Capsule Networks

Rico Meinl
Computer Science (B.Sc.), University of Applied Sciences Wedel

06.06.2018

Abstract

In this seminar paper I will describe the motivation behind Capsule Networks including its advantages over Convolutional Neural Networks which are the current state-of-the art technique for object detection and image classification. After giving an intuition about the concept of Capsules I describe and evaluate the CapsNet architecture proposed by Sabour et al. [7]. Finally I refer to another paper [6] which applies a stress test to this architecture and use it to draw a conclusion.

1 Problems with Convolutional Neural Networks

Convolutional Neural Networks are the current state-of-the art technique for object detection and image classification respectively. On a high level, the idea behind these kind of networks is to propagate images by detecting the main features and thus being able to reduce their size while keeping the most important information.
To achieve that CNNs use early layers to detect simple features like edges, borders or curves in an image and combine those features into more complex features while going deeper into the network. Finally the dense layers at the top take the combinations of high level features and produce classification predictions.
Compared to other image-classification algorithms they use relatively little preprocessing and the filters that traditional algorithms apply are learned by the network.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. Their basic building blocks are convolutional, pooling and fully connected layers.
To reduce the size of the input image and also increase the "field-of-view" of higher level Neurons one usually applies Pooling or successive Convolutional Layers.
The Convolutional layers are specified by their kernel size, which is the size of the window we're using to go over the whole image and try to detect the features. We place the window in the top left corner and slide it across all pixels in the image using the specified step size and multiply each pixel by a set of weights, then add up all the values in the window. As a result the network learns filters that activate when it detects some specific type of feature at some position in the input. Their filter size specifies how many filters are being used on the image and their stride size defines how many pixels are taken at one time.
The Max Pooling operation is the most commonly used Pooling operation, followed by Average Pooling. It looks at a frame which is specified by a kernel size and takes the maximum pixel value in that frame, which scales down the image but keeps the important features.

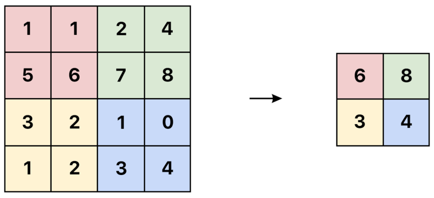


Figure 1: Max Pooling

The intuition behind that is that the exact location of the feature is less important than its rough location relative to other features. The pooling layer not only reduces the size and parameters and therefore the amount of computation in the network but also prevents overfitting.

Convolutional Neural Networks accumulate sets of features at each layer to achieve a smaller representation of the picture without loosing the essential meaning and up until now the Pooling operation has constantly been used in architectures.

The problem with this way of reducing the size of an image is that we create an invariance of activities which means that if we shift our input only by a little our Network will still be able to detect the object. According to Geoffrey Hinton this is not what we should aim for and he argues that CNNs show major weaknesses when the orientation of the image changes. They are also easily fooled by features in the wrong place.
The issue he is addressing is that the Pooling operation loses important information by not encoding spatial relationships between the features. In essence the internal data representation doesn't take spatial hierarchies between simple and complex objects into account.

"The Pooling operation used in Convolutional Neural Networks is a big mistake and the fact that it works so well is a disaster" - Geoffrey Hinton

So far people have hacked their way around the problem of invariance by using excessive training of all possible angles using lots of data. It seems like the only downside of this is that it takes a lot of time. The real downside though is that it is counterintuitive and our brain surely doesn't work that way.

"Convolutional Neural Networks are doomed" - Geoffrey Hinton

Hinton argues that the right way of tackling this issue is to not aim for invariance but equivariance, which essentially means that neuronal activities change when an object moves over the manifold of possible appearances.
The network is then able to understand that what it sees is just another view or angle respectively of something it has seen before.

2 Inverse Graphics

The idea is basically taken from the ways modern Computers graphics use to deal with constructing visual images from internal hierarchical representation of geometric data.
3D figures are stored in memory as arrays of geometrical objects and matrices that represent relative positioning and orientation of these objects. They are then rendered and put on screen.
Capsule Networks are a new type of Neural Networks that tries to perform inverse graphics, taking relative positioning of objects into account.
The motivation behind this is that the representation of objects in our brain doesn't depend on the view angle.

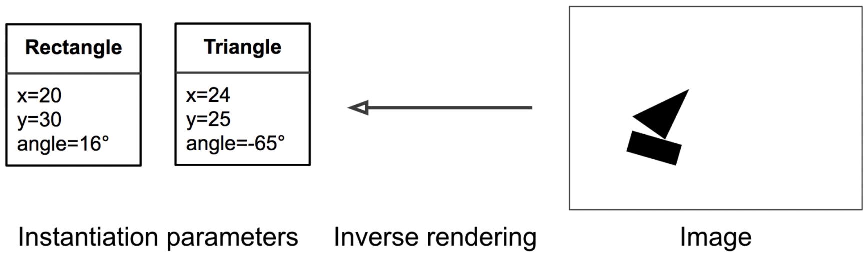


Figure 2: Inverse Rendering Process

Our eyes receive the visual information, which is used by the brain to create a hierarchical representation of the world around us. It then tries to match it with learned patterns and relationships already stored in our memory.

