International Centre for Radio Astronomy Research

Looking for change? Roll the Dice and demand Attention

Foivos I. Diakogiannis, Francois Waldner, Peter Caccetta 2020 (under review)

https://github.com/feevos/ceecnet

Change detection: what is it?

Looking for change? Roll the Dice and demand Attention

Foivos I. Diakogiannis[a,b,1], François Waldner[c], Peter Caccetta[b]

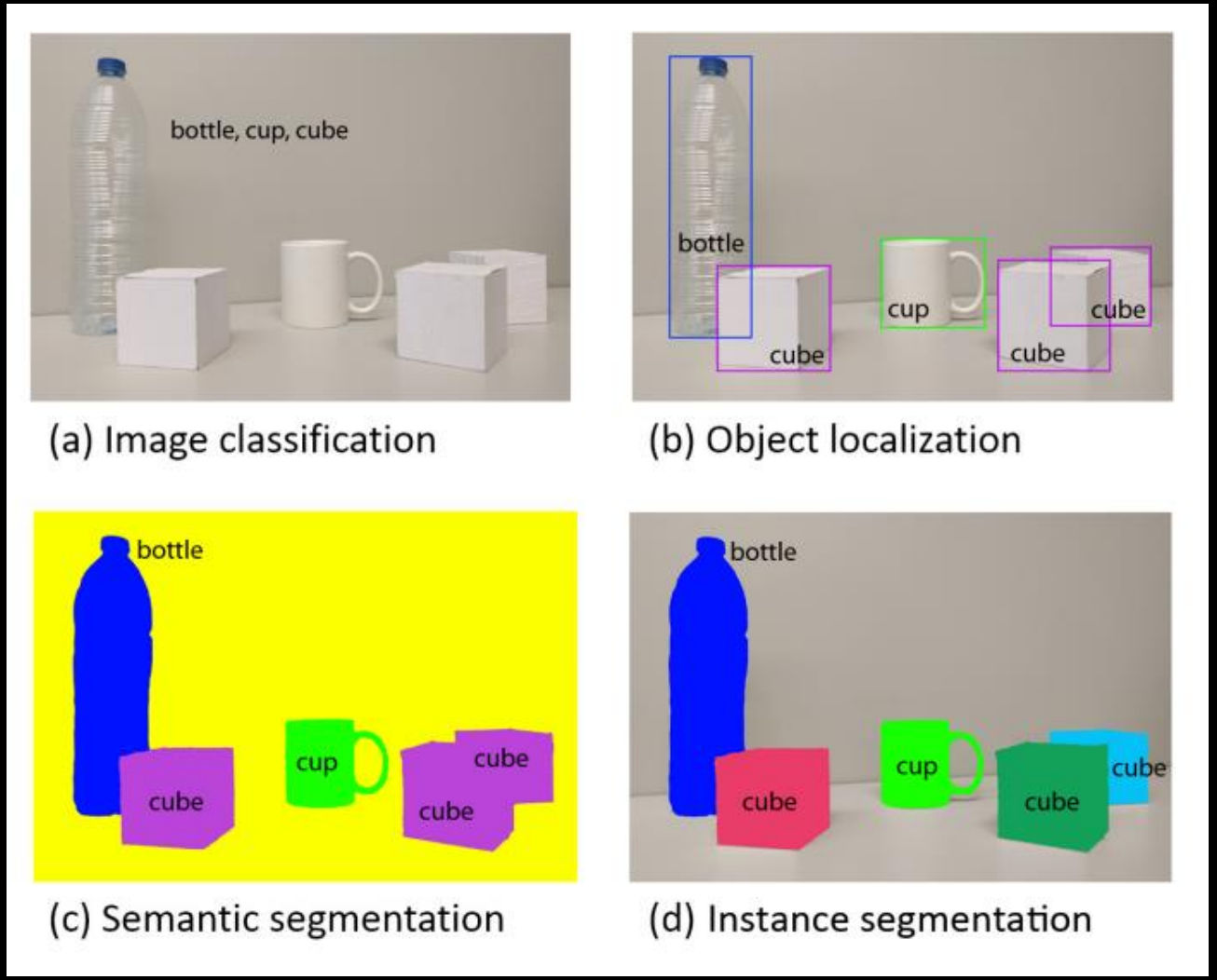
[a]ICRAR, the University of Western Australia
[b]Data61, CSIRO, Floreat WA
[c]CSIRO Agriculture & Food, St Lucia, QLD, Australia

under review

Compare (DL)

Change detection is a double input – single output **semantic segmentation** problem (with some peculiarities)

A necessary parenthesis: semantic segmentation definition + lessons learned



A Review on Deep Learning Techniques Applied to Semantic Segmentation

A. Garcia-Garcia, S. Orts-Escolano, S.O. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez

https://arxiv.org/abs/1704.06857

Semantic Segmentation: lessons learned

ISPRS Journal of Photogrammetry and Remote Sensing 162 (2020) 94–114

Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

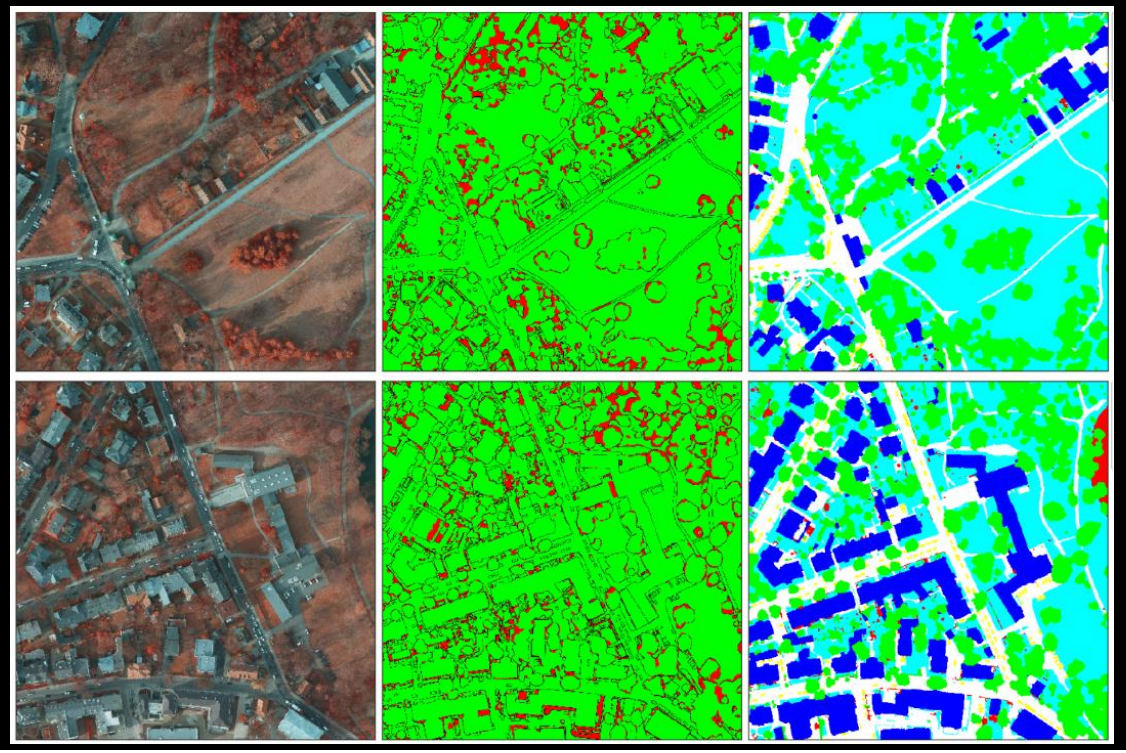
journal homepage: www.elsevier.com/locate/isprsjprs

ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data

Foivos I. Diakogiannis[a], François Waldner[b], Peter Caccetta[a], Chen Wu[c]

[a] Data61, CSIRO, Floreat, WA, Australia
[b] CSIRO Agriculture & Food, St Lucia, QLD, Australia
[c] ICRAR, The University of Western Australia, Crawley, WA, Australia

https://github.com/feevos/resuneta



1. Atrous convolutions (the good and the bad: can we do better?)
2. Importance of context information
3. Analysis of Dice family of loss functions: can we do better?
4. "Causal" (conditioned) multitasking
5. Emphasis on boundaries

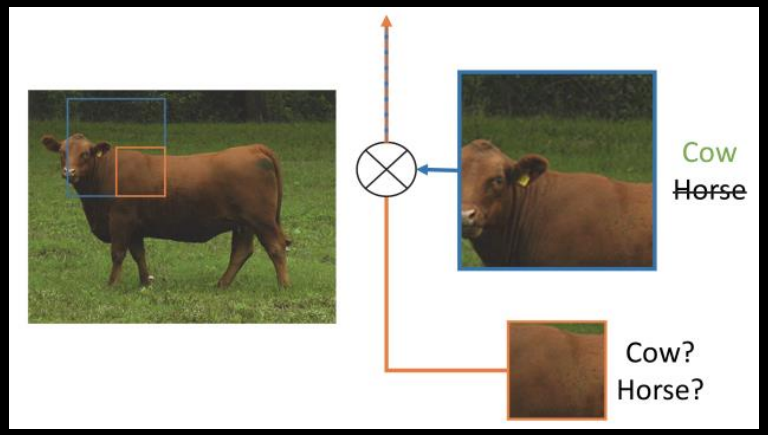
Table 5
Potsdam comparison of results (based on per class F1 score) with other authors. Best values are marked with bold, second best values are underlined, third best values are in square brackets. Models trained with FoV × 1 were trained on 256x256 patches extracted from the original resolution images.

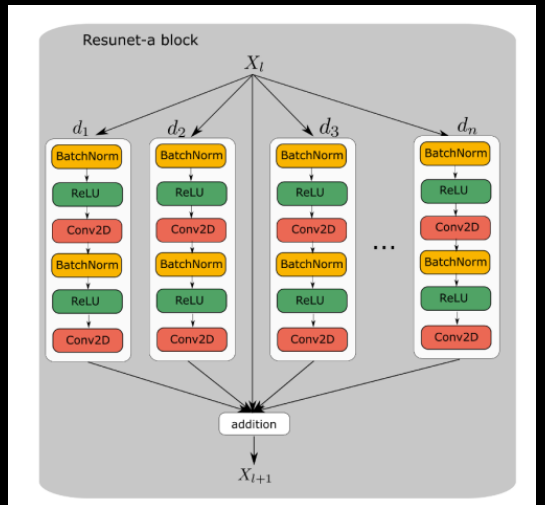
Methods	ImSurface	Building	LowVeg	Tree	Car	Avg. F1	OA
UZ_1 (Volpi and Tuia, 2017)	89.3	95.4	81.8	80.5	86.5	86.7	85.8
RIT_L7 (Liu et al., 2017b)	91.2	94.6	85.1	85.1	92.8	89.8	88.4
RIT_4 (Piramanayagam et al., 2018)	92.6	97.0	86.9	87.4	95.2	91.8	90.3
DST_5 (Sherrah, 2016)	92.5	96.4	86.7	88.8	94.7	91.7	90.3
CAS_Y3 (ISPRS)	92.2	95.7	87.2	87.6	95.6	91.7	90.1
CASIA2 (Liu et al., 2018)	93.3	97.0	[87.7]	[88.4]	96.2	92.5	91.1
DPN_MFFL (Pan et al., 2018b)	92.4	[96.4]	87.8	88.0	95.7	92.1	90.4
HSN + OI + WBP (Liu et al., 2017a)	91.8	95.7	84.4	79.6	88.3	87.9	89.4
ResUNet-a d6 cmtsk (FoV × 1)	[93.0]	97.2	87.5	[88.4]	[96.1]	[92.4]	[91.0]
ResUNet-a d7v2 cmtsk (FoV × 1)	**93.5**	**97.2**	**88.2**	**89.2**	**96.4**	**92.9**	**91.5**

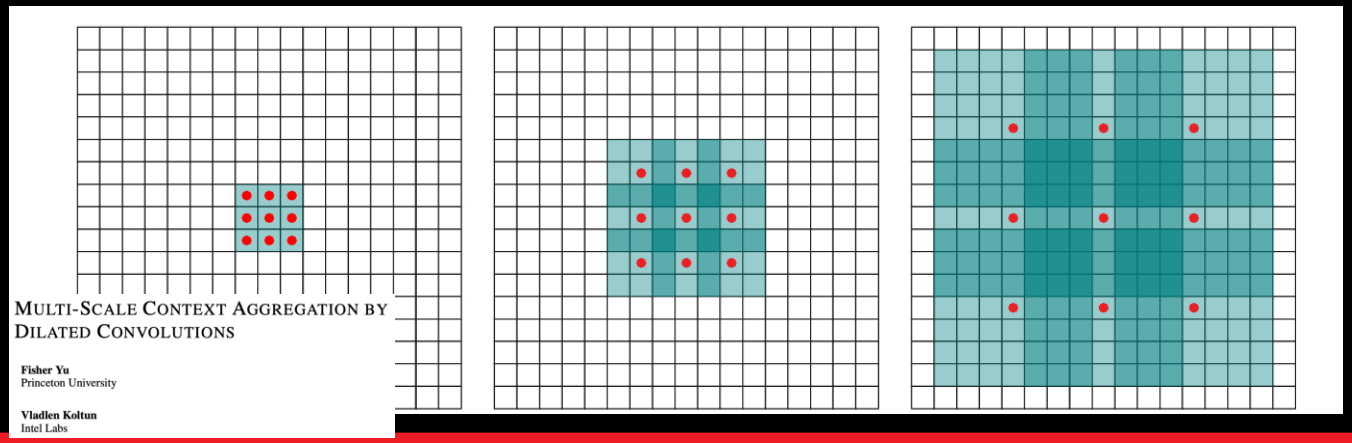
Semantic Segmentation: lessons learned

atrous convolutions:

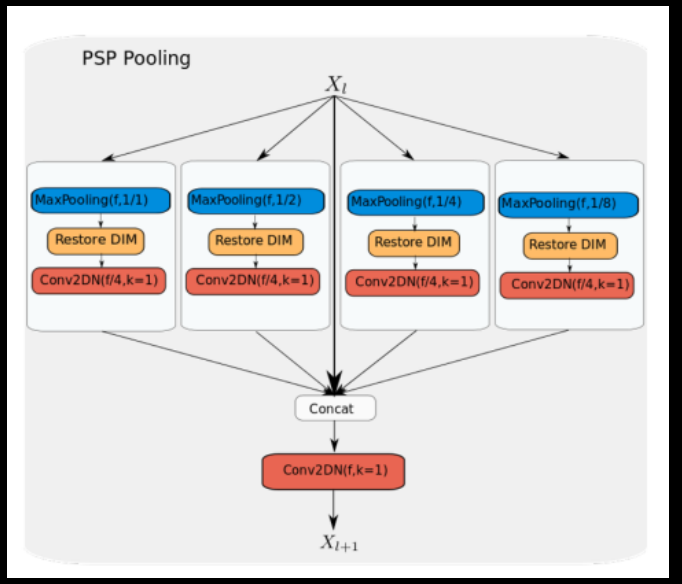
- increase receptive field (dropout-like regularization???)
- They do not emphasize specific regions of images (attention does!)





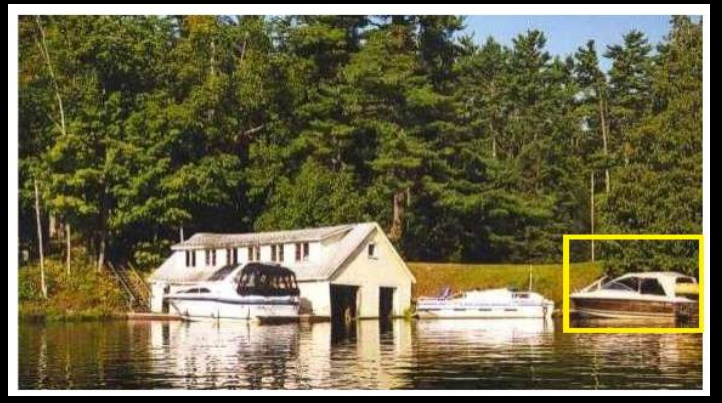


PSPPooling: context information



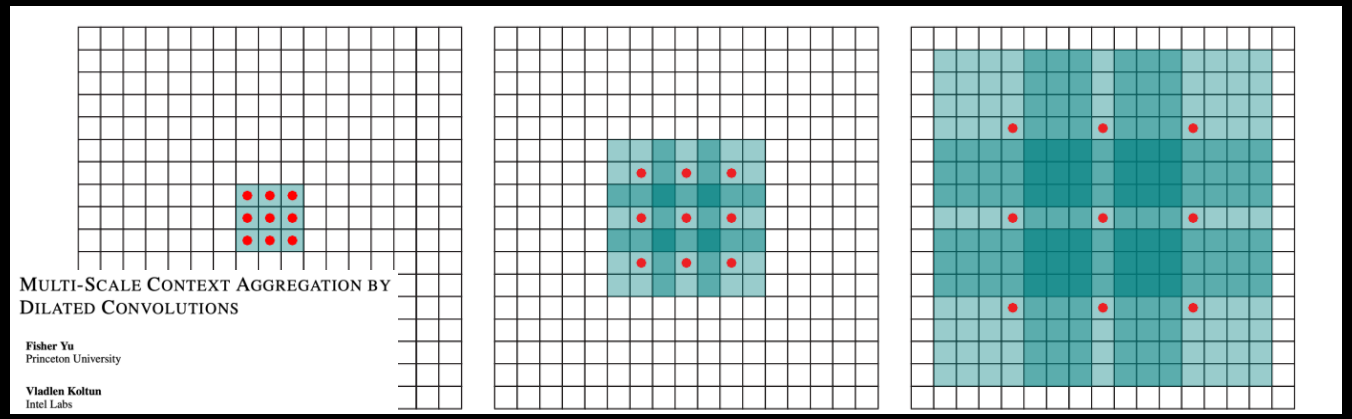

Pyramid Scene Parsing Network

Hengshuang Zhao[1] Jianping Shi[2] Xiaojuan Qi[1] Xiaogang Wang[1] Jiaya Jia[1]
[1]The Chinese University of Hong Kong [2]SenseTime Group Limited
{hszhao, xjqi, leojia}@cse.cuhk.edu.hk, xgwang@ee.cuhk.edu.hk, shijianping@sensetime.com

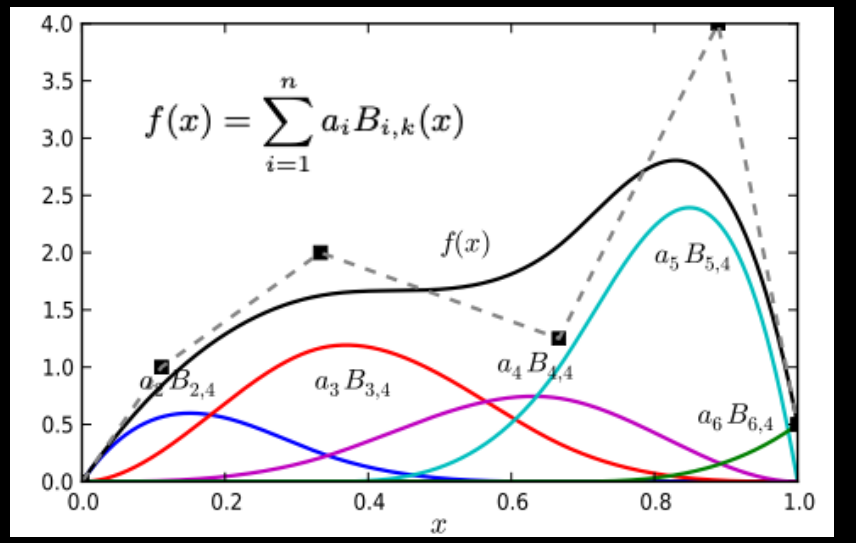


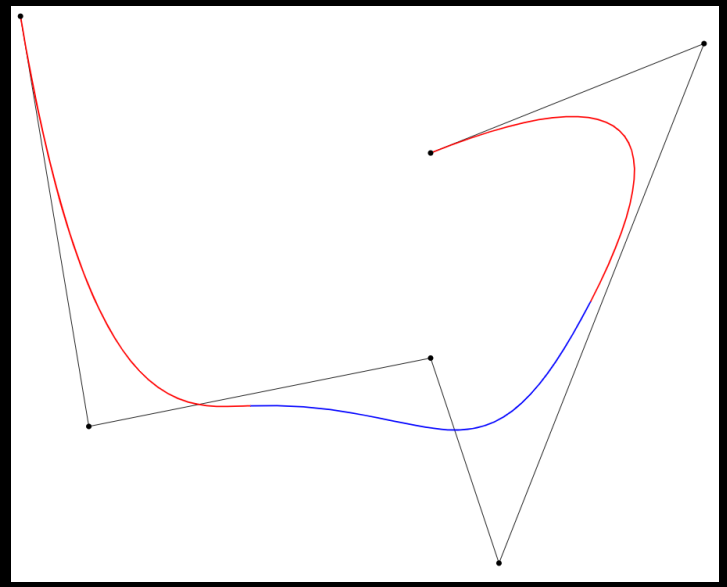
Semantic Segmentation: lessons learned

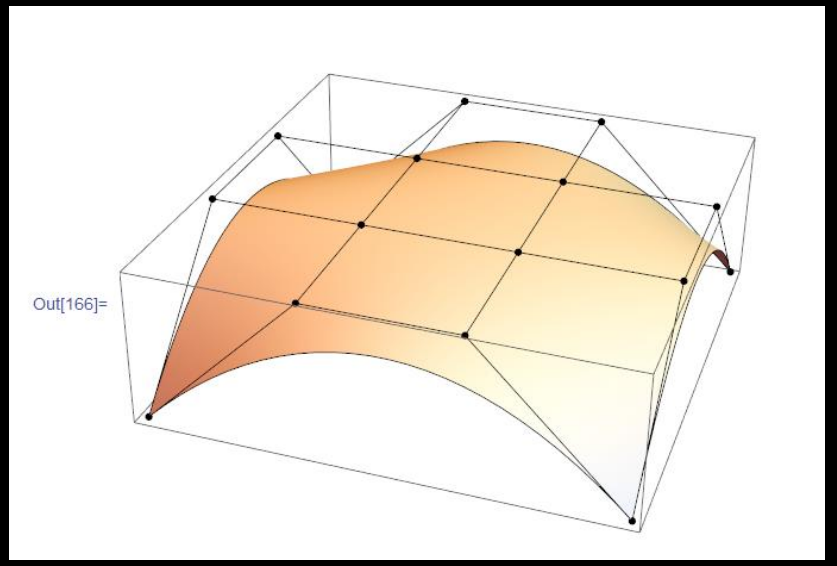
atrous convolutions - increased receptive field and/or regularization?



