1 Abstract

Capsule networks are one of the recent developments in the field of image recognition. For this project, we summarize [1] and try to reproduce the results of this paper on MNIST. We try the same architecture on FashionMNIST, a similar dataset, and try to analyze the effect of rotation on the input.

2 Problem with convolutional nets

Geoffrey Hinton outlined a few years ago the problems with convolutional nets [2], specifically

1. Subsampling or **pooling** loses the spatial relationships between components of the image.
2. CNNs cannot apply their understanding of one viewpoint to another viewpoint
   (a) we want the target to be viewpoint invariant, so we make the neural activity viewpoint invariant by using **pooling**
   (b) instead we should allow for changes in neural activity based on the viewpoint and encode viewpoint invariance in the weights

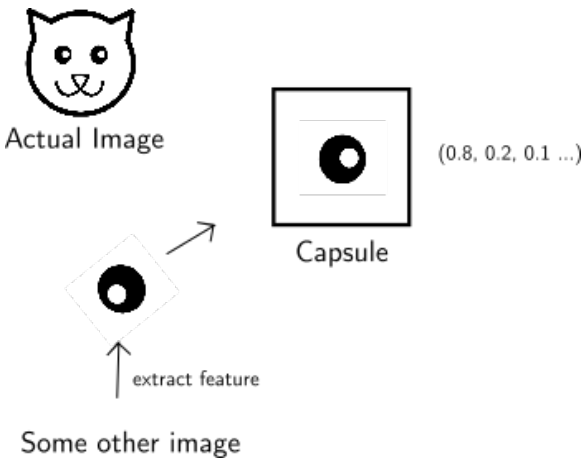

Figure 1: Capsule Idea

3 Capsule Theory

A **capsule** [3] is a group of neurons whose activity vector **A** represents the parameters of a specific component of the input image. The magnitude of this activity vector represents the probability of existence of the component in the image. The direction of this vector represents the parameters of the component like orientation, texture etc. Therefore, components that are the same but slightly modified, like rotated, enlarged etc. should have different **A**, but the same $|\mathbf{A}|$.

The paper we are analyzing is the first implementation of this theory, and it uses **dynamic routing** to construct the relationship between higher and lower level capsules.

4 Network Architecture

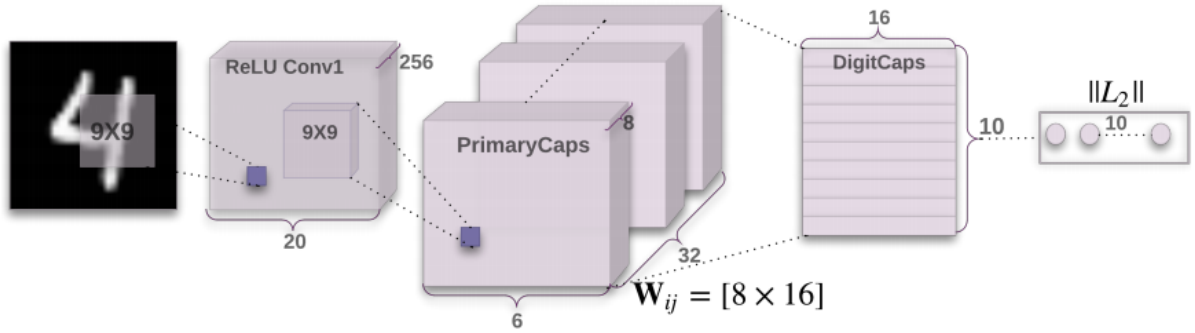

Figure 2: Architecture from the paper

The input is a $(28, 28)$ image. We apply convolution with 256 kernels, each of size $(9, 9)$ and stride of 1. This gives us 256 feature maps of size $(20, 20)$.