"Our brain does the opposite of rendering" - Geoffrey Hinton

The theory behind Capsules tries to achieve this by representing objects internally as a 4D pose matrix. Doing that, it is possible to preserve hierarchical pose relationships between object parts. An approach like this only needs a fraction of the data and is therefore much closer to how our brain actually works.

3 Architecture

To explain the idea behind Capsules and the Dynamic Routing Algorithm I will make use of the architecture from the paper [7].

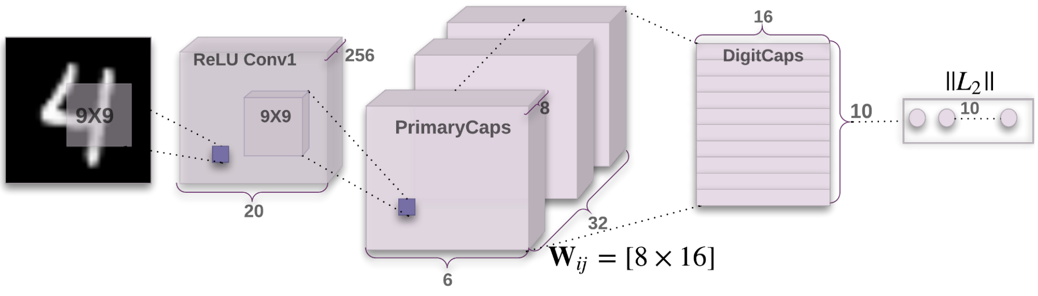


Figure 3: CapsNet architecture

The main parts of the architecture are a Convolutional Layer followed by the Primary Caps and Digit Capsules.
The output from the Digit Capsules is then used to compute the margin loss and additionally serves as the input to a decoder part which are three fully connected layers that try to recreate the input image from the learned feature representation.
The MNist dataset, which is being used in the paper is a common benchmark for Deep Learning models.
Its training set contains 60.000 images and its test set 10.000. Each image contains one handwritten digit (0-9).
Usually images have three channels (red, green, blue) but MNist images are gray-scaled so they only have one channel and a height and width of 28 pixels. Each pixel naturally has a value between 0 and 255, the brighter the pixel the larger the value.
The input image first goes through a Convolutional Layer which uses a kernel size of 9 x 9 pixels and 256 channels. The essence of this operation was already touched in chapter 1 on page 1.
We place this window in the top left corner of our image and slide it across all the pixels to extract some extremely basic features.
Using backpropagation the Neural Network is trained to pick appropriate weights for the kernel.
We then apply the ReLU (Rectified Linear Unit) function to our output to break the linearity. The ReLU function enables a better gradient propagation with fewer vanishing gradient problems compared to the sigmoid activation function for example. It is specified by the following formula:

$$x = max(0, x)$$

To give a better understanding of the dimensions we're working with I will continuously calculate the values of the output dimensions. The variable m specifies the batch size of input images.
Our input X was of size: $(m, 28, 28, 1)$
When we apply a convolutional layer the formula to calculate its output is:

$$\lfloor\frac{W - F_w + 2P}{S_w}\rfloor + 1 \quad * \quad \lfloor\frac{H - F_h + 2P}{S_h}\rfloor + 1 \quad * \quad n_c$$

So when we substitute the variables with our values we get:

$$(m, 20, 20, 256)$$

The Primary Caps Layer starts off with another convolutional layer. We use another kernel with the size of 9x9 pixels but with a stride of 2 this time to convolve over the outputs from the previous convolution.
Essentially this operation looks for more complex shapes from the edges, curves, etc. found earlier and takes those basic features and produces combinations. After applying the formula specified above we get an output of:

$$(m, 6, 6, 256)$$

In a regular Convolutional Neural Network one would now use a Pooling operation preferably Max Pooling to reduce the size of our output while still keeping the essential information.
As discussed earlier this method ends up loosing valuable information and doesn't take into account any spatial relationships between features.
So in this approach we cut up our output into 32 stacks with 8 capsule layers per deck. Each capsule layer has 36 capsules and each capsule is a 8x1 vector.

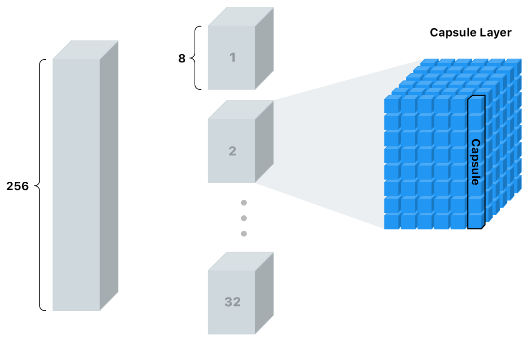
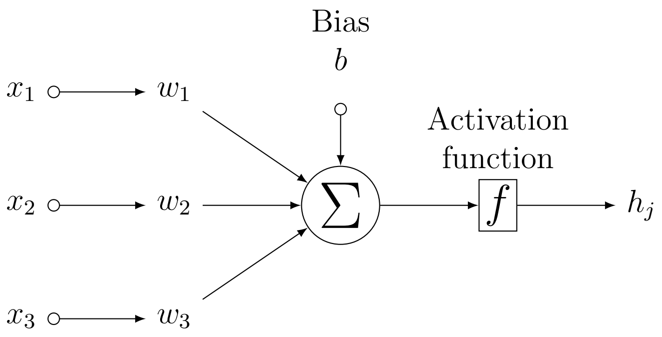


