

Capsule Networks

Sargur Srihari
srihari@buffalo.edu

This is part of lecture slides on [Deep Learning](http://www.cedar.buffalo.edu/~srihari/CSE676): <http://www.cedar.buffalo.edu/~srihari/CSE676>

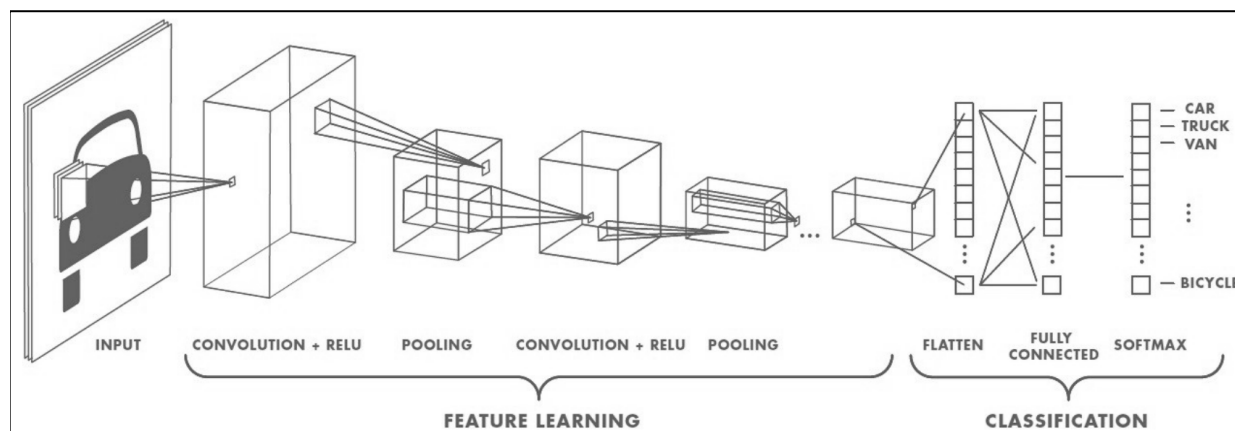
Outline

1. Limitations of Convolutional Networks
2. Definition of Capsule Networks
3. Dynamic Routing
4. Computing input/output vectors of a capsule
5. Routing by Agreement
6. CapsNet Architecture
7. Performance of Capsule Networks
8. Matrix capsules with EM routing

Recall Convolutional Network Architecture

• Convolutional Neural Networks

- Minimize computation compared to a regular neural network
- Convolution simplifies computation to a great extent without losing the essence of the data
- They are great at handling image classification
- They use the same knowledge across all image locations



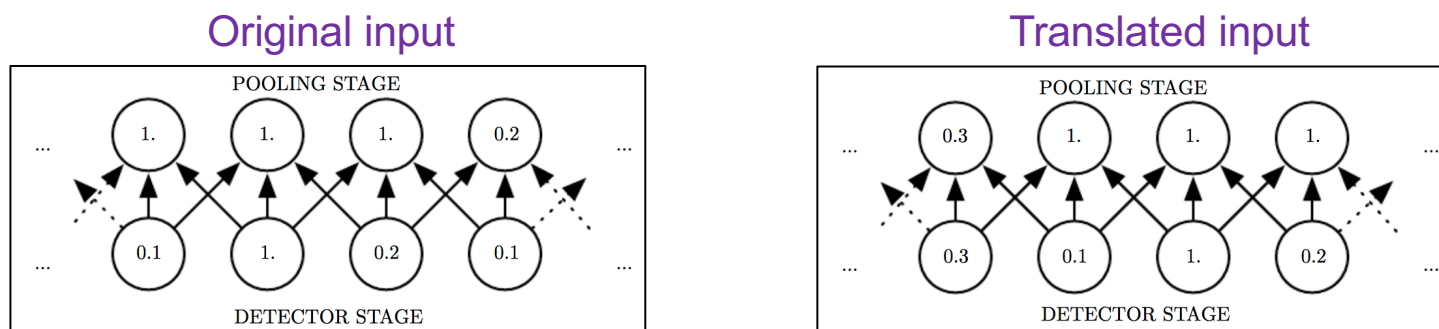
Source: <https://hackernoon.com/what-is-a-capsnet-or-capsule-network-2bfbe48769cc>

Processing Steps and Training for ConvNets

1. Given an input image, a set of kernels or filters scan it and perform the convolution operation.
2. This creates a feature map inside the network.
 - These features next pass via activation and pooling layers
 - Activation layers, e.g., ReLU, induce nonlinearity
 - Pooling (eg: max pooling) helps in reducing the training time.
 - Pooling creates “summaries” of each sub-region.
 - Helps in invariance to transformations
3. At the end, it will pass via a classifier sigmoid/softmax
 - Training is based on back propagation of error matched against labeled data.
 - Non linearity also helps solve the vanishing gradient problem

Pooling and Invariance

- Pooling is supposed to obtain positional, orientational, proportional or rotational invariance.

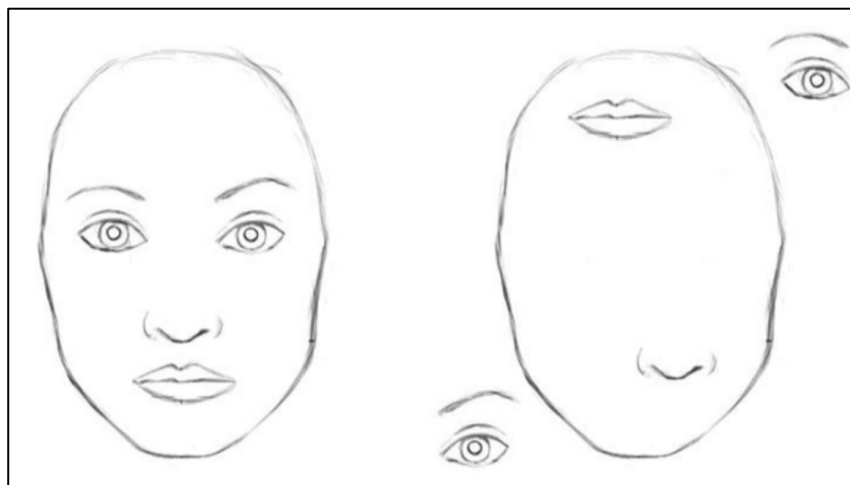


Every input value changed, but only half the output values have changed because maxpool is only sensitive to max value in neighborhood not exact value

- But it is a very crude approach
 - In reality it removes all sorts of positional invariance

Example of CNN Limitation

- CNN to recognize faces extracts features from image
 - E.g., eyes, eyebrows, a mouth, a nose
 - Then higher level layers combine those features and then check if all were found regardless of order
 - Mouth, nose have switched places but CNN still classifies it as face
 - Problem exacerbates as network gets deeper and as features become more abstract and shrink in size due to pooling and filtering



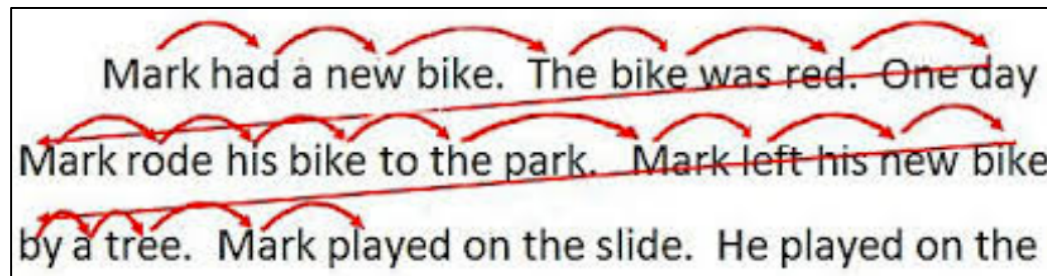
Motivation for CapsNets

- Caps nets are an improvement on CNNs
 - They are the next version of CNNs
 - Solve problems due to max pooling and deep nets
 - Loss of information regarding order and feature orientation
 - Hinton: “The pooling operation used in CNNs is a big mistake and the fact that it works so well is a disaster”

