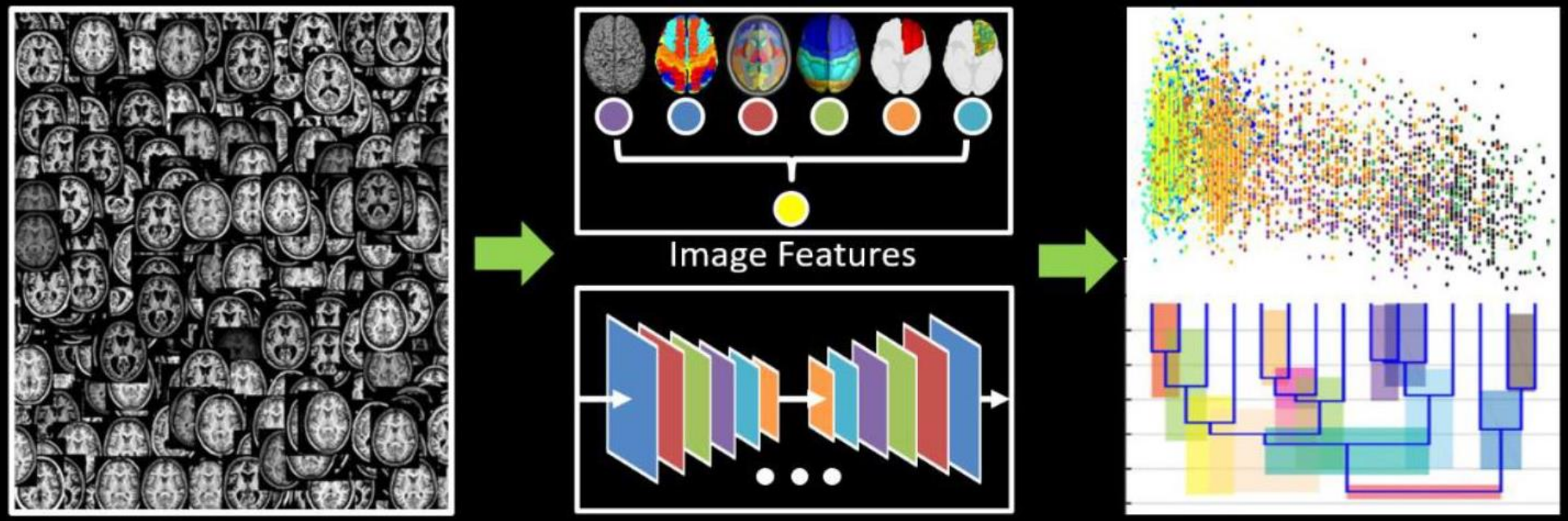


Beyond 2D: 3D Deep Networks

[Spring 2020 CS-8395 Deep Learning in Medical Image Computing]

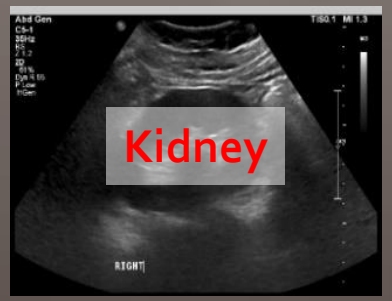
Instructor: Yuankai Huo, Ph.D.
Department of Electrical Engineering and Computer Science
Vanderbilt University

Topics

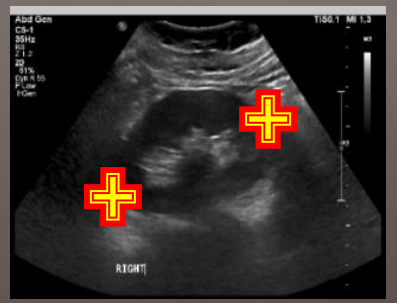
- Review
- 3D ResNet
- 3D U-Net
- Papers!

Multi-task Learning

Input

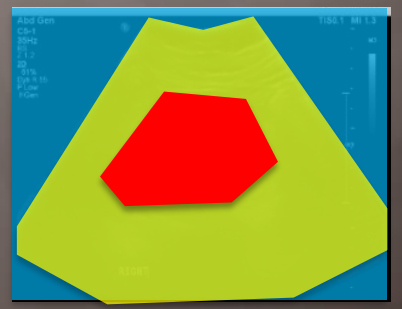


Classification



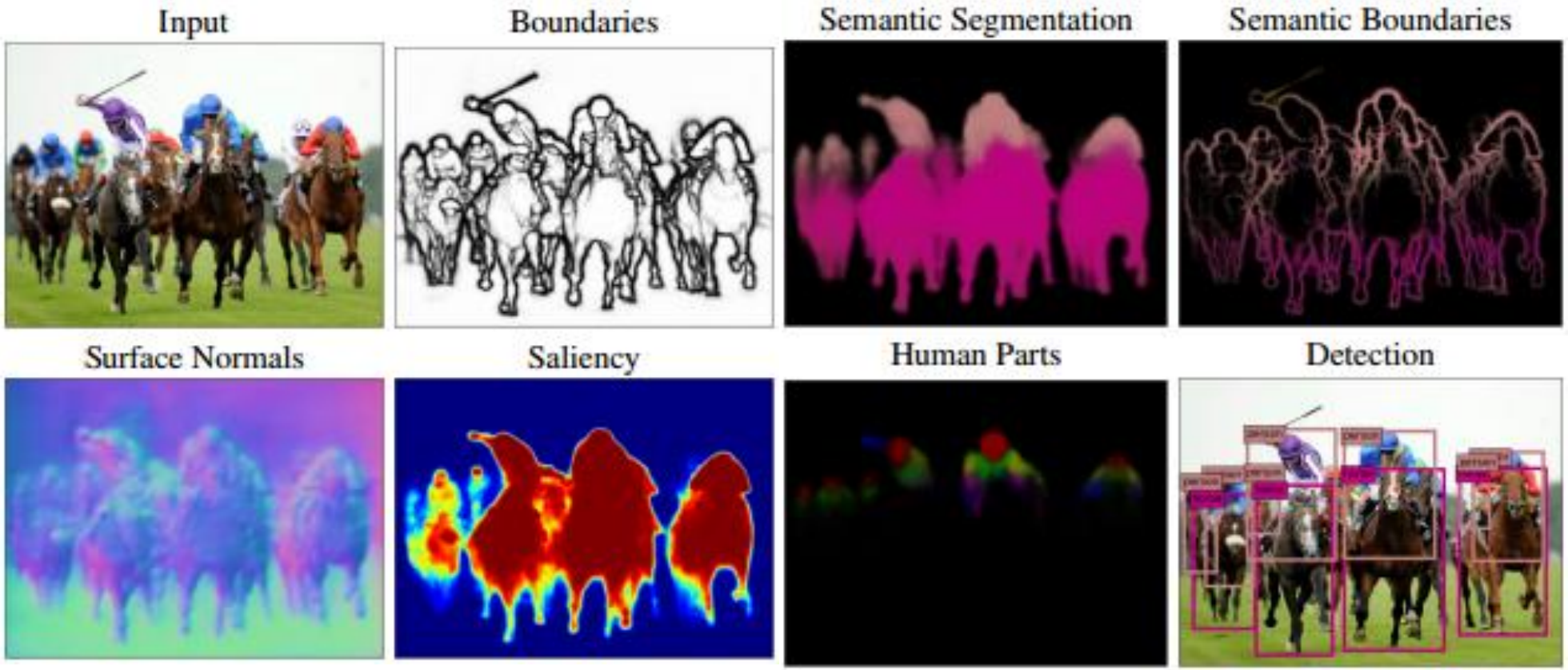
Detection

Synthesis



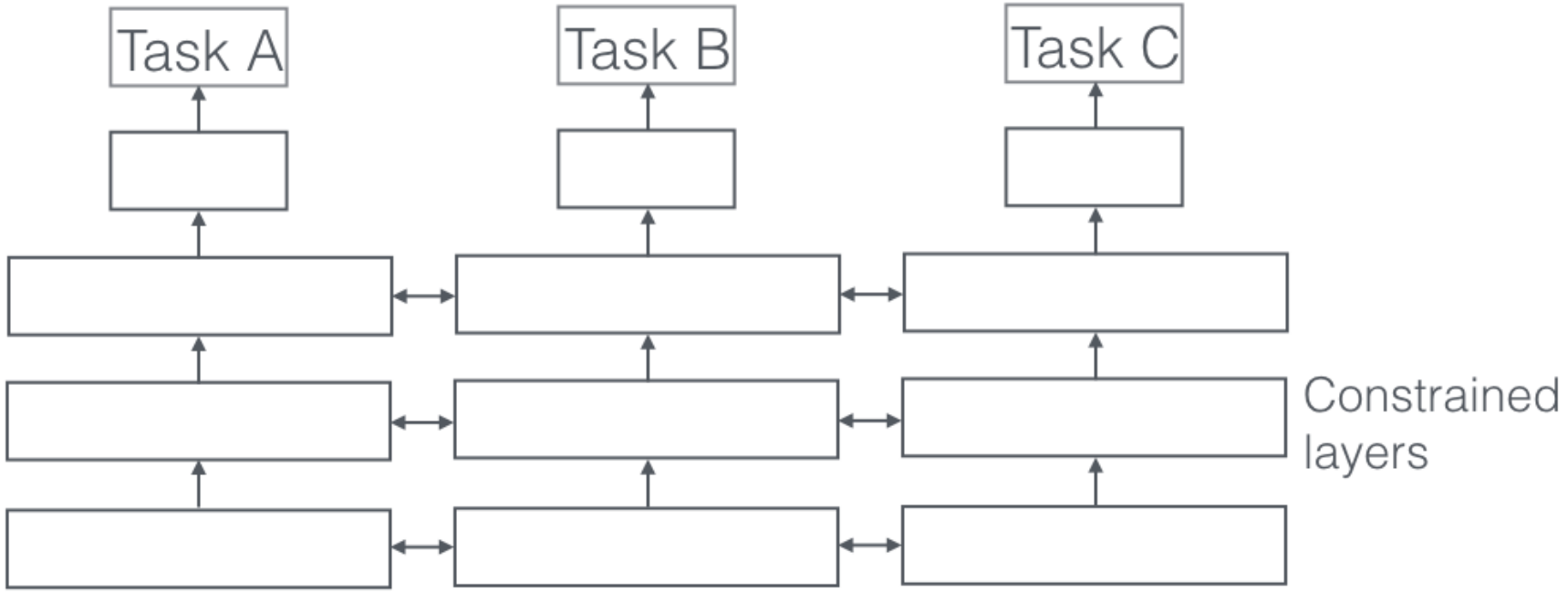
Segmentation

Introduction



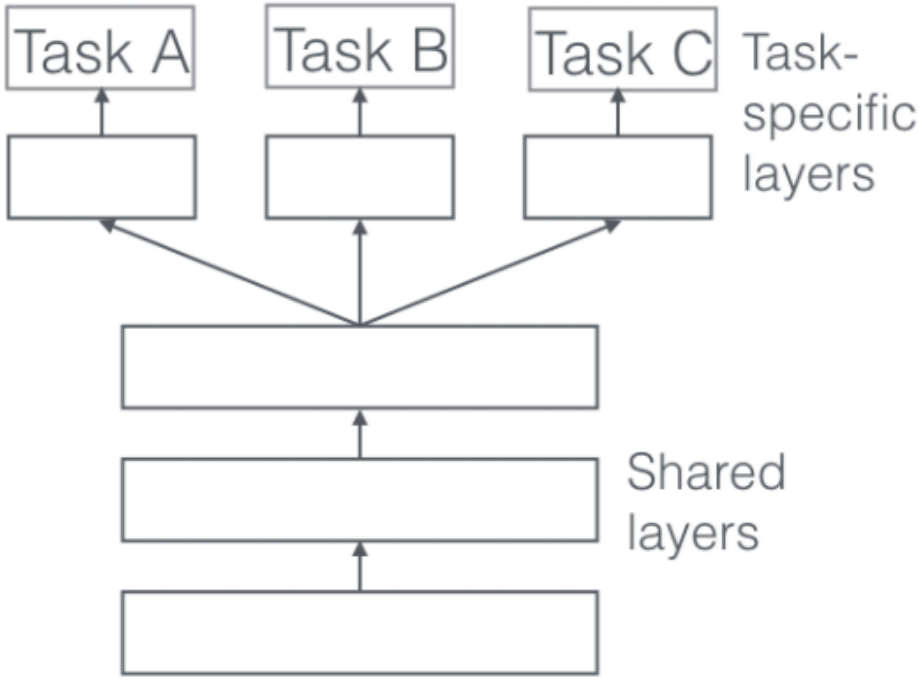
Train a single model → Fine-tune tweak the parameters → Single-Task Learning (STL)

Sharing representations between related tasks → optimizing more than one loss function → Multi-Task Learning (MTL)



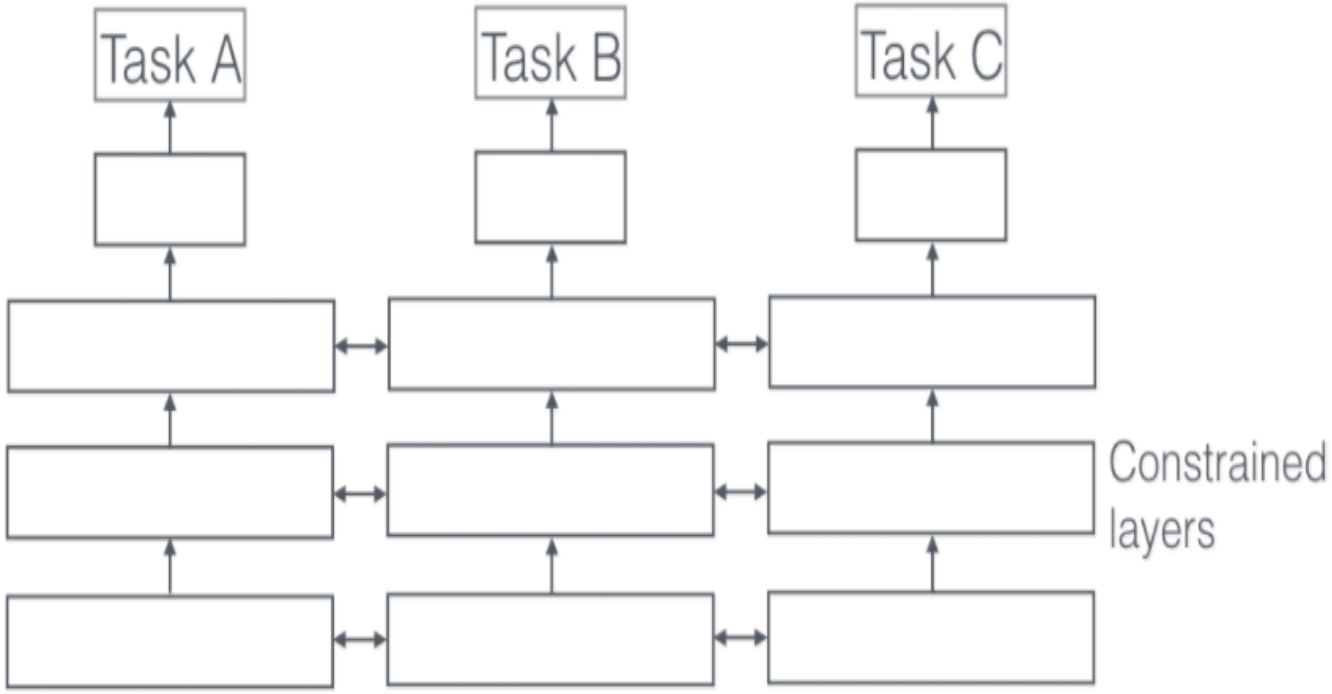
How to share the parameters of layers?

hard

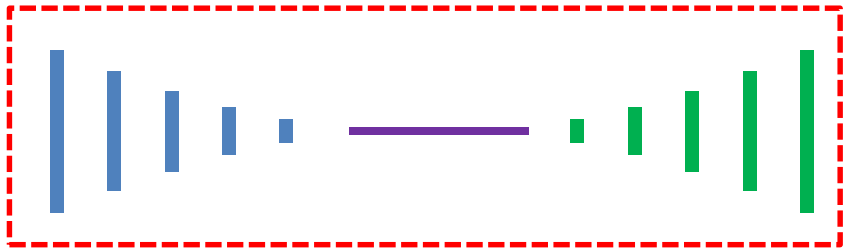


Hard parameter sharing greatly reduces the risk of overfitting

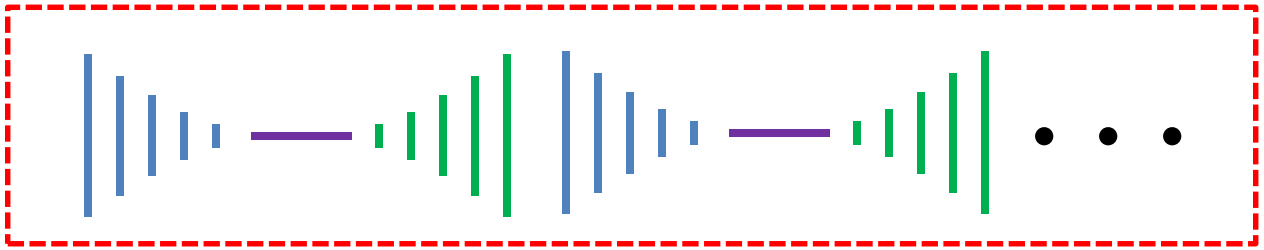
soft



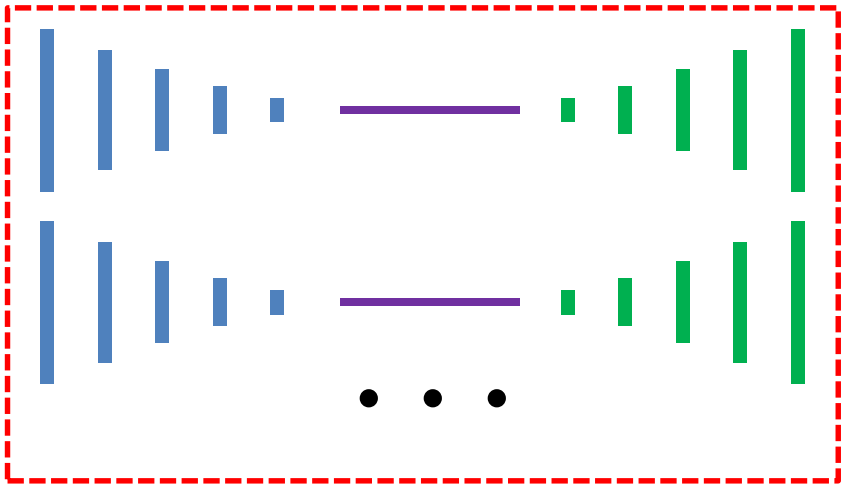
Distance between the parameters of the model is regularized in order to encourage the parameters to be similar



