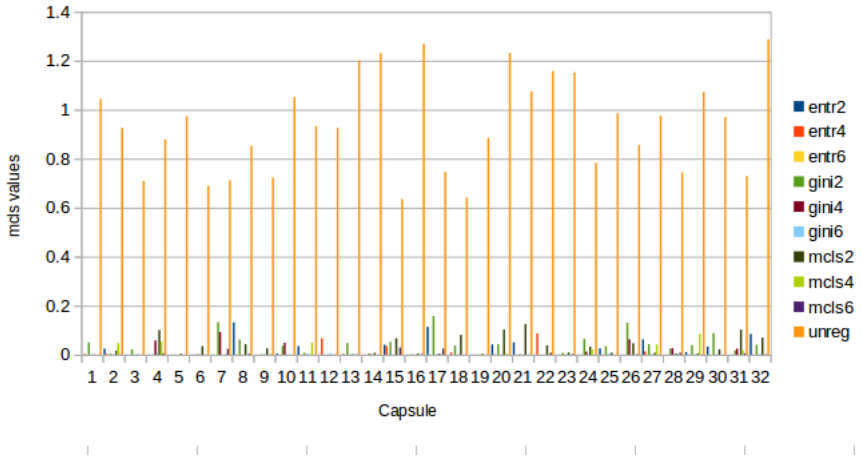


FIGURE 3.34: Average mcls for layer 1 capsules for images of class 3 (SOVNet based model)

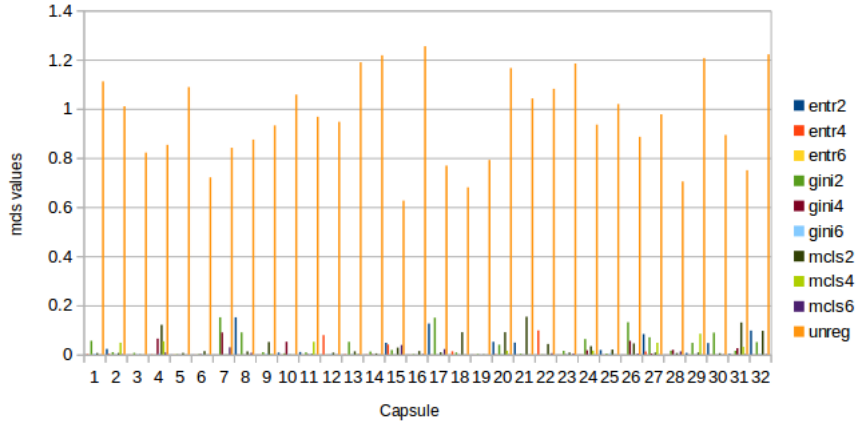


FIGURE 3.35: Average mcls for layer 1 capsules for images of class 4 (SOVNet based model)

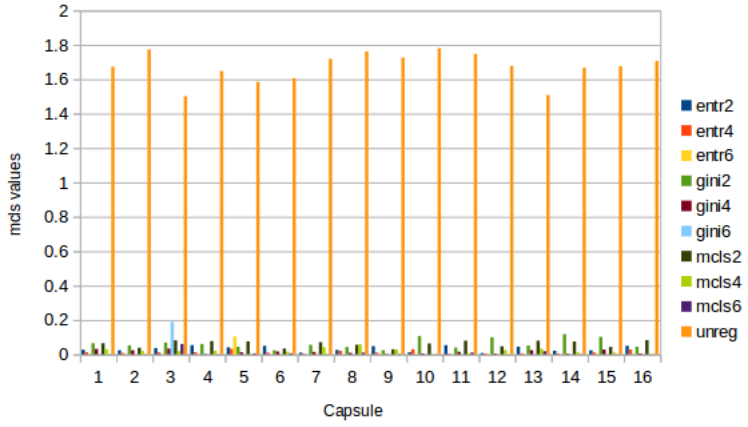


FIGURE 3.36: Average mcls for layer 2 for images of class 0 (SOVNet based model)