Figure 4: Reshape to Capsules

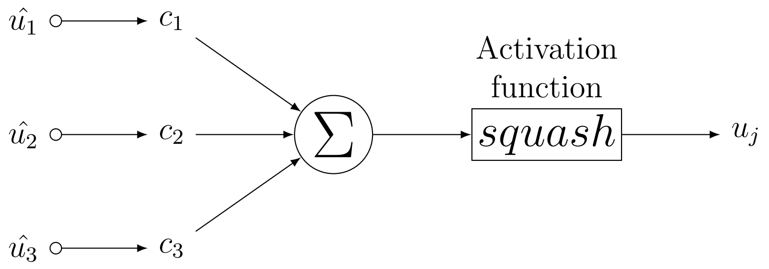
3.1 Capsules

Capsules encapsulate all important information about the state of the feature in vector form. This allows us to store more information than just wether or not we found a shape in that spot.
Each vector dimension represents a so called instantiation parameter.
This information is contained in the angle of the vector and could be the pose (position, size, orientation), deformation, velocity, thickness, etc. of a specific feature.

The length of the vector represents the probability of detection. When the detected feature moves around the image now or changes its state the orientation of the vector will change but its length and therefore probability will stay the same. Thats exactly what Hinton referred to when talking about equivariance.
To delve deeper into the idea behind capsule I want to clarify the difference between Artificial Neurons and Capsules.



Neurons receive input scalars from other neurons and multiply them by a scalar weight w. They sum up these values and pass them to a non-linear activation function which takes an input scalar and outputs a scalar.
The design of Capsules builds upon that:



$$\hat{u}_{j|i} = W_{ij} u_i$$

They receive input vectors from the capsules in the layer below and multiply them by transformation matrices. The output is then weighted, summed up and passed to the Squash Function which is a non-linearity function taking a vector as input and outputs another vector.

We can break down this procedure into four steps:

1. Matrix Multiplication of Input Vectors
2. Scalar Weighting of Input Vectors
3. Sum of Weighted Input Vectors
4. Squash

1. Matrix Multiplication of Input Vectors

The capsule receives the input vectors from the capsules in the layer below. For the example of a face the lower capsules could represent eyes, mouth and nose and the higher level capsule the face.
We then transform the vectors with a transformation matrix W to form a vote, which essentially means that each lower level capsule tries to predict the position of the higher level feature.
Looping back to the example of the face, if our eyes, mouth and nose capsules each agree on the parameters (position, state, etc.) of the face we detect a face.
The network will learn the transformation matrix for each pair of lower and higher level capsules. For our example that would be the relationship between the eyes and the face or: Where are the eyes in a face?

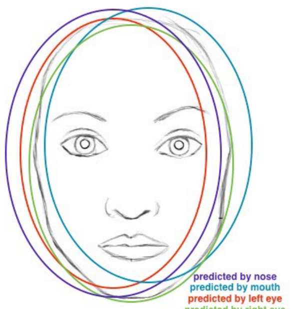


Figure 5: Predicted Position of higher level feature

2. Scalar Weighting of Input Vectors

After calculating the votes from each capsule we want to determine where each capsules routing goes. On a high level that means that each lower level capsule needs to decide to which higher level capsule it will send its output. In the case of Artificial Neurons we learn those weights by backpropagation.

In Capsule theory we use the Dynamic Routing algorithm which I will further explain in chapter 3.2 on page 10.

3. Sum of Weighted Input Vectors & 4. Squash
We take the sum over all of our prediction vectors and send the output vector to the squash function which is a novel non-linear activation function for vectors. It takes the input vector and squashes it to a length between 0 and 1 without changing its direction. To achieve that it uses the following formula:

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

By drawing the comparison between Neurons and Capsules we are able to observe that the design of capsules essentially build upon artificial neurons. With capsules we expand our feature representation to vector form which allows us more powerful representational capabilities. We also introduced the transformation matrix W to encode the spatial relationships between features of different layers.

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector$(\mathbf{u}_i)$	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j\mid i} = \mathbf{W}_{ij}\mathbf{u}_i$	–
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\mid i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\lVert\mathbf{s}_j\rVert^2}{1+\lVert\mathbf{s}_j\rVert^2}\frac{\mathbf{s}_j}{\lVert\mathbf{s}_j\rVert}$	$h_j = f(a_j)$
Output		vector$(\mathbf{v}_j)$	scalar(h_j)

Figure 6: Capsules vs. Neurons

With that intuition we are able to understand the following breakdown of the output dimensions.
We split up the output from the last convolutional layer which was $(m, 6, 6, 256)$ into 32 stacks with 8 layers of capsules per deck.

