Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehr- und Forschungsgebiet Informatik VIII
Computer Vision
Prof. Dr. Bastian Leibe

Seminar Report

Dynamic Routing Between Capsules

Mohammad Zeineldeen
Matriculation Number: 384051

August 2018

Advisor: Alexander Hermans

Abstract

Hinton argues that pooling layer in CNNs does not preserve the spatial relationship between the parts of the image which is important for object recognition. Then, he introduced the idea of capsules where a capsule is a group of neurons having an activity vector that represents the instantiation parameters of the object it is detecting. In addition, the length of this activity vector is the probability that the object exists in the image. These capsules use a routing-by-agreement algorithm to activate high-level capsules. Another version of capsules was introduced which is based on the Expected Maximization algorithm. In this version, a capsule has an activation to represent the existence of the object it is detecting and a pose matrix to learn the relationship between the object and the pose. Capsule networks achieved state-of-the-art performance on the MNIST dataset. Research work showed that using capsule networks can lead to better results in different application areas. In this report, we discuss the main concepts of the capsule theory in addition to their experiments on different datasets and some of their applications.

Contents

1 Introduction

Computer vision is a field where machines have methods to construct, recognize, and analyze images or videos [SGZ+16, LH15, JXYY13]. Not only can the machine applies these methods but also even learns them. Nowadays, because of deep learning, machines are becoming able to do human tasks and sometimes even better. We can argue here that computers, however, are not able to interpret the images by using senses as humans do. One of the important tasks in computer vision is object recognition [KSH12]. It deals with the process of detecting semantic objects of a certain class (people, car, etc).

Convolutional Neural Networks (CNNs) achieved state-of-the-art performances on large-scale tasks such as image segmentation, image classification, etc [LQD+16, HGDG17, KSH12]. CNN is composed of one or multiple 3-layer stages: conv layer, non-linearity layer, and pooling layer [LKF10]. However, using pooling layer raises an issue which is that the network is not able to learn the spatial relationship between the objects in the image [SFH17]. This issue may lead the network to have difficulties in recognizing images from different points of views. In order to address this issue, Hinton et al. [SFH17] presented the idea of capsules where a capsule is a group of neurons having an activity vector that represents the instantiation parameters of the object being detected. In addition, the length of the vector represents the probability that the detected entity is present in the image. A transformation is applied on the low level capsules that output predictions and then a "routing-by-agreement" mechanism is used to vote for the activation of a high-level capsule. The network architecture using capsules is called "CapsNet" which is composed of two parts: encoder and decoder. The encoder encodes the instantiation parameters into the capsules and then the decoder acts as a regularizer to enforce the network to learn these parameters by reconstructing the original image. The encoder consists of mainly three layers: ReLU Conv1 layer, PrimaryCaps layer, and DigitCaps layer. The decoder, on the other hand, consists of three fully connected layers (FC).

Moreover, Hinton et al. [SFH18] created a new version of capsules called matrix capsule. A matrix capsule is composed of two parts: a 4×4 pose matrix that represents the relationship between the whole object and its parts and an activation probability that represents the probability of object existence. The architecture that uses matrix capsule is called EMCapsNet. It is composed of mainly four layers: ReLU Conv1, PrimaryCaps, ConvCaps1, and ConvCaps2. One main difference between CapsNet and EMCapsNet is that the transformation weight matrices of capsules detecting the same object at different positions are now shared among these capsules.

Referring to [SFH17], CapsNet achieved state-of-the-art performance on the MNIST dataset [LCB98] with a test error of 0.25%. The states in the activity vectors represent a different variation of an MNIST digit. These representations can include stroke thickness, width, etc. In addition, capsule networks were able to segment highly overlapped digits with a classification error of 5.0%. On the other hand, CapsNet did not achieve good results on the CIFAR-10 dataset where it achieved a test error rate of 10.6% compared to the state-of-the-art test error rate of 3.47% [Gra14]. Following [SFH18], experiments were done on the SmallNorb dataset where EMCapsNets achieved good results with a test error rate of 2.6%.

Furthermore, different architectures were introduced based on capsule networks such as SegCaps where they are used for object segmentation [LB18]. CapsuleGAN [JAWN18] also competed convolutional GAN [SGZ+16] by generating better quality images and also achieving better classification test error rates.

1.1 Convolutional Neural Networks

Following [GBC16] in this subsection, CNNs use the following two main concepts when it comes to images: sparse interactions and parameters sharing. Sparse interaction deals with the idea of detecting small and meaningful features using kernels (weighted filters) smaller than the input. For example, simple features such as edges can be detected by looking only at thousands of pixels instead of millions of pixels which makes the computations more efficient since we have fewer parameters. Moreover, in the traditional neural network, each element of the weight matrix corresponds to one computation between an input neuron and its output. However, in CNNs, parameters are shared between neurons looking at the same local spatial

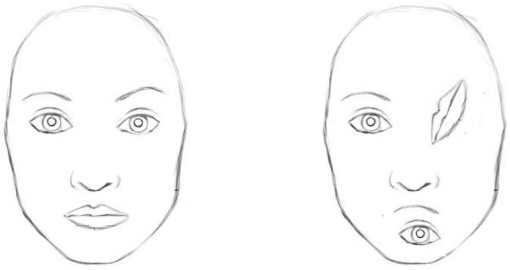


Figure 1: The two pictures are recognized as faces for a CNN. Face image source: [is]

region by having the same weight kernel.

1.1.1 Architecture

1. **Convolutional Layer**: In this layer, given an image as input, each neuron/filter looks at a local spatial region to extract features and map it to a lower space. For example, these filters can be for horizontal lines, vertical lines, etc. The result is a stack of simple feature maps that are used later through the layers to build more complex feature maps.

2. **Activation Layer**: After the convolutional layer, a non-linearity function is applied such as ReLU, where ReLU(x) = max(0, x), in order to add some non-linearity to the system so that more complex relations can be learned between the data.

3. **Pooling Layer**: In this layer, the output of the net at a certain location is replaced with a summary statistics of the neighbors. For example, the max pooling operation returns the maximum output between the pixel intensities of a specific region. It is used to remove "unnecessary" information so that the size of the feature maps is reduced to improve efficiency. However, this leads to the loss of the spatial location information.

4. **Fully Connected Layer**: Neurons in this layer have weight connections to all the activations in the previous layer. So, for a classification problem, each final neuron represents a class and a softmax layer can be added to get a probabilistic interpretation.

1.2 CNN drawbacks

Following [LUTG16] in this paragraph, compositionality is a key ingredient in human-like learning. The idea is that new representations can be constructed through the combination of simple elements. For example, in computer programming, complex functions can be created by using other simple functions (e.g libraries). For object detection, the composition or relation between parts of the image can lead to the construction of a new image. For example, eyes, nose, and mouth can lead to the construction of a face.

