Capsule Networks

Nikhil Sardana

March 2018

1 Introduction

Capsule Networks are a new type of neural network designed to overcome some of the shortcomings of traditional convolutional neural networks (CNNs). Capsule Networks were introduced by Geoffrey Hinton, one of the pioneers of deep learning, in November 2017, but were hypothesized by him much earlier. Capsule Networks, despite their promising early results, still have numerous shortcomings. So hold on to your pooling layers, your ConvNets aren't obsolete just yet!

2 Convolutional Network Shortcomings

Convolutional Neural Networks (CNNs) have proven extraordinarily effective in a variety of computer vision tasks. They represent a great leaps in terms of both speed and performance for image classification, object detection, and semantic and instance segmentation. Nonetheless, they rely on a few tricks to obtain such results.

2.1 The Curse of the Pooling Layer

Max-pooling has become an integral part of convolutional networks structures. Pooling layers serve to reduce the feature map size, reducing computation while retaining the most important information. In addition, they allow networks to be **translationally invariant**. That is, a convolutional network has no trouble classifying an object if it is translated in an image—say, located in the top left instead of the middle. That's because *pooling layers lose spatial information*. We don't know where exactly the maximum value came from for each $n \times n$ pooling section, and thus the maximum values, which represent features, can be shifted in the image without loss of accuracy.
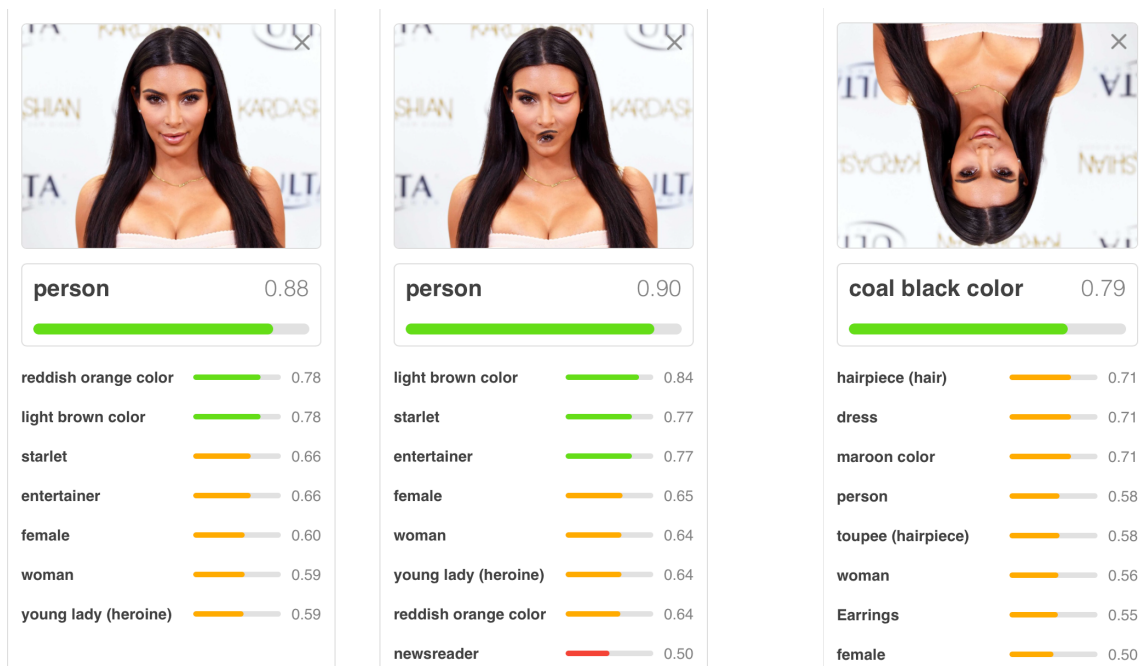
max pool with 2x2 filters and stride 2
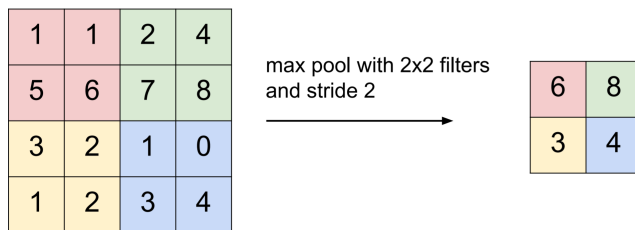
The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.

Geoffrey Hinton

So why then, does Hinton believe pooling layers are so terrible? So far, everything we've said about pooling layers seems pretty great. However, if you read the above paragraph more closely, you'll notice one particular phrase sticks out: *pooling layers lose spatial information*. Why is this so important? Consider a standard convolutional network. Later layers in a convolutional network learn higher level features from the lower-level information found in earlier feature maps. But if we don't know where exactly the lower level features lie in relation to each other spatially, then convolutional networks can be susceptible to profound errors.