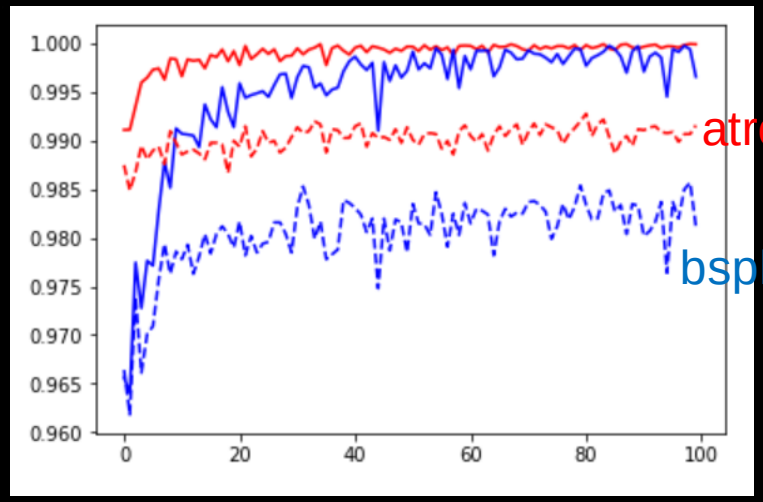
Bspline convolution: an interesting idea that failed (but we think we learned something ☺ - not published)







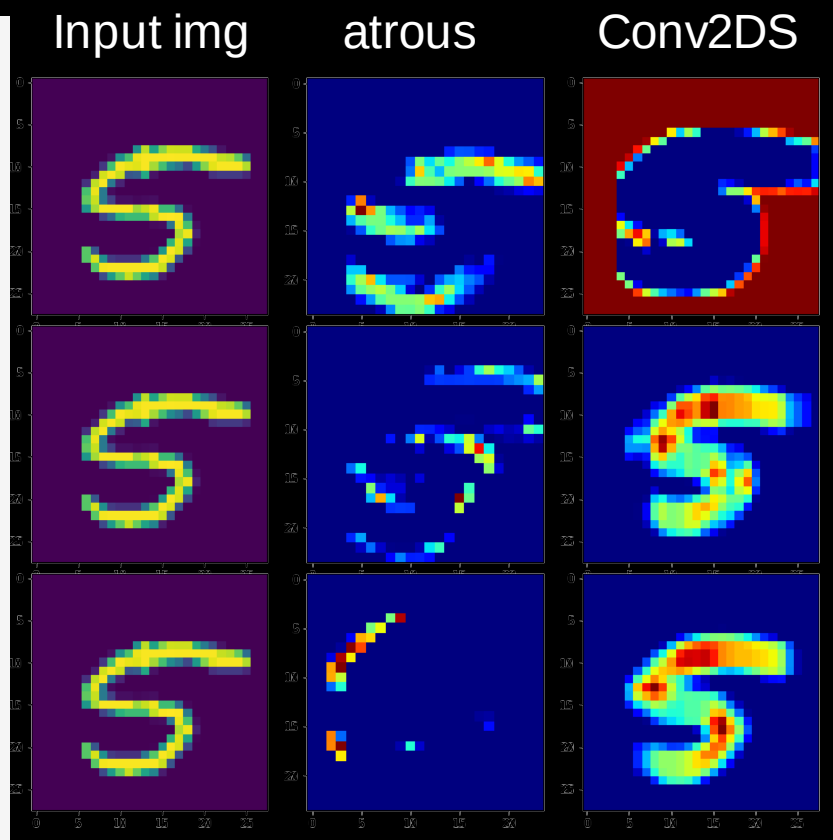
Semantic Segmentation: lessons learned



atrous convolutions - increased receptive field and/or regularization?

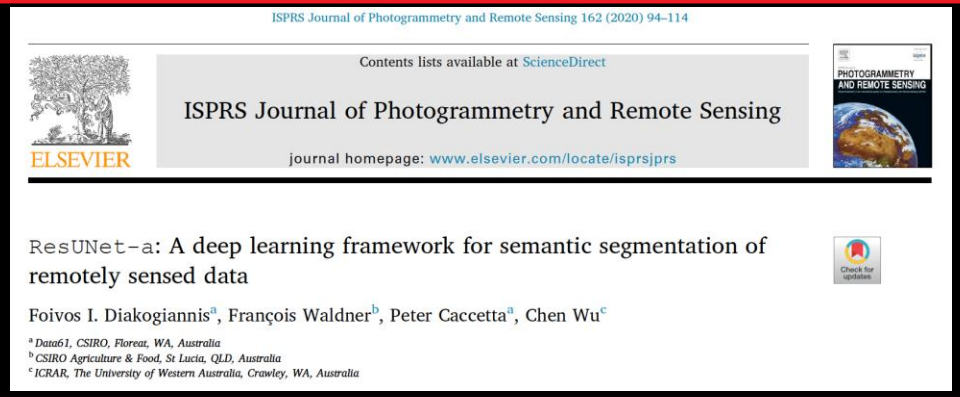
```
class net1(Block):
    def __init__(self,**kwards):
        Block.__init__(self,**kwards)
        with self.name_scope():

            self.conv1 = gluon.nn.Conv2D(channels=20, kernel_size=3,dilation=(2,2), activation='relu')
            self.pool1 = gluon.nn.MaxPool2D(pool_size=2, strides=2)
            self.BN1 = gluon.nn.BatchNorm()

            self.conv2 = gluon.nn.Conv2D(channels=50, kernel_size=3, dilation=(2,2), activation='relu')
            self.pool2 = gluon.nn.MaxPool2D(pool_size=2, strides=2)
            self.BN2 = gluon.nn.BatchNorm()

            self.flat = gluon.nn.Flatten()
            self.dense1 = gluon.nn.Dense(num_fc, activation="relu")
            self.dense2 = gluon.nn.Dense(num_outputs)


    def forward(self,_x):

        x1 = self.conv1(_x)
        x1 = self.BN1(x1)
        x = self.pool1(x1)

        x2 = self.conv2(x)
        x2 = self.BN2(x2)
        x = self.pool2(x2)

        x = self.flat(x)
        x = self.dense1(x)
        x = self.dense2(x)

        return x, x1, x2
```

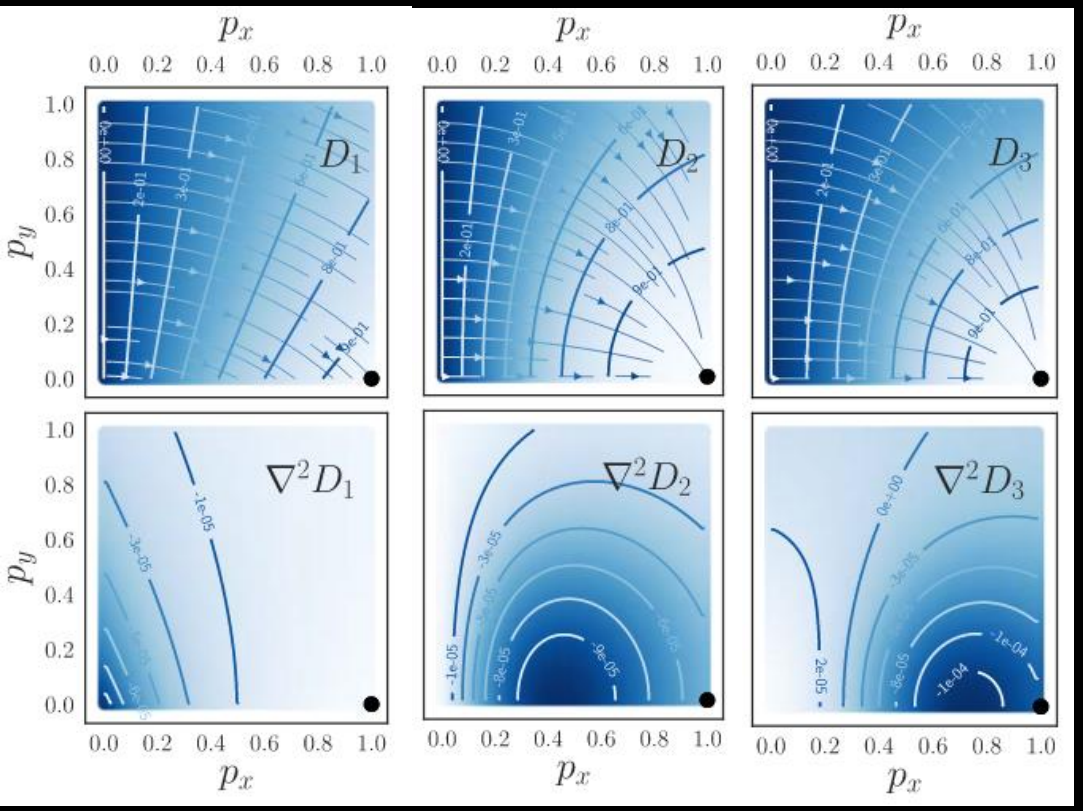
```
class net2(Block):
    def __init__(self,**kwards):
        Block.__init__(self,**kwards)
        with self.name_scope():

            self.conv1 = Conv2DS(1,20, kernel_size=3, kernel_effective_size=5)
            self.BN1 = gluon.nn.BatchNorm()
            self.pool1 = gluon.nn.MaxPool2D(pool_size=2, strides=2)


            self.conv2 = Conv2DS(20,50, kernel_size=3, kernel_effective_size=5)
            self.BN2 = gluon.nn.BatchNorm()
            self.pool2 = gluon.nn.MaxPool2D(pool_size=2, strides=2)


            self.flat = gluon.nn.Flatten()
            self.dense1 = gluon.nn.Dense(num_fc, activation="relu")
            self.dense2 = gluon.nn.Dense(num_outputs)


    def forward(self,_x):

        x1 = self.conv1(_x)
        x1 = self.BN1(x1)
        x1 = nd.relu(x1)
        x = self.pool1(x1)


        x2 = self.conv2(x)
        x2 = self.BN2(x2)
        x2 = nd.relu(x2)
        x = self.pool2(x2)


        x = self.flat(x)
        x = self.dense1(x)
        x = self.dense2(x)

        return x, x1, x2
```



Semantic Segmentation: lessons learned – Dice, a set similarity metric

Not all “Dice” are created equal …





$$D_1\left(\mathbf{p}, \mathbf{l}\right) = \frac{2\sum_i p_i l_i}{\sum_i p_i + \sum_i l_i}$$

$$D_2\left(\mathbf{p}, \mathbf{l}\right) = \frac{2\sum_i p_i l_i}{\sum_i \left(p_i^2 + l_i^2\right)}$$

$$D_3\left(\mathbf{p}, \mathbf{l}\right) = \frac{\sum_i p_i l_i}{\sum_i \left(p_i^2 + l_i^2\right) - \sum_i \left(p_i l_i\right)}$$

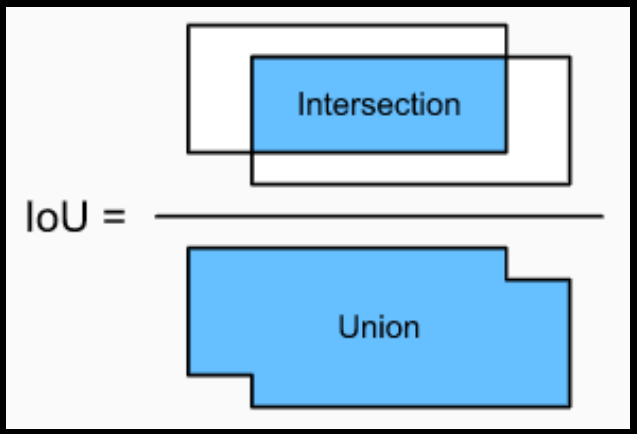


Image credit: d2l.ai

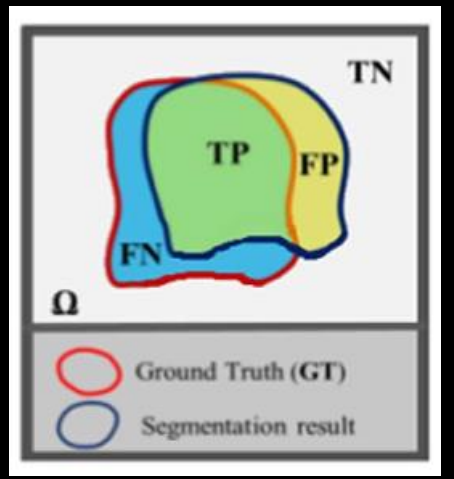
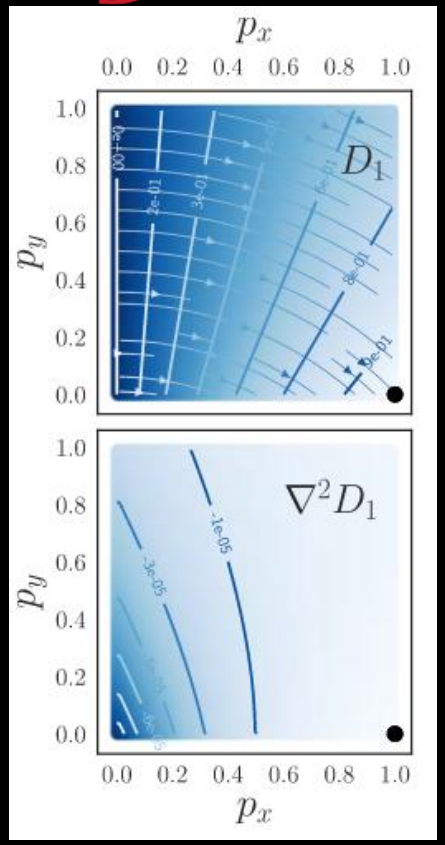
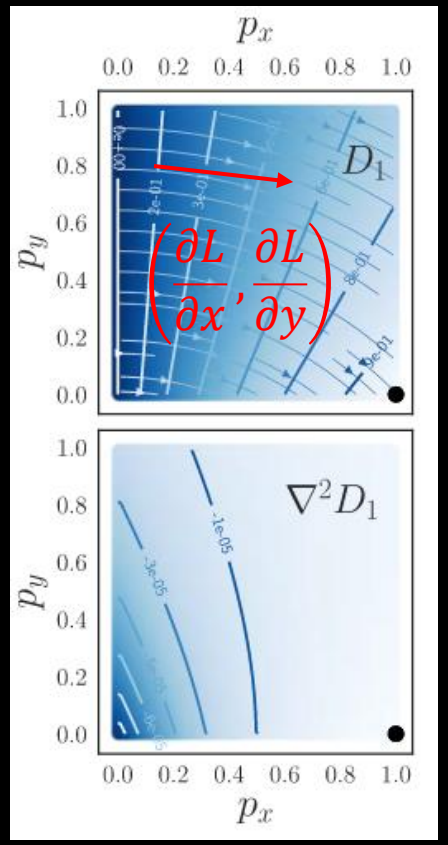


Image credit, Islam Reda et al. 2017, *A comprehensive non-invasive framework for diagnosing prostate cancer*

Semantic Segmentation: lessons learned – Dice, a **set** similarity metric





Loss derivatives w.r.t. last layer:
a tangent vector

Inner product

hidden layers

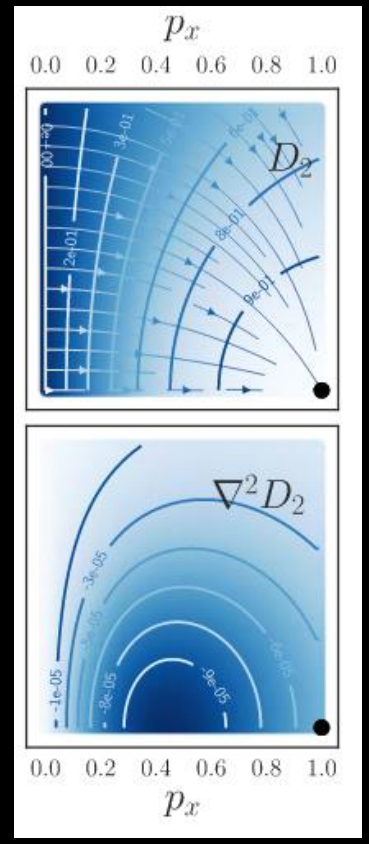
$$\frac{\partial L}{\partial \vec{w}} = \cdots \left(\frac{\partial x}{\partial \vec{w}}, \frac{\partial y}{\partial \vec{w}}\right) \circ \left(\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}\right)$$

last layer

$$D_1\left(\mathbf{p}, \mathbf{l}\right) = \frac{2\sum_i p_i l_i}{\sum_i p_i + \sum_i l_i}$$

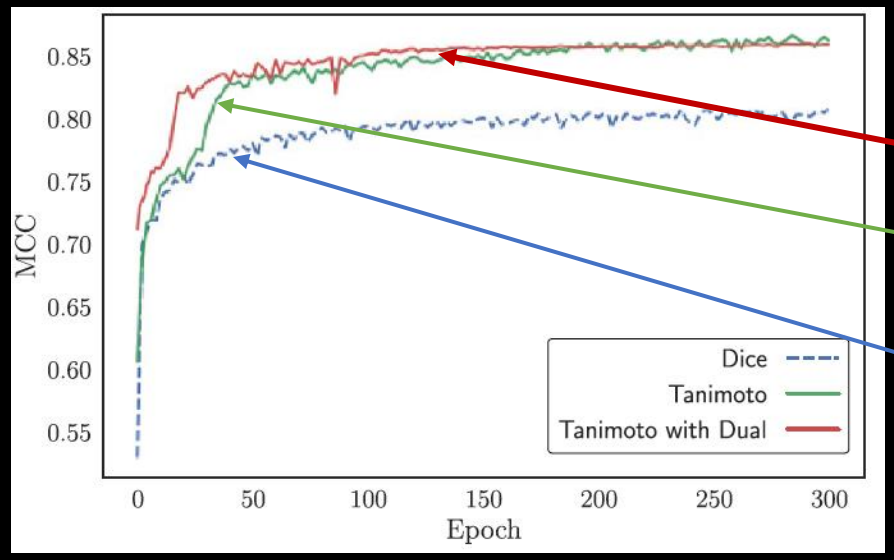
$$D_2\left(\mathbf{p}, \mathbf{l}\right) = \frac{2\sum_i p_i l_i}{\sum_i (p_i^2 + l_i^2)}$$

$$D_3\left(\mathbf{p}, \mathbf{l}\right) = \frac{\sum_i p_i l_i}{\sum_i (p_i^2 + l_i^2) - \sum_i (p_i l_i)}$$



