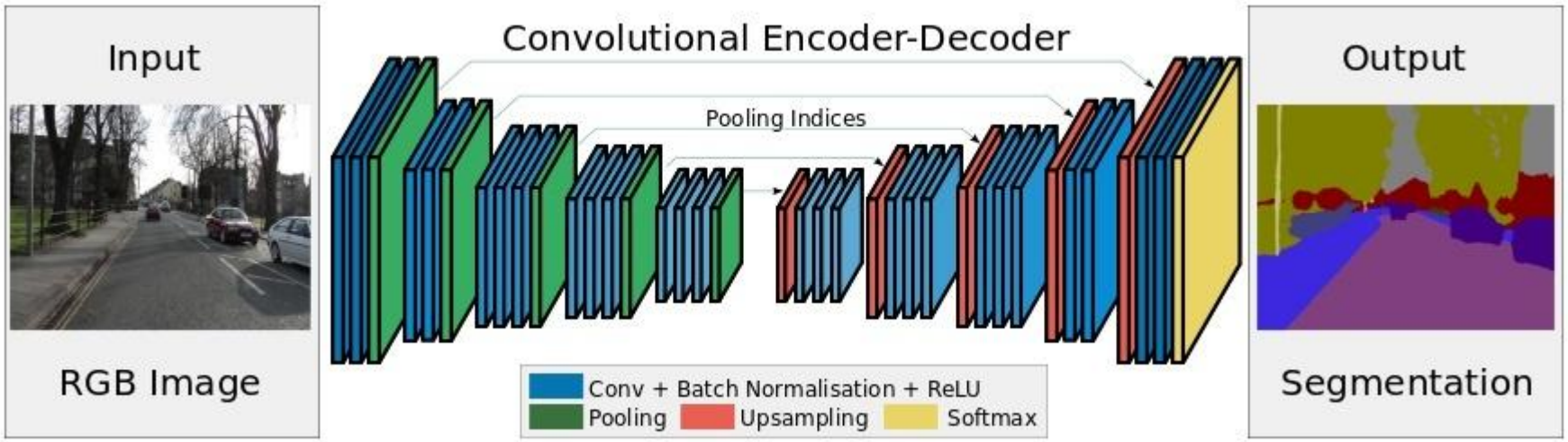
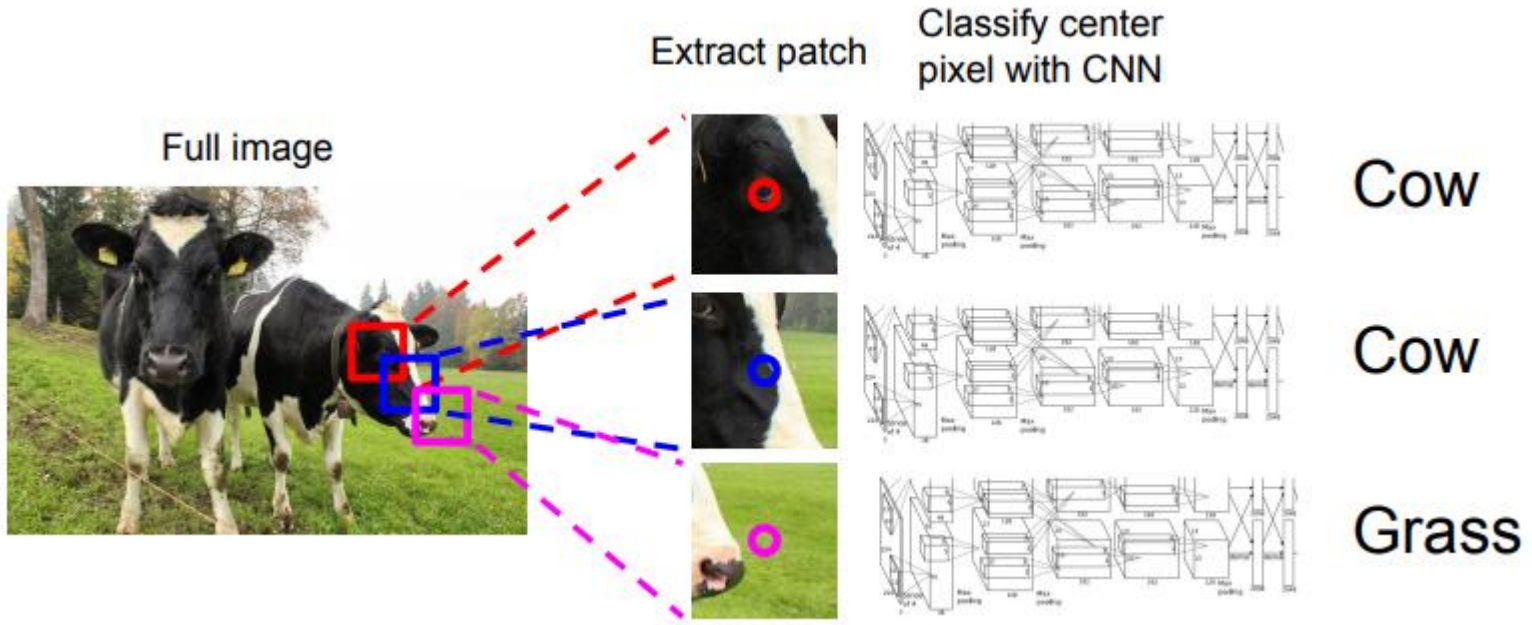
Classical Semantic Segmentation Architectures



