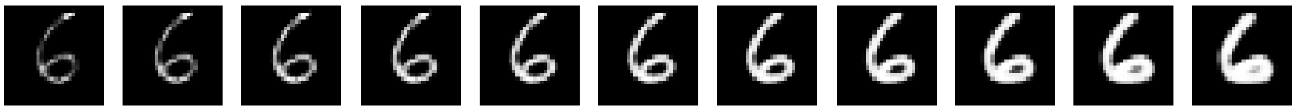
Understanding and Implementing CapsNet

Auss Abbood John Berroa Luke Effenberger

March 6, 2018



1 Task Description

We reimplemented the paper *Dynamic Routing Between Capsules* by Sara Sabour, Nicholas Frosst, and Geoffrey Hinton (Sabour et al., 2017). This involved creating a "capsule network" (CapsNet) within Tensorflow, and running various experiments such as classification and dimension perturbation that were implemented in the paper. We implemented it in a very general way to allow easy construction of architectures with different hyperparameters, than the one in the paper.

2 Related Work

The paper *Dynamic Routing Between Capsules* (Sabour et al., 2017) was the first to implement capsules, which were already proposed by Geoffrey Hinton in 1981 (Hinton, 1981). Because of it being almost brand new in the research field, there are barely any related works published yet.

There is, however, already a follow-up paper, called *Matrix Capsules with EM Routing* (Hinton et al., 2018) to the original that we implemented.

3 Theoretical Basis

Image classification is one of the many domains where artificial neural networks (ANNs) reign supreme. Ever since the ImageNet competition was won by Alexnet (Krizhevsky et al., 2012), an implementation of a convolutional neural network (CNN), CNNs have become the defacto standard in almost everything image related. Understanding the idea behind CNNs is critical to understanding how capsule networks build upon and improve them.

3.1 Convolutional Neural Networks

"Normal" neural networks, or feed-forward neural networks (FFNN), are a supervised learning algorithm and take in a vector of input, apply some number of non-linear functions to it (depending on the number of neurons), and outputs a result which, in the majority of cases, is a classification or regression of the input vector. This output is initially random, and improves over time through the use of the backpropogation algorithm. The algorithm finds a rate of change required for each weight in order to reduce the cost of the overall network, and gradient descent alters each neuron's weight by a small portion of this amount so that each neuron better "nudges" the output in the direction of the proper label. Over time, a local minimum should be reached, and with enough luck, such a minimum will solve the problem with a high amount of accuracy (Braun, 2017). However, feed forward networks have no spatial knowledge. If one must, they can input an image into an FFNN, but it would have to be flattened first into a vector, and thus lose the relationship between pixel locations in an image. That's where convolutional neural networks come in. The input into a CNN is the actual 2-dimensional image, and by using a process called convolution, features are extracted with weight kernels. Convolution (or in the case of CNNs actually cross-correlation) takes a weight kernel and slides it over the image. At each pixel, the neighborhood of pixels underneath the kernel are multiplied by their respective kernel weights, and summed. This becomes the value of the new pixel after convolution at that position. With this method, features can be found, such as edges. These features then go through a pooling layer, where the dimensionality is reduced and information in a certain neighborhood (usually 2x2) is summarized as one value (usually the maximum value). This is supposed to ease computation (due to the reduction in dimensionality), but also provide some scale and rotational invariance of *entire* features (Braun, 2017). With this model structure, image classification has become almost easy.

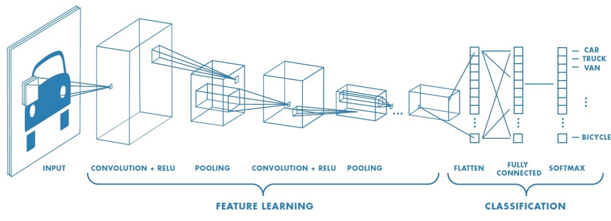


Figure 1: CNN Architecture (MathWorks, 2018)

Nevertheless, although CNNs are unbeaten in several tasks, they are not infallible. Issues such as adversarial attacks can completely fool a CNN. But one problem, that of spatial relations *between* features (explained below), can be solved through the use of a Capsule Network.

3.2 Capsule Networks

Let's start with a quote from Geoffrey Hinton:

> The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.

