Capsule Network

Room 713 CT Hsieh

Reference

- Paper
  - Transforming Auto-encoders
  - Dynamic Routing Between Capsules
  - Matrix Capsules with EM Routing
- Video
  - Geoffrey Hinton “Does the Brain do Inverse Graphics”
  - Geoffrey Hinton talk What is wrong with CNN
  - Capsule Networks Tutorial
  - How to implement CapsNets using TensorFlow
  - Capsule Networks: An Improvement to Convolutional Networks
  - https://www.youtube.com/watch?v=wC0rhjvst8I
  - https://www.youtube.com/watch?v=akq6PNnkKY8

Reference

- Slide and Article
  - https://www.slideshare.net/aureliengeron/introduction-to-capsule-networks-capsnets
  - https://www.slideshare.net/charlesmartin141/capsule-networks-84754653
  - https://www.slideshare.net/ssuser73ec8f/20171113-capsnet
  - https://medium.com/ai³-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b
  - http://helper.ipam.ucla.edu/publications/gss2012/gss2012_10754.pdf
  - https://hep-ai.org/slides/2017-12-12.pdf

Reference

- Github
  - https://github.com/XifengGuo/CapsNet-Keras
  - https://github.com/bourdakos1/capsule-networks
  - https://github.com/JunYeopLee/capsule-networks
  - https://github.com/llSourcell/capsule_networks
  - https://github.com/naturomics/CapsNet-Tensorflow
  - https://github.com/Sarasra/models/tree/master/research/capsules
  - https://github.com/jhui/machine_learning/tree/master/capsule_em
  - https://github.com/www0wwwjs1/Matrix-Capsules-EM-Tensorflow
  - https://github.com/gyang274/capsulesEM

AI Need to Start Over

- Geoffrey Hinton Says AI Needs To Start Over

In an interview with Axios Hinton is credited with saying that he is

> *"deeply suspicious" of back-propagation"*

and

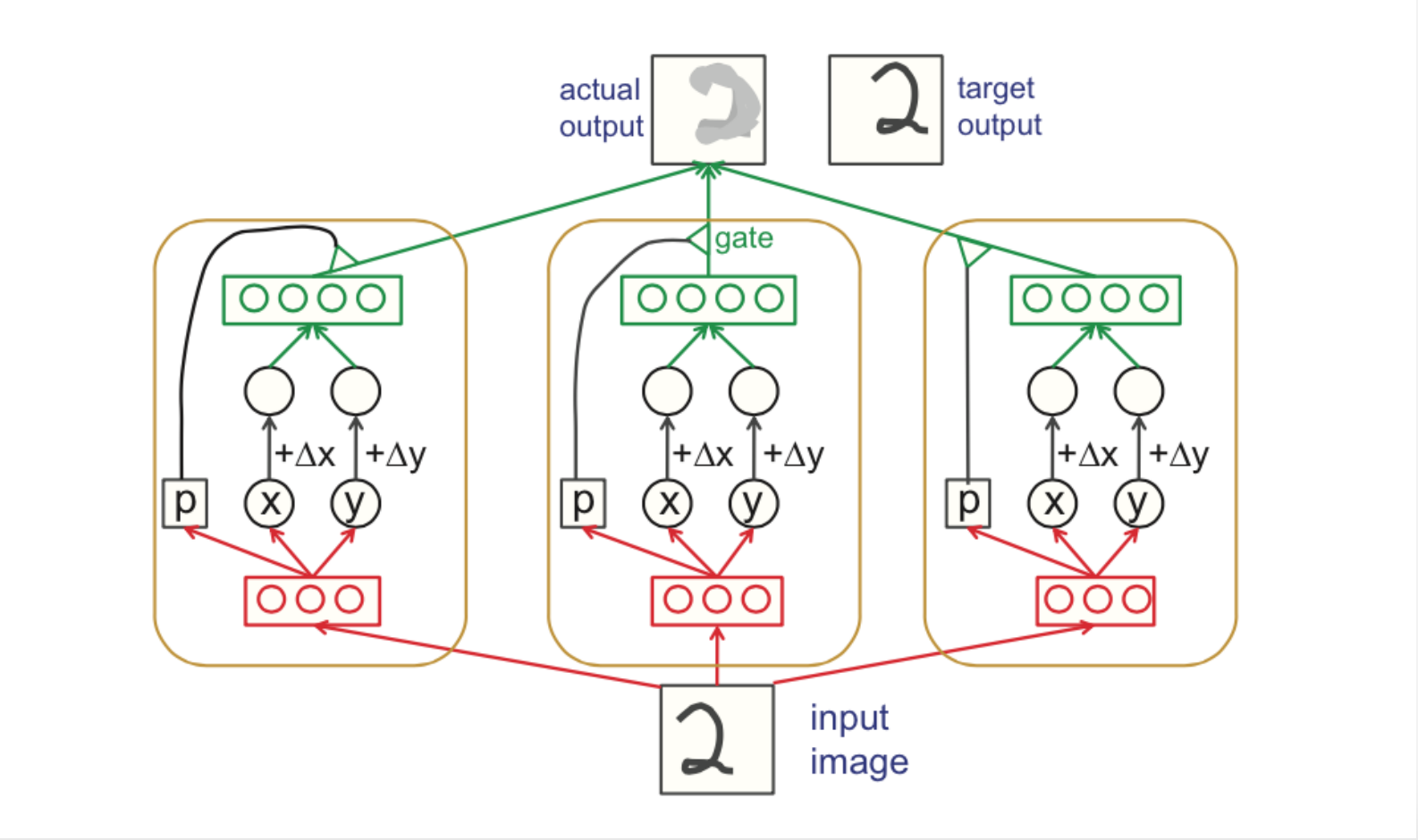
> *"My view is throw it all away and start again,"*

The worry is that neural networks don't seem to learn like we do:

> *"I don't think it's how the brain works. We clearly don't need all the labeled data."*

Transforming Auto-encoders

- Propose again in 2011 but not impressed

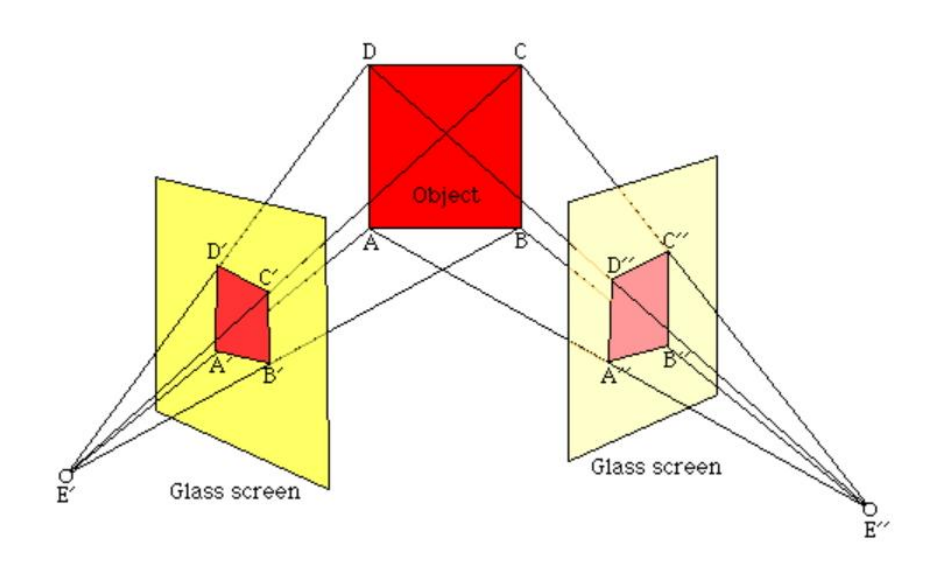


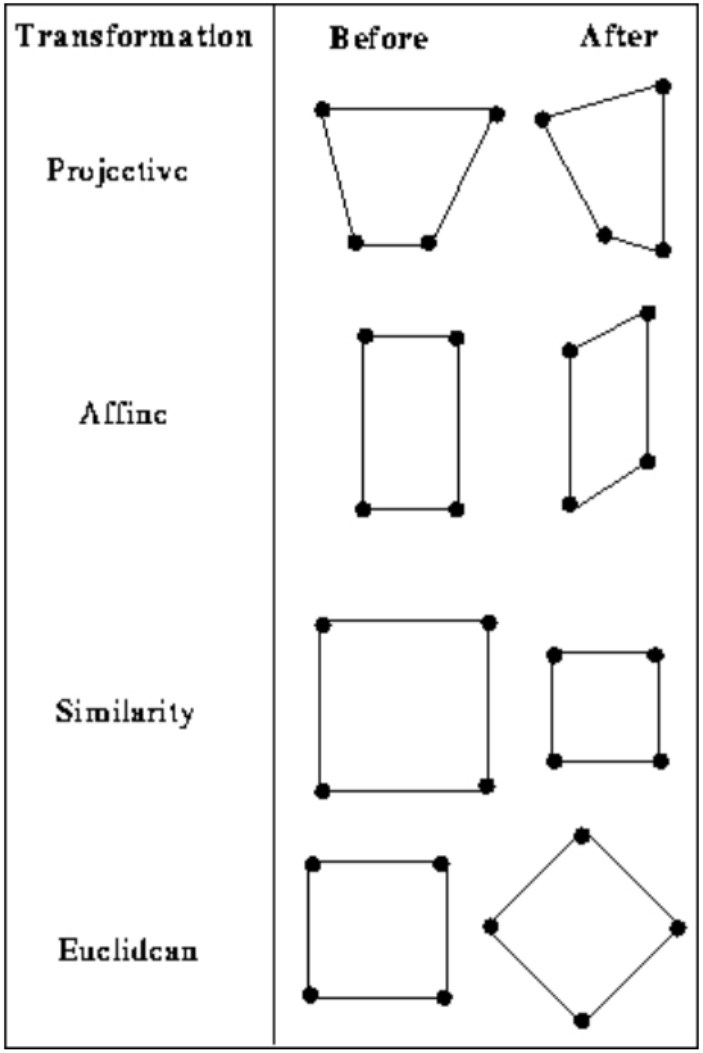
Outline

- Related Work
  - Geometric Transformation
- What's Wrong With CNN
  - Talking from Hinton
- Capsule Network
  - Idea and problem
  - MNIST
- Discuss

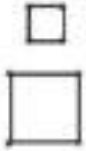
Geometric Transformation

- Perspective Geometric





Geometric Transformation

Group	Matrix	Distortion	Invariant properties
Projective 8 dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, **order of contact**: intersection (1 pt contact); tangency (2 pt contact); inflections (3 pt contact with line); tangent discontinuities and cusps. cross ratio (ratio of ratio of lengths).
Affine 6 dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on collinear or parallel lines (e.g. midpoints), linear combinations of vectors (e.g. centroids). The line at infinity, $\mathbf{l}_\infty$.
Similarity 4 dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratio of lengths, angle. The circular points, $\mathbf{I}, \mathbf{J}$ (see section 2.7.3).
Euclidean 3 dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Length, area $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

Expectation-Maximization

EM Algorithm

Input:

$\Theta = \{\theta_1, ..., \theta_p\}$ //Parameters to be Estimated

$X_{obs} = \{x_1, ..., x_k\}$ //Input Database Values Observed

$X_{miss} = \{x_{k+1}, ..., x_n\}$ //Input Database Values Missing

Output:

$\hat{\Theta}$ //Estimates for Θ

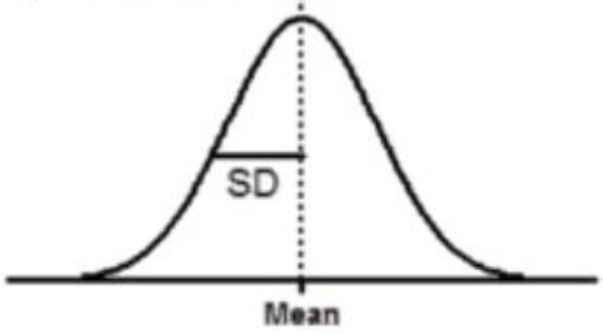
EM Algorithm:

```
i := 0;
Obtain initial parameter MLE estimate, Θ̂^i;
repeat
    Estimate missing data, X̂^i_miss;
    i++;
    Obtain next parameter estimate, θ̂^i to maximize data;
until estimate converges;
```

Gaussian Mixture Model

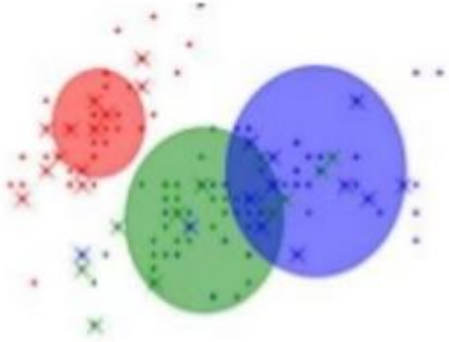
- Gaussian

"Gaussian is a characteristic symmetric "bell curve" shape that quickly falls off towards 0 (practically)"



- Mixture Model

"mixture model is a probabilistic model which assumes the underlying data to belong to a mixture distribution"



Gaussian Mixture Model

- Data with D attributes, from Gaussian sources $c_1 \ldots c_k$
  - how typical is $\mathbf{x}_i$ under source **c**

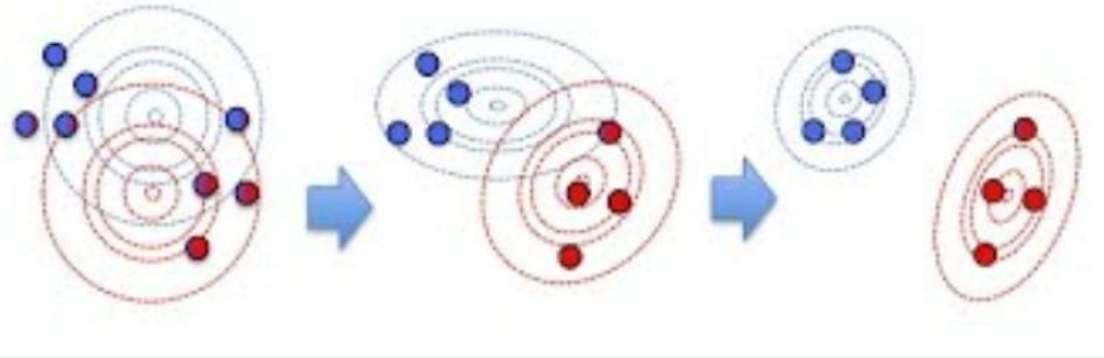
$$P(\vec{x}_i \mid c) = \frac{1}{\sqrt{2\pi|\Sigma_c|}} \exp\left\{-\tfrac{1}{2}(\vec{x}_i - \vec{\mu}_c)^T \Sigma_c^{-1} (\vec{x}_i - \vec{\mu}_c)\right\}$$

$$\sum_a \sum_b (x_{ia} - \mu_{ca}) \left[\Sigma_c^{-1}\right]_{ab} (x_{ib} - \mu_{cb})$$

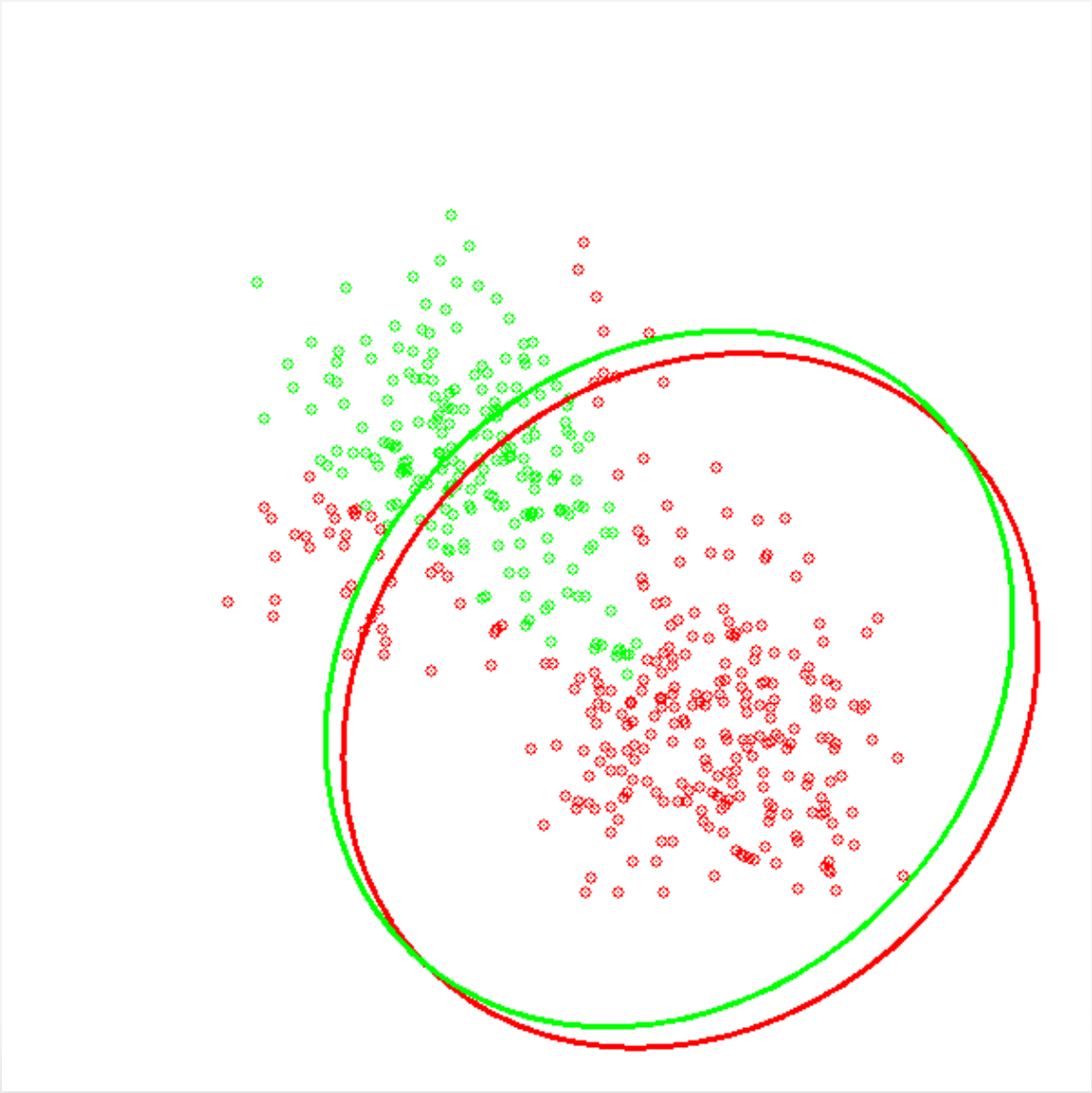
  - how likely that $\mathbf{x}_i$ came from **c**

$$P(c \mid \vec{x}_i) = \frac{P(\vec{x}_i \mid c) P(c)}{\sum_{c=1}^{k} P(\vec{x}_i \mid c) P(c)}$$

  - how important is $\mathbf{x}_i$ for source **c**: $w_{ic} = P(c \mid \vec{x}_i) \Big/ \left(P(c \mid \vec{x}_1) + \ldots + P(c \mid \vec{x}_n)\right)$
  - mean of attribute **a** in items assigned to **c**: $\mu_{ca} = w_{c1} x_{1a} + \ldots + w_{cn} x_{na}$
  - covariance of **a** and **b** in items from **c**: $\Sigma_{cab} = \sum_{i=1}^{n} w_{ci} \left(x_{ia} - \mu_{ca}\right)\left(x_{ib} - \mu_{cb}\right)$
  - prior: how many items assigned to **c**: $P(c) = \frac{1}{n}\left(P(c \mid \vec{x}_1) + \ldots + P(c \mid \vec{x}_n)\right)$