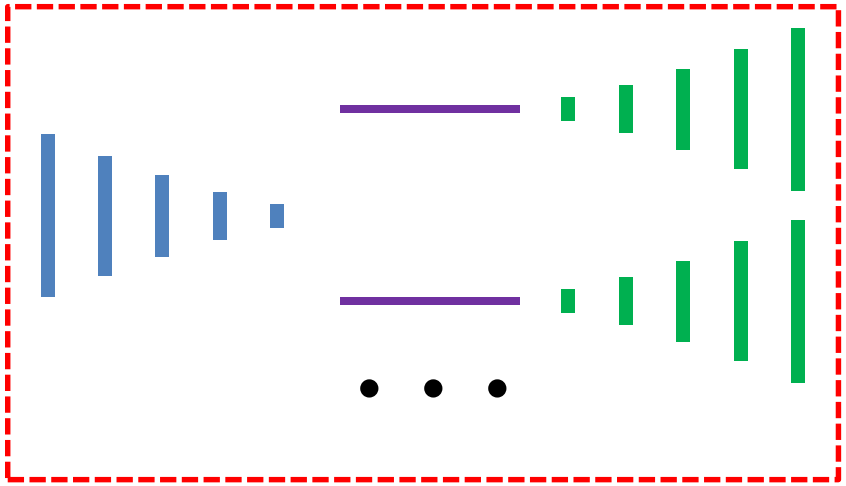
Single



Consequent



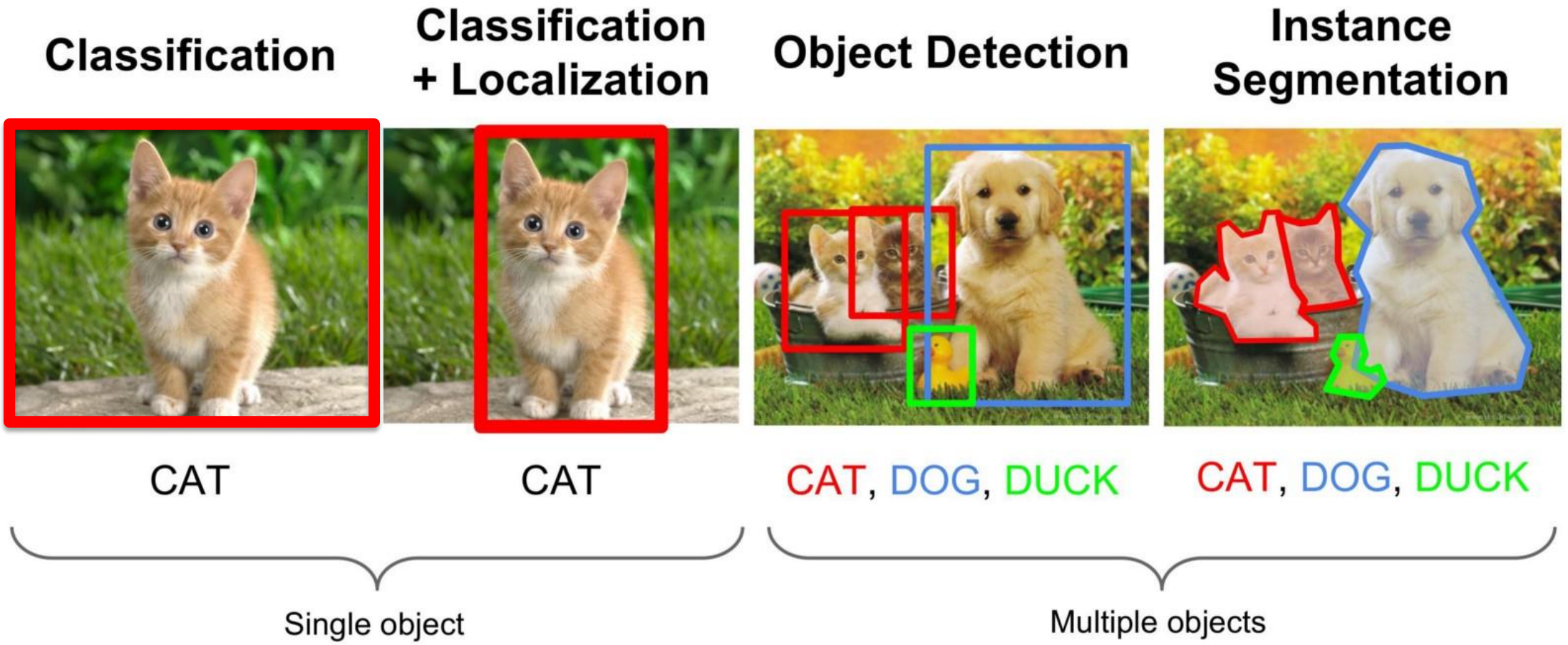
Soft



Hard

- Model Dependency Among Different Tasks
- Characterize the Commonalities and Differences Between Tasks

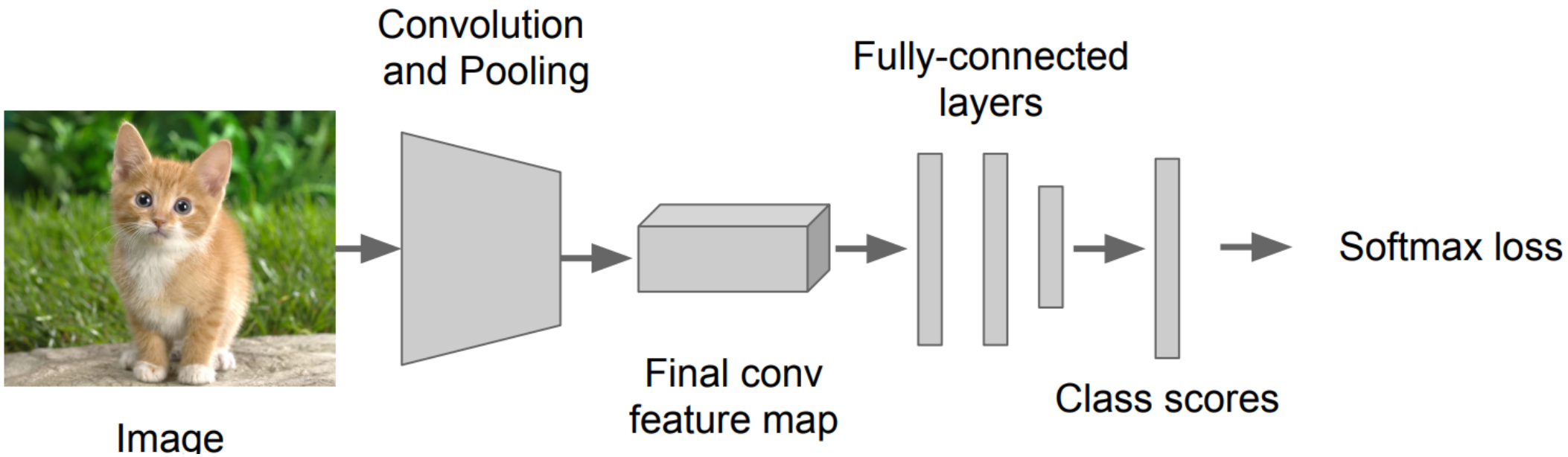
Localization & Detection



https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

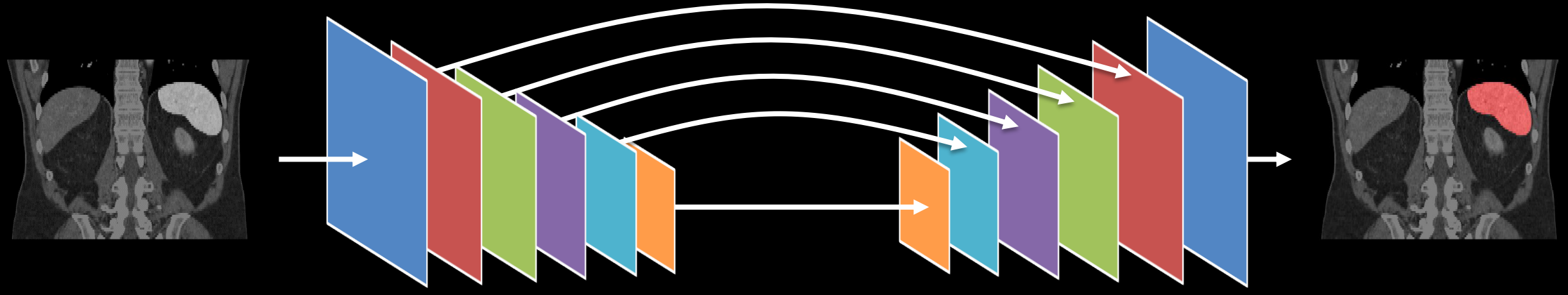
Simple Recipe for Classification

"CONV-POOL-FC"



http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

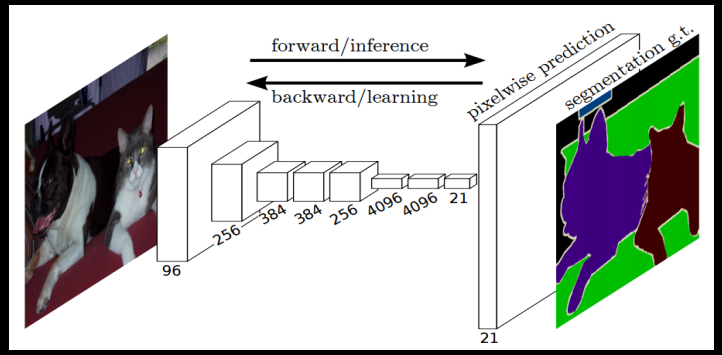
Fully Convolutional Network (FCN)



Encoder

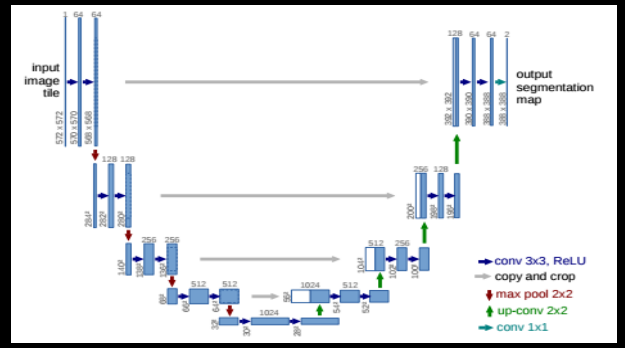
Decoder

FCN:



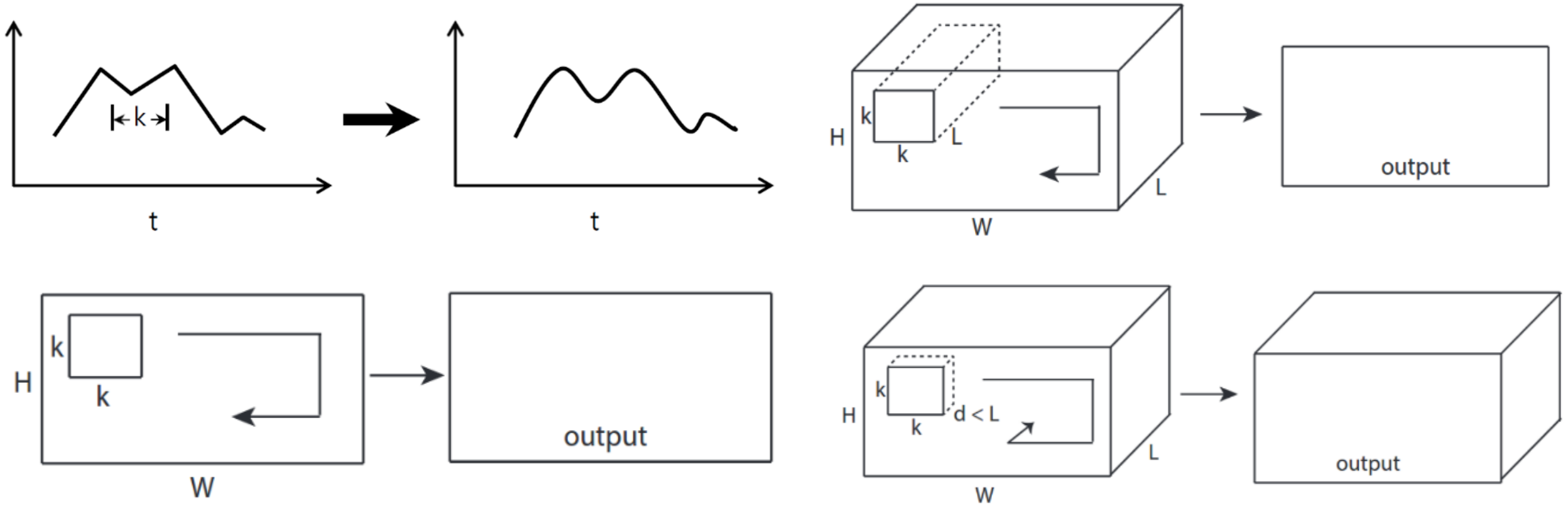
Long, et al. CVPR (2015)

Unet:



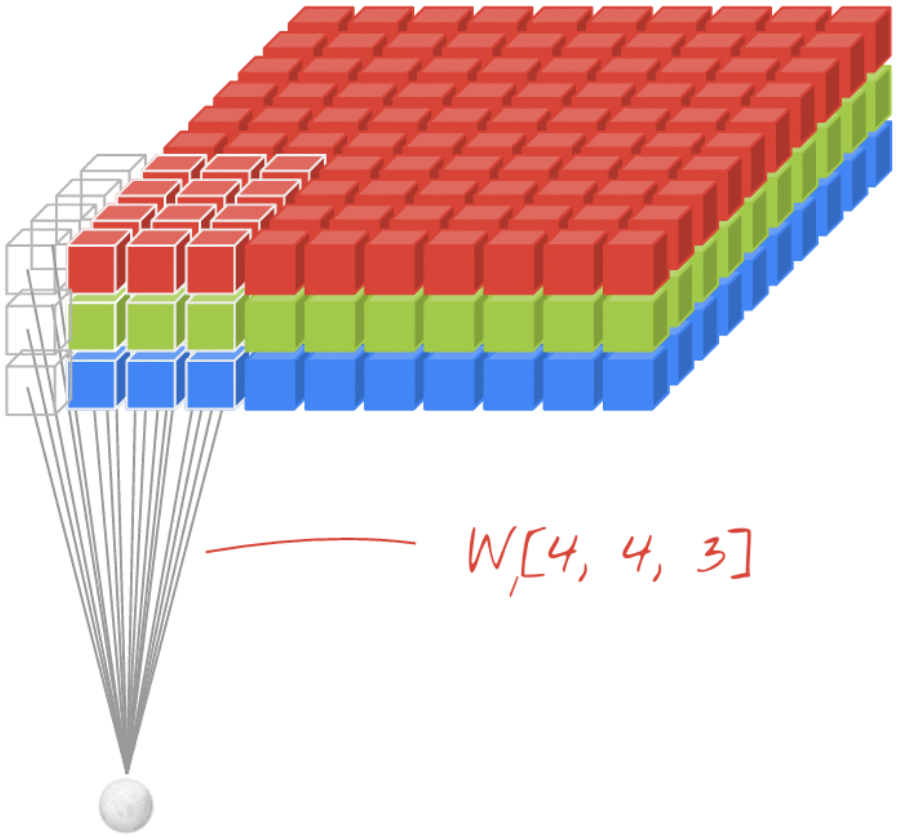
Ronneberger, et al. MICCAI (2015)

1D, 2D, 3D convolutions

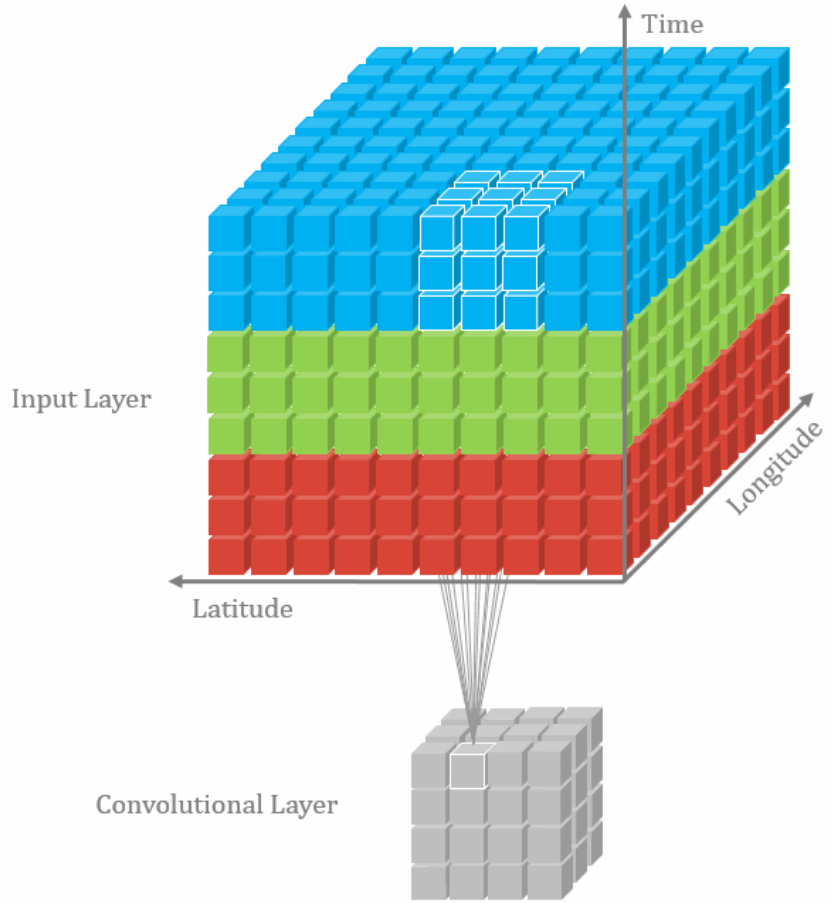


https://stackoverflow.com/questions/42883547/what-do-you-mean-by-1d-2d-and-3d-convolutions-in-cnn

3D convolution



https://stackoverflow.com/questions/42883547/what-do-you-mean-by-1d-2d-and-3d-convolutions-in-cnn

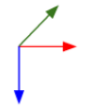


http://resuly.me/2018/05/18/3d-convolutional-networks-for-traffic-forecasting/

Summary

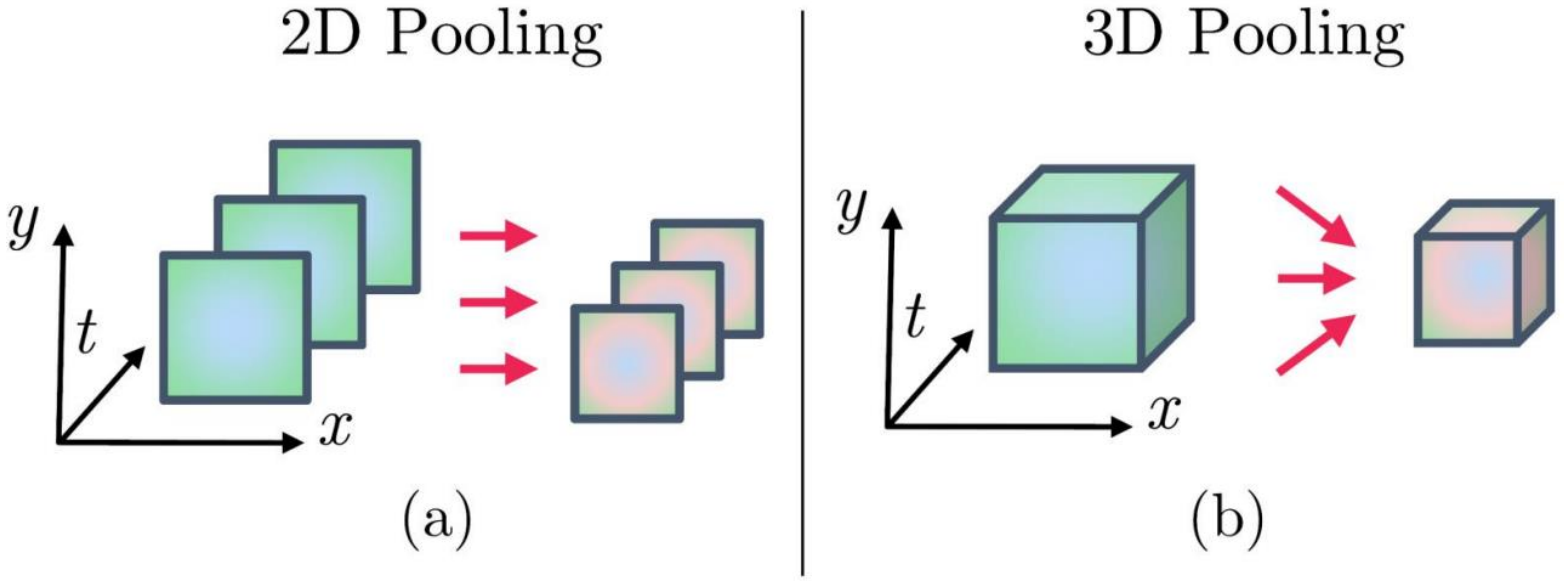
	Input				
tf.nn.conv1d	batch	in_width	in_channels		
tf.nn.conv2d	batch	in_height	in_width	in_channels	
tf.nn.conv3d	batch	in_depth	in_height	in_width	in_channels