The problem with max pooling is that it is incorrectly responding to the following things. Either it does not recognize a rotated image, or it does not differentiate between "in-picture alterations" such as misplaced parts of objects. As we see in Figure 3.2, a normal boat is depicted on the left side. On the right side, something very similar to a boat is shown which, however, cannot be sailed. The problem is that the sails are rotated around the hull. If we are unlucky, max pooling would recognize the hull based on low level features and recognize both boats as proper boats, even though the relationship between features clearly disproves the second figure's "boatness." This problem of max pooling is overcome with the concept of capsules. A capsule in this example would recognize a hull and a sail, respectively. Capsule's responsibility is shifted to higher feature capsules—this means the hull-capsule on the left predicts a boat with the sail on top, and the sail-capsule votes for a boat with the hull on the bottom. The amount of agreement resembles the confidence of the network. A high agreement will elicit a high activity of the high level boat-capsule that looks like the boat on the left. Contrast this with the right image, where the hull-capsule would vote for a boat with a sail above it, but there is no sail. Thus, the capsules on the right side won't agree how the boat-capsule's activity should look like, since the sail-capsule will not vote for a boat. This will therefore yield a low agreement. The key point here is that spatial information is kept and incorporated into a whole.

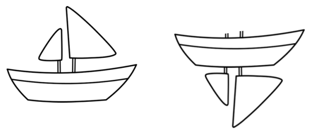


Figure 2: The Boat Example (Hackernoon, 2017)

4 Network Structure and Design

Capsule networks differ from "normal" neural networks in a profound way, especially in the way the input is propagated through the network. While there might be many different ways of implementing capsules and the forward propagation, the paper only explores one particular way of doing so. It should be seen more as a proof of concept.

4.1 CapsNet on MNIST

In this section we will explain the particular structure of the network from the paper (Sabour et al., 2017). While, as stated, it is only one specific implementation, explaining it in detail will help to understand the overall concept of capsule networks.

What is a capsule?

A capsule is basically just a group of neurons that belong together. From a technical perspective, this means we are looking at the activation of a capsule as a vector in $\mathbb{R}^n$. In theory, each capsule should learn to represent an entity (e.g. the boat, the hull or the sail in the above example). The length of the activation of such a capsule represents how sure the network is that this entity is present in the image, while the single components represent different properties (e.g. position, orientation, etc.) of the particular instantiation of the entity present in the image.

Squashing

As the input and the activation of a capsule are no longer only scalars but vectors, the paper introduces a new form of non-linearity—"squashing"—that is more suitable for vectors. If s_j is the input to a capsule, the activation of this capsule is

$$v_j = squash(s_j) := \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \in [0, 1] \tag{1}$$

Routing-by-Agreement

The 'Routing-by-Agreement' is the principle way how information is propagated from one capsule layer l with capsules of dimension m to the next capsule layer $l + 1$ with capsules of dimension n.

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 $l$: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
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 $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
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 3: The routing algorithm (Sabour et al., 2017)

Each capsule i in layer l is connected to each capsule j in layer $l+1$ with a weight matrix $W_{ij} \in \mathbb{R}^{m\times n}$ (i.e. fully connected). These are the weights that are learned during the training procedure. Each capsule i from layer l makes a prediction about the activation of each capsule j from layer $l+1$ based on its own output $u_i \in \mathbb{R}^m$. This is called the prediction vector

$$\hat{u}_{j|i} = u_i W_{ij} \in \mathbb{R}^n \tag{2}$$

These prediction vectors are the input to the routing algorithm (Fig. 3). The routing algorithm computes the coupling coefficients c_{ij}, which determine how much of their output each capsule from layer l can give to each capsule from layer $l+1$. To ensure that each capsule i from layer l can only give one "vote" in total, the coupling coefficients c_{ij} are computed by applying softmax to the logits b_{ij} for the fixed i, which are initialized to zero in the first iteration. (Note: The logits b_{ij} could also be learned during the training process, i.e. the network learns which capsules are more likely to be coupled.)