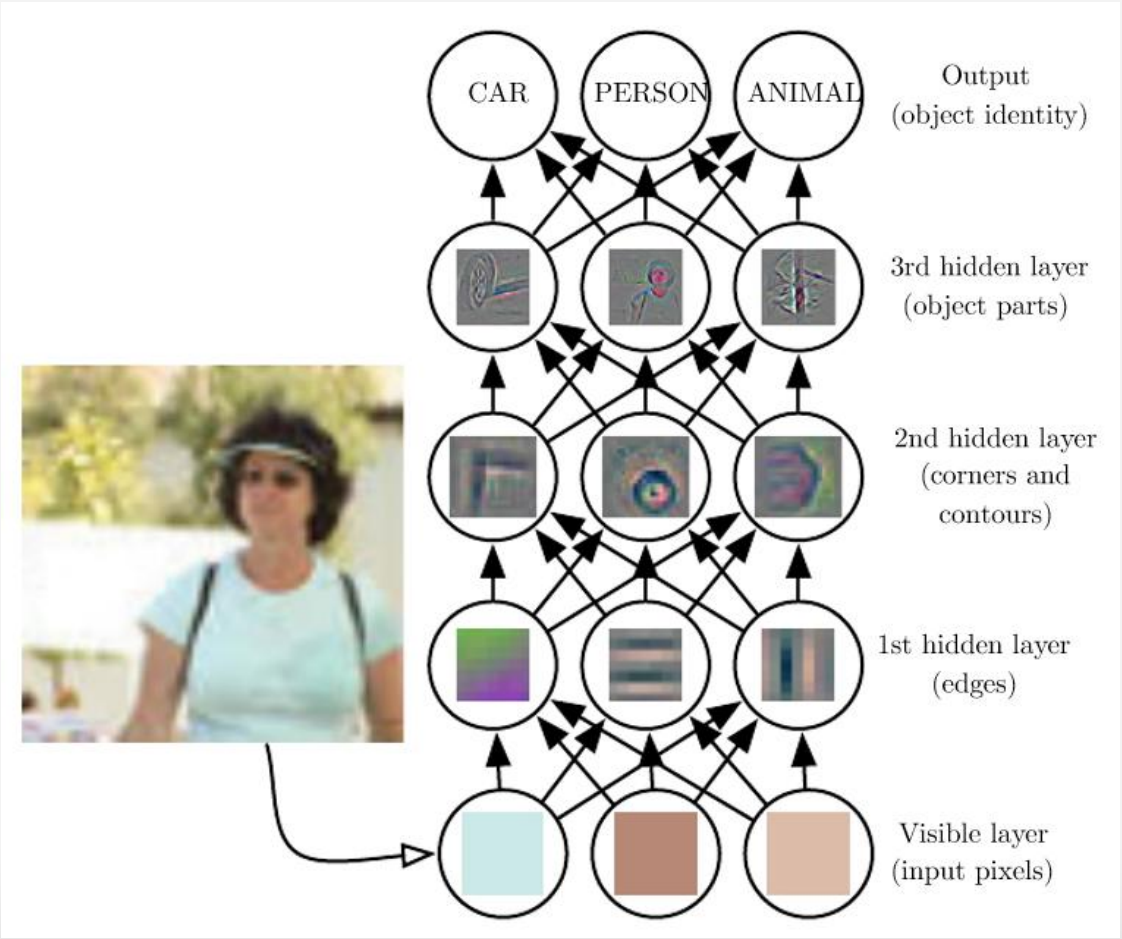


EM for GMM



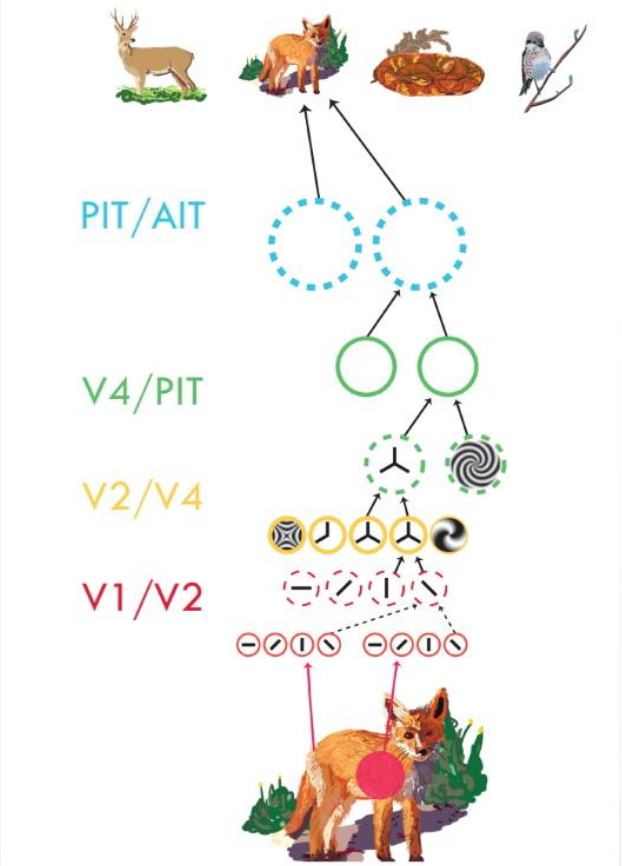
Review CNN

- Local Feature Based

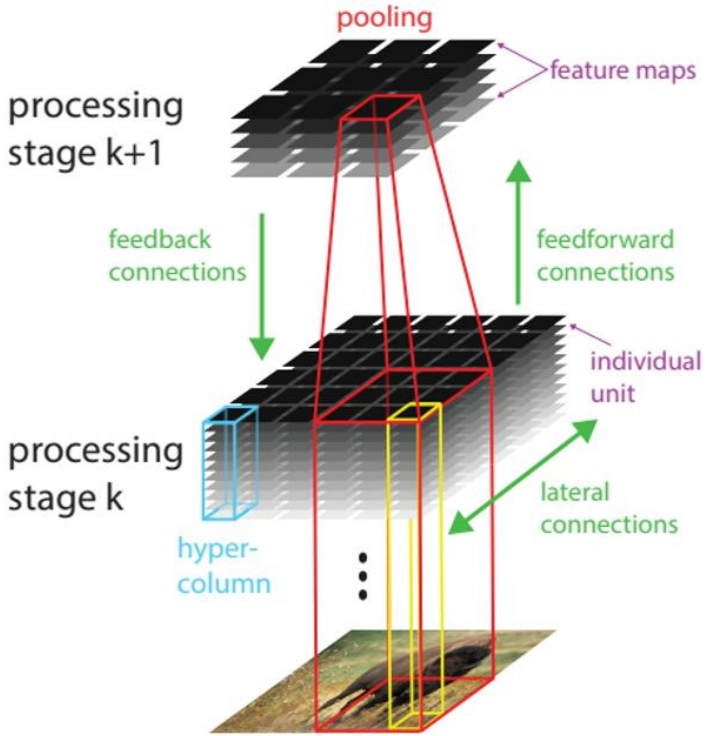


CNN

- Hierarchical Model



HMax model, Riesenhuber and Poggio (1999)

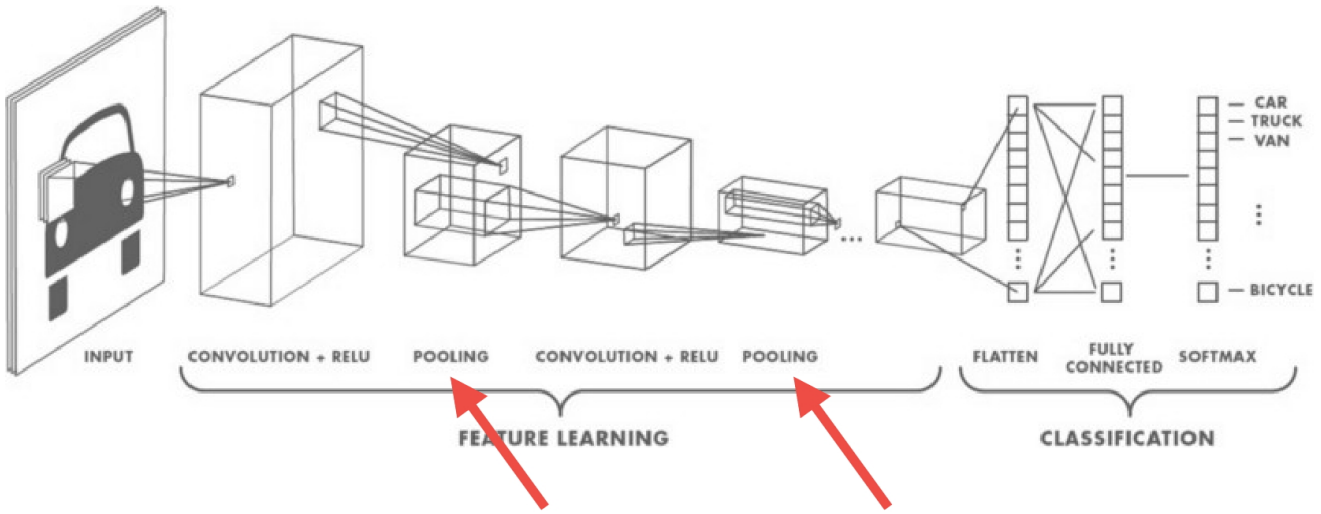


What's Wrong with CNN

- Everything is great but…..

PROBLEMS IS 'POOLING'

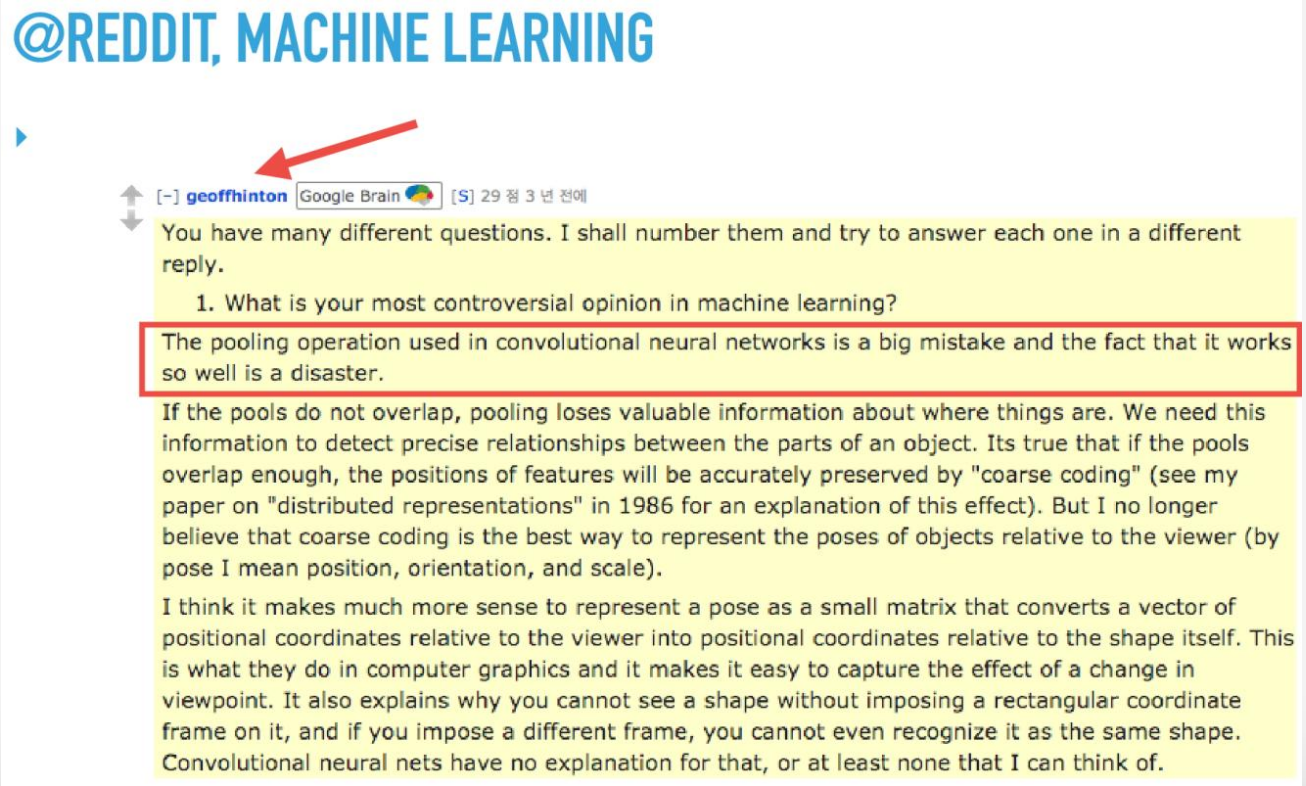
- ConvNet Architecture



Obtain translational, rotational invariance

What's Wrong with CNN

- Hinton's comment
  - https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/clyj4jv/

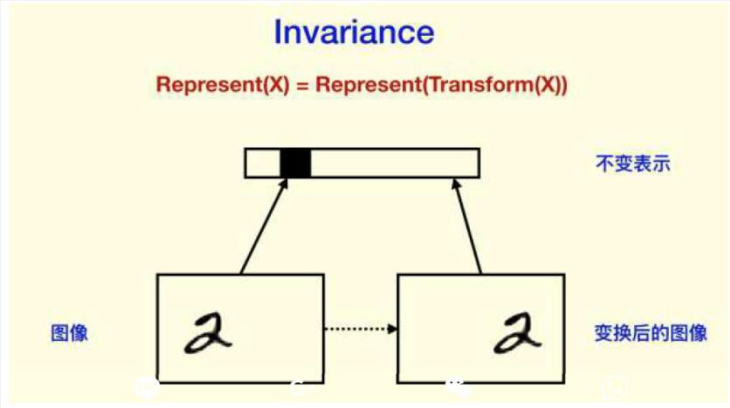


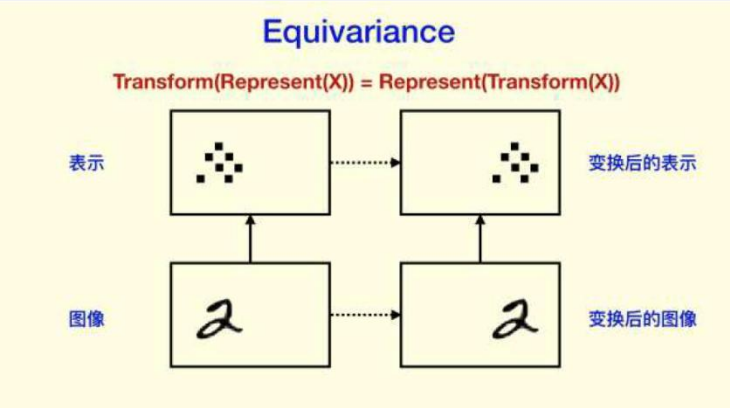
Main Problem is….

- CNN is not learn like person
  - We don’t need big data to learn
  - One shot learning / Zero shot learning is what people learn
- What’s Problem of CNN
  - What and Where
    - Where objects are in space
    - What objects are

Equivariance VS Invariance

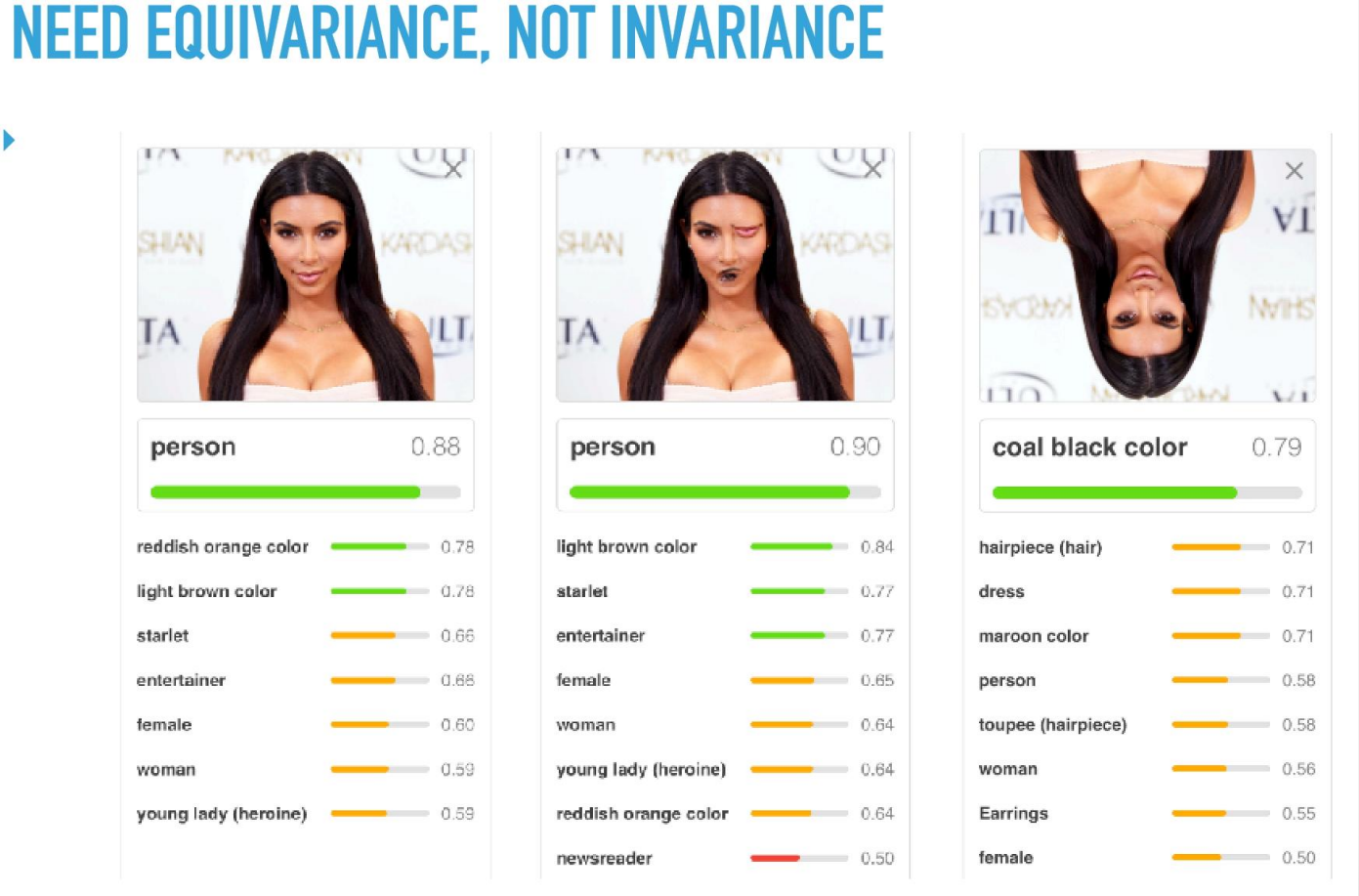
- CNN is Invariance but Hinton think we need Equivariance which is more like people do





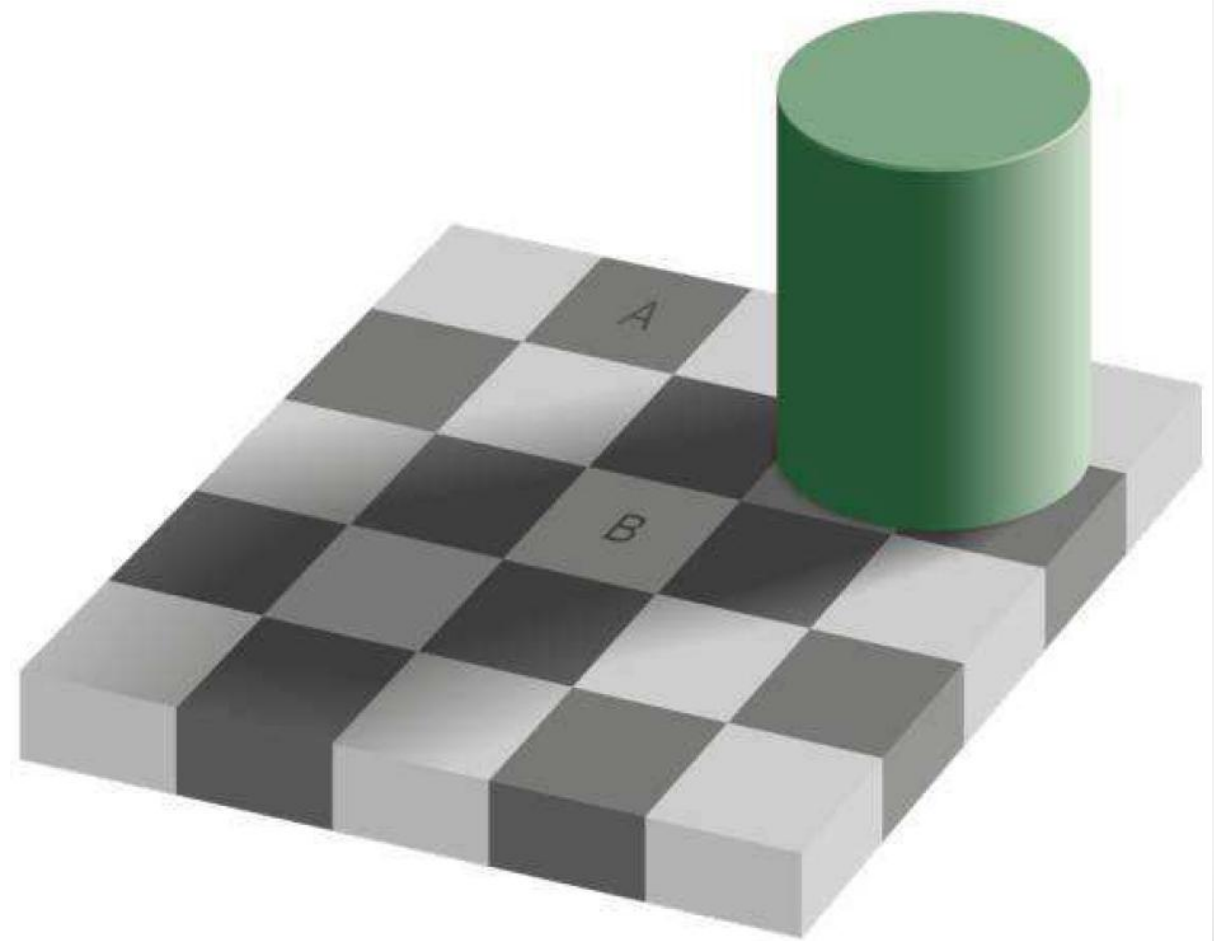
Need Equivalence

- Example



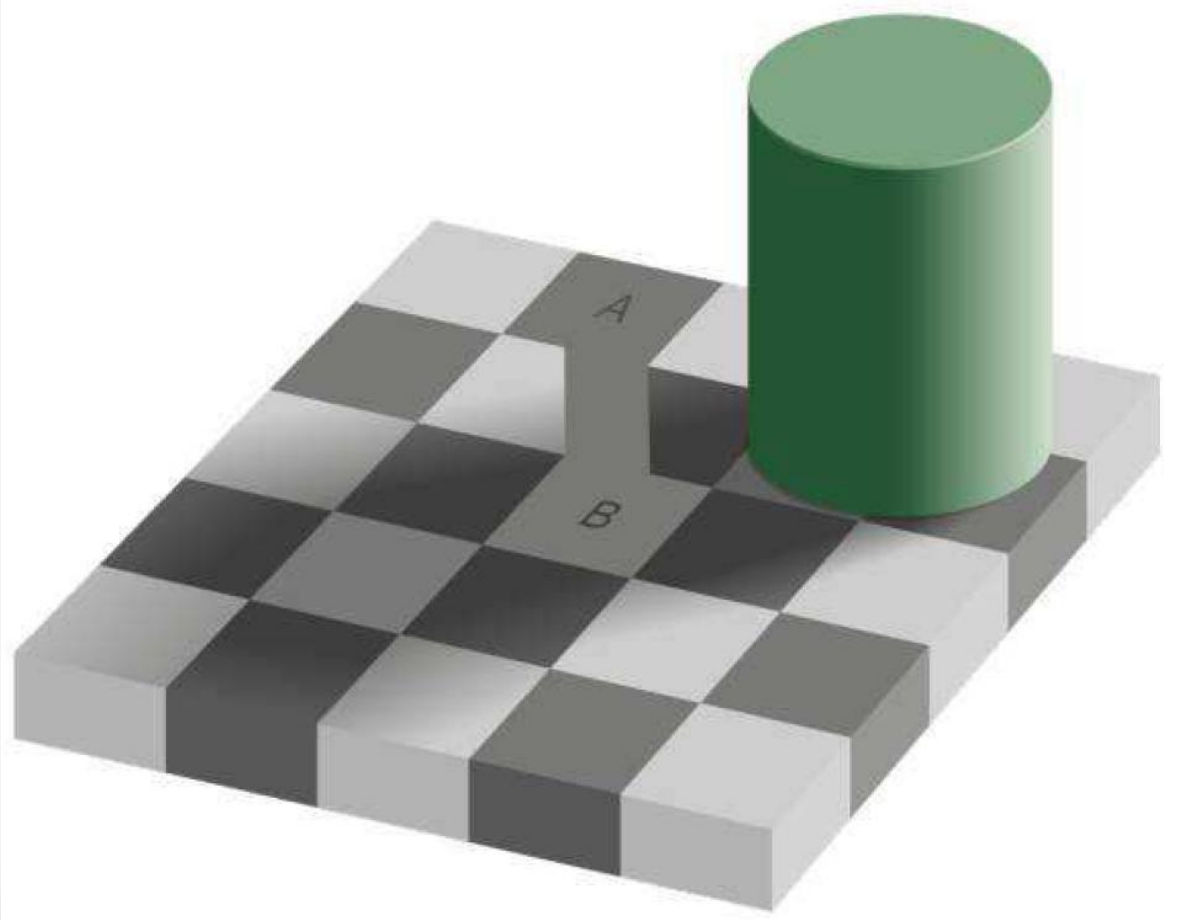
Visual Illusion

- Same Color ? A and B ?