Not all “Dice” are created equal …

Semantic Segmentation: lessons learned – Tanimoto **set** similarity metric



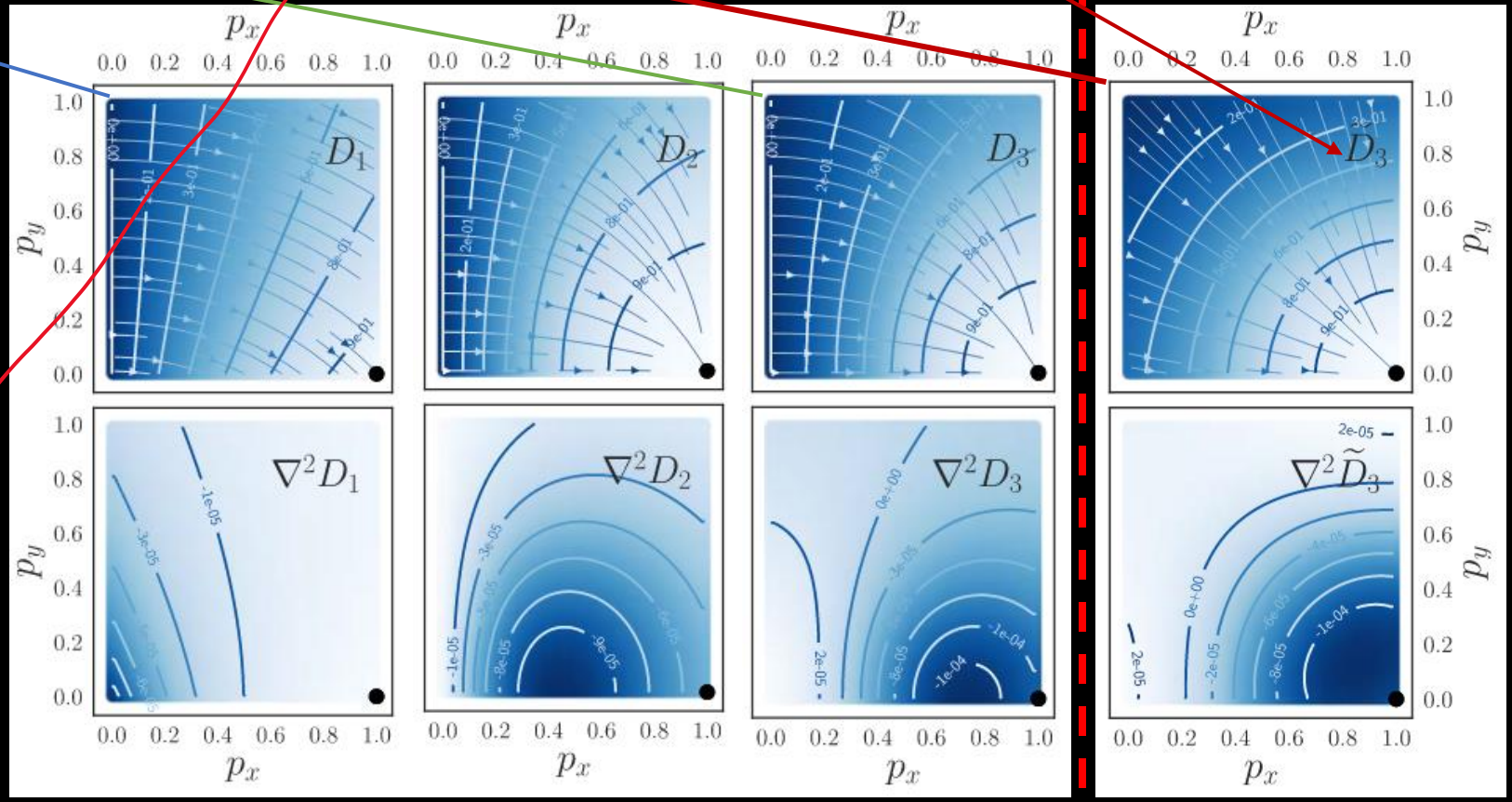
$$\widetilde{T}\left(p_i, l_i\right) = \frac{T(p_i, l_i) + T(\mathbf{1} - p_i, \mathbf{1} - l_i)}{2}$$

ResUNet-a Diakogiannis et al. 2019

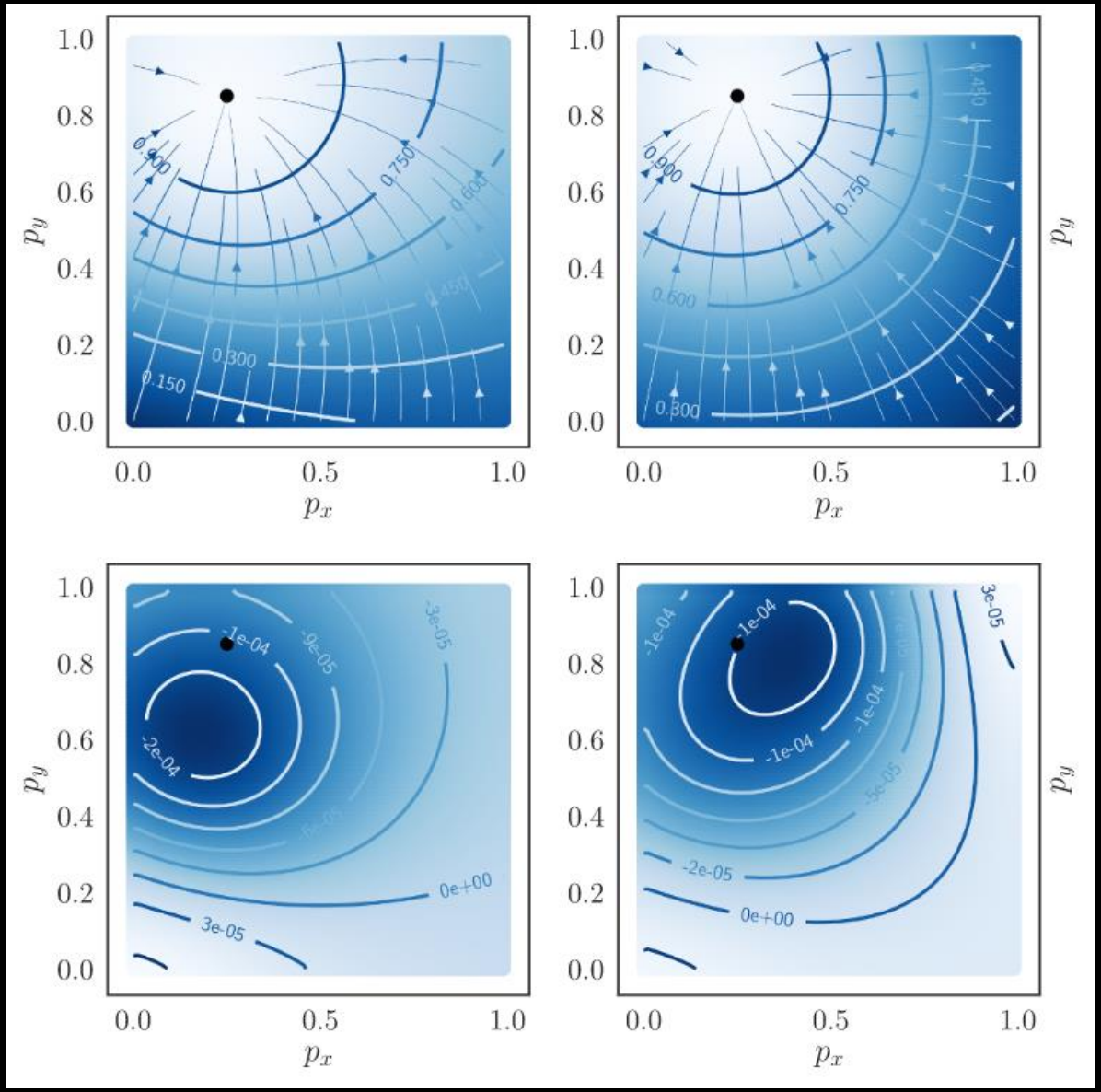
$$D_1\left(\mathbf{p}, \mathbf{l}\right) = \frac{2\sum_i p_i l_i}{\sum_i p_i + \sum_i l_i}$$

$$D_2\left(\mathbf{p}, \mathbf{l}\right) = \frac{2\sum_i p_i l_i}{\sum_i (p_i^2 + l_i^2)}$$

$$D_3\left(\mathbf{p}, \mathbf{l}\right) = \frac{\sum_i p_i l_i}{\sum_i (p_i^2 + l_i^2) - \sum_i (p_i l_i)}$$

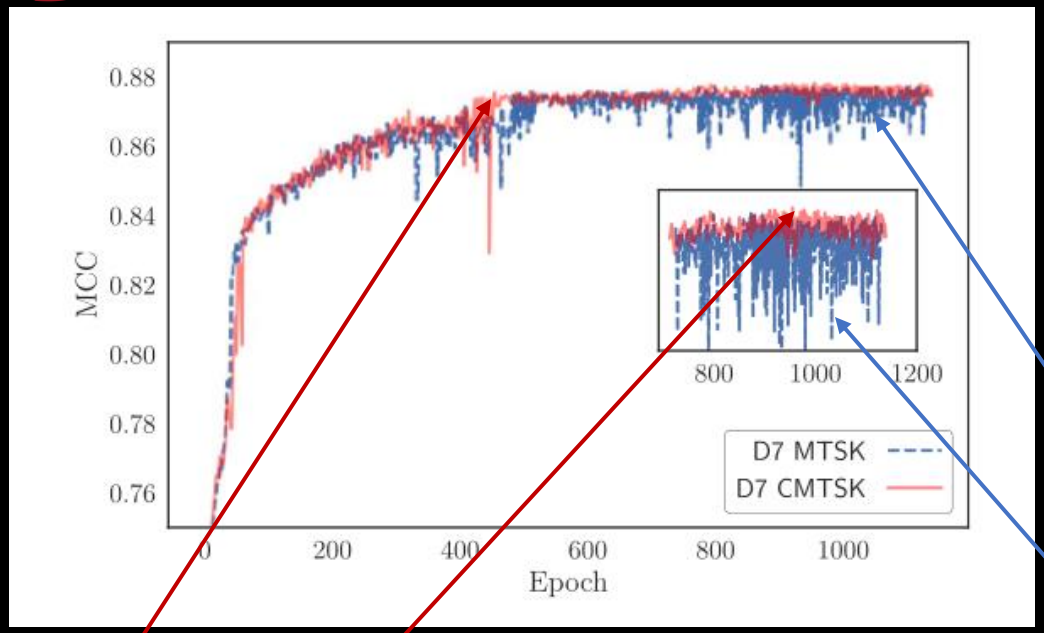


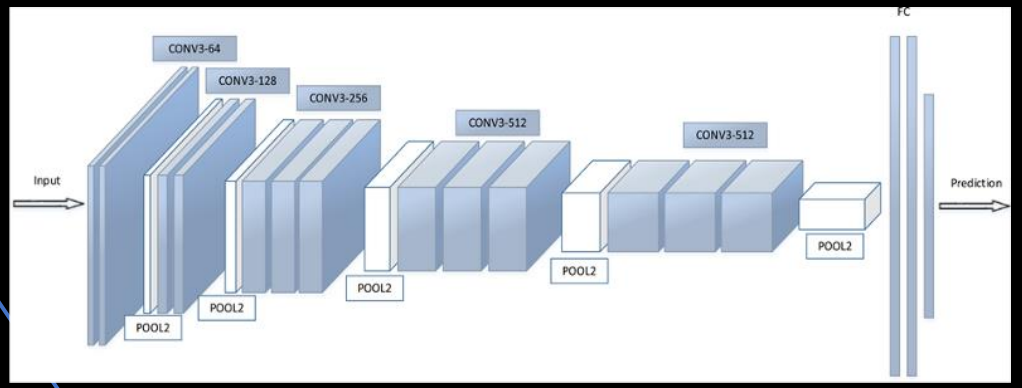
Semantic Segmentation: lessons learned – Tanimoto **set** similarity metric



Still, not spot on, not steep enough …
Can we improve?

Semantic Segmentation: lessons learned – Multitasking, “causal” vs non-“causal”

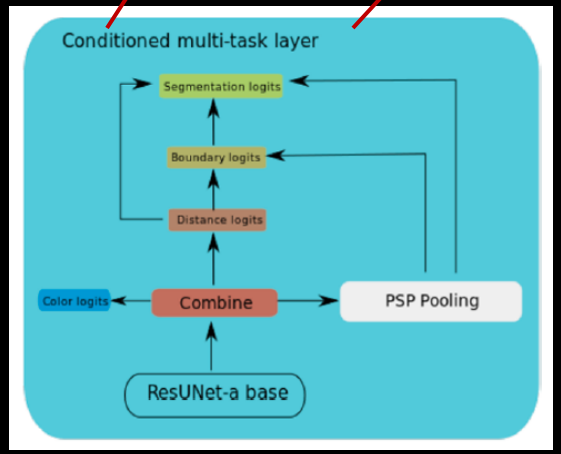


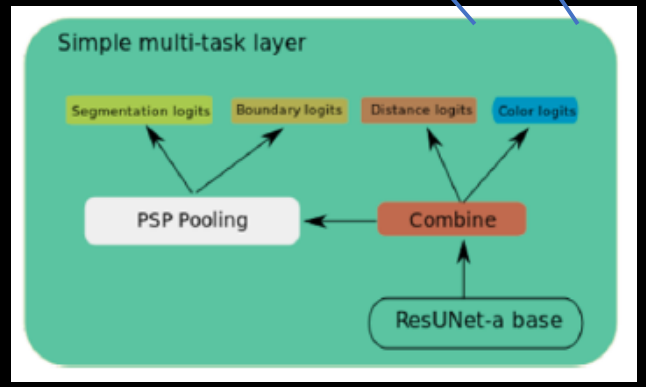


Task 1

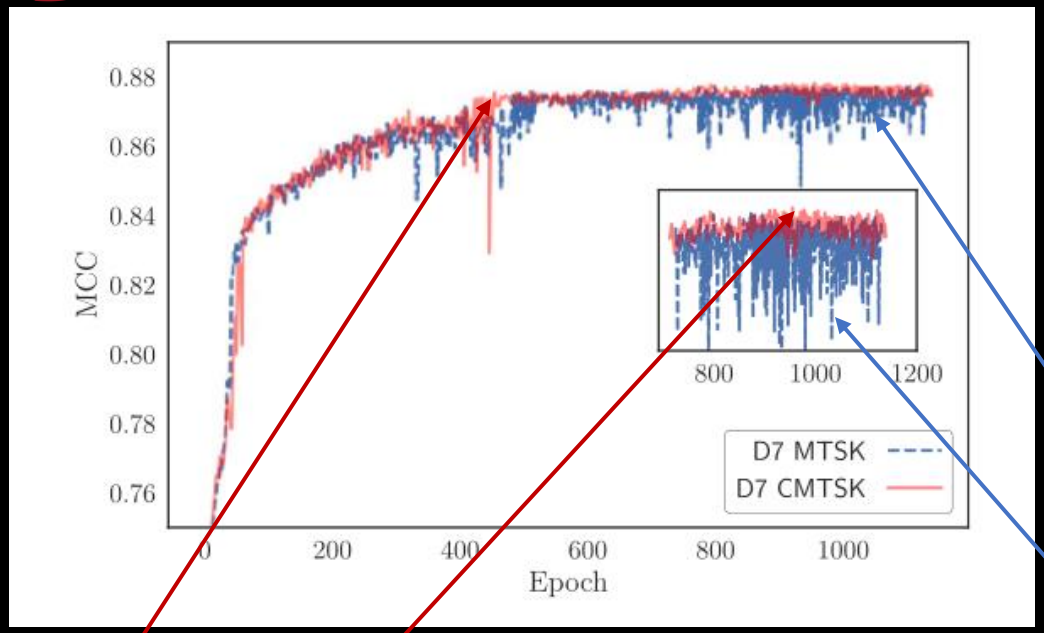
Task 2

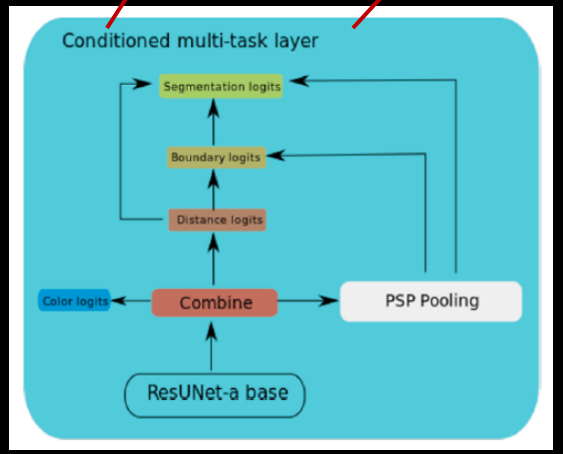
Task 3

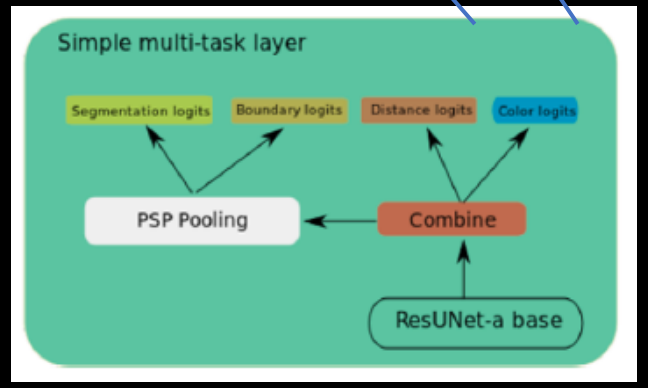


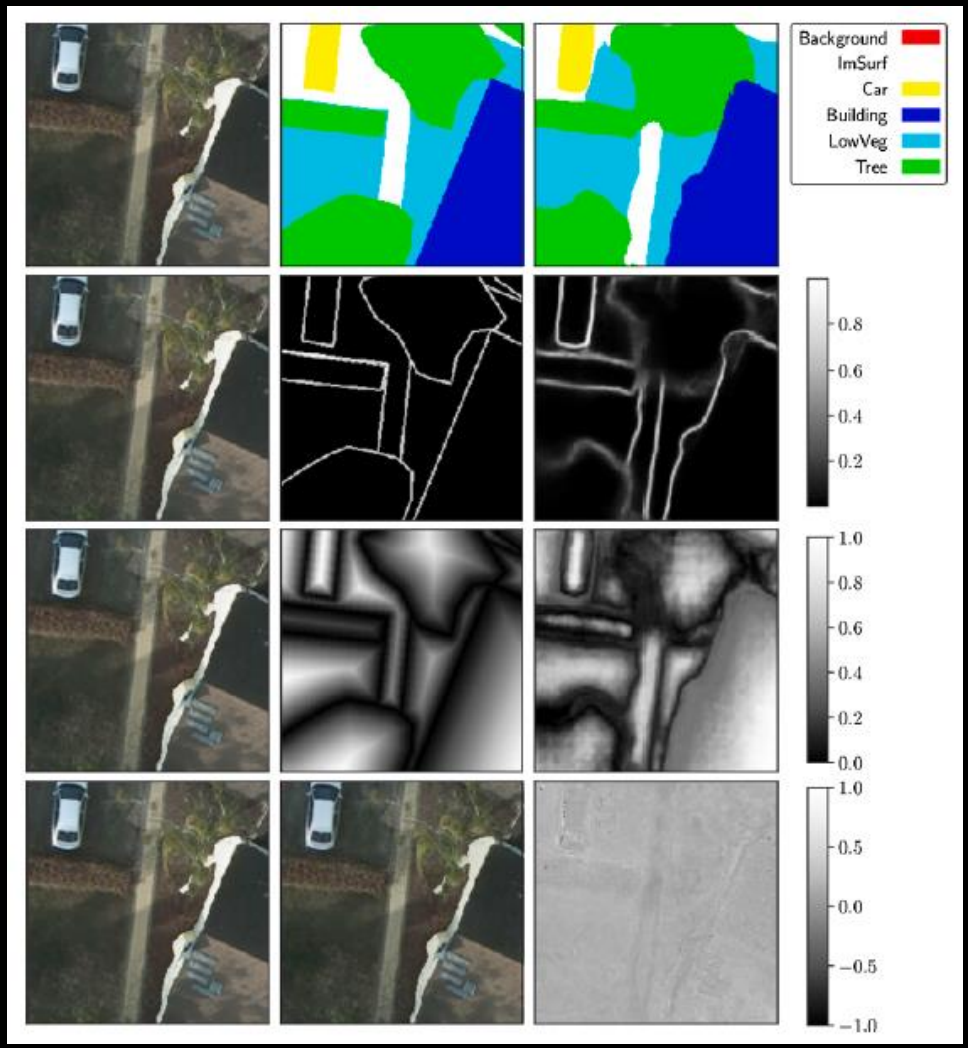


Semantic Segmentation: lessons learned - “causal” multitasking semantic segmentation