The input and the activation of capsule j can now be computed as

$$s_j = \sum_i c_{ij}\hat{u}_{j|i} \tag{3}$$

$$v_j = squash(s_j) \tag{4}$$

This is where the actual routing now happens. With the scalar product we compute the "agreement" between the prediction of a capsule i from layer l and the just computed activation of a capsule j from layer $l+1$

$$a_{ij} = v_j \cdot \hat{u}_{j|i} \tag{5}$$

This agreement is then used to update the logits $b_{ij} = b_{ij} + a_{ij}$. This procedure is repeated as often as specified by the number of routing iterations, to eventually obtain the activation of the capsules of layer $l+1$.

Margin loss

To allow for multi-labeling, e.g. the existence of multiple digits in one image, the paper introduces the margin loss

$$L = \sum_k L_k = \sum_k T_k max(0, m^+ - ||v_k||)^2 + \lambda(1-T_k)max(0, ||v_k|| - m^-)^2 \tag{6}$$

where T_k is 1 if digit k is present and 0 if not. m^+ and m^- specify at which threshold we say the network is sure that a certain digit is or is not present in the image. The paper uses $m^+ = 0.9$ and $m^- = 0.1$. This means the first summand penalizes not detecting a present digit (false negative), while the second summand penalizes if the network says an actually not present digit might be present in the image (false positive). The λ regulates the trade-off between those two penalties and is chosen in the paper as $\lambda = 0.5$.

The overall architecture

The network consists of three layers (see Fig. 4).
The first layer is a normal convolutional layer with 256 9x9 kernels with a stride of 1 and a ReLU activation.

The second layer is the first capsule layer `PrimaryCapsules` with capsules of dimension 8. From a technical perspective it is just another convolutional layer from which we then construct the capsules. The convolution is done with 256 9x9 kernels with a stride of 2. We do not apply any activation function; instead we cut the output of this convolution into chunks of 8 along the channel axis to obtain 32 capsule channels each containing 36 capsules of dimension 8. Now we apply the squashing function to the capsules to get the output of the `PrimaryCapsules` layer.

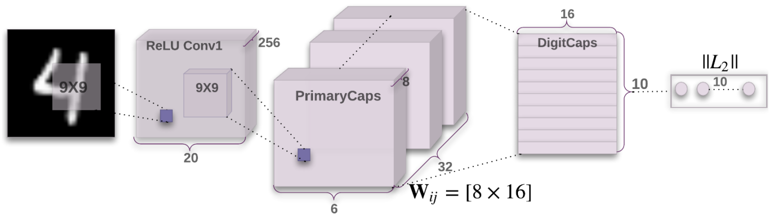


Figure 4: The CapsNet Architecture (Sabour et al., 2017)

The third layer is the read-out capsule layer `DigitCapsules` with 10 capsules of dimension 16. The activation is computed by the Routing-by-Agreement procedure described above, using 3 routing iterations. The loss is computed with the margin loss described above. On top of this network the paper introduces a reconstruction network that tries to reconstruct the images from the `DigitCapsules` and is supposed to act as a regularizer. For that, first all but the capsules corresponding to the correct digit are masked out (set to zero) and the `DigitCapsules` are flattened. This is followed by a fully connected layer with 1024 neurons and ReLU activation, a second fully connected layer with 512 neurons and ReLU activation, and lastly a fully connected read-out layer with $28 * 28 = 784$ neurons and a sigmoid activation. The loss of this reconstruction is calculated by sum of squared differences with the original images and is scaled down by 0.0005, so that it doesn't dominate the whole training procedure.

The optimizer is the `AdamOptimizer` with the TensorFlow default parameters and minimizes the sum of the two losses. All weights were initialized randomly from a truncated normal distribution with standard deviation 0.01. All biases were initialized with 1.

5 Implementation and Training Details

5.1 Implementation

Our implementation of the CapsNet architecture can be found on GitHub[1]. For easy reuse of the implementation, we modularized the architecture and implemented every type of layer (i.e.: Convolutional Layer, Convolution to Capsule Layer, Capsule Layer, Dense Layer) in the most general way possible. These modules can be found in the folder **layers**. For detailed explanation how to use these layers see the README.md and the documentation of the modules.

The data flow graph is defined in **capsnet_mnist.py**. In the same file we defined the training, validation and testing procedure.

5.2 Data

We used the original MNIST dataset (LeCun et al., 1998) with 60K training and 10K testing images (we used 600 of the testing images as validation images). We did not use any form of data augmentation.

