Using Capsules for Robust Logo Classification

Christopher Dinh CDINH2@UMBC.EDU

1. Introduction

My goal for this project was to implement an algorithm that correctly classifies images of brand logos in a way that is robust to the background that the logo is placed on as well as to homography transformations. If combined with a method for locating logos in an image, this could be used for market analysis to determine where different brands' logos are more commonly seen on social media. For this application, it is important that the logo classification be very robust to homographies because the images may be taken from nearly any position relative to the logo.

In order to ensure that the model is robust to homographies, I trained and tested it on a dataset generated by augmenting a dataset of logos with homography transformations.

2. Related Work

2.1. Logos

Logo classification is a task for which there are many existing methods. (Guru & Kumar, 2016) and (Kumar et al., 2016) introduce two such methods achieving greater than 50% classification accuracy. There is also work such as (Bianco et al., 2017), which introduces a CNN that achieved $> 95\%$ accuracy.

In addition, there are existing logo detection methods that could be used as a preprocessing step to find appropriate input for a classifier. The models used in (Su et al., 2016) achieve over 70% mean average precision on the FlickrLogo-32 dataset of real-world images of logos.

Another interesting aspect of (Su et al., 2016) is that it introduces a similar training image generation method to mine. However, they used it to train a model that was then tested on real-world data as opposed to testing the model on the synthesized data to test its robustness.

None of these existing models, however, used capsules. This is likely because capsules were only introduced in (Sabour et al., 2017) near the end of 2017.

2.2. Capsules

A capsule is essentially just a small group of neurons in a larger network layer. The outputs of these neurons are grouped together into an output vector. The output of the layer as a whole is determined based on the outputs of the capsules rather than the neurons individually.

The implementation that is described in (Sabour et al., 2017) assumes that the output of a capsule represents the existence and attributes of a specific entity such as a line.

The output vector's magnitude is treated as the probability that the entity exists in the input, so a function is defined that "squashes" it to keep its magnitude in the range $[0, 1)$.

The output vector's direction is treated as representing the attributes of the entity such as position, orientation, etc. To calculate this, (Sabour et al., 2017) also defines a routing procedure that determines the output of the whole layer as a weighted sum of the outputs of the capsules.

To ensure that this weighted sum maintains the properties of the magnitude, a softmax function is run on the weights to ensure that they sum to 1.

In (Sabour et al., 2017), a CapsNet is described based on these definitions which is shown to effectively classify MNIST and be robust to affine transformations.

3. Method

3.1. CapsNet Model

This model architecture is effectively identical to the basic CapsNet introduced in (Sabour et al., 2017), however the input was RGB rather than grayscale.

As such, its input is an $n \times n \times 3$ image. The first layer is a convolutional layer with 256 9×9 kernels with a stride of 1.

The second layer is called the PrimaryCaps layer. It consists of 32 capsules consisting of 8 9×9 convolutional kernels each that are used with a stride of 2.

Each of these kernels has an output that is $\frac{n-16}{2} \times \frac{n-16}{2}$. For simplicity of notation, let $m = \frac{n-16}{2}$

Each capsule's output is $m \times m \times 8$. By combining the output of all 32 of the capsules, the output of the PrimaryCaps layer is interpreted as $32m^2$ 8-dimensional vectors.

Next, these vectors are fed into the DigitCaps layer. This layer contains a separate 8×16 weight matrix for each of the $32m^2$ vectors. After using those weight matrices to map the vectors into 16-dimensional space, dynamic routing is used to calculate 10 16-dimensional vectors that are the output of the CapsNet.

The calculation of these output vectors involves performing a weighted sum over the 16-dimensional vectors where the weights sum to 1 for each output. Each of these outputs v_i is a 16-dimensional vector corresponding to a class label.

For each brand i, $||v_i||$ is the predicted probability that its logo is in the input image and the direction of v_i encodes the logo's attributes including the homography used to transform it.

In addition to changing the input, I needed to modify the loss function. The original loss function in (Sabour et al., 2017) has two terms, the margin loss and the reconstruction loss. The margin loss encourages correct classification while the reconstruction loss encourages the model to encode the attributes of the logo.

To calculate the reconstruction loss, the 16-dimensional output vector corresponding to the true class is passed through a 3-layer fully connected network and the result is evaluated on similarity to the reconstruction target using the mean squared error. In the original formulation, the reconstruction target was simply the input image.

This is still used for the Logo dataset, but in the Logo + Background set, the input image includes the background that the logo was placed on. If that was used as the target, the model would be encouraged to represent the background in the features that it extracts.

This is not desirable as the logo is the only important part of the image, so I used the transformed logo without the background as the reconstruction target instead. This encourages the network to learn to represent the logo's transformation and position without paying attention to the background.