That gives us:

$$(m, 6, 6, 8, 32)$$

Each capsule is a 8x1 vector so we reshape our dimensions to the following:

$$(m, 1152, 8)$$

Now we can clearly see that we have 1152 capsules of 8 dimensions.
The length of 8 is an arbitrary value chosen by the authors of the paper. For detecting the features of MNist this might work well but if we want to detect a feature like an eye or a mouth we would probably chose a bigger value.

As we try to predict 10 classes of digits we expand our dimensions, so that each capsule can make predictions for each digit class:

$$(m, 1152, 10, 8, 1)$$

Lastly we need to specify the weights of the transformation matrix W which we use to receive the votes from our lower level capsules. The authors chose following dimensions:

$$(m, 1152, 10, 16, 8)$$

So after multiplication our output has this shape:

$$(m, 1152, 10, 16, 8) * (m, 1152, 10, 8, 1) = (m, 1152, 10, 16, 1)$$

To figure out which output gets send to the next layer we define the Dynamic Routing algorithm.

3.2 Dynamic Routing

As mentioned above, each capsule tries to predict the next layers features. With our 1152 capsules and 10 output classes this leaves us with 11520 predictions.
Using the Dynamic Routing algorithm we want to figure out which information we will pass on to the next layer and which we will throw away.
This is essentially what happens between the Primary Caps and Digit Caps. In the following I will delve deeper into the intuition and implementation of the Routing algorithm.
We recall that each capsule tries to predict the presence and instantiation parameters of a particular object at a given location. We use the routing by agreement to figure out what outputs to send on, as a lower level capsule will only send its input to the higher level capsules that "agrees" with its input.

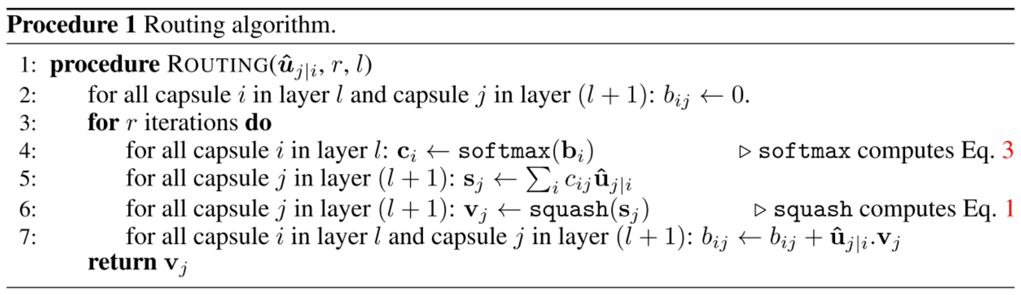
So after transforming the vectors of the input capsules with the transformation matrix W to form a vote we want to group the capsules with similar votes.
We first calculate the mean of all points (votes) and measure the distance between every point and the mean using the scalar product. This is our agreement measure to decide to which degree the predictions agree.
With that information we recalculate the mean, taking the importance of the votes into account. That means votes that are further away from the mean get small weights, while votes that are closer to the agreement cluster get high weights as they represent strong agreement.

Points that don't agree will eventually disappear (having a really small weight), because we only want to pass on relevant information to the next layer. The highest agreeing points get passed on with highest activations.
After giving this high level intuition I want to further explain the code implementation of the algorithm. Here's the procedure described in the paper [7]:

$$v_j = \frac{\|s^2\|}{1+\|s\|^2}\frac{s}{\|s\|} \tag{1}$$

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

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{3}$$



Procedure 1 Routing algorithm.

```
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$        ▷ softmax computes Eq. 3
5:         for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6:         for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$        ▷ squash computes Eq. 1
7:         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 7: Dynamic Routing algorithm

The goal of this procedure is to figure out the routing weights c_{ij}, which define a probability distribution of the output of a capsule belonging to a higher level capsule j.
The input $\hat{u}_{ij}$ is our input vector multiplied by the transformation matrix to form a vote or "prediction vector".

If this prediction has a large scalar product with the output of a possible parent we want to give top-down feedback and increase its contribution to the parent capsule.
All the capsules and their outputs in the lower level l are taken and after r iterations the routing weights will be established and the algorithm will produce the output of the higher level capsule j.
We start off in line 2 by setting the raw routing weights b_{ij} equal to zero for every predicted output.
The raw weights are a temporary value thats iteratively updated and are always reinitialised for each output calculation during training as well as testing. The routing weights take into account both the likeliness of detection and feature properties and are stored in c_{ij} after the procedure.
After this initialisation the for loop is defined which loops over the steps 4-7 for r times.
As mentioned above this loop tries to compute c_{ij} to quantify the connection between a capsule and its parent capsules. We want to learn the relationship between lower and higher level features.

The first operation inside the loop applies the softmax function specified in equation 3 for each primary capsule to get the value of vector c_i.
All our points start out with equal weights here which is equivalent to their importance. Before we have figured out any relationships the weights should be equally distributed over all the points.
In line 5 a weighted sum of predictions for each capsule in the next layer is computed. Intuitively this means we add our prediction together to produce the output vector s_j.
As there is a chance that this will get us a vector longer than 1 we pass this output vector through the squash non-linearity function to get v_j.
The squash function makes sure all vectors have a length between 0 and 1 taking their importance into account without changing their direction.
Finally in line 7 the temporary weights are updated by looking at each input and examining the corresponding weight b_{ij}.
Essentially we add the dot product of our capsule j and the input from a lower level capsule i on top of our old value. The dot product represents the similarity or the estimated agreement between the predicted output and the actual product vector.
This sums up the routing algorithm and the detailed explanation allows us once again to calculate the dimensions to give a better understanding what happens to our output vectors.

As we remember we get the following output from the Primary Caps:
$\hat{u}_{j|i} = W_{ij} u_i$ (m, 1152, 10, 16, 1)

During Loop the following updates happen:
$c_i = softmax(b_i)$ (m, 1152, 10, 1, 1)
$s_j = \sum_i c_{ij} \hat{u}_{j|i}$ (m, 1, 10, 16, 1)
$v_j = squash(s_j)$ (m, 1, 10, 16, 1)

3.3 Digit Caps

We end up with an output of 10 16-dimensional vectors, one for each digit class. The length of 16 is again an arbitrary choice and might differ if we choose to represent more complex features.
The length of each vector describes it's confidence of detecting a digit and each dimension once again represents an instantiation parameter. We then use the digit capsules to compute the margin loss and to also recreate the original input image. The paper [7] uses a special margin loss to detect two or more different digits in each image and is quite similar to the loss function used in Support Vector Machines.