The argument is that CNNs are not able to learn with compositionality because of the pooling layer [SFH17]. Applying pooling lead to the loss of information about the spatial relationship between parts [SFH17]. Following this claim, consider the two face images in Figure 1. For humans, the right image is considered unnatural and so it is not really recognized as a face. However, for a CNN, both images will be recognized as a face because it did not capture the spatial relationship between the parts of the image mainly the eye and the mouth. In addition to the down-sampling function that pooling applies, it also helps to create positional invariance. Positional invariance means that if we shift the input by some Δ or rotate it then the output value of the network does not change a lot [GBC16]. In Figure 2, we have three filters trained with separate parameters to detect the digit 5. Each filter is trained to detect a slight change in the orientation of the digit. Now, for the CNN to learn to classify the digit 5 correctly, it needs to have a high activation output regardless of the orientation of the digit. Thus, using the pooling layer, it outputs the maximum activation

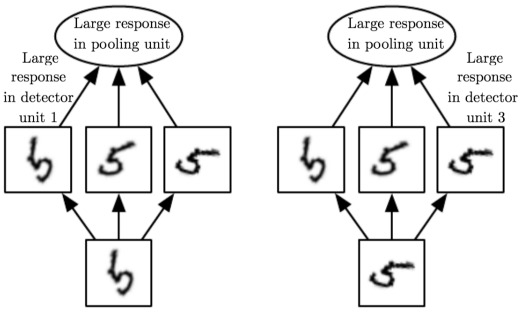


Figure 2: Positional invariance by applying max pooling. Source: [GBC16]

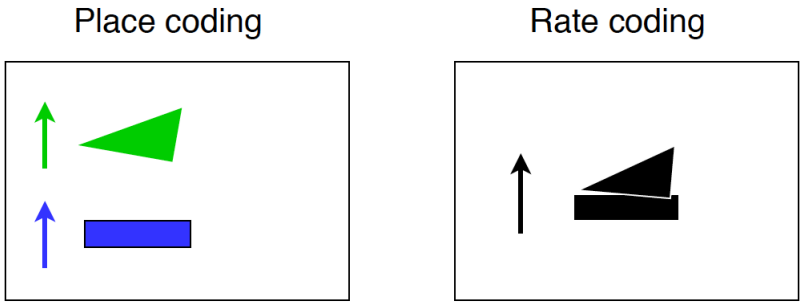


Figure 3: For low-level features (left image), we have place coding where each part of the image, which are the triangle and rectangle shapes, are detected by a group of neurons or a capsule. If these objects move in the image, then most probably they will be detected by a different capsule. On the other hand, in a high-level domain (right image), we have more complex objects and so we have rate coding. Both the triangle and rectangle parts can then form a boat and so it is detected by some capsule. If the boat object moves in the image, then the activations of the neurons will change (states of the activity vector) and not its corresponding capsule. Inspired from: [Mar]

among the outputs of the three filters. [GBC16]. Therefore, for CNN it is more important the presence of the object and not its pose (translation and rotation). To maintain this invariance, we do data augmentation such as rotation, translation, etc.

Following Hinton's speech [Hin] in this paragraph, he argues that what we want is equivariance and not invariance. In other words, a representation that if the viewpoint of the image changes, then the neurons' activities change and not the neurons themselves. There are two types of equivariance: "place-coded" where if an object in an image moves, the neurons (or capsule) representing this object change and "rate-coded" is when the object is moved, the same neurons are still representing it but with different activities. In other words, small objects in low-level domains can be place-coded equivariant in a sense that each position in the image is mapped to a certain group of neurons (capsule). On the other hand, in the high-level domain, we have more complex objects and they become bigger so they can move a long way without changing the neurons but changing their activations to encode pose information. Referring to Figure 3, suppose there are a triangle and rectangle parts in the low-level domain then they are place coded so each shape is represented by a group of neurons. Now as we go up, these two objects together can form a "boat" for example and so it is now rate coded.

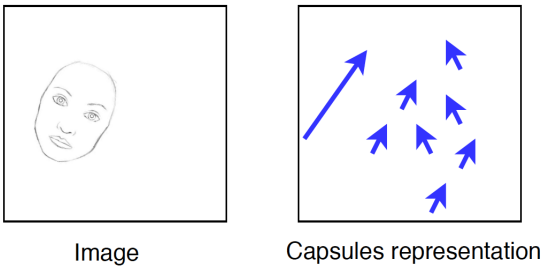


Figure 4: The arrows represent the activity vectors of the capsules. The direction of the vector represents the rotation of the object. The vector of the face capsule has a large length which implies that the probability of the presence of a face is high. Inspired from: [Gé]

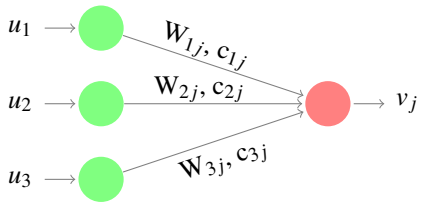


Figure 5: Capsule computation. The green circles represents three low level capsules each having an activity vector u_i for $i \in \{1,2,3\}$. Then, each vector u_i is multiplied by a transformation matrix W_{ij} which outputs a prediction vector that becomes an input to a high level capsule j (red circle). c_{ij} represents the assignment probability of the output of low level capsule i to high level capsule j.

2 Capsule Theory

In computer graphics, images are constructed given initial parameters using some rendering algorithm. These parameters carry information about the position of the objects, in addition to their pose (translation and rotation). Moreover, it needs to take into account the positional relationship between the objects of the image to render the correct image. Inverse graphics is simply the opposite process where the input is an image, and the output is a set of parameters. In a certain sense, Capsule networks do inverse graphics [Hin]. The key point to understand the importance of capsule theory is that the positional relationship between objects needs to be preserved in order to recognize from a different point of views. Recall that this is the main drawback for CNNs according to Hinton and so he argues that capsules will solve this problem.

2.1 Definition of a capsule

Following Hinton et al. [HKW11] in this subsection, instead of aiming for viewpoint invariance, which what CNNs do using the pooling layer, capsules can be used to perform some internal computations on the input and then encapsulates this information in what is called an activity vector. The length of the activity vector encodes the probability that the feature or object which the capsule is recognizing is present in the image. In addition, the state of the feature or object being detected is represented as a set of "instantiation parameters" such as pose, lighting, etc. Note that the length of the vector (probability of presence) does not change as the object moves in the image, however, the instantiation parameters are equivariant. This is because the viewpoint of an object is changed and so its coordinates in the image are changed. In Figure 4 for example, there exists a face in an image, and a capsule detects it with probability 0.9. Now, if the face is rotated in the appearance manifold, the activity vector will rotate because its state has been changed but the length (probability) stays the same because the capsule is still sure that the face is present.

2.2 How does a capsule work?

Previously, the output of a neuron is a scalar but now more information need to be stored and so the output is instead a vector. The following are the computational steps of a capsule.

2.2.1 Transforming matrix multiplication

In Figure 5, there are 3 input activity vectors or capsules that come from the layer below which detected some low-level features. They encoded their probability of existence and their state (pose, orientation, etc). These vectors are then multiplied by a transformation matrix W, which represents the spatial relationship between low-level and high-level features [HKW11]. For instance, assume that the 3 capsules in figure 5 had detected 3 low-level features of a face which are eye, nose, and mouth. Then, for example, matrix W_{1j} may encode the relationship between the eye and the face such as the width of the face is equal to 3 times the width of the eye. The same intuition follows for the other features. The result of this transformation is a prediction vector for the position of the face in relation to the positions of the low-level features [HKW11]. If the prediction vectors are a good match, then the features detected by these capsules are in the right spatial relationship and so they can vote to predict the pose of a face [HKW11]. That is one of the advantages of capsules by allowing the neural network to recognize the whole image by recognizing its parts. The prediction vector from each capsule i to each capsule j is denoted by $\hat{u}_{j|i}$ where $\hat{u}_{j|i} = W_{ij}u_i$ [SFH17].

2.2.2 Scalar weighting of input vectors

In Figure 5, each low-level capsule has a weighted coefficient c connected to the capsule in the higher level. These coefficients are determined by the "Dynamic routing" algorithm. This algorithm will be explained later in details but for now, we are going to talk about its intuition.
When each low-level capsule computes its output (prediction vector), it needs to decide to which capsule in the higher level it should propagate it to [SFH17]. This decision will affect the coupling coefficient c which is multiplied by its output before sending it to its parent (high level) [SFH17]. Now, the high-level capsule already received some input vectors from other low-level capsules. Then, with reference to Hinton's speech [Hin], imagine there exists a vector space for each capsule in the high-level where the input vectors are represented in it as 2D data points. If there are data points that form a cluster, then the vectors represented by these data points are related to each other (vector similarity). Therefore, when a low-level capsule computes its prediction vector after multiplying its activity vector with the transformation matrix, we check where its data point land in the vector space. So, if it lands near a cluster, this implies that it is related to the low-level features of this high-level capsule and based on that the corresponding parameter c is updated. The details are illustrated in Figure 6.

