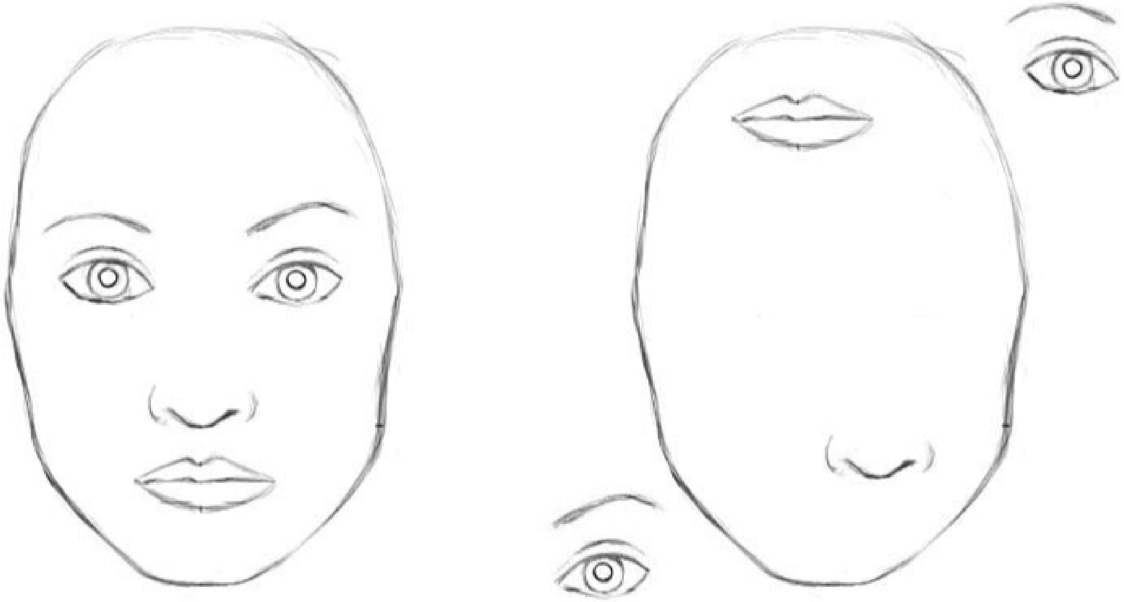
Capsules tutorial

Cher Bass
cher.bass@kcl.ac.uk
METRICS lab @ King’s College

CNNs have drawbacks

- CNNs can learn low (edges, colours) and high level (mouth, nose, eyes) features
- Orientational and relative spatial relationships between features is not captured
- i.e. higher level features don’t encode for pose (translation and rotation)
- Max pooling loses valuable information
- Requires large amounts of data and augmentation to learn



https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

Ideas behind capsules

1. Can capture spatial relationships between objects/ features
   - *Using high dimensional “W Matrix” to encode these relationships*
   - *Translation invariant*
   - *Known to need less data*
2. Can group these features into “capsules”
   - *Using “dynamic routing”*
   - *Routes features on the fly*
   - *Capsules encode for features closely related in feature space*

Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." *Advances in Neural Information Processing Systems*. 2017.

Capsules capture pose

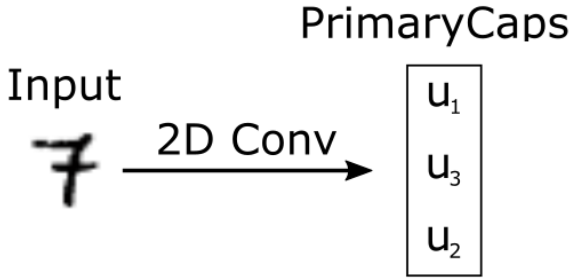
https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

Capsules capture pose

Hinton, Geoffrey E., Sara Sabour, and Nicholas Frosst. "Matrix capsules with EM routing." (2018)

How do capsules work?

Dynamic routing between capsules



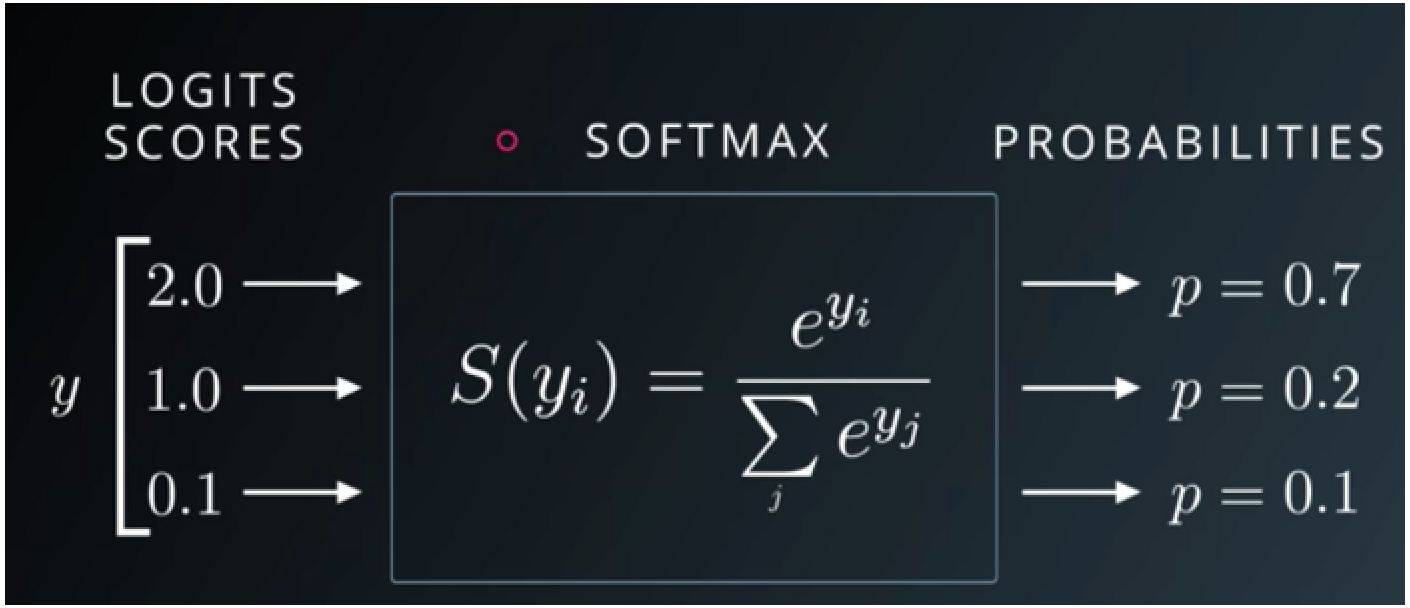
Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." *Advances in Neural Information Processing Systems*. 2017.