Solution offered by CapsNets

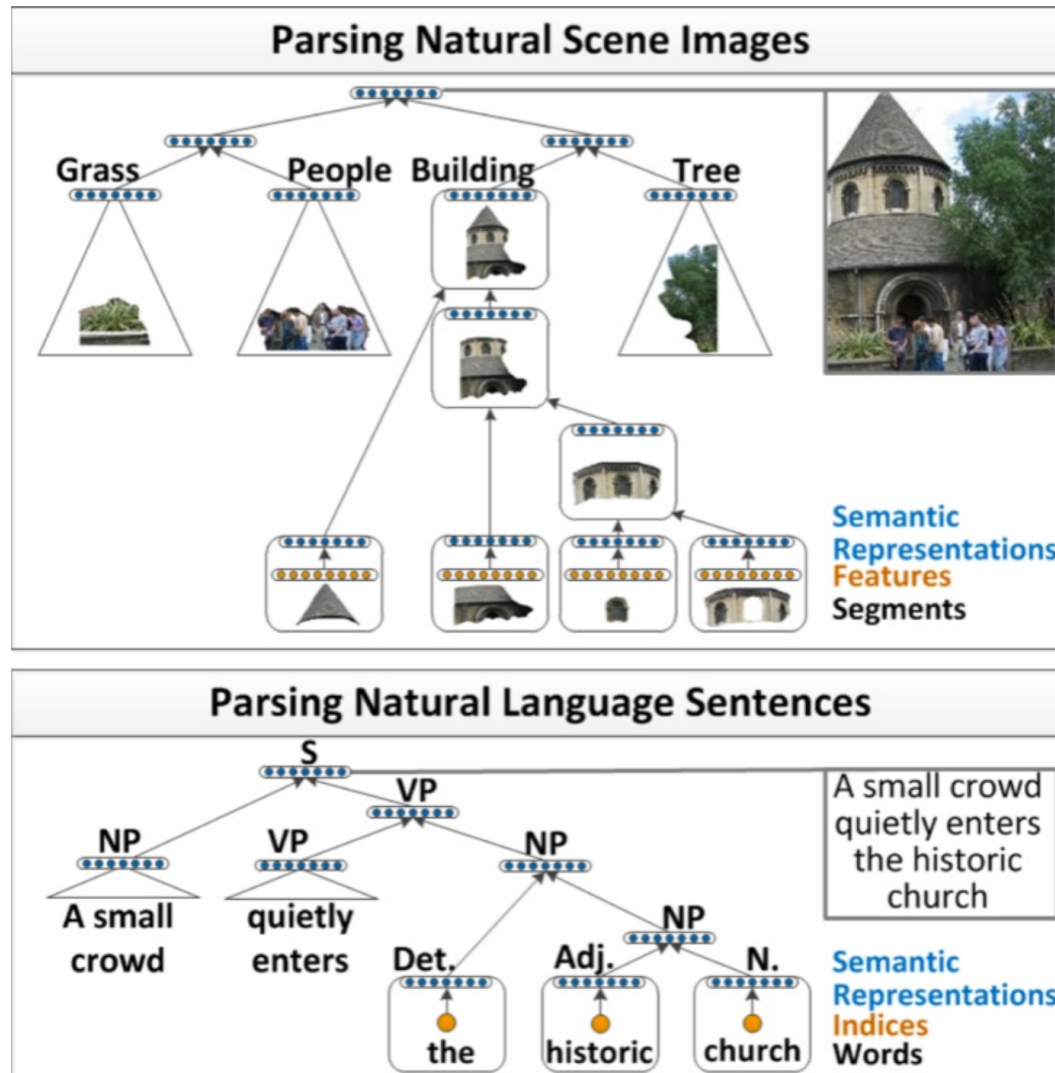
- Low level features should also be arranged in a certain order for the object to be classified as a face
- Order is determined during training when the network learns not only what features to look for but also what their relationships to one another should be
 - Learn that nose is between eyes and mouth below that
 - Images that have features in that particular order will then be classified as a face, everything else rejected

- Human vision uses saccades
 - Ignores irrelevant details by a careful sequence of fixations
 - Ensures only a tiny fraction of the optic array is ever processed at the highest resolution



- We assume a single fixation will give us
 - Much more than a single identified object and its properties
- Assume our multilayer visual system creates a parse tree on each fixation
 - We ignore coordination of parse trees over multiple fixations

Parse Trees



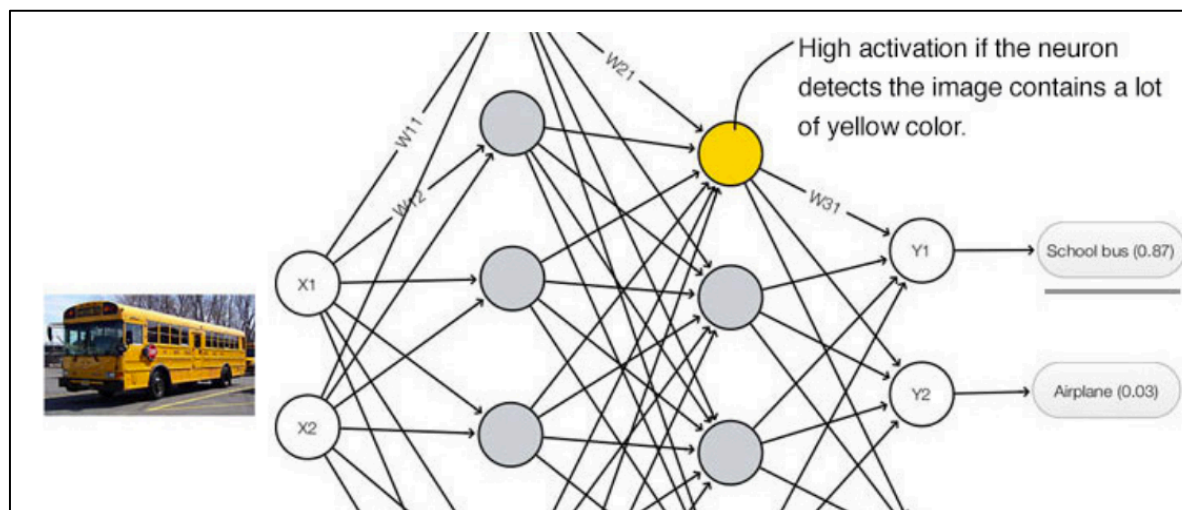
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.221.4910&rep=rep1&type=pdf>

Parse Tree of a Fixation

- For a single fixation,
 - A parse tree is carved out of fixed multilayer neural network
 - Like a sculpture from a rock
- Each layer will be divided into many small groups of neurons called “capsules”
- Each node in the parse tree will correspond to an active capsule

Activation is a likelihood

- The activation level of a neuron can be interpreted as the likelihood of detecting a specific feature



Source: <https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/>

- A capsule is a group of neurons that not only capture the likelihood but also the parameters of the specific feature.



CNN versus CapsNets

- Artificial neurons traditionally output a scalar, real-valued activation that loosely represents the probability of an observation
- CNN is scalar in nature
 - Max pool layers just try to get important information from images but lose structural orientation of those features
 - Due to which at higher level CNN only checks whether the feature is present or not irrespective of position they are in
 - Similar to bag-of-words in NLP
- Capsnets replace scalar-output feature detectors with vector-output capsules and max-pooling with routing-by-agreement.