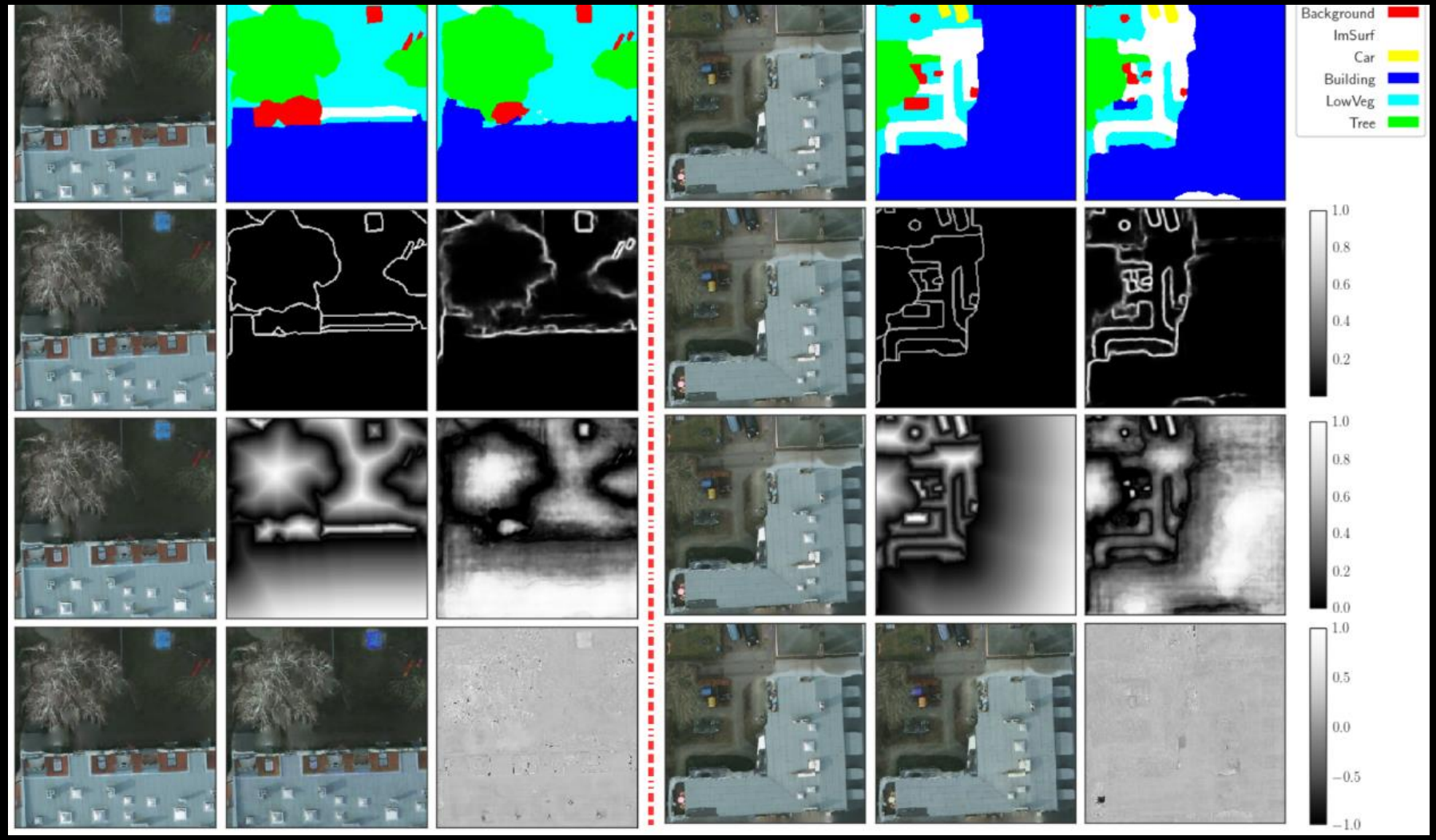








Multitasking: Questions guide performance



Semantic Segmentation: lessons learned – emphasis on boundaries – need sharp transition

Can we improve? (Inria building dataset, early attempts, unpublished)

Change detection: double input **semantic segmentation**

Looking for change? Roll the Dice and demand Attention

Foivos I. Diakogiannis[a,b,1], François Waldner[c], Peter Caccetta[b]

[a]ICRAR, the University of Western Australia
[b]Data61, CSIRO, Floreat WA
[c]CSIRO Agriculture & Food, St Lucia, QLD, Australia

under review

Compare (DL)

Putting all things together, how do we:

a) combine/process/consume/compare the two inputs?
b) Improve on the segmentation task?

Change detection: learning to compare

Identify change between two bi-temporal images, on objects of interest.

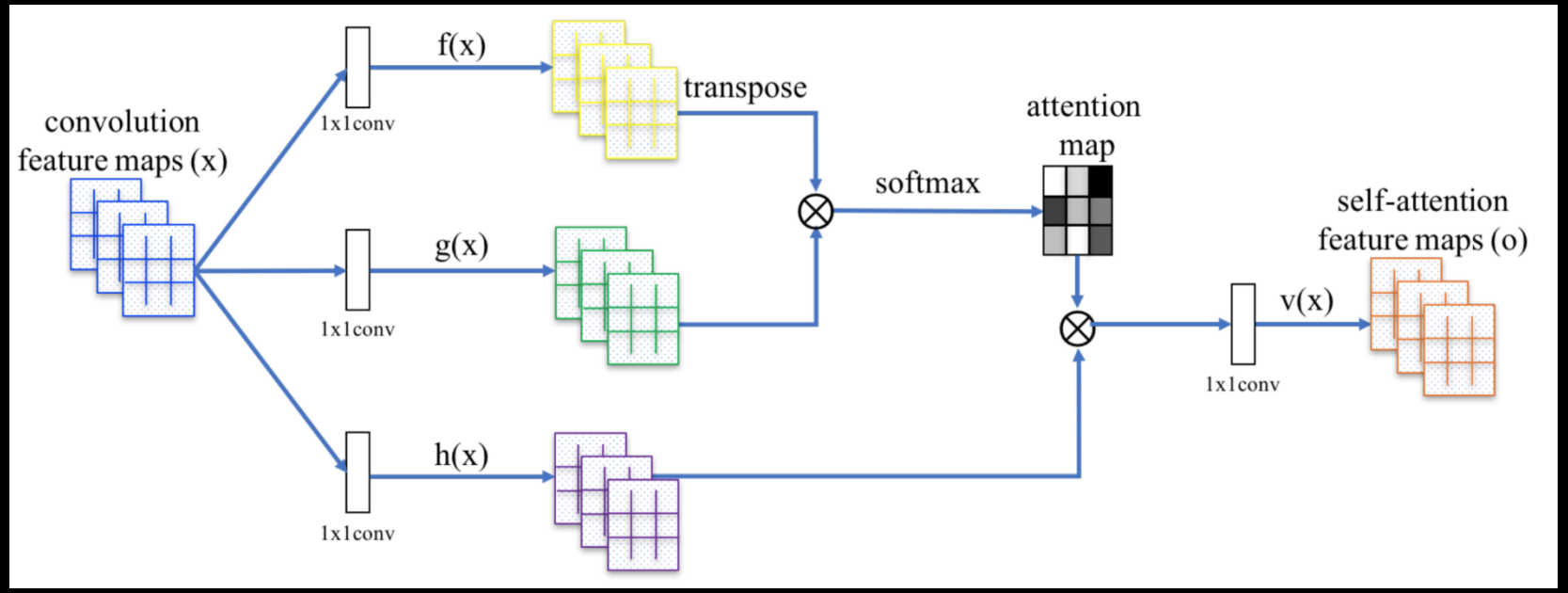
More complex images take more time to Identify where change exists

Change detection: learning to **compare**

- Hypothesis: identifying change is a higher cognitive process.
- We understand this because time to compare correlates strongly with the complexity of change.
- Change cannot be identified by subtraction (constant time operation → contradicts complexity of time correlation).
- Hypothesis: we need a mechanism similar to **Attention.**

Change detection: learning to compare

On self Attention from NLP + vision (SAGAN)



Attention (NLP) Vaswani et al. (2017)

$$\mathbf{o} = \text{softmax}\left(\frac{\mathbf{q} \circ_1 \mathbf{k}}{\sqrt{d}}\right), \qquad \in \Re^{C_q \times C}$$

$$\text{Att}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \mathbf{o} \circ_2 \mathbf{v}, \qquad \in \Re^{C_q \times H \times W}$$

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Quest:

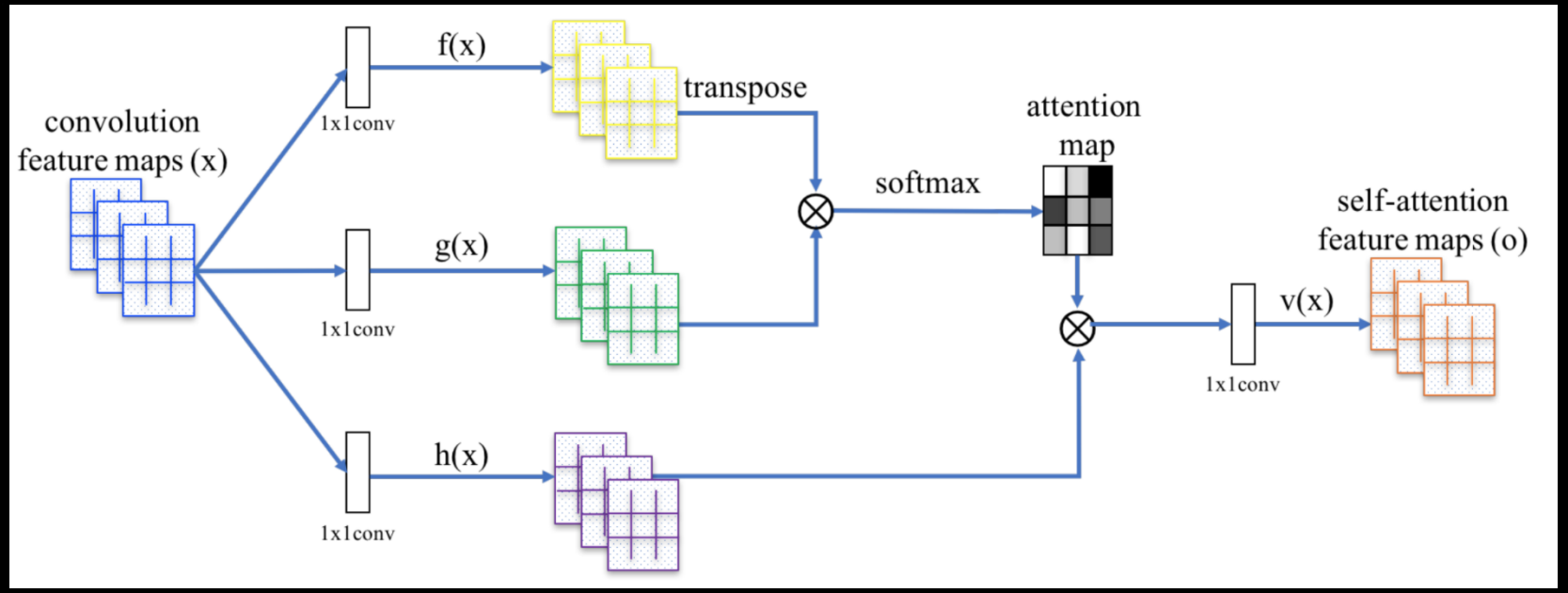
1. Understand Attention… (improve on atrous + use it for comparison)
2. Dot product is not a nice comparison operator (not bounded, not set similarirt)
3. Attention (memory) is about set operations: need LAYER set similarity
4. Memory efficient Attention

$$\mathbf{q} \circ_1 \mathbf{k} \equiv \sum_{jk} q_{ijk} k_{ljk} \in \Re^{C_q \times C}$$

$$\mathbf{o} \equiv \text{softmax}\,(\mathbf{q} \circ_1 \mathbf{k}) \equiv o_{il}$$

$$\text{Att}(\mathbf{q}, \mathbf{k}, \mathbf{v}) \equiv \text{Att}_{ikj} \equiv \mathbf{o} \circ_2 \mathbf{v} \equiv \sum_{l} o_{il} v_{lkj} \in \Re^{C_q \times H \times W}$$

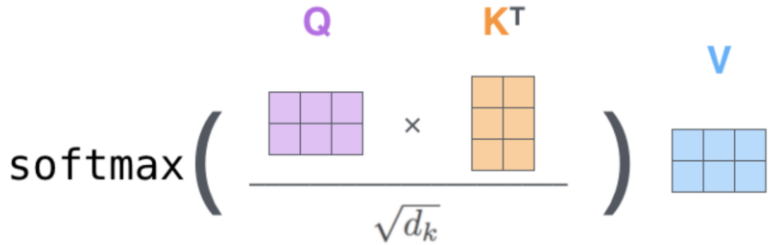
On self Attention from NLP + vision (SAGAN)



Attention (NLP) Vaswani et al. (2017)

$$\mathbf{o} = \text{softmax}\left(\frac{\mathbf{q} \circ_1 \mathbf{k}}{\sqrt{d}}\right), \quad \in \Re^{C_q \times C}$$

$$\text{Att}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \mathbf{o} \circ_2 \mathbf{v}, \quad \in \Re^{C_q \times H \times W}$$



Quest:

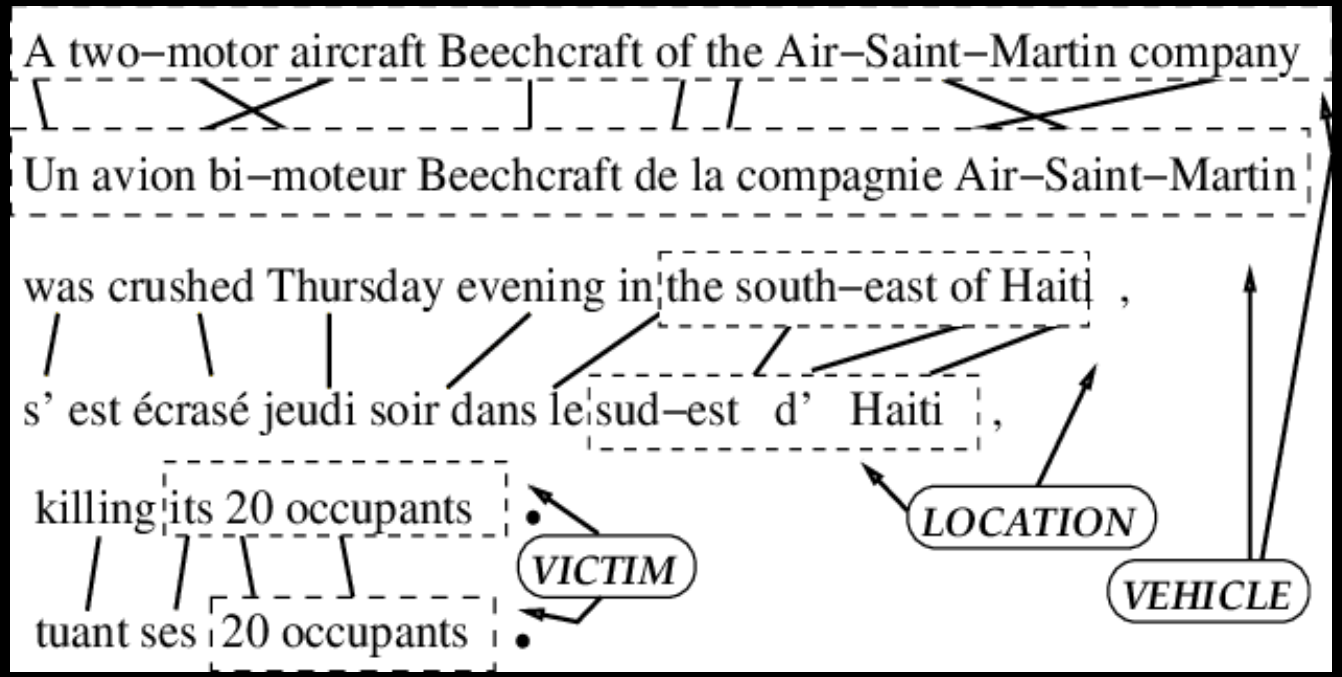
1. Understand Attention… (improve on atrous + use it for comparison)
2. Dot product is not a nice comparison operator (not bounded, not set similarity)
3. Attention (memory) is about set operations: need LAYER set similarity
4. Memory efficient Attention ← dot product

Large number of filters and/or large spatial dimensions → GPU memory problem e.g. ResUNet 6th depth layer (input: 4 x 32 x 256 x 256):

Q (4 x 1024 x 8 x 8)
K (4 x 1024 x 8 x 8)

Dot(Q,K) ~ 4 x 1024 x 1024

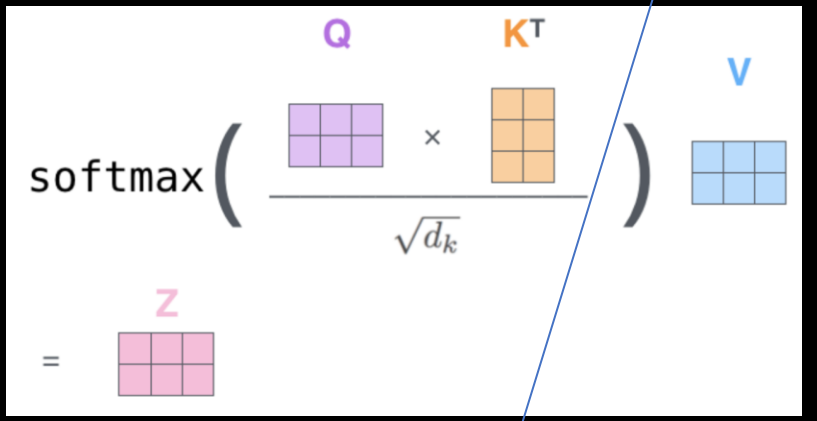
On self Attention from NLP + vision (SAGAN)



Attention (NLP) Vaswani et al. (2017)

$$\mathbf{o} = \text{softmax}\left(\frac{\mathbf{q} \circ_1 \mathbf{k}}{\sqrt{d}}\right), \quad \in \Re^{C_q \times C}$$

$$\text{Att}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \mathbf{o} \circ_2 \mathbf{v}, \quad \in \Re^{C_q \times H \times W}$$