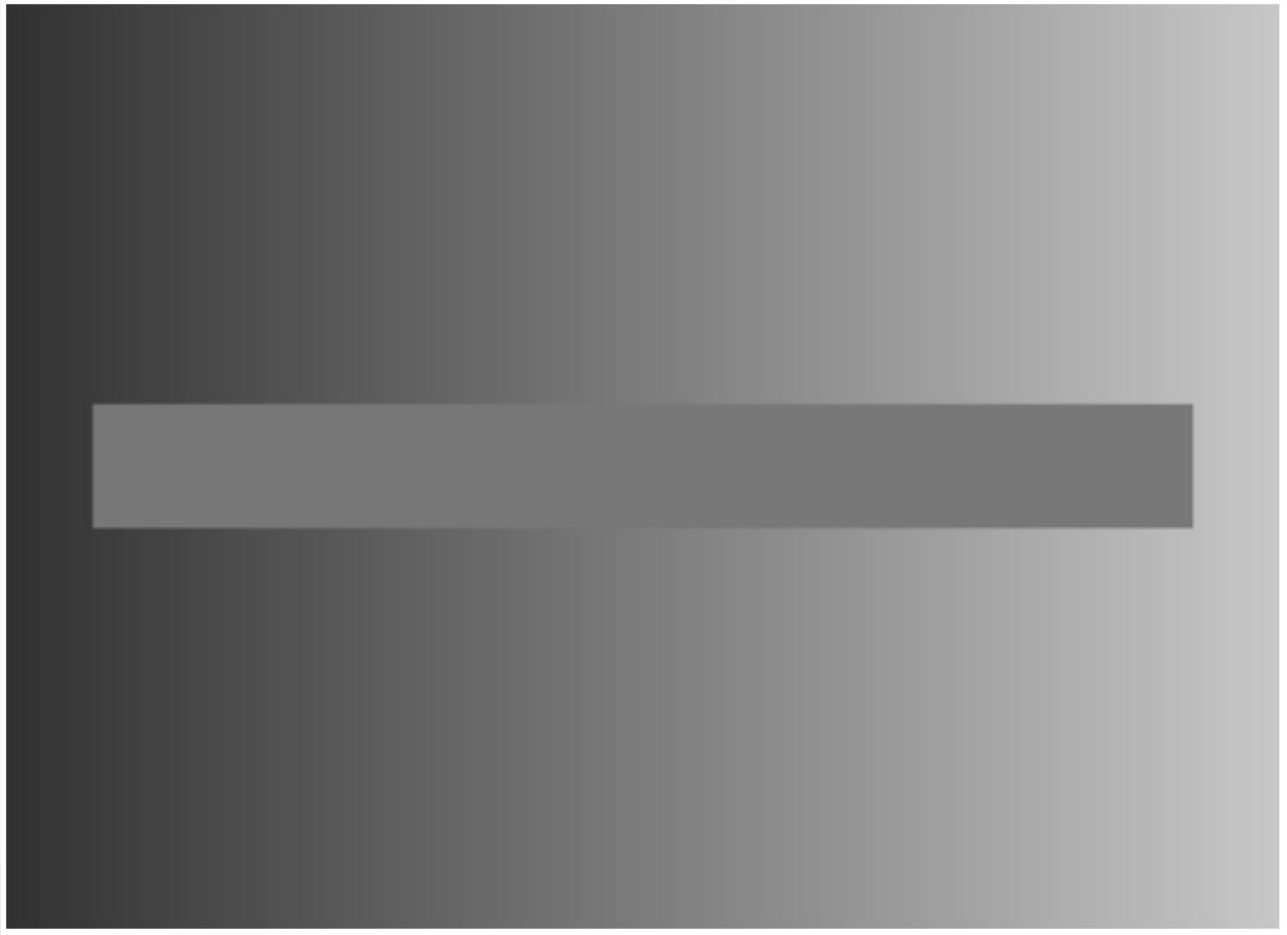
Visual Illusion

- Same Color ! We know it's shadow then….



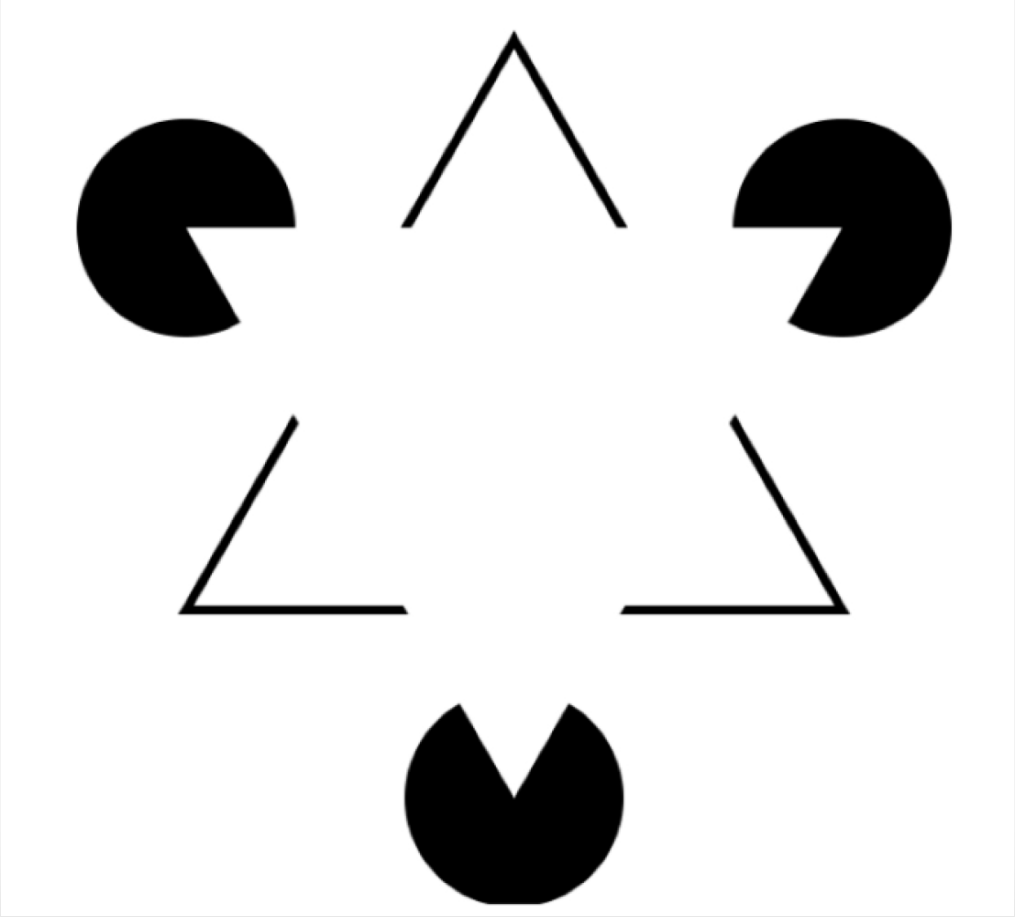
Visual Illusion

- Same Color – Contrast Sensitive



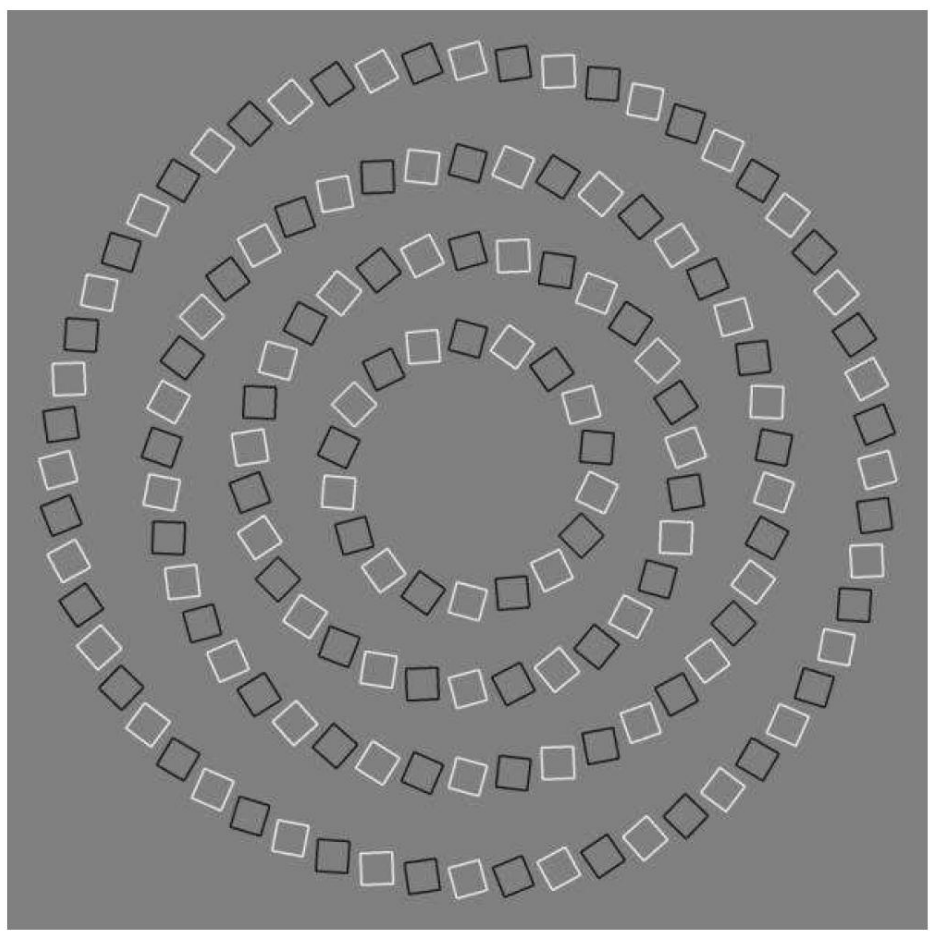
Visual Illusion

- Triangle – Space Sensitive



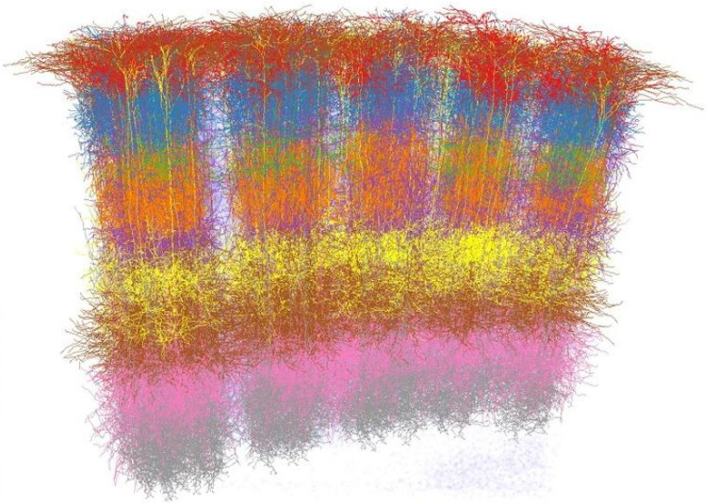
Visual Illusion

- Circle or Spiral Screw ?



Cortical Microcolumns

- AI need to learn computer graphs ?



Column through cortical layers of the brain
80-120 neurons (2X long in V1)
share the same receptive field

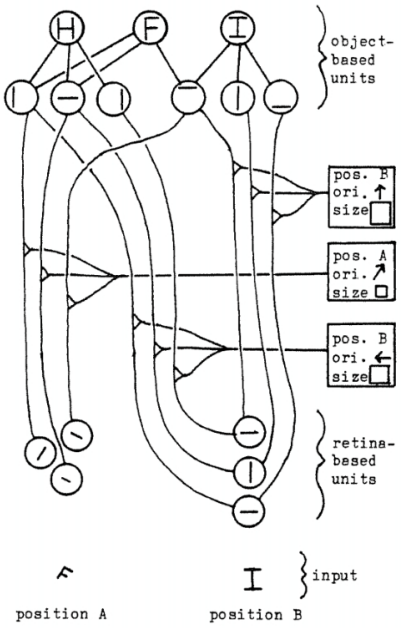
Capsules may encode
orientation scale
velocity color …

part of Hubel and Wiesel, Nobel Prize 1981

Cortical Microcolumns

- A kind of inverse computer graphs

Canonical object based frames of reference: Hinton 1981



In: Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver, B.C. Canada, 1981

A PARALLEL COMPUTATION THAT ASSIGNS CANONICAL OBJECT-BASED FRAMES OF REFERENCE

Geoffrey F. Hinton

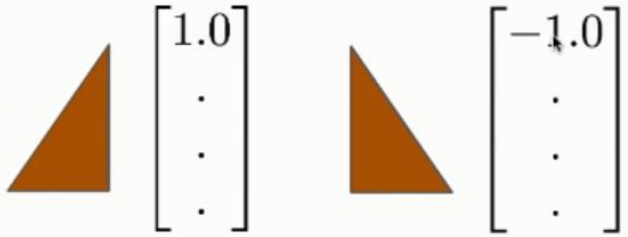
MRC Applied Psychology Unit
Cambridge, England

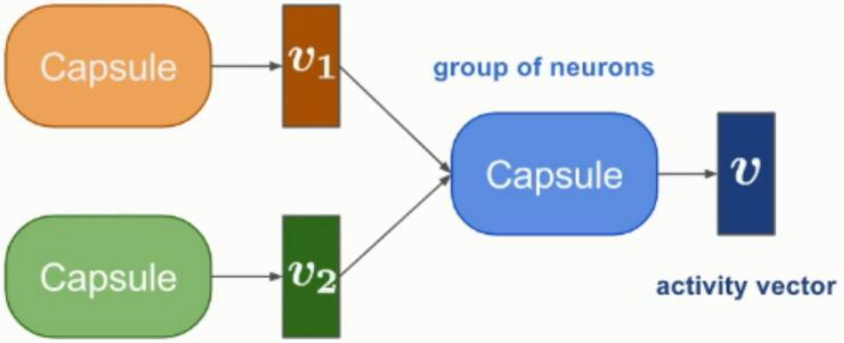
A kind of inverse computer graphics

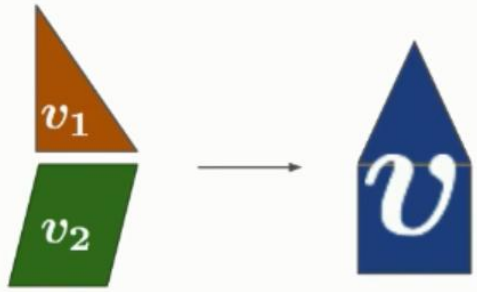
Hinton has been thinking about this a long time

What is a Capsule?

*"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 **an object part**."*





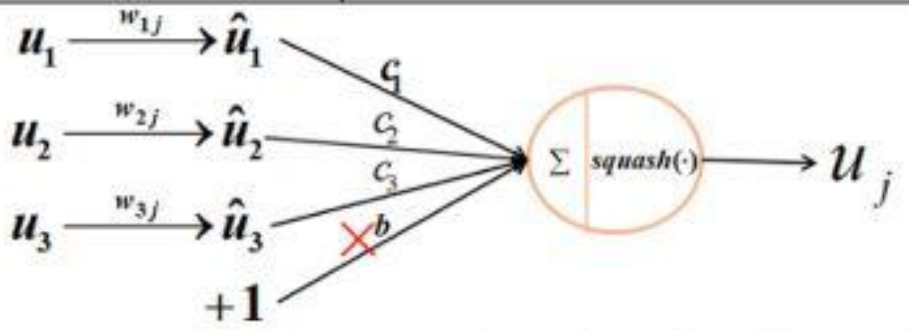


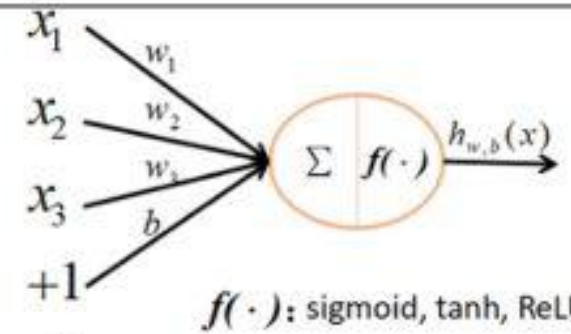
General ideas:

- Each dimension of v represents the characteristic of pattern;
- **The norm of v represents the exsistence (confidence). !!!**

Capsule Network

		capsule	vs.	traditional neuron
Input from low-level neuron/capsule		vector(u_i)		scalar(x_i)
Operation	Affine Transformation	$\hat{u}_{j\|i} = W_{ij} u_i$ (Eq. 2)		–
	Weighting	$s_j = \sum_i c_{ij} \hat{u}_{j\|i}$ (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_i)		scalar(h)



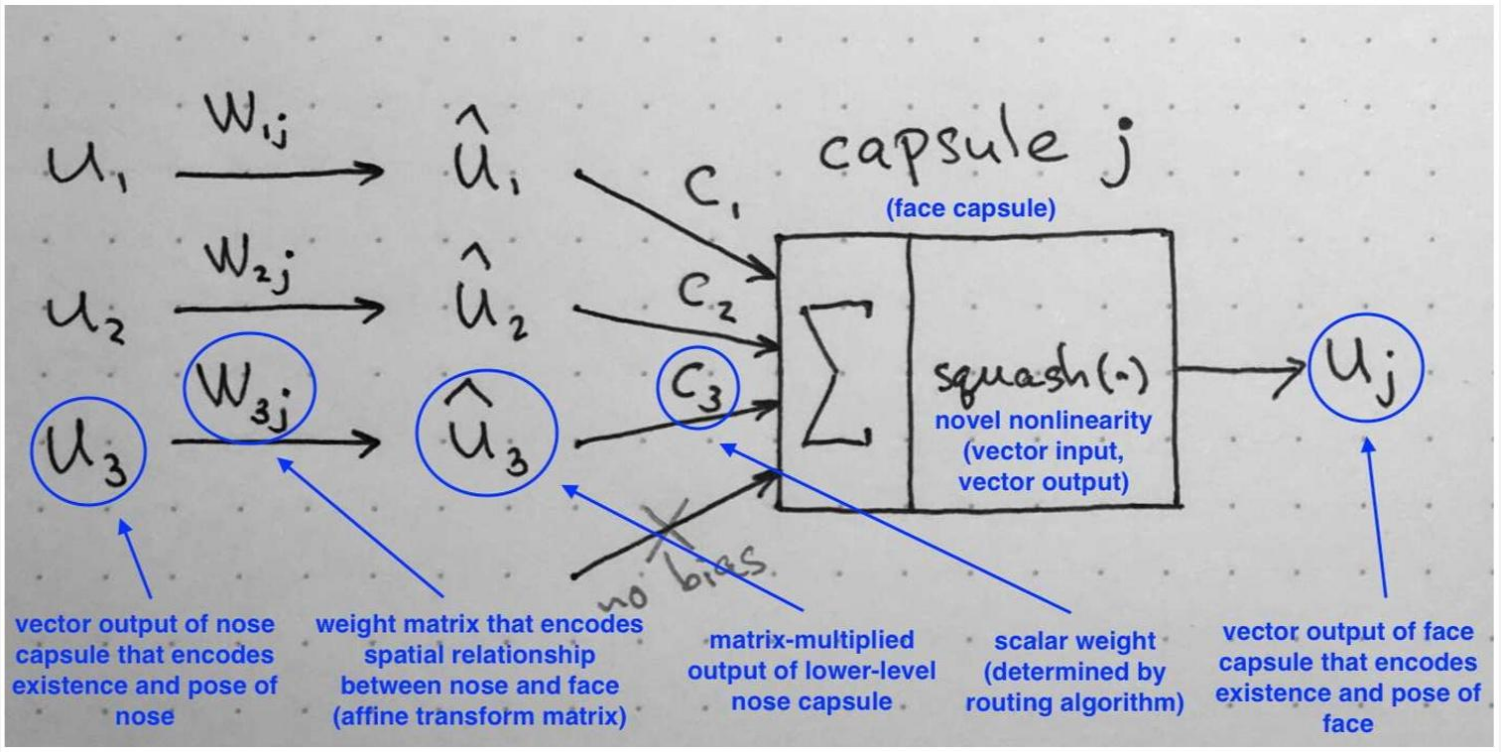


Capsule = New Version Neuron!
vector in, vector out VS. scalar in, scalar out

Capsule Network

- Summary

$$\mathbf{v}_j = \underbrace{\frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}}_{\text{additional "squashing"}} \underbrace{\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}}_{\text{unit scaling}}$$



How do capsule networks work?

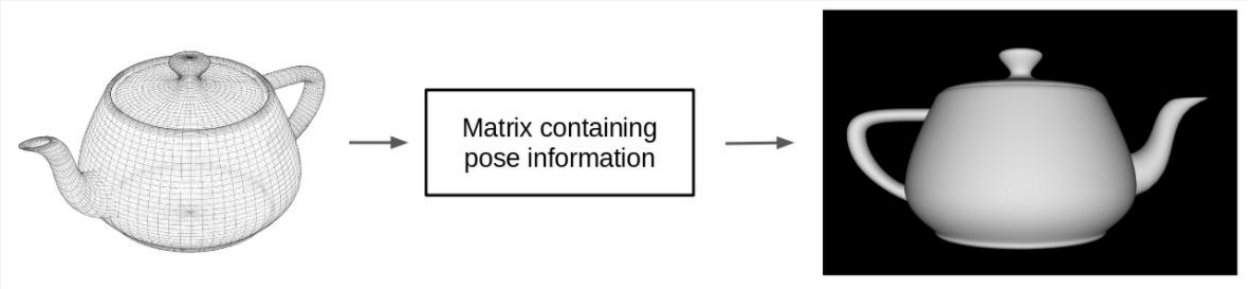
- A capsule network is composed of many capsules.
- A capsule is a function that tries to predict the presence and instantiation parameters of any particular object primitive at a particular location.
- The output of a capsule is a vector.
- The length of the vector is the estimated probability of a particular object at a location.
- The instantiation parameters of the object such orientation, thickness are all encoded along the multiple dimensions of the vectors it outputs.
- In the toy example in the video, orientation is encoded by the output vector angle

How do capsule networks work?

- A capsule function is implemented by applying convolution layers which generate as output feature maps - these maps are then used to get the vectors for each location.
- Capsule networks may perform better than convolutional neural networks for image segmentation/object detection because of a key feature - they preserve details information of the object - such as its location and pose, unlike convolutional neural networks where pooling layers can cause loss of some of this information (precise location and pose of object).
- While it looks promising it has not been tested on larger images. Also slower to train etc.

What is routing do?

- Using an iterative routing process, each active capsule will choose a capsule in the layer above to be its parent in the tree.
- For the higher levels of a visual system, this iterative process will be solving the problem of assigning parts to whole.