Definition of Capsule

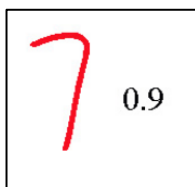
- A capsule is a group of neurons whose outputs represent different properties of the same entity
- A capsule is a group of neurons whose activity *vector* represents the instantiation parameters of a specific type of entity such as an object or object part

Example of capsule and activity vector

Conventional neuron (scalar output)


Single Neuron
(scalar output)

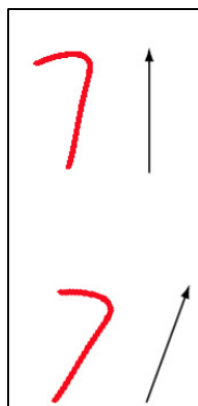
 Neuron



Conventional network:
With a single neuron probability of detecting “7” is 0.9

Capsule With 2 Neurons (vector output)

 Capsule



Digit rotated
By 20 degrees

A 2-D capsule is formed by combining 2 neurons
This capsule outputs a 2-D vector in detecting “7”.

It outputs a vector $v=(0, 0.9)$

Magnitude $\|v\|=\sqrt{0.81}=0.9$ is the probability of “7”

Direction is $\tan^{-1}(0.9/0)=\tan^{-1}(\infty)=\pi/2=90$ degrees

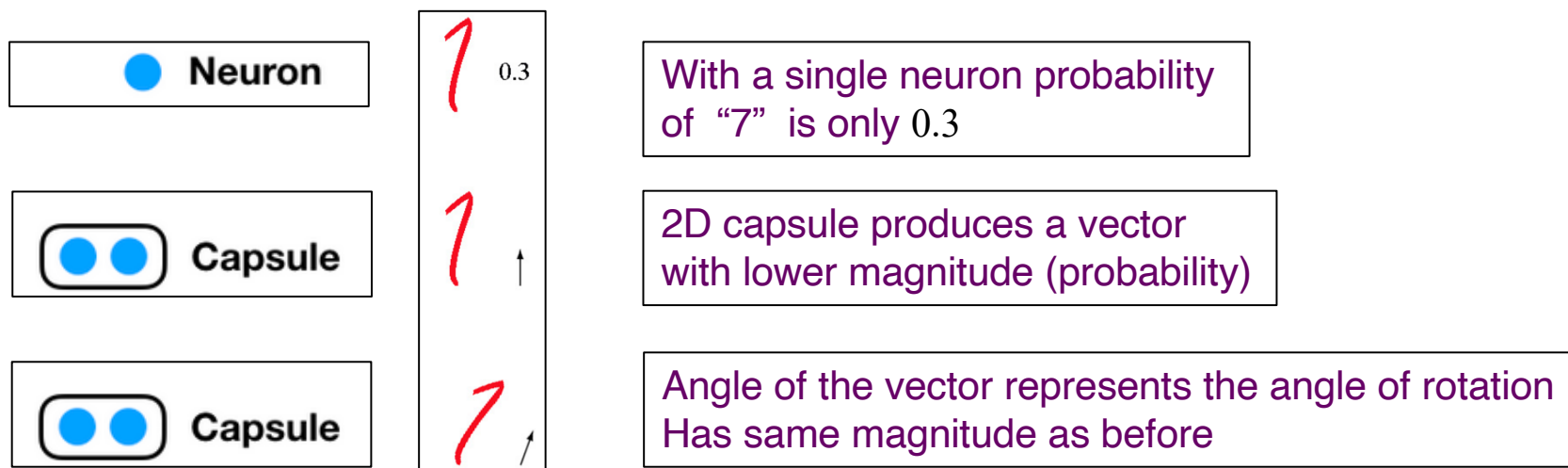
Angle of the vector represents the angle of rotation

Output vector $v=(0.1,0.84)$

Has same magnitude of 0.9

Direction is $\tan^{-1}(8.4)=83$ degrees

Another example of 2-D capsule

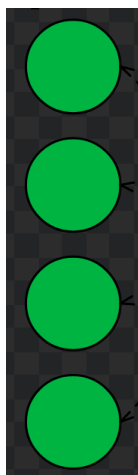


Representing Entity in Input

- Within an active capsule
 - Its neurons represent various properties of an entity present in an image, e.g.,
 - pose (position, size, orientation), deformation, velocity, hue, texture
 - One special property is existence of an instantiated entity in image
- Representing existence of entity
 - Is by a separate logistic unit
 - Whose output is probability that entity exists
 - Length of vector of instantiation parameters represents entity existence
 - Its orientation represents properties of the entity
 - We ensure length of vector output of a capsule cannot exceed 1 by applying a nonlinearity that leaves orientation unchanged but scales down its magnitude

A 4-D capsule

Images with varying orientations, size, and stroke width



A 4-D capsule with 4 neurons can capture
1,2. Magnitude and Direction
3. size and
4. stroke width

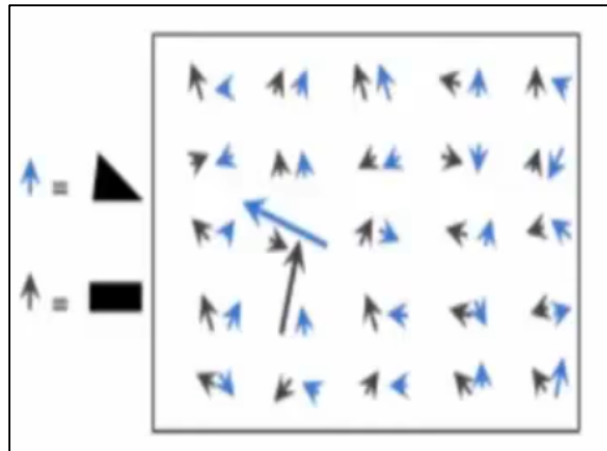
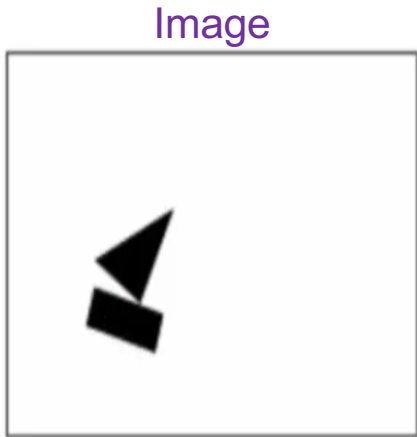
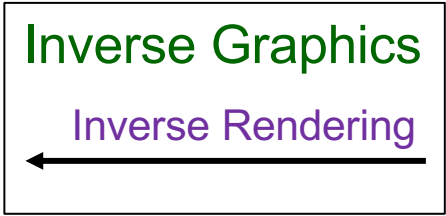
Capsule Networks perform inverse Computer Graphics

Computer graphics generates images based on descriptions

Instantiation Parameters

Rectangle
x=20
y=30
angle=16°

Triangle
x=24
y=25
angle=-65°



There is a capsule present at each location in the input image. Each capsule has two vectors corresponding to the triangle (blue) and rectangle (grey).

Pooling and Equivariance

- Pooling in a CNN is a very crude approach
 - In reality it removes all sorts of positional invariance
 - Leads to detecting right image in Figure 1 as a correct ship

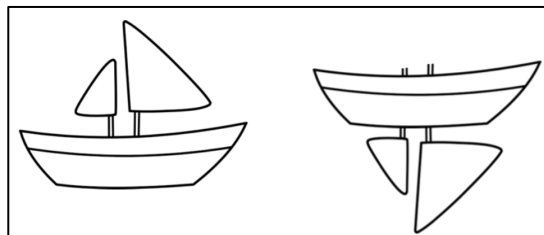


Fig 1. Disfiguration Transformation

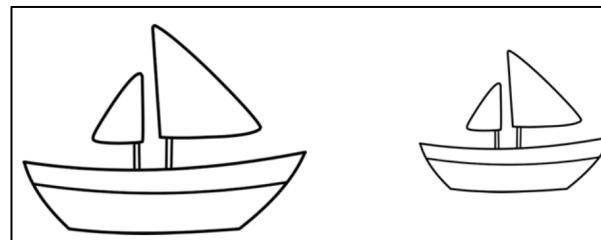


Fig. 2. Proportional Transformation

- Equivariance makes network understand the rotation or proportion change and adapt itself accordingly so that the spatial positioning inside an image is not lost.