In the images below, we know that human faces are made up of eyes, a nose, and lips. A convolutional network has clearly learned this too. But a convolutional network doesn't care where these features are

located relative to each other—it only cares if they are present. That's because pooling layers remove the spatial information generated by lower-level feature maps, which detect the eyes, nose, and mouth objects. Rotating the images proves another challenge for the convolutional network. Now, the features are upside down, so it has a much harder time detecting them.

person	0.88
reddish orange color	0.78
light brown color	0.78
starlet	0.66
entertainer	0.66
female	0.60
woman	0.59
young lady (heroine)	0.59

person	0.90
light brown color	0.84
starlet	0.77
entertainer	0.77
female	0.65
woman	0.64
young lady (heroine)	0.64
reddish orange color	0.64
newsreader	0.50

coal black color	0.79
hairpiece (hair)	0.71
dress	0.71
maroon color	0.71
person	0.58
toupee (hairpiece)	0.58
woman	0.56
Earrings	0.55
female	0.50

This is precisely why data augmentation—the rotation, shifting, zooming, and reflection of images to expand a training dataset—is so important. A convolutional network, when presented with enough examples of upside down faces, will also learn to classify them. But it has to learn many new features to classify an upside-down person, even if it can already classify a regular person with great accuracy.

2.2 Equivariance

Our goal is to have a network that can classify objects regardless of translation, rotation, scale, etc. We can accomplish this by having a network that not only classifies the probability of an object existing in a certain region, but also the rotation, translation, scale, etc. of the object. Changes in the viewpoint, or pose of an object, should be reflected in the neurons of a network. This principle is called **equivariance**.

2.3 Interpretability

Convolutional Networks are not very **interpretable**; it is difficult to see how neurons interact with each other and what features a particular neuron has learned. Visualizing feature maps is one method to yields some insight, but is still limited in the information it provides. Partial occlusion is another way to see how a convolutional network as a whole has learned features, but is not layer-specific and thus not very useful for tuning networks. As you'll later see, Capsule Networks are far more interpretable; changing capsules directly influence high-level features as the network reconstructs images.

3 Network Architecture

What exactly is a capsule network? And how does it address the shortcomings listed above of convolutional networks?

Capsule Networks begin the way standard convolutional networks do: an image input followed by a couple convolutional layers. The original Capsule Network architecture is shown below. For the MNIST

input (28×28), after the first convolutional layer, we have a $20 \times 20 \times 256$ tensor. We then perform another 9×9 convolution, but this time of stride 2, resulting in a $6 \times 6 \times 256$ tensor.
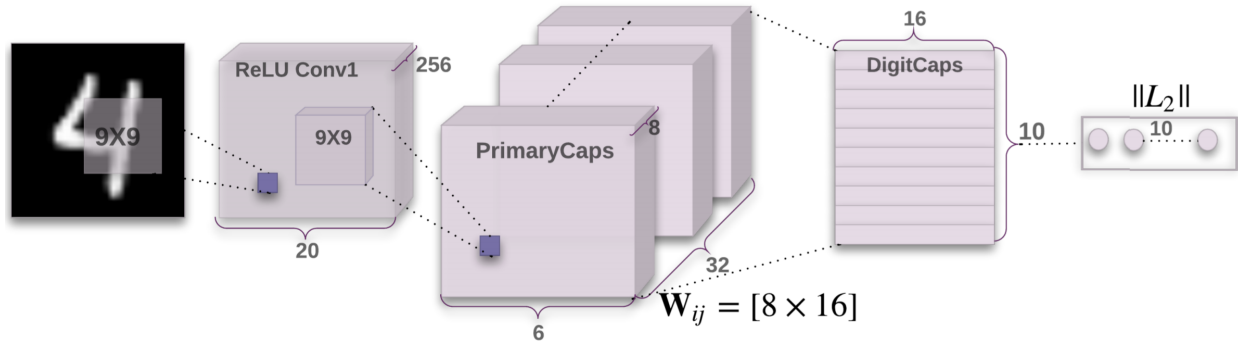
This is where the interesting stuff happens. The $6 \times 6 \times 256$ tensor is *reshaped* as shown in the figure below right. We slice the 256 feature maps every 8 maps, creating 32 different tensors. At every pixel, there are 32 sets of 8 values.

