Transforming autoencoders [1]

**Abstract**

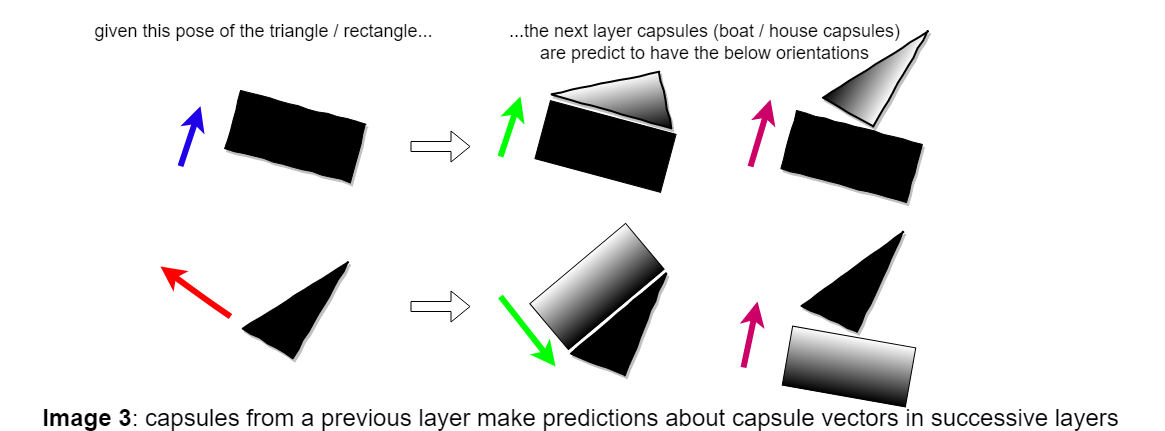
A distinction is made between the traditional hand-engineered feature extractors used in computer vision and the learned feature extractors used in CNNs. It is supported that both current methods have their limitations. CNNs use convolutional filters which produce scalar outputs. The scalar outputs do not seem encode information robust to variations in lighting, pose etc. On the other hand, traditional feature extractors (like HOG) extract a vector of features for each KeyPoint but lack the adaptability that the machine learning technics have. In this paper, Neural Networks are adjusted to merge the merits of the above systems: neural networks with learnable feature detectors which output vectors of robust features.

**Introduction**

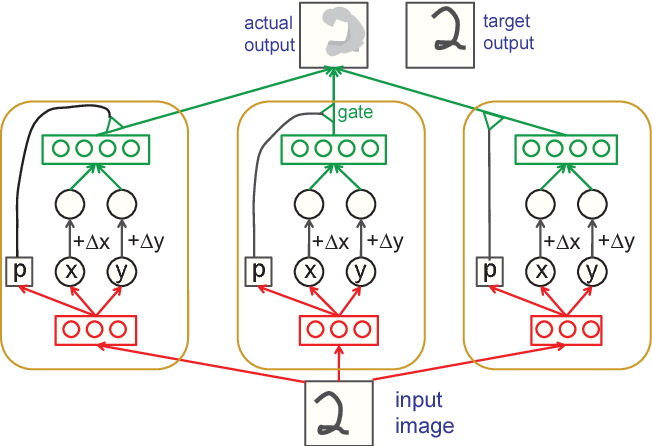
The paper tries to address some shortcomings of the other computer vision methods. It mentions that in traditional computer vision systems and in CNNs, the object recognition is performed while discarding the position and orientation information. Thus, the system’s output does not contain any spatial information. More specifically, in CNNs, viewpoint invariance is achieved to a limited extent by subsampling the activities of local pools of translated replicas of the same kernel. This subsampling is often referred as pooling. However, pose information which are discarded during pooling are crucial for tasks like face recognition. In those tasks, spatial relationship between object parts (nose, mouth, eyes) and whole (face) are required.