Dynamic 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: $\mathbf{c}_i \leftarrow \boxed{\mathtt{softmax}(\mathbf{b}_i)}$ ▷ `softmax` computes Eq. 3
        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 $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
        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 $\mathbf{v}_j$

Softmax


LOGITS
SCORES
SOFTMAX
PROBABILITIES
y
2.0
1.0
0.1
$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$
$p = 0.7$
$p = 0.2$
$p = 0.1$

Dynamic 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: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3

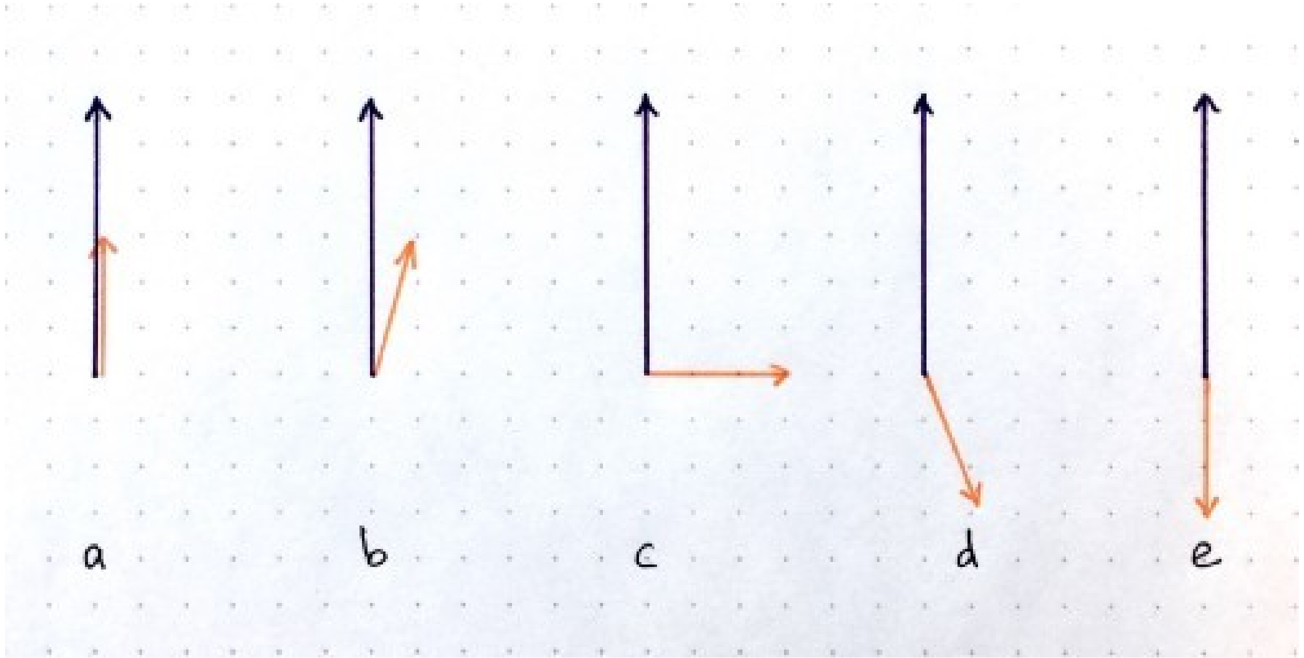
    for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \boxed{\sum_i c_{ij} \hat{\mathbf{u}}_{j|i}}$

    for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1

    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 $\mathbf{v}_j$

Dot product



Dynamic routing

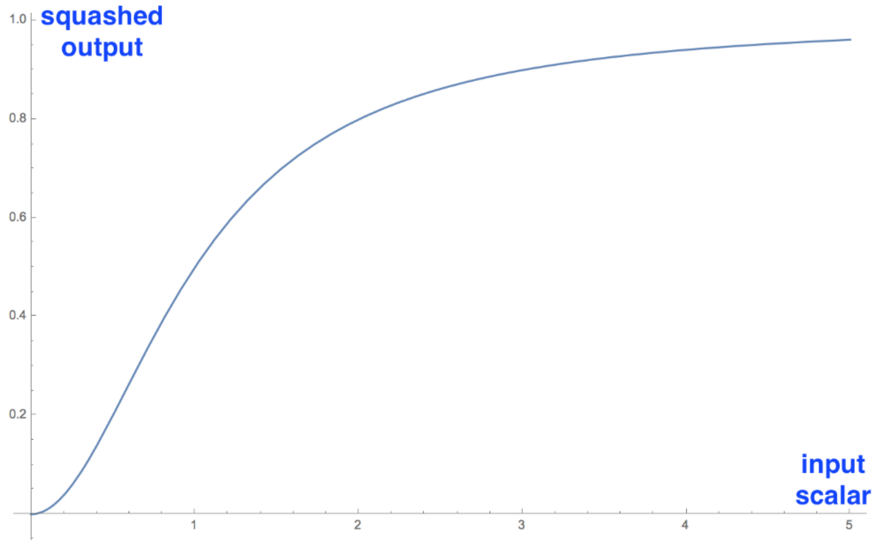
Procedure 1 Routing algorithm.

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: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
        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 $(l+1)$: $\mathbf{v}_j \leftarrow \boxed{\mathtt{squash}(\mathbf{s}_j)}$ ▷ `squash` computes Eq. 1
        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 $\mathbf{v}_j$

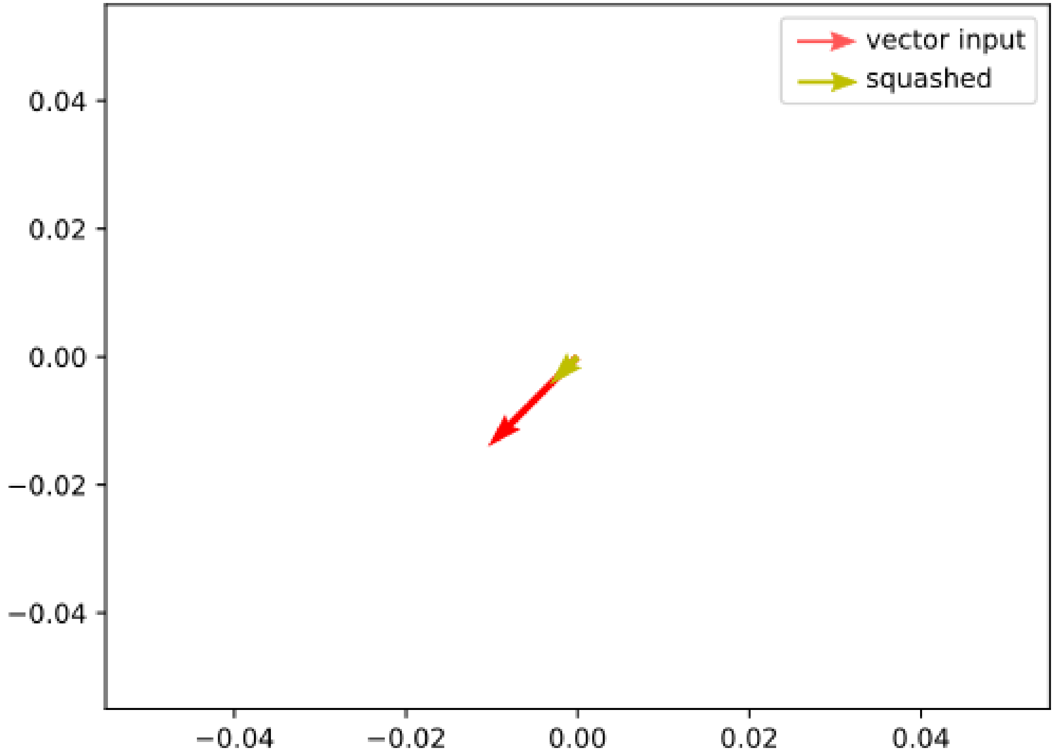
What is the purpose of the squash function?