Summary

	Conv Direction	Input	Filter	Output
tf.nn.conv1d	1-direction	3-dim	3-dim	2-dim
tf.nn.conv2d	2-direction 	4-dim	4-dim	3-dim
tf.nn.conv3d	3-direction 	5-dim	5-dim	4-dim

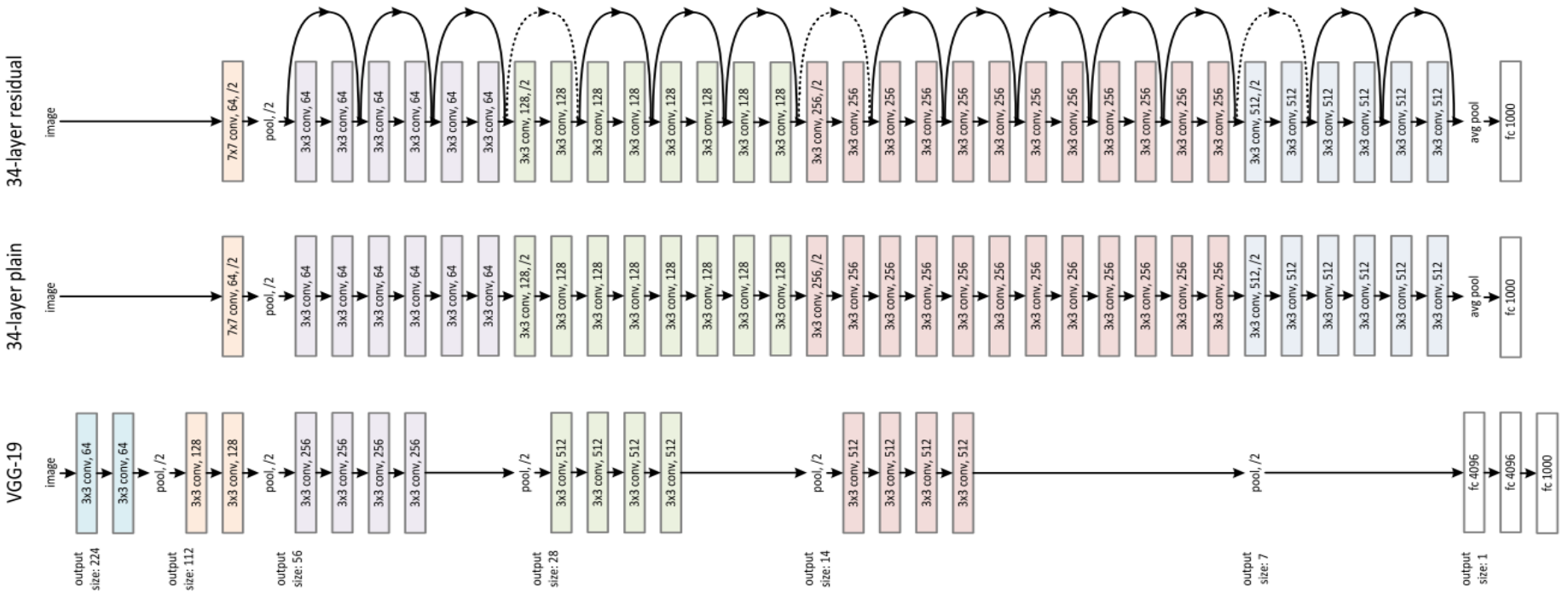
https://stackoverflow.com/questions/42883547/what-do-you-mean-by-1d-2d-and-3d-convolutions-in-cnn

3D pooling



https://feichtenhofer.github.io/pubs/Feichtenhofer_Two_Stream_Fusion_2016_CVPR_Poster.pdf

ResNet(2015)



Ay the ILSVRC 2015, the so-called Residual Neural Network (ResNet) by Kaiming He et al introduced anovel architecture with “skip connections” and features heavy batch normalization. Such skip connections are also known as gated units or gated recurrent units and have a strong similarity to recent successful elements applied in RNNs. Thanks to this technique they were able to train a NN with 152 layers while still having lower complexity than VGGNet. It achieves a top-5 error rate of 3.57% which beats human-level performance on this dataset.

https://medium.com/@pierre_guillou/understand-how-works-resnet-without-talking-about-residual-64698f157e0c

https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

ResNet

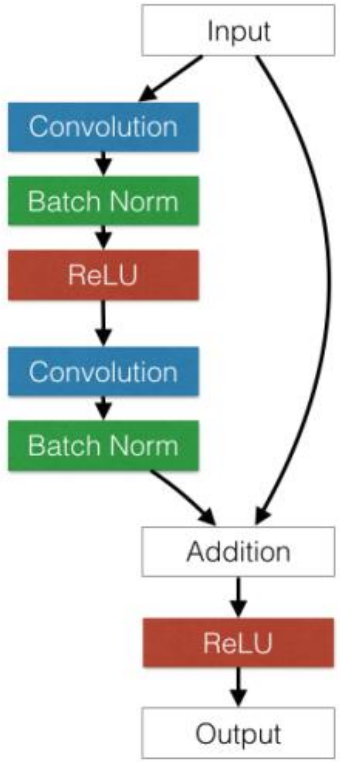


Figure 1. A RestNet basic block

https://dgschwend.github.io/netscope/#/preset/resnet-50

https://www.kdnuggets.com/2016/09/9-key-deep-learning-papers-explained.html/2

http://dgschwend.github.io/netscope/quickstart.html

2D ResNet

```python
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(Bottleneck, self).__init__()
        self.bn1 = nn.BatchNorm2d(inplanes)
        self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride,
                               padding=1, bias=False)
        self.bn3 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, planes * 4, kernel_size=1, bias=False)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride
```

3D ResNet

```python
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv3d(inplanes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm3d(planes)
        self.conv2 = nn.Conv3d(planes, planes, kernel_size=3, stride=stride,
                               padding=1, bias=False)
        self.bn2 = nn.BatchNorm3d(planes)
        self.conv3 = nn.Conv3d(planes, planes * 4, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm3d(planes * 4)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride
```

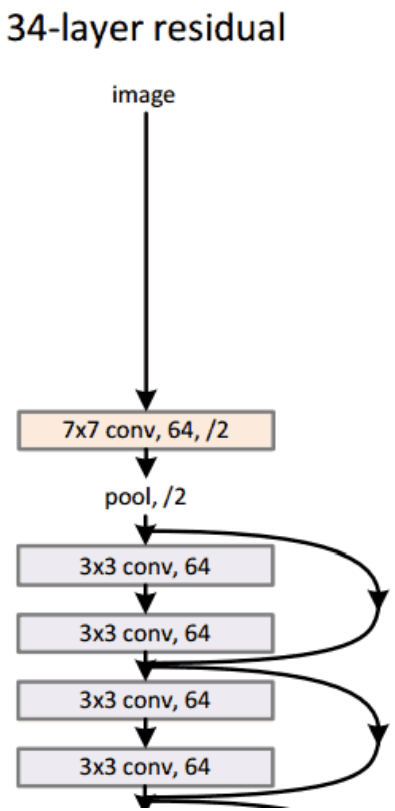
2D ResNet

```
def forward(self, x):
    residual = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)
    out = self.relu(out)

    out = self.conv3(out)
    out = self.bn3(out)

    if self.downsample is not None:
        residual = self.downsample(x)

    out += residual
    out = self.relu(out)

    return out
```

3D ResNet

```
def forward(self, x):
    residual = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)
    out = self.relu(out)

    out = self.conv3(out)
    out = self.bn3(out)

    if self.downsample is not None:
        residual = self.downsample(x)

    out += residual
    out = self.relu(out)

    return out
```

ResNet



https://dgschwend.github.io/netscope/#/preset/resnet-50

https://www.kdnuggets.com/2016/09/9-key-deep-learning-papers-explained.html/2

http://dgschwend.github.io/netscope/quickstart.html

2D ResNet

```python
class ResNet(nn.Module):

    def __init__(self, block, layers, num_classes=1000):
        self.inplanes = 16
        super(PreResNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1,
                               bias=False)
        self.layer1 = self._make_layer(block, 16, layers[0])
        self.layer2 = self._make_layer(block, 32, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 64, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 64, layers[2], stride=2)
        self.bn1 = nn.BatchNorm2d(64*block.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.avgpool = nn.AvgPool2d(8)
        self.fc = nn.Linear(64*block.expansion, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
```

3D ResNet

```python
class ResNet(nn.Module):

    def __init__(self, block, layers, sample_size, sample_duration, shortcut_type='B', num_classes=400, last_fc=True):
        self.last_fc = last_fc

        self.inplanes = 64
        super(ResNet, self).__init__()
        self.conv1 = nn.Conv3d(3, 64, kernel_size=7, stride=(1, 2, 2),
                               padding=(3, 3, 3), bias=False)
        self.bn1 = nn.BatchNorm3d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool3d(kernel_size=(3, 3, 3), stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, layers[0], shortcut_type)
        self.layer2 = self._make_layer(block, 128, layers[1], shortcut_type, stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], shortcut_type, stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], shortcut_type, stride=2)
        last_duration = math.ceil(sample_duration / 16)
        last_size = math.ceil(sample_size / 32)
        self.avgpool = nn.AvgPool3d((last_duration, last_size, last_size), stride=1)
        self.fc = nn.Linear(512 * block.expansion, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv3d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
            elif isinstance(m, nn.BatchNorm3d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
```

2D ResNet

```
def _make_layer(self, block, planes, blocks, stride=1):
    downsample = None
    if stride != 1 or self.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            nn.Conv2d(self.inplanes, planes * block.expansion,
                      kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(planes * block.expansion),
        )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample))
    self.inplanes = planes * block.expansion
    for i in range(1, blocks):
        layers.append(block(self.inplanes, planes))

    return nn.Sequential(*layers)
```

3D ResNet

```
def _make_layer(self, block, planes, blocks, shortcut_type, stride=1):
    downsample = None
    if stride != 1 or self.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            nn.Conv3d(self.inplanes, planes * block.expansion,
                      kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm3d(planes * block.expansion)
        )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample))
    self.inplanes = planes * block.expansion
    for i in range(1, blocks):
        layers.append(block(self.inplanes, planes))

    return nn.Sequential(*layers)
```

2D ResNet

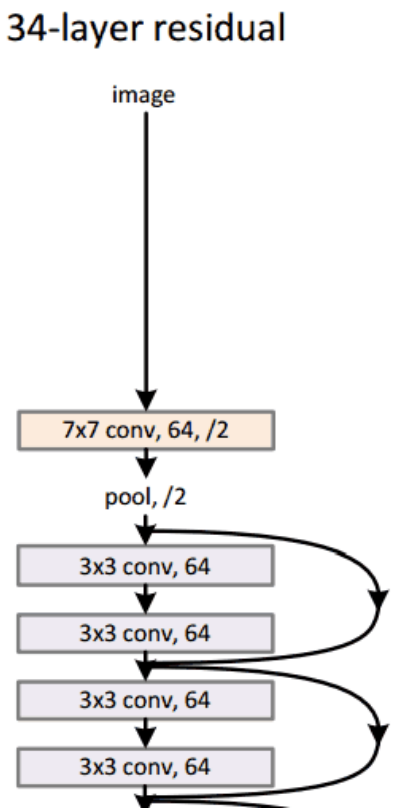
```
def forward(self, x):
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)

    x = x.view(x.size(0), -1)
    if self.last_fc:
        x = self.fc(x)

    return x
```

3D ResNet

```
def forward(self, x):
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)

    x = x.view(x.size(0), -1)
    if self.last_fc:
        x = self.fc(x)

    return x
```

```python
def resnet10(**kwargs):
    """Constructs a ResNet-18 model.
    """
    model = ResNet(BasicBlock, [1, 1, 1, 1], **kwargs)
    return model

def resnet18(**kwargs):
    """Constructs a ResNet-18 model.
    """
    model = ResNet(BasicBlock, [2, 2, 2, 2], **kwargs)
    return model

def resnet34(**kwargs):
    """Constructs a ResNet-34 model.
    """
    model = ResNet(BasicBlock, [3, 4, 6, 3], **kwargs)
    return model

def resnet50(**kwargs):
    """Constructs a ResNet-50 model.
    """
    model = ResNet(Bottleneck, [3, 4, 6, 3], **kwargs)
    return model

def resnet101(**kwargs):
    """Constructs a ResNet-101 model.
    """
    model = ResNet(Bottleneck, [3, 4, 23, 3], **kwargs)
    return model

def resnet152(**kwargs):
    """Constructs a ResNet-101 model.
    """
    model = ResNet(Bottleneck, [3, 8, 36, 3], **kwargs)
    return model
```

ResNet



Netscope CNN Analyzer

A web-based tool for visualizing and analyzing convolutional neural network architectures (or technically, any directed acyclic graph). Currently supports Caffe's prototxt format.

Basis by ethereon. Extended for CNN Analysis by dgschwend.

Gist Support

If your `.prototxt` file is part of a GitHub Gist, you can visualize it by visiting this URL:

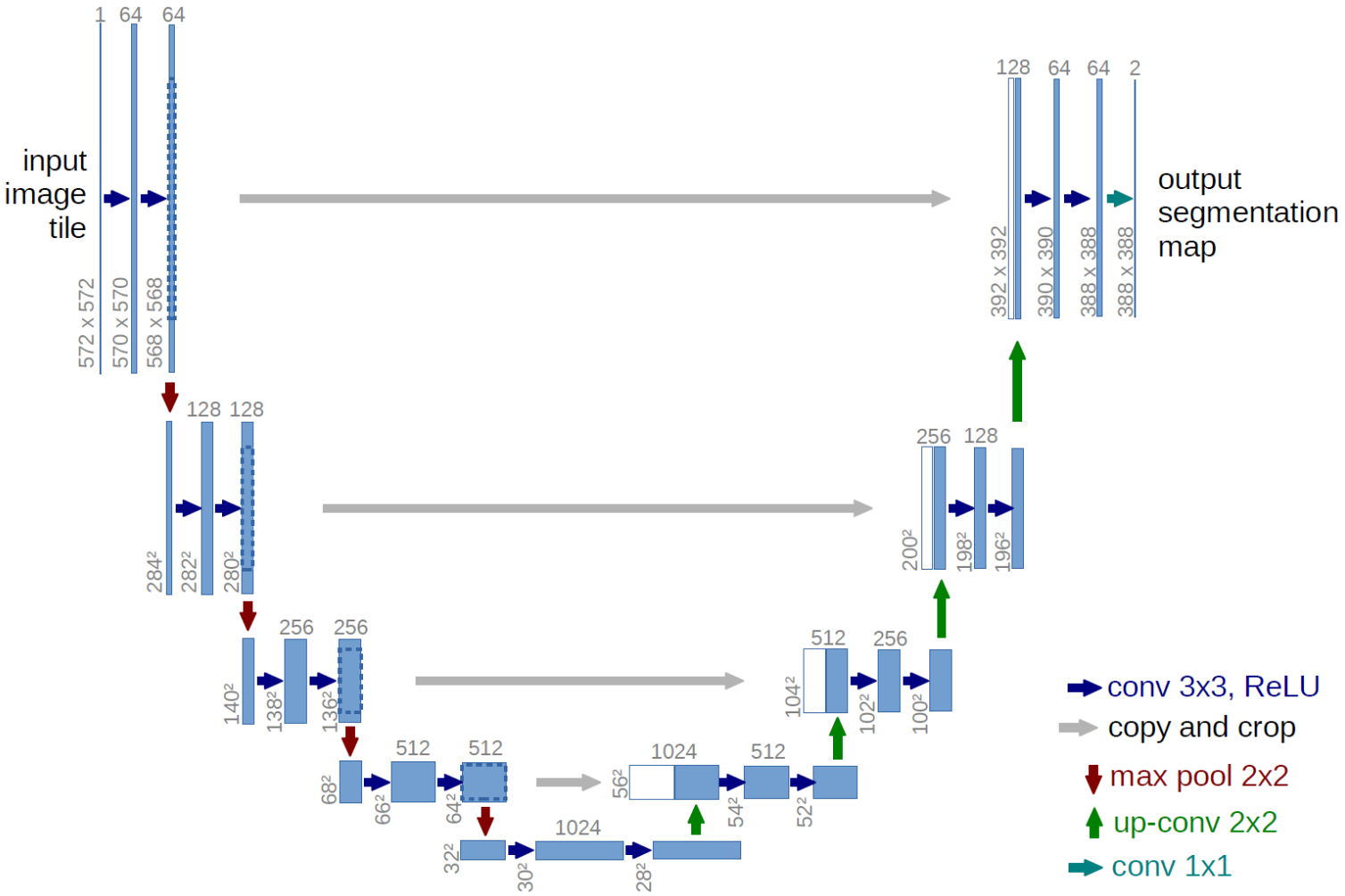
```
http://dgschwend.github.io/netscope/#/gist/your-gist-id
```

https://dgschwend.github.io/netscope/#/preset/resnet-50

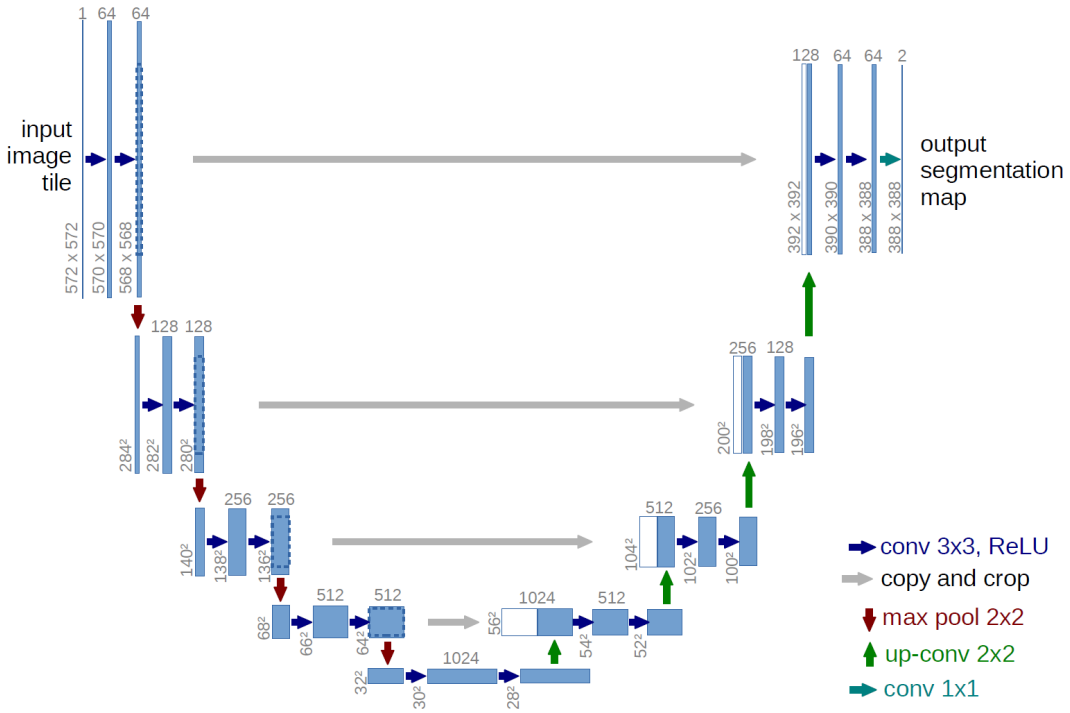
https://www.kdnuggets.com/2016/09/9-key-deep-learning-papers-explained.html/2