Example of equivariance with Capsnet



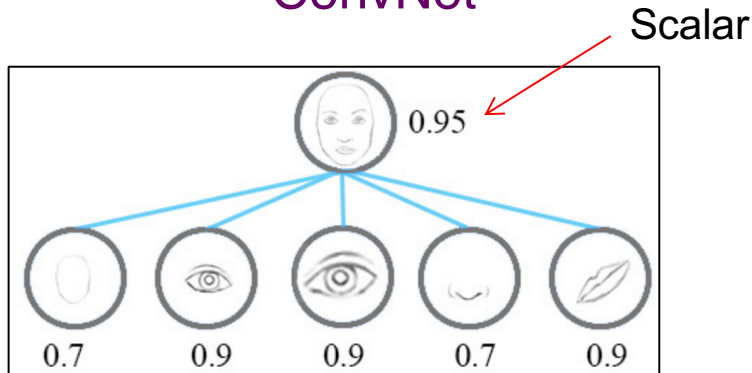
Picasso's "Portrait of woman in d'hermine pass"

Input:

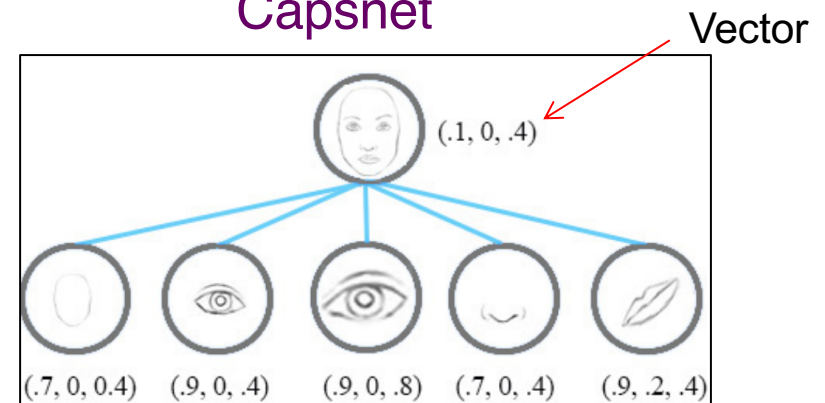


- CNN extracts features correctly in lower layer, but wrongly activates neuron in layer above for face detection

ConvNet



Capsnet



CapsNet: neuron contains likelihood as well as properties of features, e.g., a vector containing [likelihood, orientation, size]

Neuron for face detection will have much lower activation once model realizes that neurons that contribute the most signal to this parent neuron do not agree on size and orientation.

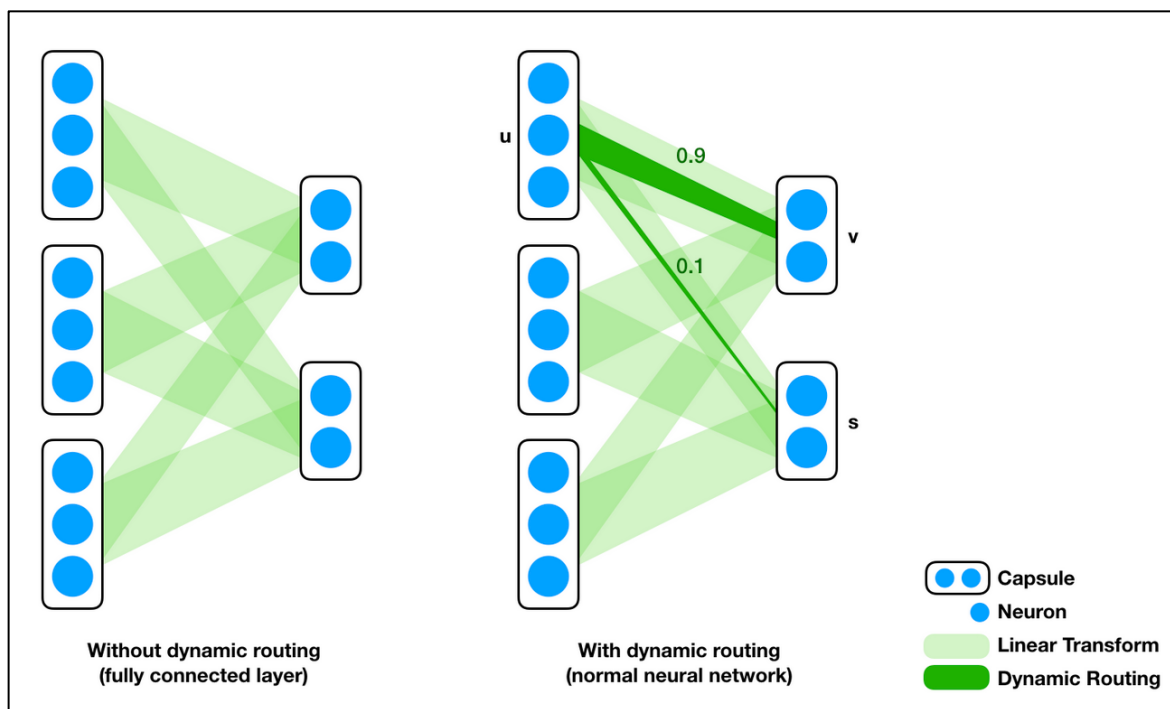
From Convolution to Capsule

- Convolutional nets are based on a simple mechanism:
 - a vision system needs to use the same knowledge at all image locations
 - This can be achieved by tying the weights of features detectors so that features learned at one location are available at other locations.
 - CNNs use translated replicas of learned feature detectors and this allows them to translate knowledge about good weight values acquired at one position in an image to other positions.
- Convolutional capsules extend the sharing of knowledge across locations to include knowledge about the part-whole relationships that characterize a familiar shape.

Dynamic Routing

- Conventional net: layers are fully connected
 - Weighting between any two neurons are trainable but once trained, these weights are *fixed*, independent of the input
- A CapsNet has these trainable weights as well
 - But additional *routing coefficients* applied to trainable wts
 - Coefficient values depend on input data (hence "dynamic")
 - Without dynamic routing, CapsNet same as regular network
- Analogous components of this dynamic routing in CNN is max pooling.
- Max pooling is a coarse way of selectively route information in the previous layer to the next
 - among a small pool of neurons, only the maximum get routed to the next layer

Coefficients operate on capsule level



The contribution of a capsule to another capsule in the next layer is determined by the weight matrix and a single scalar coefficient

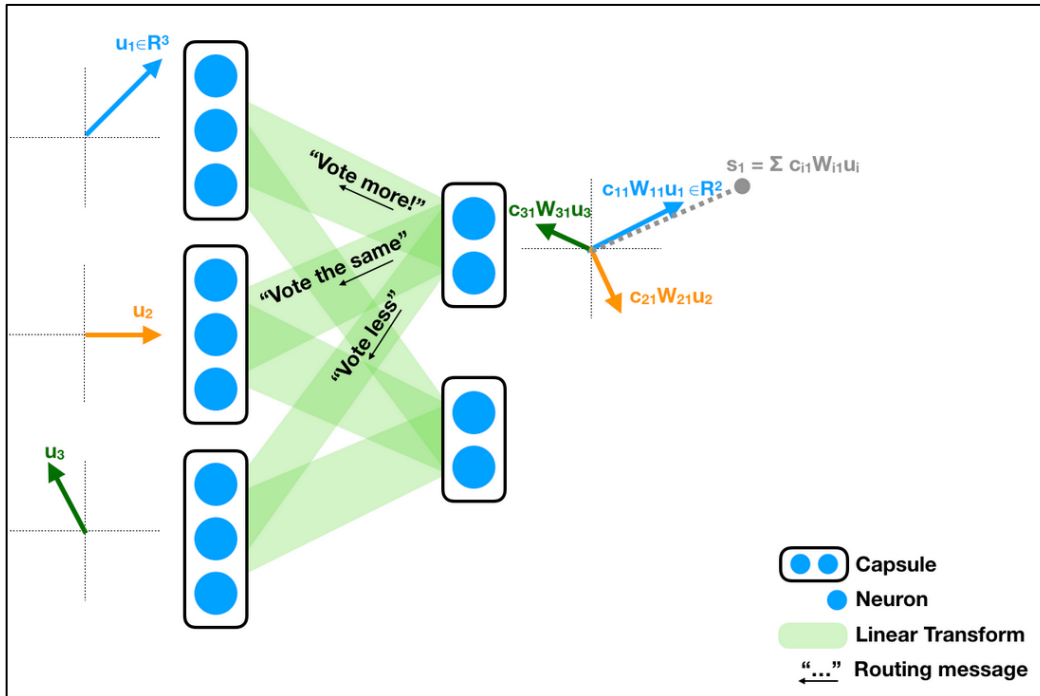
The first capsule $\mathbf{u} \in \mathbb{R}^3$ contributes to the capsule $\mathbf{s} \in \mathbb{R}^2$ in the next layer by $0.1 \cdot \mathbf{W}_{1,2} \cdot \mathbf{u}$ where 0.1 is the routing coefficient and $\mathbf{W}_{1,2}$ is a 2×3 matrix whose subscript denotes the indices of capsules in the two layers.