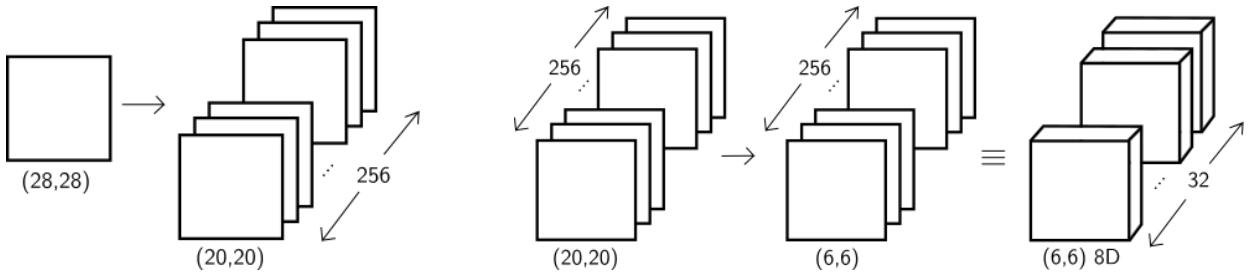

Figure 3: Input and PrimaryCaps Layers

Then, in **PrimaryCaps** we apply another convolution of the same type, with stride 2, which gives us 256 feature maps of size $(6, 6)$. After this, the data is reshaped to give 32 $(6, 6)$ 8 dimensional vectors which go to the next layer. As capsule theory requires, each vector should have a maximum magnitude of 1, since it represents a probability. So the paper uses a **squashing** function

$$f(x) = \frac{||x||^2}{1 + ||x||^2} \frac{x}{||x||}$$

The effect of this function on a vector is that it converts vectors of large magnitude to reach 1, since $1+||x||^2 \approx ||x||^2$, and then $f(x) \to \hat{x}$. For small magnitude vectors, the output stays small.

In the **DigitCaps** layer, the squashed output of previous layer u_i is converted from an 8D vector to a 16D vector by multiplying with a transformation matrix W_{ij} of dimensions $(16, 8)$. This gives us 1152 16D vectors, which we can use to create a parse tree like structure with 10 16D vectors on the higher layer, one for each class. This is where we do **dynamic routing**.

$b_{ij} \leftarrow 0$
for r iterations **do**
  for all c_{ij} **do**
    $c_{ij} \leftarrow$ softmax(b_{ij}), varying j
  end for
  for all outputs j **do**
    $s_j \leftarrow$ sum$(c_{ij} u_{j|i})$, varying i
    $v_j \leftarrow$ squash(s_j)
  end for
  for all i, j **do**
    $b_{ij} \leftarrow b_{ij} + u_{j|i} \cdot v_j$
  end for
end for

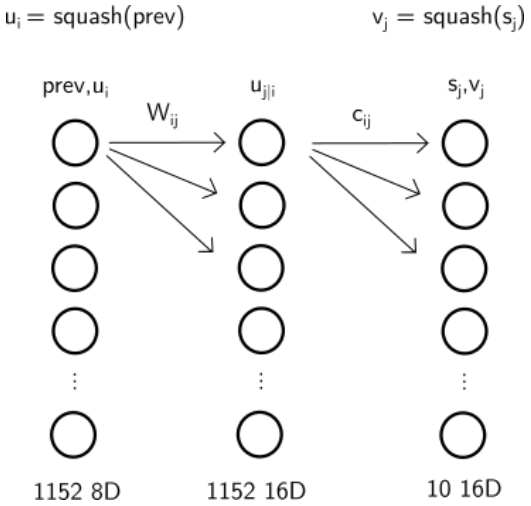

Figure 4: DigitCaps Layer

So, initially, $b_{ij} = 0$, and c_{ij} are just the softmaxes of b_{ij} over j, so for 10 output classes, they come out to be 0.1 for each link. Correspondingly, each class will receive something from the previous layers (1152 of them), and construct some vector v_j that represents the activity vector for that class. In the first iteration, these activity vectors are not perfect. But, we add to b_{ij} a similarity measure between the vector v_j of this layer and the vector $u_{j|i}$ of the previous layer. This similarity measure is a **dot product** between these two vectors. If the vectors align, we add a value close to 1, if they don't we add a value close to -1. Eventually, the c_{ij} will start having higher values when the previous layer vectors align with the class vectors.