The paper proposes the use of local “capsules”. Every “capsule” is capable of recognizing an implicitly defined visual entity (object or object part). It outputs a probability that the entity is present and a set of instantiation parameters encoding the pose or even the lighting, deformation etc. of the entity (relative to an implicit canonical version). It is important to note that the entity-probability is invariant within the appearance manifold (meaning that the value is constant regardless of the pose and other conditions of that entity). However, the instantiation parameters are equivariant (i.e., they change in accordance with the change of the entity's' appearance).

In many cases, the instantiation parameters represent the pose of the capsule’s entity. This is made possible by using a transformation matrix *(T)* that describes the coordinate transformation between the observed instantiation of the entity and the implicit canonical entity. By using this instantiation parameters, a higher-level capsule representing a whole (comprised of lower-level parts) can be activated by the following procedure. Consider that a whole entity *boat* consists of parts *rectangle* and *triangle*. Then, capsule *boat* is activated if the relative positions of *rectangle* and *triangle* are properly aligned. For this, we multiply T\_rectangle by T\_rectangle-boat and get T\_boat prediction (which is the predicted pose of the boat if we consider the pose of the rectangle). Note that T\_rectangle-boat describes the part-whole relationship between the rectangle and the boat (given the rectangle’s pose, tells where the boat’s pose) and is viewpoint invariant (does not change as the viewpoint changes). Similarly, we do the same for the triangle and get another prediction. If the two predictions agree, the capsule representing a boat is activated. If the two predictions aren’t aligned, maybe the house capsule is activated.



**Method**



The capsule network in this paper takes the form of an autoencoder. In the general case, the network has a number of capsules in each layer which enclose “recognition units” and “generation units”. The “recognition units” (in red) are used to compute the probability p of an entity being present and the instantiation parameters while the “generation units” (in green) are used to reconstruct the image.

In the simple case, the network consisted of a single layer of 30 capsules. Each capsule contains 10 recognition units and 20 generation units. The network is fed with 2D images of the MNIST dataset. More specifically, the images are randomly shifted by –2, -1 ,0 ,+1 ,+2 pixels in the y and x axis before they are fed to the network. The output image is also shifted. Additionally, relative desired shifts between the input image and the desired, output image in the two axes are also fed to the network’s generation units (denoted as Δx and Δy). It is important to note that the recognition units in the simple case learn to output the position x, y of the entity the capsule is implicitly set to recognize. The deltas are added to the estimated x and y and the sum is fed to the generation units which in turn reconstruct the image by placing patches of data in the vicinity of the entity they represent. At this point we mention that the recognition units can “see” the hole image and also that the unit's activations are multiplied by the probability p so that if an entity is not spotted, the capsule’s contribution to the image reconstruction is zero.

In the more complex case, the instantiation parameters are 9 so as to encode information of a full 2D affine transform of the detected entity. What is more, each capsule is spatially constrained to a 11x11 receptive field in the image. Deltas are replaced by transformation matrices. There is no weight sharing nor a reference to a second capsule layer although this may be possible. The images used were 2D instantiations of 3D models of cars and busses.

**Discussion**

Two of the limitations highlighted are the use of the transformations between the input and output image and the incapability a capsule to represent two instances of the same entity at the same time (crowding). Later, the authors mention the differences that capsule networks have to convolutional pooling and Kalman filters. Finally, some prospects/features of capsules are highlighted. The most important one being that we can force the outputs of a capsule to represent any property of an image that we can manipulate in a known way.

Dynamic Routing Between Capsules [2]

**Abstract**

A short revision of capsules is made in the abstract: a capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity that the capsule has learned to recognize. Contrary to [1], the presence probability is represented by the length of the activity vector (and not by a separate number computed by a logistic unit). Capsules in the next layer are selectively activated by capsules from the previous layers. As mentioned in [1], predictions of the activations of the next layer’s capsules are made by multiplying each activity vector of the previous layer by a learned transformation viewpoint-invariant matrix that describes the part-whole relationship. This paper provides a routing algorithm for the activations between the capsule layers. The capsules in a lower-layer discriminatively choose to send their outputs to capsules in the upper layer depending on the matching of their vectors.

