Investigating the Performance of a Capsule Network in Digit Classification Task

Mah Yuen Yee 30213673

1. Introduction

Convolutional neural networks (CNNs), a class of artificial neural networks that are commonly featured with 3D volumes of neurons, local connectivity, shared weights, and pooling, has achieved tremendous success in different visual imagery analysis tasks such as image and video recognition, image classification, image segmentation, etc. Despite its success, CNN has several fundamental drawbacks. CNNs are translational equivariant due to weights sharing, but they cannot handle the effects of changing viewpoints such as rotations and scaling [1]. As CNNs have no built-in understanding of 3D space like the human brain does, the current common method of gathering images displaying objects in various positions to feed the model is not efficient. Moreover, CNNs do not explicitly encode the precise spatial relationships between features which can lead to the Picasso problem [2].

Data augmentation including mixing images, feature space augmentation and generative adversarial networks (GANs) have been widely used in training CNN to reduce overfitting and improve generalization capability [3]. Transfer learning, which involves fine-tuning CNN models trained with large volumes of data for training on a small request dataset, has been employed to overcome the undersized dataset problem [1]. Lee et al. demonstrated the convolutional deep belief network (CDBN), a scalable generative model for learning hierarchical representations from un-labeled images, to bypass the need of a large of labeled data for model training [4]. Hinton et al. and Sabour et al. presented capsule network with dynamic routing, an approach closer to replicating the human vision, to resolve the fundamental limitations of CNNs [5], [6]. This paper investigates the performance of a Capsule Network (CapsNet) in digit classification task.

There are 2 main novelties in the network proposed – capsules and dynamic routing. Capsules are designed to achieve viewpoint invariance in the activities of neurons without losing any information yet encoding relative spatial relationship between features. In short, a capsule is effectively a neuron that recognizes an implicitly defined visual entity over its receptive field and outputs both the probability of the presence of the entity and a set of "instantiation parameters" that may include the precise pose (translation and rotation), lighting and deformation of the visual entity relative to the implicitly defined canonical version of that entity.

On the other hand, to recognize an object, human brain deconstructs the hierarchical representation of anything from visual information received and matches it with learned patterns and relationships in the brain. To mimic this process of so-called inverse graphics, dynamic routing between capsules was proposed. To replace max-pooling and to employ the idea of parse trees, a "prediction vector" that compares the similarity between the capsule output to each capsule input is used for the routing coefficient to be updated correspondingly. The routing coefficient determines the weights of each capsule input onto the capsule output, i.e., effectively determines the parent capsule of any capsule from the layer below.

2. Methodology

2.1 Encoder

CapsNet has 2 major differences from CNN. It replaces the scalar-output feature detectors at higher levels of CNNs with vector-output capsules and max-pooling with routing-by-agreement. A basic CapsNet architecture is shown in Fig. 1. CapsNet has its first layer as a convolutional layer (conv layer) with 256, 9x9 kernels and ReLU activation. The second layer is a convolutional layer with 256, 9x9 kernels with stride of 1. The output feature maps of size 256x6x6 are then reshaped into 32 blocks, each containing 6x6 8D vectors which are further reshaped into 1152 8D vectors. The vectors are seen as capsules. Therefore, the output here is effectively 1152 8D capsules. This layer of capsules is called PrimaryCaps where the capsules are called primary capsules. The layer above is known as DigitCaps which contains 10 16D capsules. The capsules here are called digit capsules. The PrimaryCaps layer is connected to the DigitCaps layer by dynamic routing algorithm.

