Exploration of Capsule Networks

Nigel Schuster
ns3158

Michael Chess
msc2209

Tianyu Yang
ty2345

December 17, 2017

Abstract

Capsule Networks provide a novel approach for understanding data with especially promising results in the field of computer vision. Capsule Networks discover successive transformations of an image and thereby allow deeper understand than traditional ConvNet architectures. For MNIST this technique was successfully applied to produce state of the art results. CapsNets were further found to be particularly adept at deciphering overlapping digits. We use CapsNets first to verify state of the art results and then to explore their ability to classify novel testing sets using MNIST training. Subsequently we apply CapsNets to CIFAR-10 and to building a CAPTCHA recognizer for reCAPTCHAs, leveraging the CapsNets ability to tolerate lateral shifts and zooms in character datasets very well.

1 Introduction

Modern advances in tensor-processing coupled with an explosion of interest in neural networks has lead to huge advances in image recognition. Simple convolutional neural networks can now achieve error rates of less than 10% on data sets on which traditional methods struggle to do better than 50%, and even new approaches such as DenseNets do even better. Capsule networks approach this field in a particularly busy position. MNIST character data has for years been a standard dataset for evaluating individual character recognition models. State of the art today approaches 0.20% error. But these models still struggle with recognition in settings that are often extremely important in the real world. Particularly in instances of character rotation, overlap, and transformation. In *Dynamic Routing Between Capsules* Sabour et al. demonstrate that simple capsule networks perform significantly above previous state-of-the-art efforts in recognizing overlapping characters[1].

1.1 What are capsule networks?

Capsule Networks are a form of neural network constructed of clusters of neurons, called capsules, that communicate in a hierarchy via a routing procedure to make decisions. Each capsule seeks to define some feature of the input data (for writing: orientation, stroke width, curvature etc.). Specifically capsules can be thought of as predicting whether a feature is present. Capsules in a higher layer are activated based on the predictions of the lower layers until the classification layer is reached. The predictions of each capsule are then outputted as a feature vector. To identify correct predictions a "routing by agreement" technique is used. This is, multiple capsules must agree on the properties for it to be propagated to the parent layer[4].

1.2 Why use capsule networks?

A capsule is a small computational unit representing an individual feature of the underlying data. Sabour et al claim that this structure presents a better model of human processing of complex tasks. Capsules as opposed to max- or avg-pooling layers effectively evaluate information successive high-level features without losing position data (e.g. max-pooling only preserves 1 out of n data points and modifies the spatial content of the image). By contrast, the transformation matrices between successive layers are able to encode spatial relationships between lower level features. This means that capsules are able to use local information to extract viewpoint invariant knowledge about the subject which is automatically generalized to novel viewpoints via inter-layer transformation matrices.

Further advantages are discussed in Appendix 11.1.2.

2 Related Work

Earlier this semester Sara Sabour, Nicholas Frosst, and Geoffrey Hinton published the paper *Dynamic Routing Between Capsules* in which they proposed a dynamic routing algorithm for routing information between layers of a network of capsules[1]. In their construction the output of each capsule is a vector representing the probability that the entity captured by that capsule is present in the current input. This is achieved using a squashing function to ensure that the output vectors are in the range [0,1]. For an output vector v_j with an activation vector s_j this equation is:

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

This function scales the normalized s_j vector by the factor $\|s_j\|^2/1+\|s_j\|^2$. This factor is very small for vectors with small initial magnitude and approaches one as the length of the vector approaches infinity. The s_j output vector is computed for each as the weighted sum of the prediction vectors from the layer below. These weights are called coupling coefficients because they determine the influence of a particular capsule i has on the capsule j in the layer above. The prediction vectors themselves are computed by multiplying each output vector by a weight matrix specific to each capsule pair. The coupling coefficients are computed using a softmax with logits of the log prior probabilities that a capsule i should be coupled to a capsule j. The logits are learned in the dynamic routing algorithm itself. An important note is that convolutional capsules are alternately represented as having a grid of output vectors. For each output in the grid of each capsule there is a separate weight matrix for it and each capsule in the above layer.

Further related work is discussed in Appendix 11.1.

2.1 Routing Procedure

For a layer l and layer (l+1) over r iterations the routing procedure computes the above values and uses them to recompute the logits for the softmax defining the coupling coefficients.