**Introduction**

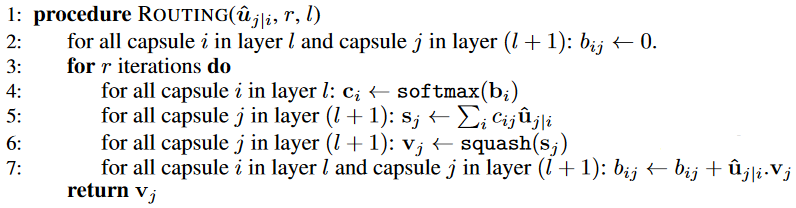
When an input – stimuli is given to the capsule network that uses dynamic routing, a parse tree is carved out by selectively activating some capsules in the network (capsules become nodes of the parse tree). The information is passed from capsule to capsule in the parse tree in the form of a vector, whose orientation encodes the properties of the entity (instantiation parameters like pose, illumination, etc.) and whose length represents the activation of that capsule. The length cannot be greater to 1.

How is the parse tree formed? This is where the Dynamic Rooting Algorithm comes to place. Instead of using a max pooling layer that discards all information but the most active feature in a local pool in the layer below, the rooting algorithm preserves valuable information about precise location and other instantiation parameters. Dynamic Rooting Algorithm is an iterative algorithm that computes coupling coefficients between capsules in adjacent layers. These coupling coefficients, along with the predictions are then used to compute the activity vector of the next layer. The algorithm inputs the predictions of each capsule of the previous capsule layer (predictions of a previous capsule = activity vector \* transformation matrix of each next-layer-capsule). The coupling coefficient between a previous-layer capsule and a next-layer capsule is analogous to the product of the prediction and the activity vector (of the capsule in the next layer). Note that in the first iteration, all the activity vectors are equally initialized. Later, they are dynamically updated by the predictions, weighted by the coupling coefficients. Consequently, we can infer that the coupling coefficients shape the activation vectors of the next layer capsules (forward parsing) and the activation vectors (along with the predictions for those activations by the previous layer) shape the coupling coefficients (local backward parsing). More details on the rooting algorithm will be presented later.

Capsule Layers can be convolutional. That is the case with all but the last layer in the network presented by this paper. This is a way to replicate learned knowledge across space. In this way, if a capsule learns to recognize an entity (e.g., an eye), it can be used to recognize the same entity in another position of the image. It is important to mention that for low level capsules, location information is place coded by which capsule is active in the grid. On the other hand, as we move towards higher level capsules that have wider receptive fields and represent more complex entities, these information become place coded.

**Dynamic Routing**

Below we present the rooting algorithm.



At first, we define the squashing function: (TODO: insert squash function)

Secondly, we need to explain in more detail how the votes are produced. For each parent-child capsule pair there is a wight matrix. The multiplication of this matrix by the activity vector produces the prediction for the capsule of the layer above. TODO: insert equations.

Lastly, we explain the algorithm line by line.

1. The algorithm takes as input the predictions u, the rooting iterations r and the layer from which the predictions come from, l.
2. At first it initializes all the b\_I\_js to zero. B\_I\_js are the logits that capsule I is assigned to capsule j in the next layer. These logits become the coupling probabilities when passed to the SoftMax function. The initialization of b\_I\_js can alternatively be learned.
3. Repeat for r iterations. Usually, r is equal to 2 or 3.
4. Compute the coupling coefficients c\_I\_j.
5. Computes a primitive stage of the activity vector for each capsule in the next layer as a weighted sum of the predictions.
6. Squashes the primitive activity vectors so as to have length smaller than 1.
7. Updates the logits b\_I\_j according to the similarity between the prediction -that a lower level capsule has made for the activity vector of the capsule in the next layer- and the actual activity vector - of the capsule in the next layer.