3.4 Margin Loss

$$L_k = T_k max(0, m^+ - \|v_k\|)^2 + \lambda(1 - T_k) max(0, \|v_k\| - m^-)^2$$

$T_k = Y\ (one - hot - encoded)$
$v_k = Output\ from\ Digit\ Caps$
$m^+ = 0.9$
$m^- = 0.1$

We use a one hot encoded version of our labels Y which turns a label into a vector representation whose length is equal to the total number of classes and where only the correct label is 1 and all other labels are 0.
So if Tk is equal to 1 the loss function only evaluates the first part as the second part zeroes out. It then takes the output vector Vk and subtracts it from m^+ which is a value specified in the paper.
Intuitively this means that we only keep the value if the confidence of detecting the digit is higher than 0.9, because otherwise it will be smaller than zero.
We then square the resulting value, as this function uses an L2 norm.

For the case that Tk is equal to 0 the first part zeroes out and the function takes the output vector and subtracts m^- from it. This is essentially the same procedure as above but this time we only keep the value if the confidence of detection is lower than 0.1.
Additionally a coefficient $\lambda = 0.5$ is used for numerical stability during training.

3.5 Reconstruction

The digit capsules are also propagated through a decoder network which takes the 16-dimensional vector from the correct digit capsule and learns to decode it into an image of a digit.
Note that this mask is only applied during training to ensure it learns a correct representation and only encodes the most important features.
The idea of the reconstruction model is taken from an Autoencoder, where the encoder and decoder need to learn a good matrix representation to relate the relationship between the latent space and input.
The decoder network is a three layer fully connected network creating an

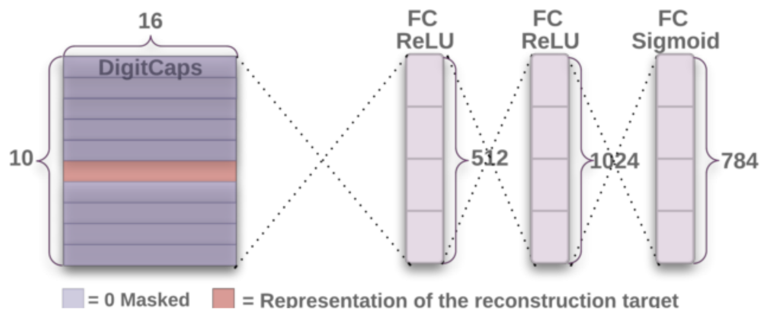


Figure 8: Decoder

output of 784 dimensions which is a flat representation of our 28 x 28 input images. A reconstruction loss is calculated by taking the euclidean distance between the reconstructed and the input image.

$$loss = margin\_loss + \alpha * reconstruction\_loss$$

This is then scaled down by $\alpha = 0.0005$ and serves as a regularising technique which forces the capsule to only learn features that are useful for reconstructing the original image.
When we tweak around one of the 16 dimensions of the digit capsules we can observe what each feature represents.

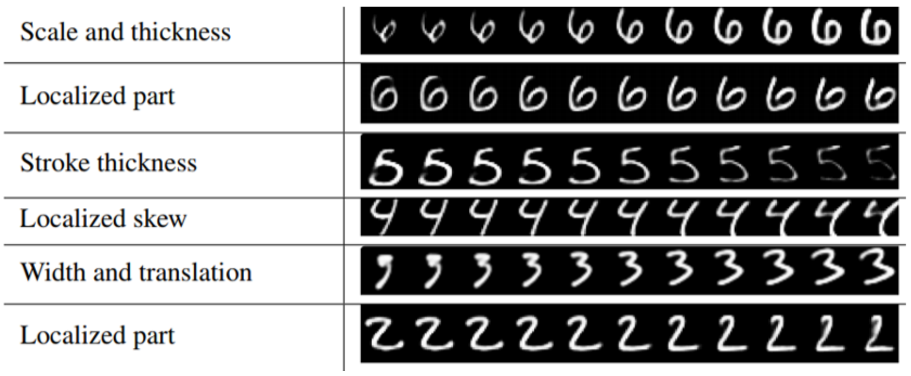


Figure 9: Reconstruction results

4 Overlapping Digits

The Dynamic Routing algorithm is a parallel routing mechanism, which enables Capsule Networks to detect multiple objects in an image even if they overlap. In the paper [7] the authors created the MULTIMNist data set where they overlayed a digit on top of another digit from the same set distribution but different class.
The images have about 80% overlay and are of size 36 x 36 pixels. The training set contains 60M training examples and the testing set 10M respectively. The two most active digit capsules are interpreted as the networks prediction. In the image below we can observe the results they achieved.

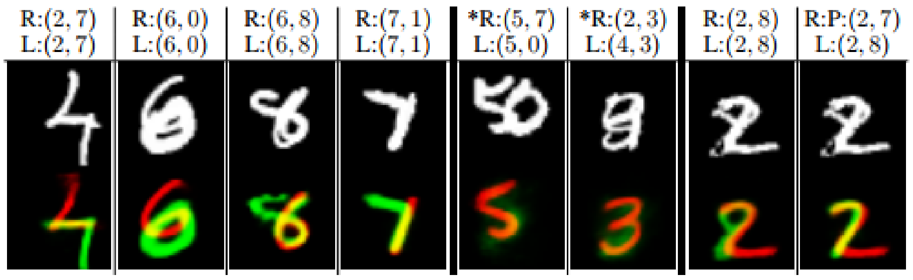


Figure 10: MultiMNIST

The upper image represents the input image and L (l_1, l_2) the true labels. R (r_1, r_2) are the digits that were used for reconstructing the image so essentially the predictions. It can be observed that the network does quite a good job segmenting the image into its original parts.
The authors are able to get a classification error rate of 5.0% which is the same as a previous paper whose images only had a overlay of less than 4%.

5 Discussion

During this paper we analysed the improvements that Capsule Networks offer in comparison to Convolutional Neural Networks.