2.2 Loss in a CapsNet

Loss in a multi-headed capsule network is a margin loss. That is the loss of absent digit classes is included but downweighted in the loss of a given digit class. For each possible head k the loss is:

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

In this loss setting $T_k = 1$ if and only if a the class that head predicts is present. What this means is that for the correct head the loss occurs if v_k (the prediction vector) is shorter than 0.9 or m^+ otherwise we consider the prediction correct and there is no loss. For incorrect heads loss occurs only if the lengthe of the final capsule's vector is greater than 0.1. Vectors for incorrect heads that are less than this are then taken to be correct and have no loss.

3 CapsNet Experiments

Our first goal was to understand for ourselves how CapsNets work and how each component affects its performance. We divided our modifications into data transformations, where we applied transformations to testing data only to establish the flexibility of the network, and architecture modifications, where we change structural components of the network described by Sabour et al.

3.1 Data Transformations

We started by exploring the ability of a Capsule Network, trained on standard MNIST data, to predict other data:

3.1.1 Rotation

One of the most common data transformation is rotation. We broke our experiments into 6 rotation levels. For a given level x the images were rotated by a random degree number in the range [0, x). In Figure 13 we see the CapsNet drops in performance as expected as the maximum rotation angle goes up. The performance regression is extremely similar to that seen in a reference CNN in Figure 14.

3.1.2 Shear

We examined how the networks handle shear, where images are effectively skewed alone one axis (eg. a special case of an affine transformation with values only in the skew positions). Both perform similarly with small maximum shear values. On larger augmentations we can see that the CapsNet starts to perform better (see Figure 15).

3.1.3 Horizontal Shift

We shifted the image horizontally, so that the digit is not located in the center of the image any more. Figure 16 we see the performance of the CapsNet vs CNN. At first we were surprised by the magnitude of the difference. After some code analysis we discovered that the reference implementation itself was augmenting the data by shifting it. To level the comparison, we removed the augmentation. After this, the majority of the performance advantage persisted suggesting that the CapsNet architecture is better able to generalize across shifts than a CNN is. This is an important effect as it allows you to locate the presence of characters in images where the character itself is not centered, a situation common to many real world character recognition tasks.

3.1.4 Zoom

Finally we augmented the data through zooming in/out in order to have different sizes of the number. Again, the CapsNet shows a slight advantage, but we also notice less consistency in accuracy as the epochs progress.

3.2 Architecture Modifications

Besides simple data augmentation, we decided to attempt algorithmic modifications:

3.2.1 Squashing Function

As outlined in section 2 and shown in equation 1, the squashing functions is core in weighing outputs of different capsules. We want to explore the space of functions that compute those outputs. Thus we chose a squashing function that puts more weight on big vectors than the provided one: $v_j = \left(\frac{\|s_j\|^2}{1+\|s_j\|^2}\right)^2 \frac{s_j}{\|s_j\|}$. Figure 18 shows that this modification has close to no impact.

3.2.2 Loss function

The loss function in each end capsule attempts to model how correct the network's predictions were. CapsNet uses a loss function that built on Hinge loss, but the loss function allows mispredictions within a certain margin. That is the hinge occurs at 0.9 for correct capsules and at 0.1 for incorrect capsules so values outside of these ranges incur no loss. This means that the network has no impetus to improve on vectors of length greater than 0.9 for and less than 0.1. We removed this margin, using instead 1.0 and 0.0, since it appears unnecessary. However Figure 19 shows better results with the original loss function.

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

$$m^+ = 1.0 \;\; m^- = 0.0$$

3.2.3 Routing iterations

The reach an agreement on the routing pattern between layers CapsNet uses an iteration based routing by agreement algorithm as described above. We increased the number of iterations to see performance differences (Figure 20). While the new routing slightly improved performance, it does not merit the additional computation required.