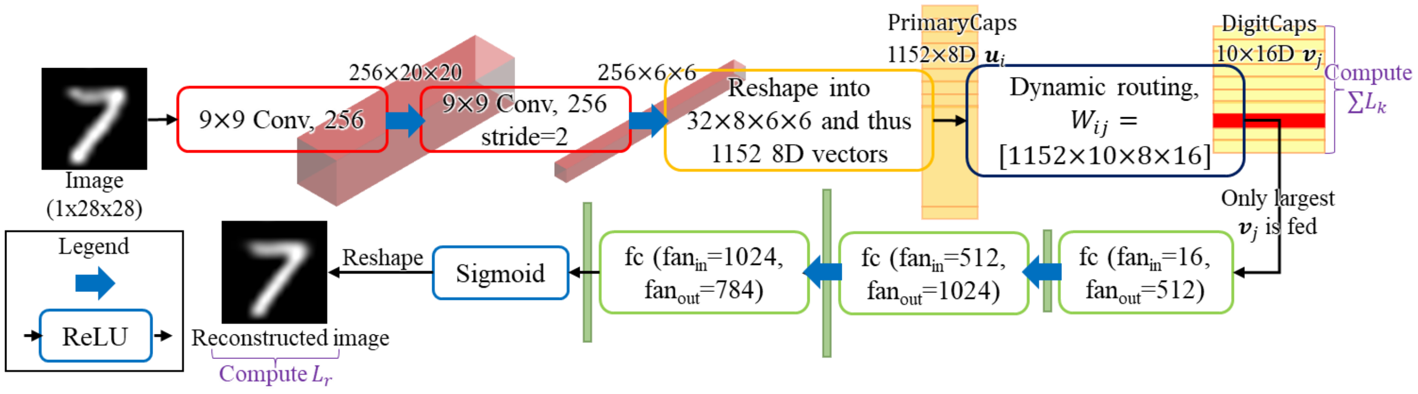

Image (1x28x28)
9×9 Conv, 256
256×20×20
9×9 Conv, 256 stride=2
256×6×6
Reshape into 32×8×6×6 and thus 1152 8D vectors
PrimaryCaps 1152×8D u_i
Dynamic routing, W_{ij} = [1152×10×8×16]
DigitCaps 10×16D v_j
Compute $\sum L_k$
Only largest v_j is fed
fc (fan$_{in}$=16, fan$_{out}$=512)
fc (fan$_{in}$=512, fan$_{out}$=1024)
fc (fan$_{in}$=1024, fan$_{out}$=784)
Sigmoid
Reshape
Reconstructed image
Compute L_r
Legend
ReLU

Fig. 1. Basic architecture of CapsNet, as well as the CapsNet28 architecture, consists of an encoder and a decoder. The encoder has 2 conv layers and 2 capsule layers (PrimaryCaps and DigitCaps). All the output $\mathbf{v}_j$ of DigitCaps layer is used to compute $\sum L_k$. Only $\boldsymbol{v}_j$ with the largest magnitude is fed to the decoder which consists of 3 FC layers, each followed by ReLU or Sigmoid activation. The output the last FC layer is reshaped to be a 28×28 reconstructed image. The reconstructed image is used to compute L_r.

All primary capsules $\boldsymbol{u}_i$, $i \in (1, 32 \times 6 \times 6)$ try to predict the output of digit capsules, i.e., the output of every capsule $j \in (1, 10)$ in the next layer. The prediction $\hat{\boldsymbol{u}}_{j|i}$ is computed by

$$\hat{\boldsymbol{u}}_{j|i} = \boldsymbol{W}_{ij}\boldsymbol{u}_i$$

where $\boldsymbol{W}_{ij}$ is a weight matrix that encodes relationship between lower level features in primary capsules and higher level features in digit capsules. Since a digit capsule encapsulates parameters for a digit class, $\boldsymbol{W}_{ij}$ effectively learns part-whole relationship. $\boldsymbol{W}_{ij}$ learns every epoch using mini-batch gradient descent as normal as how weights learn in CNNs and multi-level perceptrons (MLPs).

However, all part-whole relationships learnt are not necessarily valid for every images. For example, when there is an upright 4 on the image, primary capsule A capturing and featuring the / shape can suggest an upright 7 while primary capsule B capturing and featuring a | shape can suggest a left-rotated 7. The part-whole relationships learnt are generally correct, but they are not agreeable between primary capsule A and B for this image as they suggest the 7 appears in different orientations. The agreements and disagreements between the primary capsules determine the coupling or routing coefficients c_{ij} through iterative dynamic routing algorithm. c_{ij} is softmax of initial logits b_{ij} which are the log prior probabilities that capsule i should be coupled to capsule j for an image input.