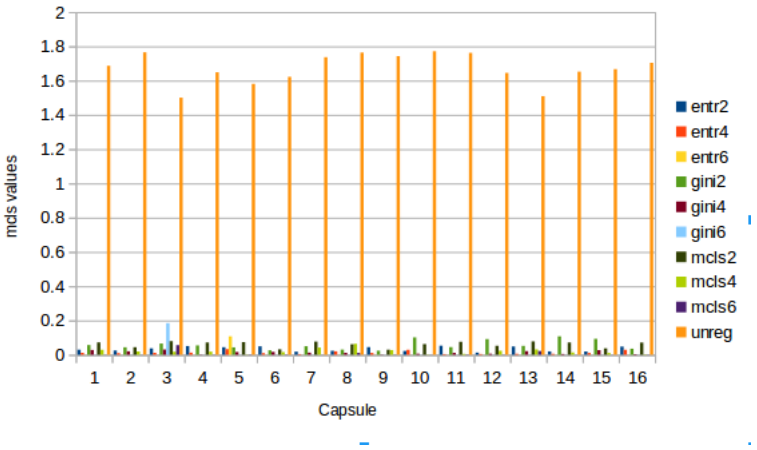


FIGURE 3.37: Average mcls for layer 2 capsules for images of class 1 (SOVNet based model)

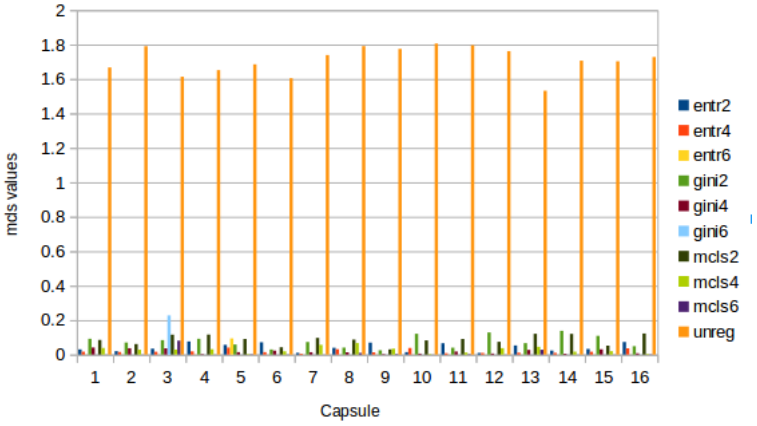


FIGURE 3.38: Average mcls for layer 2 capsules for images of class 2 (SOVNet based model)

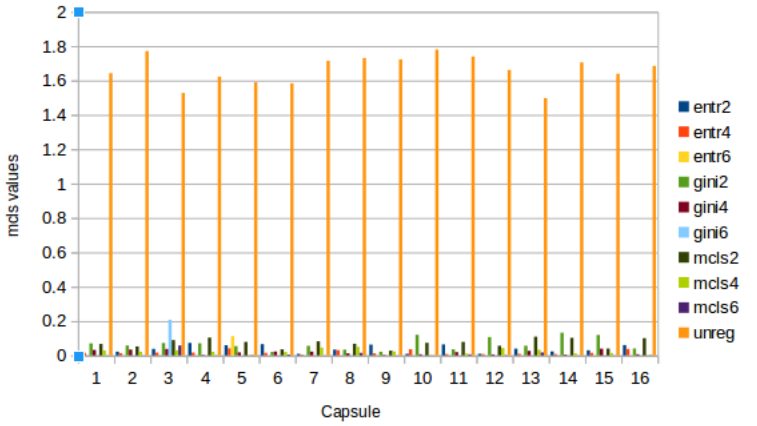


FIGURE 3.39: Average mcls for layer 2 capsules for images of class 3 (SOVNet based model)

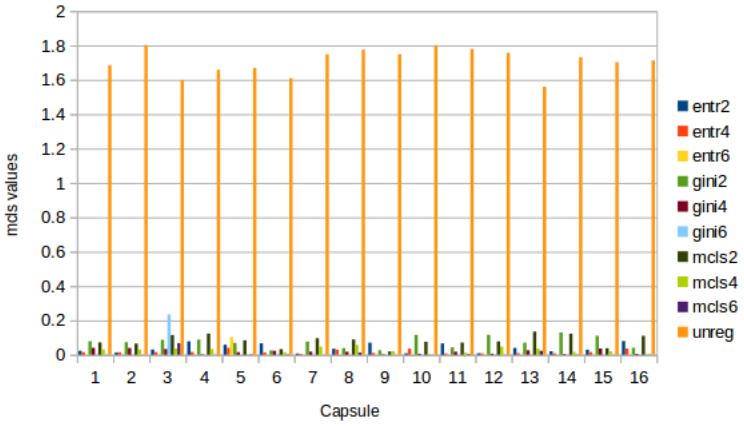


FIGURE 3.40: Average mcls for layer 2 capsules for images of class 4 (SOVNet based model)