4 CIFAR

Research should not only surface good results, but also show problems with current approaches. Dynamic Routing between Capsules suggested that CapsNet does not work well with CIFAR. We continued to explore this problem. Our analysis was going to be based on the reconstruction of images similar to Fashion-MNIST (Figure 2). However, image reconstruction tries to reconstruct each image as a whole - including any background. Therefore the problem is that the network is putting equal

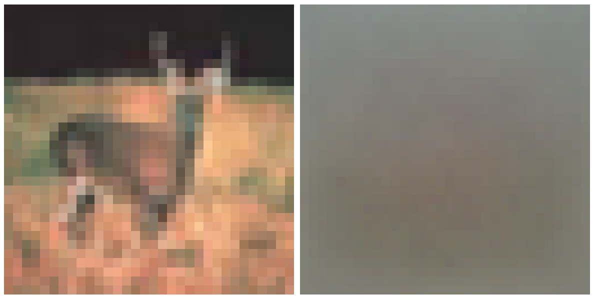

Figure 1: Photo of a deer and its reconstruction

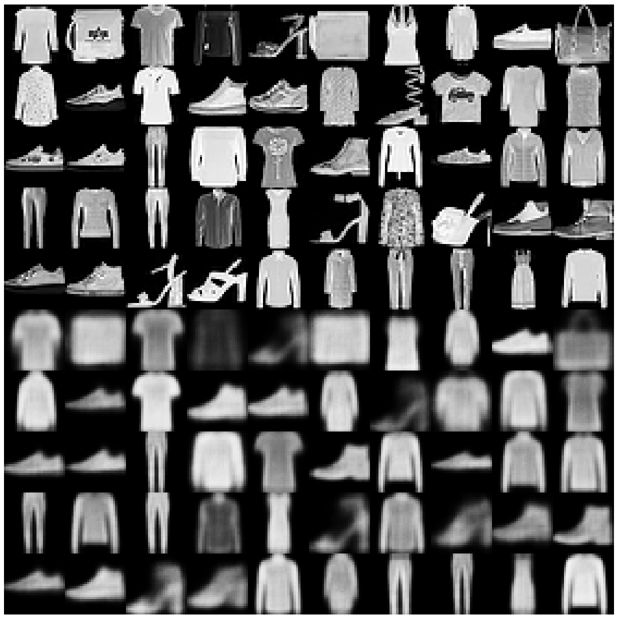

Figure 2: Reconstructed Fashion MNIST

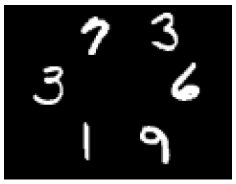

Figure 3: Three generated Multi-MNIST images with random spacing

weight on each pixel, as opposed to reconstructing the primary features of the image. Due to this issue, most images are solely showing the primary color component of the image (Figure 1).

5 Multi-Digit MNIST

We also experimented with an alternate way of generating captcha-like images. We generated long images with randomly placed MNIST characters at different locations. This makes a dataset that is analogous to reCaptcha but is slightly easier to classify.

5.1 Multi-MNIST Generation

Our multi-MNIST generation code selects a pre-defined number of random characters from the training or testing MNIST sets. These characters are then placed at random locations into an elongated image. We tested primarily on non-overlapping images to test the resilience of our system to different inter-character distances.

5.2 Multi-Class Prediction

Initially we ran similar experiments to the regular CAPTCHA section. We wrote infrastucture to train on wide images with a single MNIST character. At first we ran with a single mnist character in the center of a wide image. This turned out to have very low prediction accuracy since the wide image is 112 pixels and the generated Multi-MNIST images have characters all over. Despite the CapsNets benefit over a CNN on translation it is not able to overcome multiple times offsets in either direction. Because of this we moved to training on a single MNIST character at random locations in the image. This allowed for significant improvements in prediction with rates upwards of 70%.

5.3 Scanning Multi-MNIST Reading

For multi-MNIST we also developed a different reading method. Instead of relying on the ability of CapsNets to resist shifts and output multiple classes at a time we built a system that scans the wider multi-MNIST image and predicts regular-MNIST sized sections of it. This method gives us predictions at a sequence of locations in the image. We then find the characters represented by the longest two vectors. These represent the two locations in the array that had the highest response. These locations have two indices. One is on the position axis, which allows us to tell which character came first. The other axis is the character axis which tells us which character we found. The combination of these two factors allow us to generate a prediction of the CAPTCHAs that is sensitive to position.

5.4 Results

Preliminarily this gives reasonably good results on two character multi-MNIST CAPTCHAs. We are able to predict with a character-position accuracy of around 0.91 and a CAPTCHA accuracy of around 0.83. Our system is not yet able to predict CAPTCHAs with two characters of the same class. For MNIST data this means that we expect a max potential character-position accuracy of 0.95 and a max CAPTCHA accuracy of 0.9 due to approximately 0.1 of the CAPTCHAs having duplicate characters. This can be seen in figure 4.