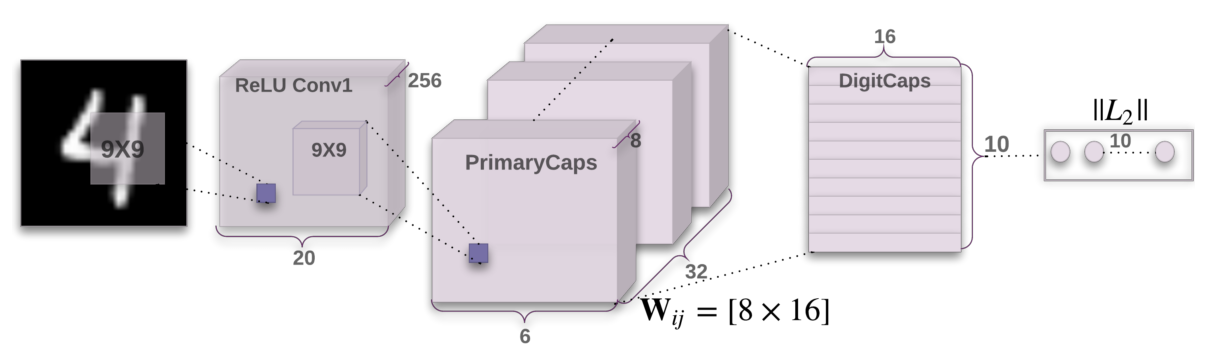
**Marin loss for digit existence**

We use a different margin loss L\_k for each digit capsule (in the last capsule layer). Lambda down weighting is used to prevent the network from shrinking all the output vectors.

{insert equation}

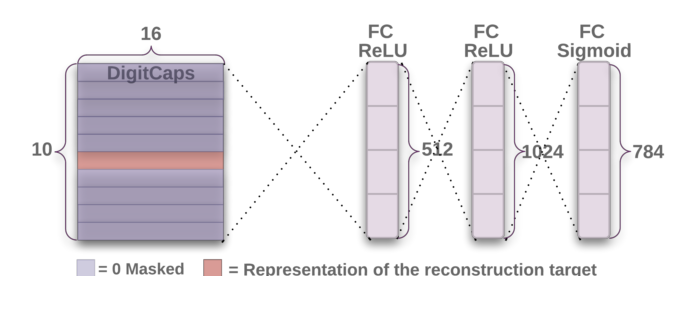
**CapsNet Architecture**

The network consists of two parts, the encoder and the decoder. We present the encoder below.



The encoder takes as input the 28x28x1 image from MNIST dataset and encodes it into a 16-dimensional vector of instantiation parameters. The first layer (Conv1) of the architecture is a simple convolutional layer with 9x9 kernel with 256 output channels and a stride of 1. This layer converts the pixel intensities to activations of local feature detectors. The primary capsule layer is a convolutional layer that performs inverse graphics. More specifically, it tries to encode the instantiation parameters of the object parts detected by the previous layer. Actually, the primary caps can be regarded as a convolutional layer with a kernel of 9x9, a stride of 2 and 32x8 channels stachked together as batches of 32 8D channels of 6x6 grids. It is important to note that no routing takes place between the two layers because Conv1 layer is a simple, 1D layer that does not have vectors. The last layer is the DigitCaps layer of 10 16D (16D=activity vector’s dimension equals to 16) capsules. The norm of each activity vector in this layer indicates the probability of the corresponding digit being present in the image. The computation of the activity vectors in the last layer is done by dynamic rooting. Every capsule in the last layer can receive predictions from every capsule in the Primary caps layer (fully connected). Note that there is a separate W matrix (the transformation part-whole matrix) for each pair of capsules, I.e for all I in [1,6x6632] and j in [1,10].

We present the decoder below.

The decoder is used as a way to regularize the results and enforce the learning of robust instantiation parameters. Note that during training, all but the correct digit is masked out using the true label. The loss is computed as the sum of the squared differences between the output (estimated) image and the actual input image. The loss is scaled by a term so as to not overshade the margin loss.

**Experiments**

For training, Adam optimizer is used along with exponentially decaying learning rate. A handful of datasets where tested, including MNIST, affNIST, MultiMNIST, CIFAR10, SVHN, smallNORB.