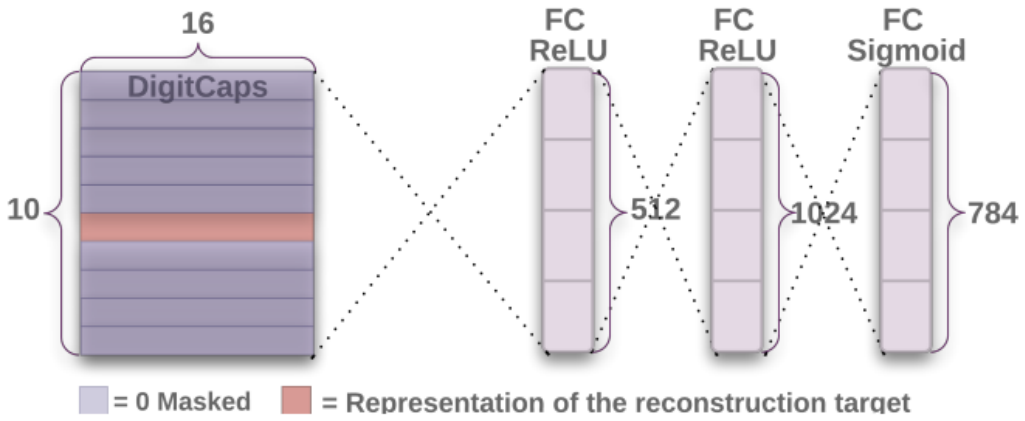

Figure 5: Decoder network

The **loss function** for training is defined over each digit k as

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

This is a modified **margin loss** function. For the paper, m^+ was chosen to be 0.9 and m^- was chosen to be 0.1. So, if a class is the correct label for the given data, $T_k = 1$. So, for that digit, the loss is squared error if $||v_k||$ is less than 0.9, otherwise it is 0. In simple terms, this means that we want the correct label to be predicted with greater than 0.9 probability. For incorrect classes, we want the prediction to go below 0.1, but we also downweight this loss by λ. Afterwards, we do a mean loss on this, and use this to backpropagate. The downweighting helps keep the correct class loss more important for overall loss.

The authors also use a **decoder network** at the end, and try to reconstruct the original image from the vector output of the correct class label. Then, a **pixel reconstruction loss** is added to the original image, and this acts like a regularizer and helps in training.

5 Experiments

The main idea behind capsule nets was to preserve properties like orientation, size, etc. However, the original paper just tested the trained model on affMNIST, which is MNIST with -20 to +20 degrees of rotation (and some minimal changes in size, stroke width etc). So, we wanted to see its performance at larger degrees of rotation. Specifically, this would verify whether or not this implementation was invariant to a crucial property, **rotation**.

We first tried to get the best accuracy as reported in the paper on **MNIST**. For this, we trained the model for 50 epochs, and each epoch took about 10 minutes. The original paper is uncertain on how many epochs they do, but this gives us a pretty good accuracy. Number of routing iterations is 3, and we use the decoder.

CapsNet (Sabour et al.)	99.75	>> 50 epochs
Our execution	99.7	50 epochs

Then, we reconstructed the image using the decoder on test data, and we found that they did resemble the input. First 5 rows are test data, next 5 rows are reconstructed images.

Figure 6: Reconstructed MNIST Images

Further, we take the trained model and modify the test images by simple rotation and see how good or bad the model performs on it. Here, we find that accuracy on rotated images was bad (180 degree rotation accuracy is better than 90 as few digits are invariant to 180 degree rotation, for example, 0, 1, 5, 8). And this affirms that the implementation does not learn rotation.

Accuracy on MNIST	99.7
Accuracy with 90 degrees	16.14
Accuracy with 180 degrees	45.96

Next, we try this same architecture on **FashionMNIST** [4], which is another dataset similar to MNIST. The accuracy graph looks like this

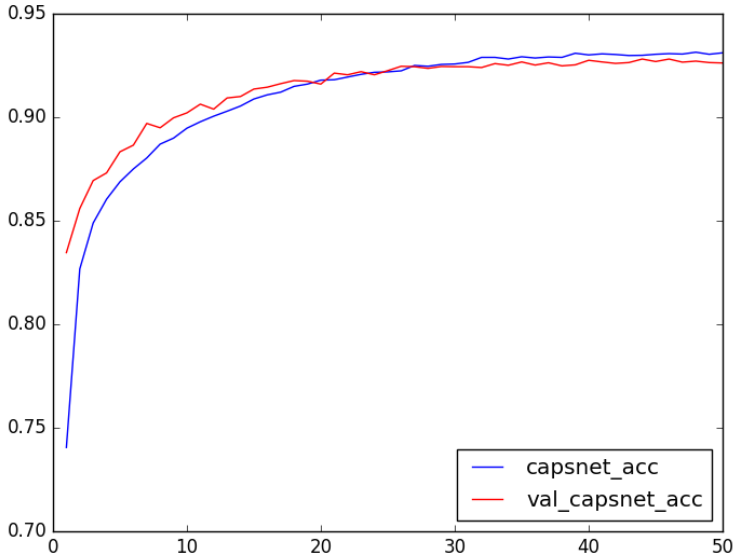

Figure 7: Training & Validation Accuracy on FashionMNIST

Another idea that we had was that **more complicated feature maps** would lead to better accuracy, because the higher layer capsules can represent more complicated structures in terms of lower layer components. To try this, we modify the architecture to have two convolutional layers of kernel size $(7, 7)$ each, before the PrimaryCaps

layer. In the PrimaryCaps layer, the convolutional layer now has a filter size of $(5, 5)$ with stride 2. Rest of the architecture stays the same. We tabulate our results below. For a comparable CNN, we wanted a network that had roughly the same number of parameters. So we chose as baseline a convolutional net with 3 conv layers $(256, 256, 128)$ (5×5 kernel, stride 1) and 2 FC layers of size $(328, 192)$.