The total contribution of a capsule is exactly 1.

Capsule \mathbf{u} route to capsule \mathbf{v} with routing coefficient 0.9 and to capsule \mathbf{s} with 0.1.

This is a voting scheme where each capsule in layer before vote for each capsule in layer after. 24

Dynamic Routing: Routing by Agreement



Updating the coefficients.

Idea: a capsule votes more if its transformation agrees with the majority of other capsules

Shows routing coefficients to the first capsule in second layer denoted by c_{i1} for capsule u_i

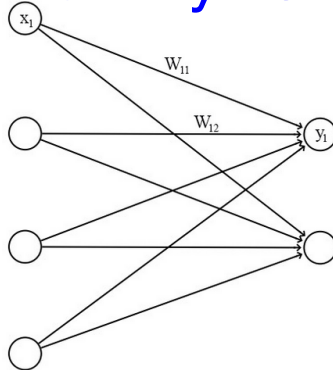
In the beginning all c_{i1} initialized to $\frac{1}{2}$ since each capsule routes to all two capsules in the next layer

The activation of the first capsule s_1 in the next layer is the sum of all transformed and scaled vectors $c_{i1}W_{i1}u_i$ from the left layer

The amount of change in coefficient (i.e. vote) is determined by the dot product between s_1 and each summand $c_{i1}W_{i1}u_i$

Comparison to fully connected neural network

For a fully connected neural network:



$$z_j = \sum_i W_{ij} x_i$$

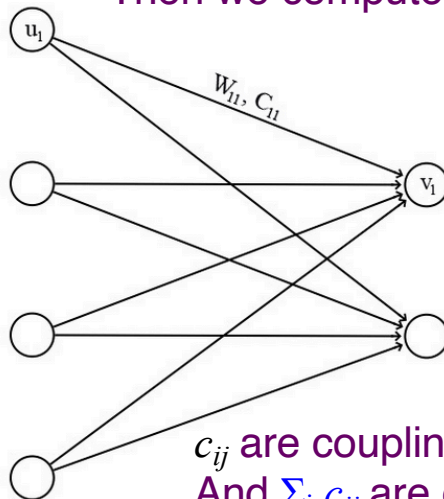
$$y_j = \text{ReLU}(z_j) \quad W_{ij}, z_j \text{ and } y_i \text{ are all scalars}$$

For a capsule: input u_i and the output v_j of a capsule are vectors.

We apply a transformation matrix W_{ij} to the capsule output u_i of the previous layer.

For example, if u_i is a k -D vector, we can apply a $m \times k$ matrix to transform it to a m -D v_i .

Then we compute a weighted sum s_j (with weights c_{ij}) for $\hat{u}_{j|i}$ from the previous layer.



$$\hat{u}_{j|i} = W_{ij} u_i$$

$$s_j = \sum_i c_{ij} \hat{u}_{j|i}$$

c_{ij} are coupling coefficients that are trained by the iterative dynamic routing process
And $\sum_j c_{ij}$ are designed to sum to 1

Capsule vs Traditional Neuron

		capsule	vs.	traditional neuron
Input from low-level neuron/capsule		vector(u_i)		scalar(x_i)
Operation	Affine Transformation	$\hat{u}_{ji} = W_{ij} u_i$ (Eq. 2)		—
	Weighting	$s_j = \sum_i c_{ij} \hat{u}_{ji}$ (Eq. 2)		$a_j = \sum_{i=1}^3 W_i x_i + b$
	Sum			
	Non-linearity activation fun	$v_j = \frac{\ s_j\ ^2}{1 + \ s_j\ ^2} \frac{s_j}{\ s_j\ }$ (Eq. 1)		$h_{w,b}(x) = f(a_j)$
output		vector(v_j)		scalar(h)
		<p>$f(\cdot)$: sigmoid, tanh, ReLU, etc.</p>		

- Vector-in, vector-out vs Scalar-in, scalar-out

Training Advantages of Capsule Nets

- Capsules bonus
 - Achieving good accuracy with far less training data.
- *Equivariance* is the detection of objects that can transform to each other.
 - A capsule detects the face is rotated right 20° (or rotated left 20°) rather than realizes the face matched a variant that is rotated right 20° .
 - Thus, since a capsule has vector information such as orientation, size, etc, it can use that information to learn a richer representation of the features as it is trained.
 - It doesn't need 50 examples of the same rotated dog; it just needs one with a vector representation which can easily be transformed.
 - By forcing the model to learn the feature variant in a capsule, we *may* extrapolate possible variants more effectively with less training data.

Length of Vector and Squashing

- Length of the output vector of a capsule represents the probability that the entity represented by the capsule is present in the current input.
- Therefore use a non-linear "squashing" function to ensure that short vectors get shrunk to almost zero length and long vectors get shrunk to a length slightly below 1
 - The squashing function gets applied to the vector output of each capsule
 - Discriminative learning makes use of this non-linearity

Computing vector output of a capsule

- Instead of applying a ReLU function, we apply a squashing function
- Vector output of a capsule (activity vector):

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1 + \|\mathbf{s}_j\|^2} \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$$

- where \mathbf{v}_j is vector output of capsule j and \mathbf{s}_j is its total input.
- It shrinks small vectors to zero and long vectors to unit vectors.

$$\begin{aligned} \mathbf{v}_j &\approx \|\mathbf{s}_j\| \mathbf{s}_j && \text{for } \mathbf{s}_j \text{ is short} \\ \mathbf{v}_j &\approx \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|} && \text{for } \mathbf{s}_j \text{ is long} \end{aligned}$$

Computing vector inputs of a capsule

- For all but the first layer of capsules, the total input to a capsule \mathbf{s}_j is a weighted sum over all *prediction vectors* $\hat{\mathbf{u}}_{j|i}$ from the capsules in the layer below
 - produced by multiplying the output \mathbf{u}_i of a capsule in the layer below by a weight matrix \mathbf{W}_{ij}

$$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}, \quad \hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij} \mathbf{u}_i$$

- where the c_{ij} are coupling coefficients that are determined by the iterative dynamic routing process

Computing Coupling Coefficients

- Coupling coefficients between capsule i and all the capsules in the layer above sum to 1
- They are determined by a “routing softmax”
- whose initial logits b_{ij} are the log prior probabilities that capsule i should be coupled to capsule j