4. ConvNet Model

I also trained a simple convolutional network with two convolutional layers and three fully connected layers. It was designed to emulate the structure of a CapsNet as much as possible.

The first layer is a convolutional layer with 256 9×9 kernels with a stride of 1, identical to the first layer of the CapsNet. The second layer is another convolutional layer with 256 9×9 kernels with a stride of 2, which essentially performs the same function as the PrimaryCaps layer.

The third layer is a fully connected layer that takes the output of the convolutions and embeds it into a higher-dimensional space, emulating the DigitCaps layer. A LeakyReLU activation function was used as it has the benefits of ReLU without the possibility of dead neurons.

The fourth layer is a fully connected layer that takes that higher dimensional space and compresses it into 16 dimensions per class, similar to the dynamic routing. A LeakyReLU activation function was used here as well.

The final layer is a fully connected layer that uses a sigmoid function to express the output as a vector of predicted probabilities that each logo was present in the image.

4.1. Data

To evaluate the CapsNet's robustness to homographies, I trained and tested it on images generated from brand logos by transforming them with a random homography and placing them on backgrounds.

I retrieved 10 brand logos using Google Image Search and rescaled them to a standard size with a black background. The backgrounds were sampled as patches of 4K images of real-world environments.

4.1.1. Instance Generation

The following is the procedure that I used to generate a $3 \times n \times n$ instance from an $3 \times a \times b$ RGB logo image $L = [l_{ijk}]$ where i is the channel, j is the x-coordinate, and k is the y-coordinate.

Generate four random points $(x_i, y_i) \in [0, n] \times [0, n]$ and use the Graham scan algorithm to find an ordered convex hull. Repeatedly generate points until the convex hull contains all four and none of the corners of the hull has an angle less than 60 deg or greater than 120 deg. This angle condition ensures that the result of the transformation resembles the original logo to some degree.

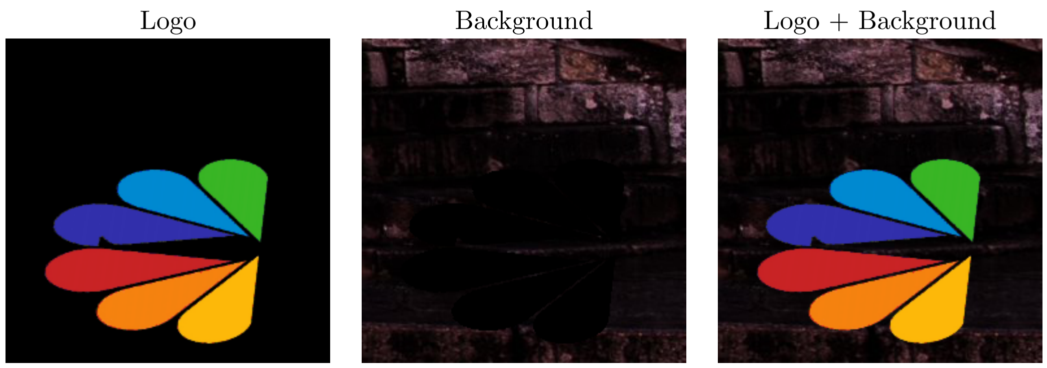
In order to include mirroring as a possible transformation, reverse the order of the convex hull with 50% probability. Once this is done, there is an array $[(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)]$ defining the corners of a convex quadrangle in order.

Use these corners to compute the perspective transform M from the original corners of the logo to these quadrangle corners in the instance image such that

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} = M \begin{bmatrix} 0 & a & a & 0 \\ 0 & 0 & b & b \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Use this to transform the logo image and place the result in an $3 \times n \times n$ image $T = [t_{ijk}]$ giving any locations that do not correspond to a location in the logo image a value of 0. If the Logo dataset is being used, this is the final result.

If the Logo + Background dataset is desired, however, select a random $3 \times n \times n$ patch $B = [b_{ijk}]$from one of the background images. To generate the instance $P = [p_{ijk}]$, replace the empty pixels in T with the corresponding pixels in B.



5. Experimental Results

While I was evaluating these models, both the training and test steps were conducted by generating image instances as needed with the training step using 10× more instances than the test step. As a result, there was effectively no difference in the losses between the training and test steps as the data was drawing from exactly the same distribution.

Due to memory limitations, these instances were limited to a 25×25 resolution. Both models were trained for 100 epochs consisting of 1000 training instances and 100 test instances using the Adam optimizer. I found that even with Adam, the results for the CapsNet were highly sensitive to the learning rate.

Using this input size, the CapsNet model contained 8,932,755 parameters to train and the ConvNet model contained 89,353,706 parameters. However, the CapsNet took roughly twice as long to run as the ConvNet, both in training and testing. This is likely a result of the CapsNet requiring more actions to be run in Python rather than in optimized compiled code.