- Nonlinear activation
- Normalises the length between 0,1
- Does not change the direction of the vector
- This will allow the next step of the dynamic routing to not be affected by the nonlinear layer, as the direction of the vector is not changed

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



The effect of squash function on a vector

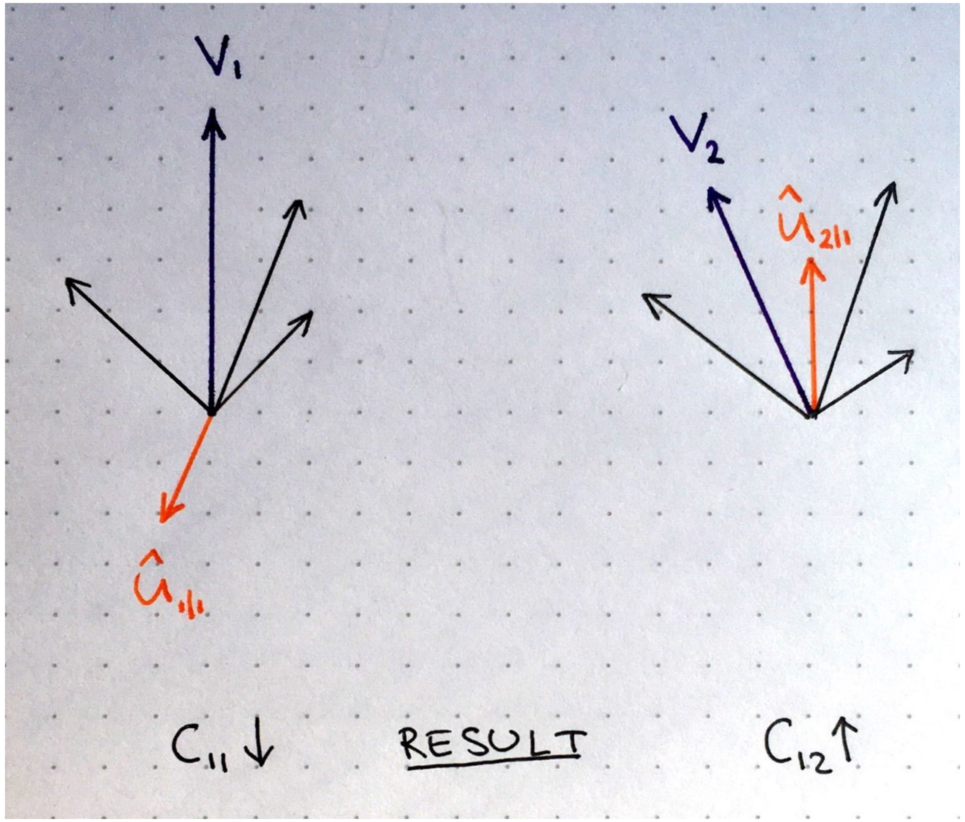


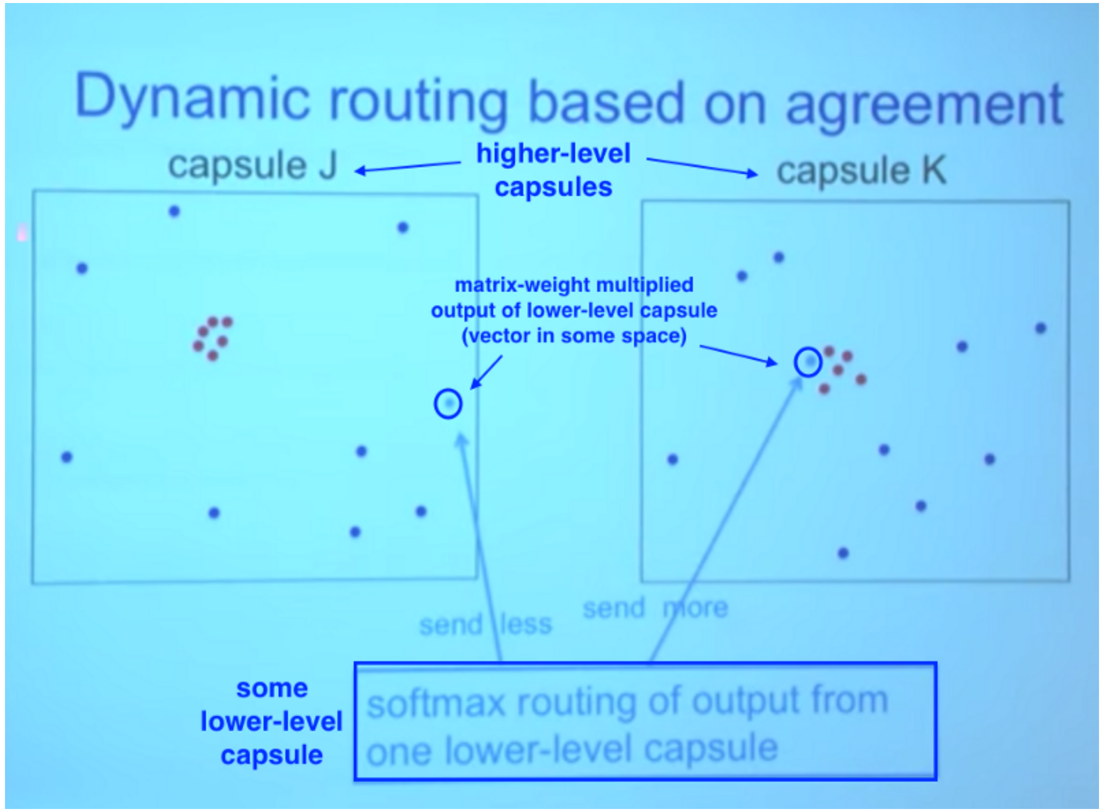
Dynamic routing

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: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
        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 $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
        for all capsule i in layer l and capsule j in layer $(l+1)$: $\boxed{b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j}$
    return $\mathbf{v}_j$