2.2.3 Sum of weighted input vectors

This step is similar to normal neural networks which represents sum over a linear combination between the input and the coupling coefficients [SFH17].

$$s_j = \sum_i c_{ij}\hat{u}_{j|i}$$

Where c_{ij} are the coupling coefficients determined by the dynamic routing algorithm [SFH17]. These coefficients represent a probability distribution for the low-level capsule output to which they are sent to high-level capsules [SFH17]. Therefore, we have $\sum_j c_{ij} = 1$ for each capsule i.

2.2.4 Squashing (non-linearity)

From the definition of a capsule, the length of its activity vector encodes the probability that the feature its detecting is present and so the length should be between 0 and 1. Therefore, we use a non-linear "squashing" function given by [SFH17]:

$$v_j = \underbrace{\frac{||s_j||^2}{1+||s_j||^2}}_{\text{additional scaling}} \cdot \underbrace{\frac{s_j}{||s_j||}}_{\text{unit scaling}} \tag{1}$$

where v_j is the vector output of capsule j and s_j is its input.

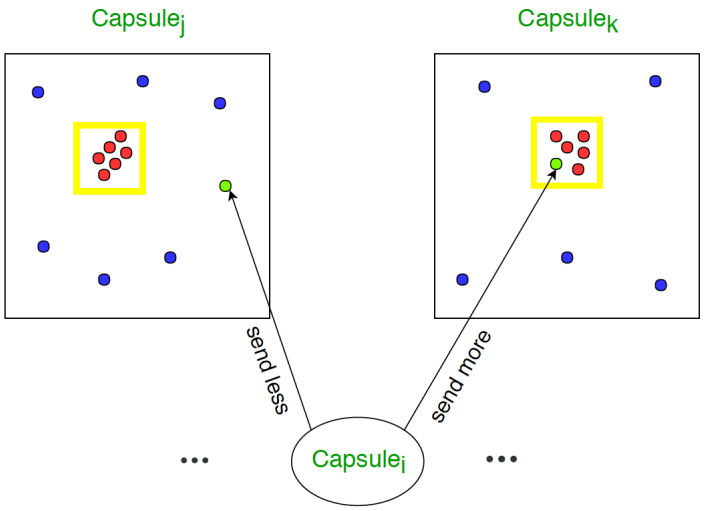


Figure 6: Capsule$_i$ is deciding to which high-level capsule$_j$ or capsule$_k$, it should send its output to. The green dot is the output of capsule$_i$ where it is similar to the other inputs of capsule$_k$ (near the cluster in the yellow box) and not to the inputs of capsule$_j$. Therefore, $c_{ik} > c_{ij}$. Inspired from [Hin]

2.3 Dynamic routing algorithm

As stated in 2.2.2, each low-level capsule needs to decide to which capsule in the higher level it needs to send its output to. The coupling coefficients c_{ij} change with respect to the decision taken and the input of capsule j will be the output of capsule i multiplied by these coefficients. They are determined by the dynamic routing algorithm. The keyword "agreement" is important to understand the essence of this algorithm since a group of capsules "agree" together to activate a higher capsule. The following are the details of each line of Algorithm 1:

Line 1: The parameters in the signature are: $\hat{u}_{j|i}$: output of low-level capsule i, r: number of routing iterations, l: number of the current layer.

Line 2: Initial logits b_{ij} are the log prior probabilities for capsule i to send its output to capsule j [SFH17]. They are simply temporary variables where at each iteration they are updated, and then their values will be used to update c_{ij} [SFH17]. They are initialized to zero.

Line 4-7: This is the main part of the algorithm. The step in **line 4** calculates the coupling coefficient vector for capsule i ($c_i = [c_{i1}, c_{i2}, \ldots, c_{ij}]$). This is done for all low-level capsules in layer l with a Softmax function[1]. The Softmax is used because as stated in 2.2.3, these coefficients have to be non-negative and sum to one to have a probabilistic interpretation. At the first iteration, the entry values of b_i are zeros and so c_i is uniformly distributed (Each entry is $1/K$ where K is the number of high-level capsules). For example, if there are 2 capsules in layer l+1, then $c_i = [1/2, 1/2]$ for capsule i in layer l. This shows that in the beginning, low-level capsules do not know where to send their output to. Moving to **line 5**, in this step, the output of each high-level capsule is calculated as a linear combination of the input of low-level capsules and the weighted coupling coefficients (2.2.3). In other words, the output is the sum over all the prediction

[1] $c_{ij} = softmax(b_i) = \frac{\exp b_{ij}}{\sum_k \exp b_{ik}}$

Algorithm 1: Routing algorithm $(\hat{u}_{j|i}, r, l)$

1 for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$
2 **for** *r iterations* **do**
3 for all capsule i in layer l: $c_i \leftarrow \texttt{softmax}(b_i)$
4 for all capsule j in layer $(l+1)$: $s_j \leftarrow \sum_i c_{ij} \hat{u}_{(j|i)}$
5 for all capsule j in layer $(l+1)$: $v_j \leftarrow \texttt{squash}(s_j)$
6 for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$
7 **return** v_j

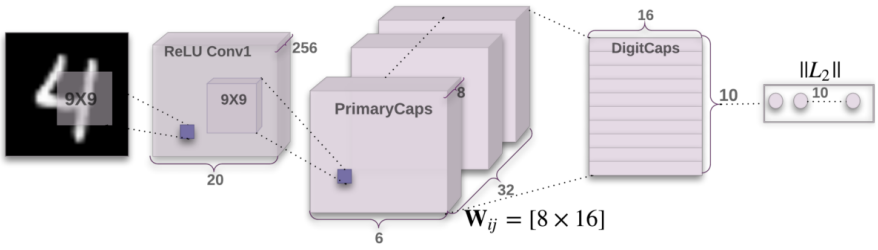


Figure 7: Encoder architecture with 3 layers. Source: [SFH17]

vectors belonging to capsule j from each low-level capsule i. In **line 6**, a non-linearity squashing function is applied to make sure that the length of the vector is between 0 and 1. After calculating the output vector of the high-level capsules, the step in **line 7** updates the log prior weights. Therefore, for each low-level capsule i, and for each high-level capsule j, the dot product between the input to capsule j from capsule i and the current output of capsule j is computed. The result of this product represents a measure of similarity between the outputs. Then, this product is added to the weights b_{ij}. Finally, the above procedure is repeated r times.

After r iterations, all the outputs for high-level capsules and the connection coefficient weights are computed.

2.4 CapsNet architecture

Following [SFH17] in this subsection, CapsNet consists of two parts: An encoder and a decoder. For the encoder, the input is a 28×28 MNIST [LCB98] digit image and it learns to output ten 16-dimensional instantiation parameters vectors (capsules). Thus, the correct digit corresponds to the vector with the largest length. The decoder has three fully connected layers and tries to reconstruct the original image using the activity vectors of the final capsules. This reconstruction is used as a regularization method to enforce the digit capsules to encode the instantiation parameters.

ReLU Conv1 is the first layer of the encoder. It is a standard conv layer that is used to detect simple features to be used later as input to the primary capsules [SFH17]. It has 256 kernels each is 9×9 with stride 1 followed by a ReLU activation function [SFH17]. Then, the size of the output [2] is $20 \times 20 \times 256$ and the number of parameters is $(9 \times 9 + 1) \times 256 = 20,992$

The second layer is the PrimaryCaps layer which has 32 primary capsules whose job is to learn the combination of the features detected in the previous convolutional layer [SFH17]. To do so, each primary capsule applies eight $9 \times 9 \times 256$ convolutional kernels with a stride of 2 instead of 9×9 kernels [SFH17]. Primary capsules can be seen as convolutional layers with squashing (equation 1) as non-linearity function instead of ReLU [SFH17]. The output volume is, calculated as before, $[32 \times 6 \times 6] \times 8$ and the number of parameters

[2] $W_2 = (W_1 - F + 2P)/S + 1$ where F is the size of the kernel, P is padding size, and S is the stride (Same for height).