6 CAPTCHAs

D. George et al. recently published a paper on Science [5] describing a model they developed that can efficiently break Captcha. One of the advantages their model has is accurately predicting Captchas with variate character distances even without training with them[Insert graph?]. Because we observed Capsnet's resistance against shifted characters during our experiment, we want to explore if Capsnet also share this advantage.

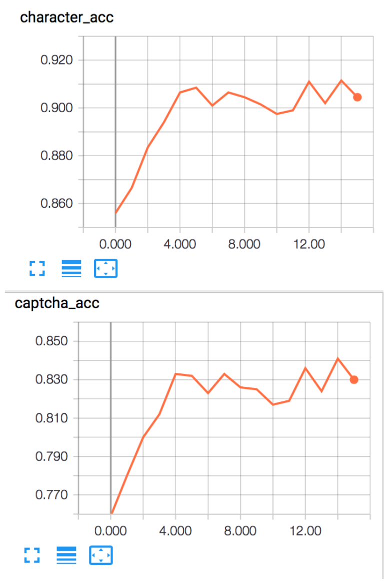

Figure 4: Character Accuracy and CAPTCHA accuracy for two character Multi-MNIST

6.1 CAPTCHA Generation

The image set is generated from Google reCaptcha image set downloaded from Vicarious, the same image set used by D. George et al. Only the capitalized characters was selected for training. This only gave us 54 images, 2 for each character except W which has 4. The images were later cropped and down-sampled into 28*28*1 images, the same dimension MNIST has. In order to sufficiently train our model, we generated the training set using heavy shear and rotation augmentations [figures?] A generator was implemented so continuously generate training images. The image is then inserted into a wide format image so there is enough space to insert two or more characters for testing. For testing, we generated a test image set that contains two characters. The characters were augmented using same method as the training set, but later been put down on the same image. Samples with two same characters were rejected to simplify the model. The distance between character is defined as the number of pixels between the cente r of gravity of each character.

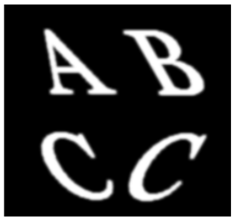

Figure 5: Google reCAPTCHA example

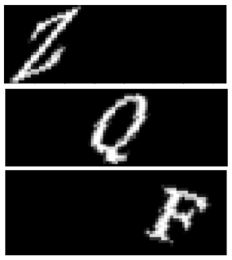

Figure 6: Augmented reCAPTCHA training set

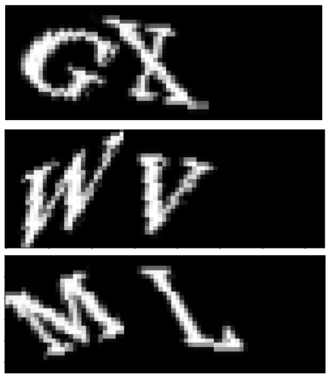

Figure 7: reCAPTCHA testing image with 20px, 25px and 30px as distance

6.2 CAPTCHAs in a CapsNet

Designing a CapsNet for CAPTCHAs has proven to be a more challenging than expected. Our hope was to scale our model to more classes and without requiring additional layers. We started by limiting our approach to two letters, so that complex mechanisms, such as an LSTM. First, we decided to have a label for each pair of letters (thus 676 classes). Since this lead to memory errors, we instead decided to have one class per letter. This allowed us to simply pick the two highest scoring letters and advertise those as our results. This approach has two drawbacks: (a) We can not correctly predict a CAPTCHA that has the same letter twice. (b) We are not able to predict the order in which the letters appear. Latter issue is addressed in section 5.3. Later we applied the same scanning methodology from the MNIST CAPTCHAs to this reCAPTCHA inspired dataset to improve results.

6.3 Multi-Class reCAPTCHA Prediction

Training with the enlarged image with randomly placed characters gave us lackluster accuracy, around 70%. As a result the two character accuracy was low (Figure 21). We suspect the model architecture need to be modified to accommodate the different training set. As we can see from the reconstruction of the image(Figure 9), the model does not recognize the characters well in our training set. But interestingly, we actually see an increase in accuracy with increased distance between characters.