	deepcaps	cl0	lyr1				
	entr2	entr4	gini2	gini4	mcls2	mcls4	unreg
1	0.7537378073	0.7535468936	0.7537697554	0.7537694573	0.0014143845	0.0001715288	0.7537697554
2	0.7537293434	0.7535469532	0.7537697554	0.7537629604	0.0009295269	0.0004512182	0.7537697554
3	0.753652215	0.7535466552	0.7537697554	0.7537683845	0.0010109375	0.0016545465	0.7537697554
4	0.7536088824	0.7535466552	0.7537697554	0.7537690997	0.0005372515	0.0023085391	0.7537697554
5	0.7537378669	0.7535470128	0.7537682056	0.7537689209	0.0012777641	1.14343320092303E-05	0.7537697554
6	0.7537043095	0.753547132	0.7537697554	0.75376755	0.0017301919	0.0026622894	0.7537697554
7	0.7537540197	0.753547132	0.7537696958	0.753764987	0.0011944043	0.0009422108	0.7537697554
8	0.7537456751	0.7535470724	0.7537697554	0.753757596	0.0079616942	0.000294794	0.7537697554
9	0.7536408305	0.7535470724	0.7537674308	0.7537683249	0.0128474142	1.0631945770001E-05	0.7537697554
10	0.7536033988	0.7535468936	0.7537697554	0.7537693977	0.0033590109	0.0001385825	0.7537697554
11	0.7536669374	0.7535467148	0.7537697554	0.7537597418	0.0023465671	0.000331562	0.7537697554
12	0.7537434697	0.7535467148	0.7537697554	0.7537689805	0.001274962	1.30993812490487E-05	0.7537697554
13	0.7536290884	0.7535473704	0.7537697554	0.7537661195	0.0006285568	6.76892450428568E-05	0.7537697554
14	0.7535945177	0.7535470128	0.7537697554	0.75373739	0.0010555306	0.0005727461	0.7537697554
15	0.7537626624	0.7535467744	0.7537697554	0.7537674308	0.0012437076	7.51842708268669E-06	0.7537697554
16	0.7537494302	0.753546834	0.7537697554	0.7536296248	0.0009344888	0.00040853	0.7537697554
17	0.7537497878	0.7535470128	0.7537697554	0.7536395192	0.0011491806	0.0003168524	0.7537697554
18	0.7537621856	0.7535430193	0.7537683845	0.7537341714	0.0036243552	6.52586240903474E-05	0.7537697554
19	0.7537567616	0.7535465956	0.7537697554	0.7537465692	0.0004046242	0.0003851641	0.7537697554
20	0.753583312	0.7535470724	0.7537697554	0.7537306547	0.0017684153	0.0003204735	0.7537697554
21	0.7537546158	0.7535465956	0.7537697554	0.7537692189	0.0032363771	0.0007372848	0.7537697554
22	0.7537400723	0.7535475492	0.7537697554	0.7537676096	0.0008737185	0.0026459815	0.7537697554
23	0.7537552714	0.7535470724	0.7537697554	0.7537694573	0.0018478067	0.0002275281	0.7537697554
24	0.753749907	0.7535478473	0.7537697554	0.7537065744	0.0015378689	0.0012699338	0.7537697554
25	0.7536430955	0.7535472512	0.7537697554	0.7537603378	0.001516988	0.0002814024	0.7537697554
26	0.7537693381	0.7535471916	0.7537697554	0.7537695169	0.0007744673	0.0002339975	0.7537697554
27	0.7537517548	0.7535467148	0.7537697554	0.7537359595	0.0766470283	1.12770694613573E-05	0.7537697554
28	0.7537207007	0.7535471916	0.7537697554	0.7536688447	0.0007909807	0.0003633611	0.7537697554
29	0.7537606955	0.7535468936	0.7537697554	0.7537639141	0.0018435472	0.0010352919	0.7537697554
30	0.7535564899	0.7535470128	0.7537697554	0.7537621856	0.001251532	0.0001227856	0.7537697554
31	0.7537377477	0.753547132	0.7537697554	0.7537673116	0.0037137282	0.0002433534	0.7537697554
32	0.7536771894	0.7535468936	0.7537697554	0.7537676096	0.0389176309	0.0002446436	0.7537697554