CNNs show major difficulties in generalising to novel viewpoints. The choice between replicating feature detectors or increasing the training set size to avoid this problem seems counterintuitive and far away from how the human brain works.
Capsule Networks try to solve this problem of invariance by converting from pixel intensities to vectors of instantiation parameters to represent recognised fragments. They therefore use a distributed representation, an activity vector to encode essential features like the pose of an entity at a given location. They apply transformation matrices to those recognised fragments to predict the instantiation parameters of larger fragments in the layer above.
The Dynamic Routing algorithm is an improvement to the Pooling operation as it makes sure the network uses neural activities that vary as viewpoints vary rather than eliminating view points.
In addition to that Capsules can theoretically be combined to form hierarchical structures which are representing activities.
In this last part I want to discuss further results achieved with Capsule Networks and finally draw a conclusion based on their performance.

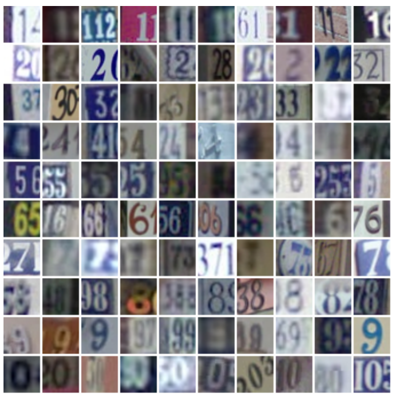
6 Stress Test

The paper [6] tried to push the limits on Capsule Networks by testing it on three additional datasets:
The goal was to find out to what degree Capsule Networks actually generalise to new data which is unlike that which they have seen before.
To test the assumption that they show robustness to new data the authors first created a new dataset with the following deformations:

1. Rotation (uniformly sampled angle within [-20°, 20°])
2. Shear (sheared along x, y axes by uniformly sampled parameters within [-0.2, 0.2])
3. Translation (translated along x, y axes by uniformly sampled parameters within [-1, +1])
4. Scale (by 150%)

(a) MNIST [5]

(b) Fashion-MNIST [8]


(c) SVHN [1]

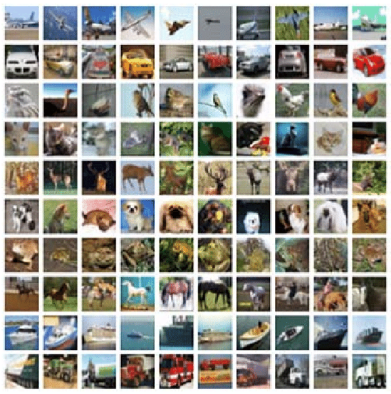
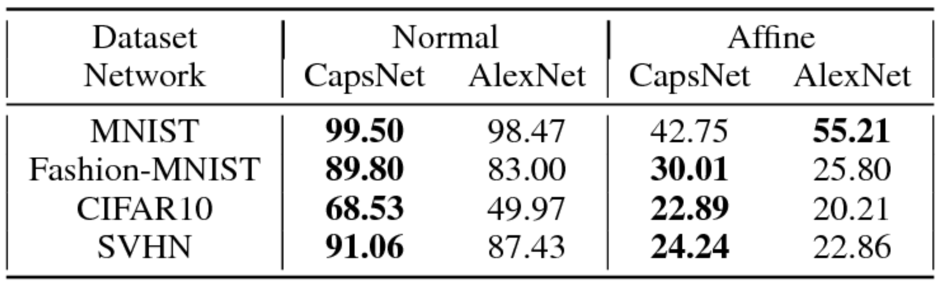

(d) CIFAR-10 [3]

Figure 11: Datasets

As a benchmark model the Convolutional Neural Network architecture AlexNet [4] was used. They trained the network on the original images from each dataset with 30 CapsNet iterations, where each one took about 4 hours. After approximately 120 hours of training time the tests were run with the affine-transformed datasets and achieved following results:

Something that can be observed right away is that the results are reasonable but far from state-of-the-art performance. A simple explanation for this might be that the hyperparameters for both networks were not chosen under optimal conditions nor did the authors spend much time optimising them. For AlexNet the authors used the default parameters and solely tweaked the learning rate. For CapsNets the architecture from the paper [7] was used, which is optimised for the MNIST dataset.



Dataset	Normal		Affine	
Network	CapsNet	AlexNet	CapsNet	AlexNet
MNIST	**99.50**	98.47	42.75	**55.21**
Fashion-MNIST	**89.80**	83.00	**30.01**	25.80
CIFAR10	**68.53**	49.97	**22.89**	20.21
SVHN	**91.06**	87.43	**24.24**	22.86

Figure 12: Stress Test results

One surprising results is how AlexNet outperforms CapsNet on the affinely transformed MNIST dataset. This could be due to the simplicity of MNIST though and the authors assume the AlexNet was simply overfitted.
A truly interesting outcome of the tests is that CapsNets has a larger drop on each dataset compared to AlexNet, which draws the question wether they are really better at generalising to deformed data.
For an architecture where this holds true it seems unusual to have such a large drop in accuracy after applying affine deformations to each image.

6.1 No. of Iterations

CapsNets was already converging and varying hyperparameters such as batch-size, learning rate, learning rate decay or momentum did not further change the final performance and thus would've only decreased training time.