Comparable CNN (30 epochs)	92.98
Accuracy on FashionMNIST (1 conv)	92.81
Accuracy with 90 degrees	8.21
Accuracy with 180 degrees	26.25
Accuracy on FashionMNIST (2 conv)	93.95

As tabulated, we got a better accuracy with more complicated feature maps, i.e. 93.95% instead of 92.81%. Here is a visualization of some test images.

Figure 8: Reconstructed FashionMNIST Images

Next, we passed rotated images on the trained model to see the visualization of the reconstructed images. We also tried to see what capsules were firing corresponding to some test image. In the second figure below, the first image is the test image, and the rest show the visualization on those classes, had those been correct.

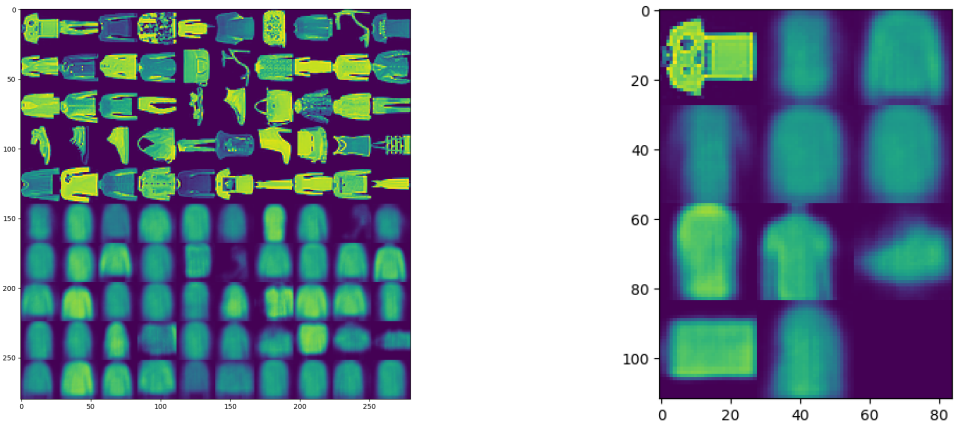

Figure 9: Reconstructed Images on rotated examples for correct class & on one rotated image for all classes

As we can see, the decoder outputs are not rotated and also they don't belong to the correct class. This indicates that the either the activity vector is not very good at capturing rotational information and/or the decoder is not trained well enough to reconstruct the correct images.

We further tried to test whether the similarity measure, i.e. the dot product was decreasing or not when we rotate the test image by a few degrees. We did this test on 10 images, going from 0 to 90 (exclusive), with a step of 10. Three of those are given below. We did not do this test during routing, but instead do it on the trained model, while testing.

0°	10°	20°	30°	40°	50°	60°	70°	80°
1.0	0.99723	0.830556	0.481795	-0.00969078	-0.500877	-0.666482	-0.760729	-0.675675
1.0	0.848538	-0.405796	-0.45842	-0.463034	-0.0212348	0.624204	0.901812	0.687763
1.0	0.850194	0.0545592	-0.162416	-0.319865	0.0625221	-0.0175566	-0.349497	0.108656

If the capsules implemented in the paper did represent the capsules in the original theory, these values should have decreased successively as we moved from 0 to 90 degrees. This is because the dot product happens between the activity vector of the trained model that represents a non rotated class, and the activity vector on the test image. But, this does not happen.

6 Conclusions & Critique

Capsule nets are an interesting idea that can change how we do image recognition. The performance of this network on MNIST is promising, but it does not capture the most essential attributes of components, like rotation. The training is comparatively slower than CNNs, and as demonstrated for FashionMNIST, there are no significant gains in accuracy. However, it is potentially a good topic that can see better implementations in the future. The code is available on Github [5].

References

[1] Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in Neural Information Processing Systems. 2017.

[2] Geoffrey Hinton talk "What is wrong with convolutional neural nets ?", https://www.youtube.com/watch?v=rTawFwUvnLE

[3] http://cseweb.ucsd.edu/ gary/cs200/s12/Hinton.pdf

[4] Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747 (2017).

[5] https://github.com/agaurav77/capsulenets