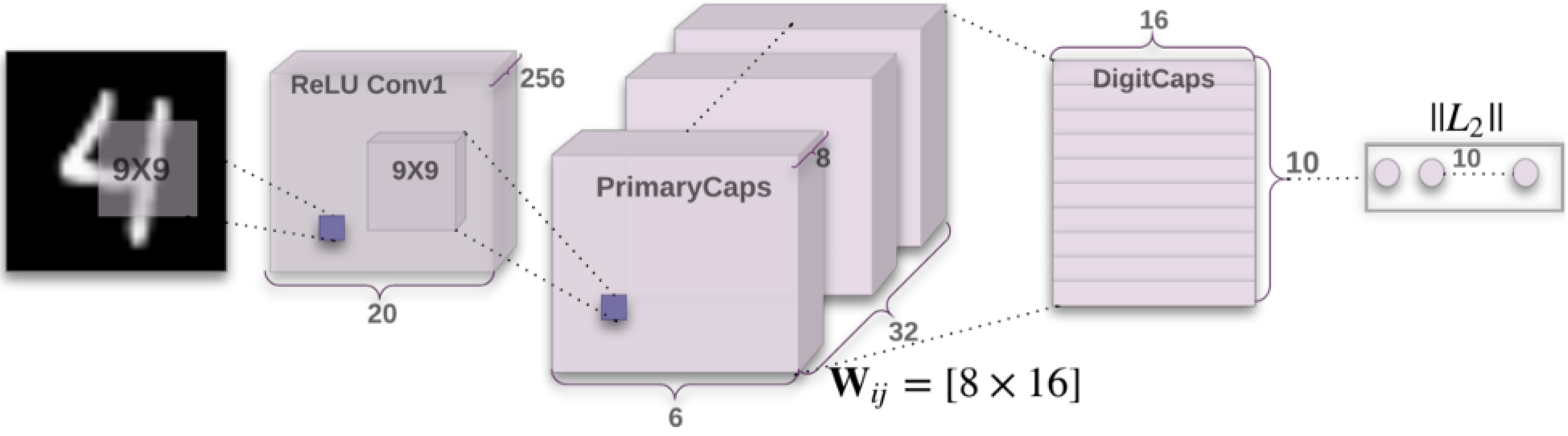
Dynamic routing - update step



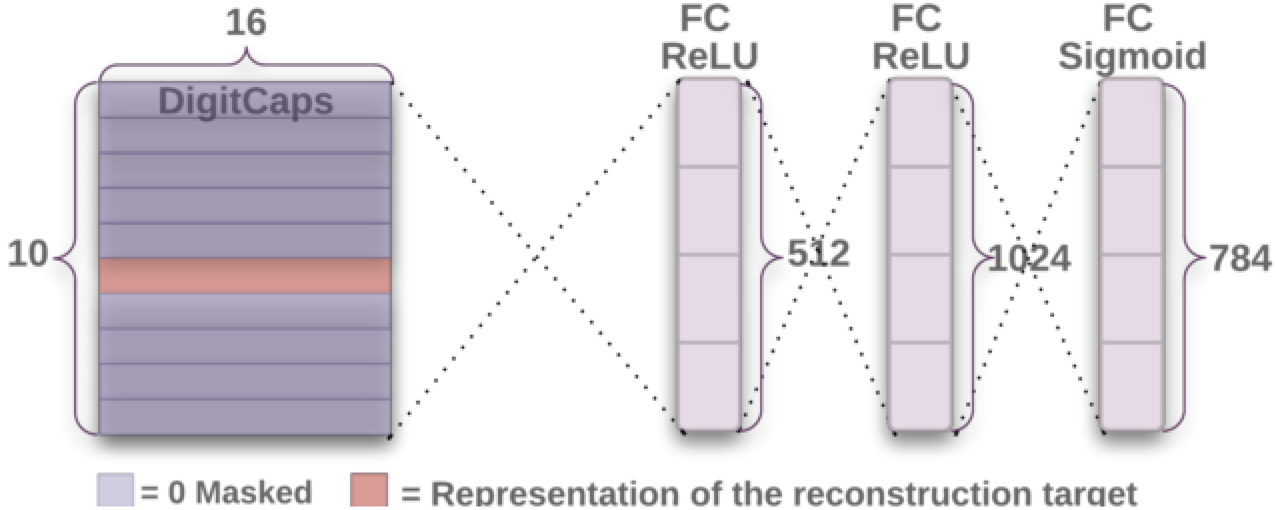


CapsNet Architecture

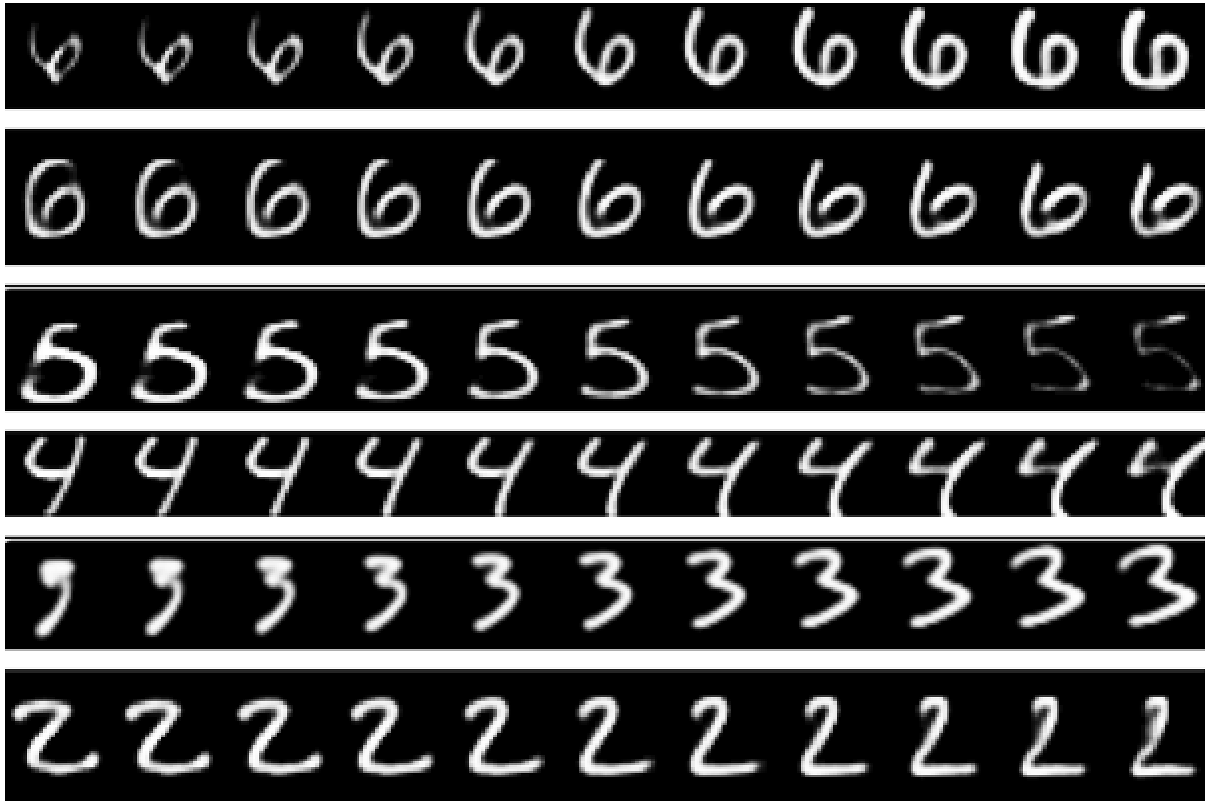
CapsNet architecture



CapsNet architecture



Capsules learn representations



Loss function

CapsNet Loss Function

calculated for correct DigitCap

calculated for incorrect DigitCaps

loss term for one DigitCap

L2 norm

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

1 when correct DigitCap, 0 when incorrect

zero loss when correct prediction with probability greater than 0.9, non-zero otherwise

0.5 constant used for numerical stability

1 when incorrect DigitCap, 0 when correct

zero loss when incorrect prediction with probability less than 0.1, non-zero otherwise

Note: correct DigitCap is one that matches training label, for each training example there will be 1 correct and 9 incorrect DigitCaps

Convolutional Capsules

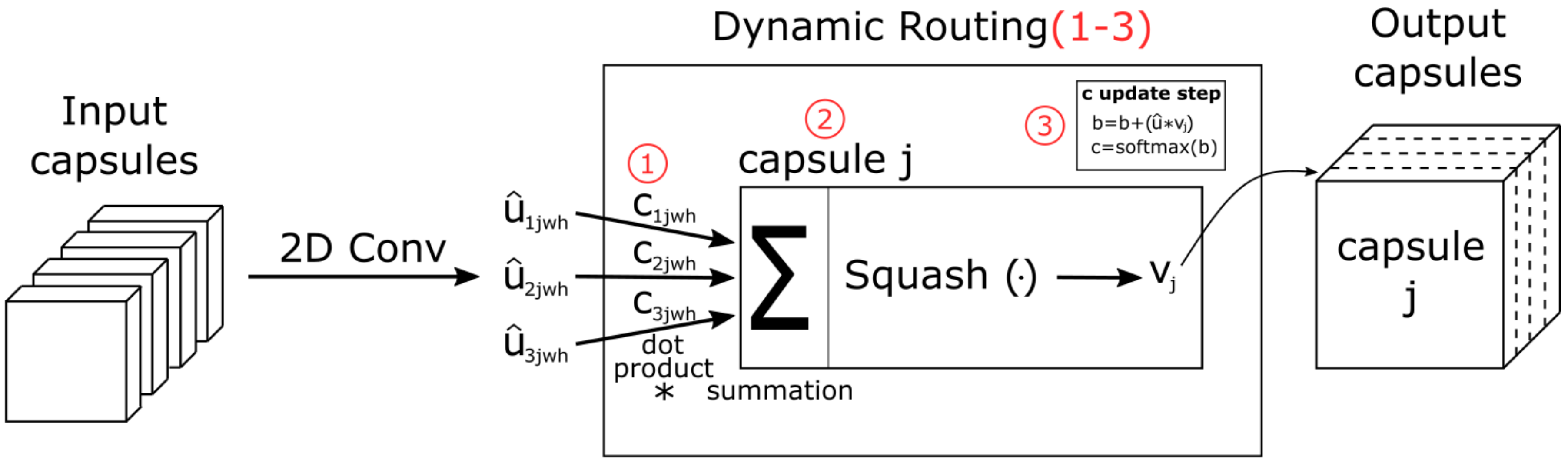
The original capsules were not used much in the literature due to high computational cost and slow training