TABLE 3.3: DeepCaps conv-caps3DLayer entropies for class 0

	deepcaps	cl1	lyr1				
	entr2	entr4	gini2	gini4	mcls2	mcls4	unreg
1	0.7537662387	0.7535471916	0.7537697554	0.7537527084	0.0042511942	0.0001292539	0.7537697554
2	0.7537651658	0.7535465956	0.7537697554	0.7536436319	0.0300299693	0.005818285	0.7537697554
3	0.7536645532	0.7535465956	0.7537697554	0.7537685037	0.0020731289	0.0035880583	0.7537697554
4	0.7537692189	0.7535461187	0.7537697554	0.7537299395	0.0086554158	0.0001466552	0.7537697554
5	0.7537690401	0.7535465956	0.7535867095	0.7537383437	0.0052293194	4.17188130086288E-05	0.7537697554
6	0.753582418	0.753546834	0.7537697554	0.7537145615	0.0018592058	0.000169949	0.7537697554
7	0.7537686825	0.7535469532	0.7537369728	0.7536865473	0.0576472171	0.0102183996	0.7537697554
8	0.7537658811	0.7535465956	0.7537697554	0.7537174821	0.0015555886	0.0003501005	0.7537697554
9	0.7537648678	0.7535470128	0.7537696362	0.7537332177	0.002457862	1.33456869662041E-05	0.7537697554
10	0.7536005974	0.7535467148	0.7537697554	0.7537592649	0.0020578692	0.0010748412	0.7537697554
11	0.7537696362	0.753547132	0.7537697554	0.7537364364	0.0019932676	0.0001183895	0.7537697554
12	0.7537670732	0.7535470128	0.7537697554	0.7537169456	0.0064189783	3.25128967233468E-05	0.7537697554
13	0.7537693977	0.7535467148	0.7537697554	0.753641963	0.0157304183	0.0011789722	0.7537697554
14	0.7537039518	0.7535470724	0.7537697554	0.7535545826	0.0831095204	0.0008139595	0.7537697554
15	0.7537688017	0.7535465956	0.7537697554	0.7536793947	0.0033374853	7.47036619941355E-06	0.7537697554
16	0.7537693381	0.7535463572	0.7537696362	0.7537327409	0.033820238	0.0083293216	0.7537697554
17	0.7537683845	0.7535468936	0.7537697554	0.7537373304	0.0021393641	0.0025592663	0.7537697554
18	0.7537689805	0.7535477877	0.7536052465	0.753554523	0.0050446773	3.77144533558749E-05	0.7537697554
19	0.7537692785	0.7535468936	0.7537697554	0.753749609	0.027561307	0.001475375	0.7537697554
20	0.7537457347	0.7535468936	0.7537697554	0.7536685467	0.0011856254	0.0030245997	0.7537697554
21	0.7537696362	0.7535465956	0.7537697554	0.7537507415	0.0084596165	0.0008641552	0.7537697554
22	0.7537688017	0.7535464168	0.7537697554	0.7537606955	0.0058856099	5.29653370904271E-05	0.7537697554
23	0.7537667155	0.753546834	0.7537697554	0.7537494302	0.0046007507	0.0013103116	0.7537697554
24	0.7537682056	0.7535462976	0.7537697554	0.7535552979	0.0021196418	0.0018448314	0.7537697554
25	0.7537695765	0.7535464764	0.7537697554	0.7537367344	0.0024184233	0.0012068246	0.7537697554
26	0.7537463903	0.7535470724	0.7537697554	0.753750205	0.0143843256	0.0034571118	0.7537697554
27	0.7537693977	0.7535468936	0.7537697554	0.7536796927	0.0060306275	0.0002614442	0.7537697554
28	0.7537626028	0.7535475492	0.7537697554	0.753765285	0.004732525	0.0009751699	0.7537697554
29	0.7537696958	0.7535465956	0.7537697554	0.7535840273	0.0058921129	0.0068734237	0.7537697554
30	0.7535544634	0.7535467744	0.7537697554	0.7536043525	0.010520923	0.0016329662	0.7537697554
31	0.7537670732	0.7535472512	0.7537697554	0.7537555695	0.0032095467	0.0016516933	0.7537697554
32	0.7537204027	0.7535467148	0.7537697554	0.753654778	0.0073079602	0.0013963337	0.7537697554