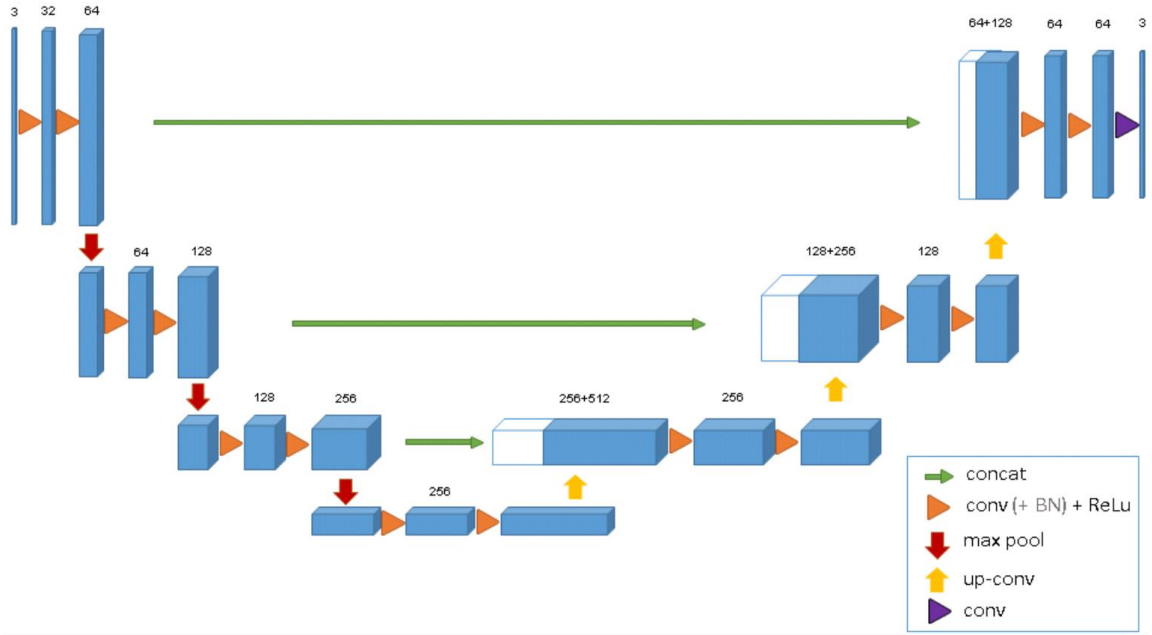
http://dgschwend.github.io/netscope/quickstart.html

U-Net


input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

3D U-Net





https://arxiv.org/abs/1606.06650

All Code for U-Net

Encoder

```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

Decoder

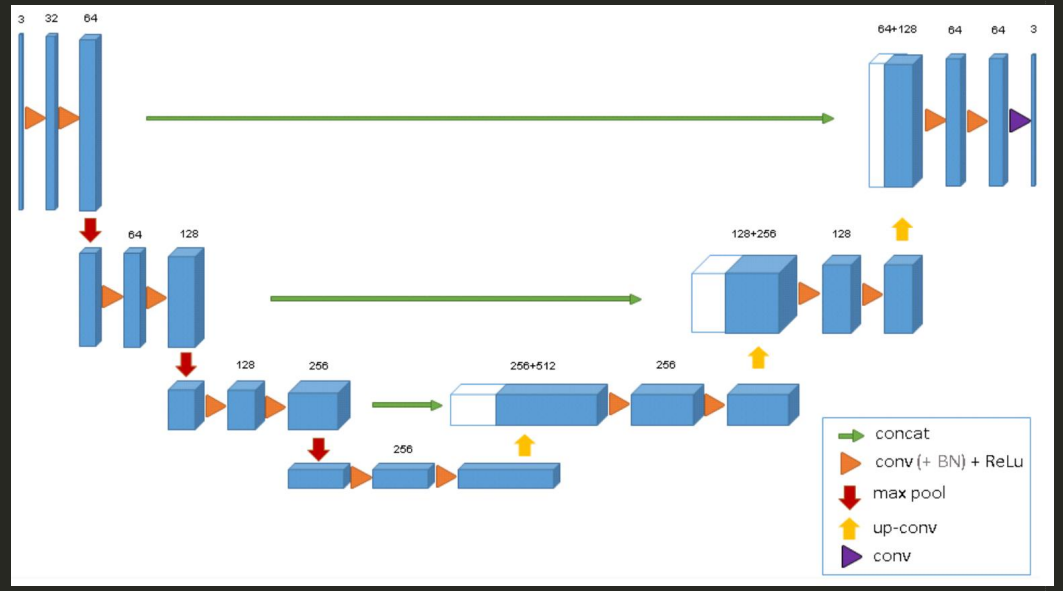
```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))

layer7 = F.relu(self.conv7_up(layer6))
layer7 = torch.cat((layer3, layer7), 1)
layer7 = F.relu(self.conv7_input(layer7))
layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8_up(layer7))
layer8 = torch.cat((layer2, layer8), 1)
layer8 = F.relu(self.conv8_input(layer8))
layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9_up(layer8))
layer9 = torch.cat((layer1, layer9), 1)
layer9 = F.relu(self.conv9_input(layer9))
layer9 = F.relu(self.conv9(layer9))
layer9 = self.final(self.conv9_output(layer9))

return layer9
```

Parameters

```
self.conv1_input =      nn.Conv2d(1, 64, 3, padding=1)
self.conv1 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv2_input =      nn.Conv2d(64, 128, 3, padding=1)
self.conv2 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv3_input =      nn.Conv2d(128, 256, 3, padding=1)
self.conv3 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv4_input =      nn.Conv2d(256, 512, 3, padding=1)
self.conv4 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv5_input =      nn.Conv2d(512, 1024, 3, padding=1)
self.conv5 =            nn.Conv2d(1024, 1024, 3, padding=1)

self.conv6_up =         nn.ConvTranspose2d(1024, 512, 2, 2)
self.conv6_input =      nn.Conv2d(1024, 512, 3, padding=1)
self.conv6 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv7_up =         nn.ConvTranspose2d(512, 256, 2, 2)
self.conv7_input =      nn.Conv2d(512, 256, 3, padding=1)
self.conv7 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv8_up =         nn.ConvTranspose2d(256, 128, 2, 2)
self.conv8_input =      nn.Conv2d(256, 128, 3, padding=1)
self.conv8 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv9_up =         nn.ConvTranspose2d(128, 64, 2, 2)
self.conv9_input =      nn.Conv2d(128, 64, 3, padding=1)
self.conv9 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv9_output =     nn.Conv2d(64, 2, 1)
```

https://github.com/jonnedtc/U-Net-PyTorch/blob/master/networks.py

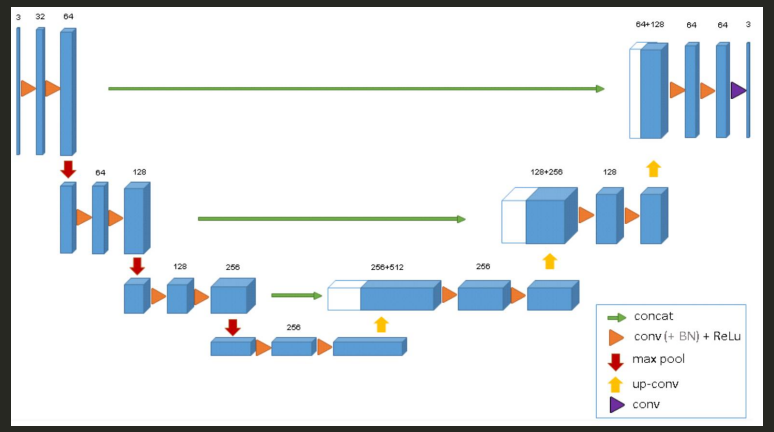
All Code for 3D U-Net




```
class UNet3D(nn.Module):
    def __init__(self, in_channel, n_classes):
        self.in_channel = in_channel
        self.n_classes = n_classes
        super(UNet3D, self).__init__()
        self.ec0 = self.encoder(self.in_channel, 32, bias=False, batchnorm=False)
        self.ec1 = self.encoder(32, 64, bias=False, batchnorm=False)
        self.ec2 = self.encoder(64, 64, bias=False, batchnorm=False)
        self.ec3 = self.encoder(64, 128, bias=False, batchnorm=False)
        self.ec4 = self.encoder(128, 128, bias=False, batchnorm=False)
        self.ec5 = self.encoder(128, 256, bias=False, batchnorm=False)
        self.ec6 = self.encoder(256, 256, bias=False, batchnorm=False)
        self.ec7 = self.encoder(256, 512, bias=False, batchnorm=False)

        self.pool0 = nn.MaxPool3d(2)
        self.pool1 = nn.MaxPool3d(2)
        self.pool2 = nn.MaxPool3d(2)

        self.dc9 = self.decoder(512, 512, kernel_size=2, stride=2, bias=False)
        self.dc8 = self.decoder(256 + 512, 256, kernel_size=3, stride=1, padding=1, bias=False)
        self.dc7 = self.decoder(256, 256, kernel_size=3, stride=1, padding=1, bias=False)
        self.dc6 = self.decoder(256, 256, kernel_size=2, stride=2, bias=False)
        self.dc5 = self.decoder(128 + 256, 128, kernel_size=3, stride=1, padding=1, bias=False)
        self.dc4 = self.decoder(128, 128, kernel_size=3, stride=1, padding=1, bias=False)
        self.dc3 = self.decoder(128, 128, kernel_size=2, stride=2, bias=False)
        self.dc2 = self.decoder(64 + 128, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.dc1 = self.decoder(64, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.dc0 = self.decoder(64, n_classes, kernel_size=1, stride=1, bias=False)
```

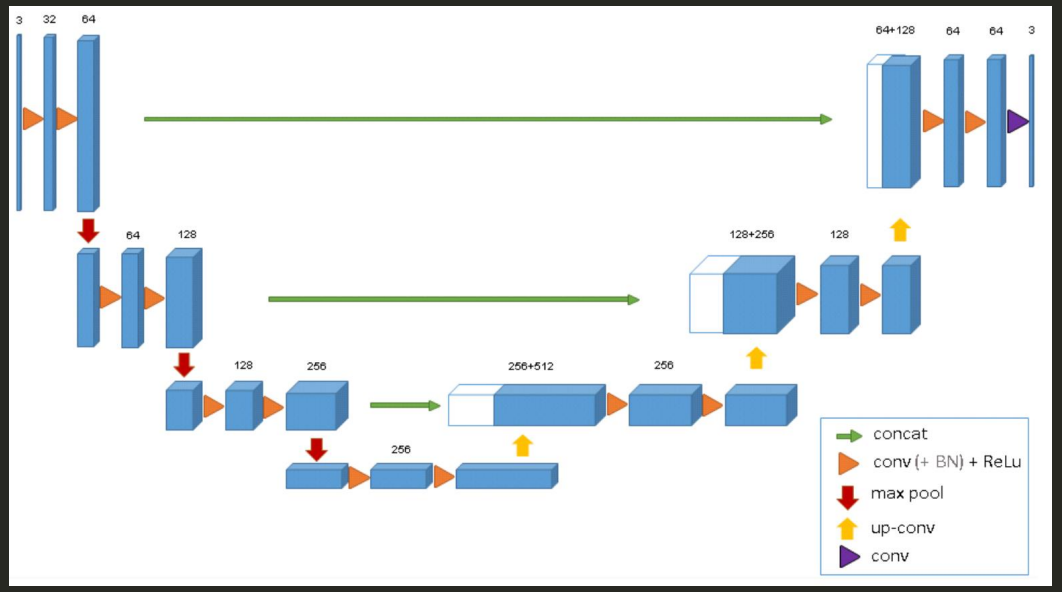
https://github.com/jonnedtc/U-Net-PyTorch/blob/master/networks.py

All Code for 3D U-Net



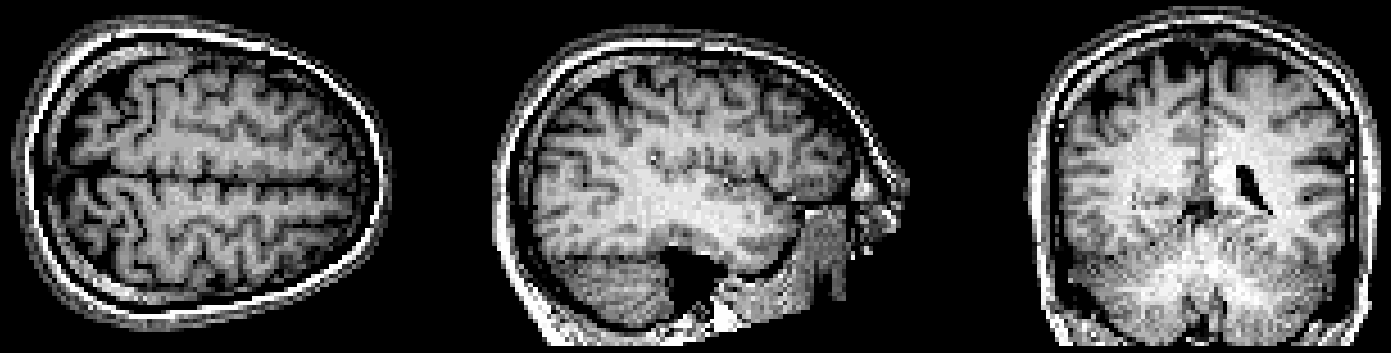

```
def encoder(self, in_channels, out_channels, kernel_size=3, stride=1, padding=0,
            bias=True, batchnorm=False):
    if batchnorm:
        layer = nn.Sequential(
            nn.Conv3d(in_channels, out_channels, kernel_size, stride=stride, padding=padding, bias=bias),
            nn.BatchNorm2d(out_channels),
            nn.ReLU())
    else:
        layer = nn.Sequential(
            nn.Conv3d(in_channels, out_channels, kernel_size, stride=stride, padding=padding, bias=bias),
            nn.ReLU())
    return layer


def decoder(self, in_channels, out_channels, kernel_size, stride=1, padding=0,
            output_padding=0, bias=True):
    layer = nn.Sequential(
        nn.ConvTranspose3d(in_channels, out_channels, kernel_size, stride=stride,
                           padding=padding, output_padding=output_padding, bias=bias),
        nn.ReLU())
    return layer
```

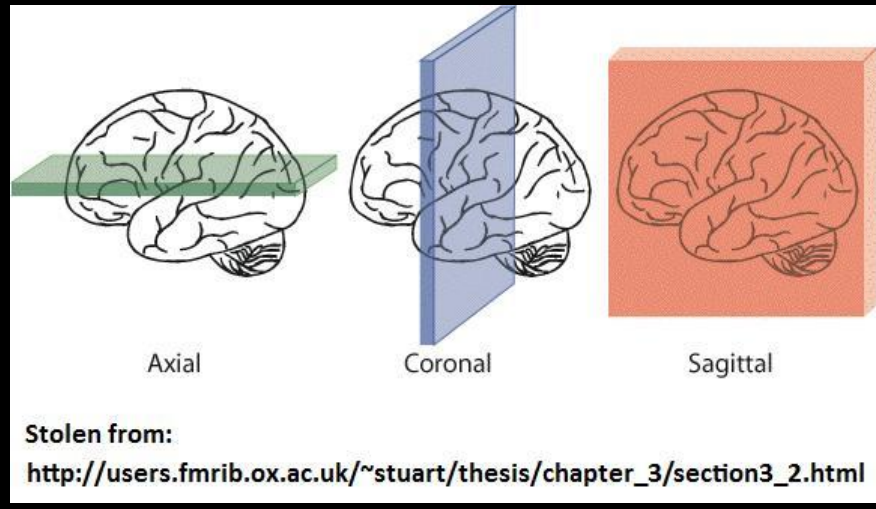
All Code for 3D U-Net




```
def forward(self, x):
    e0 = self.ec0(x)
    syn0 = self.ec1(e0)
    e1 = self.pool0(syn0)
    e2 = self.ec2(e1)
    syn1 = self.ec3(e2)
    del e0, e1, e2

    e3 = self.pool1(syn1)
    e4 = self.ec4(e3)
    syn2 = self.ec5(e4)
    del e3, e4

    e5 = self.pool2(syn2)
    e6 = self.ec6(e5)
    e7 = self.ec7(e6)
    del e5, e6

    d9 = torch.cat((self.dc9(e7), syn2))
    del e7, syn2

    d8 = self.dc8(d9)
    d7 = self.dc7(d8)
    del d9, d8
    d6 = torch.cat((self.dc6(d7), syn1))
    del d7, syn1

    d5 = self.dc5(d6)
    d4 = self.dc4(d5)
    del d6, d5

    d3 = torch.cat((self.dc3(d4), syn0))
    del d4, syn0

    d2 = self.dc2(d3)
    d1 = self.dc1(d2)
    del d3, d2

    d0 = self.dc0(d1)
    return d0
```

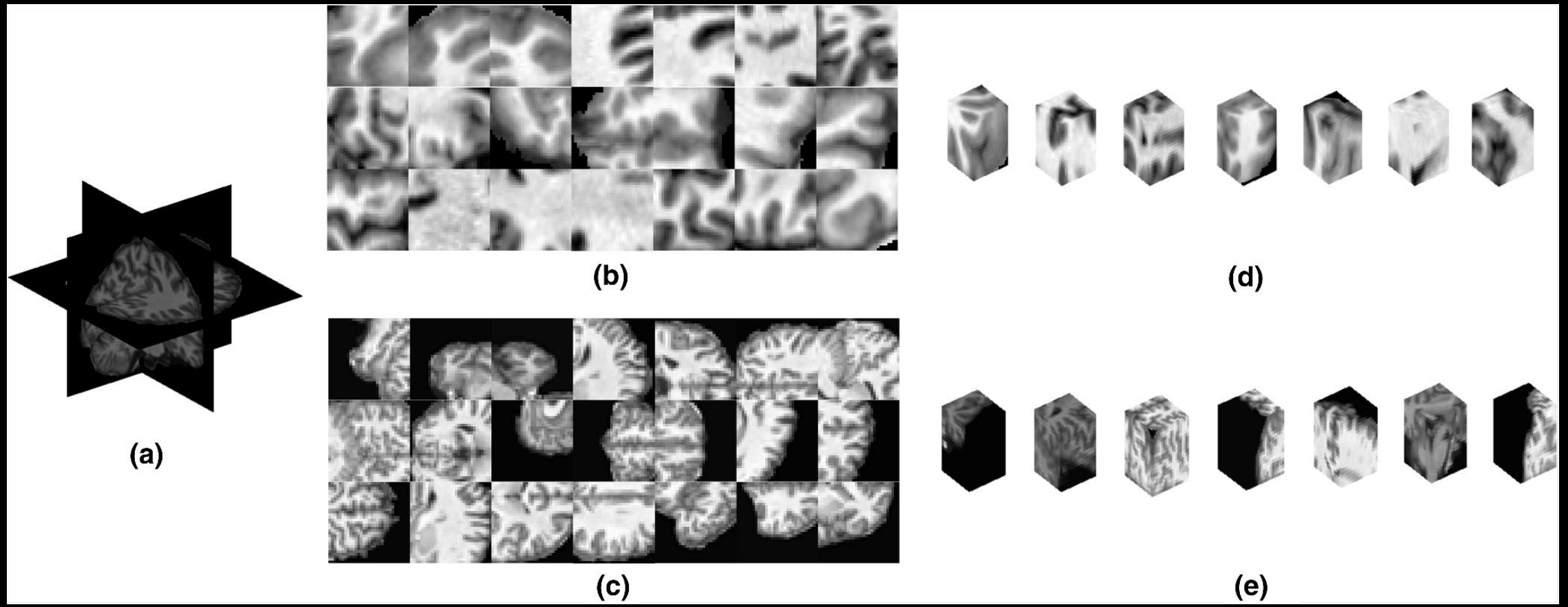
Axial, Coronal, Sagittal





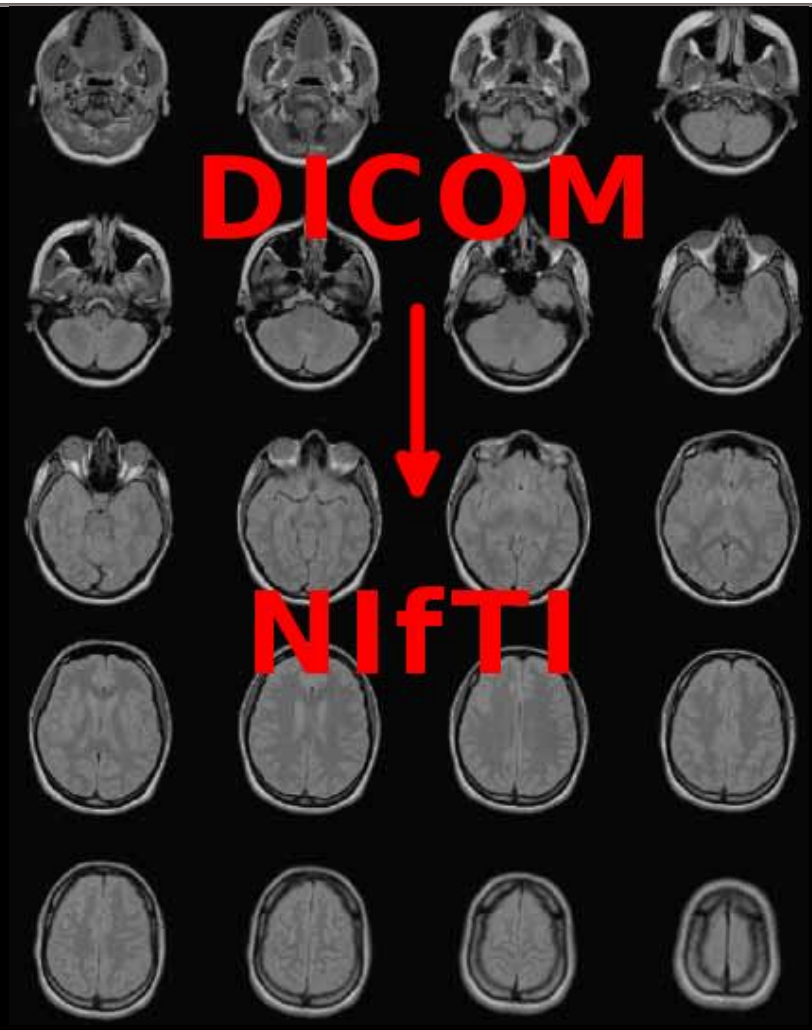
https://www.medgif.com/brain-mri-scan/ https://sites.google.com/a/wisc.edu/neuroradiology/image-acquisition/the-basics

Segmentation



https://www.ncbi.nlm.nih.gov/pubmed/28439524

DICOM and NIFTI



https://www.mathworks.com/matlabcentral/fileexchange/41874-dicom-to-nifti-converter

NIFTI is a 3D volume

https://www.youtube.com/watch?v=f7K53J_w_RU

Read 3D volume

https://github.com/MASILab/SLANTbrainSeg/blob/master/python/torchsrc/imgloaders/imgloader_CT_3D.py