- W matrix multiplication (high dimensional matrix) – high memory requirement
- Dynamic routing – slow to train
- Was only applied to small images (28x28)

Convolutional capsules:

- Reduced computation
- Can be applied to larger images
- Allow for image-to-image tasks
- Spatial filters allows analysis of features

Convolutional Capsules



LaLonde, Rodney, and Ulas Bagci. "Capsules for Object Segmentation." *arXiv preprint arXiv:1804.04241.* 2018.

Convolutional Capsules

Algorithm 1: Convolutional Capsules + Dynamic Routing

Input: a, capsules in layer l; l, layer; r, iterations; bias; weight

Output: v_j, capsules in layer $(l+1)$

$\boxed{\hat{u}_{i,ch_i,j,ch_j} \leftarrow bias_{j \times ch_j} + \sum_{n=0}^{ch_i} weight_{j \times ch_j, n} * a_n}$

for all capsules i in layer l and capsules j in layer $(l+1)$: $\boxed{b_{ij} \leftarrow 0}$

b shape = [in_caps, width, height, out_caps]

for *1* **to** r **do**

- for all capsules i in layer l and capsule j in layer $(l+1)$: $c_{ij} \leftarrow softmax(b_{ij})$ ▷ Eq. 4
- for all capsules j in layer $(l+1)$: $s_j \leftarrow \sum_i c_{ij}\hat{u}_{ij}$
- for all capsules j in layer $(l+1)$: $v_j \leftarrow squash(s_j)$ ▷ Eq. 5
- for all capsules i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{u}_{ij} \cdot v_j$

end

LaLonde, Rodney, and Ulas Bagci. "Capsules for Object Segmentation." *arXiv preprint arXiv:1804.04241.* 2018.

Capsule applicatic

- Segmentation
- Conditional image synthesis
- Classification (MNIST)

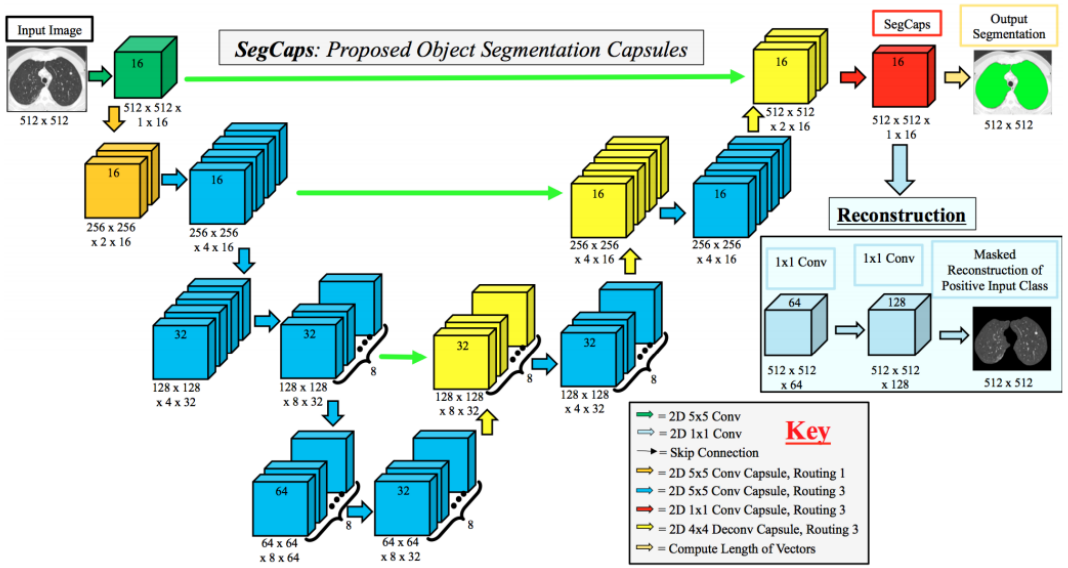
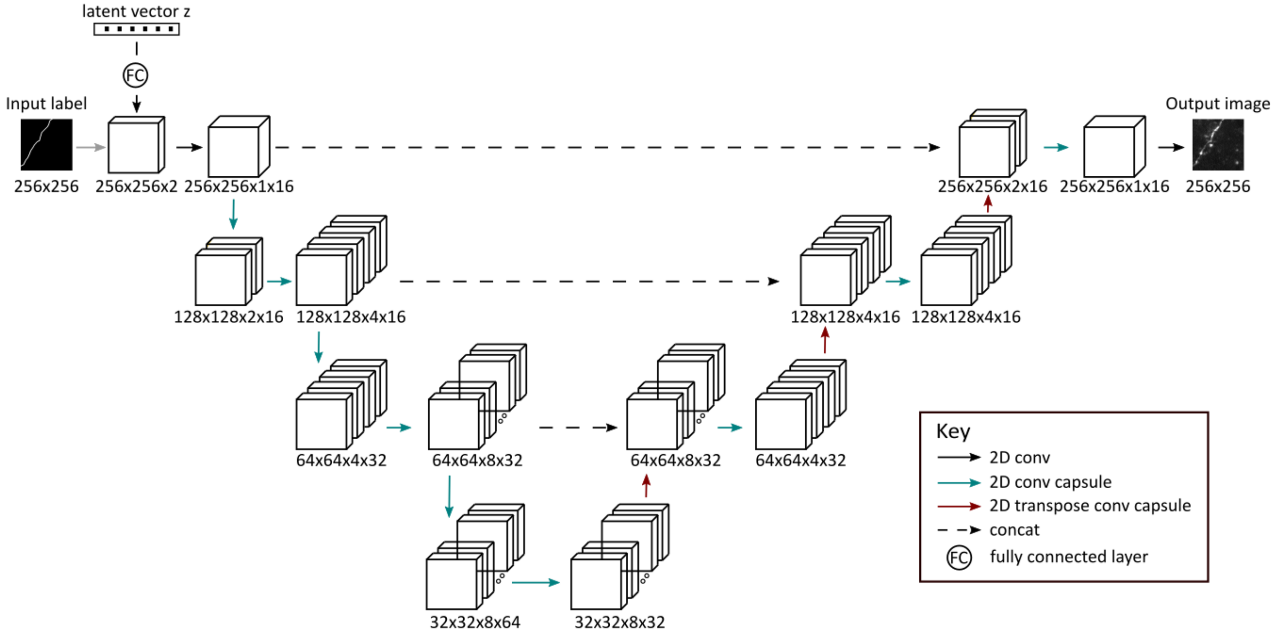


Figure 2: The proposed **SegCaps** architecture for object segmentation.