TABLE 3.4: DeepCaps conv-caps3DLayer entropies for class 1

	deepcaps	cl2	lyr1				
	entr2	entr4	gini2	gini4	mcls2	mcls4	unreg
1	0.75372684	0.7535466552	0.7537697554	0.7537697554	0.0049189711	0.0009101058	0.7537697554
2	0.7537035346	0.7535465956	0.7537697554	0.7537697554	0.0050986758	0.0002403513	0.7537697554
3	0.7536484003	0.7535465956	0.7537697554	0.7537696958	0.0585193411	0.0024721448	0.7537697554
4	0.7536839843	0.7535102963	0.7537697554	0.7537697554	0.0022221555	0.0008555031	0.7537697554
5	0.7536903024	0.753546536	0.7537697554	0.7537696958	0.009181425	0.0002626758	0.7537697554
6	0.7535868287	0.7535464168	0.7537697554	0.7537685037	0.0023605246	0.0019001837	0.7537697554
7	0.7536367178	0.7535464764	0.7537696958	0.7537680864	0.0014298714	0.0015515598	0.7537697554
8	0.7537453771	0.7535465956	0.7537697554	0.7537389994	0.004985522	0.0005119761	0.7537697554
9	0.7535931468	0.7535462976	0.7537697554	0.7537697554	0.0035100717	0.000306918	0.7537697554
10	0.75356251	0.7535463572	0.7537697554	0.7537697554	0.0093883453	0.0045589334	0.7537697554
11	0.7537325025	0.7535468936	0.7537697554	0.7537696958	0.000925742	0.001156952	0.7537697554
12	0.7536867261	0.7535464168	0.7537697554	0.7537697554	0.0062771281	0.0001442858	0.7537697554
13	0.7535880804	0.7535464764	0.7537697554	0.7537696362	0.013793543	0.000119143	0.7537697554
14	0.7536587715	0.7535467148	0.7537697554	0.7537689805	0.0013541715	0.0007204433	0.7537697554
15	0.7537146211	0.7535464168	0.7537697554	0.7537697554	0.0019954694	0.0002498984	0.7537697554
16	0.7537077665	0.7535467148	0.753764689	0.7536042929	0.0021499447	0.001168379	0.7537697554
17	0.7536961436	0.7535465956	0.7537697554	0.7537696362	0.0046995259	0.0007196693	0.7537697554
18	0.7537105083	0.7535473108	0.7537633777	0.7537691593	0.0051223636	9.27254659472965E-05	0.7537697554
19	0.7537052035	0.753546834	0.7537697554	0.7537696958	0.012230726	0.0914068818	0.7537697554
20	0.7536877394	0.7535467148	0.7537697554	0.7537697554	0.0162618849	0.0007551789	0.7537697554
21	0.7537333369	0.7535464168	0.7537697554	0.7537697554	0.0104395011	0.000480206	0.7537697554
22	0.7536494732	0.7535465956	0.7537697554	0.7537697554	0.0019981668	0.0009829204	0.7537697554
23	0.7536777258	0.753546536	0.7537697554	0.7537697554	0.0037953961	0.0013110698	0.7537697554
24	0.7537012696	0.7535472512	0.7537697554	0.7537692189	0.0048547983	0.0011551027	0.7537697554
25	0.7536625266	0.7535464764	0.7537697554	0.7537697554	0.0092511736	0.0046306662	0.7537697554
26	0.7537102699	0.7535468936	0.7537697554	0.7537697554	0.0207535569	0.0812523961	0.7537697554
27	0.7536517382	0.753546536	0.7537697554	0.7537697554	0.0030558342	0.0007860739	0.7537697554
28	0.753695786	0.753546834	0.7537697554	0.7537696362	0.1385338455	0.052287221	0.7537697554
29	0.7537117004	0.7535390258	0.7537697554	0.7537696958	0.0039497372	0.0003239501	0.7537697554
30	0.7535676956	0.753546536	0.7537697554	0.7537696958	0.0151068028	0.0029342643	0.7537697554
31	0.7537071705	0.7535468936	0.7537697554	0.7537686825	0.0026517243	0.0001429533	0.7537697554
32	0.7536600828	0.7535460591	0.7537697554	0.7537697554	0.0095782727	0.0004534529	0.7537697554