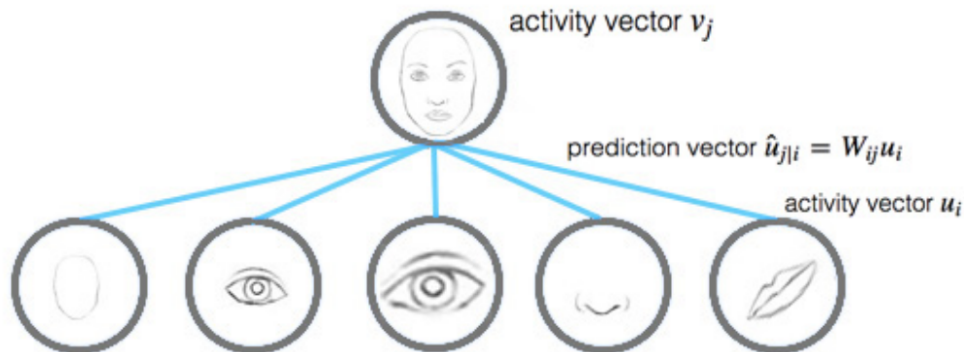
$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$$

Logit is inverse of sigmoid $\sigma(a)$
i.e., $a = \ln(\sigma / (1 - \sigma))$

Capsule choosing a parent

- In deep learning, we use backpropagation to train model parameters. The transformation matrix W_{ij} in capsules are trained with backpropagation
- Nevertheless, the coupling coefficients c_{ij} are calculated with an iterative dynamic routing method
- Using an iterative process each active capsule will choose a capsule in the layer above to be its parent
 - For higher levels of a visual system, this iterative process will be solving the problem of assigning parts to wholes

Iterative Dynamic Routing



- Coupling coefficients c_{ij} determines the relevancy of a capsule in activating another capsule in the next layer.
- After applying a transformation matrix W_{ij} to previous capsule output u_i we compute the *prediction vector* $\hat{u}_{j|i}$

$$\hat{u}_{j|i} = W_{ij}u_i$$

- Which u_i is the activity vector for the capsule i in the layer below

Computing Similarity for Coupling

- Similarity of prediction vector with capsule output v_j is related to relevancy of u_i in activating v_i .
 - i.e., contribution of a capsule output to the next layer capsule output after transformation of its properties.
 - Higher similarity implies larger coupling coefficients c_{ij}
 - Such similarity is measured using the scalar product and we adjust a relevancy score b_{ij} according to the similarity

$$b_{ij} \leftarrow \hat{u}_{j|i} \cdot v_j$$

- Coupling coefficients c_{ij} are computed by softmax of b_{ij}

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

- Moreover, b_{ij} is updated iteratively in multiple iterations
 - Typically in 3 iterations

Routing output of a capsule to parent above

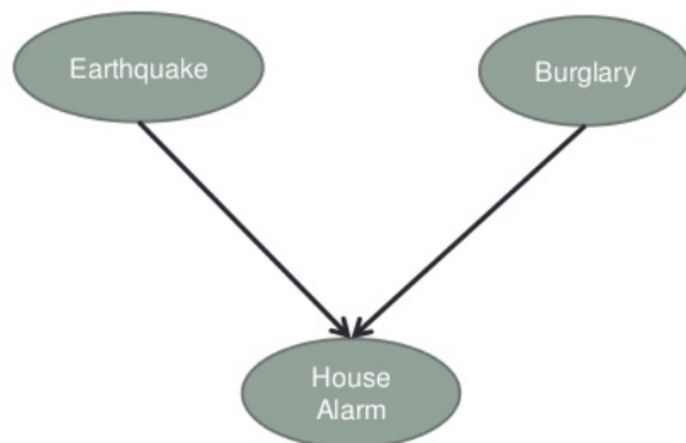
- Output of a capsule is a vector
 - Which makes it possible to use a powerful dynamic routing mechanism to ensure the output of a capsule gets sent to an appropriate parent in the layer above
- Initially the output is routed to all possible parents but it is scaled down by coupling coefficients that sum to 1
- For each possible parent, capsule computes a “prediction vector” by multiplying its own output by a weight matrix

Routing by agreement

- If the prediction vector has a large scalar product then it increases the coupling coefficient for that parent and decreasing it for other parents
- This increases the contribution that the capsule makes to that parent thus further increasing the scalar product of the capsule's prediction with the parent's output
- This type of routing by agreement is more effective than primitive routing of max-pooling, which allows neurons in one layer to ignore all but the most active feature detector in a local pool in the layer below

Dynamic Routing as Explaining Away

- The dynamic routing mechanism is an effective way to implement the explaining away that is needed to segment highly overlapping objects
 - Explaining away is a common pattern of reasoning in which the confirmation of one cause of an observed or believed event reduces the need to invoke alternative causes
 - Observed in a V-structure in a Bayesian Network



$$p(e^1|h^1) > p(e^1|h^1, b^1)$$

Routing Algorithm

Procedure 1 Routing algorithm.

```

1: procedure ROUTING( $\hat{\mathbf{u}}_{j|i}, r, l$ )
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  $l$ :  $\mathbf{c}_i \leftarrow \text{softmax}(\mathbf{b}_i)$  ▷ softmax computes Eq. 3
5:     for all capsule  $j$  in layer  $(l + 1)$ :  $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$ 
6:     for all capsule  $j$  in layer  $(l + 1)$ :  $\mathbf{v}_j \leftarrow \text{squash}(\mathbf{s}_j)$  ▷ squash computes Eq. 1
7:     for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$ 
   return  $\mathbf{v}_j$ 

```

Eq 3 softmax:

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

Eq 1 squash:

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1 + \|\mathbf{s}_j\|^2} \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$$

Capsules capture part-whole relationships

- Viewpoint changes have complicated effects on pixel intensities but simple, linear effects on the pose matrix that represents the relationship between an object or object-part and the viewer.
- The aim of capsules is to make good use of this underlying linearity, both for dealing with viewpoint variation and improving segmentation decisions

Capsule Network (CapsNet) Architecture

- The network has modules called “capsules”.
- These capsules are particularly good at handling different types of visual stimulus and encoding things like pose (position, size, orientation), deformation, velocity, albedo, hue, texture etc.
- The network must have a mechanism for “routing” low level visual information to what it believes is the best capsule for handling it.

From ConvNet to CapsNet

1. Replace scalar-output feature detectors of CNNs with vector-output capsules
2. Replace max-pooling with routing-by-agreement,
3. We still like to replicate learned knowledge across space, so we make all but the last layer of capsules be convolutional.
4. As with CNNs, we make higher-level capsules cover larger regions of the image, but unlike max-pooling we do not throw away information about the precise position of the entity within the region.

Routing by Agreement

- *Routing by agreement* method is superior than the current mechanism like max-pooling
- Max pooling routes based on the strongest feature detected in the lower layer

Margin Loss for Digit Existence

- The length of the instantiation vector represents the probability that a capsule's entity exists, so we need top-level capsule for digit class k to have a long instantiation vector iff that digit is present in the image.

- To allow for multiple digits, we use a separate margin loss, L_k for each digit capsule, k :

$$L_c = T_c \max(0, m^+ - \|\mathbf{v}_c\|)^2 + \lambda (1 - T_c) \max(0, \|\mathbf{v}_c\| - m^-)^2$$

- where $T_c = 1$ iff a digit of class c is present.