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

Any image went through the first 2 conv layers and would reach the dynamic routing process which connects PrimaryCaps layer to DigitCaps layer. $\hat{\boldsymbol{u}}_{j|i}$ is computed as the beginning of dynamic routing algorithm. After that, b_{ij} are initiated as 0s. c_{ij} are then computed to have 0.5s. The total input to a capsule $\boldsymbol{s}_j$ is then computed as a weighted sum over all $\hat{\boldsymbol{u}}_{j|i}$ with factors of c_{ij}

$$\boldsymbol{s}_j = \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i} .$$

Since Sabour et al. [6] wanted the length of the output vector of digit capsules to represent the probability that the entity represented by the capsule is present in the current input, a non-linear "squashing" function is used to induce non-linearity and magnitude of 1. The vector output $\boldsymbol{v}_j$ of digit capsule j is computed by

$$\boldsymbol{v}_j = \frac{\left|\left|\boldsymbol{s}_j\right|\right|^2}{1 + \left|\left|\boldsymbol{s}_j\right|\right|^2} \frac{\boldsymbol{s}_j}{\left|\left|\boldsymbol{s}_j\right|\right|} .$$

$\boldsymbol{v}_j$ is then used, along with $\hat{\boldsymbol{u}}_{j|i}$, to update b_{ij} by

$$b_{ij} := b_{ij} + \hat{\boldsymbol{u}}_{j|i} \cdot \boldsymbol{v}_j$$

c_{ij} is then recalculated accordingly. The same process is then repeated for 3 iterations. The routing algorithm is detailed in Procedure 1 in Appendix.

Length or magnitude of the digit capsule or vector is used to represent the chance that its digit exists in the image. To allow for multiple digits in an image, a separate margin loss L_k is used for each digit capsule k:

$$L_k = T_k \max\left(0, m^+ - \left|\left|\boldsymbol{v}_k\right|\right|\right)^2 + \lambda\,(1 - T_k)\max\left(0, \left|\left|\boldsymbol{v}_k\right|\right| - m^-\right)^2$$

where $T_k = 1$ iff a digit of class k is present, $m^+ = 0.9$, $m^- = 0.1$ and $\lambda = 0.5$. The λ down-weighting of the loss for absent digit classes stops the initial learning from shrinking the lengths of the activity vectors $\boldsymbol{v}_k$ of

TABLE I
Training dataset details.

	CapsNet28	CapsNet56
Train set size	60k	6k
Test set size	10k	10k
Trained on	28×28 MNIST	56×56 MNIST
Train set transform	Random crop to size 28 with padding = 2	Random crop to size 56 with padding = 14

all the digit capsules. The total margin loss is the sum of the losses of all digit capsules.

2.2 Decoder

Only $\boldsymbol{v}_j$ with the largest magnitude is fed into the decoder for reconstruction and compared against true label. The reconstruction loss L_r to be minimized is Mean Squared Error (MSE) loss. L_r is used to encourage the digit capsules to correctly encode the instantiation parameters of the input digit in the input image. The decoder has an input of size 1×16 and output of size $1 \times 28 \times 28$. Its architecture is shown and explained in Fig. 1. Therefore, the final loss for training includes both L_k and L_r. L_r is scaled down by $0.0005 \times 28^2 = 0.392$ so that it does not dominate L_k during training. Total loss is

$$L_T = 0.392 L_r + \sum_k L_k$$

3. Experiments and Results

Two CapsNet are trained: CapsNet28 and CapsNet56. The training details are shown in TABLE I. 56×56 MNIST images are created by upscaling original 28×28 MNIST images with zero padding as shown in Fig. 2. CapsNet28 architecture is shown in Fig. 1. CapsNet56 is added 1 more conv layer as the 2nd layer with stride of 2 and has only 128 channels in its 1st conv layer. The changes are to accommodate the increase in size of input images.