The paper of Sabour et. al [7] recommends three iterations for the Dynamic Routing algorith but here the authors found out that two work just as well and even better in some instances. This is a signal that the routing algorithm might be too abrupt, because it seems unrealistic that a complex process like deciding which higher level capsule to route ones information to only needs two iterations.

6.2 Reconstruction

As was mentioned in 3.5 on page 14 Capsule Networks are capable of learning the essential features of an image and reconstruct it using a decoder that takes the output of the Digit Caps layer.

During their tests the authors found out that the classification accuracy plateaus earlier than the reconstruction accuracy which has them question if the reconstruction necessarily helps with regularisation and eventually classification.

In the results below the images on the left are the results after 2 epochs, the ones in the middle after 50 and the ones on the right show the ground truth:

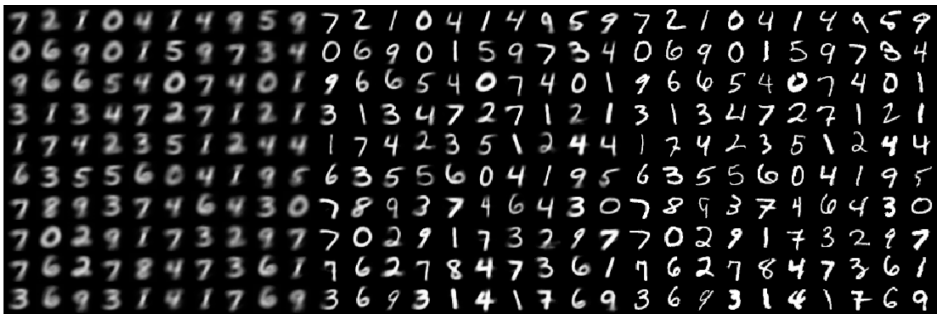


Figure 13: Reconstruction MNIST

For the MNIST set we are able to observe good improvements, mostly matching ground truth.

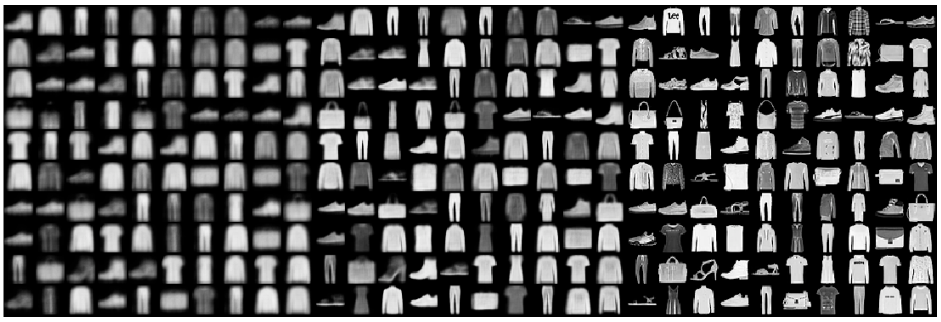


Figure 14: Reconstruction Fashion-MNIST

For the Fashion-MNIST set we can also see good improvements, but finer details are not captured. This is most likely due to the embedding being too small or the decoder too shallow respectively.
For the SVHN set the network goes from failing to being able to gray render. It still fails to replicate colours.
For the CIFAR-10 set the results are simply unrecognisable.

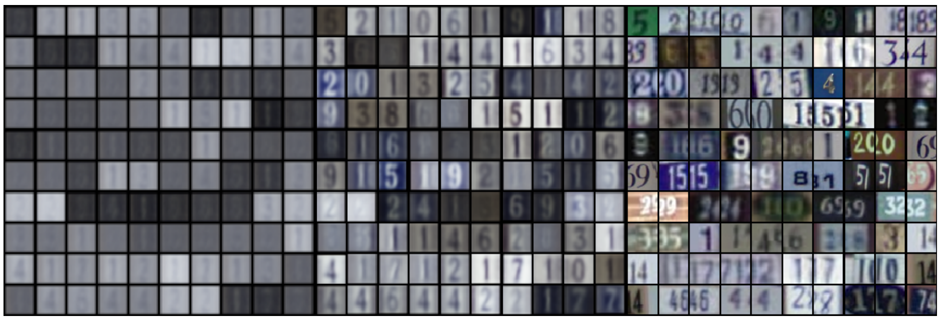


Figure 15: Reconstruction SVHN

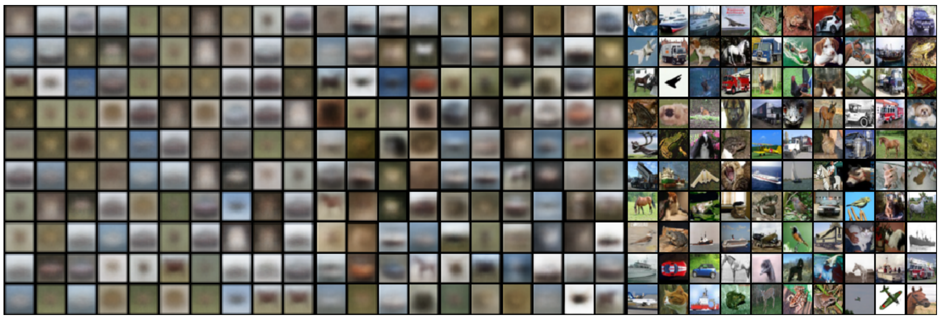


Figure 16: Reconstruction CIFAR-10

6.3 Performance

To draw a conclusion on the main findings on this paper: CapsNet is able to achieve better performance than AlexNet on datasets that are marginally harder than MNIST.
It appears to be not as good with deformations as initially thought and is slow during training. Another critical factor is that it still has not been tested on images larger than 32 x 32.
It is shown that the architecture is capable of learning useful information about spatial relationships, even if it's not making full use of the Dynamic Routing algorithm which we showed in 6.1 on page 18. There is simply no improvement in going from two to three iterations.
Regarding the reconstruction CapsNet was able to convincingly reconstruct MNIST and Fashion-MNIST samples where the embedding measured meaningful qualities like thickness or skew.
It failed to do so for SVHN and CIFAR where the embeddings mainly encoded intensity and color.

7 Conclusion

Finally it is safe to say that the concept of Capsules is intuitively appealing and the network is able to achieve reasonable performance on datasets like MNIST, Fashion-MNIST, SVHN and CIFAR-10.
The downsides explained in section 6 are mostly due to the architecture simply being the first implementation.
As stated in Sabour et. al [7] the purpose is "to simply show that one straightforward implementation works well and that dynamic routing helps".
Capsule Networks look promising for image segmentation and object detection and its routing by agreement procedure works great for overlapping objects.
The goal of this paper is to give the reader a taste of the intuition behind this concept and to lay the foundation for further research. At this points I want to refer to a different implementation where Matrix Capsules and a routing procedure called EM-Routing in Sabour et. al [2].

References

[1] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, abs/1312.6082, 2013.

[2] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

[3] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[4] 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.

[5] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[6] P. M. Nair, Rohan Doshi, and Stefan Keselj. Pushing the limits of capsule networks. 2018.

[7] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.

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