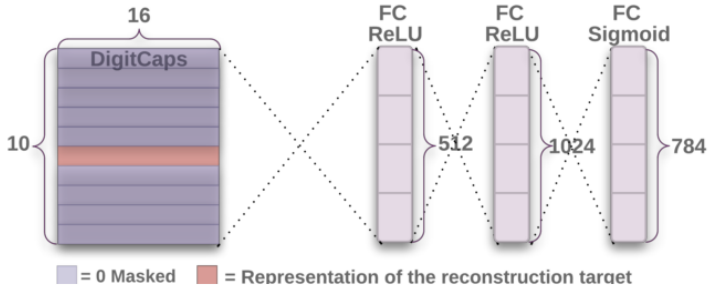


Figure 8: Decoder architecture with 3 FC layers. Source: [SFH17]

is $32 \times 8(\times 9 \times 9 \times 256 + 1) = 5,308,672$.

The final layer (DigitCaps) has 10 digit capsules, one for each digit [SFH17]. Now, as explained in 2.2.1, we have for each 8D vector a transformation weight matrix of size 8×16 that maps 8D capsules to 16D capsules. Since there are 1152 8D vectors, then there are 1152 such weight matrices, 1152 coupling coefficients c, and 1152 routing logits b that are used in the dynamic routing algorithm. So, in total, there are $10 \times (1152 \times 8 \times 16) = 1,474,560$ parameters.

Therefore, by summing up the number of parameters at each layer, the total number of parameters is approximately 6.8M. Note that there is routing only between PrimaryCaps and DigitCaps layers. Conv1 layer output scalar value which does not present enough information. [SFH17].

Recall that the length of the activity vector of a capsule represents the probability that the entity it is detecting is present. The digit of class k will have the longest vector if and only if it is present in the input image [SFH17]. The output of DigitCaps is 10 16D vectors. So, during the training, the loss is calculated for every 10 vectors and then summed up to get the total loss. To calculate the loss of each vector, the following margin loss function is given by [SFH17]:

$$L_k = \underbrace{T_k \max(0, m^+ - ||\mathrm{v}_k||)^2}_{\text{For correct label}} + \lambda \underbrace{(1 - T_k) \max(0, ||\mathrm{v}_k|| - m^-)^2}_{\text{For not correct label}}$$

Where $T_k = 1$ if and only if the digit of class k is present and $m^+ = 0.9$ and $m^- = 0.1$ [SFH17]. When T_k is 1, the length of the DigitCaps k^{th} vector is subtracted from m^+. This means that the loss will be 0 if and only if the probability of detecting the entity by the capsule is greater than or equal to 0.9. Same for the incorrect label detection case, but the loss will be 0 if and only if DigitCap predicts an incorrect label with a probability less than or equal to 0.1. Moreover, λ is used for loss balance and it is set to 0.5 [SFH17].

In the decoder phase, all the digit capsules from the DigitCaps layer are masked out except the correct one [SFH17]. The selected 16D vector is used then as input for the decoder that tries to learn to reconstruct a 28×28 image with the aim of minimizing the sum of squared differences between the reconstructed image and the original image [SFH17]. In this way, it helps the capsules to learn instantiation parameters to construct the original image [SFH17]. This is the same as FC layers in CNN where the 1×16 input is weighted and connected to 512 neurons of the next layer. Then, the 512 neurons are connected to 1024 neurons and finally, the 1024 neurons are connected to 784 neurons which can be represented as 28×28. The total number of parameters is $(16 + 1) \times 512 + (512 + 1) \times 1024 + (1024 + 1) \times 784 = 1,337,616$. Figure 9 shows the results of the reconstruction.

2.5 Capsules on MNIST

Following [SFH17], the training is performed on 28×28 images from the MNIST dataset [LCB98]. The images are shifted up to 2 pixels in each direction of the x-axis and y-axis. The training set size is 60k

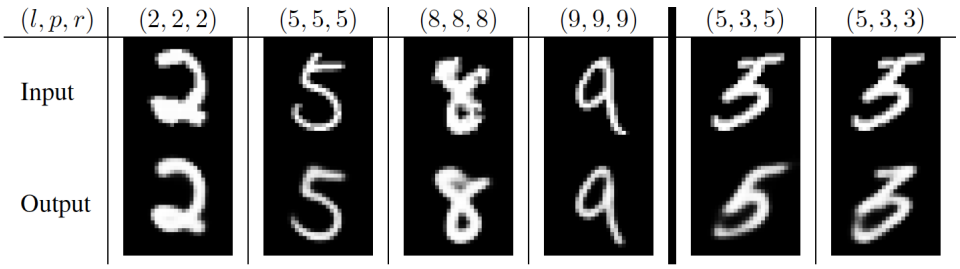


Figure 9: Reconstruction of input result. (l, p, r) represents the label, the prediction, and the reconstructed target respectively. The results show how the model preserves many of the details of the input image. Moreover, the two rightmost columns represent a reconstruction failure where the model got confused between digit 3 and digit 5. Source: [SFH17]

Method	Routing	Reconstruction	MNIST (%)	MultiMNIST (%)
Baseline	-	-	0.39	8.1
CapsNet	1	no	$0.34_{\pm 0.032}$	-
CapsNet	1	yes	$0.29_{\pm 0.011}$	7.5
CapsNet	3	no	$0.35_{\pm 0.036}$	-
CapsNet	3	yes	$\mathbf{0.25}_{\pm 0.005}$	**5.2**

Table 1: CapsNet classification test accurary. Source: [SFH17]

images and the testing set is 10k images. A single model is used without averaging and achieved a low test error rate of 0.25% (state-of-the-art performance) using CapsNet (2.4). Table 1 shows the test error rate for different CapsNets having a different number of routing iterations and also reconstruction layers. The best model is the one with 3 routing iterations and reconstruction. The reconstruction acts as a regularizer that boosts the routing performance by applying the pose that is encoded in the activity vectors. The baseline is a standard CNN with three convolutional layers of 256, 256, 128 channels and each has 5×5 kernels and stride of 1. Moreover, the number of parameters of this baseline is 35.4M, whereas, the CapsNet has only 8.2M parameters and 6.8M without the reconstruction subnetwork.

In addition, experiments show that CapsNets are robust to affine transformations [3] during training. To test this, a new MNIST dataset was created called "affNIST." In affNIST, a random small affine transformation operation is applied on each MNIST digit. Then, two models: CapsNet and a traditional CNN (with Max-pooling and Dropout) were trained on a padded and translated MNIST dataset where each training sample is an MNIST digit placed randomly on a 40×40 pixel black background. Furthermore, the two models were tested on the affNIST dataset and the results were that the CapsNet got a test accuracy of 79% while the CNN got a test accuracy of only 66%.

2.5.1 Capsule representation in MNIST

Following [SFH17], in the DigitCaps layer, the dimensions of the correct class digit capsule should learn the different variations of how the digit can be initiated. This includes stroke thickness, skew, width, etc. To test this claim, a perturbation can be applied to the activity vector of the correct digit and then passed as input to the decoder. Figure 10, shows how changing one dimension of the activity vector is affecting the reconstruction.