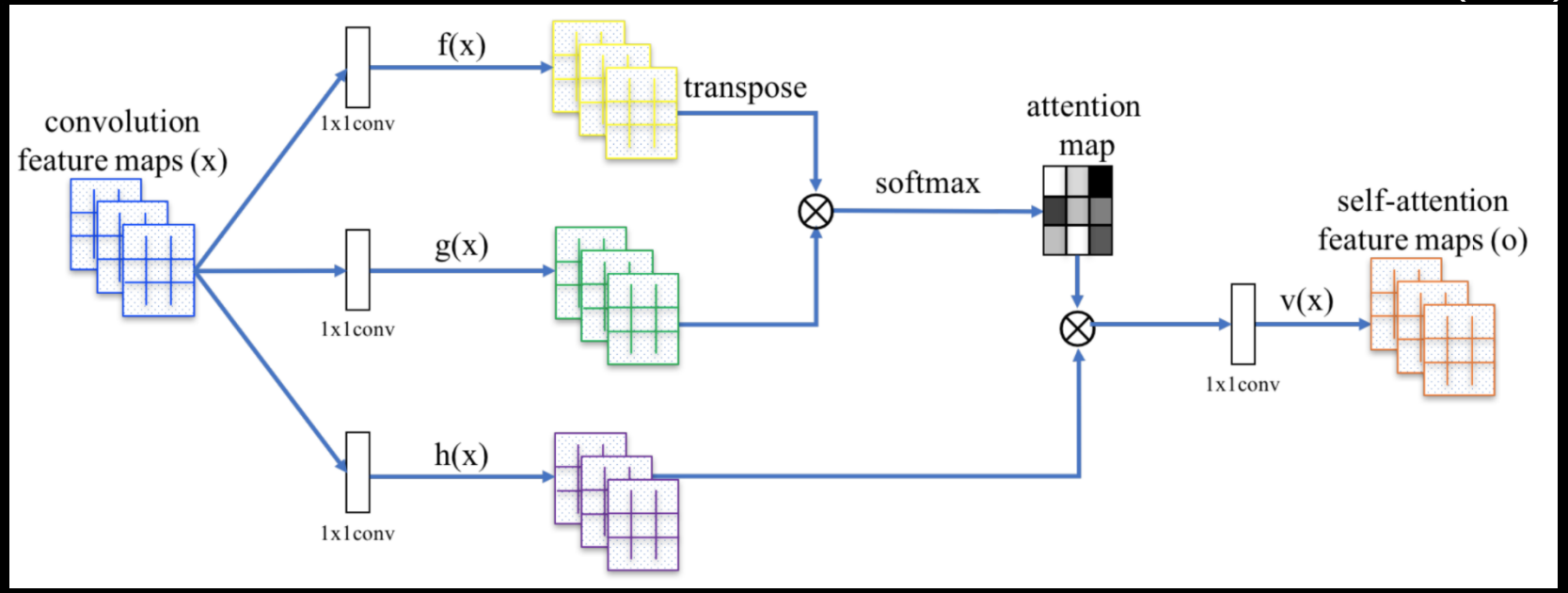
In NMT relative position is important (syntax changes). In vision this is not necessarily the case: element wise channel comparison → memory efficiency

Quest:

1. Understand Attention… (improve on atrous + use it for comparison)
2. Dot product is not a nice comparison operator (not bounded, not set similarity)
3. Attention (memory) is about set operations: need LAYER set similarity
4. Memory efficient Attention 👉 dot product

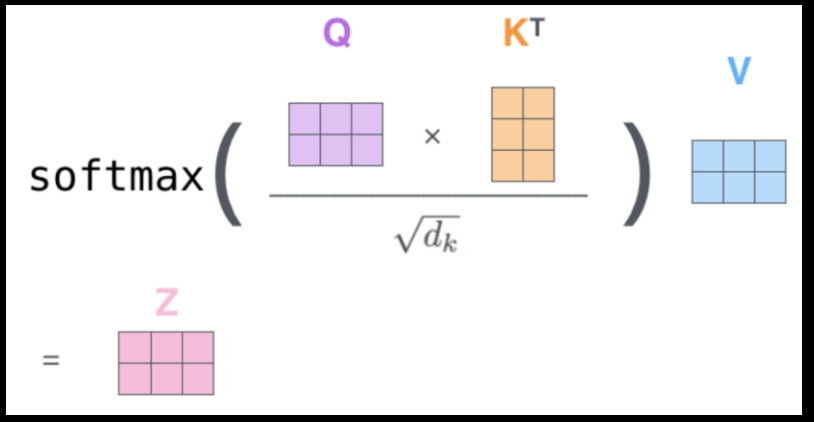
On (traditional) Attention (from NLP)

Attention (NLP) Vaswani et al. (2017)



$$\mathbf{o} = \text{softmax}\left(\frac{\mathbf{q} \circ_1 \mathbf{k}}{\sqrt{d}}\right), \qquad \in \Re^{C_q \times C}$$

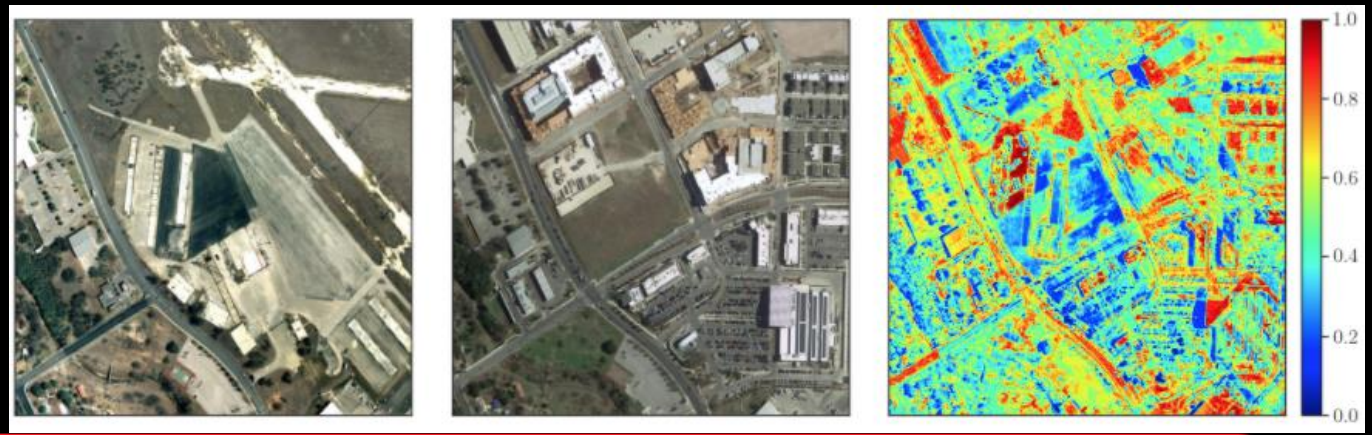
$$\text{Att}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \mathbf{o} \circ_2 \mathbf{v}, \qquad \in \Re^{C_q \times H \times W}$$



Quest:
1. Understand Attention.
2. Dot product is not a nice comparison operator (not bounded, not set similarity)
3. Attention (memory) is about set operations: need LAYER set similarity
4. Memory efficient Attention ← ~~dot product~~

==> New **Layer** (set) **similarity** metric (replace dot)
==> New evolving Loss strategy
==> New Attention module (channel+spatial ==> memory efficient)
==> New attention-based building blocks (x2)
==> New treatment of boundaries
==> New change detection learning macro-topology

Dot product → Tanimoto set similarity (Diakogiannis et al. (2019)



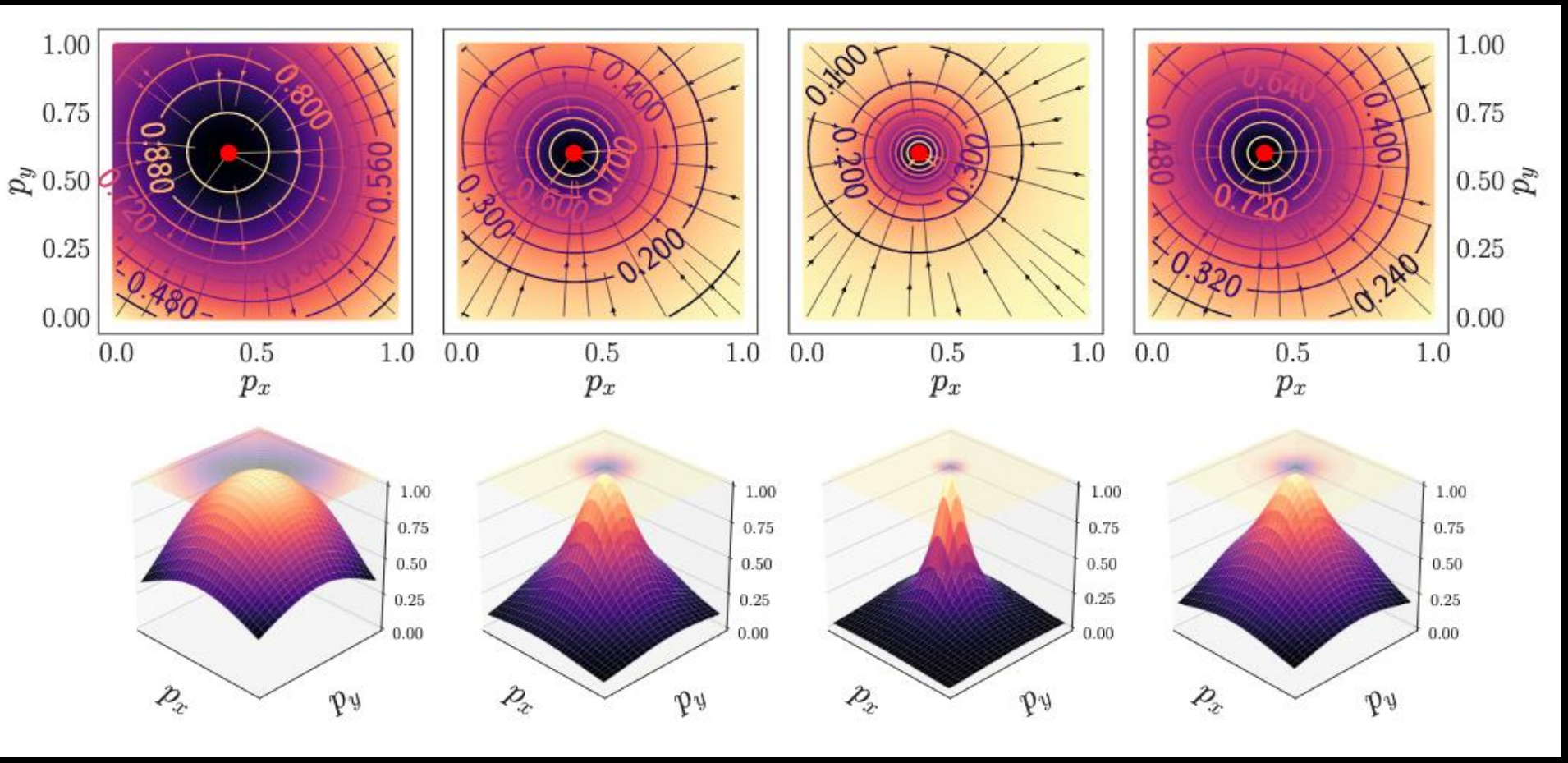
A new set similarity: the FracTAL Tanimoto similarity

$$\mathcal{T}^0 \equiv T(\mathbf{p},\mathbf{l}) = \frac{\mathbf{p}\cdot\mathbf{l}}{\mathbf{p}^2+\mathbf{l}^2-\mathbf{p}\cdot\mathbf{l}}$$

$$\mathcal{T}^d = \frac{\mathcal{T}^{d-1}(\mathbf{p},\mathbf{l})}{\mathcal{T}^{d-1}(\mathbf{p},\mathbf{p})+\mathcal{T}^{d-1}(\mathbf{l},\mathbf{l})-\mathcal{T}^{d-1}(\mathbf{p},\mathbf{l})}$$

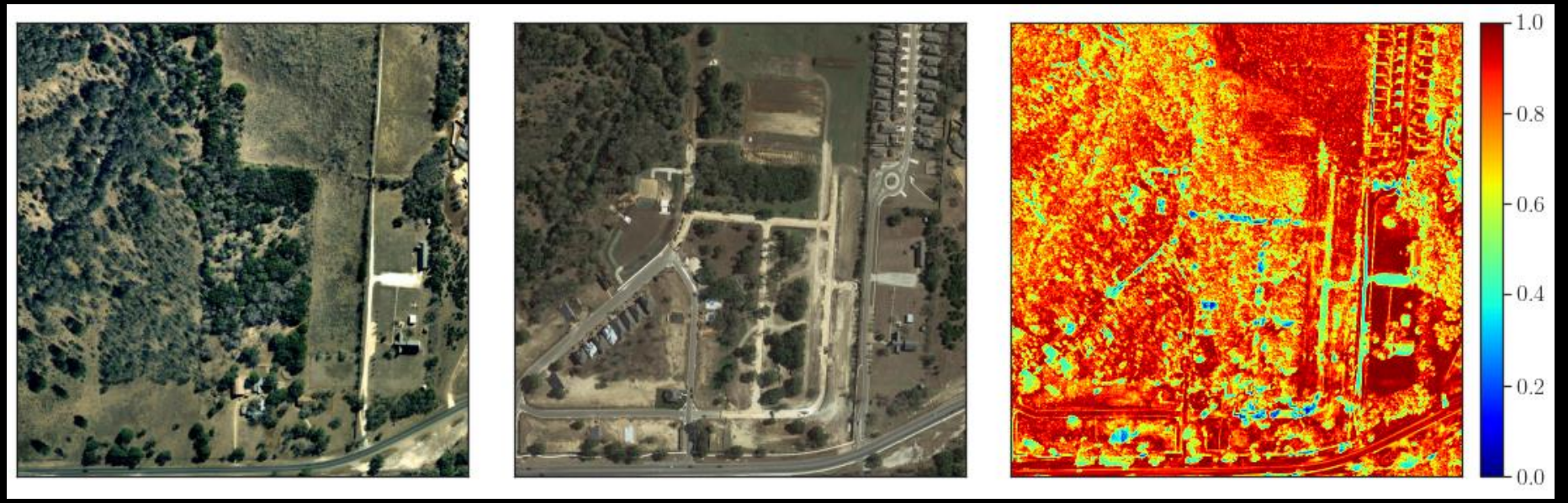
d=3

$$\mathcal{T}^3(\mathbf{p},\mathbf{l}) \frac{\mathbf{p}\cdot\mathbf{l}}{(\mathbf{l}^2-\mathbf{p}\cdot\mathbf{l}+\mathbf{p}^2)\left(2-\frac{\mathbf{p}\cdot\mathbf{l}}{\mathbf{l}^2-\mathbf{p}\cdot\mathbf{l}+\mathbf{p}^2}\right)\left(2-\frac{\mathbf{p}\cdot\mathbf{l}}{(\mathbf{l}^2-\mathbf{p}.\mathbf{l}+\mathbf{p}^2)\left(2-\frac{\mathbf{p}\cdot\mathbf{l}}{\mathbf{l}^2-\mathbf{p}\cdot\mathbf{l}+\mathbf{p}^2}\right)}\right)} \quad (4)$$

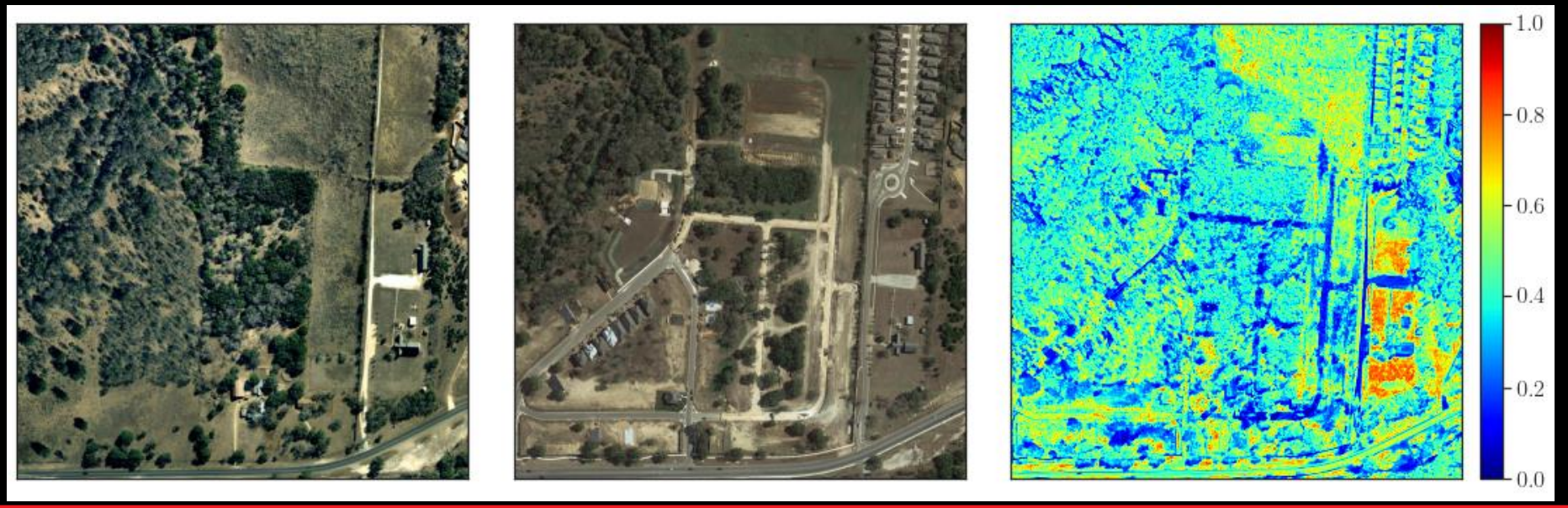


$$\mathcal{T}^d(\mathbf{p},\mathbf{l}) = \frac{\mathbf{p}\cdot\mathbf{l}}{2^d(\mathbf{p}^2+\mathbf{l}^2)-(2^{d+1}-1)\mathbf{p}\cdot\mathbf{l}}$$

Finer layer (set) similarity (or else, why we did all this)