In the first three datasets, the results were very positive. Considering the number of parameters, the network was able to achieve high classification accuracy with relatively small number of parameters. In the case of MultiMNIST, the results were the same compared to the state-of-the-art model which was tested in a much easier case of only 4% overlap (compared to 80% overlap of digits that the CapsNet was tested on). Note that for reconstruction in the case of multi-classification, the corresponding true DigitCaps are fed to the decoder separately and never at the same time. One drowback of the capsules is that they tend to try to explain everything in the image. Consequently, their results in dataset with complex backgrounds like CIFAR is relatively poor (1.6% test error on CIFAR10).

**Discussion**

Capsule networks try to address the inefficiencies of the convolutional neural networks. It is supported that neural networks by the convolution operation can use the same feature detectors to recognize entities regardless of their position in the image, thus, being translational equivariant. However, regarding the other affine transformations CNNs are inefficient. One has to multiply the size of the dataset by inserting affine transformed images and also multiply the number of feature detectors. In a way, the network learns to recognize two different rather than the same object but with a different pose.

This paper provided the Dynamic Routing algorithm which lets the network recognize more complex objects by recognized object parts. This algorithm has similarities with the attention mechanism.

Capsules are competent in segmentation tasks. However, the crowding liability discussed in the [1] still remains.

Matrix Capsules with EM Routing [3]

**Abstract**

Capsules are groups of neurons that learn to recognize entities. Each capsule is entangled with a single entity and outputs it's presence probability and a matrix of different properties. Capsules are organized in layers. When two capsule layers are adjacent, a rooting algorithm takes place which forms the outputs of the next layer. Each capsule in the previous layer places a vote on many capsules in the next layer by multiplying its entity-property matrix with a learned viewpoint-invariant transformation matrix. The output for each capsule is then formed by the sum of the votes weighted by weighting coefficients (computed by the E-M algorithm). In this way, capsules in the next layer receive outputs from a cluster of similar votes in the previous layer. In a way, the intuition is that parts pertaining the same entity are clustered together.

**Introduction**

Traditional CNNs replicate their feature detectors across space by tying the weights of the convolutional matrices across space. Convolutional capsule networks extend this idea by sharing knowledge of part-whole relationships across locations. This knowledge lies within the transformation matrices by which the votes are computed. It is a fact that an affine transformation of an object changes the pixel intensities in an irregular way, the instantiation parameters of the object, encoded by the capsule’s output matrix changes in a coordinated way. This is why the values in the matrix are viewpoint equivariant.

It is stated that capsules use high dimensional coincidence filtering. This means that objects are detected by agreement between the votes of the parts. For example, if two parts agree on the pose of the object, the capsule which represents that object is more probable to be activated. An iterative process called “rooting by agreement” is responsible for estimating the probabilities of assigning parts to wholes. The assignment is based on the proximity of a vote to the other votes from other parts to this whole.

**How capsules work**

In this paragraph we introduce the notation used later. Ω\_L denotes the set of capsules in layer L. V\_I\_j is the vote from capsule I in the previous layer to capsule j of the next layer. As discussed before, V\_I\_j = Mi \* W\_I\_j where Mi is the matrix of capsule I. In typical neural networks, a non-linear function is applied to the scalar output of a linear filter. In capsule networks, this is performed by a complex non-linearity (E-M based procedure) that receives as input the votes and activations of the capsules in the previous layer and outputs the corresponding parameters of the capsules in the next layer.