TABLE 3.5: DeepCaps conv-caps3DLayer entropies for class 2

	deepcaps	cl3	lyr1				
	entr2	entr4	gini2	gini4	mcls2	mcls4	unreg
1	0.75373739	0.7535464764	0.7537697554	0.7537696958	0.0032542264	0.0010747925	0.7537697554
2	0.7537365556	0.7535467148	0.7537697554	0.7537593842	0.0851413757	0.0012578656	0.7537697554
3	0.7535791993	0.7535464764	0.7537697554	0.7537642121	0.0011014738	0.0018201993	0.7537697554
4	0.7537668943	0.7535468936	0.7537697554	0.7537696958	0.0039710836	0.0081163021	0.7537697554
5	0.7537642121	0.753546536	0.753739655	0.7537696958	0.0033136681	4.86071039631497E-05	0.7537697554
6	0.7537251711	0.7535464168	0.7537697554	0.7537615895	0.0055719293	0.0041381139	0.7537697554
7	0.7537004352	0.7535464168	0.7536597252	0.7537484169	0.0032880951	0.005454706	0.7537697554
8	0.7537438869	0.7535464168	0.7537697554	0.7537690997	0.0027673771	0.0013133828	0.7537697554
9	0.7537582517	0.7535464168	0.7537697554	0.7537692189	0.0020804417	3.28373571392149E-05	0.7537697554
10	0.7537395954	0.7535465956	0.7537697554	0.7537513375	0.0056148567	0.0002811498	0.7537697554
11	0.7537548542	0.753546834	0.7537697554	0.7536480427	0.0010209056	0.001137943	0.7537697554
12	0.7537591457	0.7535464168	0.7537697554	0.7537695765	0.0066992608	8.59293577377684E-05	0.7537697554
13	0.7536779642	0.7535468936	0.7537697554	0.7537690997	0.0091524534	0.0021486564	0.7537697554
14	0.7535544038	0.7535445094	0.7537697554	0.7537187934	0.043229688	0.0022868805	0.7537697554
15	0.7537490726	0.7535464168	0.7537697554	0.7537696958	0.0363100171	0.0001072505	0.7537697554
16	0.7537620664	0.753546536	0.7537695169	0.7537697554	0.0294710416	0.0001252923	0.7537697554
17	0.7537482381	0.7535464764	0.7537697554	0.7537696958	0.0054771295	0.0009361937	0.7537697554
18	0.7537124157	0.7535480261	0.7535903454	0.7537372708	0.0028689748	0.0052555604	0.7537697554
19	0.7536466718	0.7535466552	0.7537697554	0.7537696958	0.0017911155	1.03766897154856E-05	0.7537697554
20	0.7537665963	0.753544271	0.7537697554	0.7537620068	0.0059136553	0.0005703105	0.7537697554
21	0.7537468076	0.7535467744	0.7537697554	0.7537696362	0.0089227855	0.0146431671	0.7537697554
22	0.7537340522	0.7535465956	0.7537199259	0.7537682056	0.0304415394	0.0002987939	0.7537697554
23	0.7536594272	0.7535459399	0.7537697554	0.7537693977	0.0046092384	0.0003872694	0.7537697554
24	0.753721714	0.7534919381	0.7537697554	0.7537044883	0.0027942148	0.0056616967	0.7537697554
25	0.7537689805	0.753546536	0.7537697554	0.7537670732	0.002329444	0.0019018186	0.7537697554
26	0.7535850406	0.7535464764	0.7537673712	0.7537696958	0.0203061383	0.0014516148	0.7537697554
27	0.7537311912	0.7535464764	0.7537697554	0.7535738349	0.0018352146	0.0079297991	0.7537697554
28	0.7537264824	0.753546536	0.7537697554	0.7537621856	0.003211326	0.0020342211	0.7537697554
29	0.7537465096	0.7535466552	0.7537696958	0.7537599802	0.0129530327	0.0297714602	0.7537697554
30	0.7537686825	0.7535464168	0.7537696958	0.7537696958	0.0097655701	4.90418206027243E-05	0.7537697554
31	0.7537259459	0.7535464168	0.7537697554	0.753741622	0.0024995478	0.0003489459	0.7537697554
32	0.7537106276	0.7535464764	0.7537697554	0.7537676096	0.0056520388	0.0490457676	0.7537697554