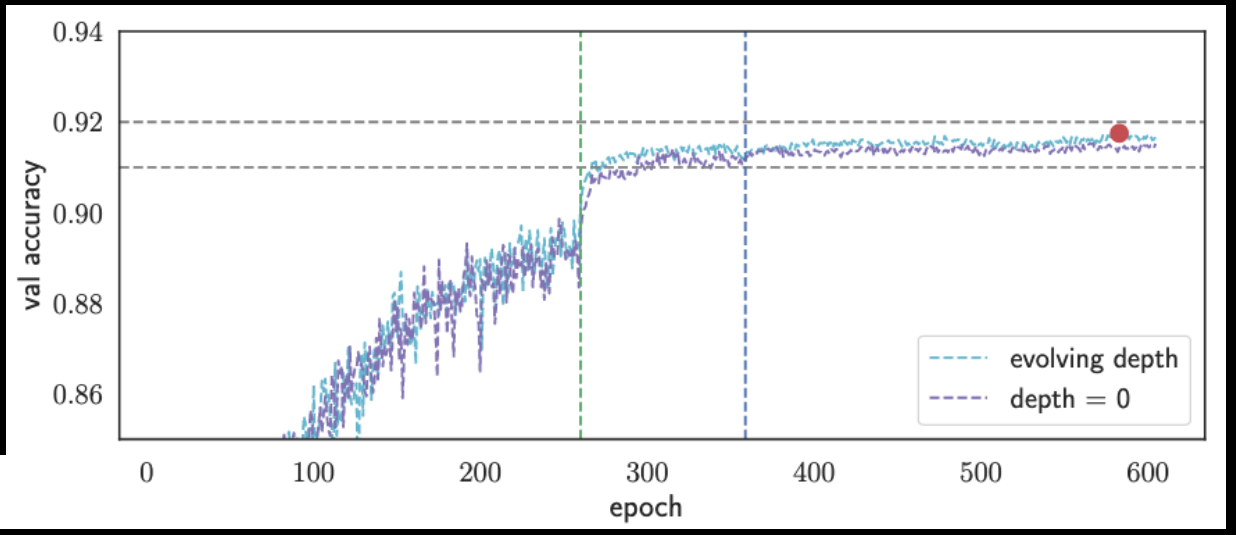
fractal depth=0

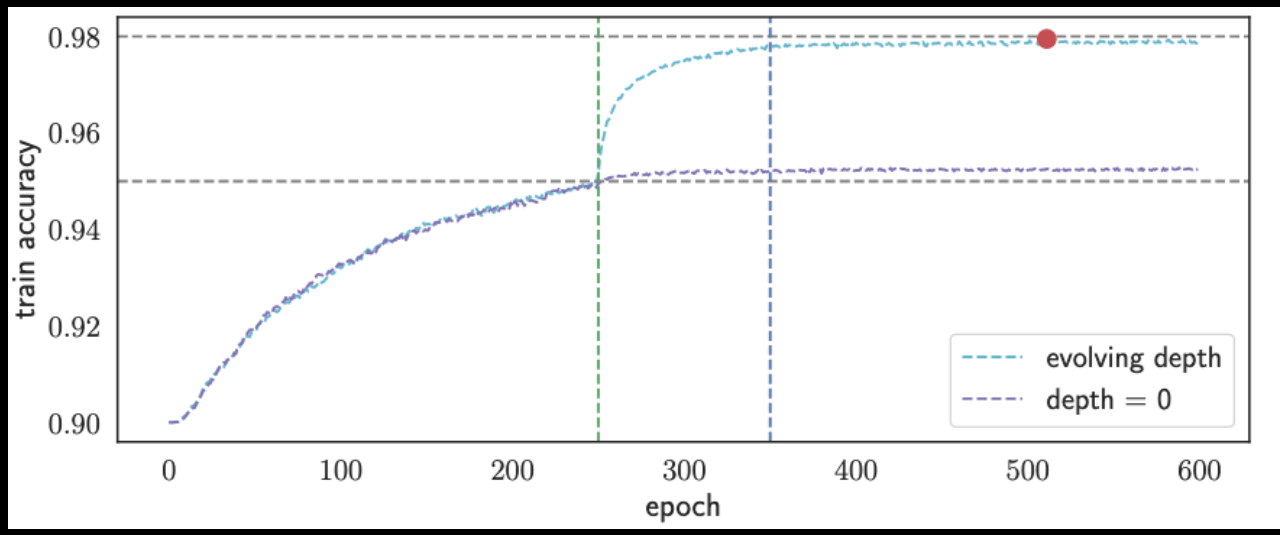


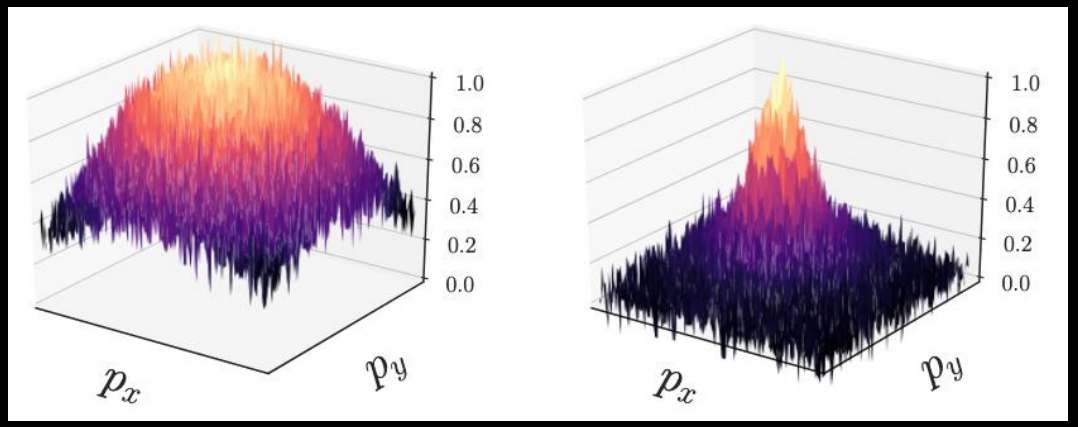
fractal depth=10

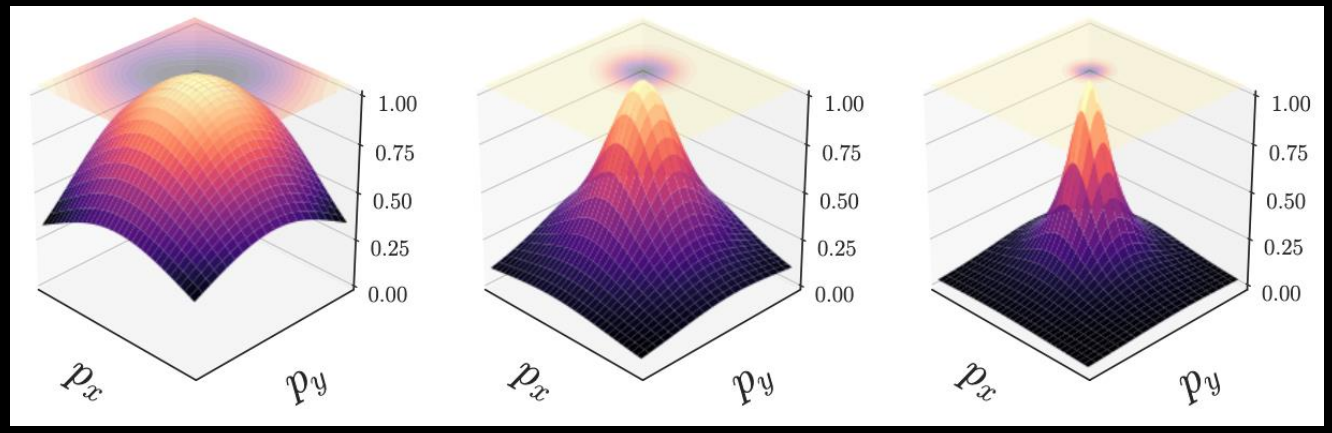
Corollary: from set similarity to evolving loss strategy

$$\mathcal{T}^d(\mathbf{p},\mathbf{l}) = \frac{\mathbf{p}\cdot\mathbf{l}}{2^d(\mathbf{p}^2+\mathbf{l}^2) - (2^{d+1}-1)\mathbf{p}\cdot\mathbf{l}}$$







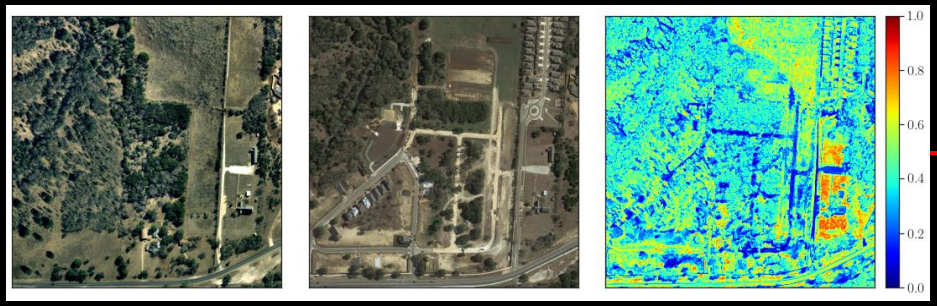


A new channel + spatial attention layer – the FracTAL

$$\mathrm{Att}(\mathbf{q},\mathbf{k},\mathbf{v}) = 0.5(\mathrm{Att}_{\boxtimes} + \mathrm{Att}_{\boxdot})$$

$$\mathcal{T}^d_{\boxtimes}(\mathbf{q},\mathbf{k}) = \frac{\mathbf{q}\boxtimes\mathbf{k}}{2^d(\mathbf{q}\boxtimes\mathbf{q}+\mathbf{k}\boxtimes\mathbf{k}) - (2^{d+1}-1)\mathbf{q}\boxtimes\mathbf{k}} \in \Re^C \quad (11)$$

$$\mathcal{T}^d_{\boxdot}(\mathbf{q},\mathbf{k}) = \frac{\mathbf{q}\boxdot\mathbf{k}}{2^d(\mathbf{q}\boxdot\mathbf{q}+\mathbf{k}\boxdot\mathbf{k}) - (2^{d+1}-1)\mathbf{q}\boxdot\mathbf{k}} \in \Re^{H\times W} \quad (12)$$



IoU – like similarity for binary images

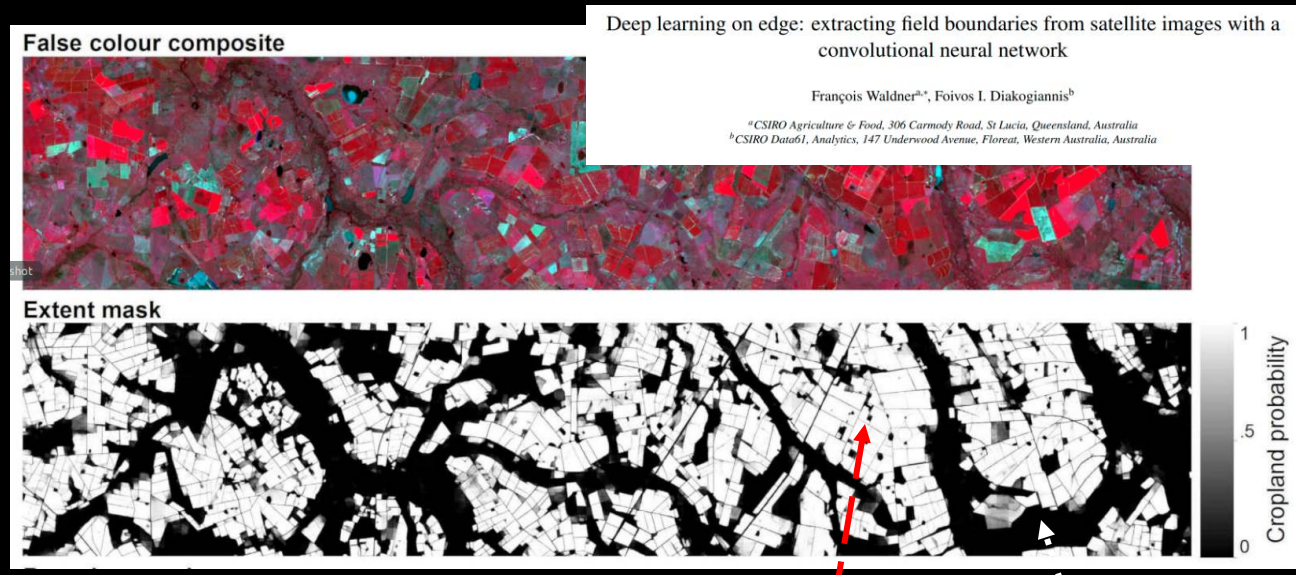
```
class RelFTAttention2D(HybridBlock):
    def __init__(self, nkeys, kernel_size=3, padding=1,nheads=1, norm = 'BatchNorm', norm_groups=None,ftdepth=5,**kwards):
        super().__init__(**kwards)


        self.query   = Conv2DNormed(channels=nkeys,kernel_size= kernel_size, padding = padding, _norm_type= norm, norm_groups=norm_groups)
        self.key     = Conv2DNormed(channels=nkeys,kernel_size= kernel_size, padding = padding, _norm_type= norm, norm_groups=norm_groups)
        self.value   = Conv2DNormed(channels=nkeys,kernel_size= kernel_size, padding = padding, _norm_type= norm, norm_groups=norm_groups)


        self.metric_channel = FTanimoto(depth=ftdepth, axis=[2,3])
        self.metric_space = FTanimoto(depth=ftdepth, axis=1)

        self.norm = get_norm(name=norm, axis=1, norm_groups= norm_groups)

    def forward(self,  input1, input2, input3):

        # These should work with ReLU as well
        q = mx.npx.sigmoid(self.query(input1))
        k = mx.npx.sigmoid(self.key(input2))# B,C,H,W
        v = mx.npx.sigmoid(self.value(input3)) # B,C,H,W

        att_spat =  self.metric_space(q,k) # B,1,H,W
        v_spat   =  att_spat * v # emphasize spatial features

        att_chan =  self.metric_channel(q,k) # B,C,1,1
        v_chan   =  att_chan * v # emphasize spatial features


        v_cspat =  0.5*(v_chan + v_spat)
        v_cspat = self.norm(v_cspat)

        return v_cspat
```

$$\mathrm{Att}_{\boxtimes}(\mathbf{q},\mathbf{k},\mathbf{v}) = \mathcal{T}^d_{\boxtimes}(\mathbf{q},\mathbf{k}) \odot \mathbf{v}$$
$$\mathrm{Att}_{\boxdot}(\mathbf{q},\mathbf{k},\mathbf{v}) = \mathcal{T}^d_{\boxdot}(\mathbf{q},\mathbf{k}) \odot \mathbf{v}$$

```
from mxnet.gluon import nn
class FTanimoto(nn.Block):
    def __init__(self,depth=5, axis=[2,3],**kwards):
        super().__init__(**kwards)
        self.depth = depth
        self.axis=axis

    def inner_prod(self, prob, label):
        prdct = prob*label #dim:(B,C,H,W)
        prdct = prdct.sum(axis=self.axis,keepdims=True)
        return prdct #dim:(B,C,1,1)

    def forward(self, prob, label):
        a = 2.**self.depth
        b = -(2.*a-1.)

        tpl= self.inner_prod(prob,label)
        tpp= self.inner_prod(prob,prob)
        tll= self.inner_prod(label,label)

        denum = a*(tpp+tll)+b*tpl
        ftnmt = tpl/denum
        return ftnmt #dim:(B,C,1,1)
```

Self attention: how do you combine the FracTAL with a features layer?

```
def forward(self,  input1, input2, input3):

    # These should work with ReLU as well
    q = mx.npx.sigmoid(self.query(input1))
    k = mx.npx.sigmoid(self.key(input2))# B,C,H,W
    v = mx.npx.sigmoid(self.value(input3)) # B,C,H,W

    att_spat =  self.metric_space(q,k) # B,1,H,W
    v_spat   =  att_spat * v # emphasize spatial features

    att_chan =  self.metric_channel(q,k) # B,C,1,1
    v_chan   =  att_chan * v # emphasize spatial features


    v_cspat =   0.5*(v_chan + v_spat) # emphasize spatial features
    v_cspat = self.norm(v_cspat)

    return v_cspat
```



Erases information where A ➔ 0

1s

0s

$$\mathbf{F} = \mathbf{L} + \gamma \mathbf{L} \odot \mathbf{A} = \mathbf{L} \odot (1 + \gamma \mathbf{A})$$

Start without attention $\gamma_0 = 0$ (SAGAN)

Relative attention: how do you combine the FracTAL with two feature layers?

```
def forward(self,  input1, input2, input3):

    # These should work with ReLU as well
    q = mx.npx.sigmoid(self.query(input1))
    k = mx.npx.sigmoid(self.key(input2))# B,C,H,W
    v = mx.npx.sigmoid(self.value(input3)) # B,C,H,W

    att_spat =  self.metric_space(q,k) # B,1,H,W
    v_spat   =  att_spat * v # emphasize spatial features

    att_chan =  self.metric_channel(q,k) # B,C,1,1
    v_chan   =  att_chan * v # emphasize spatial features


    v_cspat =   0.5*(v_chan + v_spat) # emphasize spatial features
    v_cspat = self.norm(v_cspat)

    return v_cspat
```

What is important on Layer 1 (resp. 2) based o

Information that exists on Layer 2 (resp. 1)?

$$\mathbf{F}_1 = \mathbf{L}_1 \odot [\mathbf{1} + \gamma_1 \mathbf{A}_{122}(\mathbf{q}(\mathbf{L}_1), \mathbf{k}(\mathbf{L}_2), \mathbf{v}(\mathbf{L}_2))] \quad (15)$$

$$\mathbf{F}_2 = \mathbf{L}_2 \odot [\mathbf{1} + \gamma_2 \mathbf{A}_{211}(\mathbf{q}(\mathbf{L}_2), \mathbf{k}(\mathbf{L}_1), \mathbf{v}(\mathbf{L}_1))] \quad (16)$$

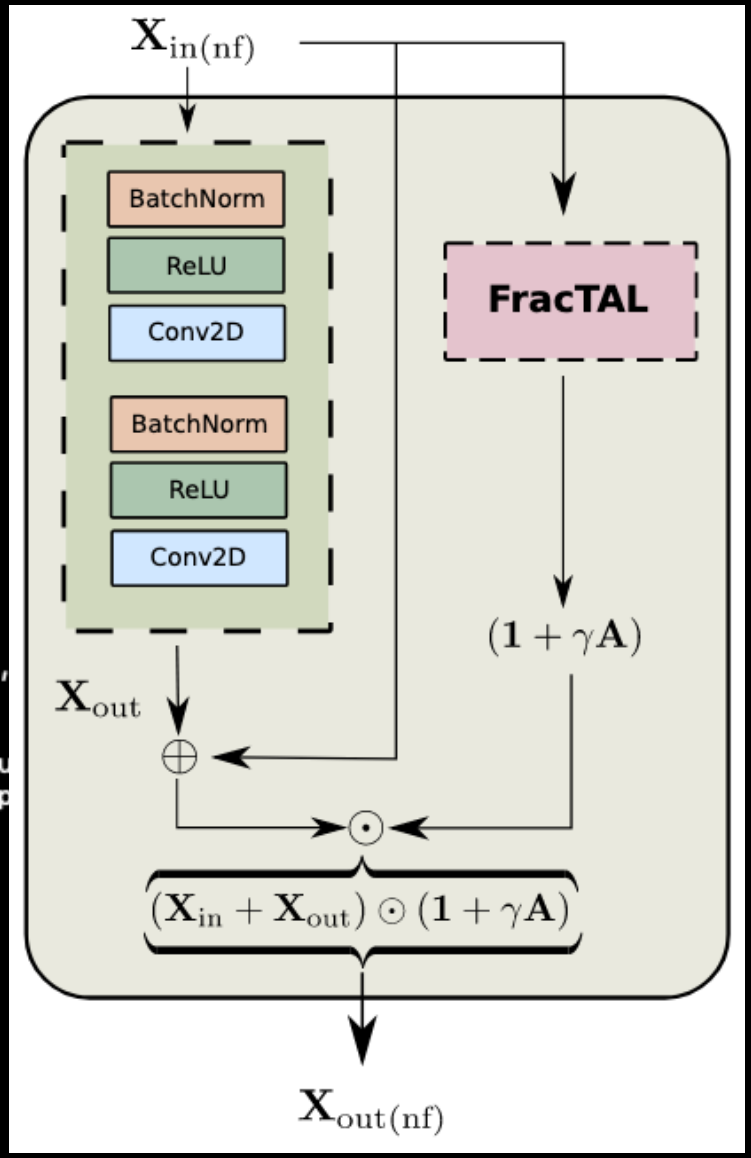
$$\mathbf{F} = \mathrm{Conv2DN}(\mathrm{concat}([\mathbf{F}_1, \mathbf{F}_2])) \quad (17)$$

CEECNet & FracTAL ResNet micro-topologies (units)

FracTAL ResNet (Robin)
Based on self attention ideas

Robin

```python
class FracTALResNet_unit(HybridBlock):
    def __init__(self, nfilters, ngroups=1, nheads=1, kernel_size=(3,3), dilation_rate=(1,1), norm_type = 'BatchNorm',
        super().__init__(**kwards)

        self.block1 = ResNet_v2_block(nfilters,kernel_size,dilation_rate,_norm_type = norm_type, norm_groups=norm_grou
        self.attn = FTAttention2D(nkeys=nfilters, nheads=nheads, kernel_size=kernel_size, norm = norm_type, norm_group

        self.gamma  = gluon.Parameter('gamma', shape=(1,), init=mx.init.Zero())

    def forward(self, input):
        out1 = self.block1(input)

        att2 = self.attn(input)
        att2 = self.gamma.data() * att2

        out  = (input + out1) * (mx.np.ones_like(out1) + att2)
        return out
```