[3] "An affine transformation is an important class of linear 2-D geometric transformations which maps variables (e.g. pixel intensity values located at position (x_1, y_1) in an input image) into new variables (e.g. (x_2, y_2) in an output image) by applying a linear combination of translation, rotation, scaling and/or shearing (i.e. non-uniform scaling in some directions) operations" [RFW00]

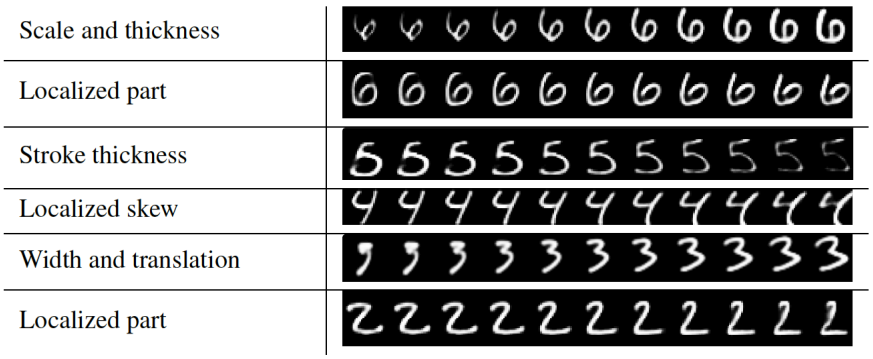


Figure 10: Each row shows the reconstruction after changing one dimension in the class capsule vector. Source: [SFH17]

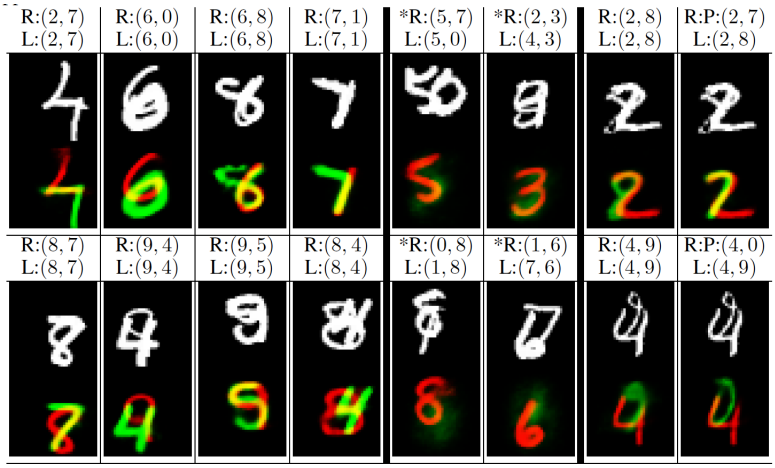


Figure 11: Reconstruction using the MutliMNIST dataset. The constructed images are shown in green and red as the lower image. The upper image is the input image. $R : (r_1, r_2)$ represents the label for the two reconstructed images and $L : (l_1, l_2)$ represents the label for the two digits in the image. The two rightmost columns show two wrong reconstruction classification from the label and prediction P. For example, in the (2,8) example, the model shows confusion between 8 and 7. In addition, the first four columns show correct classification result. The reconstruction from digits that are neither from the label nor from the prediction is shown in the columns with the (*) mark. For example, in the case of (5,0), the model can not construct 7 instead because it is sure that 7 does not exist in the image. CapsNet is correctly able to segment the image into two digits even if the pixels are overlapped. This is because the segmentation process is done at a higher level and not at the pixels level. The variation of each digit is encoded in the activity vector of the corresponding digit capsule and then reconstructed using the decoder. Source: [SFH17]

2.5.2 Segmenting highly overlapping digits

In reference to [SFH17], the dynamic routing algorithm allows capsules in low-level to contribute to the activation of "some" high-level capsules and ignoring the others. This feature allows CapsNets to detect multiple objects in the image even if they are overlapped. For that, a MultiMNIST training and test datasets were created by overlapping one digit on top of the another (80% overlap) with a shift of 4 pixels on each axis. A 3 layer CapsNet model was trained on the MultiMNIST dataset and achieved a classification error rate of 5.0%. When testing, the two most active digit capsules are selected as the classified digits. During reconstruction, one digit capsule is chosen at a time and its activity vector is used to reconstruct the original image. Figure 11 shows the results of this experiment.

Figure 12: Reconstruction of MNIST and CIFAR-10 images using Hinton MNIST model presented in [SFH17]. Source: [XBJ17]

2.6 CIFAR-10 dataset

The CIFAR-10 dataset [KNH] consists of 60000 32×32 images of 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The classes are airplanes, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. CapsNet did not do well when tested on the CIFAR-10 dataset where it achieved 10.6% test error rate with an ensemble of 7 models each trained with 3 routing iterations [SFH17]. Xi et al. [XBJ17] did an experiment to show the difference between real and reconstructed images for both the MNIST and CIFAR-10 datasets using Hinton et al. MNIST model [SFH17]. In Figure 12, the top half is the input images and the bottom half is the reconstructed images. As observed in the figure, MNIST reconstructed images have a clear structure while CIFAR-10 reconstructed images are blurred. A reason behind this issue could be that a CIFAR-10 image may have different viewpoints compared to simple MNIST digit image. Thus, applying the regularization reconstruction technique might lead to wrong values which explain the blurring in the reconstructed images of the CIFAR-10 dataset in Figure 12.

2.7 EM routing by agreement

Hinton et al. [SFH18] described a new version of capsules where each capsule has a logistic unit that represents the presence of an entity and a 4×4 pose matrix that could learn the relationship between part and its whole. This lead to the EM Routing algorithm that is implemented differently from the dynamic routing algorithm.

2.7.1 Matrix capsule

As visually illustrated in Figure 13, a matrix capsule consists of two parts: activation probability that represents the existence of the entity and a pose matrix that captures the relationship between a part and its whole [SFH18]. The main difference between the standard activation in neural networks and capsule activation is that the last one is based on the comparison between multiple incoming pose prediction vectors while the other one is based on the comparison between a single incoming activity and a learned weight vector [SFH18].

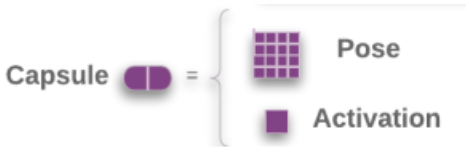


Figure 13: Matrix capsule. Each capsule consists of two parts: a 4×4 pose matrix and an activation probabiliy. Source: [SFH18]

2.7.2 EM routing algorithm

In general, by referring to [Bis06], the EM algorithm is an iterative method to estimate the maximum likelihood estimates for the parameters of a statistical model which depend on unobserved data. The algorithm works by alternating between an Expectation step (E-step), where the current estimated parameters are used to evaluate the posterior probabilities, and a Maximization step (M-step), where the parameters are re-estimated.

Assume that the poses and activation probabilities of all capsules in layer l are given and we want to decide which capsules in layer $l+1$ should be activated. Then, this can be formulated as an unsupervised learning problem and can be tackled using the Expectation Maximization algorithm (EM algorithm) [SFH18]. Then, each high-level capsule can correspond to a Gaussian distribution and the pose of each low-level capsule is a data point that needs to be assigned to these distributions [SFH18].
In EM routing, the 4×4 pose matrix is modeled as a Gaussian distribution but with 16 μ and 16 σ where each μ represents a pose's matrix component [SFH18]. Let V_{ij} be the vote from capsule i to capsule j and V_{ij}^h be its h^{th} component [SFH18]. So, the probability of V_{ij}^h belonging to capsule j's Gaussian model is given by [SFH18] as follows:

$$p_{i|j}^h = \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} \exp\Big\{ -\frac{(V_{ij}^h - \mu_j^h)}{2(\sigma_j^h)^2} \Big\}$$

Taking the natural logarithm yields the following:

$$\ln\left(p_{i|j}^h\right) = -\ln(2\pi)/2 - \ln(\sigma_j^h) - \frac{(V_{ij}^h - \mu_j^h)}{2(\sigma_j^h)^2}$$

Let $cost_{ij}^h$ be the cost of activating capsule j by capsule i be defined as $cost_{ij}^h = -\ln(p_{i|j}^h)$ [SFH18]. Now, if the cost is low, the more likely the high-level capsule j would be activated which means that there is agreement between the low-level capsules. On the other hand, high cost means that the votes do not match and so they are not from the same Gaussian distribution.
The total cost from all low-level capsules can be derived as follows [SFH18]:

$$cost_j^h = \sum_i r_{ij} cost_{ij}^h = \sum_i -r_{ij} \ln(p_{i|j}^h) = \frac{\sum_i r_{ij}(V_{ij}^h - \mu_{ij}^h)^2}{2(\sigma_{ij}^h)} + (\ln(\sigma_j^h) + \ln(2\pi)/2) \sum_i r_{ij} = (\ln(\sigma_j^h + C) \sum_i r_{ij}$$

where $C = \ln(2\pi)/2 + 1/2$ is a constant and r_{ij} is the assignment probability of low-level capsule i to high-level capsule j [SFH18]. For example, the assignment probability of a capsule representing a hand to capsule representing a face is 0.
Finally, to determine the activation of high-level capsule j, the following equation is given by [SFH18]:

$$a_j = g\Big(\lambda\Big(\beta_a - \beta_u \sum_i r_{ij} - \sum_h cost_j^h\Big)\Big)$$

where g is a logistic function (e.g sigmoid). β_u is a learned parameter and it is the cost paid per data point when the high-level capsule is not activated, for describing all the poses of the low-level capsules [SFH18]. β_a is also learned and it is the cost paid for describing the high-level capsule's mean and variance in case it is activated [SFH18]. Finally, λ is the inverse temperature parameter [SFH18].