```
def __getitem__(self, index):
        num_labels = self.num_labels
        sub_name = self.img_subs[index]
        x = np.zeros((1, output_z, output_x, output_y))
        img_file = self.img_files[index]
        img_3d = nib.load(img_file)
        img = img_3d.get_data()
        img = (img - img.min())/(img.max()-img.min())
        img = img*255.0
        img = np.transpose(img,(2, 0, 1))
        x[0,:,:,:] = img[0:output_z,0:output_x,0:output_y]
        x = x.astype('float32')

        y = np.zeros((num_labels, output_z, output_x, output_y))
        seg_file = self.seg_files[index]
        seg_3d = nib.load(seg_file)
        seg = seg_3d.get_data()
        seg = np.transpose(seg,(2, 0, 1))
        y[0,:,:,:] = np.ones([output_z,output_x,output_y])
        for i in range(1,num_labels):
            seg_one = seg == labels[i]
            y[i,:,:,:] = seg_one[0:output_z,0:output_x,0:output_y]
            y[0,:,:,:] = y[0,:,:,:] - y[i,:,:,:]
        y = y.astype('float32')
```

3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation

Özgün Çiçek[1,2], Ahmed Abdulkadir[1,4], Soeren S. Lienkamp[2,3], Thomas Brox[1,2], and Olaf Ronneberger[1,2,5]

[1] Computer Science Department, University of Freiburg, Germany
[2] BIOSS Centre for Biological Signalling Studies, Freiburg, Germany
[3] University Hospital Freiburg, Renal Division, Faculty of Medicine, University of Freiburg, Germany
[4] Department of Psychiatry and Psychotherapy, University Medical Center Freiburg, Germany
[5] Google DeepMind, London, UK
cicek@cs.uni-freiburg.de

Abstract. This paper introduces a network for volumetric segmentation that learns from sparsely annotated volumetric images. We outline two attractive use cases of this method: (1) In a semi-automated setup, the user annotates some slices in the volume to be segmented. The network learns from these sparse annotations and provides a dense 3D segmentation. (2) In a fully-automated setup, we assume that a representative, sparsely annotated training set exists. Trained on this data set, the network densely segments new volumetric images. The proposed network extends the previous u-net architecture from Ronneberger et al. by replacing all 2D operations with their 3D counterparts. The implementation performs on-the-fly elastic deformations for efficient data augmentation during training. It is trained end-to-end from scratch, i.e., no pre-trained network is required. We test the performance of the proposed method on a complex, highly variable 3D structure, the *Xenopus* kidney, and achieve good results for both use cases.

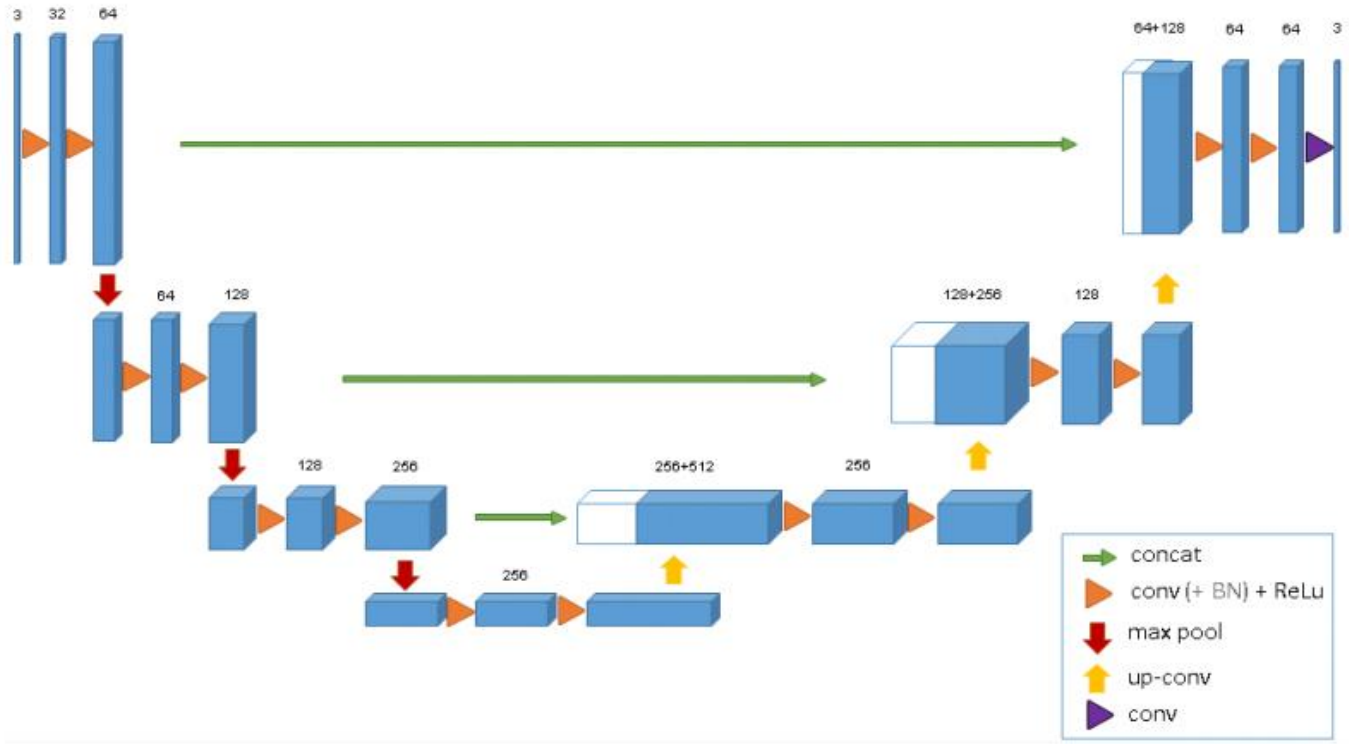


Fig. 2: The 3D u-net architecture. Blue boxes represent feature maps. The number of channels is denoted above each feature map.

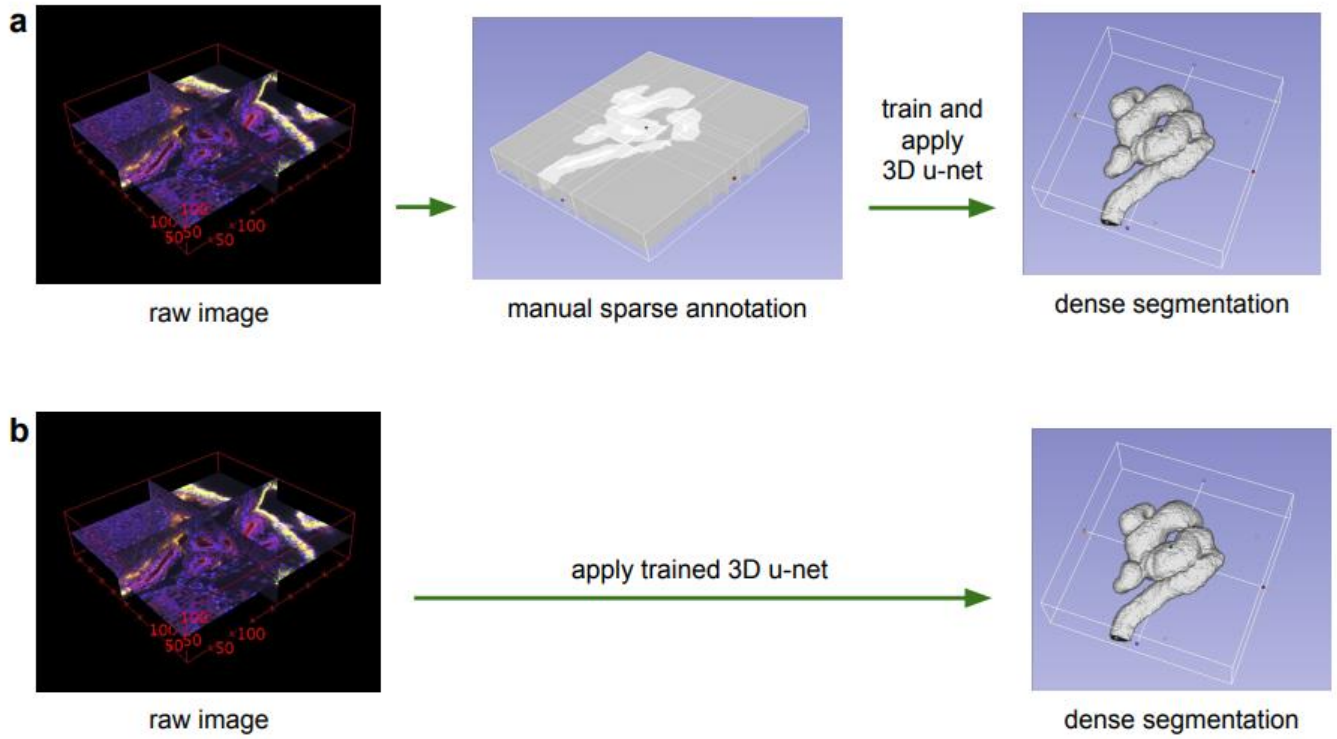


Fig. 1: Application scenarios for volumetric segmentation with the 3D u-net. (a) Semi-automated segmentation: the user annotates some slices of each volume to be segmented. The network predicts the dense segmentation. (b) Fully-automated segmentation: the network is trained with annotated slices from a representative training set and can be run on non-annotated volumes.

$132 \times 132 \times 116$

Table 1: Cross validation results for semi-automated segmentation (IoU)

test slices	3D w/o BN	3D with BN	2D with BN
subset 1	0.822	0.855	0.785
subset 2	0.857	0.871	0.820
subset 3	0.846	0.863	0.782
average	0.842	0.863	0.796

Table 2: Effect of # of slices for semi-automated segmentation (IoU)

GT slices	GT voxels	IoU S1	IoU S2	IoU S3
1,1,1	2.5%	0.331	0.483	0.475
2,2,1	3.3%	0.676	0.579	0.738
3,3,2	5.7%	0.761	0.808	0.835
5,5,3	8.9%	0.856	0.849	0.872

Table 3: Cross validation results for fully-automated segmentation (IoU)

test volume	3D w/o BN	3D with BN	2D with BN
1	0.655	0.761	0.619
2	0.734	0.798	0.698
3	0.779	0.554	0.325
average	0.723	0.704	0.547

Idea: Patch

Spatially Localized Atlas Network Tiles Enables 3D Whole Brain Segmentation from Limited Data

Yuankai Huo[1(✉)], Zhoubing Xu[1], Katherine Aboud[1], Prasanna Parvathaneni[1], Shunxing Bao[1], Camilo Bermudez[1], Susan M. Resnick[2], Laurie E. Cutting[1], and Bennett A. Landman[1]

[1] Vanderbilt University, Nashville, TN, USA
yuankai.huo@vanderbilt.edu
[2] Laboratory of Behavioral Neuroscience, National Institute on Aging, Baltimore, MD, USA

Abstract. Whole brain segmentation on a structural magnetic resonance imaging (MRI) is essential in non-invasive investigation for neuroanatomy. Historically, multi-atlas segmentation (MAS) has been regarded as the *de facto* standard method for whole brain segmentation. Recently, deep neural network approaches have been applied to whole brain segmentation by learning random patches or 2D slices. Yet, few previous efforts have been made on detailed whole brain segmentation using 3D networks due to the following challenges: (1) fitting entire whole brain volume into 3D networks is restricted by the current GPU memory, and (2) the large number of targeting labels (e.g., >100 labels) with limited number of training 3D volumes (e.g., <50 scans). In this paper, we propose the spatially localized atlas network tiles (SLANT) method to distribute multiple independent 3D fully convolutional networks to cover overlapped sub-spaces in a standard atlas space. This strategy simplifies the whole brain learning task to localized sub-tasks, which was enabled by combing canonical registration and label fusion techniques with deep learning. To address the second challenge, auxiliary labels on 5111 initially unlabeled scans were created by MAS for pre-training. From empirical validation, the state-of-the-art MAS method achieved mean Dice value of 0.76, 0.71, and 0.68, while the proposed method achieved 0.78, 0.73, and 0.71 on three validation cohorts. Moreover, the computational time reduced from >30 h using MAS to ≈15 min using the proposed method. The source code is available online (https://github.com/MASILab/SLANTbrainSeg).