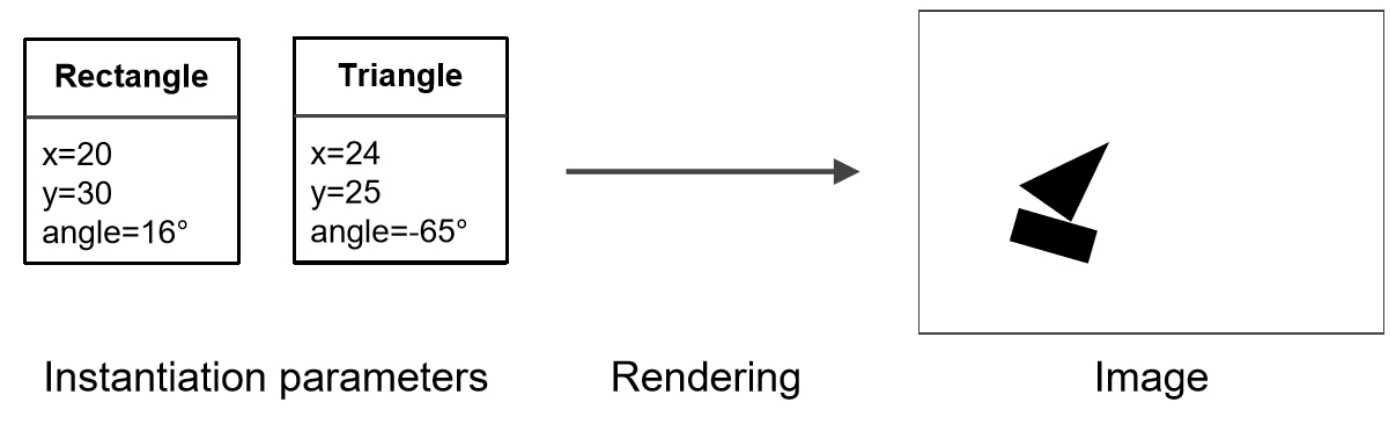


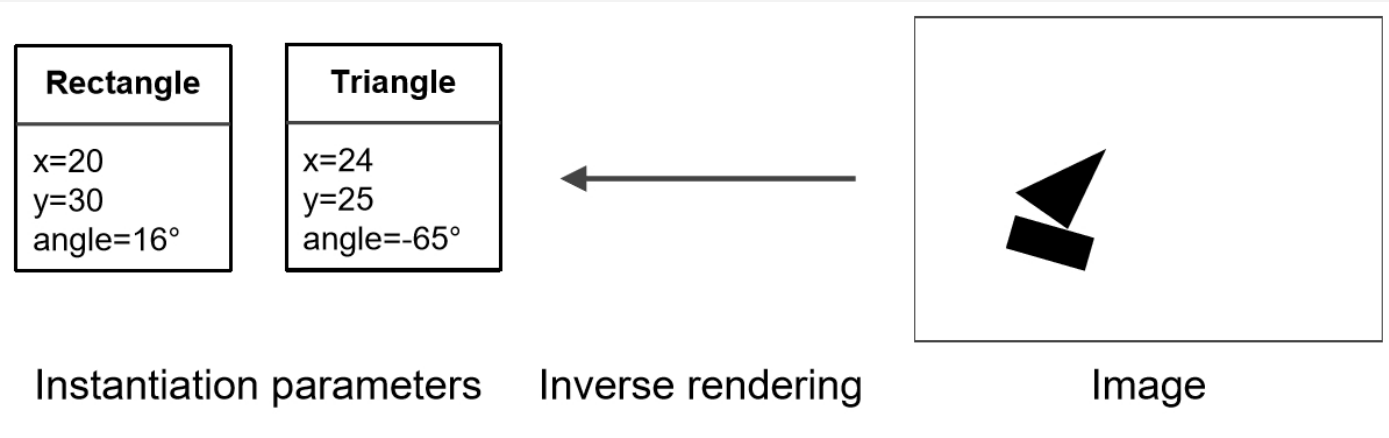
Inverse Graphics

You can think of (computer) vision as “Inverse Graphics” - Geoffrey Hinton

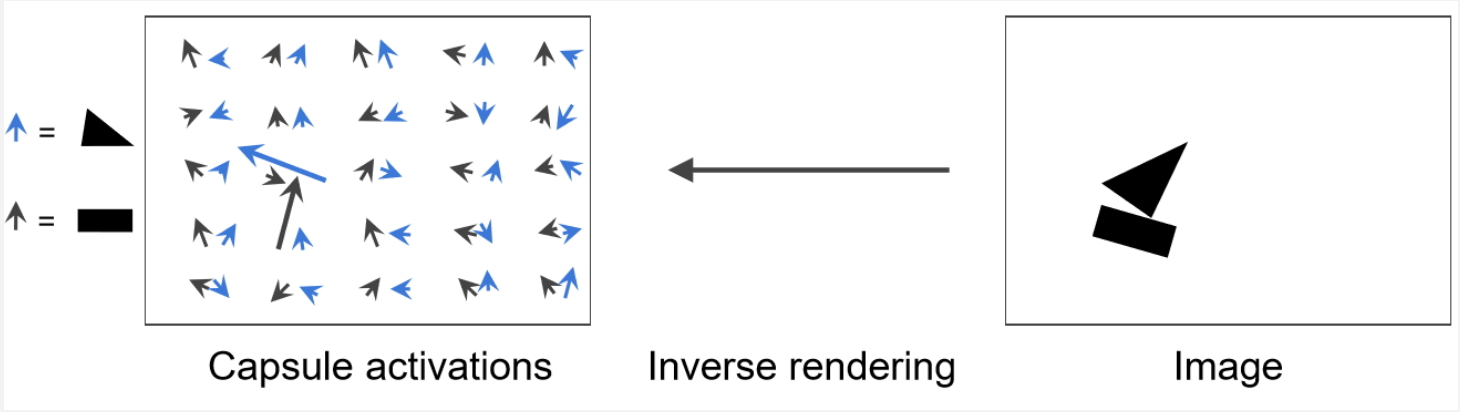
Dynamic Routing Between Capsules

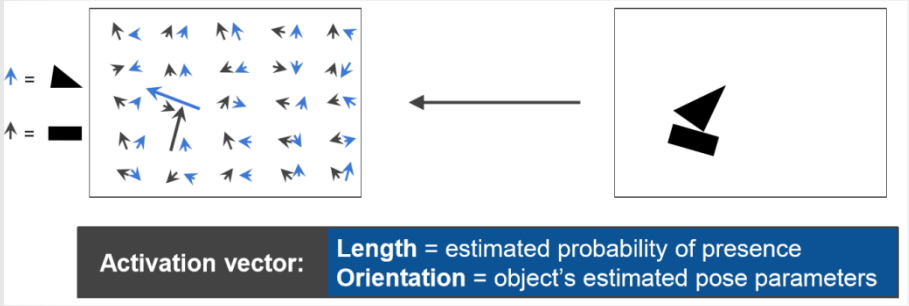
- Computer Graphics and Inverse Graphics

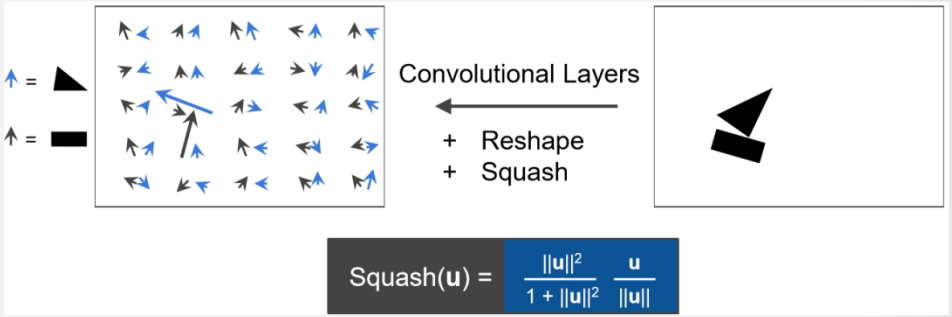




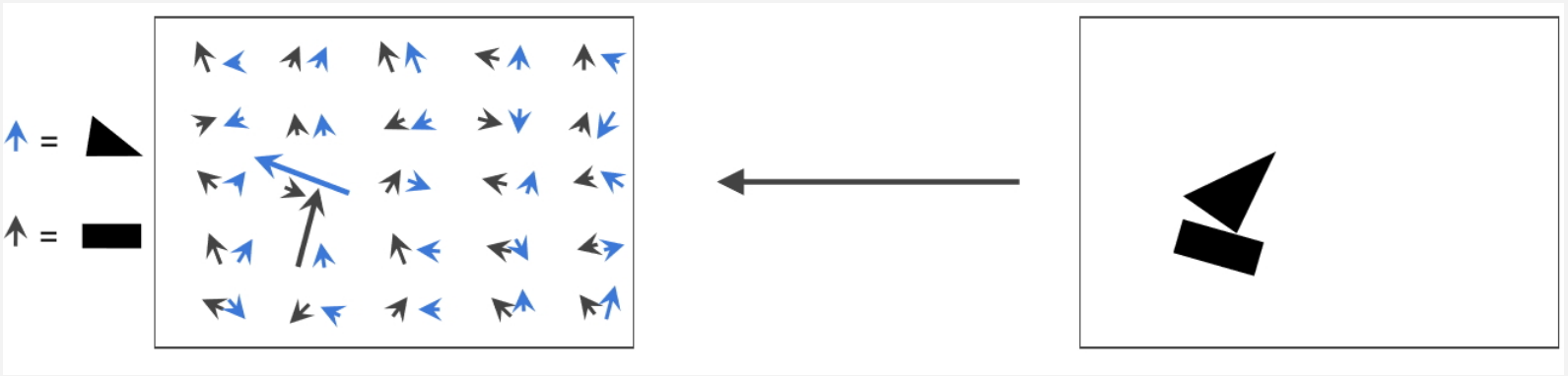
Learning Vector and Squash

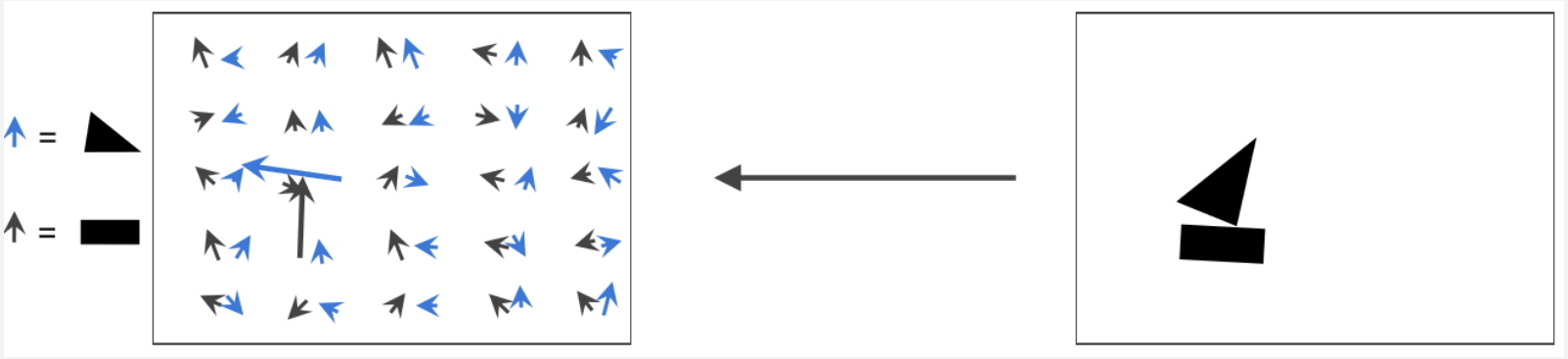






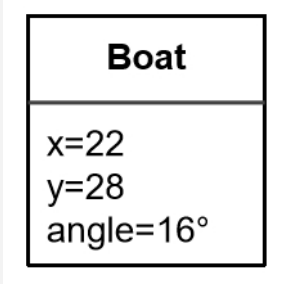
Parameters Fine Tuning

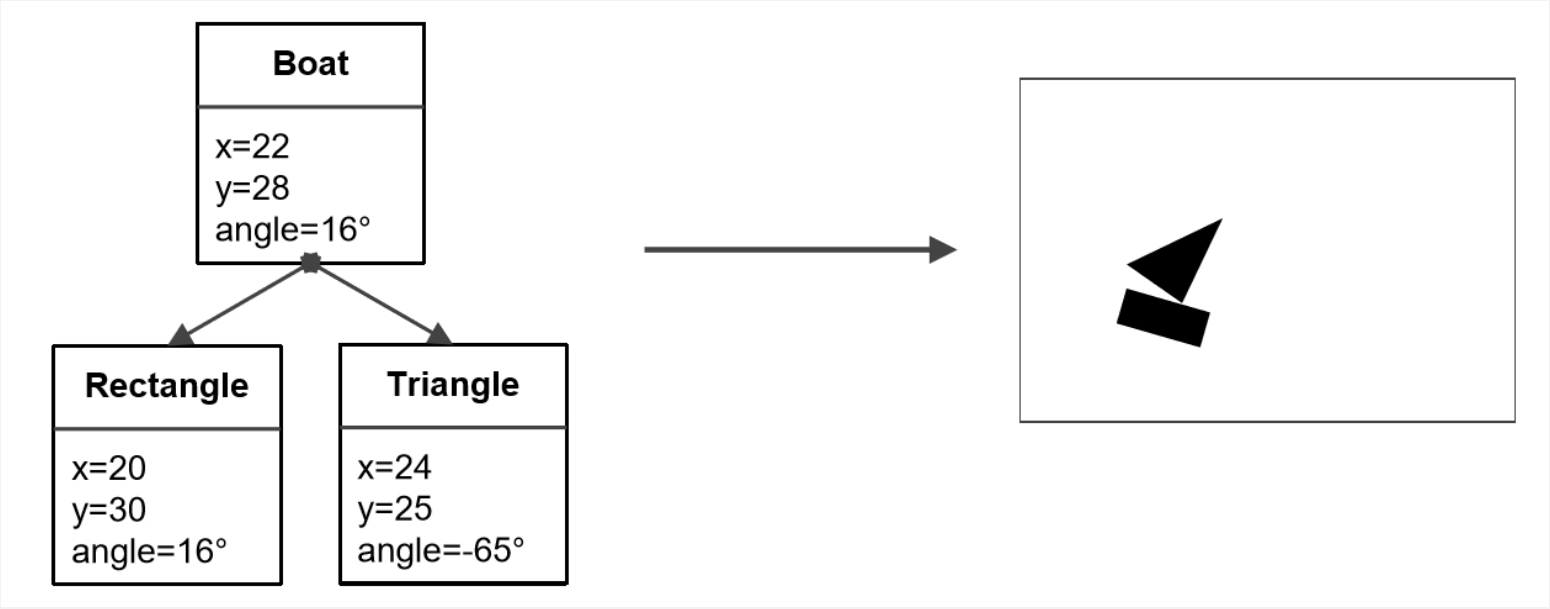




Boat Learned

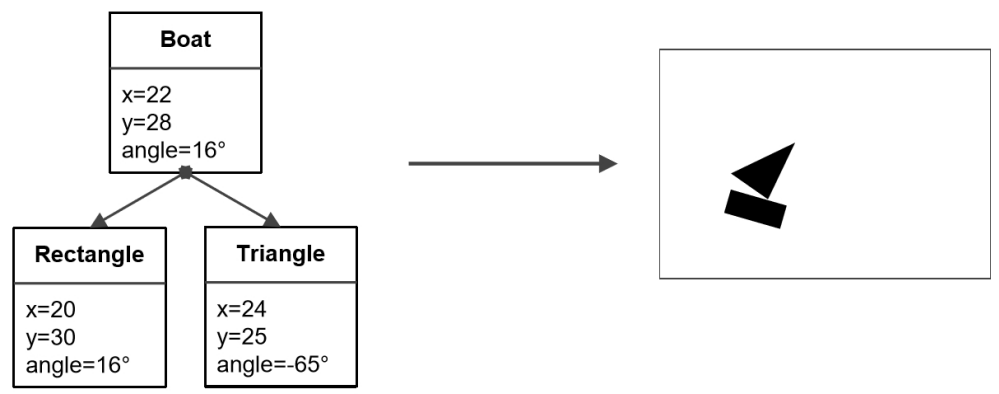
- Composed by two part

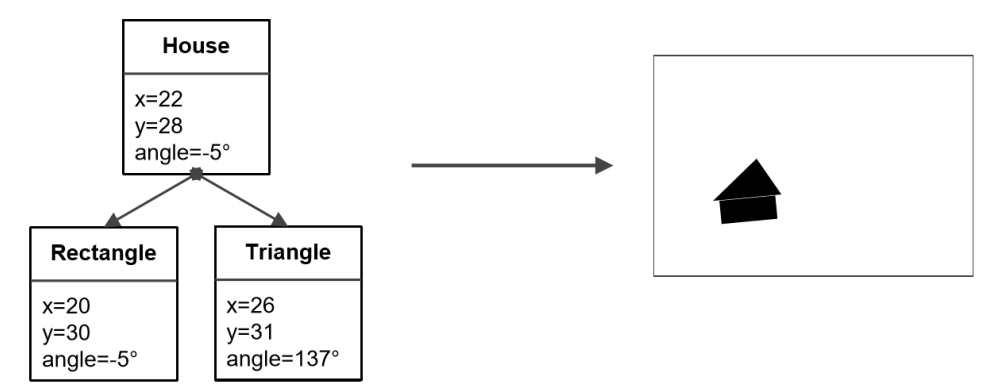




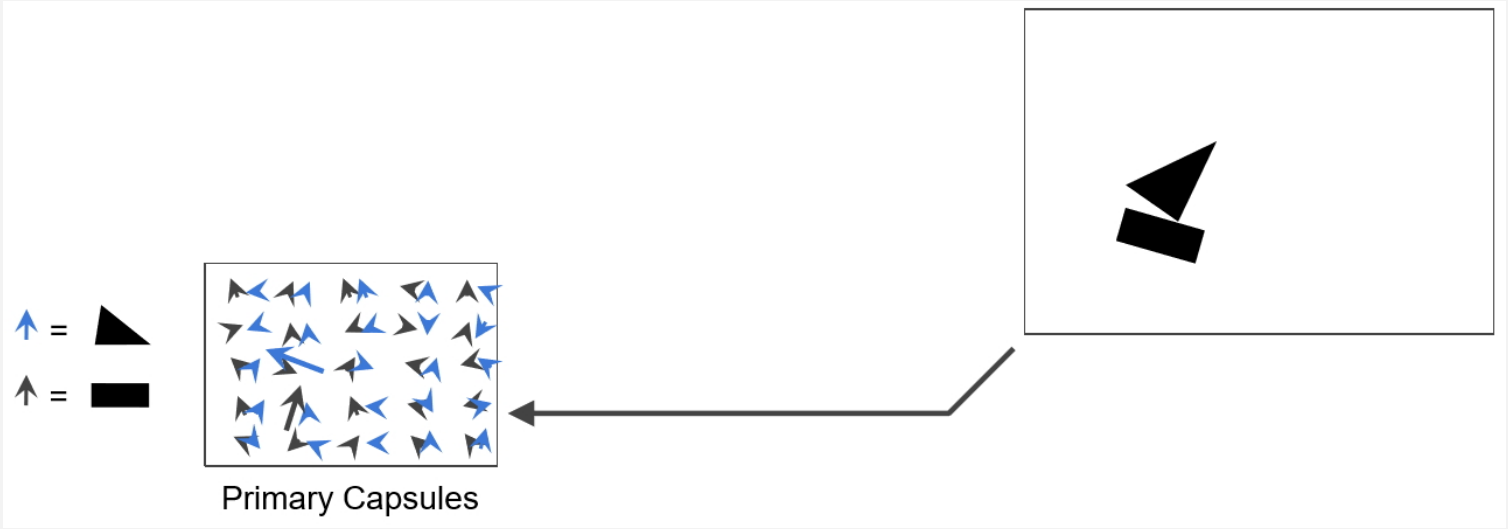
How to Predict ?

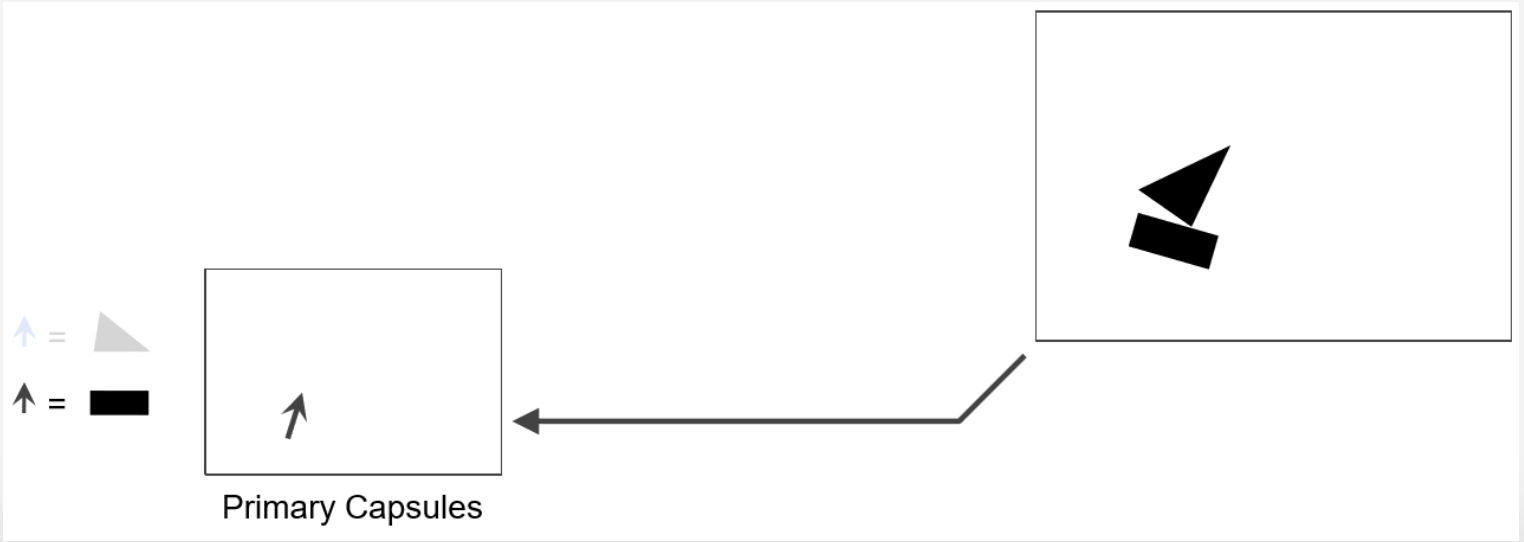
- A boat or a house? Hierarchy of parts
  - Predict a boat for example