The EM routing algorithm fits data points (low-level capsules) into a mixture of Gaussian models (high-level capsules) through an iterative alternation between E-step and M-step methods [SFH18]. The E-step determines the probability assignment r_{ij} [SFH18]. Then, the M-step recalculates the parameters of the Gaussian models based on estimated r_{ij} [SFH18]. At the end, we get a_j, the activation of the high-level capsule j [SFH18]. The 16 μ of the Gaussian models are reshaped into 4x4 matrix which represents the pose matrix of the high-level capsule [SFH18].

1: **procedure** EM ROUTING($\boldsymbol{a}, V$)
2: $\quad\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
3: $\quad$**for** t iterations **do**
4: $\quad\quad\forall j \in \Omega_{L+1}$: M-STEP($\boldsymbol{a}, R, V, j$)
5: $\quad\quad\forall i \in \Omega_L$: E-STEP($\mu, \sigma, \boldsymbol{a}, V, i$)
$\quad$**return** $\boldsymbol{a}, M$

1: **procedure** M-STEP($\boldsymbol{a}, R, V, j$) ▷ for one higher-level capsule, j
2: $\quad\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * \boldsymbol{a}_i$
3: $\quad\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
4: $\quad\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} (V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
5: $\quad cost^h \leftarrow \left(\beta_u + log(\sigma_j^h)\right) \sum_i R_{ij}$
6: $\quad a_j \leftarrow logistic(\lambda(\beta_a - \sum_h cost^h))$

1: **procedure** E-STEP($\mu, \sigma, \boldsymbol{a}, V, i$) ▷ for one lower-level capsule, i
2: $\quad\forall j \in \Omega_{L+1}$: $\boldsymbol{p}_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} \exp\left(-\sum_h^H \frac{(V_{ij}^h - \boldsymbol{\mu}_j^h)^2}{2(\sigma_j^h)^2}\right)$
3: $\quad\forall j \in \Omega_{L+1}$: $\boldsymbol{R}_{ij} \leftarrow \frac{\boldsymbol{a}_j p_j}{\sum_{k \in \Omega_{L+1}} \boldsymbol{a}_k p_k}$

Figure 14: EM routing algorithm. Source: [SFH18]

In Figure 14, a and V are the activation and votes from the low-level capsules. The probability assignments are uniformly distributed initially. Then, for t iterations, M-step is called to update the Gaussian models based on $a, V,$ and r_{ij} (line 4 in EM routing procedure). After that, E-step is called to recalculate r_{ij} based on the new Gaussian models (line 5 in EM routing procedure). μ and σ^2 are recalculated based on the activation of each low-level capsule i (lines 3 and 4 in M-STEP procedure). The cost and the activation a_j are also recomputed for high-level capsule j (lines 5 and 6 in M-step procedure). At last, in the E-step, the probability assignments r_{ij} are recalculated based on the updated parameters μ, σ, and a_j (line 3 in E-step procedure).

To learn the transformation weight matrices and the hyperparameters β_a and β_u, a spread loss function is used for the backpropagation algorithm to maximize the gap between the activation of the correct class and the activation of the other classes [SFH18]. The loss function for class i is defined by [SFH18] as:

$$L_i = (max(0, m - (a_t - a_i))^2, \qquad L = \sum_{i \neq t} L_i \tag{2}$$

where a_t is the activation of the true class and a_i is the activation of class i. If the margin between a_t and a_i is greater than m, then a squared penalty is added to the loss. We start with a small margin of 0.2 and then increase throughout the training phase to 0.9 so in this way too many dead capsules can be avoided during the earlier layers [SFH18].

2.7.3 CapsNetEM architecture

Following [SFH18], in Figure 15, the first layer is a ReLU Conv1 layer using 5×5 conv filter with stride of 2 outputting 32 feature maps ($A = 32$). Then, it is followed by a PrimaryCaps layer where a 1×1 conv filter is applied to transform the 32 channels into 32 capsules ($B = 32$). Each capsule has a 4×4 pose matrix plus an activation value. This layer is followed by two covolutional layers with 3×3 filter size ($K = 3$) each with 32 capsule types (C=D=32), and strides of 2 and 1 respectively. The ConvCaps layers take capsules as input and output capsules based on the EM Routing algorithm that was discussed in 2.7.2. Finally, the outputs are connected to the Class Capsules using a 1×1 filter and it outputs one capsule per class (E is the

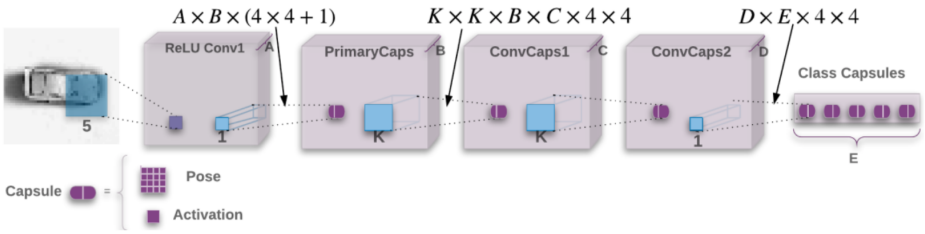


Figure 15: CapsNetEM architect. Source: [SFH18]

number of classes).

Capsules of the same type are detecting the same entity at different locations or positions is established by sharing the transformation matrix across these capsules. "We therefore share the transformation matrices between different positions of the same capsule type and add the scaled coordinate (row, column) of the center of the receptive field of each capsule to the first two elements of the right-hand column of its vote matrix. We refer to this technique as Coordinate Addition. This should encourage the shared final transformations to produce values for those two elements that represent the fine position of the entity relative to the center of the capsule's receptive field" [SFH18].

Hinton et al. [SFH18] created the idea of matrix capsules in order to overcome some of the limitations of using vectors in the routing-by-agreement algorithm:

1. The length of the vector represents the probability that an entity exists in the image. To keep this between 0 and 1, we use a nonlinear squashing function which prevents the existence of any objective function to be minimized [SFH18].

2. It uses the cosine to measure the agreement between two pose vectors. However, the cosine is not a good measure to distinguish between "good" and "very good" agreement (EM routing use log variance of a Gaussian cluster) [SFH18].

3. It uses a vector of length n rather than a matrix to represent the pose and so the transformation matrix is only with n parameters instead of n^2 [SFH18].

2.7.4 SmallNORB dataset

The SmallNORB dataset contains 5 classes of toys: airplanes, cars, trucks, humans, and animals. Every toy is pictured at 18 different azimuths(0-340), 9 elevations, and 6 lighting conditions [SFH18]. The model (using EM routing) achieved a test error rate of 1.8% where the best-reported result was 2.56% [SFH18]. In addition, it was tested on the NORB dataset (a modified version of smallNORB with added background) and achieved a 2.6% error rate where the state-of-the-art performance is 2.7% [SFH18].
Figure 16 illustrates how EM routing works. The distance of the votes to the mean (pose) of each class is represented by histograms during the routing iterations [SFH18]. In the first iterations, the votes are uniformly distributed. Later, capsules start to agree and so the assignment probabilities increase and most of the votes are assigned to the detected cluster [SFH18].