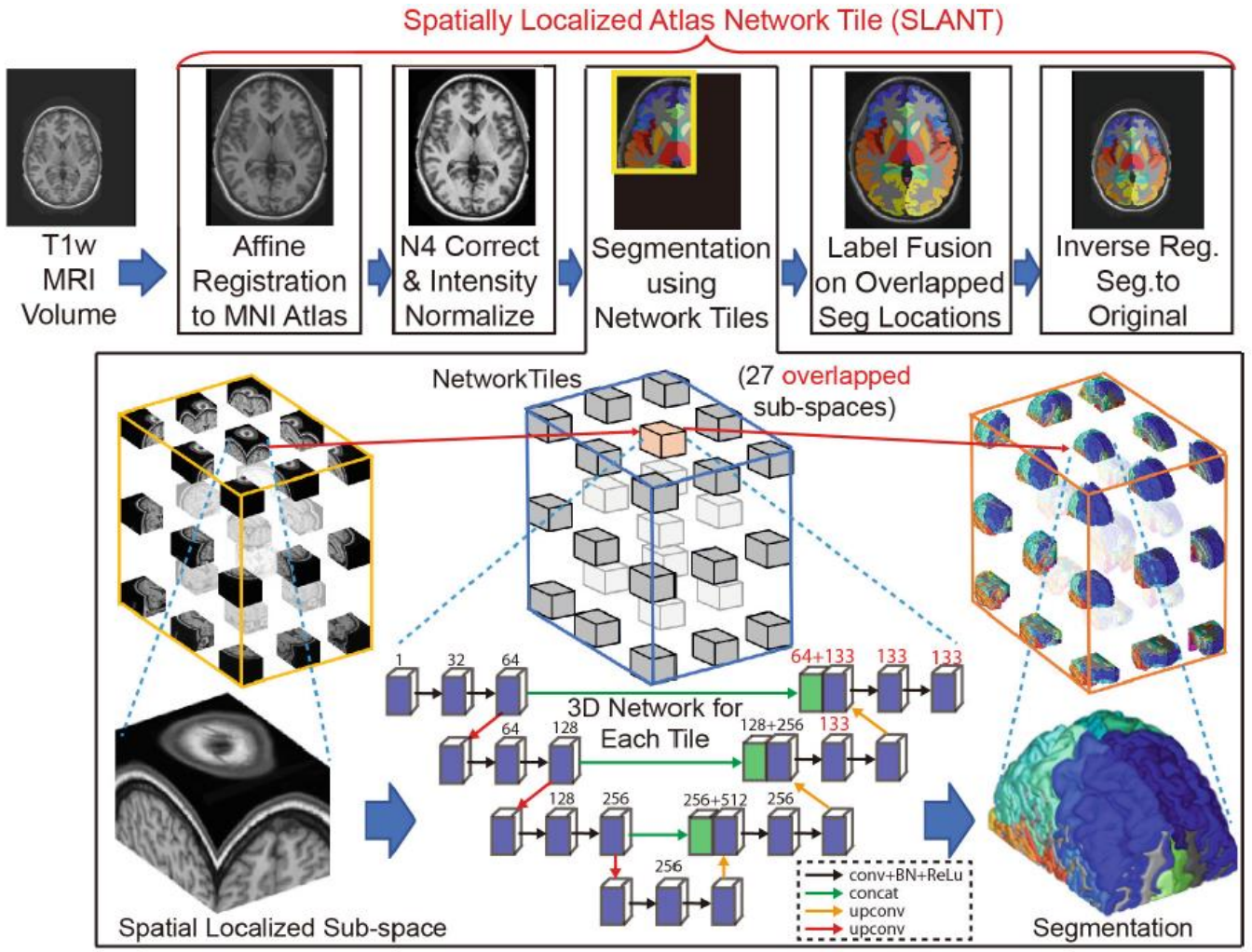


Fig. 1. The proposed SLANT-27 (27 network tiles) method is presented, which combines canonical medical image processing methods (registration, harmonization, label fusion) with 3D network tiles. 3D U-Net is used as each tile, whose deconvolutional channel numbers are modified to 133. The tiles are spatially overlapped in MNI space.

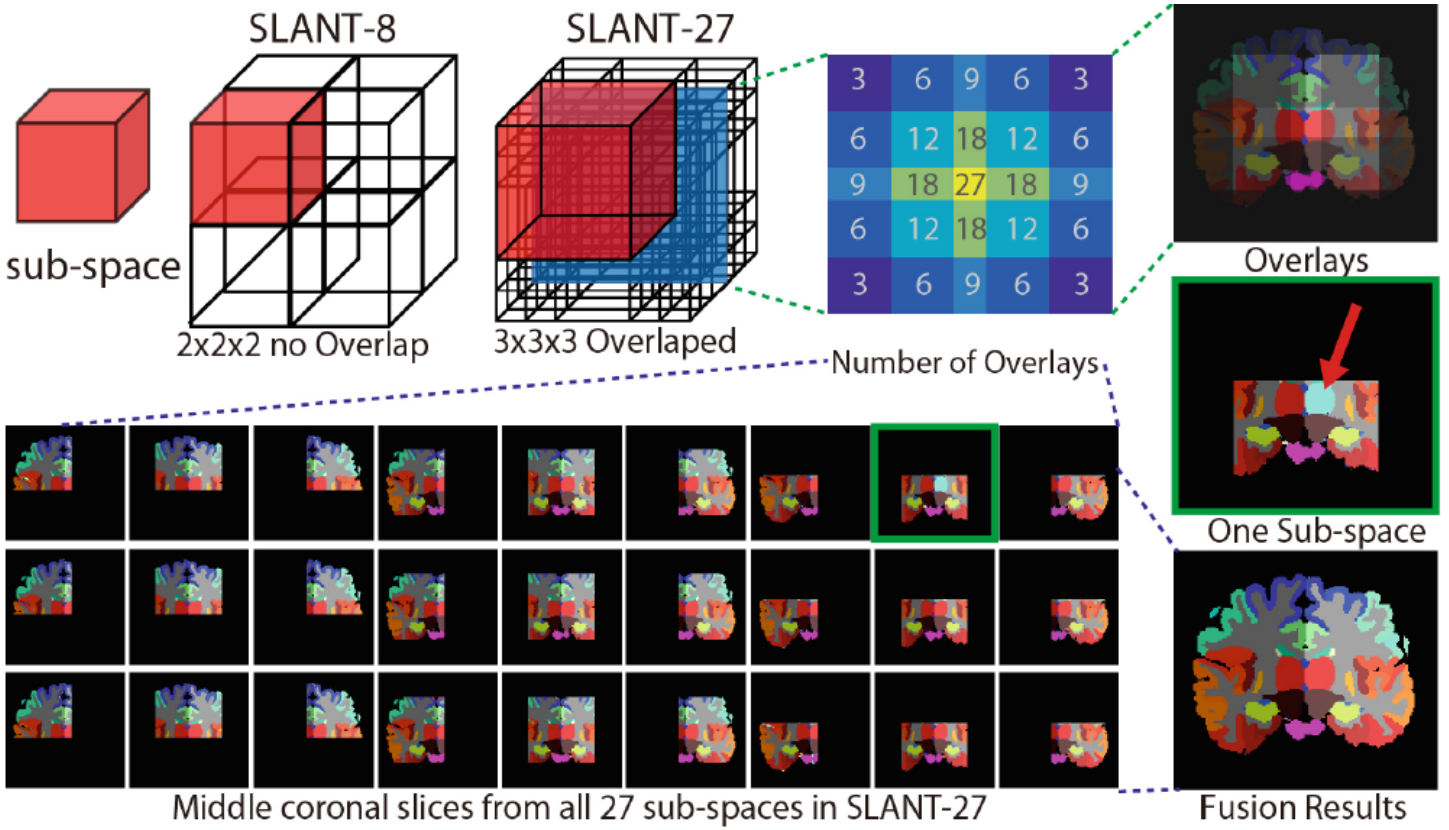


Fig. 2. SLANT-8 covered eight non-overlapped sub-spaces in MNI, while SLANT-27 covered 27 overlapped sub-spaces in MNI. Middle coronal slices from all 27 sub-spaces were visualized (lower left panel). The number of overlays, as well as sub-spaces overlays, were showed (upper right panel). The incorrect labels (red arrow) in one sub-space were corrected in final segmentation by performing majority vote label fusion.

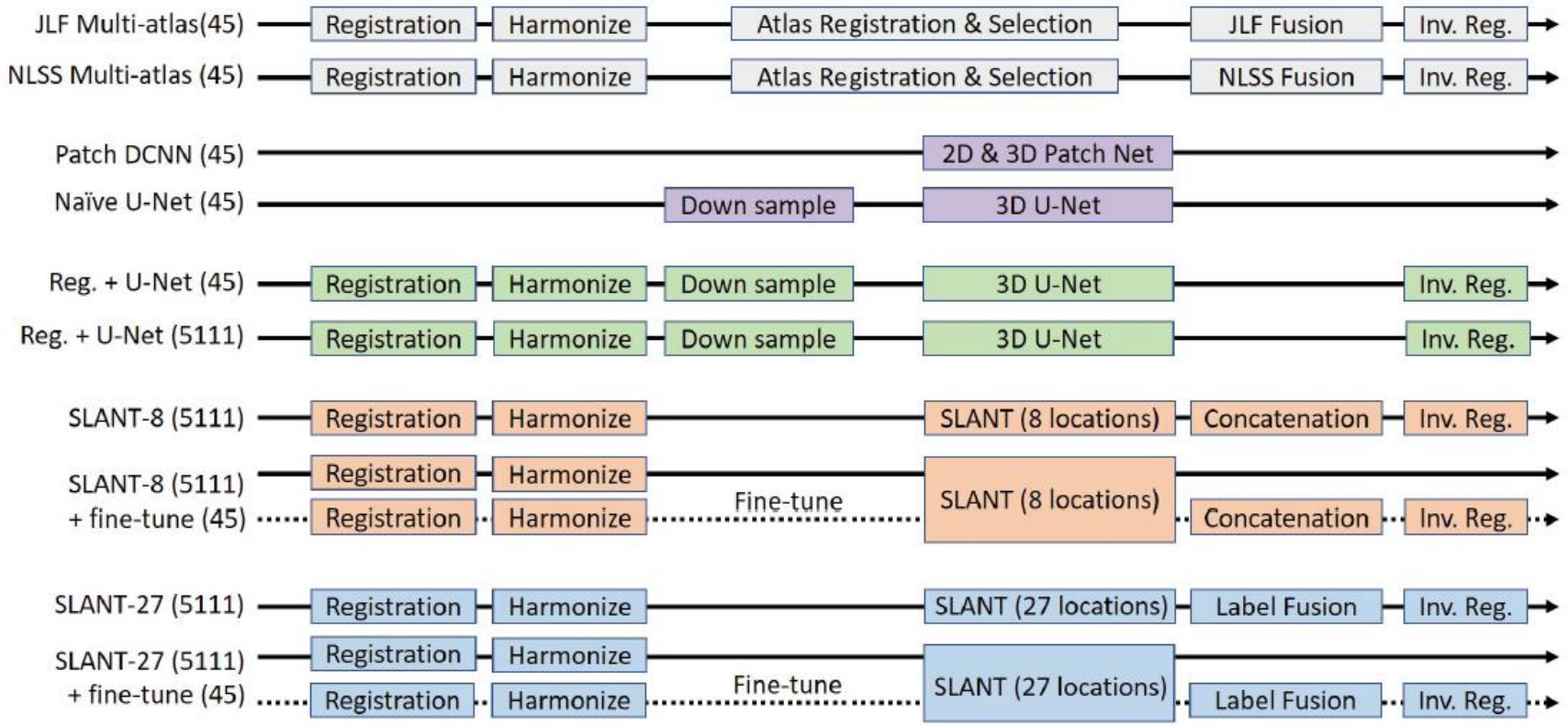


Fig. 3. This figure demonstrates the major components of different segmentation methods. (45) indicated the 45 OASIS manually traced images were used in training, while (5111) indicated the 5111 auxiliary label images were used in training.

AssemblyNet: A Novel Deep Decision-Making Process for Whole Brain MRI Segmentation

Pierrick Coupé[1], Boris Mansencal[1], Michaël Clément[1], Rémi Giraud[2], Baudouin Denis de Senneville[3], Vinh-Thong Ta[1], Vincent Lepetit[1] and José V. Manjon[4]

[1] CNRS, Univ. Bordeaux, Bordeaux INP, LABRI, UMR5800, F-33400 Talence, France
[2] Bordeaux INP, Univ. Bordeaux, CNRS, IMS, UMR 5218, F-33400 Talence, France
[3] CNRS, Univ. Bordeaux, IMB, UMR 5251, F-33400 Talence, France
[4] ITACA, Universitat Politècnica de València, 46022 Valencia, Spain

Abstract. Whole brain segmentation using deep learning (DL) is a very challenging task since the number of anatomical labels is very high compared to the number of available training images. To address this problem, previous DL methods proposed to use a global convolution neural network (CNN) or few independent CNNs. In this paper, we present a novel ensemble method based on a large number of CNNs processing different overlapping brain areas. Inspired by parliamentary decision-making systems, we propose a framework called AssemblyNet, made of two "assemblies" of U-Nets. Such a parliamentary system is capable of dealing with complex decisions and reaching a consensus quickly. AssemblyNet introduces sharing of knowledge among neighboring U-Nets, an "amendment" procedure made by the second assembly at higher-resolution to refine the decision taken by the first one, and a final decision obtained by majority voting. When using the same 45 training images, AssemblyNet outperforms global U-Net by 28% in terms of the Dice metric, patch-based joint label fusion by 15% and SLANT-27 by 10%. Finally, AssemblyNet demonstrates high capacity to deal with limited training data to achieve whole brain segmentation in practical training and testing times.

Keywords: Whole brain segmentation, CNN, Ensemble learning, transfer learning, multiscale framework.

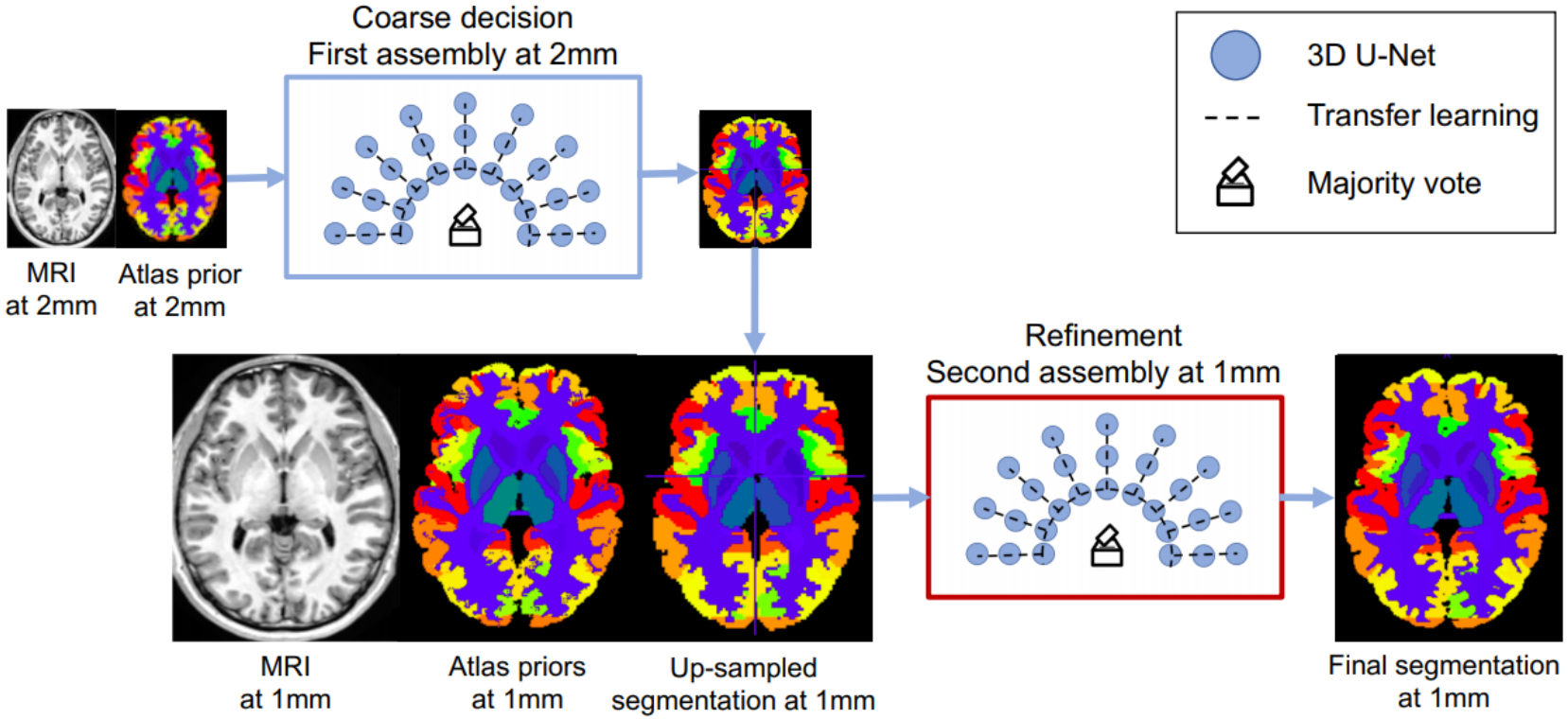


Fig. 1. Illustration of the proposed AssemblyNet framework.

Idea: 3D to 2D

3D Context Enhanced Region-Based Convolutional Neural Network for End-to-End Lesion Detection

Ke Yan[1], Mohammadhadi Bagheri[2], and Ronald M. Summers[1(✉)]

[1] Imaging Biomarkers and Computer-Aided Diagnosis Laboratory, Bethesda, MD, USA

[2] Clinical Image Processing Service, Department of Radiology and Imaging Sciences, National Institutes of Health Clinical Center, Bethesda, MD 20892-1182, USA

{ke.yan,mohammad.bagheri,rms}@nih.gov

Abstract. Detecting lesions from computed tomography (CT) scans is an important but difficult problem because non-lesions and true lesions can appear similar. 3D context is known to be helpful in this differentiation task. However, existing end-to-end detection frameworks of convolutional neural networks (CNNs) are mostly designed for 2D images. In this paper, we propose 3D context enhanced region-based CNN (3DCE) to incorporate 3D context information efficiently by aggregating feature maps of 2D images. 3DCE is easy to train and end-to-end in training and inference. A universal lesion detector is developed to detect all kinds of lesions in one algorithm using the DeepLesion dataset. Experimental results on this challenging task prove the effectiveness of 3DCE.

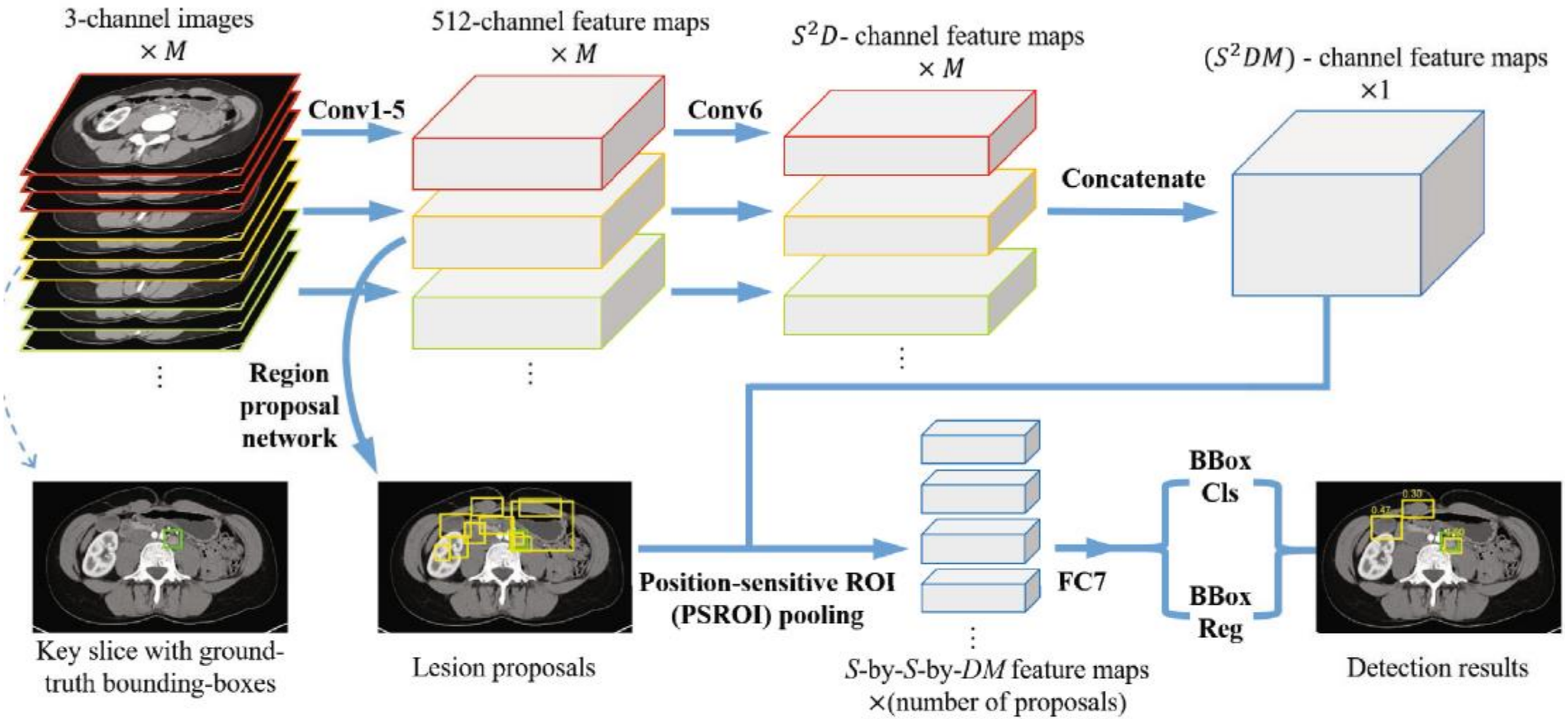


Fig. 1. The framework of 3DCE for lesion detection.

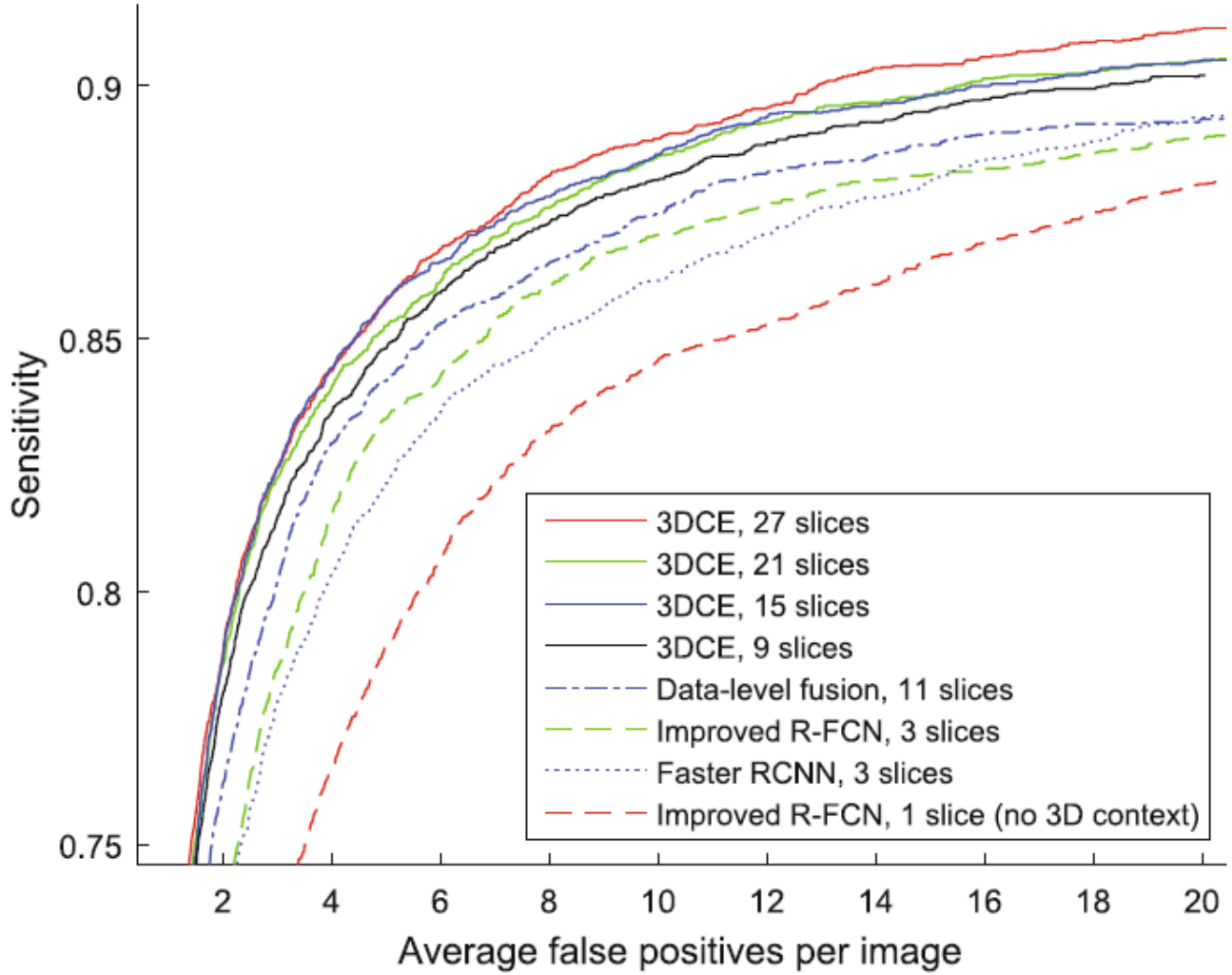


Fig. 2. FROC curves of various methods on the test set of DeepLesion (4802 images).

Harnessing 2D Networks and 3D Features for Automated Pancreas Segmentation from Volumetric CT Images

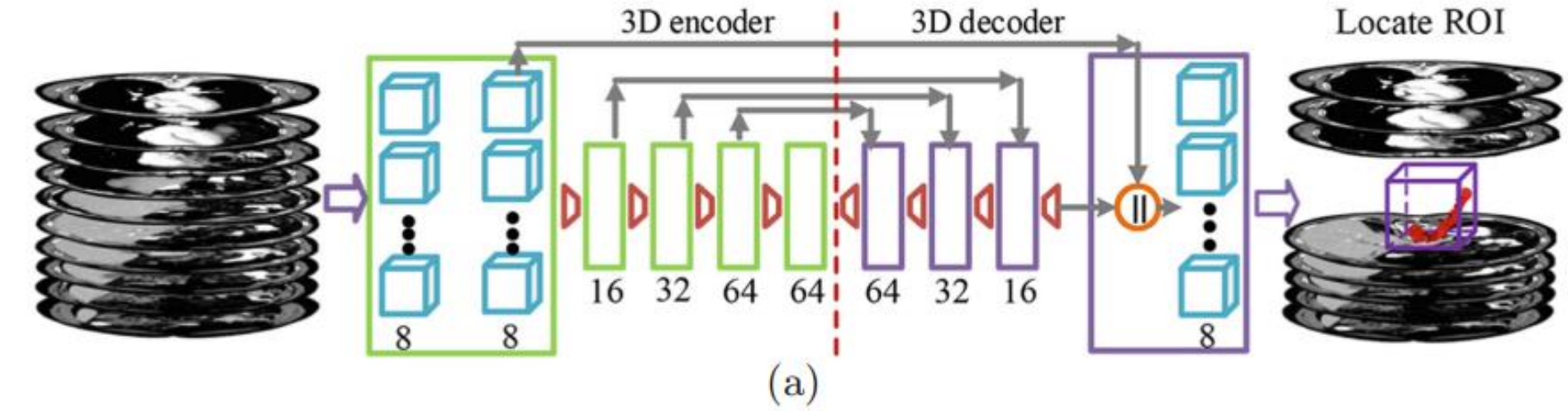
Huai Chen[1], Xiuying Wang[2], Yijie Huang[1], Xiyi Wu[1], Yizhou Yu[3], and Lisheng Wang[1(✉)]

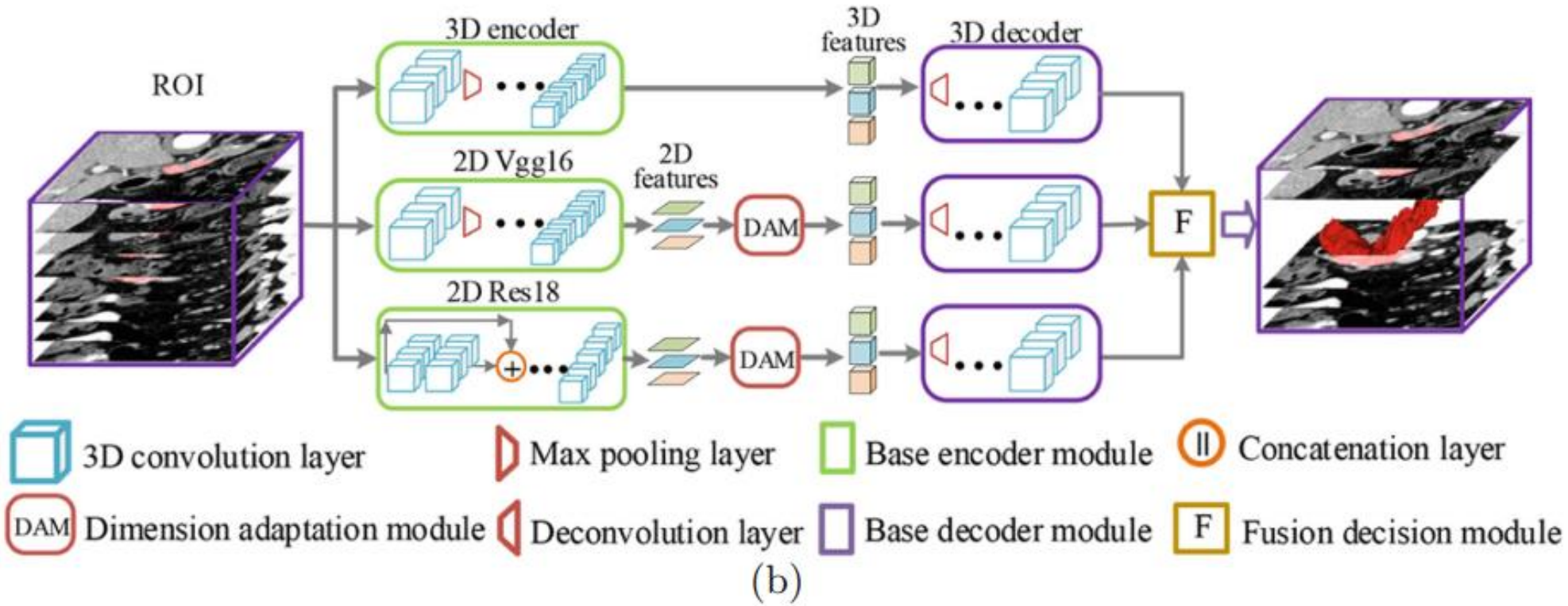