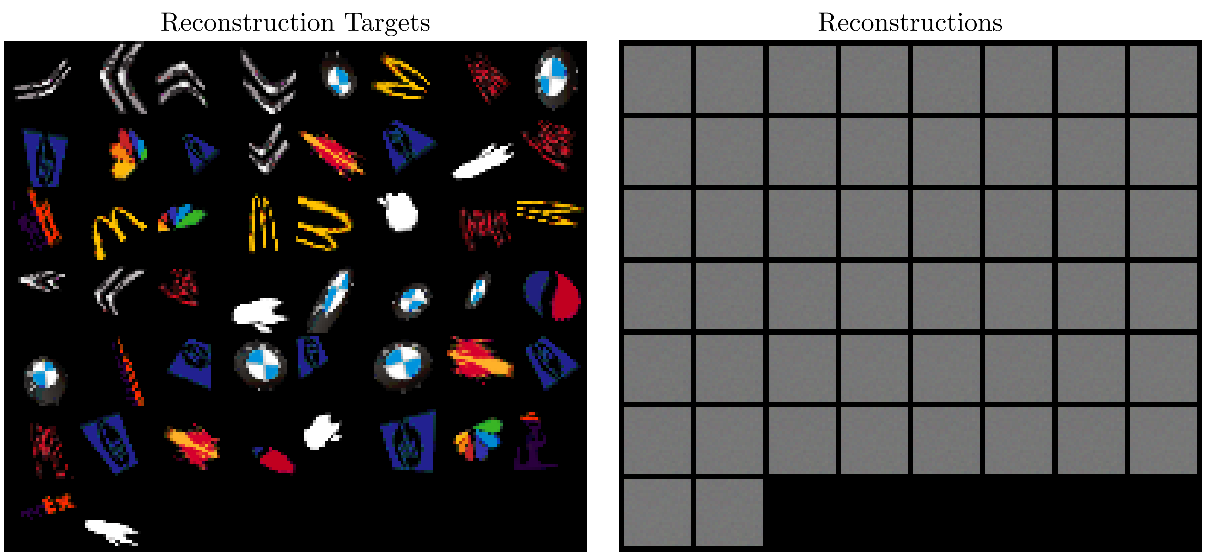
Dataset	Model Type	Classification Accuracy
Logo	CapsNet	73%
Logo	ConvNet	18%
Logo + Background	CapsNet	31%
Logo + Background	ConvNet	10%

As expected, both models performed much worse on the Logo + Background dataset due to it being a more complex domain.

Interestingly, based on examining the output of the ConvNet while it was running on the Logo dataset, it is only learning to choose between two of the classes. As a result, it classifies those classes fairly accurately, but it fails completely at all of the other classes, causing the accuracy to be only slightly above that of random chance. The CapsNet, however, manages to learn to classify all of the logos to some degree despite having many fewer parameters with which to perform that classification.

In the Logo + Background dataset, however, the ConvNet is simply learning to guess a single class, resulting in accuracy that is exactly as expected from random chance. The CapsNet doesn't fall into this trap, it simply fails to classify effectively.

I also found that the CapsNets failed to learn to reconstruct the logo images despite classifying them relatively well, indicating that the reconstruction loss was not particularly helpful for classification. The reconstruction targets were the same for both the Logo and the Logo + Background datasets.



6. Conclusion

Based on my experiments, the CapsNet architecture is superior to a similar ConvNet as it achieves roughly triple the classification accuracy with 1/10 as many parameters. In addition, as the datasets include a large number of possible transformations of the logos being classified, the CapsNet seems to be highly robust to homography transformations, not the affine transformations demonstrated in (Sabour et al., 2017). Finally, the reconstruction loss might be useful if the CapsNet is used as a feature extractor, but for classification it doesn't seem to help at all.

References

Bianco, Simone, Buzzelli, Marco, Mazzini, Davide, and Schettini, Raimondo. Deep learning for logo recognition. *CoRR*, abs/1701.02620, 2017. URL `http://arxiv.org/abs/1701.02620`.

Guru, D. S. and Kumar, N. Vinay. Symbolic representation and classification of logos. *CoRR*, abs/1612.08796, 2016. URL `http://arxiv.org/abs/1612.08796`.

Kumar, N. Vinay, Pratheek, Kantha, V. Vijaya, Govindaraju, K. N., and Guru, D. S. Features fusion for classification of logos. *CoRR*, abs/1609.01414, 2016. URL `http://arxiv.org/abs/1609.01414`.

Sabour, Sara, Frosst, Nicholas, and Hinton, Geoffrey E. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017. URL `http://arxiv.org/abs/1710.09829`.

Su, Hang, Zhu, Xiatian, and Gong, Shaogang. Deep learning logo detection with data expansion by synthesising context. *CoRR*, abs/1612.09322, 2016. URL `http://arxiv.org/abs/1612.09322`.