5.3 Training

We trained the network on the grid of the Institute of Cognitive Science. We trained the network for 60 epochs. As the routing algorithm is very memory-expensive, we could not, despite running on a Titan X with a GPU working memory of 11.8 GB, use batch sizes above 600. This is why we used "only" the first 600 of the test images for validation. For training we used, as in the paper, a batch size of 128.

6 Results

6.1 CapsNet trained on MNIST

You can see the results, i.e. the plots from the training process in Figure 5. We reach a test error of 0.71%, which is not completely consistent with the test error reached in the original paper (0.25%). This might be due to different

[1] `https://github.com/lukeeffenberger/caps-net`

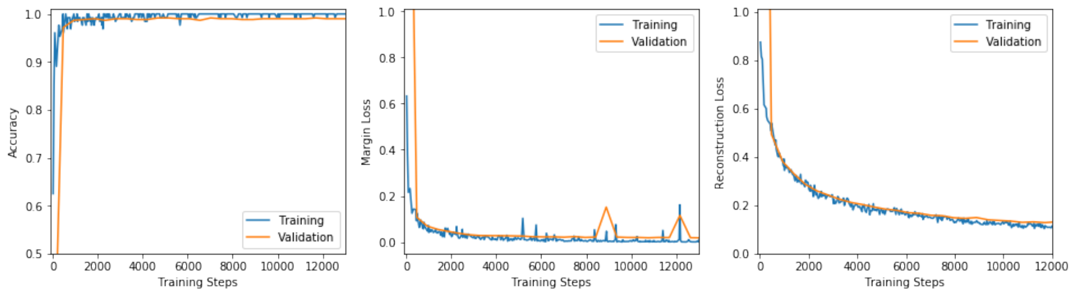


Figure 5: Training Process

weight initialization, more training time, and also probably the data augmentation (shifting the images by up to 2 pixels in one direction) used in the paper.

6.2 What the dimensions of the capsules represent

Using the reconstruction network we can look at what happens if we perturb one of the dimensions of a capsule a little bit, i.e. adding small values between -0.25 and 0.25 to the dimension. One can observer that some of the dimensions of some capsules encode very concrete information about the instantiation of a digit. For affirmation one can experiment with our notebook **dimension_representation.ipynb**. Figure 6.1 shows some examples: Dimension 6 of the capsule for digit 4 encodes the bending of the right line, dimension 16 of the capsule for digit 5 encodes the ratio of the upper part to the lower part, dimension 3 of the capsule for digit 6 encodes the stroke thickness, dimension 3 of the capsule for digit 9 encodes the size of the upper circle.

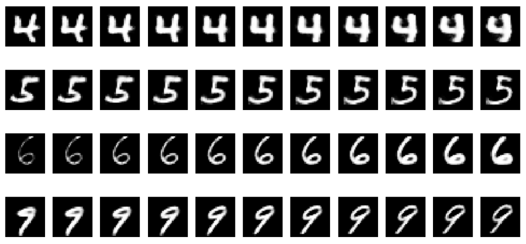


Figure 6: Dimension Permutation

7 Future Work

Improvements upon our code would be to add data augmentation and train for longer periods. Also, since capsule networks are pretty much brand new, there are a hundreds of research possibilities. There is a mass of datasets to apply them to. More interestingly, capsules are in general not restricted to the problem of computer vision. They might also be very helpful for speech recognition and other machine learning tasks.

8 Conclusion

We implemented capsule networks (Sabour et al., 2017) and trained it on the MNIST database. Our results closely mirrored the paper, in that we achieved exceptional accuracy results, and were able to perturb the dimensions of the image to see what the capsules were encoding.

We learned a lot about the theoretical assumptions behind CNNs, and how current neural network research is conducted by creating our own capsule net. Thanks for reading!

References

Braun, L. (2017). Implementing anns with tensorflow slides.

Hackernoon (2017). Boat example.

Hinton, G., Frosst, N., and Sabour, S. (2018). Matrix capsules with em routing.

Hinton, G. F. (1981). A parallel computation that assigns canonical object-based frames of reference. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 683–685, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

MathWorks (2018). Cnn architecture.

Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869.