We can think of the 256 feature maps as a 256-d vector at each pixel of a 6×6 grid. After reshaping, this 256-d vector is now 32 8-d vectors. Each of these 8-d vectors is called a **capsule**.
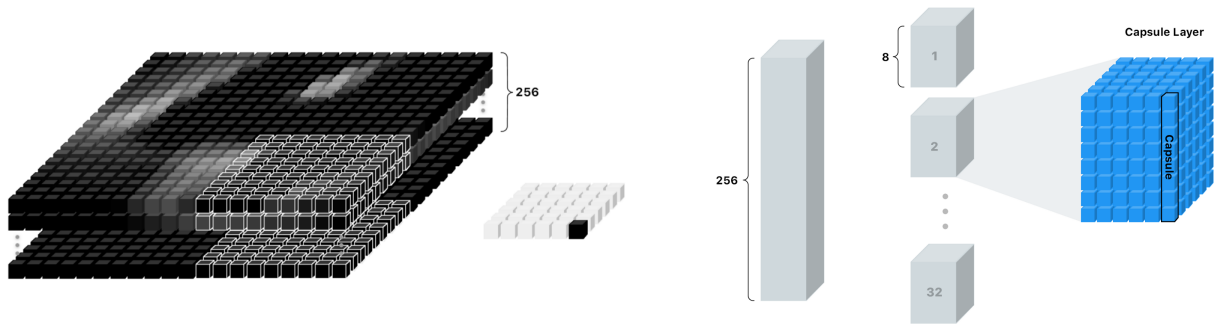
We can think of Capsule Networks as vectorized neural networks. A neural networks is composed of neurons, which are scalar values. A capsule network (at least, the capsule layers of a capsule network) is composed of capsules, which as we've seen, are just vectors.

3.1   Capsules

What purpose do capsules serve? Simply put, we think of the length of a capsule at a certain location as the probability that an object (or feature) exists at that location. The elements of the capsule represent the state of the detected feature. For example, if two images show the same object, but one images shows the object rotated, the length of the capsule detecting the object should remain the same for both inputs, but the direction of the capsule will change. Capsules are able to encode information about the relative position, rotation, scale, etc. of an object, making them equivariant. Hinton refers to the information about an object's appearance as *instantiation parameters*, and capsules are able to capture and retain that information in ways a traditional convolutional network never could.
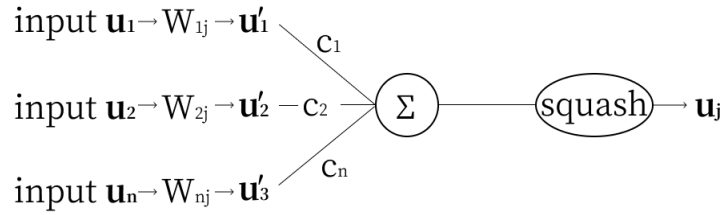
3.2   Forward Propagation

How exactly do the capsule networks work? How do they forward propagate, backpropagate, and learn to classify and to understand instantiation parameters?

The capsule layers differ significantly from a standard neural network their "forward propagation" and "backpropagation" techniques.
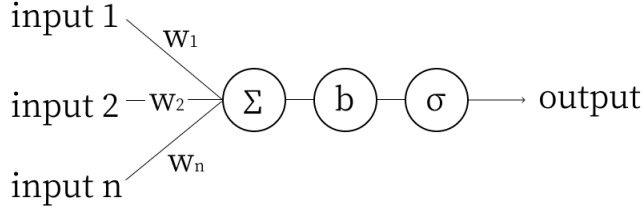
Consider the standard neuron above. Forward propagation takes three steps:

1. Scalar weighting of input scalars

2. Summation of weighted inputs and bias

3. Non-linear activation function (Sigmoid, tanh, ReLU, etc.)

By contrast, the capsule network forward propagates in four steps:

1. Matrix transformation of input vectors

2. Scalar weighting of input vectors

3. Summation of weighted input vectors (without the addition of a bias)

4. Non-linear vector squashing function

The first step, the matrix transformation of input vectors, stands out here as it has no scalar equivalent in the neuron. These transformation matrices serve to encode spatial information relating the lower-level features learned by the input vector to higher-level features. For example, if capsule j is trying to detect a face, the inputs $\vec{u}_1$ might contain information about an eye, $\vec{u}_2$ about a nose, and $\vec{u}_3$ about a mouth. Then, matrix W_{1j} might contain information predicting the position, scale, and rotation of the face given the eye feature. W_{2j} would contain the same given the nose feature, and W_{3j} given the mouth feature. If these vectors agree in size and direction, then we know a face is most likely at that location. Weight matrices are learned through standard backpropagation.

For some input vector $\vec{u}_i$, we have a transformation matrix W_{ij}, a prediction vector $\vec{u'}_{ij}$, and a scalar weight c_{ij} for every capsule j in the next layer. The interesting part is the scalar weighting of these input vectors. We have to learn how large each c_{ij} should be. In other words, we need to determine which higher-level features a lower-level feature should correspond to. For example, if we have two overlapping objects, it may be difficult to tell which features correspond to which object. The scalar weights will determine which higher-level feature (object) the lower-level features are associated with.