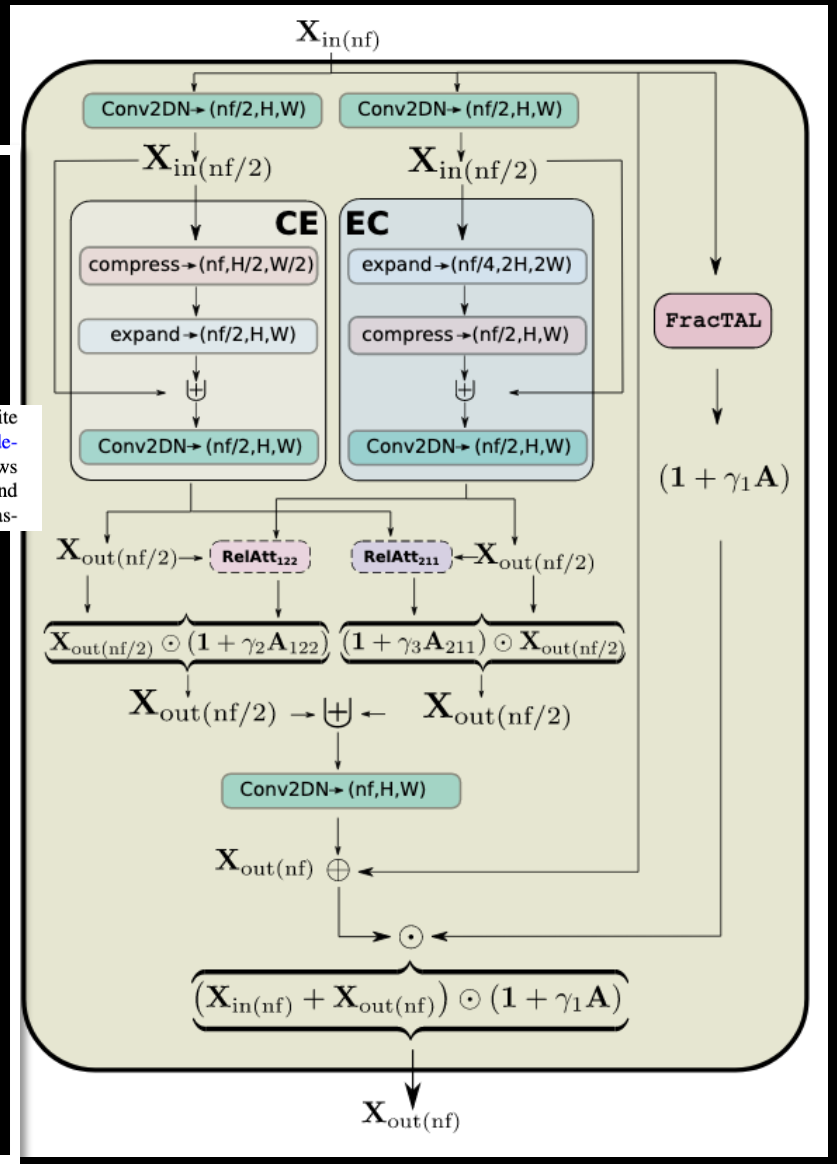
CEECNet & FracTAL ResNet micro-topologies (units)

CEECNet – relative + self FracTAL

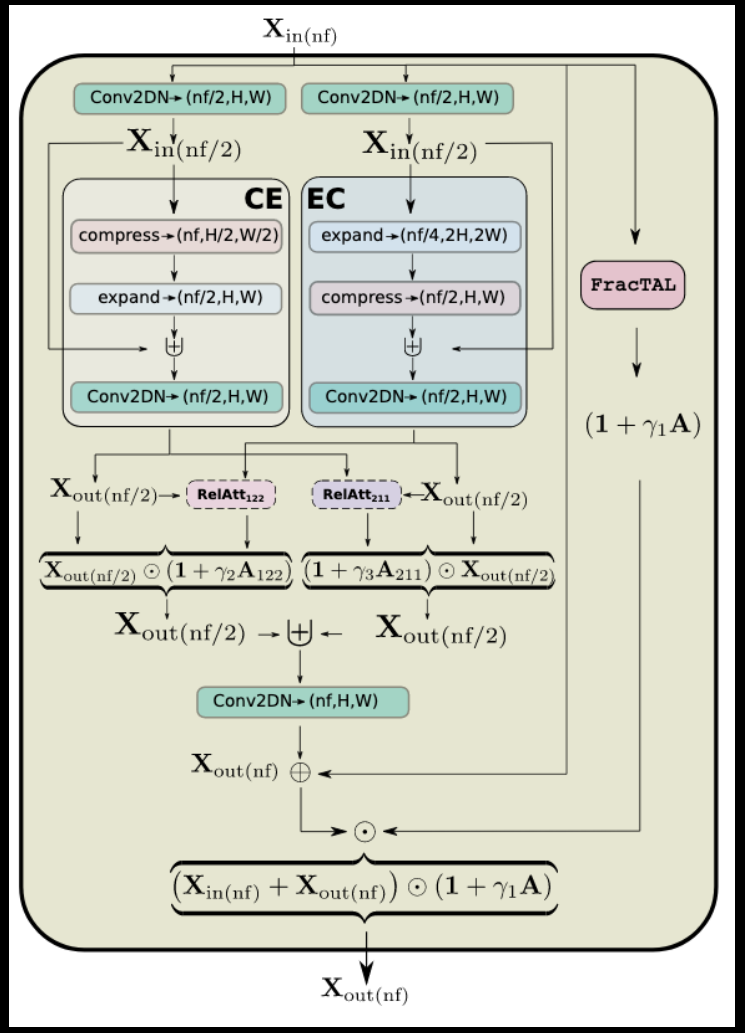
Batman

```python
    def forward(self, input):

        # ========== UNet branch ==========
        out10 = self.conv_init_1(input)
        out1 = self.compr11(out10)
        out1 = mx.npx.relu(out1)
        out1 = self.compr12(out1)
        out1 = mx.npx.relu(out1)
        out1 = self.expand1(out1,out10)
        out1 = mx.npx.relu(out1)


        # ========== \capNet branch ======

        out20 = self.conv_init_2(input)
        out2 = self.expand2(out20)
        out2 = mx.npx.relu(out2)
        out2 = self.compr21(out2)
        out2 = mx.npx.relu(out2)
        out2 = self.compr22(mx.np.concatenate([out2,out20],axis=1))
        out2 = mx.npx.relu(out2)

        att  = self.gamma1.data() * self.att(input)
        ratt122 = self.gamma2.data() * self.ratt122(out1,out2,out2)
        ratt211 = self.gamma3.data() * self.ratt211(out2,out1,out1)

        ones1 = mx.np.ones_like(out10)
        ones2 = mx.np.ones_like(input)

        # Enhanced output of 1, based on memory of 2
        out122 = out1 * (ones1 + ratt122)
        # Enhanced output of 2, based on memory of 1
        out211 = out2 * (ones1 + ratt211)

        out12 = mx.npx.relu(self.collect(mx.np.concatenate([out122,out211],axis=1)))

        # Emphasize residual output from memory on input
        out_res = (input + out12) * ( ones2 + att)
        return out_res
```

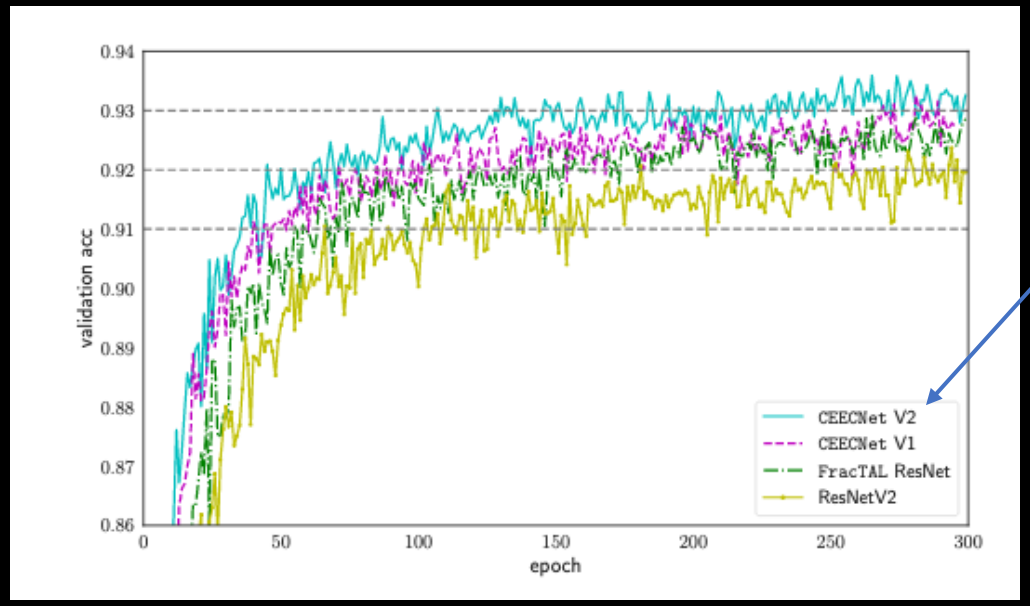
Our hypothesis and motivation for this approach is quite similar to the scale-space analysis in computer vision (Lindeberg, 1994): viewing input features at different scales, allows the algorithm to focus on different aspects of the inputs, and thus perform more efficiently. The fact that by merely increas-



CEECNet & FracTAL ResNet (cifar10 test)

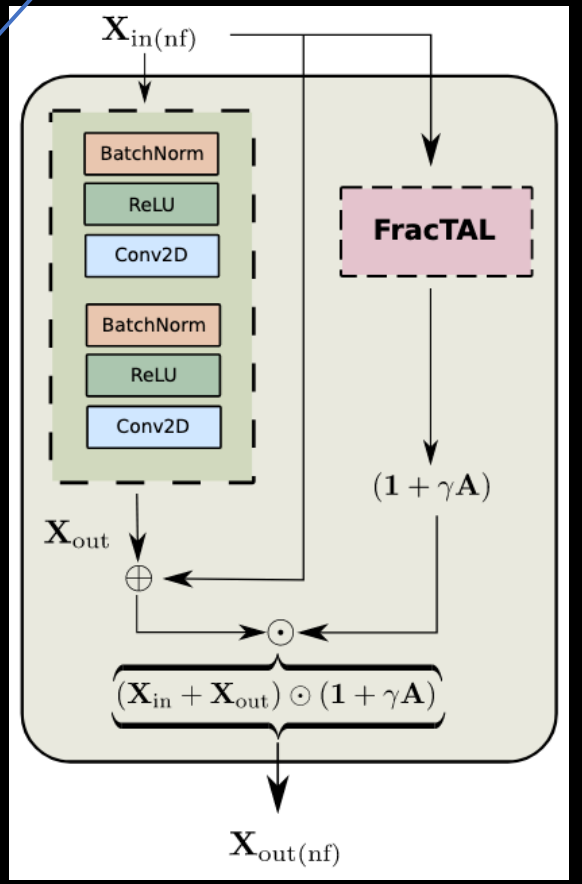


Batman



(there exist also a Hulk...)

Robin



How do we consume/compare two inputs? The mantis macro-topology

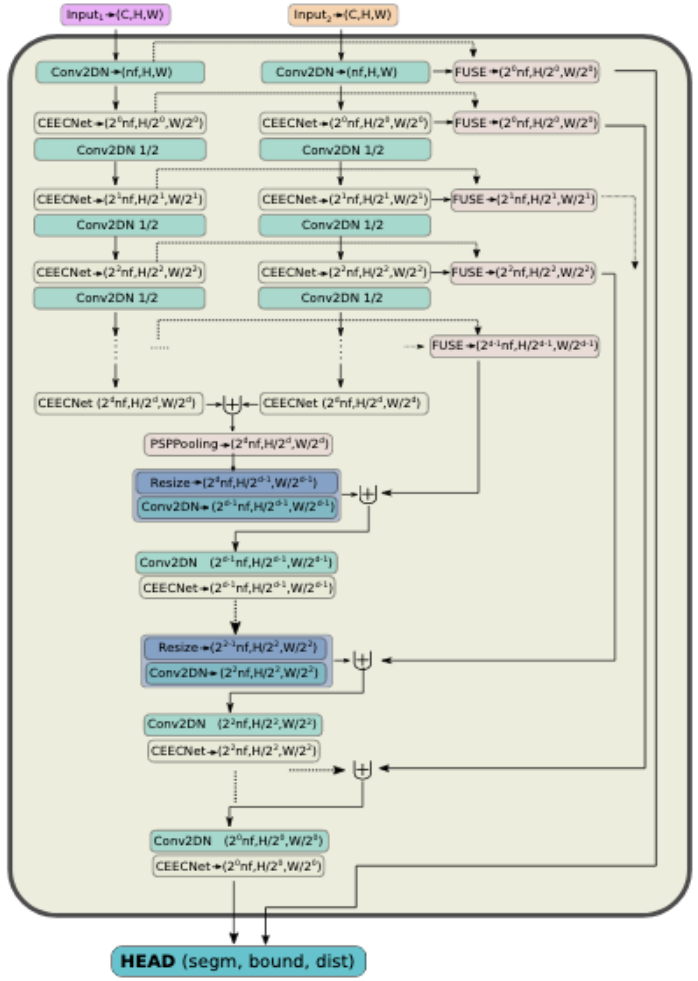


Figure 6: The `mantis CEECNetV1` architecture for the task of change detection. The Fusion operation (FUSE) is described with MXNET/GLUON style pseudocode in detail on Listing 3.

Inductive bias: What is important on image 1 (resp. based on Information that exists on image 2 (r

$$\mathbf{F}_1 = \mathbf{L}_1 \odot [\mathbf{1} + \gamma_1 \mathbf{A}_{122}(\mathbf{q}(\mathbf{L}_1), \mathbf{k}(\mathbf{L}_2), \mathbf{v}(\mathbf{L}_2))] \quad (15)$$

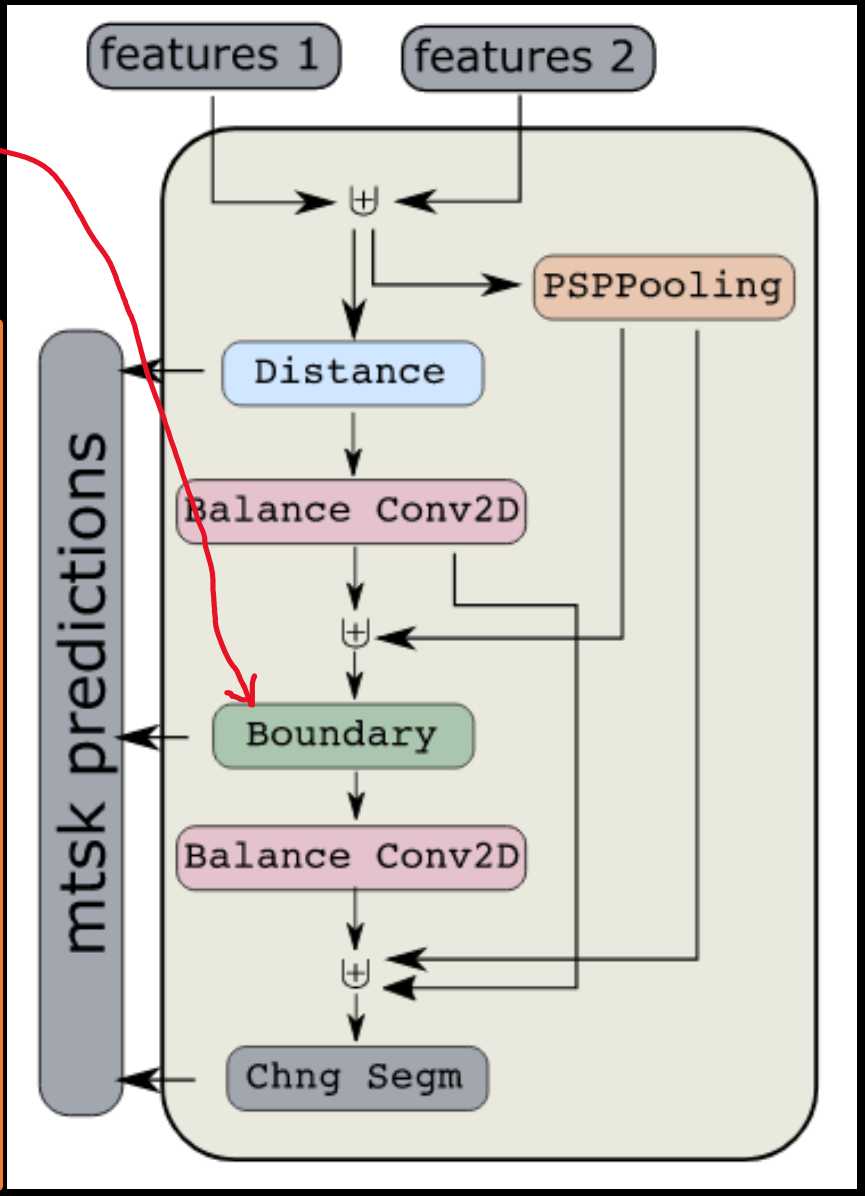
$$\mathbf{F}_2 = \mathbf{L}_2 \odot [\mathbf{1} + \gamma_2 \mathbf{A}_{211}(\mathbf{q}(\mathbf{L}_2), \mathbf{k}(\mathbf{L}_1), \mathbf{v}(\mathbf{L}_1))] \quad (16)$$

$$\mathbf{F} = \text{Conv2DN}(\text{concat}([\mathbf{F}_1, \mathbf{F}_2])) \quad (17)$$

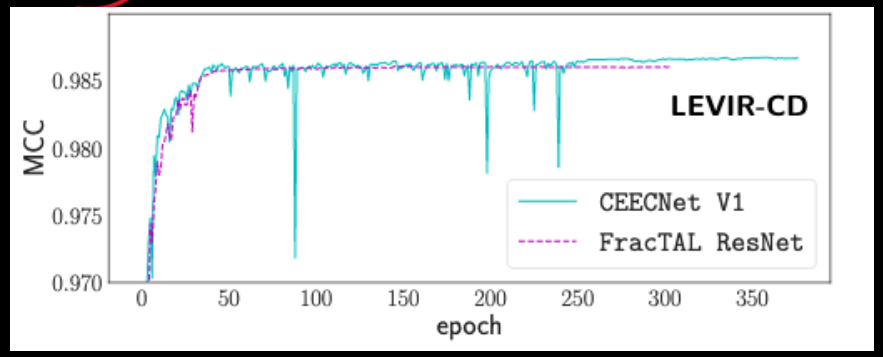
Improved (balanced) “causal” multitasking head + boundaries

$$\mathrm{sigmoid}_{\mathrm{crisp}}(x) = \mathrm{sigmoid}(x/\gamma), \quad \gamma \in [\epsilon, 1]$$

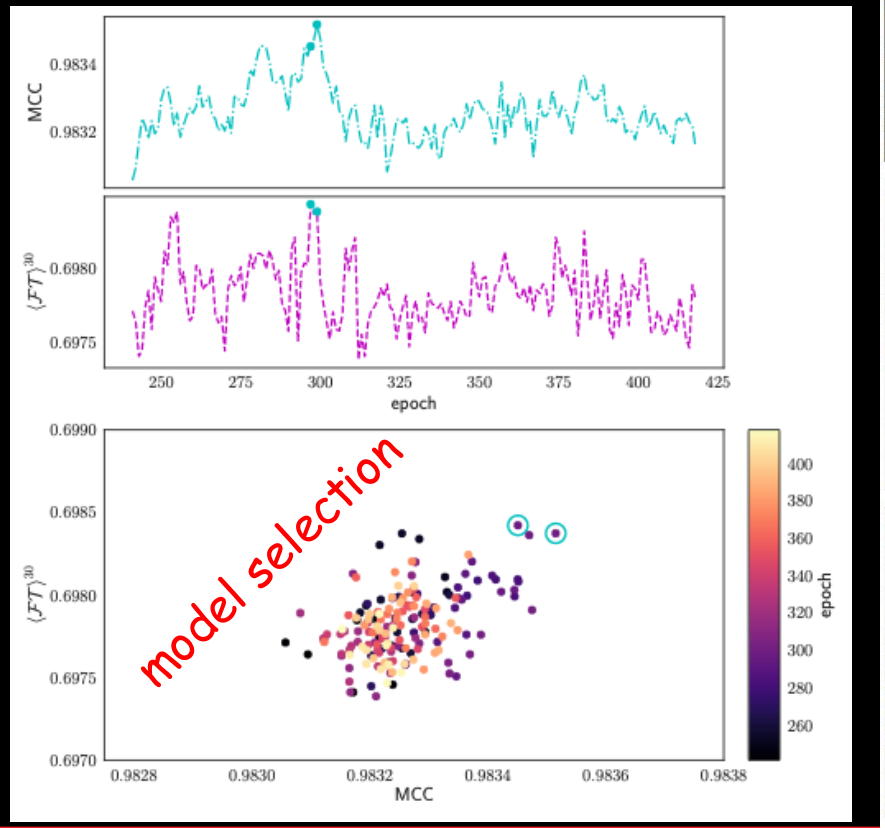
```python
    def forward(self, UpConv4, conv1):
        # second last layer
        conv1 = mx.np.concatenate([conv1,UpConv4],axis=1)
        conv = self.psp_2ndlast(conv1)
        conv = mx.npx.relu(conv)

        # logits

        # 1st find distance map, skeleton like, topology info
        dist = self.distance_logits(conv1) # do not use max pooling for distance
        dist = self.ChannelAct(dist)
        distEq = mx.npx.relu(self.dist_Equalizer(dist)) # makes nfilters equals to conv and conv1


        # Then find boundaries
        bound = mx.np.concatenate([conv, distEq],axis=1)
        bound = self.bound_logits(bound)
        bound    = self.CrispSigm(bound) # Boundaries are not mutually exclusive
        boundEq = mx.npx.relu(self.bound_Equalizer(bound))


        # Now combine all predictions in a final segmentation mask
        # Balance first boundary and distance transform, with the features
        comb_bd = self.Comb_bound_dist(mx.np.concatenate([boundEq, distEq],axis=1))
        comb_bd = mx.npx.relu(comb_bd)

        all_layers = mx.np.concatenate([comb_bd, conv],axis=1)
        final_segm = self.final_segm_logits(all_layers)
        final_segm = self.ChannelAct(final_segm)


        return  final_segm, bound, dist
```