3 CapsNet applications

LaLonde et al. [LB18] claim that performing object segmentation with a capsule-based network is difficult for some reasons. The dynamic routing algorithm (Algorithm 1) is computationally expensive in terms of both execution time and space. The computation is expensive because the routing process is taking place between a parent in a high-level and "all" the children in the lower level. In addition, the number of parameters needed grows so fast. Therefore, the solution suggested was to modify the dynamic routing

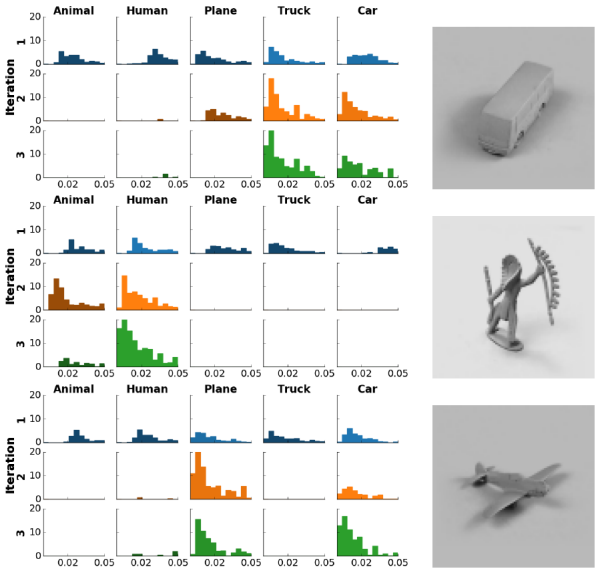


Figure 16: The distance of votes to the mean of each 5 final capsules is represented by histograms [SFH18]. The EM routing algorithm correctly routes the votes for the truck and human image, but it fails for the plane image. Source: [SFH18].

algorithm so that children are only routed to parents within a local spatial window and also the transformation matrices are shared only between capsules of the same type. This lead to a new architecture known as "SegCaps." The results in Table 2 shows that SegCaps outperforms other approaches and also with less number of parameters with a score of 98.479% on average. In addition, Figure 17 highlights U-Net [RFB15] segmentation leakages in comparison to SegCaps.

Method	Parameters	Split-0 (%)	Split-1 (%)	Split-2 (%)	Split-3 (%)	Average (%)
U-Net	31.0 M	98.353	98.432	98.476	**98.510**	98.449
Tiramisu	2.3 M	98.394	98.358	**98.543**	98.339	98.410
Baseline Caps	1.7 M	82.287	79.939	95.121	83.608	83.424
SegCaps (R1)	**1.4 M**	98.471	98.444	98.401	98.362	98.419
SegCaps	**1.4 M**	**98.499**	**98.523**	98.455	98.474	**98.479**

Table 2: SegCaps test results. Source: [LB18]

Moreover, capsule networks can be used in the field of adversarial training. So, Jaiswal et al. [JAWN18] present a framework that uses capsule networks instead of CNNs as discriminators within the Generative Adversarial Network (GAN) setting. A GAN consists of a generator which tries to transform images drawn from a prior distribution to more complex ones hoping to fool the discriminator by making them look real [SGZ+16]. On the other hand, the discriminator takes the fake images along with some real input images as input and decides if the generated images are fake or no [SGZ+16]. They showed that CapsuleGANs works better than CNN-based GANs.

CaspsuleGAN discriminator has the same architecture as CapsNet (2.4). However, the final layer of the CapsuleGAN discriminator has a capsule where the length of its activity vector represents the probability of having a real or fake image [JAWN18].

The model was tested on MNIST and CIFAR-10 datasets [JAWN18]. Figure 18 shows images generated

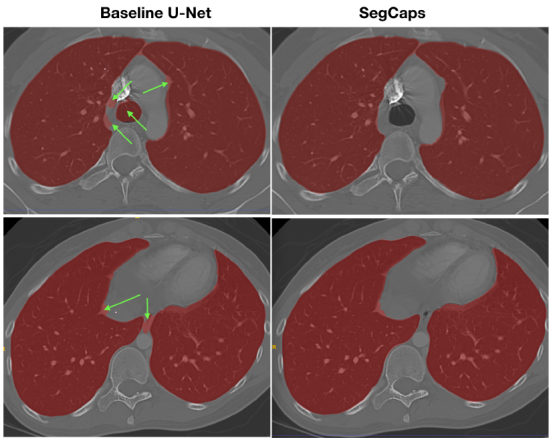


Figure 17: U-Net (state-of-the-art method of for segmentation in medical imagining) [RFB15] model vs Segcaps. The green arrows shows where U-Net did mistakes. Source: [LB18]

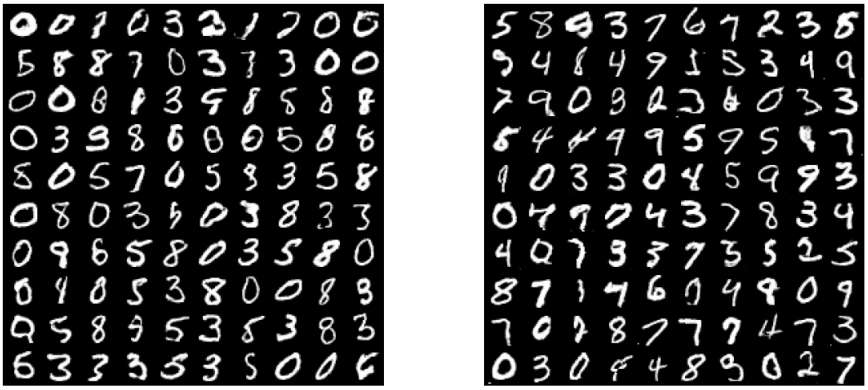
using the standard convolutional-GAN and CapsuleGAN on the MNIST dataset. Both models produce images of similar quality. However, the images generated by CapsuleGAN are more diverse compared to the images generated by the standard GAN. For example, there are many zero digit images in Figure 18(a). Moreover, Figure 19 shows the result of this experiment on the CIFAR-10 dataset where images generated by CapsuleGAN looks more cleaner and crisper [JAWN18]. In addition to this visual quality evaluation, the performance of both models was measured on semi-supervised classification task [JAWN18]. In this experiment, $50,000$ images are randomly generated using both models. Table 3 shows the results of this experiment on the MNIST dataset while Table 4 shows the results of the experiment on the CIFAR-10 dataset. In both tables, n represents the number of labeled images where $n \in \{100, 1,000, 10,000\}$[JAWN18]. The error rates are high because raw pixels are provided as features for the classification algorithm [JAWN18]. The results show that CapsuleGAN outperforms convolutional GAN for all error rates with a margin of $1.7 - 3.97\%$ for MNIST and $0.91 - 3.22$ for CIFAR-10.