TABLE 3.6: DeepCaps conv-caps3DLayer entropies for class 3

	deepcaps	cl4	lyr1				
	entr2	entr4	gini2	gini4	mcls2	mcls4	unreg
1	0.7536866069	0.7535469532	0.7537697554	0.7537697554	0.0025546744	0.0008087113	0.7537697554
2	0.7537225485	0.753546834	0.7537697554	0.7537696958	0.0819577873	0.0003987217	0.7537697554
3	0.753695786	0.753546834	0.7537697554	0.7537645102	0.000966039	0.0007999042	0.7537697554
4	0.7535925508	0.7535478473	0.7537697554	0.7537697554	0.0032966551	0.0136575187	0.7537697554
5	0.7537029386	0.7535468936	0.7537696362	0.7537665963	0.0040682736	1.40935844683554E-05	0.7537697554
6	0.7536598444	0.7535469532	0.7537697554	0.7537560463	0.0022833506	0.0004804009	0.7537697554
7	0.7536850572	0.7535467744	0.7537682056	0.7537648678	0.0049144374	0.0006159309	0.7537697554
8	0.753706634	0.7535469532	0.7537697554	0.7537693977	0.0054288255	0.0001018057	0.7537697554
9	0.7537159324	0.7535468936	0.7537697554	0.7537697554	0.0054132389	4.93601328344084E-05	0.7537697554
10	0.7536205649	0.7535469532	0.7537697554	0.7537696958	0.0035084332	0.0006385163	0.7537697554
11	0.7535978556	0.7535470128	0.7537697554	0.7537555695	0.0026440572	7.58977839723229E-05	0.7537697554
12	0.7537012696	0.7535468936	0.7537697554	0.7537696958	0.0079253763	1.09127768155304E-05	0.7537697554
13	0.7536787391	0.753547132	0.7537697554	0.7537697554	0.0047018928	0.0005390689	0.7537697554
14	0.7535549402	0.7535462379	0.7537697554	0.7537693381	0.0515019484	0.0002197543	0.7537697554
15	0.7537269592	0.753546834	0.7537697554	0.7537697554	0.030754054	3.49372894561384E-05	0.7537697554
16	0.7537336349	0.7535470724	0.7537697554	0.75369519	0.015365853	0.000108136	0.7537697554
17	0.7536161542	0.7535470128	0.7537697554	0.7537695169	0.0026587334	0.0008196821	0.7537697554
18	0.7536277771	0.7535474896	0.7537619472	0.7537696958	0.0028497805	2.14138017327059E-05	0.7537697554
19	0.7537379861	0.753546834	0.7537697554	0.7537693977	0.0011476184	1.13005007733591E-05	0.7537697554
20	0.7537444234	0.7535459399	0.7537697554	0.7537605762	0.0053031542	0.0008208949	0.7537697554
21	0.753708601	0.7535470724	0.7537697554	0.7537696958	0.0098489719	0.0007890055	0.7537697554
22	0.7536942363	0.7535469532	0.7537696362	0.7537697554	0.0151208807	0.0062103057	0.7537697554
23	0.7537074685	0.7535467148	0.7537697554	0.7537697554	0.0032726044	0.000837292	0.7537697554
24	0.7536994815	0.7535436153	0.7537697554	0.7537488937	0.0018671916	0.0004588876	0.7537697554
25	0.7537173033	0.7535470128	0.7537697554	0.7537645102	0.0017778787	0.0010233255	0.7537697554
26	0.7536177039	0.7535472512	0.7537697554	0.7537696958	0.0125508597	0.0001867577	0.7537697554
27	0.75369066	0.7535468936	0.7537697554	0.753708005	0.0017658806	0.0001182198	0.7537697554
28	0.75371629	0.7535466552	0.7537697554	0.7537693977	0.0053672711	0.0004620224	0.7537697554
29	0.7536809444	0.7535470128	0.7537697554	0.7537697554	0.0116258357	0.0008233539	0.7537697554
30	0.7537686229	0.7535468936	0.7537697554	0.7537697554	0.0077011059	0.0003231585	0.7537697554
31	0.7536913753	0.7535470128	0.7537697554	0.7537693381	0.0018924334	0.0007735405	0.7537697554
32	0.7537087202	0.7535469532	0.7537697554	0.7537696958	0.0055871648	0.002986439	0.7537697554

TABLE 3.7: DeepCaps conv-caps3DLayer entropies for class 4

Chapter 4

Conclusion and Future Work

We introduced a simple regularization for ensuring capsule-networks learn tree-like parent-child relationships among capsules of adjacent layers. We used impurity measures of the routing coefficients as the regularization term in the loss. The overall loss is a convex combination of the supervision loss and the regularization term. We tested our idea on FashionMNIST, KMNIST, and SmallNorb. An analysis of this regularization as seen in the test-time impurities of the routing coefficients showed that a more compositional representation is being learnt. Our results also suggest that our model works better with deeper networks and causes it perform better.