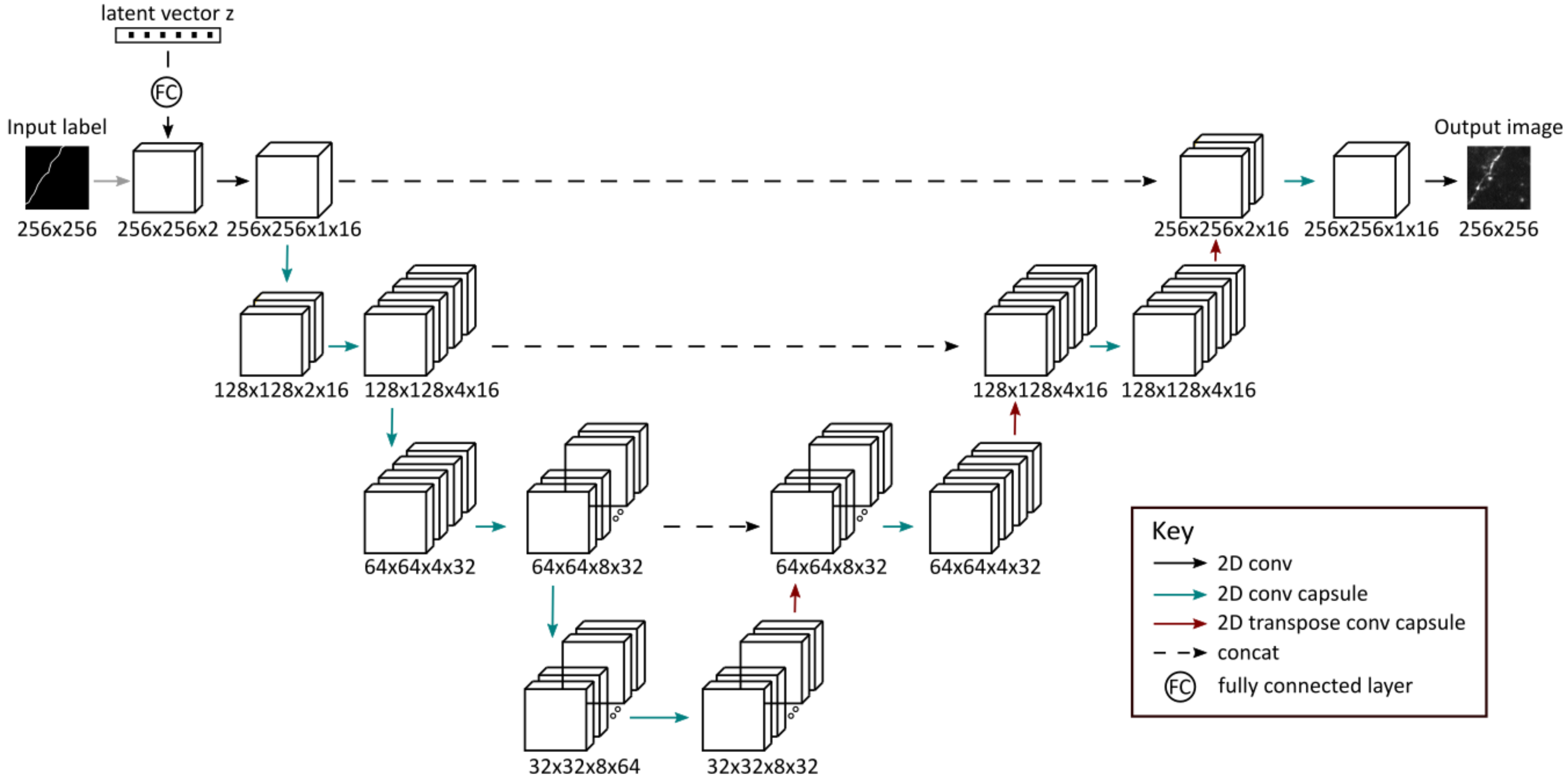
LaLonde, Rodney, and Ulas Bagci. "Capsules for Object Segmentation." *arXiv preprint arXiv:1804.04241.* 2018.



Bass, Cher, et al. "Image synthesis with a convolutional capsule generative adversarial network." *International Conference on Medical Imaging with Deep Learning*. 2019.