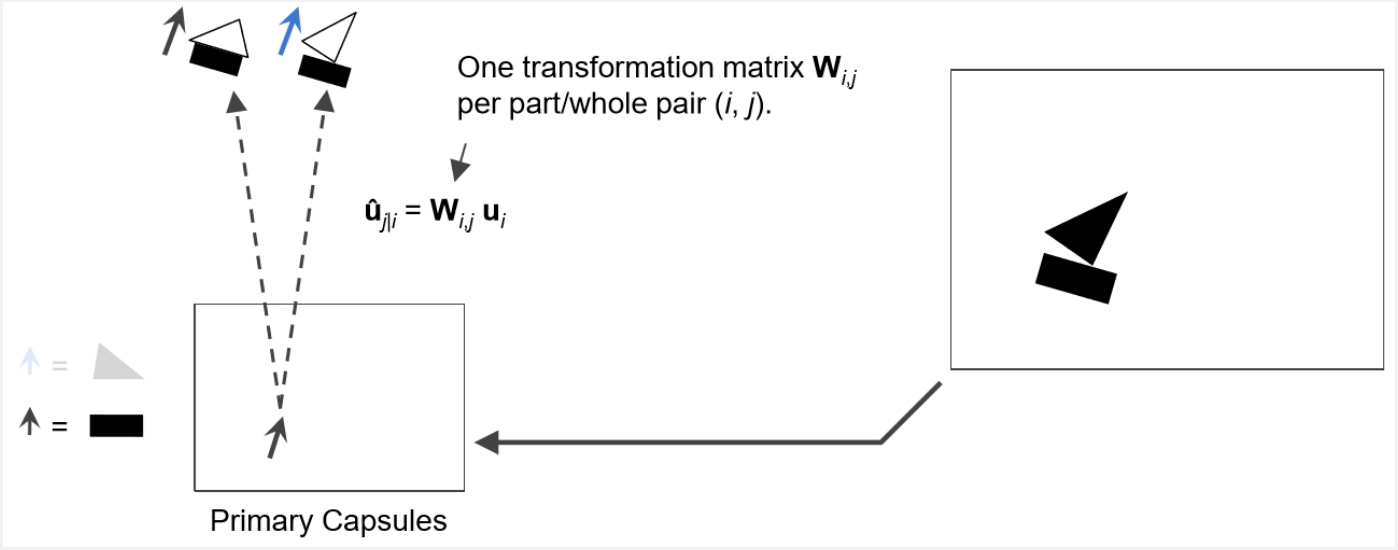


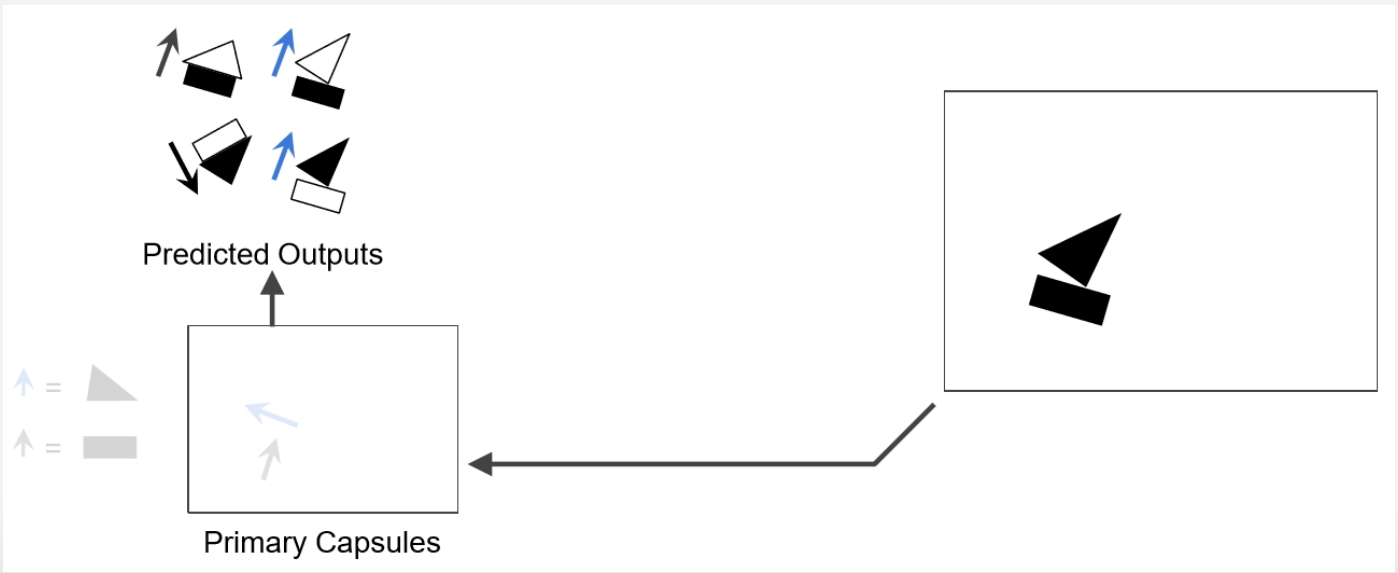
Find Primary Capsules First





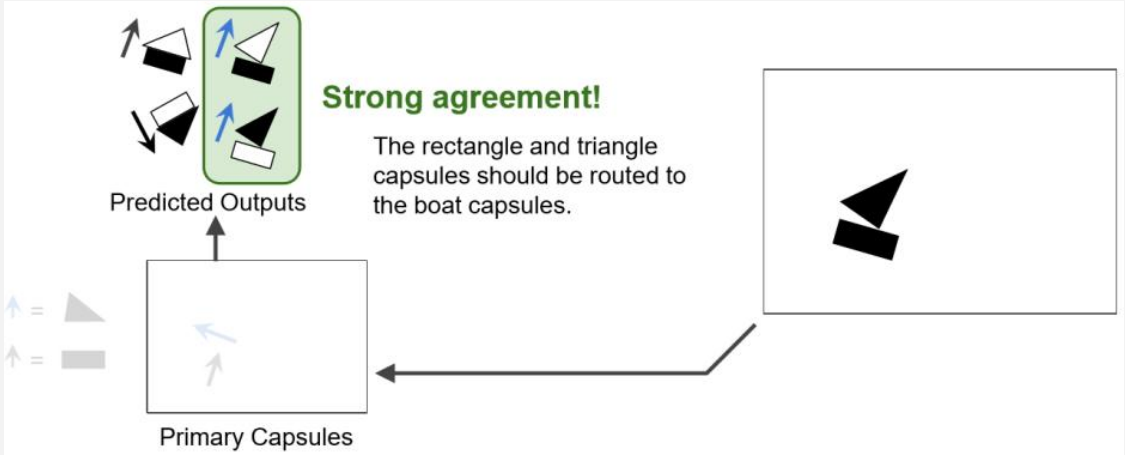
Predict Next Layer's Output

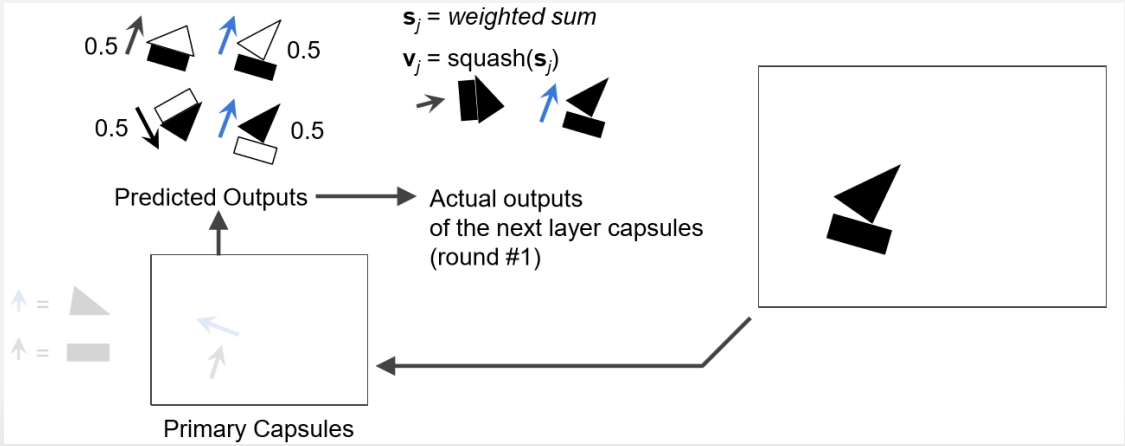




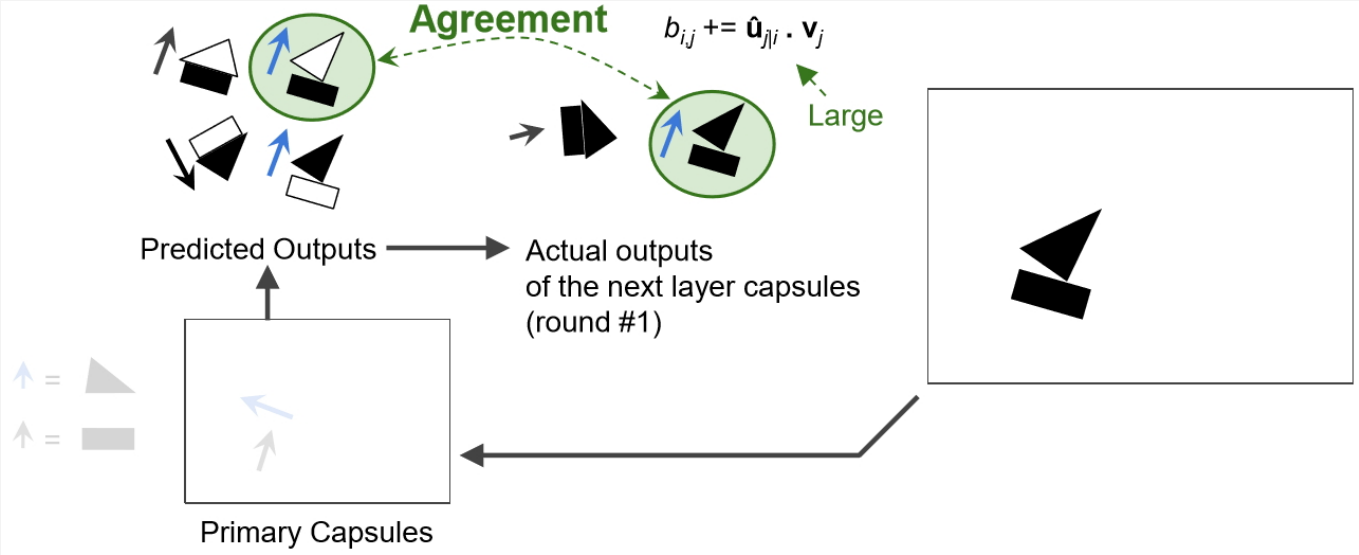
Routing By Agreement

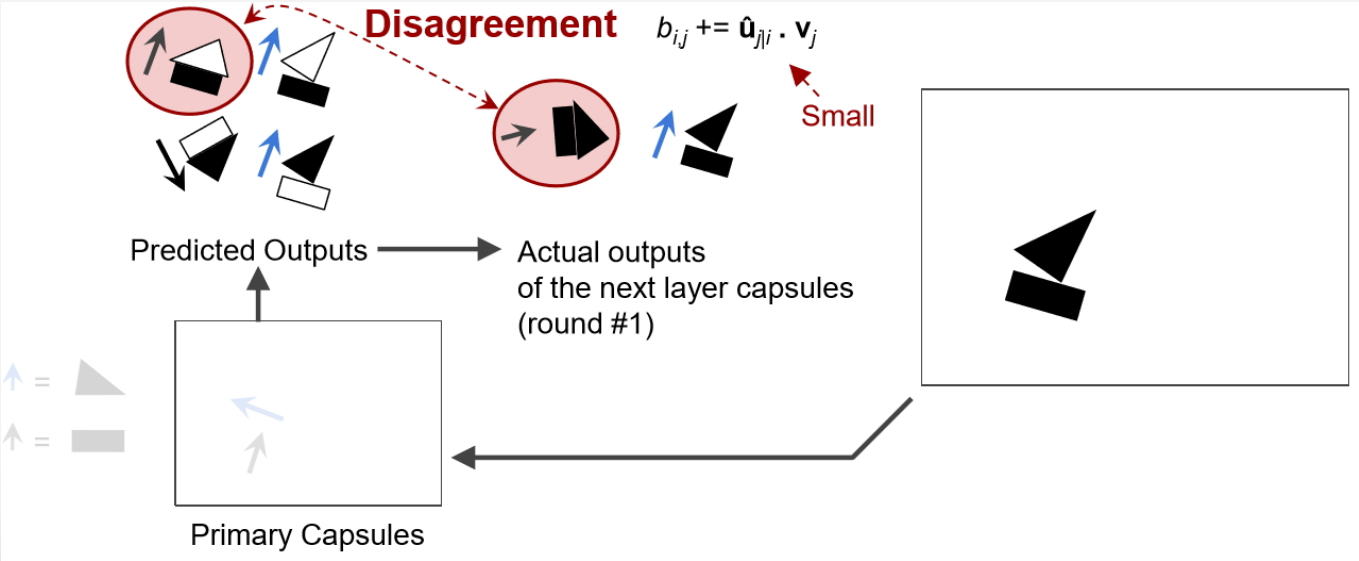
- Strong Agreement





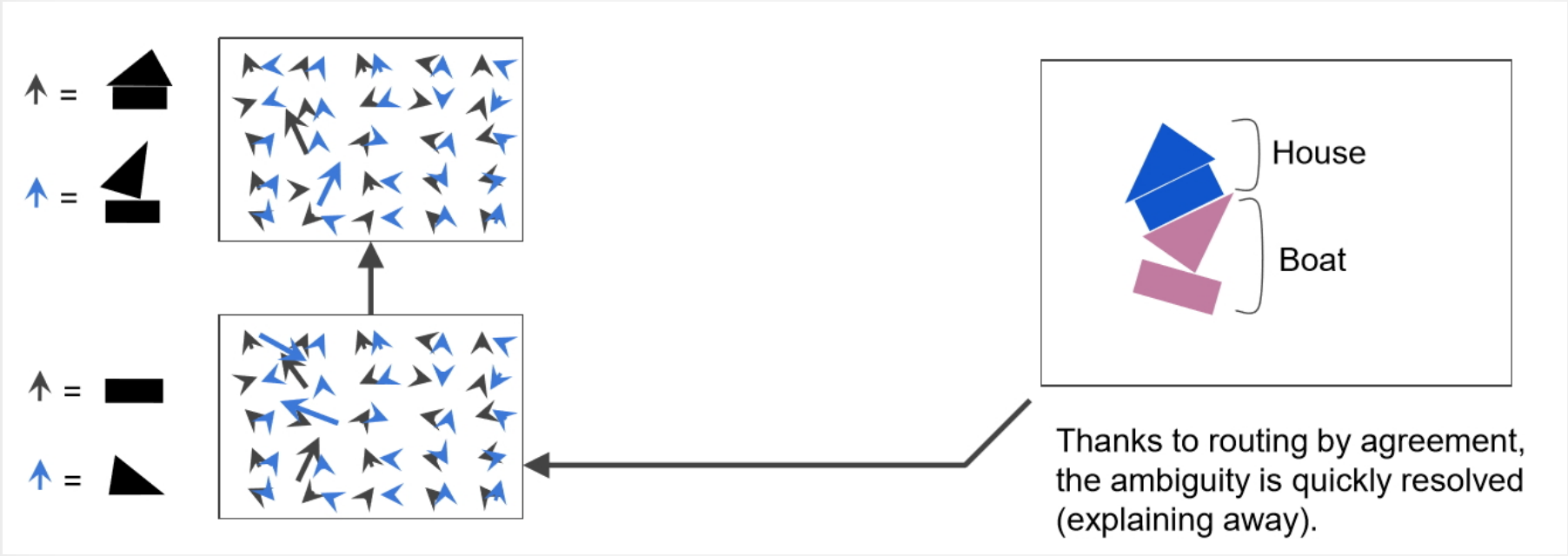
Strong Agreement





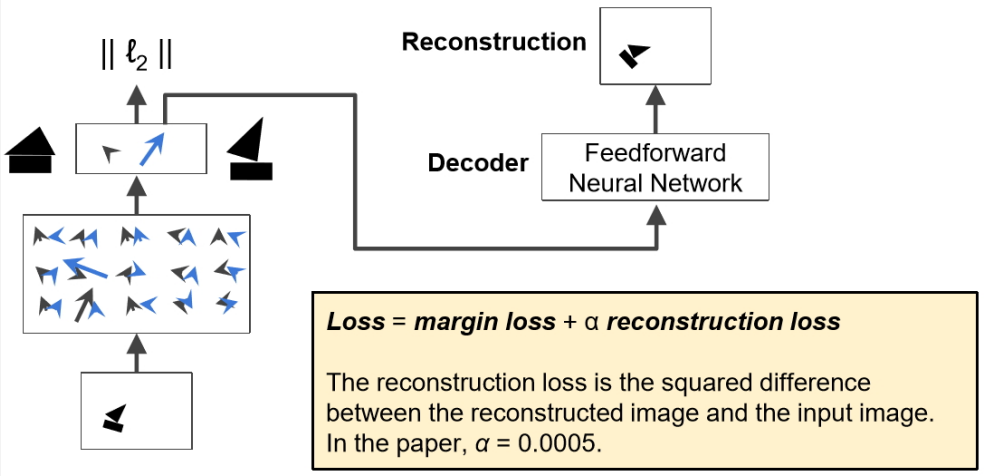
Handling Crowded Scenes

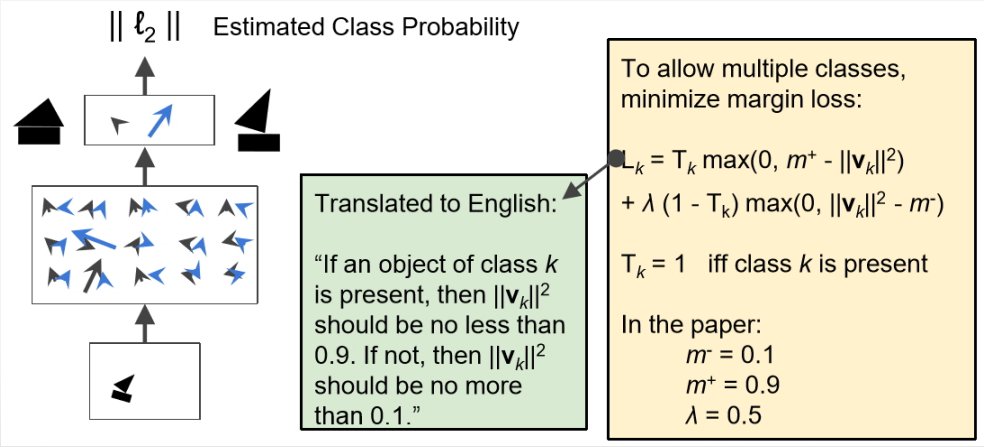
- Easy for Capsule Network



Regularization

- Regularization by Reconstruction





Dynamic VS EM Routing

- Dynamic Routing
  - Vectors as Pose (usually 8), Squash as Activation
  - Calculate Inner Product for Routing

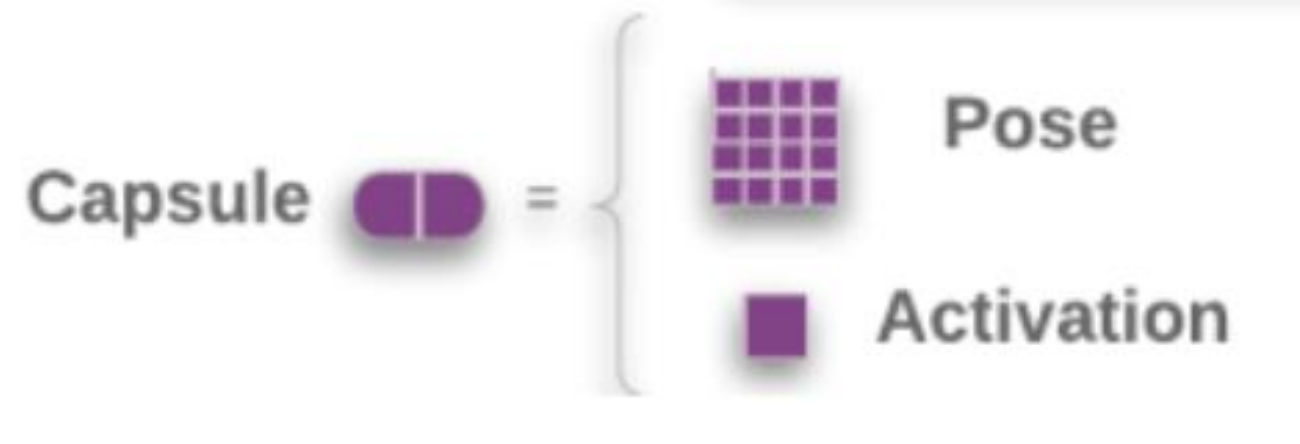
Procedure 1 Routing algorithm.

procedure ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
  for r iterations **do**
    for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
    for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
    for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
    for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} . \mathbf{v}_j$
  return $\mathbf{v}_j$

- EM Routing
  - Pose Matrix (4X4 in paper), A Probability of Activation
  - EM Approach Routing

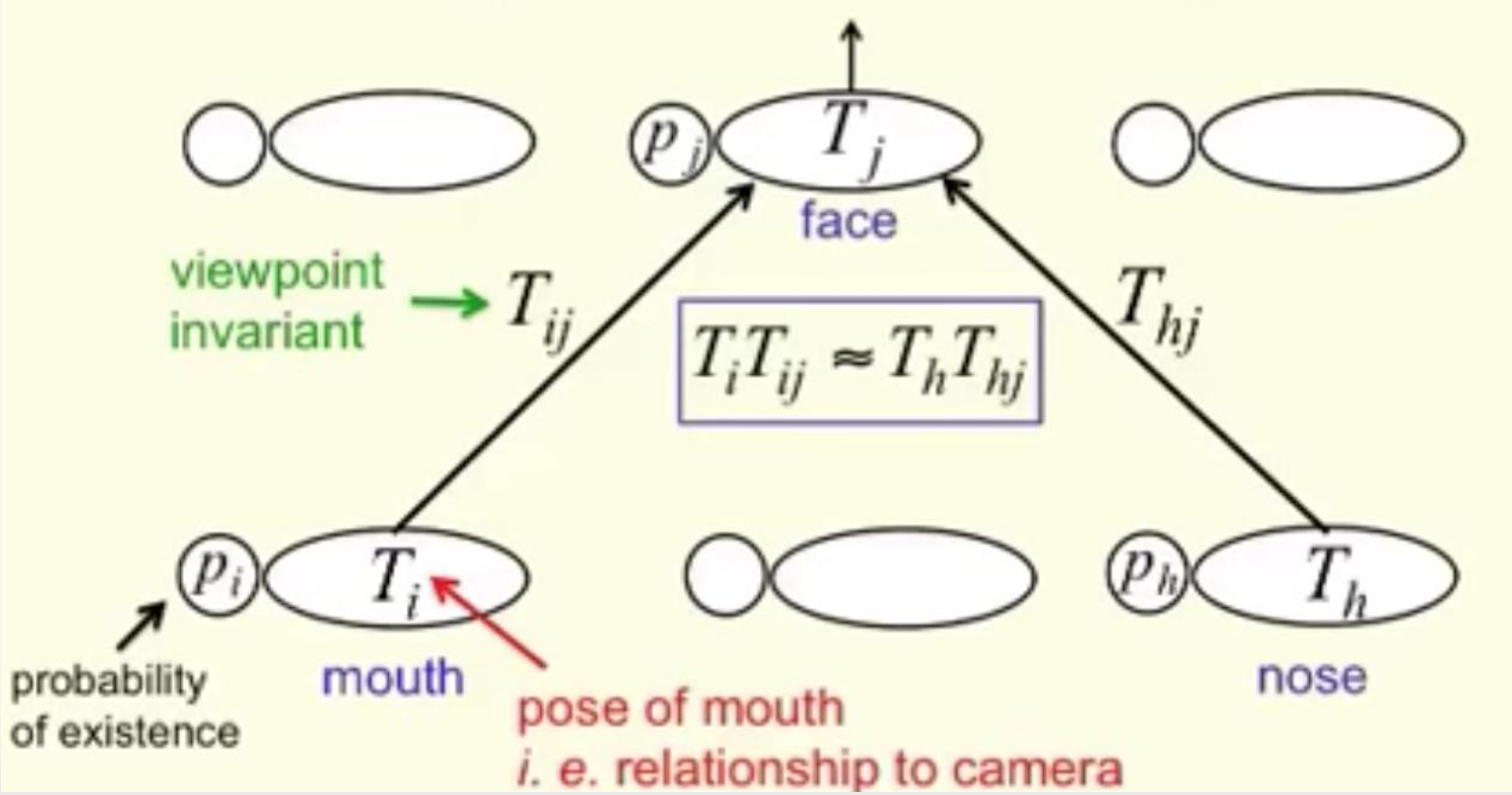
Dynamic VS EM Routing

- Dynamic Routing
  - Vectors as Pose (usually 8), Squash as Activation
  - Calculate Inner Product for Routing
- EM Routing
  - Pose Matrix (4X4 in paper), A Probability of Activation
  - EM Approach Routing



EM Routing Method

- Computer Graphics Transformation
  - Learn to represent part-whole relationships



EM Routing Algorithm

Procedure 1 Routing algorithm returns **activation** and **pose** of the capsules in layer $L+1$ given the **activations** and **votes** of capsules in layer L. V_{ij}^h is the h^{th} dimension of the vote from capsule i with activation a_i in layer L to capsule j in layer $L+1$. β_a, β_v are learned discriminatively and the inverse temperature λ increases at each iteration with a fixed schedule.