[1] Institute of Image Processing and Pattern Recognition, Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, People's Republic of China
lswang@sjtu.edu.cn

[2] School of Computer Science, The University of Sydney, Sydney, Australia

[3] Deepwise AI Lab, Beijing, People's Republic of China

Abstract. Segmenting pancreas from abdominal CT scans is an important prerequisite for pancreatic cancer diagnosis and precise treatment planning. However, automated pancreas segmentation faces challenges posed by shape and size variances, low contrast with regard to adjacent tissues and in particular negligibly small proportion to the whole abdominal volume. Current coarse-to-fine frameworks, either using triplanar schemes or stacking 2D pre-segmentation as prior to 3D networks, have limitation on effectively capturing 3D information. While iterative updates on region of interest (ROI) in refinement stage alleviate accumulated errors caused by coarse segmentation, extra computational burden is introduced. In this paper, we harness 2D networks and 3D features to improve segmentation accuracy and efficiency. Firstly, in the 3D coarse segmentation network, a new bias-dice loss function is defined to increase ROI recall rates to improve efficiency by avoiding iterative ROI refinements. Secondly, for a full utilization of 3D information, dimension adaptation module (DAM) is introduced to bridge 2D networks and 3D information. Finally, a fusion decision module and parallel training strategy are proposed to fuse multi-source feature cues extracted from three sub-networks to make final predictions. The proposed method is evaluated in NIH dataset and outperforms the state-of-the-art methods in comparison, with mean Dice-Sørensen coefficient (DSC) of 85.22%, and with averagely 0.4 min for each instance.


3D encoder
3D decoder
Locate ROI
8
8
16
32
64
64
64
32
16
8
(a)


ROI
3D encoder
3D features
3D decoder
2D Vgg16
2D features
DAM
2D Res18
DAM
F
3D convolution layer
Max pooling layer
Base encoder module
Concatenation layer
DAM
Dimension adaptation module
Deconvolution layer
Base decoder module
F
Fusion decision module
(b)

Idea: 2D to 3D

Bridging the Gap Between 2D and 3D Organ Segmentation with Volumetric Fusion Net

Yingda Xia[1], Lingxi Xie[1(✉)], Fengze Liu[1], Zhuotun Zhu[1], Elliot K. Fishman[2], and Alan L. Yuille[1]

[1] The Johns Hopkins University, Baltimore, MD 21218, USA
198808xc@gmail.com
[2] The Johns Hopkins University School of Medicine, Baltimore, MD 21287, USA

Abstract. There has been a debate on whether to use 2D or 3D deep neural networks for volumetric organ segmentation. Both 2D and 3D models have their advantages and disadvantages. In this paper, we present an alternative framework, which trains 2D networks on different viewpoints for segmentation, and builds a 3D **Volumetric Fusion Net** (VFN) to fuse the 2D segmentation results. VFN is relatively shallow and contains much fewer parameters than most 3D networks, making our framework more efficient at integrating 3D information for segmentation. We train and test the segmentation and fusion modules individually, and propose a novel strategy, named *cross-cross-augmentation*, to make full use of the limited training data. We evaluate our framework on several challenging abdominal organs, and verify its superiority in segmentation accuracy and stability over existing 2D and 3D approaches.

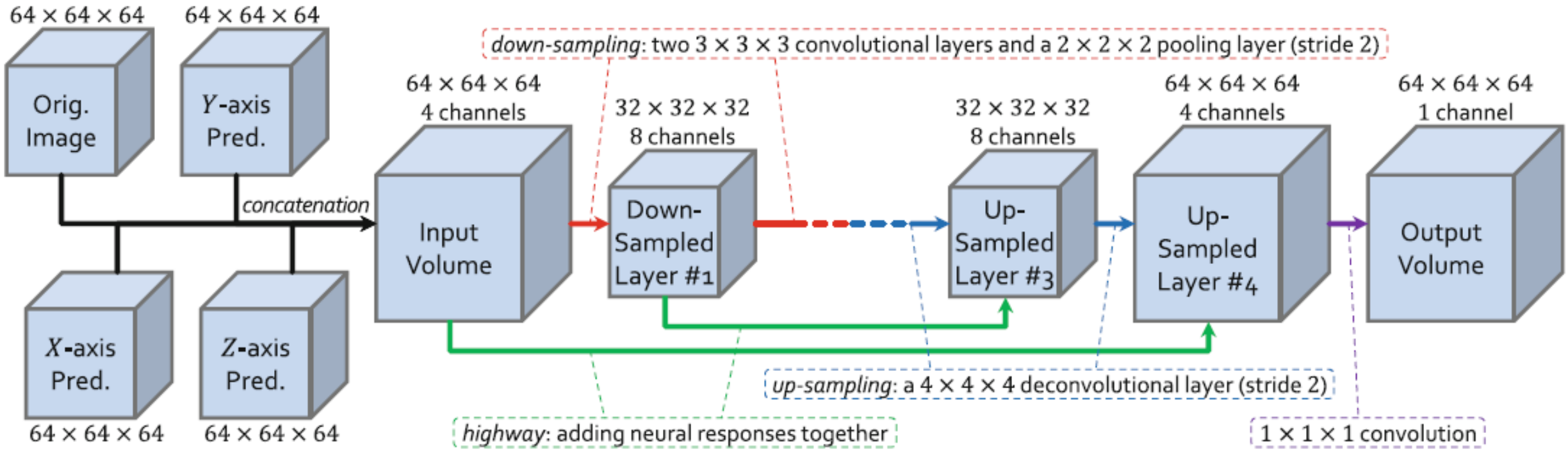


Fig. 1. The network structure of VFN. We only display one down-sampling and one up-sampling stages, but there are 3 of each. Each down-sampling stage shrinks the spatial resolution by 1/2 and doubles the number of channels. We build 3 highway connections (2 are shown). We perform batch normalization and ReLU activation after each convolutional and deconvolutional layer.

3D Anisotropic Hybrid Network: Transferring Convolutional Features from 2D Images to 3D Anisotropic Volumes

Siqi Liu[1(✉)], Daguang Xu[1], S. Kevin Zhou[1], Olivier Pauly[2], Sasa Grbic[1], Thomas Mertelmeier[2], Julia Wicklein[2], Anna Jerebko[2], Weidong Cai[3], and Dorin Comaniciu[1]

[1] Medical Imaging Technologies, Siemens Healthineers, Princeton, NJ, USA
siqi.liu@siemens-healthineers.com
[2] X-Ray Products, Siemens Healthineers, Erlangen, Germany
[3] School of Information Technologies, University of Sydney, Sydney, Australia

Abstract. While deep convolutional neural networks (CNN) have been successfully applied to 2D image analysis, it is still challenging to apply them to 3D medical images, especially when the within-slice resolution is much higher than the between-slice resolution. We propose a 3D Anisotropic Hybrid Network (AH-Net) that transfers convolutional features learned from 2D images to 3D anisotropic volumes. Such a transfer inherits the desired strong generalization capability for within-slice information while naturally exploiting between-slice information for more effective modelling. We experiment with the proposed 3D AH-Net on two different medical image analysis tasks, namely lesion detection from a Digital Breast Tomosynthesis volume, and liver and liver tumor segmentation from a Computed Tomography volume and obtain state-of-the-art results.

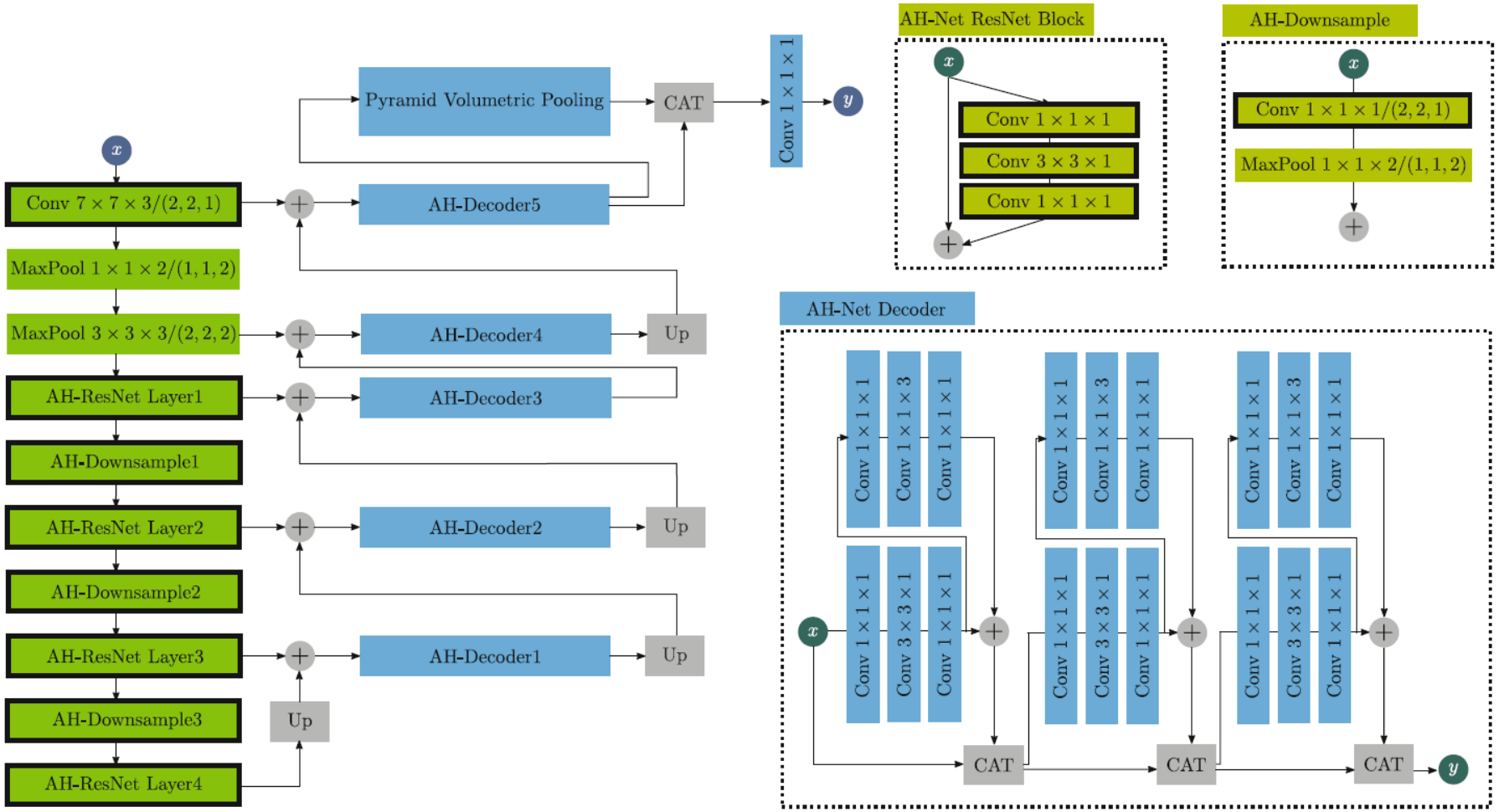
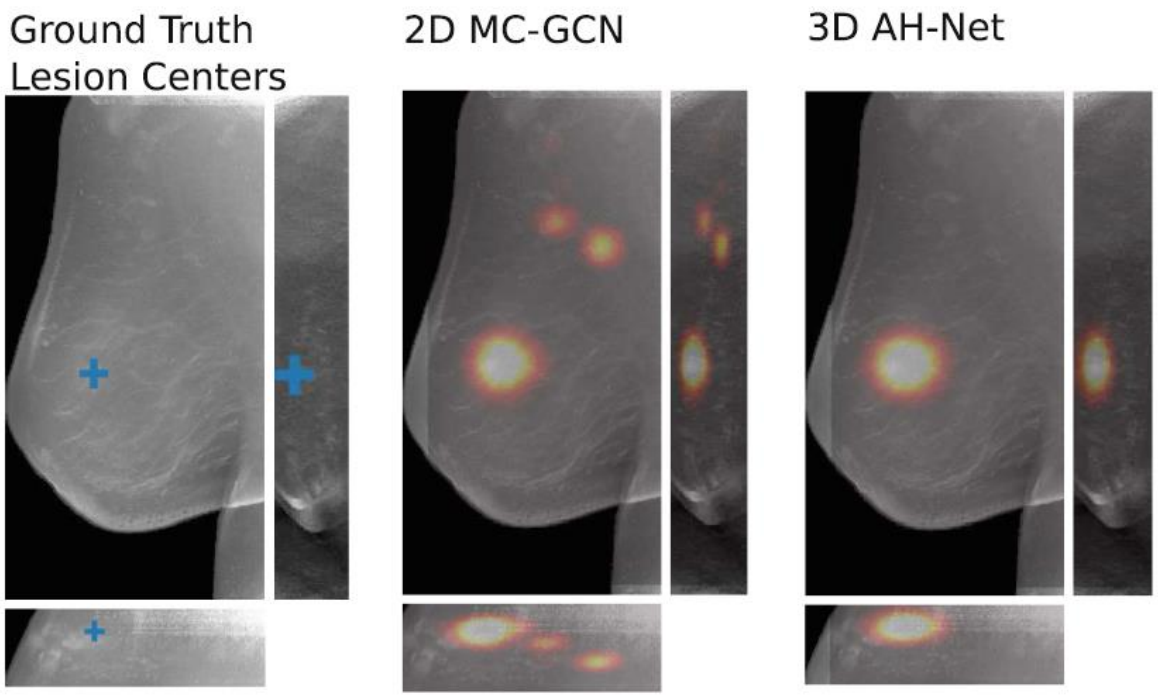
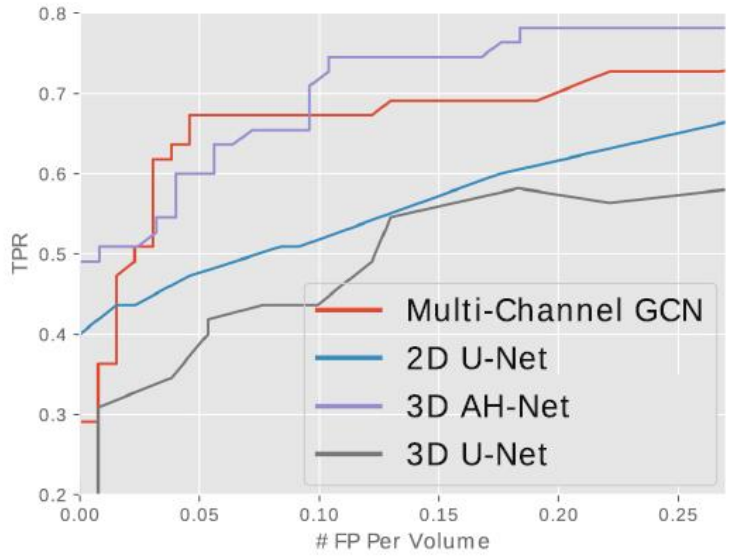