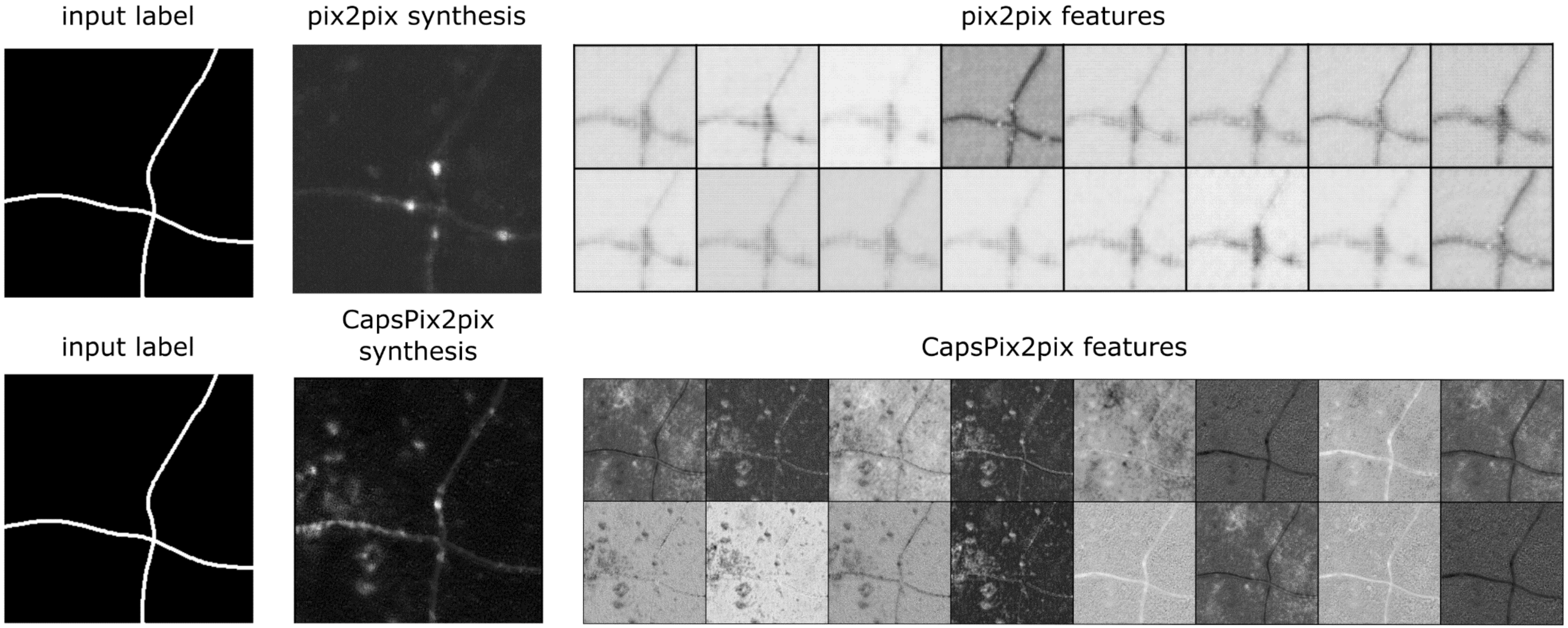
Capsules for Object Segmentation

Image synthesis with a convolutional capsule GAN

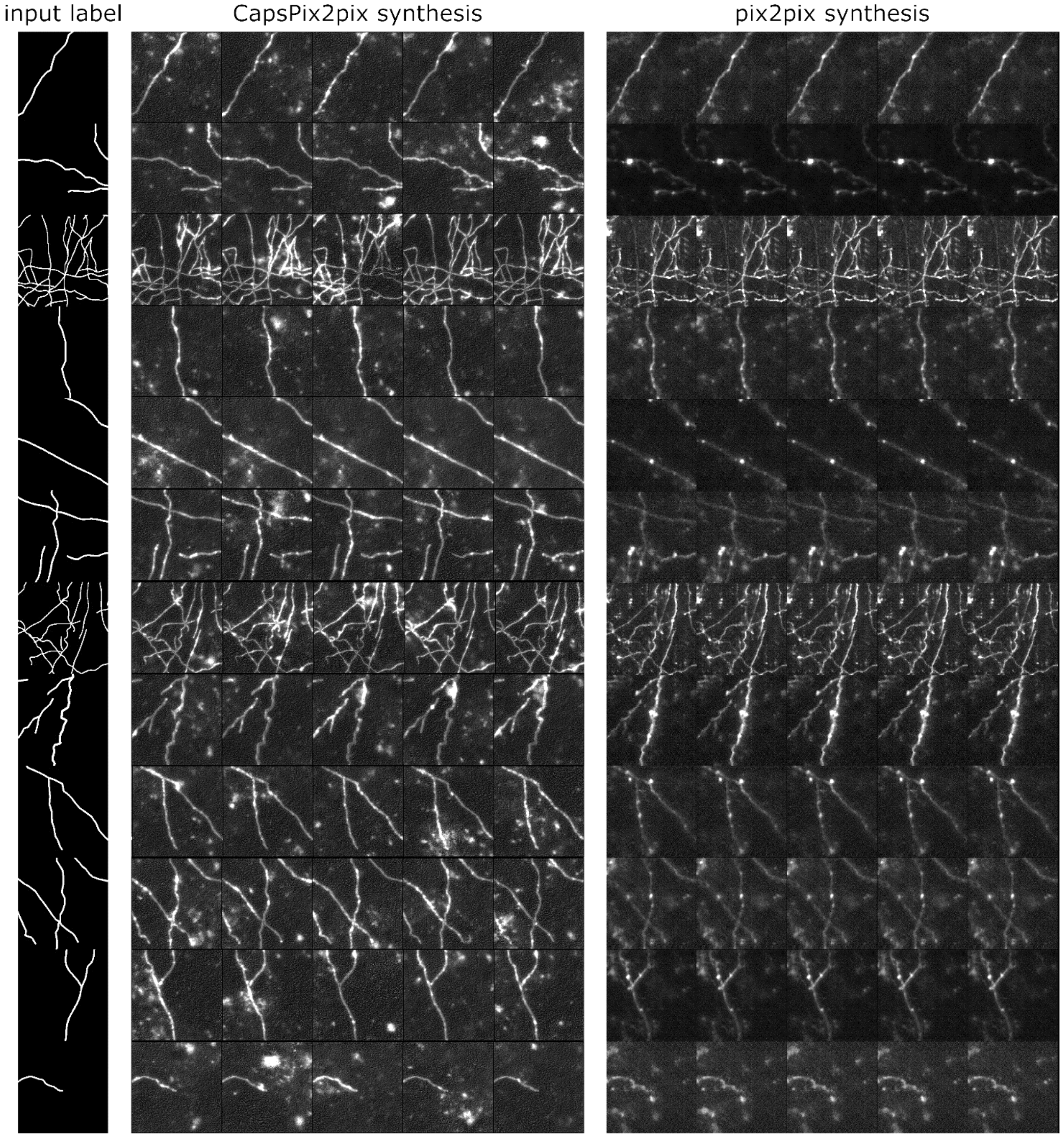


Bass, Cher, et al. "Image synthesis with a convolutional capsule generative adversarial network." International Conference on Medical Imaging with Deep Learning. 2019.