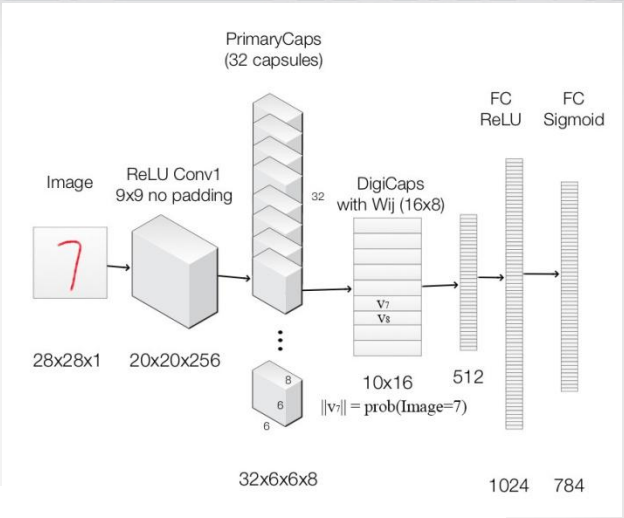
procedure EM ROUTING($\boldsymbol{a}$, V)
  $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
  for t iterations **do**
    $\forall j \in \Omega_{L+1}$: M-STEP($\boldsymbol{a}$, R, V, j)
    $\forall i \in \Omega_L$: E-STEP(μ, σ, $\boldsymbol{a}$, V, i)
  return $\boldsymbol{a}$, M

procedure M-STEP($\boldsymbol{a}$, R, V, j) ▷ for one higher-level capsule
  $\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * \boldsymbol{a}_i$
  $\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
  $\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} (V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
  $cost^h \leftarrow (\beta_v + log(\sigma_j^h)) \sum_i R_{ij}$
  $a_j \leftarrow sigmoid(\lambda(\beta_a - \sum_h cost^h))$

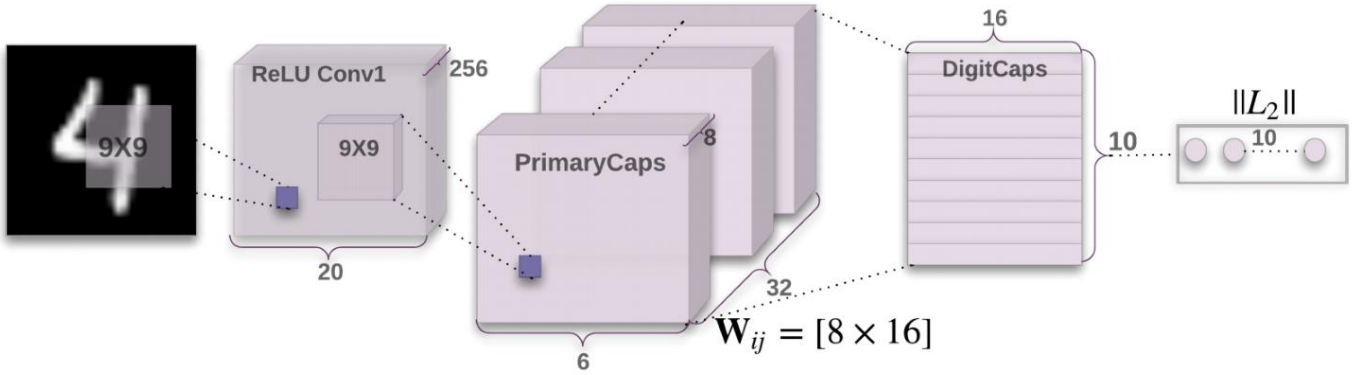
procedure E-STEP(μ, σ, $\boldsymbol{a}$, V, i) ▷ for one lower-level capsule
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{p}_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} e^{-\sum_h^H \frac{(V_{ij}^h - \boldsymbol{\mu}_j^h)^2}{2(\sigma_j^h)^2}}$
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{R}_{ij} \leftarrow \frac{\boldsymbol{a}_j \boldsymbol{p}_j}{\sum_{u \in \Omega_{L+1}} \boldsymbol{a}_u \boldsymbol{p}_u}$

Architecture

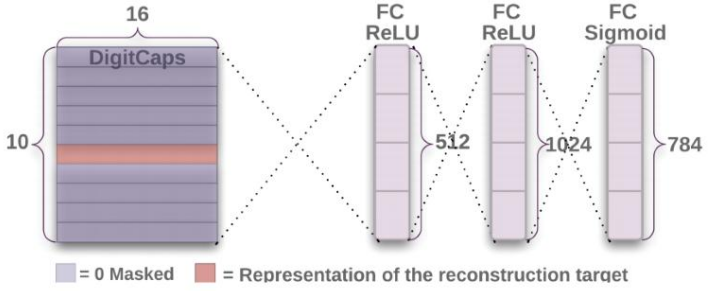
- Dynamic Routing on MNIST



supervised | max norm loss



+ unsupervised | reconstruction loss



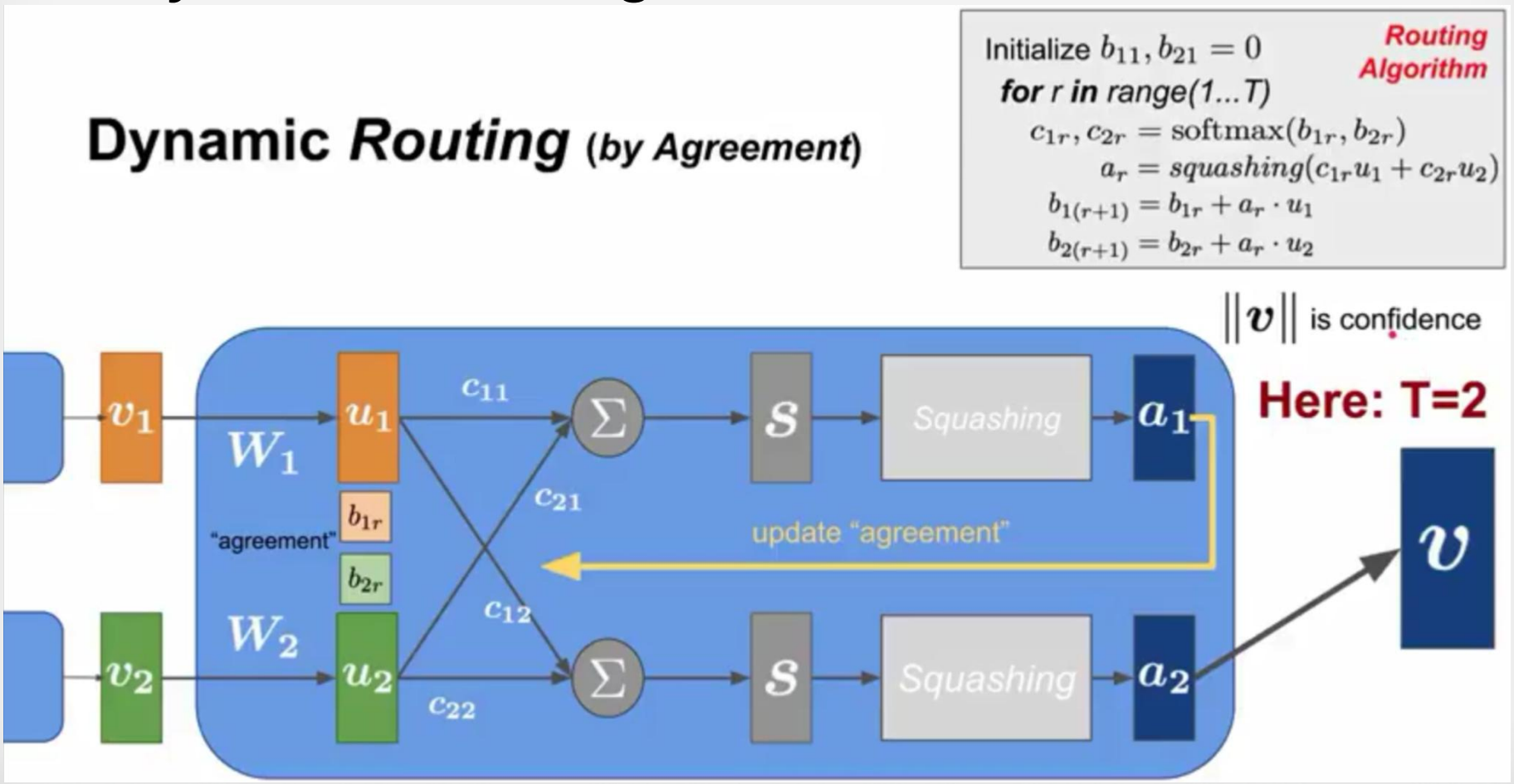
Architecture

- Dynamic Routing on MNIST
  - https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/

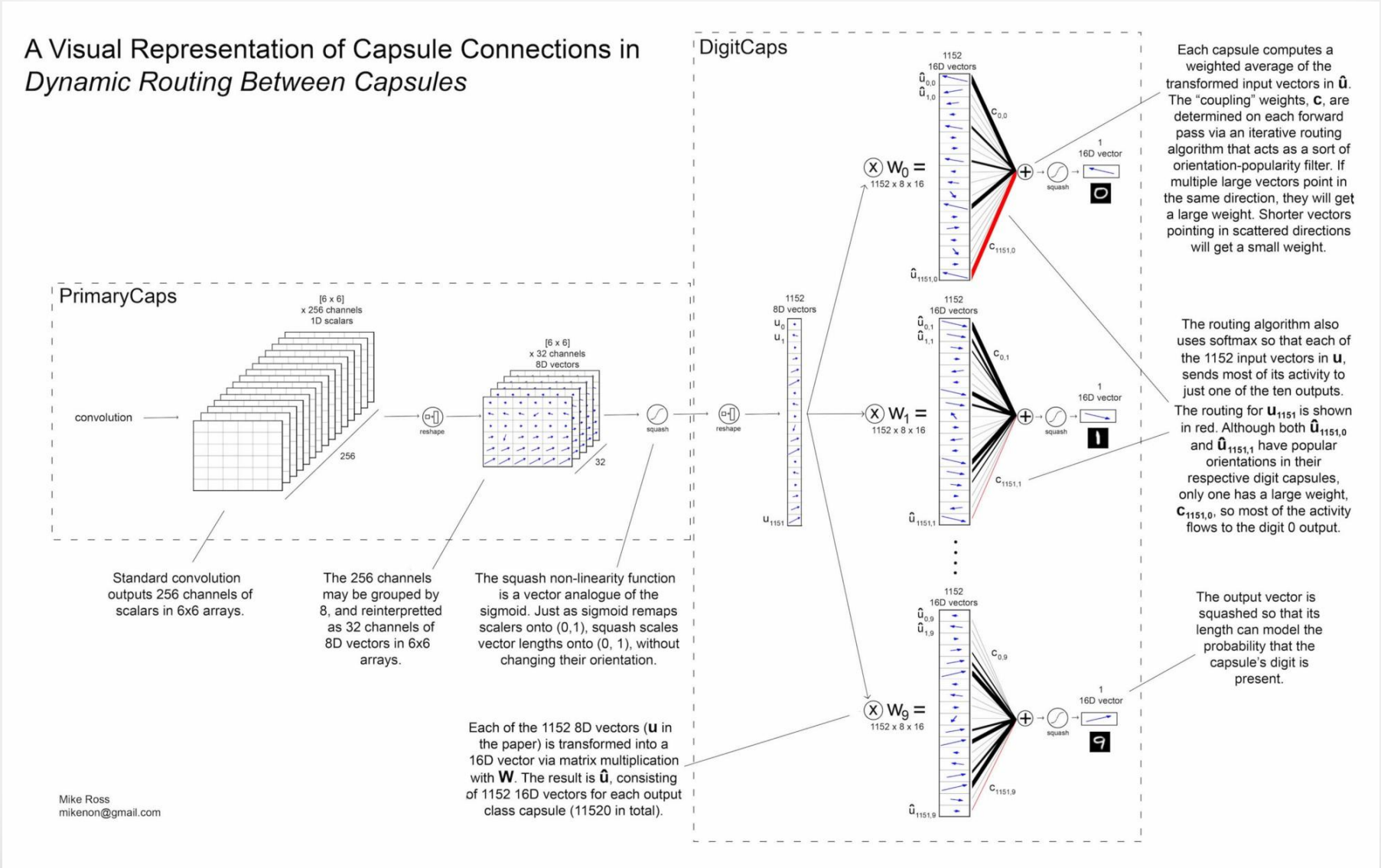
Layer Name	Apply	Output shape
Image	Raw image array	28x28x1
ReLU Conv1	Convolution layer with 9x9 kernels output 256 channels, stride 1, no padding with ReLU	20x20x256
PrimaryCapsules	Convolution capsule layer with 9x9 kernel output 32x6x6 8-D capsule, stride 2, no padding	6x6x32x8
DigiCaps	Capsule output computed from a W_{ij} (16x8 matrix) between u_i and v_j (i from 1 to 32x6x6 and j from 1 to 10).	10x16
FC1	Fully connected with ReLU	512
FC2	Fully connected with ReLU	1024
Output image	Fully connected with sigmoid	784 (28x28)

Routing Method

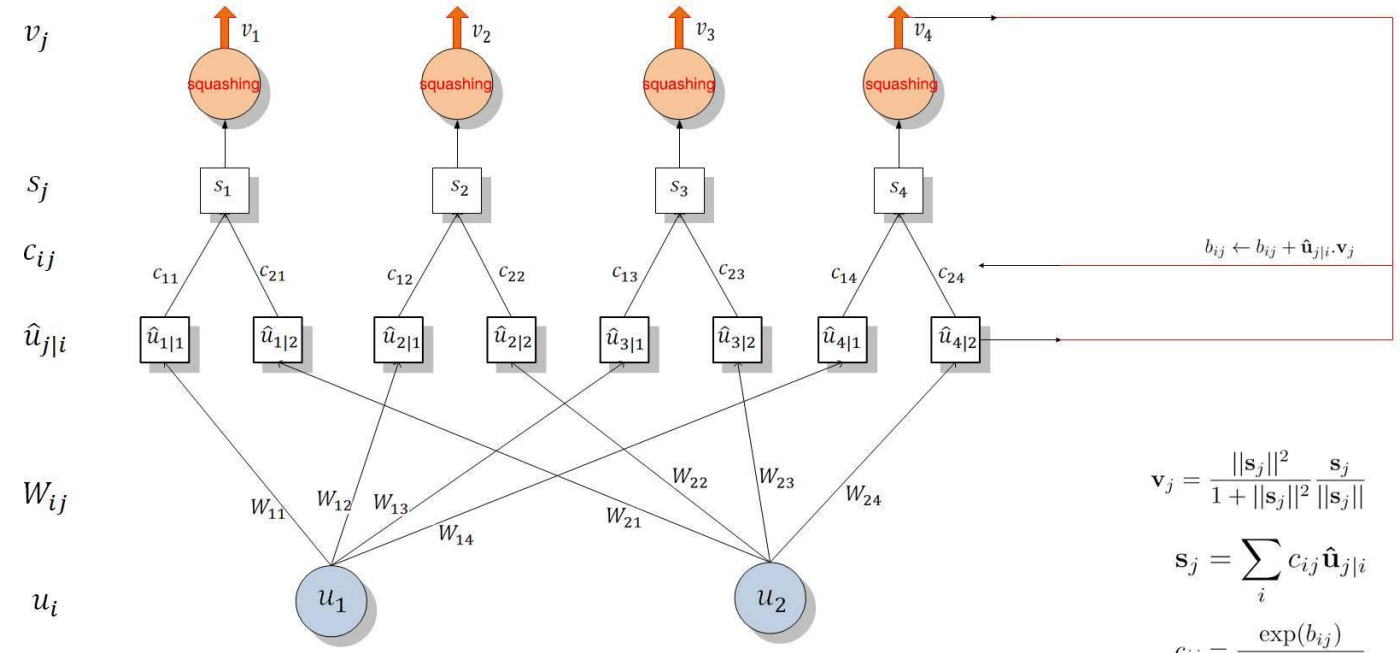
- Dynamic Routing



Dynamic Routing



Dynamic Routing



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

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

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

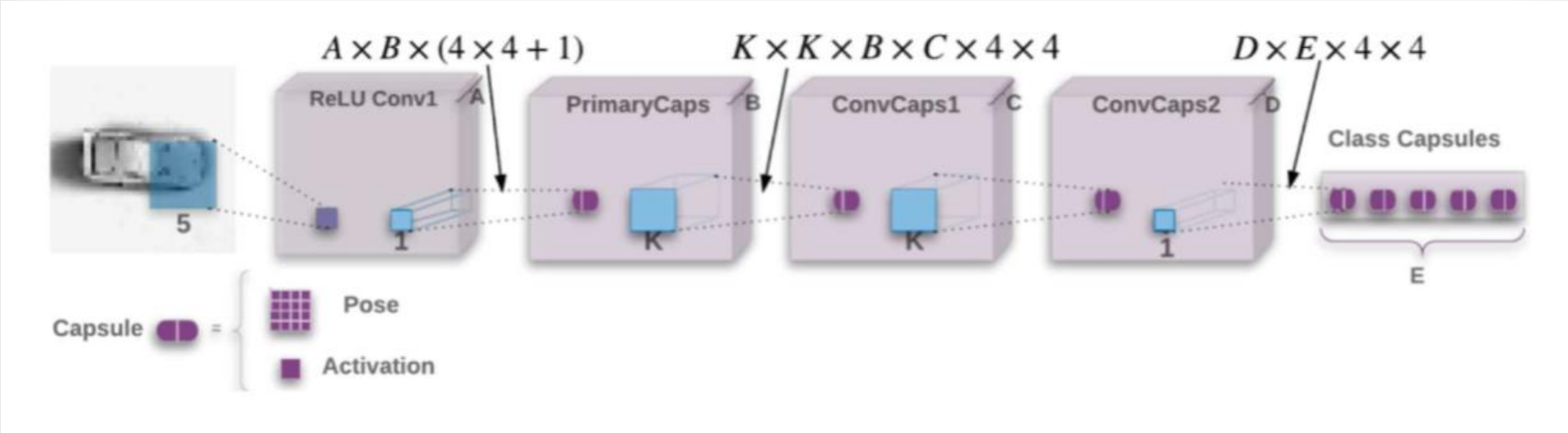
$$\hat{u}_{j|i} = W_{ij} u_i$$

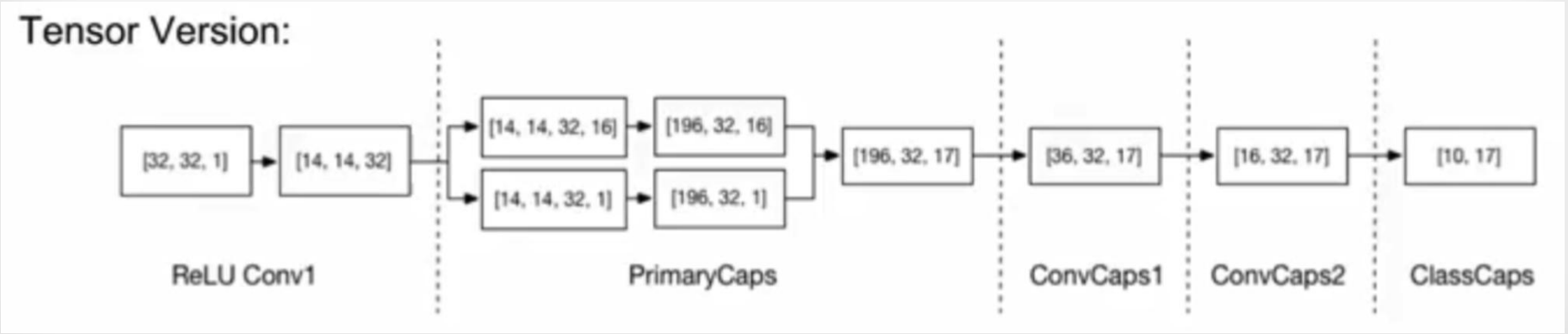
Procedure 1 Routing algorithm.