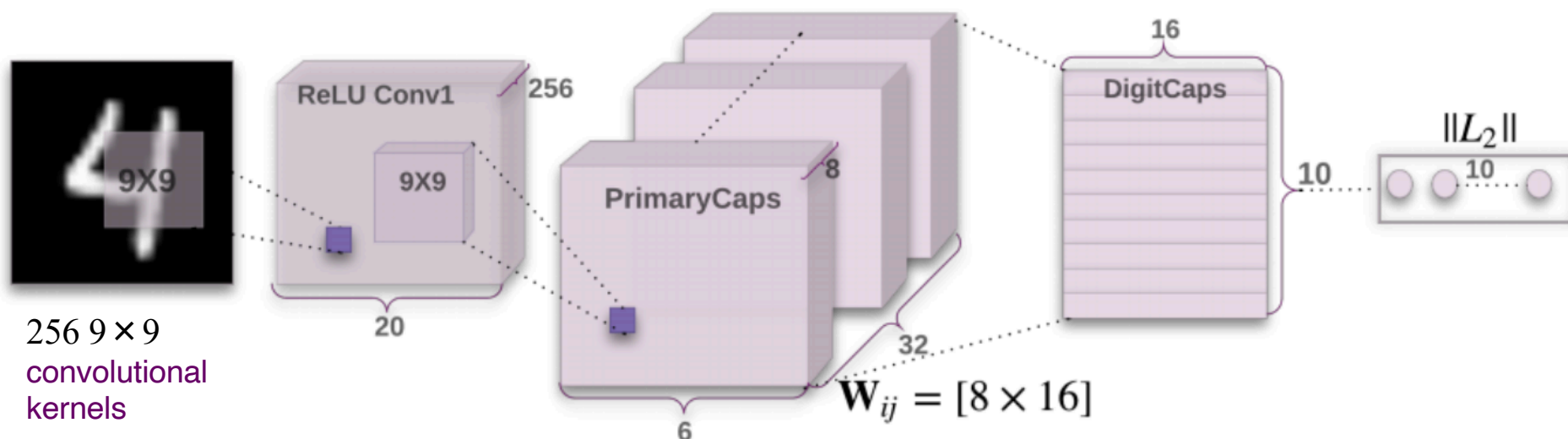
and $m^+ = 0.9$ and $m^- = 0.1$. The λ down-weighting of the loss for absent digit classes stops the initial learning from shrinking the lengths of the activity vectors of all the digit capsules. Suggest $\lambda = 0.5$.

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

CapsNet Architecture

- A simple CapsNet architecture for MNIST shown next
- The architecture is shallow with only two convolutional layers and one fully connected layer.
 - Module labeled as ReLU Conv1 has 256, 9×9 convolution kernels with a stride of 1 and ReLU activation.
 - This layer converts pixel intensities to the activities of local feature detectors that are then used as inputs to the *primary* capsules

A simple CapsNet with 3 layers

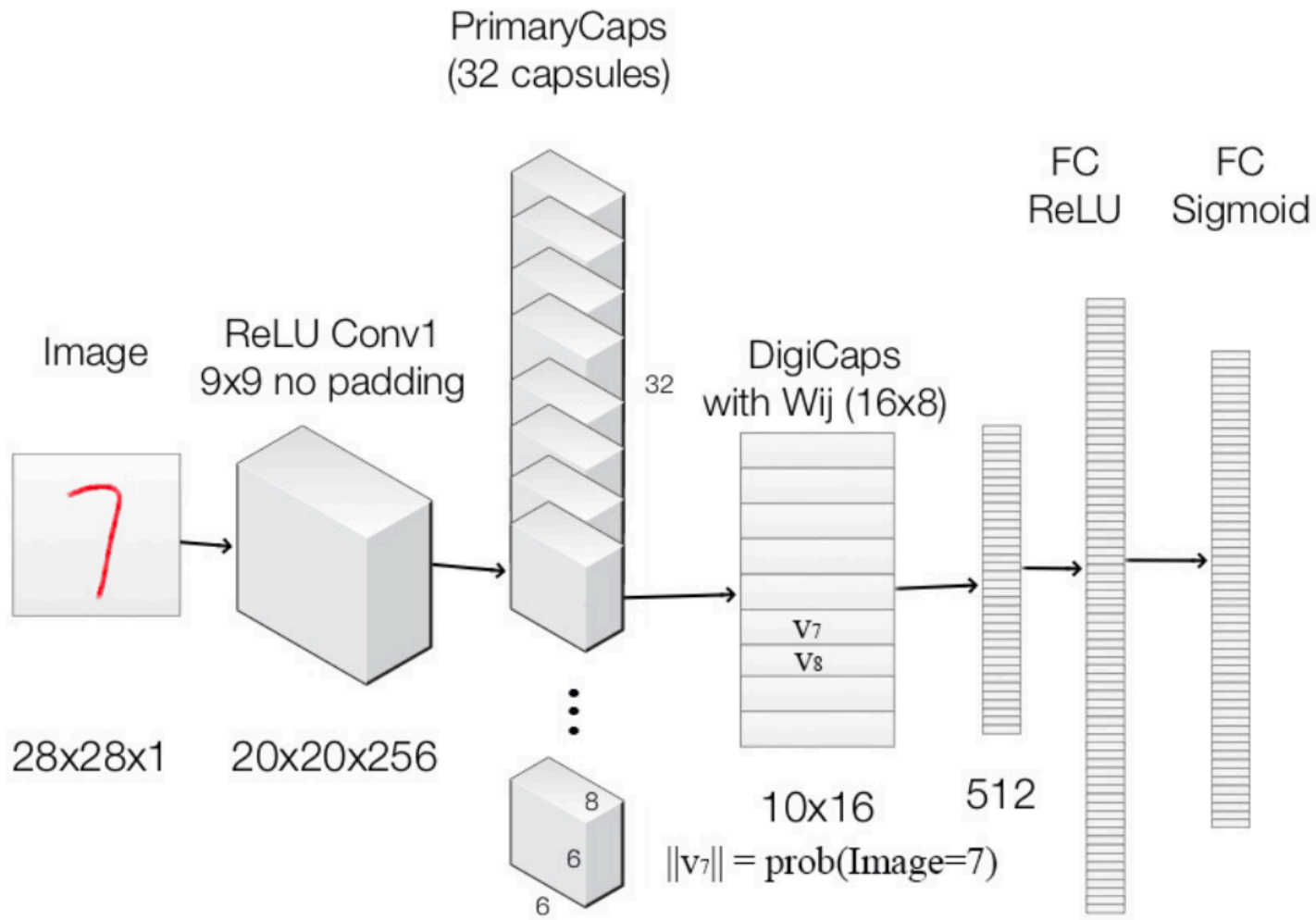


Length of activity vector of each capsule in DigitCapsules layer indicates presence of an instance of each class and is used to calculate classification loss.

W_{ij} is a weight matrix between each $\mathbf{u}_i, i \in (1, 32 \times 6 \times 6)$ in PrimaryCapsules and $\mathbf{v}_j, j \in (1, 10)$.

This model gives comparable results to deep convolutional networks

CapsNet Architecture for MNIST



Primary Capsules

- The primary capsules are the lowest level of multi-dimensional entities and, from an inverse graphics perspective, activating the primary capsules corresponds to inverting the rendering process.
- This is a very different type of computation than piecing instantiated parts together to make familiar wholes, which is what capsules are designed to be good at.

- 2nd layer (PrimaryCapsules) is a convolutional capsule layer with 32 channels of convolutional 8D capsules
 - i.e. each primary capsule contains 8 convolutional units with a 9×9 kernel and a stride of 2
 - Each primary capsule output sees the outputs of all 25681 Conv1 units whose receptive fields overlap with the location of the center of the capsule.
 - In total PrimaryCapsules has [32; 6; 6] capsule outputs (each output is an 8D vector) and each capsule in the [6; 6] grid is sharing their weights with each other.
 - One can see PrimaryCapsules as a Convolution layer with Eq. 1 as its block non-linearity.
 - The final Layer (DigitCaps) has one 16D capsule per digit class and each of these capsules receives input from all capsules in layer below.