6.4 Scanning reCAPTCHA Prediction

After finding unsatisfactory results from the multi-class prediction we decided to apply the scanning architecture we initially developed with MNIST. As with multi-MNIST this method had significantly better results on our reCAPTCHA dataset. After training on our augmented character dataset in a 28x28 window the scanning network was able to achieve a individual character-position accuracy of 0.52 and a CAPTCHA level accuracy of 0.4 (Figure 8). These results mean we can predict augmented random-offset minorly-overlapping two digit reCAPTCHA images with an accuracy of 40%.

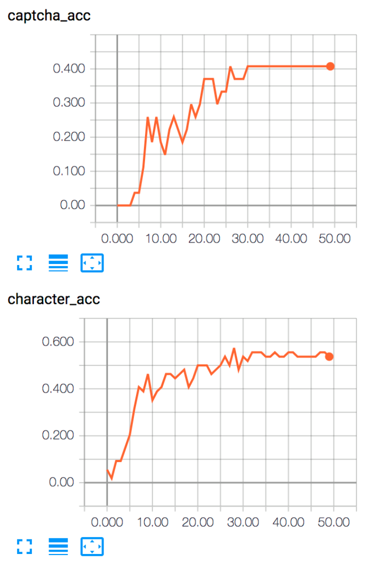

Figure 8: Character Accuracy and CAPTCHA accuracy for two character reCAPTCHA

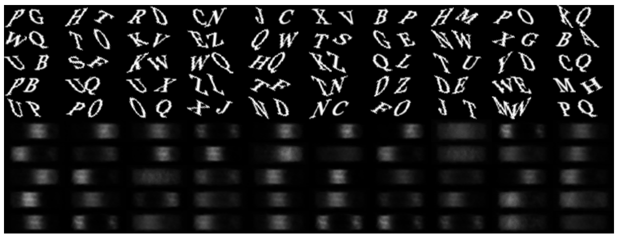

Figure 9: Reconstructed reCAPTCHA training image after training

7 Resistance of Gaussian Noise

We also experimented the model resistance on Gaussian noise to test if Capsnet possibly has an advantage on this over CNN. For simplicity, we selected MNIST as our training and testing data set.

7.1 Motivation

This section was motivated by the hypothesis that CapsNets might me more resistant to noise than CNNs are. We believe this due to the existance of the weight matrix in the routing procedure. When noise occurs in a regular CNN the higher activation of noisy regions is passed through the relevant convolutions and through the max pooling layers and makes its way to the final connected layers. This means that the final connected layers that do the actual learning from the convolutional features will be significantly influenced by the noise vectors. In the Dynamic Routing procedure the logits which are learned emphasize only the output vectors of the lower level capsules that are directly relevant to the upper level capsule. This theoretically allows the upper level capsule to shut out extraneous influences from non-relevant capsules in the lower layer meaning if the noise has an effect on irrelevant capsules then that noise will not make it to the upper level capsule.

7.2 Methodology

Gaussian noise was added to the original image to make it noisy. A Gaussian noise image was generated for each image with a set mean. The standard deviation was set to 2 times the mean so the noise level started at approximately 0. After adding the noise overlay to the original image, they were clipped so the image wouldn't contain negative value

We conducted 3 different tests to explore the affect of noise on Capsnet. 1. We trained the Capsnet and CNN normally, then tested it on noisy images. 2. We trained the Capsnet noise images with same noise level, and tested it on images with different noise levels. 3. We trained the Capsnet on images with random noise level, and tested it on images with different noise lebels. For this setup we also compared the CapsNet's performance with a CNN in the same conditions.

7.3 Results

For test 1, we observed a sharp decrease in terms of accuracy when predicting noisy image for Capsnet, and a more gracefully decrease in accuracy for CNN For test 2,

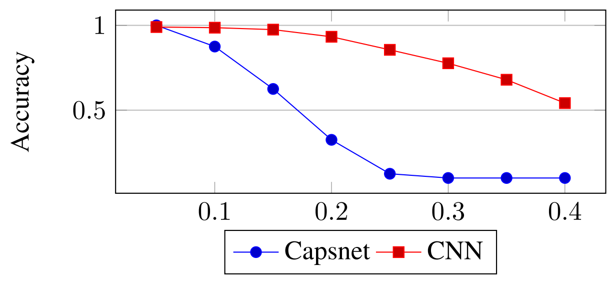

Figure 10: CapsNet trained normally tests on noisy images

where the model is trained with noisy images with certain noisy level, we observed the model can classify noisy images really well, but can no longer accurately classify clear images. Most likely this means that the model was able to learn the noise as an intrinsic feature, in particular an average value over the entire area.