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: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
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 $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

Architecture

- EM Routing on MNIST





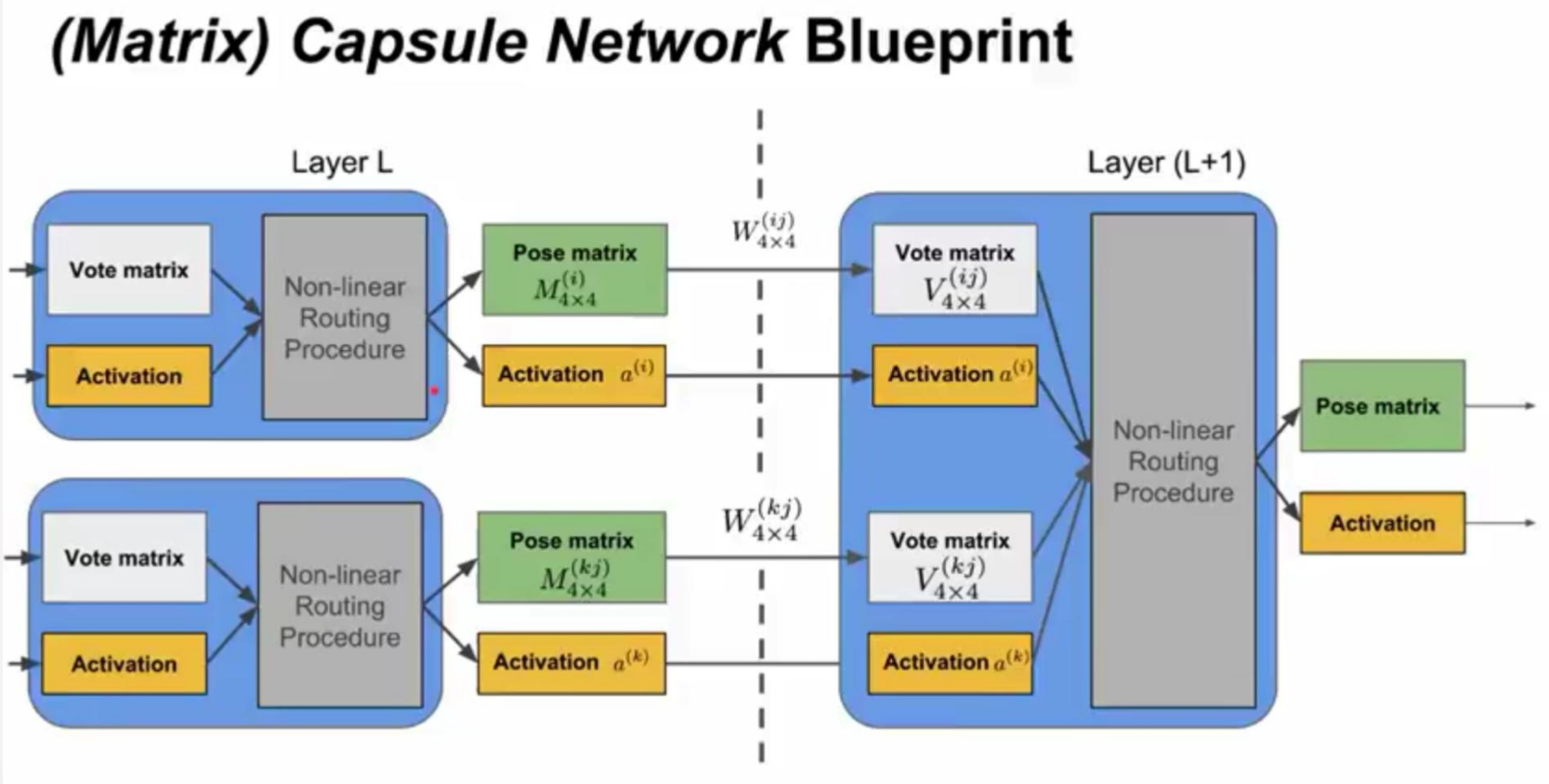
Architecture

- EM Routing on MNIST
  - https://jhui.github.io/2017/11/14/Matrix-Capsules-with-EM-routing-Capsule-Network/

Layer Name	Apply	Output shape
MNist image		28, 28, 1
ReLU Conv1	Regular Convolution (CNN) layer using 5x5 kernels with 32 output channels, stride 2 and padding	14, 14, 32
PrimaryCaps	Modified convolution layer with 1x1 kernels, strides 1 with padding and outputing 32 capsules. Requiring 32x32x(4x4+1) parameters.	pose (14, 14, 32, 4, 4), activations (14, 14, 32)
ConvCaps1	Capsule convolution with 3x3 kernels, strides 2 and no padding. Requiring 3x3x32x32x4x4 parameters.	poses (6, 6, 32, 4, 4), activations (6, 6, 32)
ConvCaps2	Capsule convolution with 3x3 kernels, strides 1 and no padding	poses (4, 4, 32, 4, 4), activations (4, 4, 32)
Class Capsules	Capsule with 1x1 kernel. Requiring 32x10x4x4 parameters.	poses (10, 4, 4), activations (10)

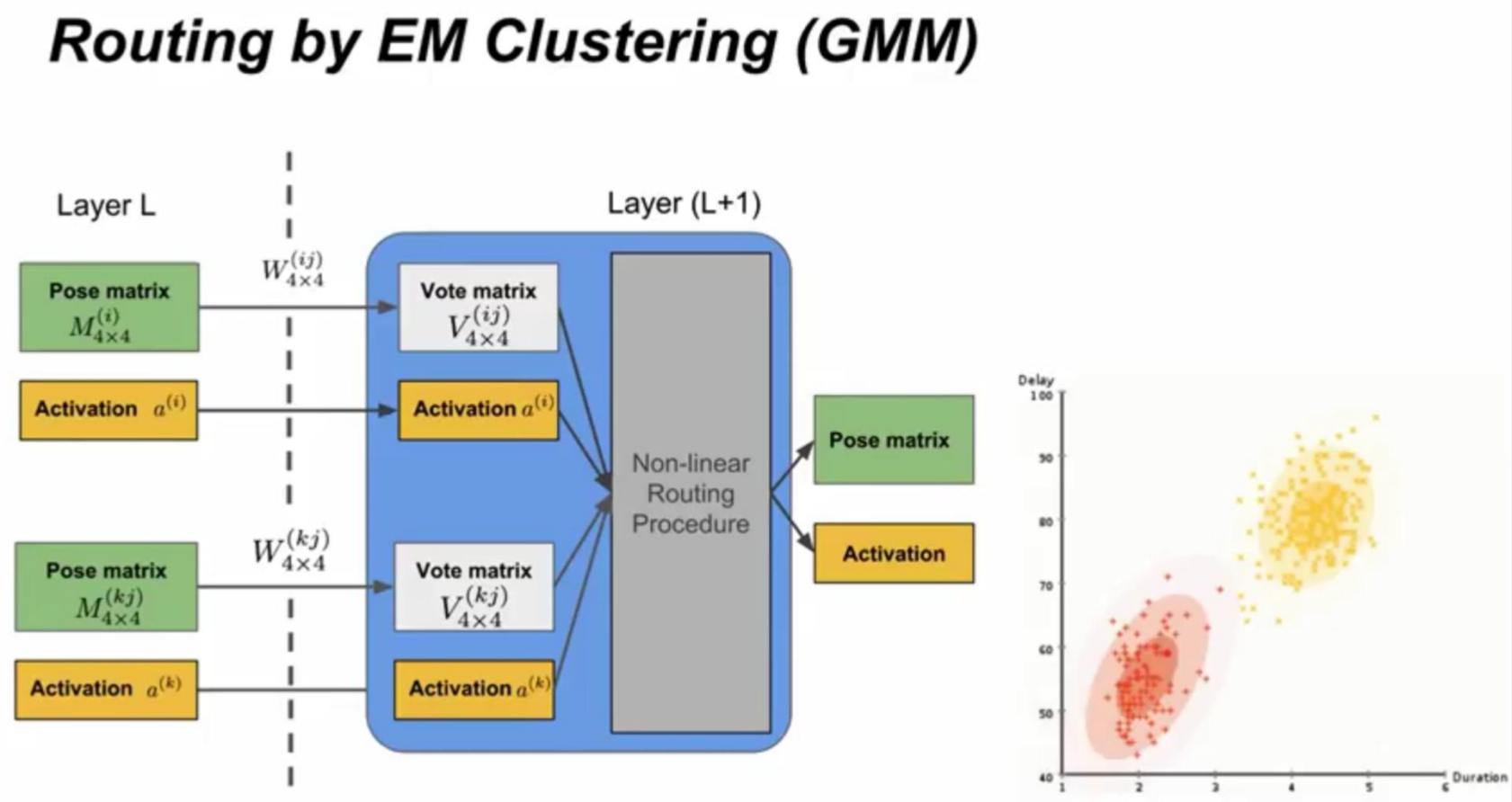
Routing Method

- EM Routing

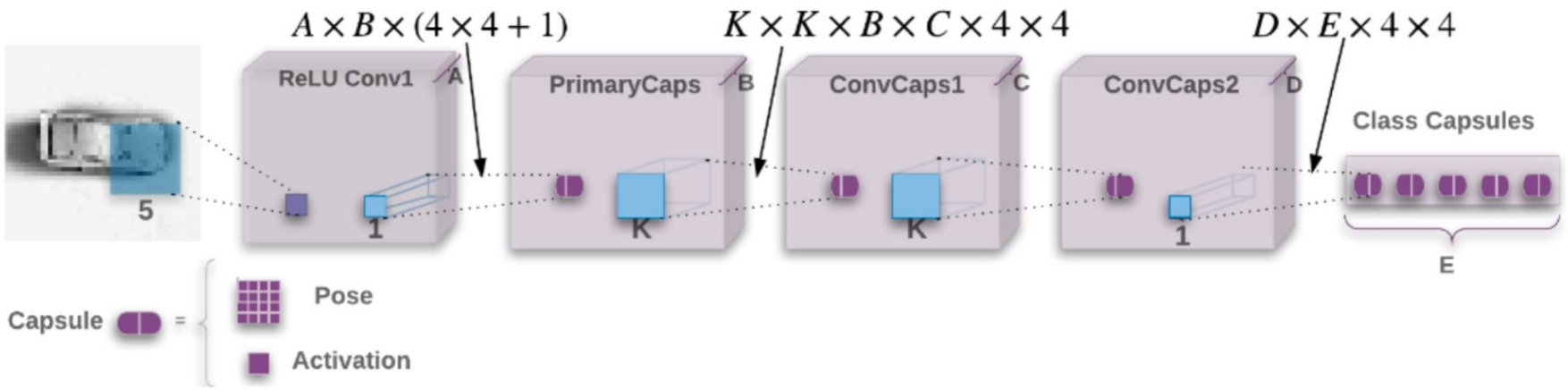


Routing Method

- EM Routing



EM Routing



- Organize pose as 4x4 matrix + activation logit instead of vector. Transformation weights are a 4x4 matrix.
- Primary capsules' poses are learned linear transform of local features. Activation is sigmoid of learned weighted sum of local features.
- Convolutional capsules share transformation weights and see poses from a local kernel.

EM Routing

- Model higher layer as mixture of Gaussians that explains lower layer's poses.

M

- Start with uniform routing priors c_{ij}, weight by the activations of the lower capsules a_i:

$$r_{ij} = c_{ij} a_i$$

- Determine mean and variance:

$$\mu_{jh} = \frac{\sum_i r_{ij} \hat{u}_{ijh}}{\sum_i r_{ij}} \qquad \sigma^2_{jh} = \frac{\sum_i r_{ij} \left(\hat{u}_{ijh} - \mu_{jh}\right)^2}{\sum_i r_{ij}}$$ per pose component h

- Activate upper capsule as:

$$a_j = \text{sigmoid}\left[\lambda\left(\beta_a - \sum_h (\beta_v + \log(\sigma_{jh})) \sum_i r_{ij}\right)\right]$$

β_a, β_v learned by backprop.
λ fixed schedule.

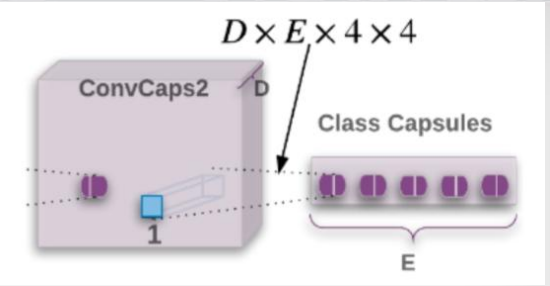
E

- Calculate new routing coefficients:

$$p_{ij} = \frac{1}{\sqrt{2\pi \sum_h \sigma^2_{ijh}}} e^{\frac{-\sum_h (\hat{u}_{ijh} - \mu_{jh})^2}{2\sigma^2_{ijh}}} \qquad c_{ij} = \frac{a_j p_{ij}}{\sum_j a_j p_{ij}}$$

- Iterate 3 times.

Loss



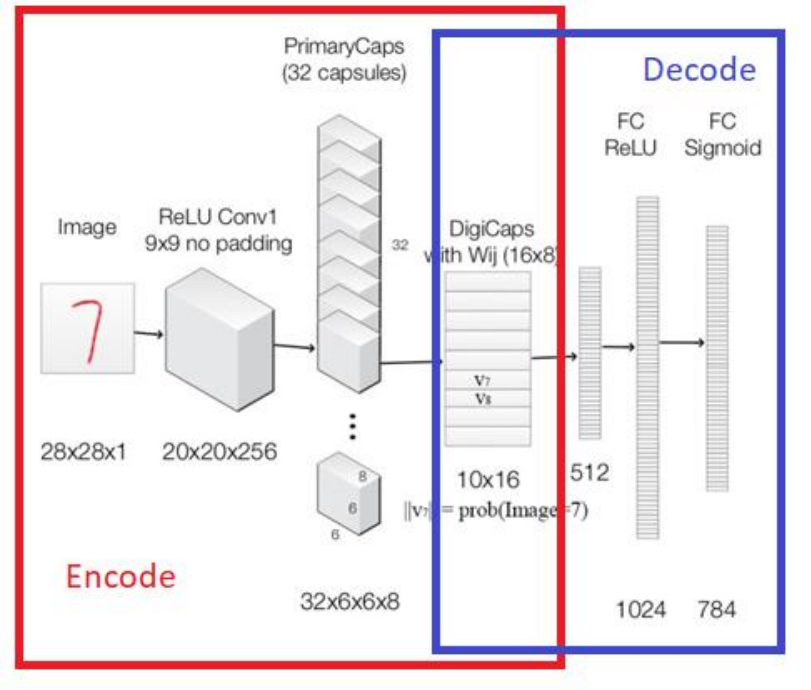
- Connection to class capsules uses coordinate addition scheme
  - Weights shared across locations, like convolutional layer.
  - Explicit (x,y) offset of kernel added to first two elements of pose passed to class capsules.
- Spread loss:

$$L_i = (max(0, m - (a_t - a_i))^2, \quad L = \sum_{i \neq t} L_i$$

- Margin increases linearly from 0.2 to 0.9 during training.

Code Study Dynamic Routing

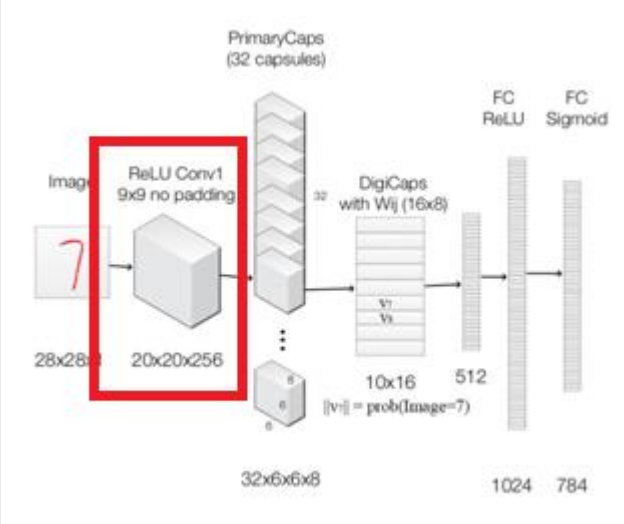
- https://github.com/XifengGuo/CapsNet-Keras




```python
def CapsNet(input_shape, n_class, routings):
    """
    A Capsule Network on MNIST.
    :param input_shape: data shape, 3d, [width, height, channels]
    :param n_class: number of classes
    :param routings: number of routing iterations
    :return: Two Keras Models, the first one used for training, and the second one for evaluation.
            `eval_model` can also be used for training.
    """
    x = layers.Input(shape=input_shape)

    # Layer 1: Just a conventional Conv2D layer
    conv1 = layers.Conv2D(filters=256, kernel_size=9, strides=1, padding='valid', activation='relu', name='conv1')(x)

    # Layer 2: Conv2D layer with `squash` activation, then reshape to [None, num_capsule, dim_capsule]
    primarycaps = PrimaryCap(conv1, dim_capsule=8, n_channels=32, kernel_size=9, strides=2, padding='valid')

    # Layer 3: Capsule layer. Routing algorithm works here.
    digitcaps = CapsuleLayer(num_capsule=n_class, dim_capsule=16, routings=routings,
                             name='digitcaps')(primarycaps)

    # Layer 4: This is an auxiliary layer to replace each capsule with its length. Just to match the true label's shape.
    # If using tensorflow, this will not be necessary. :)
    out_caps = Length(name='capsnet')(digitcaps)

    # Decoder network.
    y = layers.Input(shape=(n_class,))
    masked_by_y = Mask()([digitcaps, y])  # The true label is used to mask the output of capsule layer. For training
    masked = Mask()(digitcaps)  # Mask using the capsule with maximal length. For prediction

    # Shared Decoder model in training and prediction
    decoder = models.Sequential(name='decoder')
    decoder.add(layers.Dense(512, activation='relu', input_dim=16*n_class))
    decoder.add(layers.Dense(1024, activation='relu'))
    decoder.add(layers.Dense(np.prod(input_shape), activation='sigmoid'))
    decoder.add(layers.Reshape(target_shape=input_shape, name='out_recon'))

    # Models for training and evaluation (prediction)
    train_model = models.Model([x, y], [out_caps, decoder(masked_by_y)])
    eval_model = models.Model(x, [out_caps, decoder(masked)])

    # manipulate model
    noise = layers.Input(shape=(n_class, 16))
    noised_digitcaps = layers.Add()([digitcaps, noise])
    masked_noised_y = Mask()([noised_digitcaps, y])
    manipulate_model = models.Model([x, y, noise], decoder(masked_noised_y))
    return train_model, eval_model, manipulate_model
```

Code Study Dynamic Routing

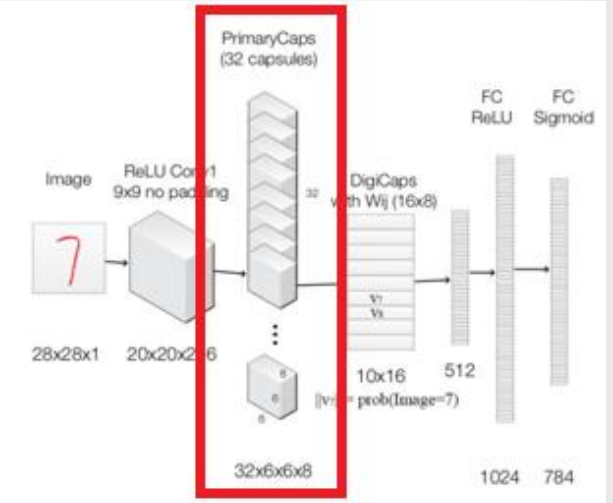
- ReLu Conv1




```python
def CapsNet(input_shape, n_class, routings):
    """
    A Capsule Network on MNIST.
    :param input_shape: data shape, 3d, [width, height, channels]
    :param n_class: number of classes
    :param routings: number of routing iterations
    :return: Two Keras Models, the first one used for training, and the second one for evaluation.
            `eval_model` can also be used for training.
    """
    x = layers.Input(shape=input_shape)

    # Layer 1: Just a conventional Conv2D layer
    conv1 = layers.Conv2D(filters=256, kernel_size=9, strides=1, padding='valid', activation='relu', name='conv1')(x)

    # Layer 2: Conv2D layer with `squash` activation, then reshape to [None, num_capsule, dim_capsule]
    primarycaps = PrimaryCap(conv1, dim_capsule=8, n_channels=32, kernel_size=9, strides=2, padding='valid')

    # Layer 3: Capsule layer. Routing algorithm works here.
    digitcaps = CapsuleLayer(num_capsule=n_class, dim_capsule=16, routings=routings,
                             name='digitcaps')(primarycaps)

    # Layer 4: This is an auxiliary layer to replace each capsule with its length. Just to match the true label's shape.
    # If using tensorflow, this will not be necessary. :)
    out_caps = Length(name='capsnet')(digitcaps)

    # Decoder network.
    y = layers.Input(shape=(n_class,))
    masked_by_y = Mask()([digitcaps, y])  # The true label is used to mask the output of capsule layer. For training
    masked = Mask()(digitcaps)  # Mask using the capsule with maximal length. For prediction

    # Shared Decoder model in training and prediction
    decoder = models.Sequential(name='decoder')
    decoder.add(layers.Dense(512, activation='relu', input_dim=16*n_class))
    decoder.add(layers.Dense(1024, activation='relu'))
    decoder.add(layers.Dense(np.prod(input_shape), activation='sigmoid'))
    decoder.add(layers.Reshape(target_shape=input_shape, name='out_recon'))

    # Models for training and evaluation (prediction)
    train_model = models.Model([x, y], [out_caps, decoder(masked_by_y)])
    eval_model = models.Model(x, [out_caps, decoder(masked)])

    # manipulate model
    noise = layers.Input(shape=(n_class, 16))
    noised_digitcaps = layers.Add()([digitcaps, noise])
    masked_noised_y = Mask()([noised_digitcaps, y])
    manipulate_model = models.Model([x, y, noise], decoder(masked_noised_y))
    return train_model, eval_model, manipulate_model
```

Code Study Dynamic Routing

- PrimaryCaps

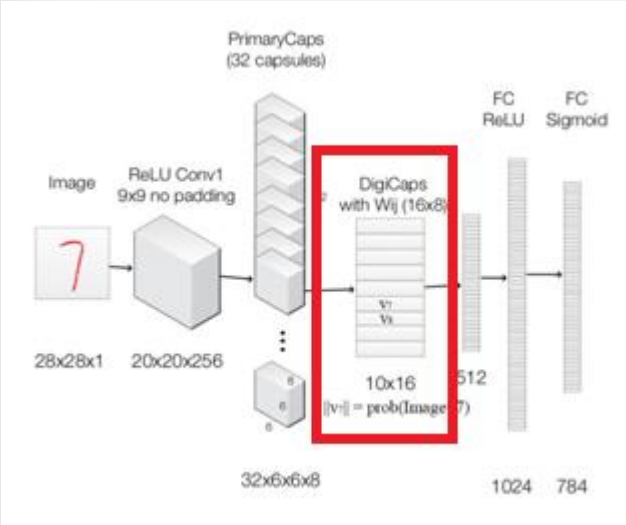



```
def PrimaryCap(inputs, dim_capsule, n_channels, kernel_size, strides, padding):
    """
    Apply Conv2D `n_channels` times and concatenate all capsules
    :param inputs: 4D tensor, shape=[None, width, height, channels]
    :param dim_capsule: the dim of the output vector of capsule
    :param n_channels: the number of types of capsules
    :return: output tensor, shape=[None, num_capsule, dim_capsule]
    """
    output = layers.Conv2D(filters=dim_capsule*n_channels, kernel_size=kernel_size, strides=strides, padding=padding,
                           name='primarycap_conv2d')(inputs)
    outputs = layers.Reshape(target_shape=[-1, dim_capsule], name='primarycap_reshape')(output)
    return layers.Lambda(squash, name='primarycap_squash')(outputs)
```

```
def squash(vectors, axis=-1):
    """
    The non-linear activation used in Capsule. It drives the length of a large vector to near 1 and small vector to 0
    :param vectors: some vectors to be squashed, N-dim tensor
    :param axis: the axis to squash
    :return: a Tensor with same shape as input vectors
    """
    s_squared_norm = K.sum(K.square(vectors), axis, keepdims=True)
    scale = s_squared_norm / (1 + s_squared_norm) / K.sqrt(s_squared_norm + K.epsilon())
    return scale * vectors
```

Code Study Dynamic Routing

- DigiCaps



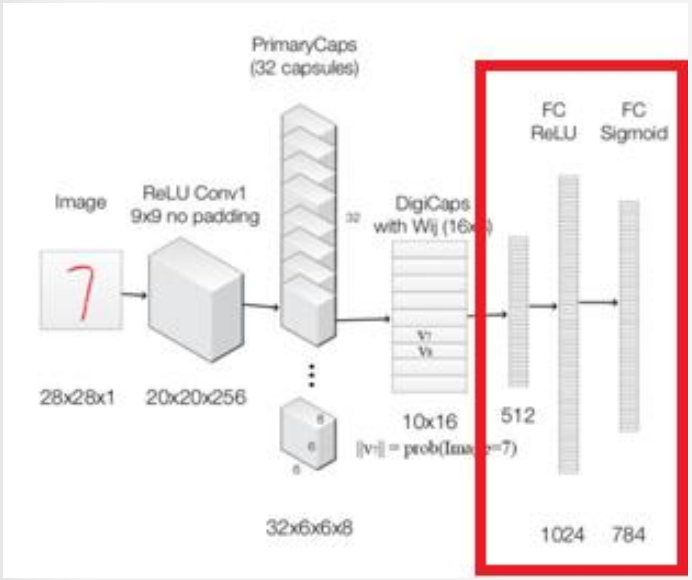
Procedure 1 Routing algorithm.