Implementation in Tensorflow

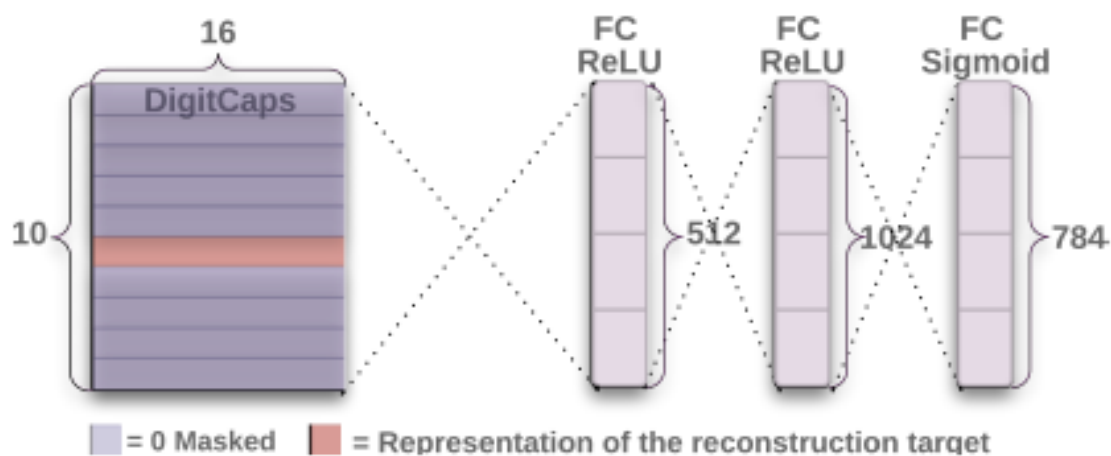
- We have routing only between two consecutive capsule layers (e.g. PrimaryCapsules and DigitCaps).
- Since Conv1 output is 1D there is no orientation in its space to agree on. Therefore, no routing is used between Conv1 and PrimaryCapsules.
- All the routing logits (b_{ij}) are initialized to zero.
- Therefore, initially a capsule output (u_i) is sent to all parent capsules (v_0, \dots, v_{10}) with equal probability (c_{ij}).
- Implementation is with Adam optimizer with TensorFlow default parameters, exponentially decaying learning rate, to minimize the sum of the margin losses in $L_c = T_c \max(0, m^+ - \|\mathbf{v}_c\|)^2 + \lambda (1 - T_c) \max(0, \|\mathbf{v}_c\| - m^-)^2$

Reconstruction as a Regularization Method

- Additional reconstruction loss used to encourage digit capsules to encode instantiation parameters of input digit.
 - During training, mask out all but activity vector of correct digit capsule.
 - Then use this activity vector to reconstruct.
 - The output of the digit capsule is fed into a decoder consisting of 3 fully connected layers that model the pixel intensities as described next
 - Minimize sum of squared differences between outputs of logistic units and the pixel intensities.
 - Scale down reconstruction loss by 0.0005 so that it does not dominate the margin loss during training.
 - As illustrated reconstructions from the 16D output of the CapsNet are robust while keeping only important details.













Decoder Structure

- Decoder structure to reconstruct a digit from the DigitCaps layer representation.
 - The Euclidean distance between the image and the output of the Sigmoid layer is minimized during training
 - The true label is used as reconstruction target during training



Capsules on MNIST

- Training on 28 28 MNIST images that have been shifted by up to 2 pixels in each direction with zero padding. No other data augmentation/deformation is used. Dataset has 60K and 10K images for training and testing respectively
- Sample MNIST test reconstructions of a CapsNet with 3 routing iterations. $(l; p; r)$ represents label, prediction and reconstruction target
 - Two rightmost columns show two reconstructions of a failure example and explains how model confuses a 5 and a 3. Other columns are from correct classifications and shows that model picks on the details while smoothing the noise.

(l, p, r)	(2, 2, 2)	(5, 5, 5)	(8, 8, 8)	(9, 9, 9)	(5, 3, 5)	(5, 3, 3)
Input						
Output						

Detecting overlapping digits

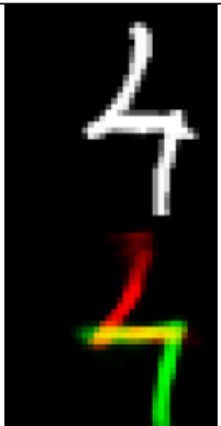


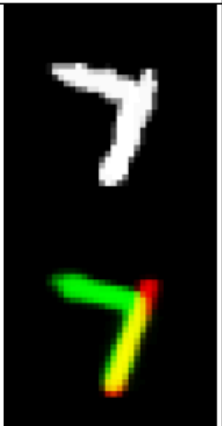

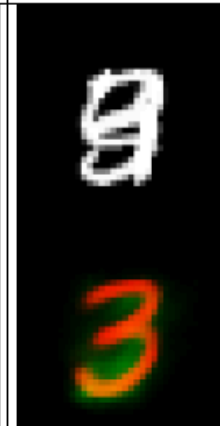



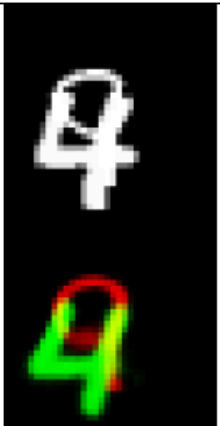
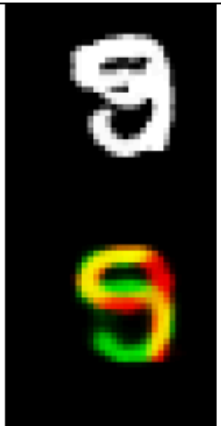
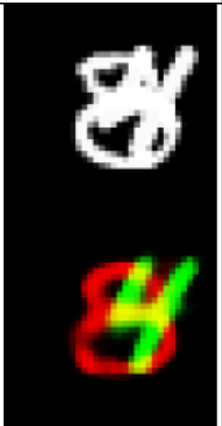
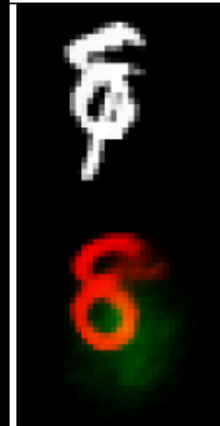
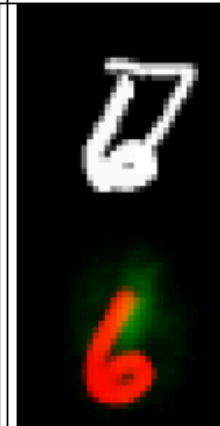
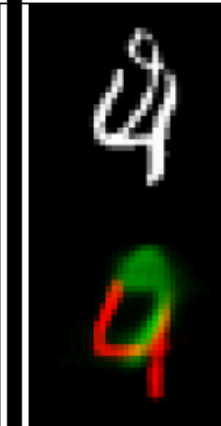

- There is a short coming using the max pool in CNN. In max pool, we only keep the most dominating (max) features.
- Capsules maintain a weighted sum of features from the previous layer.
- Hence, it is more suitable in detecting overlapping features.
 - For example detecting multiple overlapping digits in handwriting



Multi MNIST data set

- Generate the MultiMNIST training and test dataset by overlaying a digit on top of another digit from the same set (training or test) but different class.
- Each digit is shifted up to 4 pixels in each direction resulting in a 3636 image. Considering a digit in a 2828 image is bounded in a 2020 box, two digits bounding boxes on average have 80% overlap.
- For each digit in the MNIST dataset we generate 1K MultiMNIST examples.
- So the training set size is 60M and the test set size is 10M.

Reconstructions of MultiMNIST

R:(2, 7) L:(2, 7)	R:(6, 0) L:(6, 0)	R:(6, 8) L:(6, 8)	R:(7, 1) L:(7, 1)	*R:(5, 7) L:(5, 0)	*R:(2, 3) L:(4, 3)	R:(2, 8) L:(2, 8)	R:P:(2, 7) L:(2, 8)
							
R:(8, 7) L:(8, 7)	R:(9, 4) L:(9, 4)	R:(9, 5) L:(9, 5)	R:(8, 4) L:(8, 4)	*R:(0, 8) L:(1, 8)	*R:(1, 6) L:(7, 6)	R:(4, 9) L:(4, 9)	R:P:(4, 0) L:(4, 9)
							

References

1. S. Sabour, N. Frosst and G. Hinton, *Dynamic routing between capsules*, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA
2. J. Hui, <https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/>
3. Debarko De, <https://hackernoon.com/what-is-a-capsnet-or-capsule-network-2bfbe48769cc>
4. <https://www.quora.com/What-is-it-that-Capsule-Theory-is-trying-to-achieve-in-Deep-Learning>