Model	**Error Rate**		
	n = 100	n = 1,000	n = 10,000
GAN	0.2900	0.1539	0.0702
CapsuleGAN	**0.2724**	**0.1142**	**0.0531**

Table 3: Results of semi-supervised classification on the MNIST dataset. Source: [JAWN18]

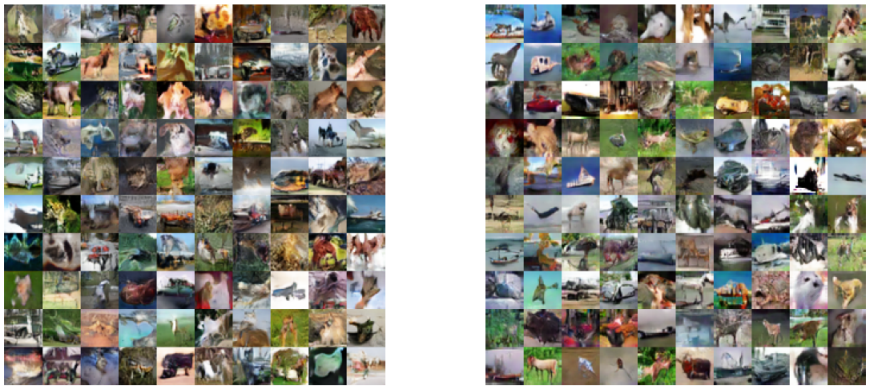
Model	**Error Rate**		
	n = 100	n = 1,000	n = 10,000
GAN	0.8305	0.7587	0.7209
CapsuleGAN	**0.7983**	**0.7496**	**0.7102**

Table 4: Results of semi-supervised classification on the CIFAR-10 dataset. Source: [JAWN18]



(a) GAN (b) CapsuleGAN

Figure 18: Randomly generated MNIST images using convolutional GAN and CapsuleGAN. Source: [JAWN18]



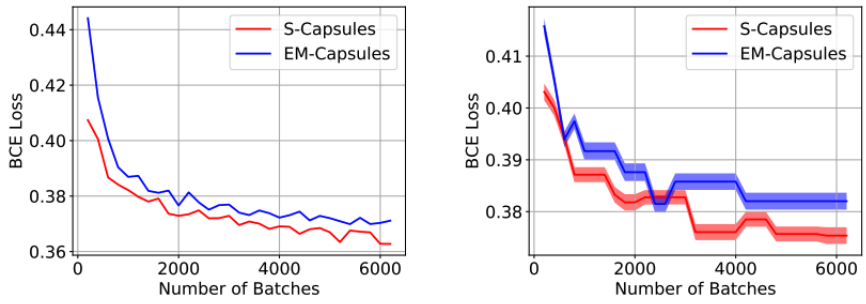
(a) GAN (b) CapsuleGAN

Figure 19: Randomly generated CIFAR-10 images using convolutional GAN and CapsuleGAN. Source: [JAWN18]

4 Discussion

Regardless of how good CapsNets are, they have some limits and drawbacks. Hinton et al. [SFH17] point out a drawback of capsules when tested on the CIFAR-10 dataset. Capsules work better when the background is clear so that it does not need to recognize many categories during the dynamic routing algorithm [SFH17]. An MNIST digit image has a clear background and so capsules only need to detect the features of this digit. However, in CIFAR, the background of the image can have multiple objects which are not related to each other (such as a tree, grass, cloud, ...) and so the capsule responsible for detecting the background in the last layer of CapsNet will not be able to detect everything. In CIFAR-10, the background is too varied and so adding more capsules will make the net bigger which leads to poor performance [SFH17].

In addition, applying EM routing algorithm at each layer might be complex and computationally expensive [oLRI18]. For that, Spectral capsules were suggested where they converge faster than capsule networks that use EM routing [Bah18]. The idea behind these new capsules is to create a linear subspace dominated by the dominant eigenvector computed from the singular value decomposition of the matrix of votes of low-level capsules to high-level capsules [Bah18]. This is somehow similar to the Principle Component Analysis algorithm where the first (dominant) eigenvector is normal to the linear subspace and it preserves most of the variance in the vote vectors of low-level capsules [Bah18]. Experiments were done on a learning



(a) Training loss (b) Test loss

Figure 20: Spectral capsules networks vs normal capsule networks. We can see that S-Capsule Network is faster in convergence. Source: [Bah18]

diagnose task [LKEW15] which is a multivariate time series classification task where we need to predict the diseases that a patient have based on his times series of vital signs and lab results. Figure 20 shows the convergence behavior of S-Capsules compared to EM-Capsules. S-Capsule learns faster and generalizes better compared to EM-Capsules [Bah18]. We think that this methodology is a good motivation for using capsule networks later because one of the main problems of capsule networks is that they are slow to converge and so cannot be applied on large datasets (e.g ImageNet).

5 Conclusion

To sum up, Hinton argues that CNNs have a major drawback. The problem with a CNN is that it does not model the spatial relationship between the components of the image. This arises specifically because of the max pooling layer that throws "important" information away when applied in their architecture. Therefore, capsule networks were created to tackle this problem.
A capsule encapsulates internal information in an activity vector of instantiation parameters and the length of the vector represents the probability that the entity which the capsule is detecting is present in the image. A capsule output an activity vector that is multiplied with a transformation matrix that represents the relationship between this capsule and a capsule of a high-level feature. After that, the dynamic routing algorithm is applied where each capsule tries to decide optimally to which high-level capsule it should send its output to. This is done using the idea of "agreement" between the low-level capsules. Later, the EM routing was introduced where a capsule has an activation and a pose matrix. The routing is now done using the EM algorithm where each capsule is a data point that needs to be assigned to a high-level capsule that is represented by a Gaussian distribution.
Capsule Networks achieved state-of-the-art performance on the MNIST dataset. However, they did not do well on the CIFAR-10 dataset due to its background variation. In addition, experiments were done on the smallNORB dataset using EMCapsNet and showed good results.
Experiments were done for different tasks using Capsule networks. SegCaps is one example that was created with the aim of doing object recognition task using capsule networks. Moreover, in adversarial learning, CapsGAN which is based on capsule networks outperformes Convolutional-GAN.

6 References

[Bah18] Mohammad Taha Bahadori. Spectral capsule networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Gra14] B. Graham. Fractional Max-Pooling. *arXiv preprints arXiv:1412.6071*, 2014.

[Gé] Aurélien Géron. Capsule networks (capsnets) – tutorial. https://www.youtube.com/watch?v=pPN8d0E3900.

[HGDG17] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. *arXiv preprints arXiv:1703.06870*, 2017.

[Hin] Geoffrey Hinton. Geoffrey hinton talk "what is wrong with convolutional neural nets ?". https://www.youtube.com/watch?v=rTawFwUvnLE&feature=youtu.be.

[HKW11] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. *Lecture Notes in Computer Science Artificial Neural Networks and Machine Learning – ICANN 2011*, page 44–51, 2011.

[is] iaba sd.us. Back to post :easy faces to draw. http://iaba-sd.us/easy-faces-to-draw/easy-faces-to-draw-how-to-draw-a-realistic-face-female-download/. [Online; accessed June 1, 2018].

[JAWN18] A. Jaiswal, W. AbdAlmageed, Y. Wu, and P. Natarajan. CapsuleGAN: Generative Adversarial Capsule Network. *arXiv preprint arXiv:1802.06167*, 2018.

[JXYY13] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013.

[KNH] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[LB18] R. LaLonde and U. Bagci. Capsules for Object Segmentation. *arXiv preprint arXiv:1804.04241*, 2018.

[LCB98] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwriteen digits. 1998. http://yann.lecun.com/exdb/mnist/.

[LH15] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375, June 2015.

[LKEW15] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzel. Learning to Diagnose with LSTM Recurrent Neural Networks. *arXiv preprints arXiv:1511.03677*, 2015.

[LKF10] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, May 2010.

[LQD+16] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei. Fully Convolutional Instance-aware Semantic Segmentation. *arXiv preprint arXiv:1611.07709*, 2016.

[LUTG16] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building Machines That Learn and Think Like People. *arXiv preprint arXiv:1604.00289*, 2016.

[Mar] Charles H. Martin. Capsule networks: overview. https://www.youtube.com/watch?v=YqazfBLLV4U.

[oLRI18] International Conference on Learning Representations (ICLR). Openreview.net, 2018. https://openreview.net/forum?id=HJWLfGWRb.

[RFB15] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv preprint arXiv:1505.04597*, 2015.

[RFW00] Ashley Walker Robert Fisher, Simon Perkins and Erik Wolfart. Affine transformation, 2000. https://homepages.inf.ed.ac.uk/rbf/HIPR2/affine.htm.

[SFH17] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In *Neural Information Processing Systems (NIPS)*, 2017.

[SFH18] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Matrix capsules with em routing. In *International Conference on Learning Representations (ICLR)*, 2018.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

[XBJ17] E. Xi, S. Bing, and Y. Jin. Capsule Network Performance on Complex Data. *arXiv preprint arXiv:1712.03480*, 2017.