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: $\mathbf{c}_i \leftarrow \mathrm{softmax}(\mathbf{b}_i)$
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 $(l+1)$: $\mathbf{v}_j \leftarrow \mathrm{squash}(\mathbf{s}_j)$
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

```
# inputs_tiled.shape=[None, num_capsule, input_num_capsule, input_dim_capsule]
inputs_tiled = K.tile(inputs_expand, [1, self.num_capsule, 1, 1])

# Compute `inputs * W` by scanning inputs_tiled on dimension 0.
# x.shape=[num_capsule, input_num_capsule, input_dim_capsule]
# W.shape=[num_capsule, input_num_capsule, dim_capsule, input_dim_capsule]
# Regard the first two dimensions as `batch` dimension,
# then matmul: [input_dim_capsule] x [dim_capsule, input_dim_capsule]^T -> [dim_capsule].
# inputs_hat.shape = [None, num_capsule, input_num_capsule, dim_capsule]
inputs_hat = K.map_fn(lambda x: K.batch_dot(x, self.W, [2, 3]), elems=inputs_tiled)

# Begin: Routing algorithm ---------------------------------------------------------------------#
# The prior for coupling coefficient, initialized as zeros.
# b.shape = [None, self.num_capsule, self.input_num_capsule].
b = tf.zeros(shape=[K.shape(inputs_hat)[0], self.num_capsule, self.input_num_capsule])

assert self.routings > 0, 'The routings should be > 0.'
for i in range(self.routings):
    # c.shape=[batch_size, num_capsule, input_num_capsule]
    c = tf.nn.softmax(b, dim=1)

    # c.shape =  [batch_size, num_capsule, input_num_capsule]
    # inputs_hat.shape=[None, num_capsule, input_num_capsule, dim_capsule]
    # The first two dimensions as `batch` dimension,
    # then matmal: [input_num_capsule] x [input_num_capsule, dim_capsule] -> [dim_capsule].
    # outputs.shape=[None, num_capsule, dim_capsule]
    outputs = squash(K.batch_dot(c, inputs_hat, [2, 2]))  # [None, 10, 16]

    if i < self.routings - 1:
        # outputs.shape =  [None, num_capsule, dim_capsule]
        # inputs_hat.shape=[None, num_capsule, input_num_capsule, dim_capsule]
        # The first two dimensions as `batch` dimension,
        # then matmal: [dim_capsule] x [input_num_capsule, dim_capsule]^T -> [input_num_capsule].
        # b.shape=[batch_size, num_capsule, input_num_capsule]
        b += K.batch_dot(outputs, inputs_hat, [2, 3])
# End: Routing algorithm -----------------------------------------------------------------------#

return outputs
```

Code Study Dynamic Routing

- Decode




```python
def CapsNet(input_shape, n_class, routings):
    """
    A Capsule Network on MNIST.
    :param input_shape: data shape, 3d, [width, height, channels]
    :param n_class: number of classes
    :param routings: number of routing iterations
    :return: Two Keras Models, the first one used for training, and the second one for evaluation.
            `eval_model` can also be used for training.
    """
    x = layers.Input(shape=input_shape)

    # Layer 1: Just a conventional Conv2D layer
    conv1 = layers.Conv2D(filters=256, kernel_size=9, strides=1, padding='valid', activation='relu', name='conv1')(x)

    # Layer 2: Conv2D layer with `squash` activation, then reshape to [None, num_capsule, dim_capsule]
    primarycaps = PrimaryCap(conv1, dim_capsule=8, n_channels=32, kernel_size=9, strides=2, padding='valid')

    # Layer 3: Capsule layer. Routing algorithm works here.
    digitcaps = CapsuleLayer(num_capsule=n_class, dim_capsule=16, routings=routings,
                             name='digitcaps')(primarycaps)

    # Layer 4: This is an auxiliary layer to replace each capsule with its length. Just to match the true label's shape.
    # If using tensorflow, this will not be necessary. :)
    out_caps = Length(name='capsnet')(digitcaps)

    # Decoder network.
    y = layers.Input(shape=(n_class,))
    masked_by_y = Mask()([digitcaps, y])  # The true label is used to mask the output of capsule layer. For training
    masked = Mask()(digitcaps)  # Mask using the capsule with maximal length. For prediction

    # Shared Decoder model in training and prediction
    decoder = models.Sequential(name='decoder')
    decoder.add(layers.Dense(512, activation='relu', input_dim=16*n_class))
    decoder.add(layers.Dense(1024, activation='relu'))
    decoder.add(layers.Dense(np.prod(input_shape), activation='sigmoid'))
    decoder.add(layers.Reshape(target_shape=input_shape, name='out_recon'))

    # Models for training and evaluation (prediction)
    train_model = models.Model([x, y], [out_caps, decoder(masked_by_y)])
    eval_model = models.Model(x, [out_caps, decoder(masked)])

    # manipulate model
    noise = layers.Input(shape=(n_class, 16))
    noised_digitcaps = layers.Add()([digitcaps, noise])
    masked_noised_y = Mask()([noised_digitcaps, y])
    manipulate_model = models.Model([x, y, noise], decoder(masked_noised_y))
```

Code Study Dynamic Routing

- Training Loss
  - Margin Loss

```
def margin_loss(y_true, y_pred):
    """
    Margin loss for Eq.(4). When y_true[i, :] contains not just one `1`, this loss should work too. Not test it.
    :param y_true: [None, n_classes]
    :param y_pred: [None, num_capsule]
    :return: a scalar loss value.
    """
    L = y_true * K.square(K.maximum(0., 0.9 - y_pred)) + \
        0.5 * (1 - y_true) * K.square(K.maximum(0., y_pred - 0.1))

    return K.mean(K.sum(L, 1))
```

In the paper:

$m^- = 0.1$

$m^+ = 0.9$

$\lambda = 0.5$

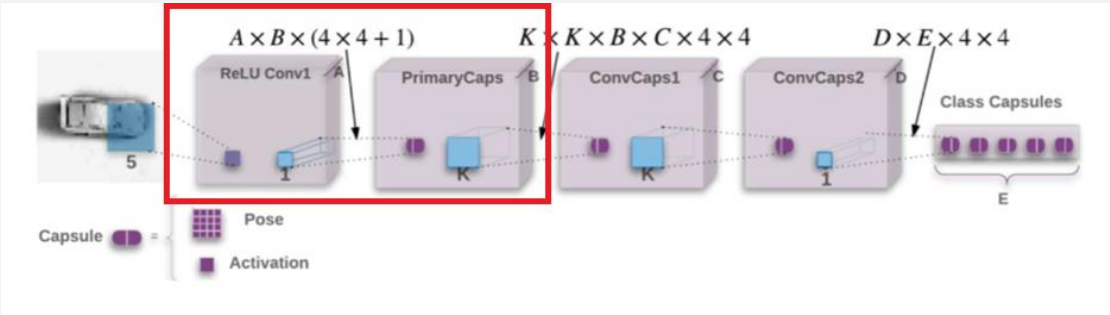
Loss function (Margin loss)

In our example, we want to detect multiple digits in a picture. Capsules use a separate margin loss L_c for each category c digit present in the picture:

$$L_c = T_c \max(0, m^+ - \|v_c\|)^2 + \lambda(1 - T_c)\max(0, \|v_c\| - m^-)^2$$

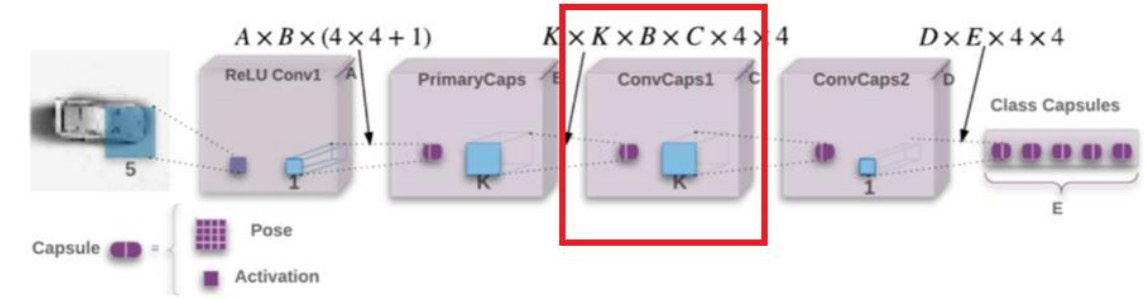
Code Study EM Routing

- https://github.com/www0wwwjs1/Matrix-Capsules-EM-Tensorflow



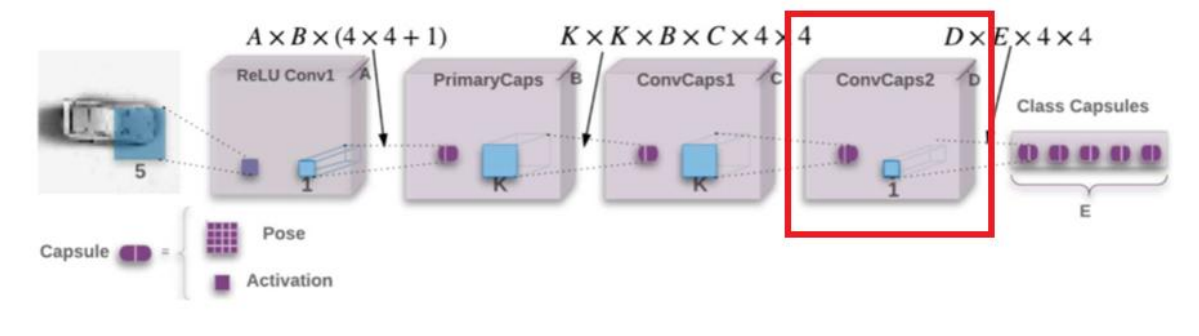

```
with tf.variable_scope('relu_conv1') as scope:
    output = slim.conv2d(input, num_outputs=cfg.A, kernel_size=[
                         5, 5], stride=2, padding='VALID', scope=scope, activation_fn=tf.nn.relu)
    data_size = int(np.floor((data_size - 4) / 2))

    assert output.get_shape() == [cfg.batch_size, data_size, data_size, cfg.A]
    tf.logging.info('conv1 output shape: {}'.format(output.get_shape()))

with tf.variable_scope('primary_caps') as scope:
    pose = slim.conv2d(output, num_outputs=cfg.B * 16,
                       kernel_size=[1, 1], stride=1, padding='VALID', scope=scope, activation_fn=None)
    activation = slim.conv2d(output, num_outputs=cfg.B, kernel_size=[
                             1, 1], stride=1, padding='VALID', scope='primary_caps/activation', activation_fn=tf.nn.sigmoid)
    pose = tf.reshape(pose, shape=[cfg.batch_size, data_size, data_size, cfg.B, 16])
    activation = tf.reshape(
        activation, shape=[cfg.batch_size, data_size, data_size, cfg.B, 1])
    output = tf.concat([pose, activation], axis=4)
    output = tf.reshape(output, shape=[cfg.batch_size, data_size, data_size, -1])
    assert output.get_shape() == [cfg.batch_size, data_size, data_size, cfg.B * 17]
    tf.logging.info('primary capsule output shape: {}'.format(output.get_shape()))
```

Code Study EM Routing



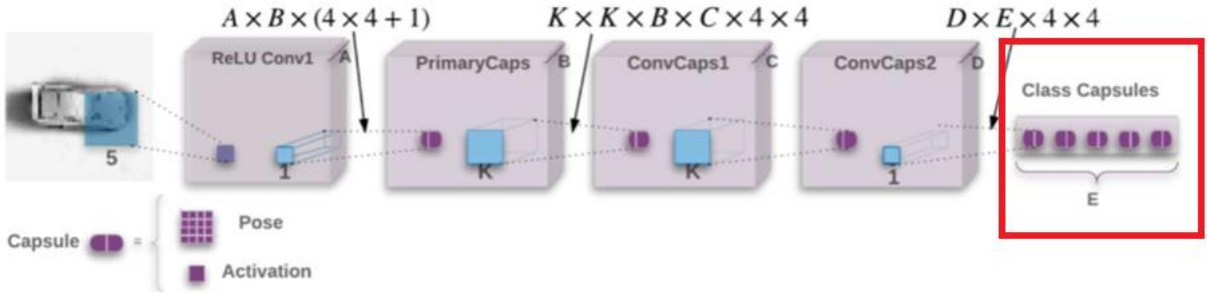

```
with tf.variable_scope('conv_caps1') as scope:
    output = kernel_tile(output, 3, 2)
    data_size = int(np.floor((data_size - 2) / 2))
    output = tf.reshape(output, shape=[cfg.batch_size *
                                       data_size * data_size, 3 * 3 * cfg.B, 17])
    activation = tf.reshape(output[:, :, 16], shape=[
                            cfg.batch_size * data_size * data_size, 3 * 3 * cfg.B, 1])

    with tf.variable_scope('v') as scope:
        votes = mat_transform(output[:, :, :16], cfg.C, weights_regularizer, tag=True)
        tf.logging.info('conv cap 1 votes shape: {}'.format(votes.get_shape()))

    with tf.variable_scope('routing') as scope:
        miu, activation, _ = em_routing(votes, activation, cfg.C, weights_regularizer)
        tf.logging.info('conv cap 1 miu shape: {}'.format(miu.get_shape()))
        tf.logging.info('conv cap 1 activation before reshape: {}'.format(
            activation.get_shape()))

    pose = tf.reshape(miu, shape=[cfg.batch_size, data_size, data_size, cfg.C, 16])
    tf.logging.info('conv cap 1 pose shape: {}'.format(pose.get_shape()))
    activation = tf.reshape(
        activation, shape=[cfg.batch_size, data_size, data_size, cfg.C, 1])
    tf.logging.info('conv cap 1 activation after reshape: {}'.format(
        activation.get_shape()))
    output = tf.reshape(tf.concat([pose, activation], axis=4), [
                        cfg.batch_size, data_size, data_size, -1])
    tf.logging.info('conv cap 1 output shape: {}'.format(output.get_shape()))
```

Code Study EM Routing




```
with tf.variable_scope('conv_caps2') as scope:
    output = kernel_tile(output, 3, 1)
    data_size = int(np.floor((data_size - 2) / 1))
    output = tf.reshape(output, shape=[cfg.batch_size *
                                       data_size * data_size, 3 * 3 * cfg.C, 17])
    activation = tf.reshape(output[:, :, 16], shape=[
                            cfg.batch_size * data_size * data_size, 3 * 3 * cfg.C, 1])

    with tf.variable_scope('v') as scope:
        votes = mat_transform(output[:, :, :16], cfg.D, weights_regularizer)
        tf.logging.info('conv cap 2 votes shape: {}'.format(votes.get_shape()))

    with tf.variable_scope('routing') as scope:
        miu, activation, _ = em_routing(votes, activation, cfg.D, weights_regularizer)

    pose = tf.reshape(miu, shape=[cfg.batch_size * data_size * data_size, cfg.D, 16])
    tf.logging.info('conv cap 2 pose shape: {}'.format(votes.get_shape()))
    activation = tf.reshape(
        activation, shape=[cfg.batch_size * data_size * data_size, cfg.D, 1])
    tf.logging.info('conv cap 2 activation shape: {}'.format(activation.get_shape()))
```

Code Study EM Routing




```
with tf.variable_scope('class_caps') as scope:
    with tf.variable_scope('v') as scope:
        votes = mat_transform(pose, num_classes, weights_regularizer)

        assert votes.get_shape() == [cfg.batch_size * data_size *
                                     data_size, cfg.D, num_classes, 16]
        tf.logging.info('class cap votes original shape: {}'.format(votes.get_shape()))

        coord_add = np.reshape(coord_add, newshape=[data_size * data_size, 1, 1, 2])
        coord_add = np.tile(coord_add, [cfg.batch_size, cfg.D, num_classes, 1])
        coord_add_op = tf.constant(coord_add, dtype=tf.float32)

        votes = tf.concat([coord_add_op, votes], axis=3)
        tf.logging.info('class cap votes coord add shape: {}'.format(votes.get_shape()))

    with tf.variable_scope('routing') as scope:
        miu, activation, test2 = em_routing(
            votes, activation, num_classes, weights_regularizer)
        tf.logging.info(
            'class cap activation shape: {}'.format(activation.get_shape()))
        tf.summary.histogram(name="class_cap_routing_hist",
                             values=test2)

    output = tf.reshape(activation, shape=[
                        cfg.batch_size, data_size, data_size, num_classes])
```

Code Study EM Routing

- Loss Function

```
# spread loss
output1 = tf.reshape(output, shape=[cfg.batch_size, 1, num_class])
y = tf.expand_dims(y, axis=2)
at = tf.matmul(output1, y)
"""Paper eq(5)."""
loss = tf.square(tf.maximum(0., m - (at - output1)))
loss = tf.matmul(loss, 1. - y)
loss = tf.reduce_mean(loss)

# reconstruction loss
# pose_out = tf.reshape(tf.matmul(pose_out, y, transpose_a=True), shape=[cfg.batch_size, -1])
pose_out = tf.reshape(tf.multiply(pose_out, y), shape=[cfg.batch_size, -1])
tf.logging.info("decoder input value dimension:{}".format(pose_out.get_shape()))

with tf.variable_scope('decoder'):
    pose_out = slim.fully_connected(pose_out, 512, trainable=True, weights_regularizer=tf.contrib.layers.l2_regularizer(5e-04))
    pose_out = slim.fully_connected(pose_out, 1024, trainable=True, weights_regularizer=tf.contrib.layers.l2_regularizer(5e-04))
    pose_out = slim.fully_connected(pose_out, data_size * data_size,
                                    trainable=True, activation_fn=tf.sigmoid, weights_regularizer=tf.contrib.layers.l2_regularizer(5e-04))

    x = tf.reshape(x, shape=[cfg.batch_size, -1])
    reconstruction_loss = tf.reduce_mean(tf.square(pose_out - x))

if cfg.weight_reg:
    # regularization loss
    regularization = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
    # loss+0.0005*reconstruction_loss+regularization#
    loss_all = tf.add_n([loss] + [0.0005 * data_size* data_size * reconstruction_loss] + regularization)
else:
    loss_all = tf.add_n([loss] + [0.0005 * data_size* data_size * reconstruction_loss])
```

Code Study EM Routing

- EM Routing

Procedure 1 Routing algorithm returns **activation** and **pose** of the capsules in layer $L+1$ given the **activations** and **votes** of capsules in layer L. V_{ij}^h is the h^{th} dimension of the vote from capsule i with activation a_i in layer L to capsule j in layer $L+1$. β_a, β_v are learned discriminatively and the inverse temperature λ increases at each iteration with a fixed schedule.

procedure EM ROUTING($\boldsymbol{a}, V$)
  $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
  for t iterations **do**
    $\forall j \in \Omega_{L+1}$: M-STEP($\boldsymbol{a}, R, V, j$)
    $\forall i \in \Omega_L$: E-STEP($\mu, \sigma, \boldsymbol{a}, V, i$)
  return $\boldsymbol{a}, M$

procedure M-STEP($\boldsymbol{a}, R, V, j$) ▷ for one higher-level capsule
  $\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * \boldsymbol{a}_i$
  $\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
  $\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} (V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
  $cost^h \leftarrow (\beta_v + log(\sigma_j^h)) \sum_i R_{ij}$
  $a_j \leftarrow sigmoid(\lambda(\beta_a - \sum_h cost^h))$

procedure E-STEP($\mu, \sigma, \boldsymbol{a}, V, i$) ▷ for one lower-level capsule
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{p}_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} e^{-\sum_h^H \frac{(V_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}}$
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{R}_{ij} \leftarrow \frac{\boldsymbol{a}_j p_j}{\sum_{u \in \Omega_{L+1}} \boldsymbol{a}_u p_u}$

```
for iters in range(cfg.iter_routing):
    # if iters == cfg.iter_routing-1:

    # e-step
    if iters == 0:
        r = tf.constant(np.ones([batch_size, caps_num_i, caps_num_c], dtype=np.float32) / caps_num_c)
    else:
        # Contributor: Yunzhi Shi
        # log and exp here provide higher numerical stability especially for bigger number of iterations
        log_p_c_h = -tf.log(tf.sqrt(sigma_square)) - \
                    (tf.square(votes_in - miu) / (2 * sigma_square))
        log_p_c_h = log_p_c_h - \
                    (tf.reduce_max(log_p_c_h, axis=[2, 3], keep_dims=True) - tf.log(10.0))
        p_c = tf.exp(tf.reduce_sum(log_p_c_h, axis=3))

        ap = p_c * tf.reshape(activation_out, shape=[batch_size, 1, caps_num_c])

        # ap = tf.reshape(activation_out, shape=[batch_size, 1, caps_num_c])

        r = ap / (tf.reduce_sum(ap, axis=2, keep_dims=True) + cfg.epsilon)

    # m-step
    r = r * activation_in
    r = r / (tf.reduce_sum(r, axis=2, keep_dims=True)+cfg.epsilon)

    r_sum = tf.reduce_sum(r, axis=1, keep_dims=True)
    r1 = tf.reshape(r / (r_sum + cfg.epsilon),
                    shape=[batch_size, caps_num_i, caps_num_c, 1])

    miu = tf.reduce_sum(votes_in * r1, axis=1, keep_dims=True)
    sigma_square = tf.reduce_sum(tf.square(votes_in - miu) * r1,
                                 axis=1, keep_dims=True) + cfg.epsilon

    if iters == cfg.iter_routing-1:
        r_sum = tf.reshape(r_sum, [batch_size, caps_num_c, 1])
        cost_h = (beta_v + tf.log(tf.sqrt(tf.reshape(sigma_square,
                                                     shape=[batch_size, caps_num_c, n_channels])))) * r_sum

        activation_out = tf.nn.softmax(cfg.ac_lambda0 * (beta_a - tf.reduce_sum(cost_h, axis=2)))
    else:
        activation_out = tf.nn.softmax(r_sum)
```

Discuss

- Proposed Methods proposed still using CNN to extract features.
- And then apply routing algorithm to learn part-whole relationships.
- Not solve all invariance problem, Proposed methods are Viewpoint-Invarient.
- Not Solution for One-Shot or Zero Shot Learning.
- Maybe need to add label for features.
- Problem now assume objects are Rigid and maybe cause problem while applying on flexible object that maybe good at CNN treating as invariance.

Discuss

- Maybe add flexible or joint consideration in the following version.
- It’s high computing consuming algorithm which now only apply on toy problem.
- It is clammed to reduce attack for image classification compared to CNN.
- The Capsule Network is an idea and maybe more methods will be proposed.