**Using E-M for rooting by agreement**

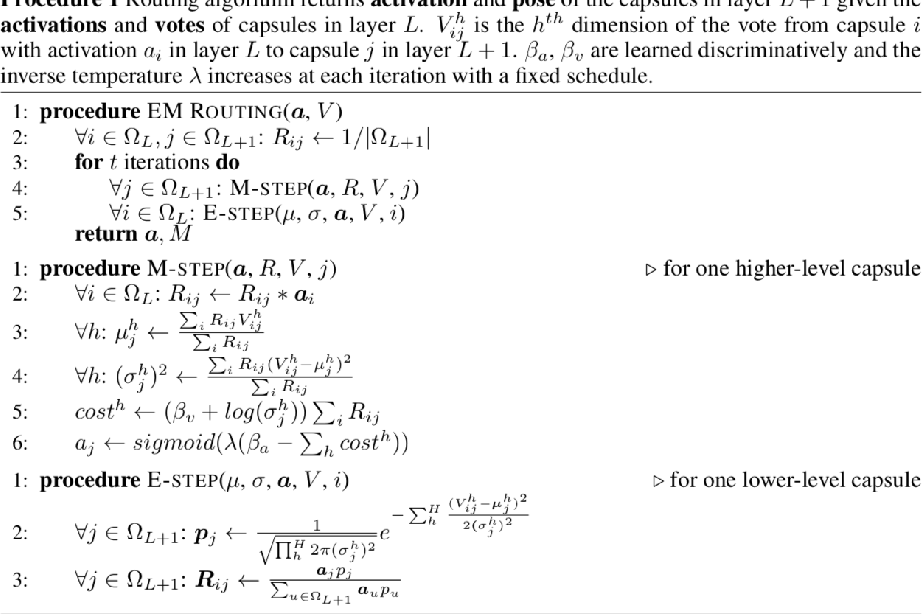
For rooting the votes of lower-level capsules to those in the higher-level, a rooting process takes place that resembles the EM rooting algorithm. It is helpful to consider the higher capsules as Gaussians that try to fit the datapoints which in this case are the votes of the lower capsules. The goodness of the fit is measured using a cost that is inspired by the minimum description principle. For every higher capsule, we have to decide whether to activate it or not. If we don’t activate it, we pay a fixed cost of -β\_u per data point for describing the votes of all the lower-level capsules that are assigned to the higher-level capsule. For partial activations of the capsules, this cost is scaled down. If we choose to activate it, we pay a fixed cost -β\_a for coding its mean and variance and also pay additional costs, pro-rated by the assignment probabilities, for describing the discrepancies between the lower-level means and the predictions for those means by the higher-level capsules (produced by multiplying the inverse transformation matrix by the higher-level capsule’s mean). While this is correct, it is computationally inefficient. Thus, instead of estimating every lower-level capsule’s mean, we compare the votes of the lower-level capsules to the means of the higher-level capsules. This is not a problem as the transformation matrices are discriminatively trained. Note that a capsule’s mean is actually its pose matrix but reshaped as a vector. Under this modification, the description cost of a datapoint by an upper capsule is sum\_h (–log(Pi\_j\_h)) where h is the hth dimension of the vector and P\_I\_j\_H = [TODO insert formula].

Sum(r\_I\_j) is the amount of data assigned to j.

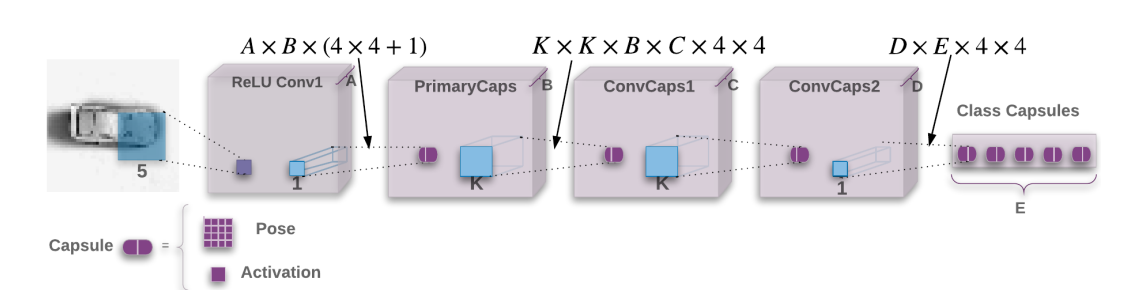
The activation of an upper capsule is computed by the following formula (TODO)

Lambda is an inverse temperature parameter, β\_a and β\_u is learned discriminatively.