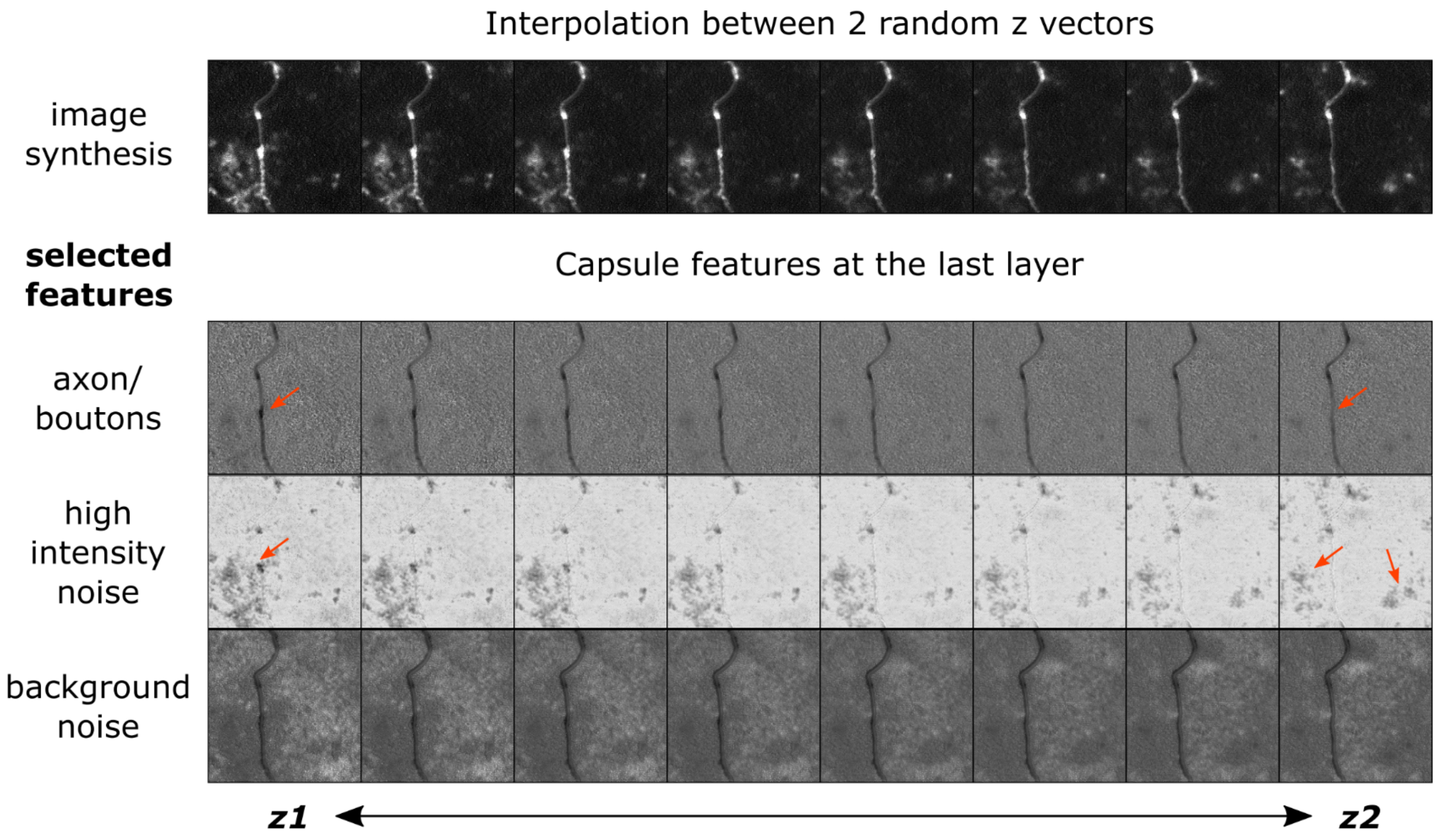
Qualitative results - features



Qualitative results - image synthesis variations

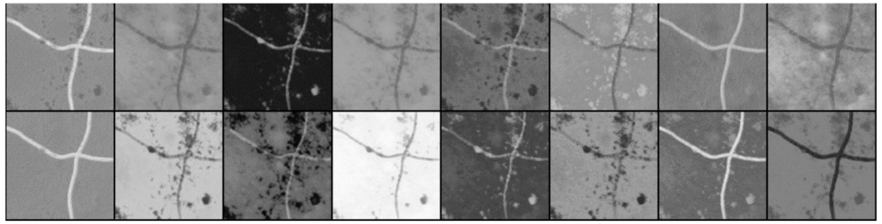


Qualitative results - interpolation



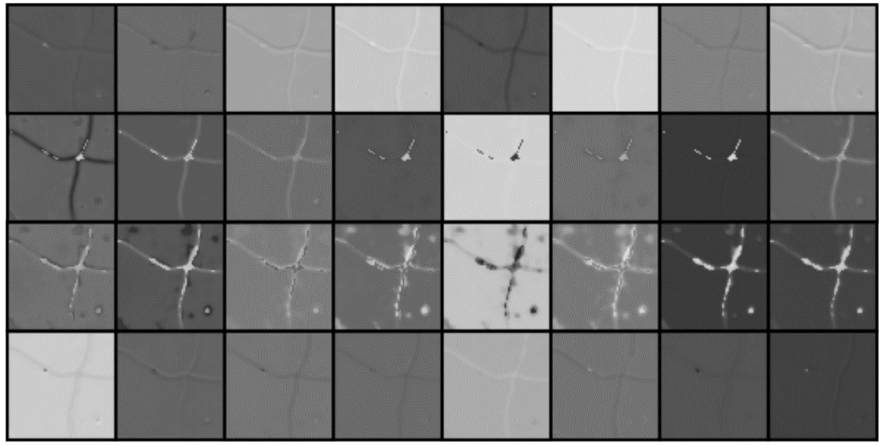
Qualitative results- intermediate features

capsule conv 1



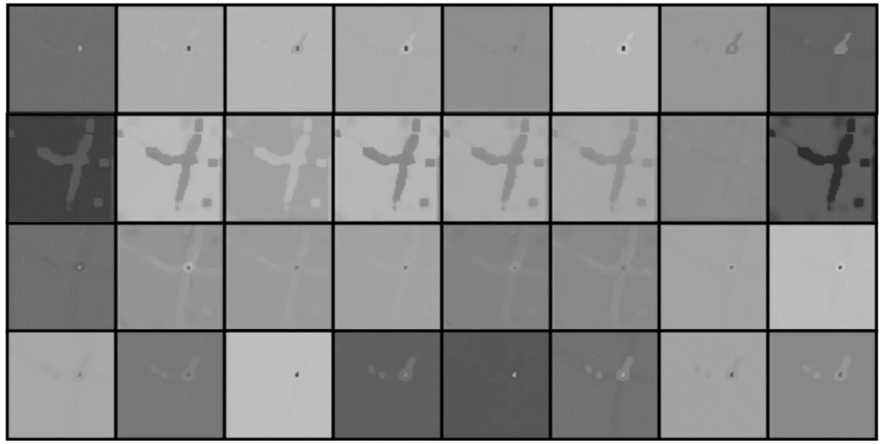
Capsule number	Num features shown per capsule
1/1	16/16

capsule conv 3



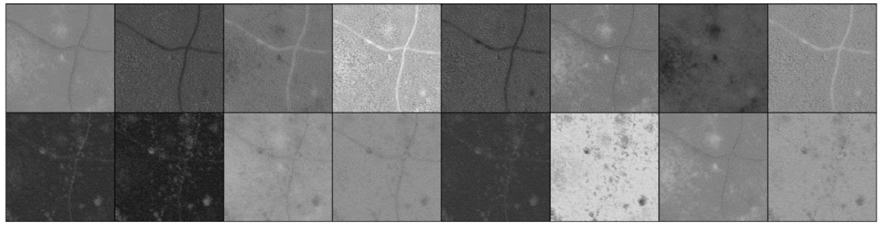
Capsule number	Num features shown per capsule
1/4	8/32
2/4	8/32
3/4	8/32
4/4	8/32

capsule conv 4



Capsule number	Num features shown per capsule
1/8	8/32
2/8	8/32
3/8	8/32
4/8	8/32

capsule conv 10



Capsule number	Num features shown per capsule
1/1	16/16

Code examples in pytorch

Code – W matrix multiplication

```
def forward(self, x):
    batch_size = x.size(0)
    x = torch.stack([x] * self.num_capsules, dim=2).unsqueeze(4)
    x = x.permute(0, 3, 2, 1, 4)
    W = torch.cat([self.W] * batch_size, dim=0)
    u_hat = torch.matmul(W, x)
```

“W Matrix” multiplication

Code – dynamic routing

```
b_ij = Variable(torch.zeros(1, self.in_channels, self.num_capsules, 1))
num_iterations = 3
    for iteration in range(num_iterations):
        c_ij = F.softmax(b_ij)
        c_ij = torch.cat([c_ij] * batch_size, dim=0).unsqueeze(4)

        s_j = (c_ij * u_hat).sum(dim=1, keepdim=True)
        v_j = self.squash(s_j)
        v_j= v_j.squeeze(1)

        if iteration < num_iterations - 1:
            temp = u_hat.permute(0, 2, 1, 3, 4).squeeze(4)
            temp2 = v_j
            a_ij = torch.matmul(temp, temp2).transpose(1,2)
            b_ij = b_ij + a_ij.mean(dim=0)
return v_j
```

b vector zero initialisation – notice shape

Routing x3 times

softmax

Dot product- sum out input capsule d

Squash norm

b update step

Code – convolutional capsules

```
def forward(self, x):
        batch_size = x.size(0)
        in_width, in_height = x.size(3), x.size(4)
        x = x.view(batch_size*self.in_capsules, self.in_channels, in_width, in_height)
        u_hat = self.conv2d(x)
        out_width, out_height = u_hat.size(2), u_hat.size(3)
        u_hat = u_hat.view(batch_size, self.in_capsules, out_width, out_height,
self.out_capsules, self.out_channels)
```

Weight sharing
between capsules

Code – local dynamic routing

b_ij = Variable(torch.zeros(1, self.in_capsules, **out_width, out_height**, self.out_capsules))

2 extra dimensions for routing

for iteration in range(self.num_routes):

c_ij = F.softmax(b_ij, dim=1)

c_ij = torch.cat([c_ij] * batch_size, dim=0).unsqueeze(5)

Dot product- sum out input capsule d

s_j = (**c_ij * u_hat**).sum(dim=1, keepdim=True)

v_j = v_j.squeeze(1)

if iteration < self.num_routes - 1:

temp = u_hat.permute(0, 2, 3, 4, 1, 5)

temp2 = v_j.unsqueeze(5)

a_ij = torch.matmul(temp, temp2).squeeze(5) # dot product here

a_ij = a_ij.permute(0, 4, 1, 2, 3)

b_ij = b_ij + a_ij.mean(dim=0)

return v_j