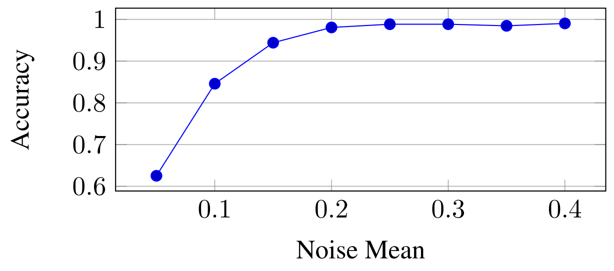

Figure 11: CapsNet, trained with noisy image with noise mean of 0.4, tested on noisy images with different noisy level

For test 3, we trained the model with images that random noisy level was added. We find out the model can classify clear and noisy images with very high accuracy. We also trained a CNN with same training image generation method. We observed marginally better accuracy Capsnet has over CNN.

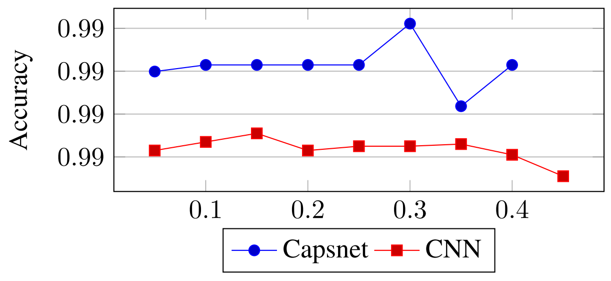

Figure 12: Capsnet and CNN trained with images with random noisy level, tested on images with different noisy levels

8 Conclusion

CapsNets are a very new architecture. The research to date has shown that CapsNets can be applied to data in which most features are relevant (such as MNIST or CAPTCHAs). In more complex situations CapsNets are not yet fit for use. This topic has shown to be an interesting field of research, especially in regards to the novelty of the approach. In practice, research has to develop a better understanding of Capsule Networks, before a real world application is feasible.

9 Code

Our code is up on github at `https://github.com/Neitsch/deep-learning-project`. The mnist_chainer directory contains a fork of a Chainer implementation of CapsNet that we have modified to allow data augmentation. Data augmentation takes place in train.py using the Keras image data aumentation object. The CapsNet-Keras contains a fork of a Keras based implementation that was more flexible to work with. The results directory contains a large portion of our results excluding some of those from the cnn which are in mnist_cnn. The reference CNN we used is the example implementation from Keras.