Next, we present the EM routing algorithm.



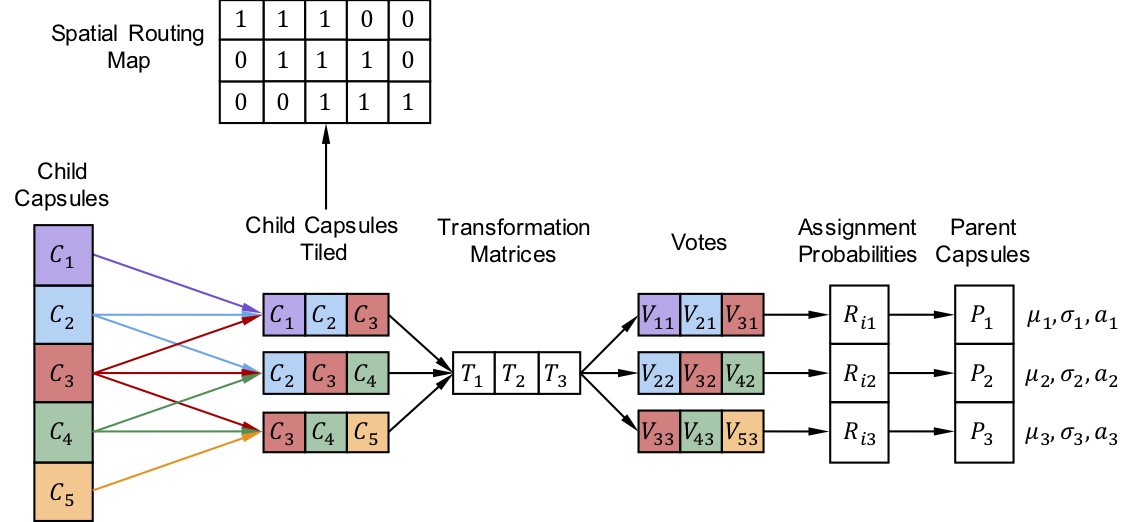
**Architecture**



* Conv1: Typical convolutional layer with a kernel of 5x5, A=32 channels, stride of 2 and ReLU non linearity.
* PrimaryCaps: It is comprised of B=32 layers of capsules which are produced by simple convolution with 1x1 kernel. The output channels of this layer is 4x4x32 so as to produce 32 layers of 16D capsules (4x4 pose matrix). The activations are produced by a 1x1 kernel with 32 channels output that applies a sigmoid non-linear function onto the ReLU layer.
* ConvCaps1: convolutional capsule layer with kernel 3x3 with C=32 capsule types.
* ConvCaps2: convolutional capsule layer with kernel 3x3 with D=32 capsule types
* Class Capsules: The number of them equals the number of output classes. It is a fully connected layer.

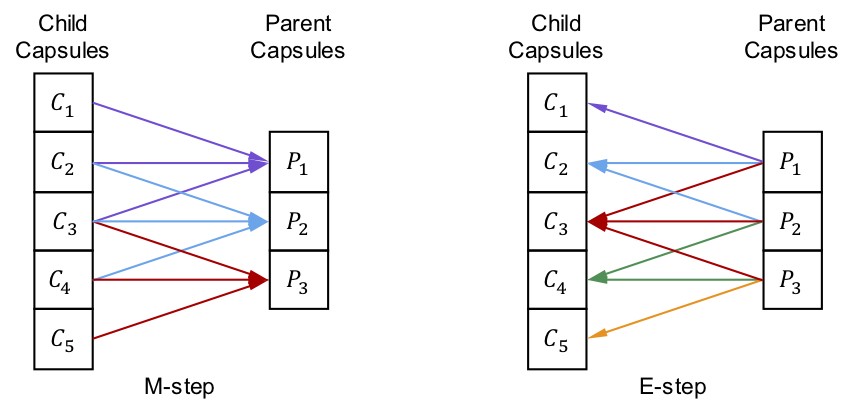
Some notes regarding the architecture.