Fig. 1. The architecture of 3D AH-Net. We hide the batch normalization and ReLu layers for brevity. The parameters of the blocks with bold boundaries are transformed from the pre-trained 2D ResNet50 encoder.



(a) DBT Visual



(b) DBT FROC

Fig. 2. Left: The visual comparisons of the network responses on a DBT volume from 2D MC-GCN and the 3D AH-Net with the encoder weights transferred from it. Right: FROC curves of the compared networks on the DBT dataset.

3D Fetal Skull Reconstruction from 2DUS via Deep Conditional Generative Networks

Juan J. Cerrolaza[1(✉)], Yuanwei Li[1], Carlo Biffi[1], Alberto Gomez[2], Matthew Sinclair[1], Jacqueline Matthew[2], Caronline Knight[2], Bernhard Kainz[1], and Daniel Rueckert[1]

[1] Biomedical Image Analysis Group, Imperial College London, London, UK
j.cerrolaza-martinez@imperial.ac.uk
[2] Division of Imaging Sciences and Biomedical Engineering, King's College London, London, UK

Abstract. 2D ultrasound (US) is the primary imaging modality in antenatal healthcare. Despite the limitations of traditional 2D biometrics to characterize the true 3D anatomy of the fetus, the adoption of 3DUS is still very limited. This is particularly significant in developing countries and remote areas, due to the lack of experienced sonographers and the limited access to 3D technology. In this paper, we present a new deep conditional generative network for the 3D reconstruction of the fetal skull from 2DUS standard planes of the head routinely acquired during the fetal screening process. Based on the generative properties of conditional variational autoencoders (CVAE), our reconstruction architecture (REC-CVAE) directly integrates the three US standard planes as conditional variables to generate a unified latent space of the skull. Additionally, we propose HiREC-CVAE, a hierarchical generative network based on the different clinical relevance of each predictive view. The hierarchical structure of HiREC-CVAE allows the network to learn a sequence of nested latent spaces, providing superior predictive capabilities even in the absence of some of the 2DUS scans. The performance of the proposed architectures was evaluated on a dataset of 72 cases, showing accurate reconstruction capabilities from standard non-registered 2DUS images.

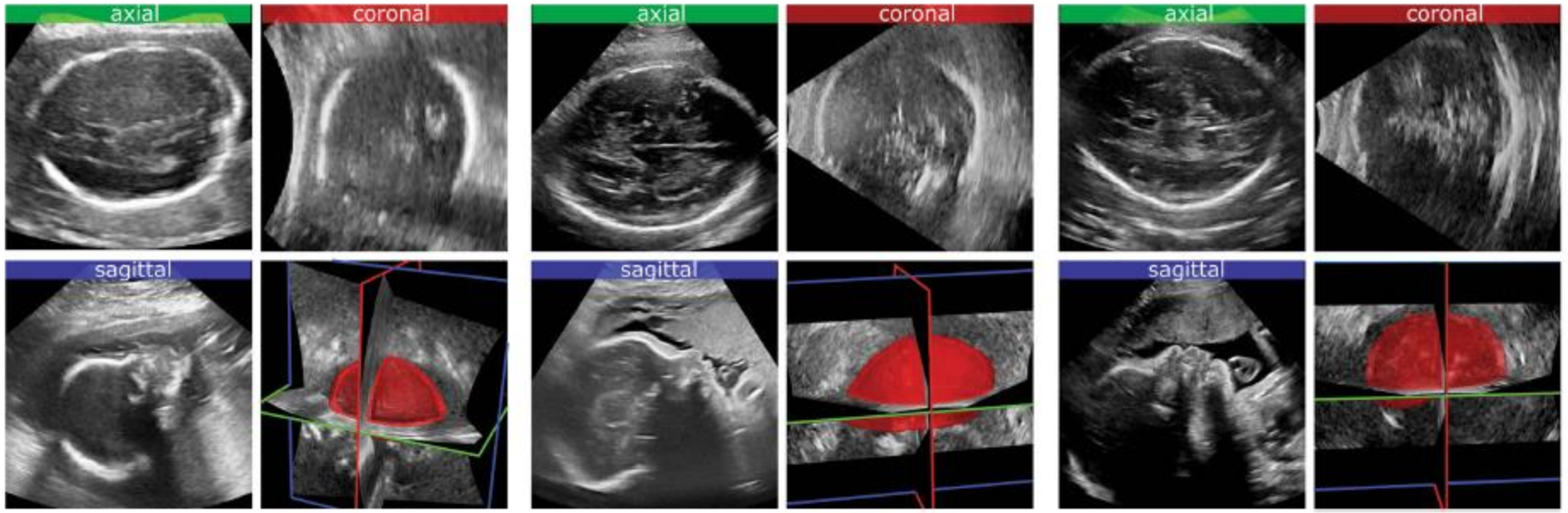


Fig. 1. Fetal standard US scans of the head. Example of the axial (green), sagittal (blue), and coronal (red) standard planes of three patients acquired in freeheand 2DUS during the routine mid-trimester US examination. The image also shows a 3D representation of the skull manually segmented from the corresponding 3DUS volume.

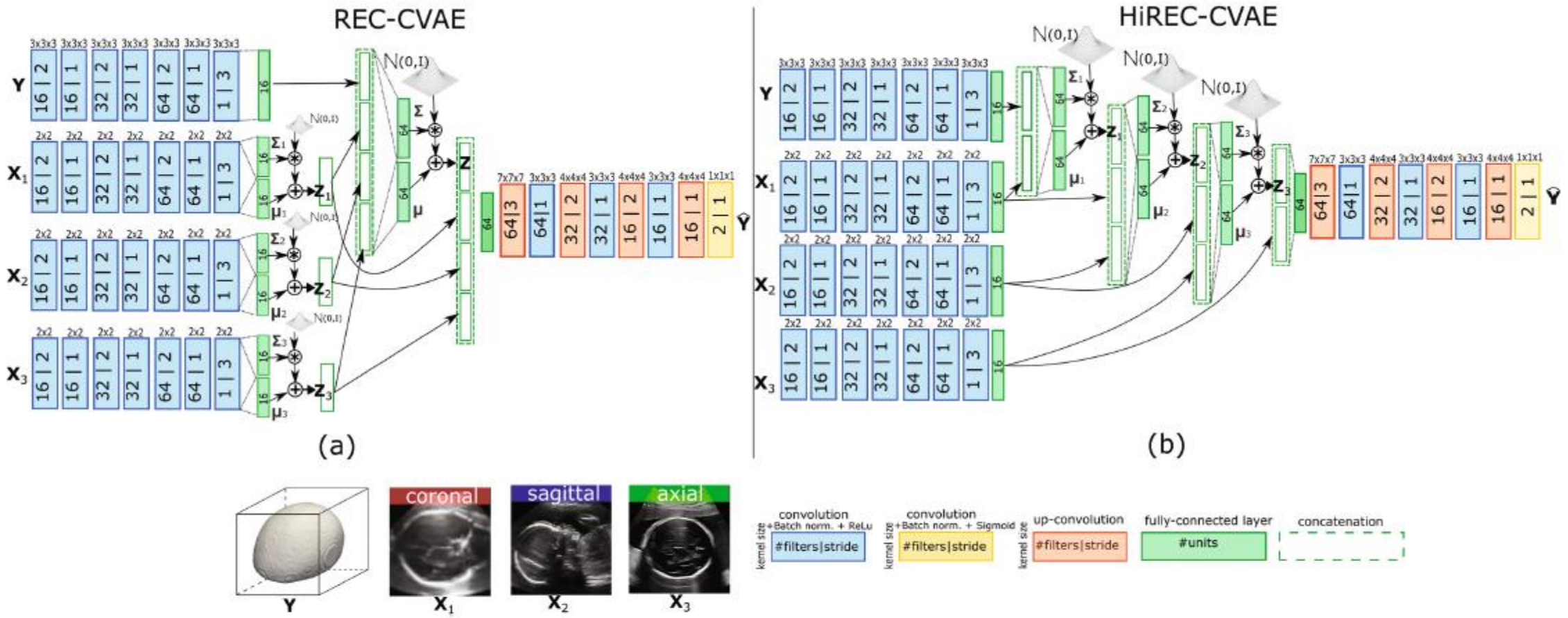


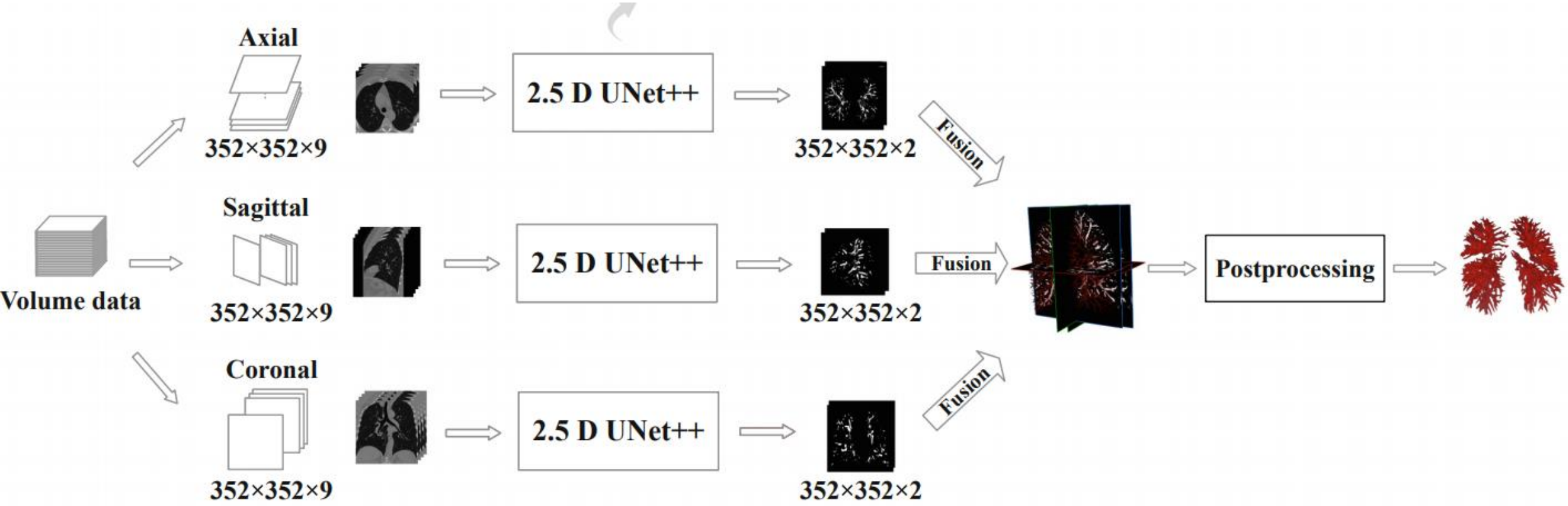
Fig. 2. Deep generative networks for the reconstruction of the fetal skull. (a) REC-CVAE: Reconstruction network based on the conditional variational autoencoder framework. (b) HiREC-CVAE: Hierarchical reconstruction network.

Pulmonary Vessel Segmentation Based on Orthogonal Fused U-Net++ of Chest CT Images

Hejie Cui[1,2], Xinglong Liu[1], and Ning Huang[1(✉)]

[1] SenseTime Research, Beijing, China
huangning@sensetime.com
[2] Emory University, Atlanta, GA, USA

Abstract. Pulmonary vessel segmentation is important for clinical diagnosis of pulmonary diseases, while is also challenging due to the complicated structure. In this work, we present an effective framework and refinement process of pulmonary vessel segmentation from chest computed tomographic (CT) images. The key to our approach is a 2.5D segmentation network applied from three orthogonal axes, which presents a robust and fully automated pulmonary vessel segmentation result with lower network complexity and memory usage compared to 3D networks. The slice radius is introduced to convolve the adjacent information of the center slice and the multi-planar fusion optimizes the presentation of intra and inter slice features. Besides, the tree-like structure of pulmonary vessel is extracted in the post-processing process, which is used for segmentation refining and pruning. In the evaluation experiments, three fusion methods are tested and the most promising one is compared with the state-of-the-art 2D and 3D structures on 300 cases of lung images randomly selected from LIDC dataset. Our method outperforms other network structures by a large margin and achieves by far the highest average DICE score of 0.9272 and a precision of 0.9310, as per our knowledge from the pulmonary vessel segmentation models available in literature.


Axial
352×352×9
2.5 D UNet++
352×352×2
Fusion
Sagittal
Volume data
352×352×9
2.5 D UNet++
352×352×2
Fusion
Postprocessing
Coronal
352×352×9
2.5 D UNet++
352×352×2
Fusion

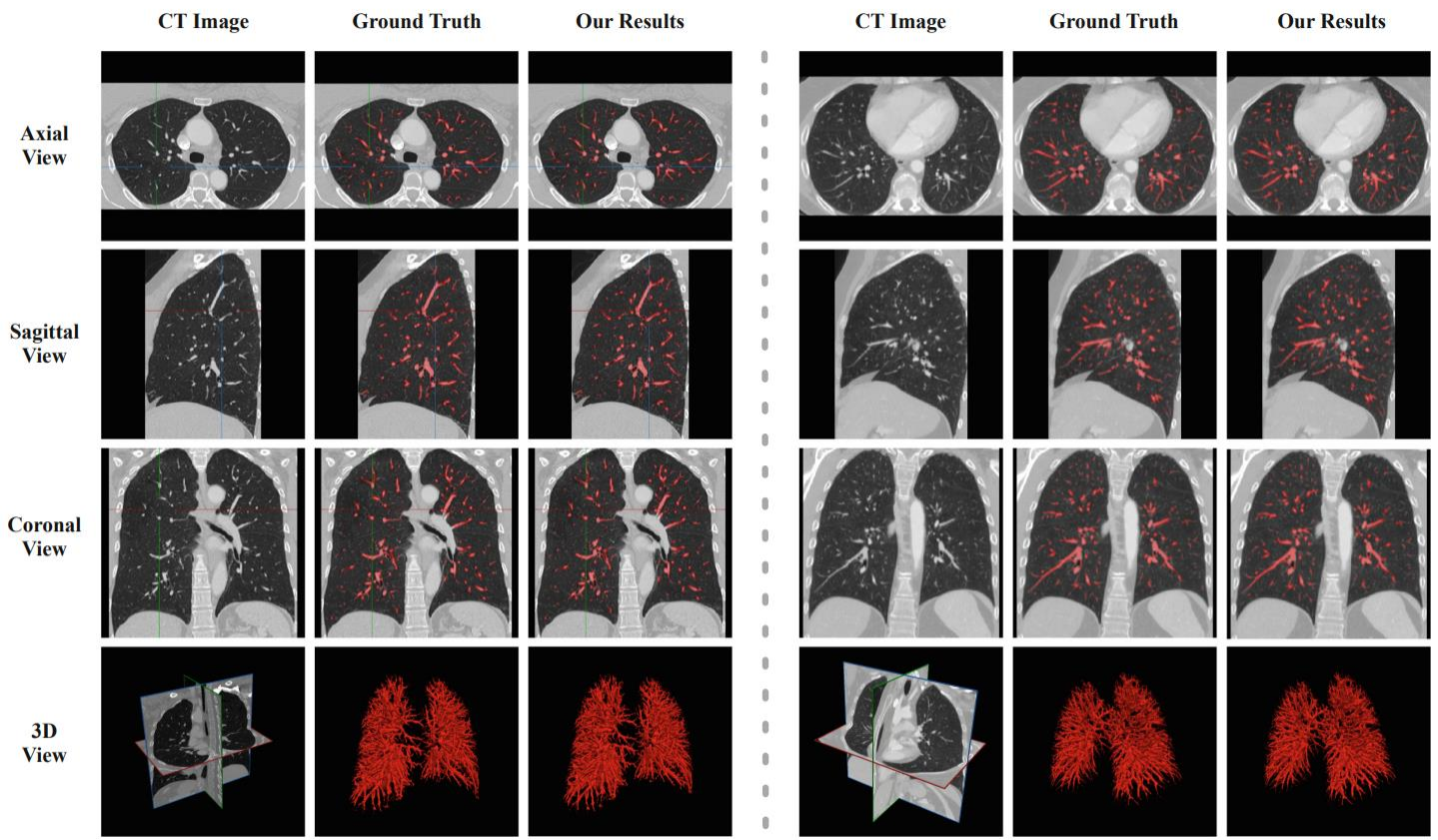


Table 2. Comparison of three state-of-art structures.

Structures	Min dice	Max dice	Avg. dice	Precision/Recall
2D U-net++	0.5201	0.7376	0.6628	0.6629/0.6767
3D U-Net++	0.4385	0.8038	0.7286	0.7425/0.7436
2.5D U-Net++	**0.8779**	**0.9627**	**0.9262**	**0.9310/0.9272**

Idea: Memory

High Resolution Medical Image Segmentation Using Data-Swapping Method

Haruki Imai[1(✉)], Samuel Matzek[2], Tung D. Le[1], Yasushi Negishi[1], and Kiyokuni Kawachiya[1]

[1] IBM Research, Tokyo, Japan
{imaihal, tung, negishi, kawatiya}@jp.ibm.com
[2] IBM Systems, Rochester, MN, USA
smatzek@us.ibm.com

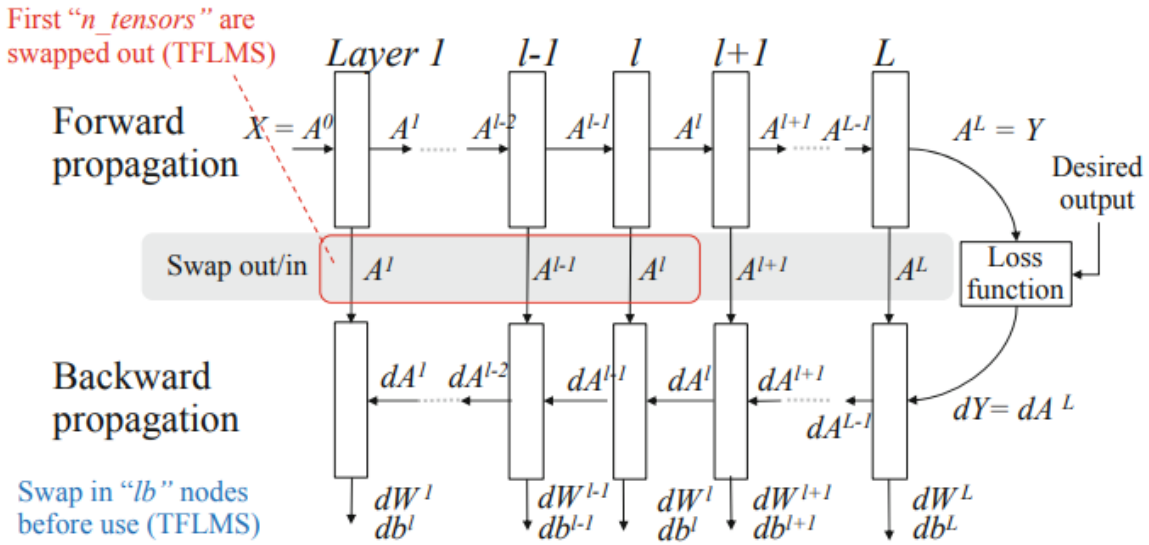


Fig. 1. Processing and data flow of L layers in data-swapping method

192 x 192 x 192 voxels