The baseline against CapsNet28 is a standard CNN (CNN28) with 3 conv layers of 256, 256, 128 channels. Each has 5x5 kernels and stride of 1. The conv layers are followed by 2 FC layers of size 328, 192. The last FC layer is connected with dropout to the 10 class softmax layer with cross entropy loss. The changes made to CapsNet28 to create CapsNet56 are applied to CNN28 to create baseline against CapsNet56 (CNN56). Hyperparameters for model training are shown in TABLE III. TABLE II shows the model results on different test sets.

TABLE II
Results on different test set.

	28^2 MNIST	56^2 MNIST	Affine transformed 56^2 MNIST	2-digit 56^2 MNIST
CNN28	99.28%	-	-	-
CapsNet28	99.62%	-	-	-
CNN56	-	97.64%	83.24%	54.66%
CapsNet56	-	98.85%	89.45%	97.05%

CapsNet28 achieved a low test error of 0.38% which is on-par with the state-of-the-art deeper CNNs [7]. CapsNet56 is trained mainly to test its ability to encode instantiation parameters of digits and capability to recognize a combination of two digits in an image as demonstrated in Fig. 4. although it was only trained with 1-digit image. CapsNet56 achieved 97.05% accuracy in predicting the 2-digit 56×56 MNIST while the baseline CNN achieves 54.66%. This experiment shows that the capsule layers managed to encode instantiation parameters for each digit class while the dynamic routing can efficiently address the appearance of more than 1 digit on the test image.

CapsNet56 is also tested on affine transformed 56×56 MNIST images shown in Fig. 3. to test its ability to learn more robust representation for each class from the natural variance in skew, rotation, style, etc in handwritten digits. CapsNet56 is never trained with affine transformations other than translation and any natural transformation appeared in standard MNIST. CapsNet56 achieved 89.45% accuracy on the affine transformed test set while the traditional baseline CNN56 with more parameters achieved 83.24% on the affine transformed test set. CapsNet56 is proved to be more robust to unseen small affine transformations.

4. Conclusion

From the 2 experiments conducted, CapsNet is proved to be able to make predictions with high accuracy when there is more than 1 digit on the test image although it is only trained with 1-digit image. In addition, CapsNet is more robust to unseen small affine transformations. All these are achieved by capsule layers and dynamic routing algorithm. The results suggest that CapsNet can potentially reduce the training data needed to perform decently in image classification tasks. CapsNet can be particularly useful in any field where only small dataset is available such as manufacturing defect detection.

5. References

[1] L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” Journal of Big Data, vol. 8, no. 53, Mar. 2021.

[2] J. D. Kelleher, “The Future of Deep Learning,” in Deep Learning. Cambridge, U.S: MIT Press, 2019.

[3] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” Journal of Big Data, vol. 6, no. 60, 2019.

[4] H. Lee, R. Grosse, R. Ranganath and A. Y. Ng, “Unsupervised learning of hierarchical representations with convolutional deep belief networks,” Commun. ACM, vol. 54, no. 10, Oct. 2011.

[5] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming Auto-Encoders,” in International conference on artificial neural network, 2011, pp. 44-51, doi: 10.1007/978-3-642-21735-7_6.

[6] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in Advances in neural information processing systems 30, 2017.

[7] “Image Classification on MNIST.” Paperswithcode.com. https://paperswithcode.com/sota/image-classification-on-mnist (accessed May 22, 2022)

Appendix

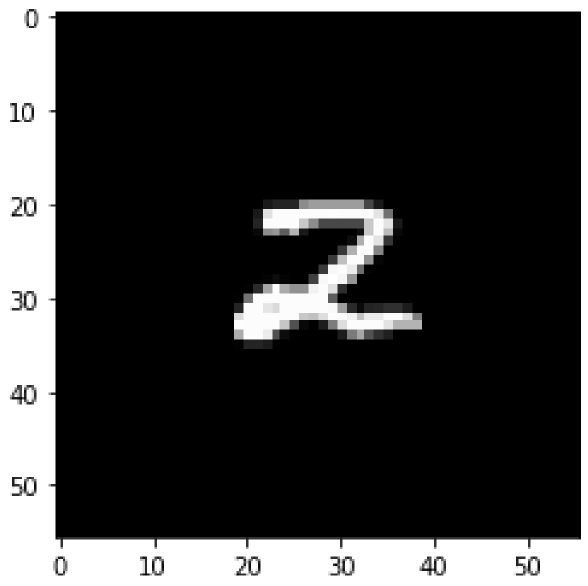

Fig. 2. An example image of 56x56 MNIST.