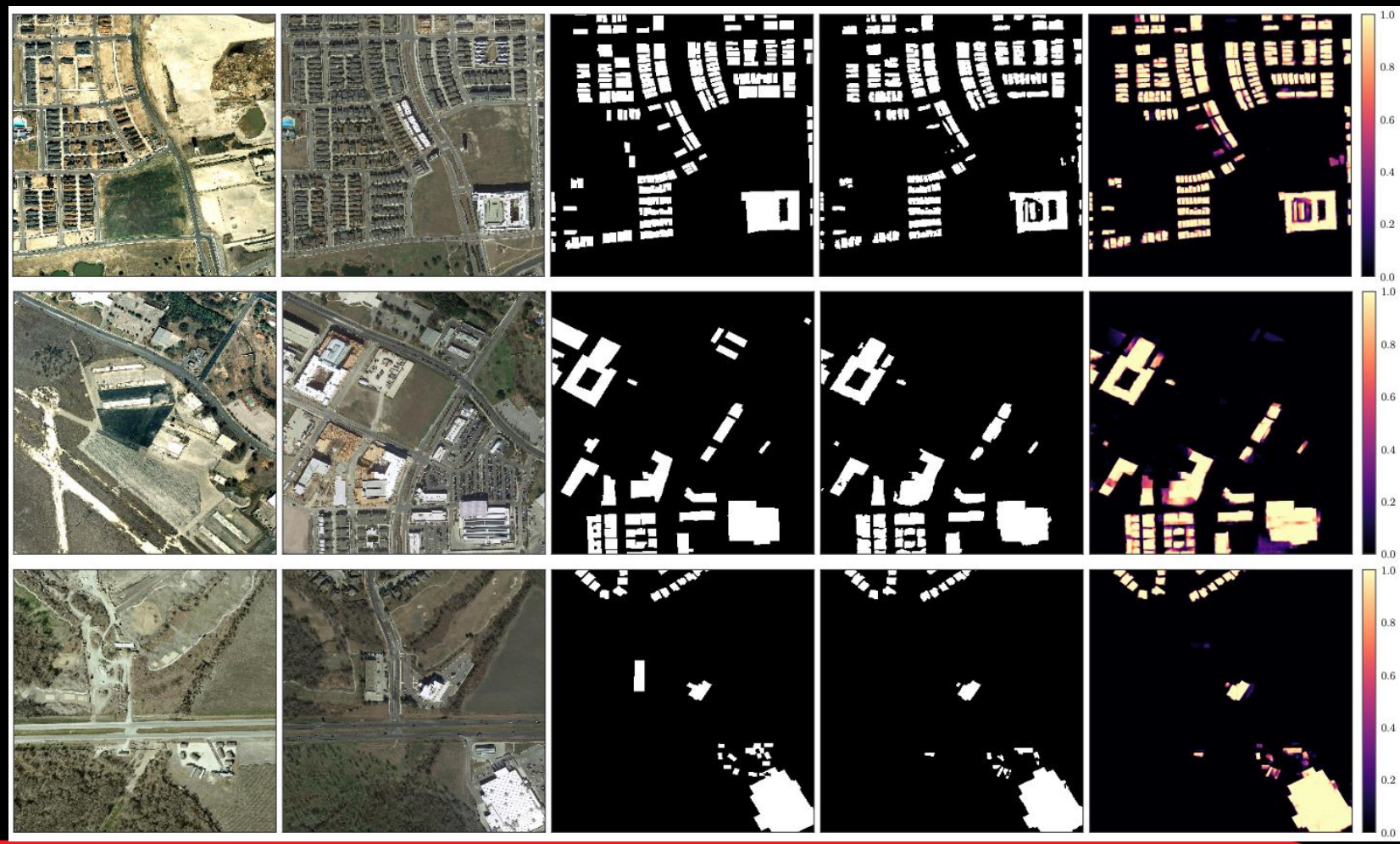


Change detection: D6nf32 performance on LEVIRCD (achieves SOTA)

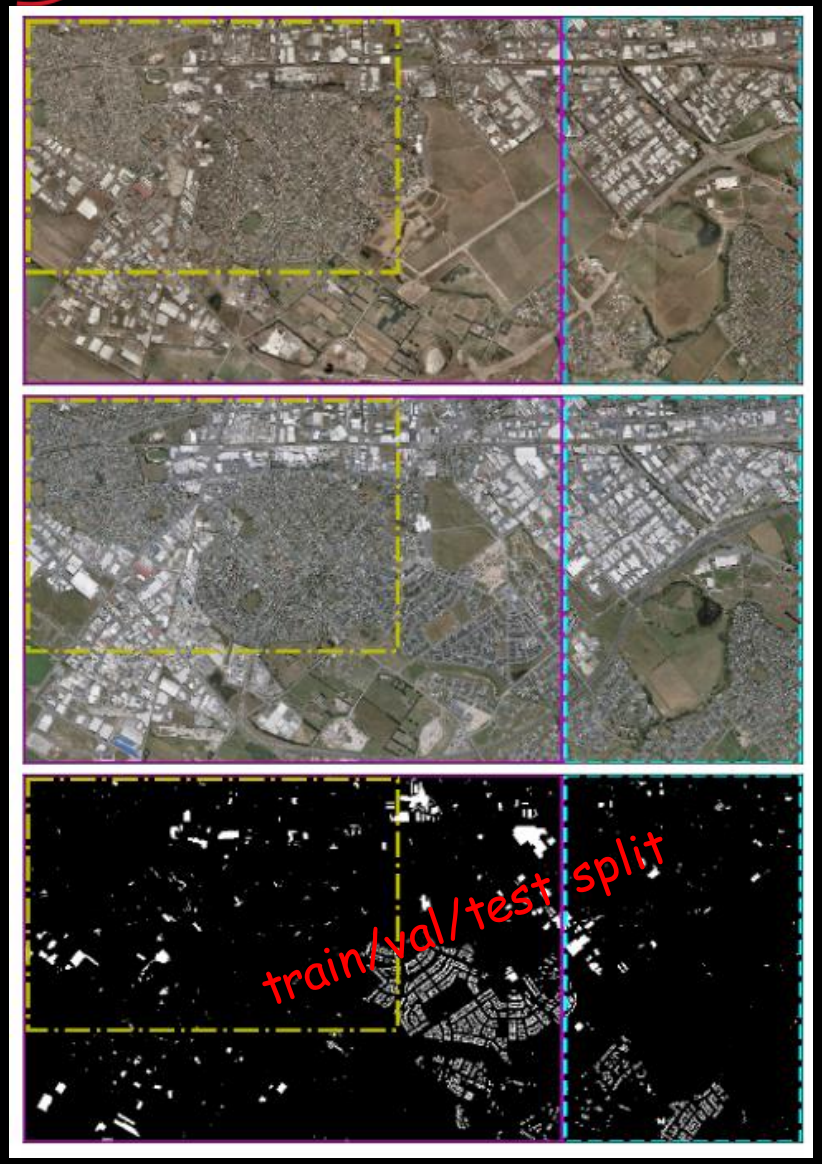


Model	Precision	Recall	F1	MCC	IoU
Chen and Shi (2020)	83.80	**91.00**	87.30	-	-
FracTAL ResNet	93.60	89.38	91.44	91.02	84.23
CEECNetV1	**93.73**	89.93	**91.79**	**91.38**	**84.82**

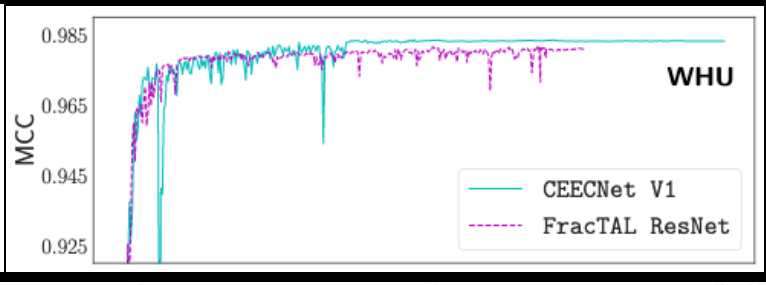


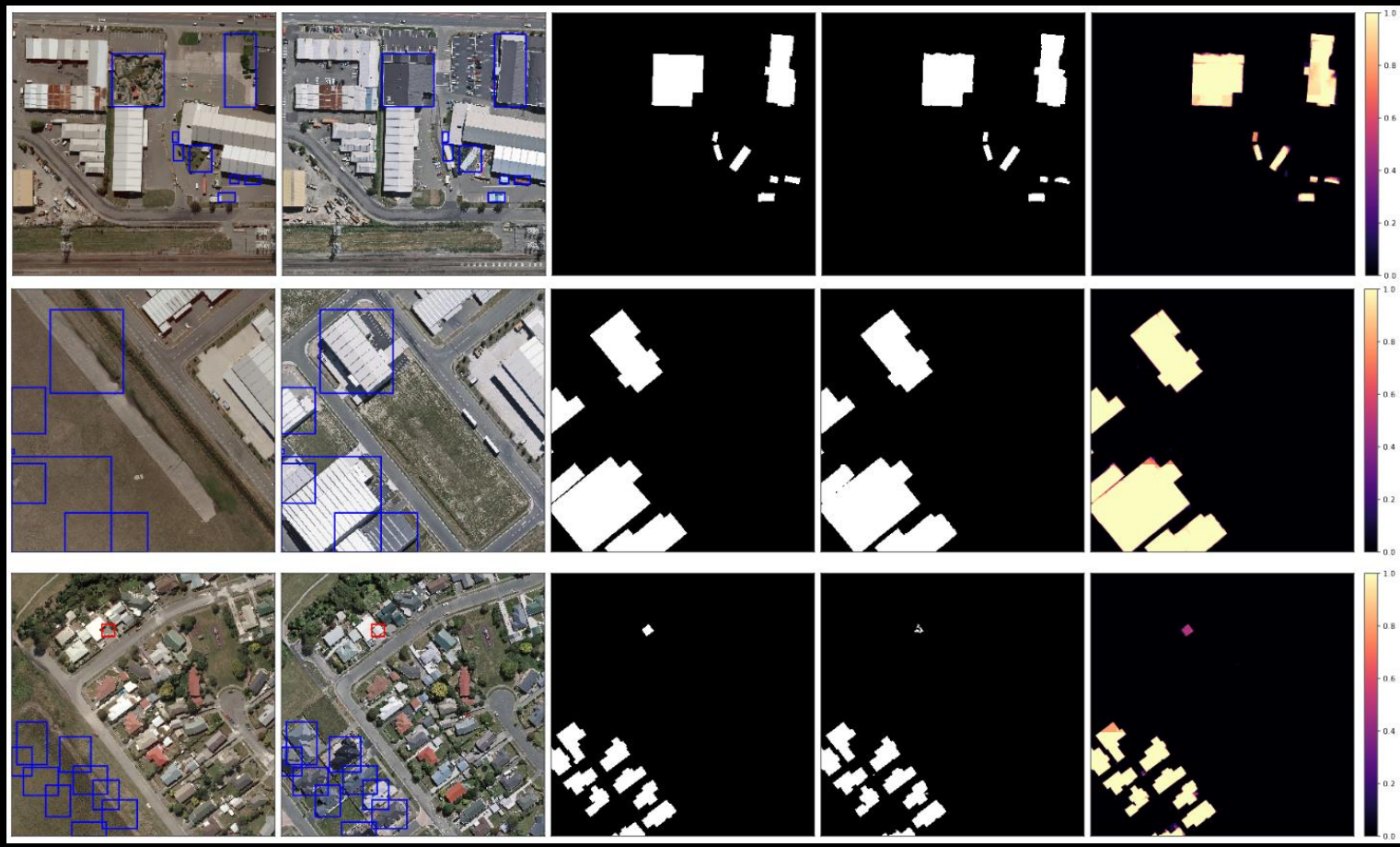


Change detection: D6nf32 performance on WHU (achieves SOTA*)

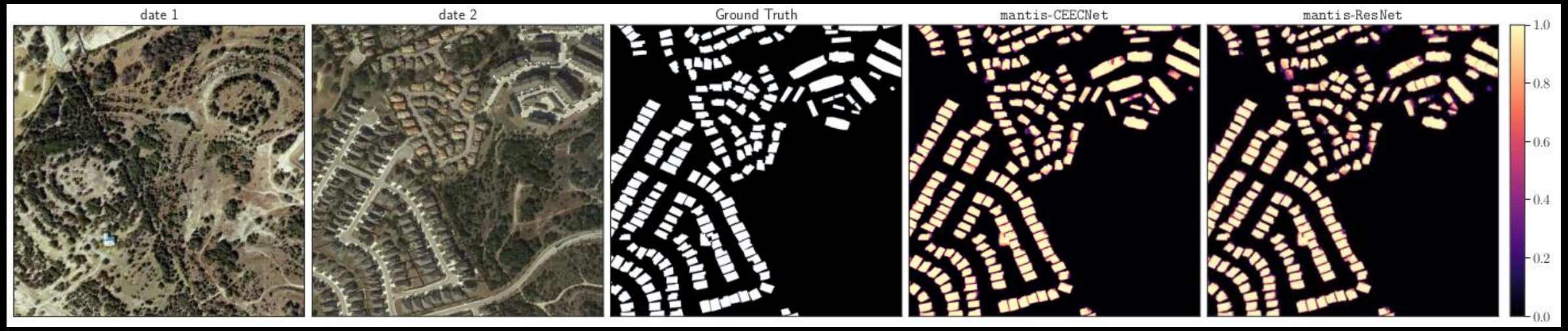


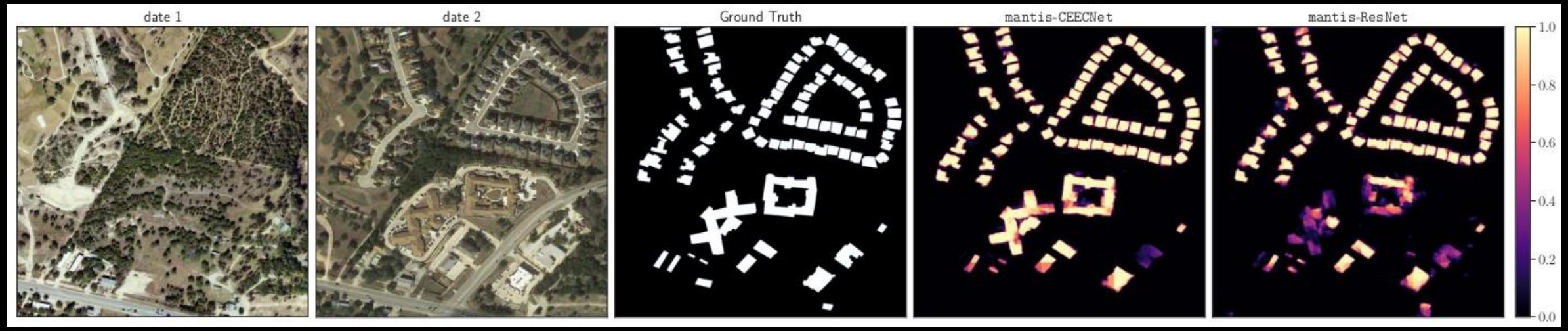
Model	Precision	Recall	F1	MCC	IoU
Ji et al. (2019a) M1	93.100	89.200	91.108	-	83.70
Ji et al. (2019a) M2	93.800	87.800	90.700	-	83.00
Chen et al. (2020)	89.2	90.5	89.80	-	-
Cao et al. (2020)	94.00	79.37	86.07	-	-
Liu et al. (2019)	90.15	89.35	89.75	-	81.40
FracTAL ResNet	95.350	90.873	93.058	92.892	87.02
CEECNetV1	**95.571**	**92.043**	**93.774**	**93.616**	**88.23**





CEECNet V1 vs FracTAL ResNet performance





Not just change: Fusion

The Fusion method is generic: not just for change detection!

Data Fusion of two strongly correlated types of inputs

(replace all your concat operations)

Inductive bias: What is important on "image" 1 (resp. 2) based on Information that exists on "image" 2 (resp. 1)?

$$\mathbf{F}_1 = \mathbf{L}_1 \odot [\mathbf{1} + \gamma_1 \mathbf{A}_{122}(\mathbf{q}(\mathbf{L}_1), \mathbf{k}(\mathbf{L}_2), \mathbf{v}(\mathbf{L}_2))] \quad (15)$$

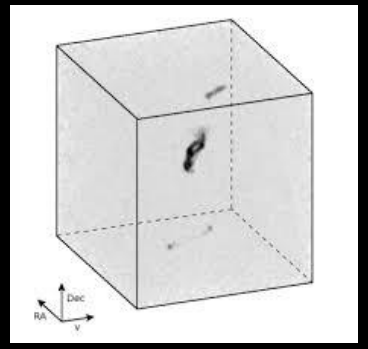
$$\mathbf{F}_2 = \mathbf{L}_2 \odot [\mathbf{1} + \gamma_2 \mathbf{A}_{211}(\mathbf{q}(\mathbf{L}_2), \mathbf{k}(\mathbf{L}_1), \mathbf{v}(\mathbf{L}_1))] \quad (16)$$

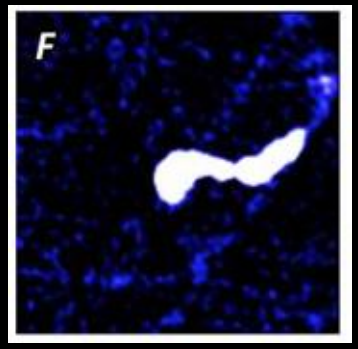
$$\mathbf{F} = \text{Conv2DN}(\text{concat}([\mathbf{F}_1, \mathbf{F}_2])) \quad (17)$$

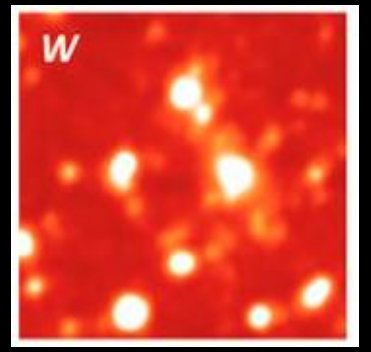
Relational inductive biases, deep learning, and graph networks

Peter W. Battaglia[1]*, Jessica B. Hamrick[1], Victor Bapst[1], Alvaro Sanchez-Gonzalez[1], Vinicius Zambaldi[1], Mateusz Malinowski[1], Andrea Tacchetti[1], David Raposo[1], Adam Santoro[1], Ryan Faulkner[1], Caglar Gulcehre[1], Francis Song[1], Andrew Ballard[1], Justin Gilmer[2], George Dahl[2], Ashish Vaswani[2], Kelsey Allen[3], Charles Nash[4], Victoria Langston[1], Chris Dyer[1], Nicolas Heess[1], Daan Wierstra[1], Pushmeet Kohli[1], Matt Botvinick[1], Oriol Vinyals[1], Yujia Li[1], Razvan Pascanu[1]

[1]DeepMind; [2]Google Brain; [3]MIT; [4]University of Edinburgh







ClaRAN3D
(under construction)

Thank you

Interesting DL problem (1D/2D/3D)?

Talk to us!

foivos.diakogiannis@icrar.org