10 Figures

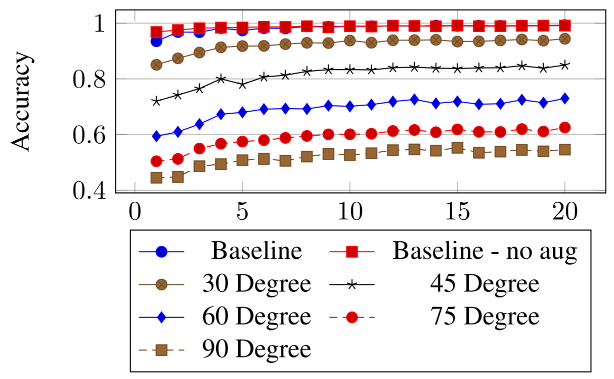

Figure 13: CapsNet performance at different rotations

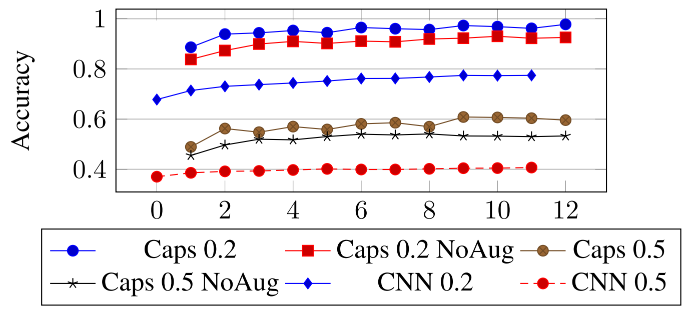

Figure 16: CapsNet vs CNN Performance on horizontal shift

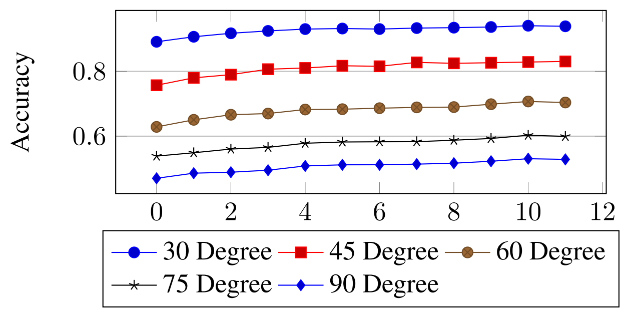

Figure 14: CNN performance at different rotations

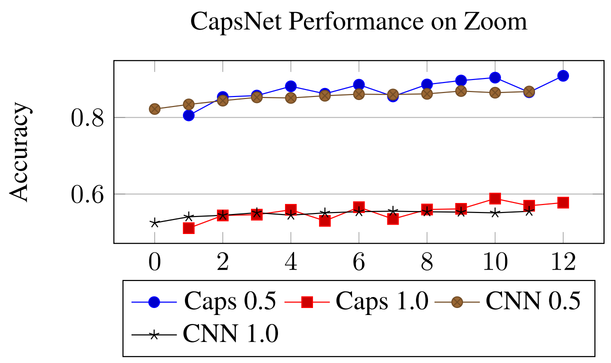

Figure 17: CapsNet vs CNN Performance on zoom

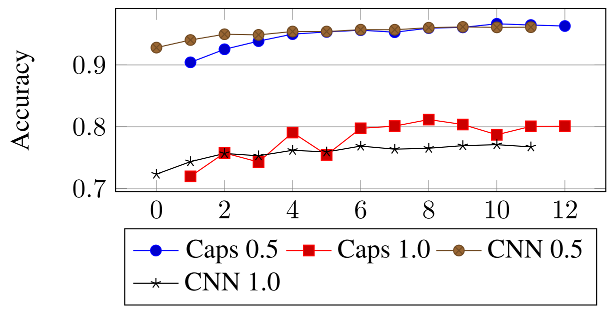

Figure 15: CapsNet vs CNN Performance on shear

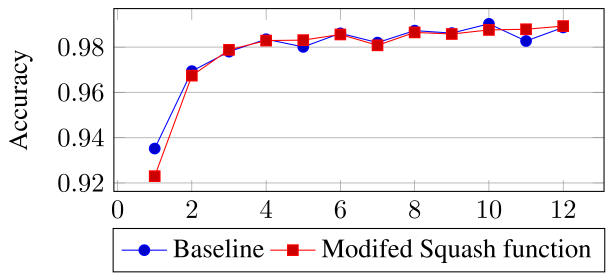

Figure 18: CapsNet performance with the new squashing function

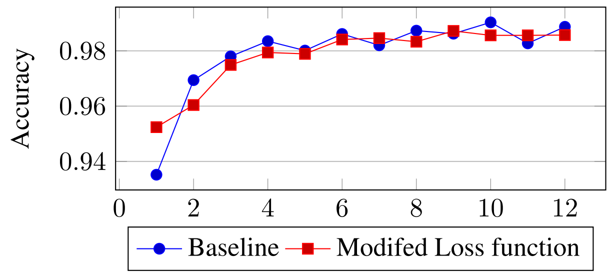

Figure 19: CapsNet performance with the new loss function

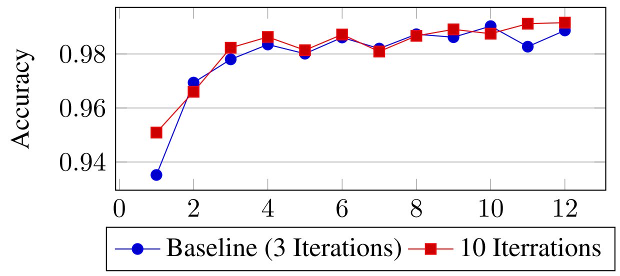

Figure 20: CapsNet performance with different number of routing iterations

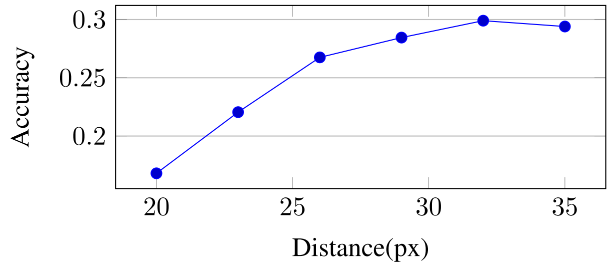

Figure 21: CapsNet performance when predicting two character captchas with different distance between characters

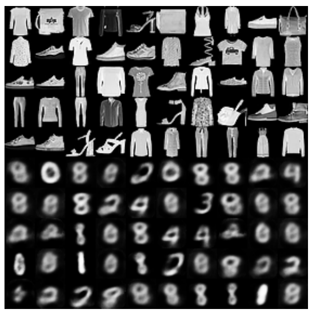

Figure 22: This is a fun image that shows what happens when you pass fashion mnist through a network trained on regular MNIST. You can see that some of the wedge shaped shoes become twos and the dresses become thin eights or ones.

11 Appendix

11.1 Matrix Capsules with EM Routing

The paper *Matrix Capsules with EM Routing* was publicized a few weeks after the Sabour et al. published the dynamic routing paper. This paper, currently under review, proposes a different routing algorithm that is based on a gaussian mixture model. As this paper was not the focus of our paper we include only a brief description of its methodology.

11.1.1 Expectation Maximization Routing

The goal of this method was to develop a more mathematically rigorous routing algorithm. While Dynamic Routing relies on the heuristic squashing function, the expectation maximization algorithm has a much deeper mathematical underpinning. Essentially the routing algorithm seeks to fit a mixture of Gaussians for each upper level capsule over each of the lower level capsules. The expectation maximization algorithm is used to fit these Gaussians providing a model of agreement for each i-j pair that is much more expressive than simply taking the dot product of two vectors (as is done in Dynamic Routing).