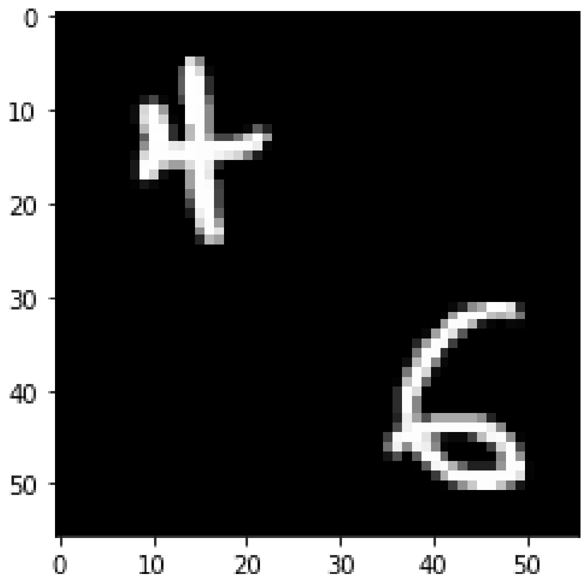

Fig. 4. An example image of 2-digit 56x56 MNIST test set.

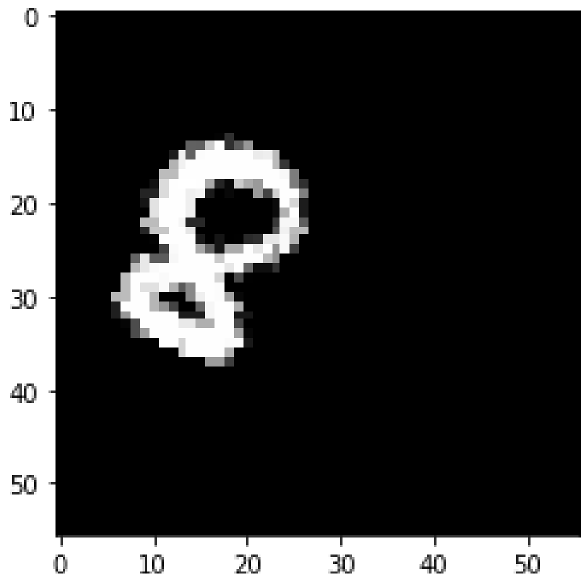

Fig. 3. An example image of affine transformed 56x56 MNIST test set.

Procedure 1 Routing algorithm

```
1: procedure ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
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 $l$: $\boldsymbol{c}_i \leftarrow softmax(\boldsymbol{b}_i)$
5:       for all capsule $j$ in layer $(l+1)$: $\boldsymbol{s}_j \leftarrow \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i}$
6:       for all capsule $j$ in layer $(l+1)$: $\boldsymbol{v}_j \leftarrow squash(\boldsymbol{s}_j)$
7:       for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\boldsymbol{u}}_{j|i} \cdot \boldsymbol{v}_j$
     return $\boldsymbol{v}_j$
```

TABLE III
Hyperparameters and number of parameters for the models.

	CNN28	CapsNet28	CNN56	CapsNet56
Epoch	5	50	28	70
Batch size	100	100	100	100
Learning rate	0.001	0.001	0.0005	0.0008
Learning rate decay	N/A	0.9	N/A	0.96
Optimizer	Adam	Adam	Adam	Adam
Number of parameters	13 277 970	8 215 568	14 916 626	13 270 336