Finally, these input vectors are summed, and then squashed to a length between 0 and 1. This squashing serves the same purpose as activation functions of a standard neuron. We don't want our vector sum to be of a great length, and overpower the rest of the network. However, we also don't want to just cap the magnitude of these vectors at 1. The squashing function is as follows:

$$\vec{u}_j = \frac{||\vec{s}_j||^2}{1 + |\vec{s}_j||^2} \frac{\vec{s}_j}{||\vec{s}_j||}$$

4

where $\vec{s}_j$ is the summation of the weighted input vectors, and $\vec{u}_j$ is the final output vector of the capsule. Visualizing the squashing function for a scalar value, we get the following function. The non-linearity of the function is clear.
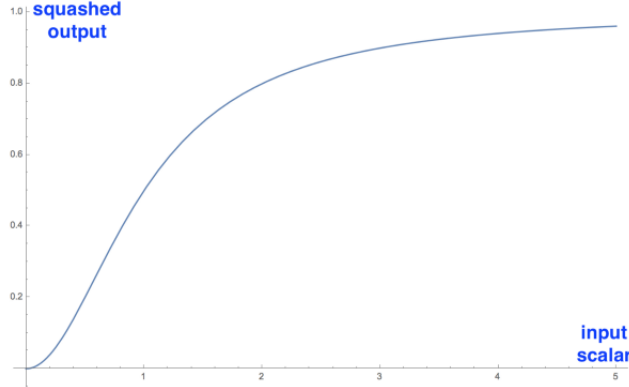
3.3   Dynamic Routing

How do we determine the scalar weighting of input vectors? Hinton's solution is called **Dynamic Routing**, or **Routing by Agreement**, and we can think of it as the equivalent of backpropagation for capsule networks: it allows us to learn which higher-level features should be associated with a particular low-level feature.

Considering again the computation of the output of a capsule, we have some input vector $\vec{u}_i$, multiply by a weight matrix W_{ij}, which encodes spatial information about it, generating a prediction vector $\vec{u'}_{ij}$. This prediction vector is predicting the position, scale, rotation, etc. of the higher-level feature that capsule j is trying to predict. If $\vec{u'}_{ij}$ is mostly correct, it should agree in direction with $\vec{u}_j$ (the output of the capsule). We can thus compute a similarity score between the output and a prediction vector:

$$b_{ij} = \vec{u'}_{ij} \cdot \vec{u}_j$$

Thus, b_{ij} takes into account similarity of features, or similarity of *instantiation parameters*. If the lower level features predict the instantiantion parameters well, then b_{ij} will be larger because the prediction vector will be closer in direction to the output vector. We can then adjust the coefficients to change the scalar weighting using the softmax like so:

$$c_{ij} = \frac{e^{b_{ij}}}{\sum_{k=0}^{n} e^{b_{ik}}}$$

where n is the number of capsules in the layer containing capsule j. In practice, we update b_{ij} multiple times, like so:
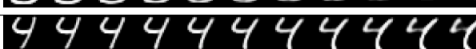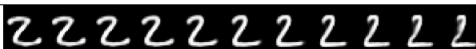
$$b_{ij} = b_{ij} + \vec{u'}_{ij} \cdot u_j$$

We can repeat this process for the next layer of capsules, just like we would for a neural network.

4   Performance

Capsule Networks have numerous advantages over standard convolutional networks. On MNIST, for example, Capsule Networks require far fewer parameters to reach equivalent performance. Capsule Networks also need less training data than convolutional networks.

Capsule Networks are also far more interpretable. Perturbing some of the capsules in a network trained on MNIST, we can clearly see the high-level features each capsule has learned.

Scale and thickness	
Localized part	
Stroke thickness	
Localized skew	
Width and translation	
Localized part	

There are quite a few drawbacks, however. Capsule networks are slow, especially for the number of parameters. This is attributed to the dynamic routing algorithm; it is far slower than the traditional and easily-parallelizable convolutional forward and backpropagation. In addition, capsule networks are untested on larger images (MNIST and CIFAR10 are datasets of very small images), and running on more traditional 300×300 images may yield vastly different results.

5 Conclusion

Hinton spent years working on Capsule Networks (he published a paper outlining capsules in 2011), but could not achieve groundbreaking or even equivalent results to the best convolutional networks. If one of the pioneers and great contributers to deep learning couldn't get capsule networks working with the support of Google's resources, are they really destined for greatness? Do they have the potential to solve the drawbacks of convolutional networks in practice, not just in theory? Can dynamic routing be made efficient enough to create larger, more potent capsule networks that train faster? Capsule Networks are nascent; no one knows if they will replace our current networks, even though they do seem to approximate human vision and work towards solving fundamental issues plaguing convolutional networks. Hinton may be barking up the wrong tree, but given his track record, it may not be wise to bet against him. Only time will tell if capsule networks will change the field of machine learning, or if they will slowly fade away and remain an interesting experiment.