Between two consecutive capsule layers, EM routing algorithm takes place. So, everywhere else but between the Conv1 and PrimaryCaps, EM routing is used to compute the next capsule’s parameters. The running of EM routing with convolution is not intuitive. That is why we provide the following image for the simple case of 1D convolution with routing.



We first unroll the convolution by tiling the child capsules (meaning their outputs). Note that the weights are shared in a layer so that if two capsules are in the same layer, vote for the same next layer and share the same spatial dimension (e.g., upper left in the 3x3 kernel) then they share the same transformation matrix. This allows for knowledge transfer across space. In the original case, the number of different transformation matrices between two capsule layers is “number of lower layer capsule types” \* “number of higher layer capsule types” \* “kernel width” \* “kernel height”.

The picture below shows that the EM routing, as the Dynamic routing is an iterative process that has forward and backward steps. In this case, the parent capsules send feedback during the E step to the capsules from which they received a vote. Some corner capsules receive only one vote. It is easy to see which capsules compete to which parents. It is not forbidden for a child capsule to be fully assigned to two parents.



One last note: when computing the votes for the last layer, we don’t want to lose location information so we do a technic called Coordinate Addition by which we add the coordinates of the center of each capsule’s receptive field to the right-hand column of the vote matrix.

**Spread loss**

The loss used for training the capsule network is the following (TODO)

**Experiments**

**Adversarial Robustness**

**Conclusion**

Capsule Routing via Variational Bayes

**Abstract**

In the abstract, we are reminded of what capsules are. Capsules are groups of neurons which recognize entities in the image and output a vector or a matrix representing their properties. The entities are parts or objects. The lower-level capsule layers recognize parts that are then transformed by viewpoint-invariant transformation matrices that map the individual parts to their relative pose within the whole object’s frame. The presence of an object is determined by the level of agreement of the object’s parts. The agreement process is called “routing by agreement”. Various routing algorithms have been suggested. This paper proposes a new algorithm derived from Variational Bayes that transforms the capsule networks to Capsule Variational AutoEncoders (VAE). This method addresses some weaknesses of the previous algorithm like the variance collapse problem and achieves significant results.

**Introduction**

The general idea behind the capsules is that they try to understand the underlying structure of the entities in an image. Anything that has a consistent underlying structure across viewpoints can be considered as entity. By making the assumption that objects are comprised of parts, and learning the equivariant relationships between parts-objects capsules are able to generalize to novel viewpoints. They keep the convolution process which is an efficient method for transferring knowledge across space but they discard the pooling layer. Instead, they use a routing by agreement method that resembles the attention mechanism. As it may be already known, each capsule poses a vote V\_I\_j to each capsule in the upper layer. The vote is computed by (TODO) where M\_i in R^4 is the capsules parameters (encodes the part’s pase with respect to the viewer) and W\_I\_j is the transformation matrix (encodes the part to object relationship). The parameters of the new capsules can be computed by simply taking the mean of votes (TODO equation) where R\_I\_j is the posterior responsibilities of each capsule j for capsules I (computed by various algorithms, e.g., dynamic routing).

The motivation for this paper were the weaknesses found in the routing algorithm based in EM. A probabilistic approach derived from Variational Bayes can overcome the liabilities and provide more flexible control over capsule complexity by tuning priors to induce sparsity and reducing the variance collapse singularities in EM. The proposed method models uncertainty over the capsules parameters as well as the routing weights.

**Variational Bayes Capsule Routing**

[1] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming Auto-encoders.” Accessed: Jan. 18, 2021. [Online].

[2] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic Routing Between Capsules,” 2017. Accessed: Jan. 17, 2021. [Online].

[3] G. Hinton, S. Sabour, and N. Frosst Google Brain Toronto, “MATRIX CAPSULES WITH EM ROUTING,” Feb. 2018. Accessed: Jan. 17, 2021. [Online].