11.1.2 Resistance to Adversarial Attacks

Another interesting potential that we were unable to test is the potential of capsule networks to resist adversarial attacks as shown in the anonymous paper *Matrix Capsules with EM Routing* [2]. We hypothesize that this resistance is due to the influence of the weight matrices in the routing procedure. The goal of a universal adversarial attack on a categorical neural network is to shift images imperceptibly to a different part of the high-dimensional image-space that will be categorized as something else. Standard CNNs are vulnerable to this because the influence of the attack will be propagated through the convolutions to the connected layers where each initial neuron has the same set of inputs. CapsNets have a different architecture where there is a weight matrix between each i-j pair of lower and upper level capsules. The weight matrix effectively encodes a different view of the lower level capsules output space meaning that a shift in the outputs of one lower level capsule has different effects on each upper level capsule in the network. Because each upper level capsule effectively has a different view of the output space of a given lower level capsule it would be more difficult to create a universal adversarial attack than for a standard CNN.

References

[1] Sabour S., Frosst N., Hinton G.E. Dynamic Routing Between Capsules. arXiv:1710.09829, 2017. 1, 2

[2] Anonymous authors. Matrix Capsules with EM Routing. Under review as a conference paper at ICLR 2018. 11

[3] Zhou Y., Ye Q., Qiu Q., Jiao J. Oriented Response Networks. arXiv:1701.01833, 2017.

[4] Hinton G.E., Krizhevsky A., Wang S.D. (2011) Transforming Auto-Encoders. In: Honkela T., Duch W., Girolami M., Kaski S. (eds) Artificial Neural Networks and Machine Learning ICANN 2011. ICANN 2011. Lecture Notes in Computer Science, vol 6791. Springer, Berlin, Heidelberg 1

[5] George, D. and Lehrach, W. and Kansky, K. and Lázaro-Gredilla, M. and Laan, C. and Marthi, B. and Lou, X. and Meng, Z. and Liu, Y. and Wang, H. and Lavin, A. and Phoenix, D. S. A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs. http://science.sciencemag.org/content/early/2017/10/26/science.aag261 2017 5