As future work, we intend to:

(i) test the regularization term for different routing algorithms.

(ii) test the performance of layer-wise trained models as the regularization term is sum of the layer-wise impurity measures.

(iii) test the performance of state-of-the-art models trained with the regularization term.

Bibliography

[1] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017. URL http://arxiv.org/abs/1710.09829.

[2] Yann Lecun and Yoshua Bengio. *Convolutional networks for images, speech, and time-series*. MIT Press, 1995.

[3] Jathushan Rajasegaran, Vinoj Jayasundara, Sandaru Jayasekara, Hirunima Jayasekara, Suranga Seneviratne, and Ranga Rodrigo. Deepcaps: Going deeper with capsule networks. *CoRR*, abs/1904.09546, 2019. URL http://arxiv.org/abs/1904.09546.

[4] Olga Davydova. 10 applications of artificial neural networks in natural language processing, 2017. URL https://medium.com/@datamonsters/artificial-neural-networks-in-natural-language-processing-bcf62aa9151a.

[5] Jonathan Hui. GAN - Some cool applications of GANs., 2018. URL https://medium.com/@jonathan_hui/gan-some-cool-applications-of-gans-4c9ecca35900.

[6] VUKU - an AI platform for more effective, faster decisions, 2018. URL https://www.prowler.io/platform.

[7] VUKU - An AI platform for more effective, faster decisions, 2018. URL https://www.prowler.io/platform.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira,

C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[11] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016.

[12] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://dblp.uni-trier.de/db/journals/corr/corr1311.html#ZeilerF13.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[15] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In *ICANN*, 2011.

[16] Roozbeh Yousefzadeh and Dianne P. O'Leary. Interpreting neural networks using flip points. *CoRR*, abs/1903.08789, 2019. URL http://arxiv.org/abs/1903.08789.

[17] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *CoRR*, abs/1706.07979, 2017. URL http://arxiv.org/abs/1706.07979.

[18] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. *CoRR*, abs/1710.00935, 2017. URL http://arxiv.org/abs/1710.00935.

[19] C.-C. Jay Kuo, Min Zhang, Siyang Li, Jiali Duan, and Yueru Chen. Interpretable convolutional neural networks via feedforward design. *Journal of Visual Communication and Image Representation*, 60:346 – 359, 2019. ISSN 1047-3203. doi: https://doi.org/10.1016/j.jvcir.2019.03.010. URL http://www.sciencedirect.com/science/article/pii/S104732031930104X.

[20] Sai Raam Venkataraman, S. Balasubramanian, and R. Raghunatha Sarma. Building deep equivariant capsule networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJgNJgSFPS.

[21] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[22] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

[23] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

[24] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.

[25] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[26] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJWLfGWRb.

[27] Mohammad Taha Bahadori. Spectral capsule networks. 2018. URL https://openreview.net/pdf?id=HJuMvYPaM.

[28] Jan Eric Lenssen, Matthias Fey, and Pascal Libuschewski. Group equivariant capsule networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8858–8867, Red Hook, NY, USA, 2018. Curran Associates Inc.

[29] Xinpeng Ding, Nannan Wang, Xinbo Gao, Jie Li, and Xiaoyu Wang. Group reconstruction and max-pooling residual capsule network. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2237–2243. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/310. URL https://doi.org/10.24963/ijcai.2019/310.

[30] Dilin Wang and Qiang Liu. An optimization view on dynamic routing between capsules, 2018. URL https://openreview.net/forum?id=HJjtFYJDf.

[31] Taewon Jeong, Youngmin Lee, and Heeyoung Kim. Ladder capsule network. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3071–3079, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/jeong19b.html.

[32] Assaf Hoogi, Brian Wilcox, Yachee Gupta, and Daniel L. Rubin. Self-attention capsule networks for image classification. *CoRR*, abs/1904.12483, 2019. URL http://arxiv.org/abs/1904.12483.

[33] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2: II–104 Vol.2, 2004.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.