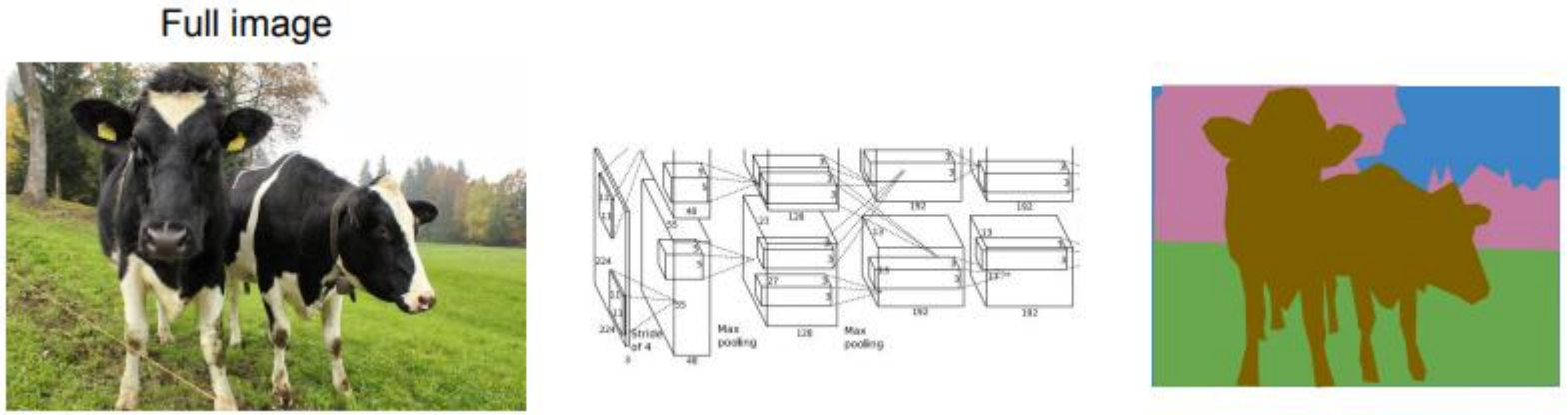
滑动窗口

语义分割任务最初流行的深度学习方法是图像块分类（patch classification），即利用像素周围的图像块对每一个像素进行独立的分类。使用图像块分类的主要原因是分类网络中包含全连接层（fully connected layer），它需要固定尺寸的图像。



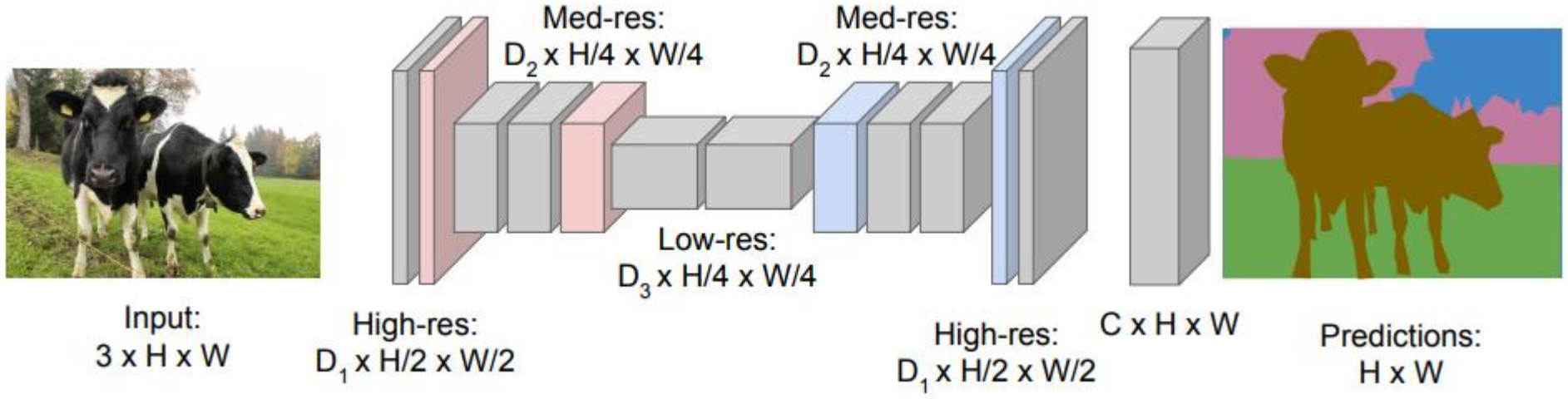
直接卷积

将整个图像放入卷积网络，最后输出分割图像？
尺寸不匹配



FCN分割网络

2014 年，加州大学伯克利分校的 Long等人提出全卷积网络（FCN），这使得卷积神经网络无需全连接层即可进行密集的像素预测。使用这种方法可生成任意大小的图像分割图，且该方法比图像块分类法要高效许多。FCN-8s 架构在 Pascal VOC 2012 数据集上的性能相比以前的方法提升了 20%，达到了 62.2% 的 mIOU。这种encoder-decoder架构是语义分割的基础，此后一些新的和更好的体系结构都基于此。



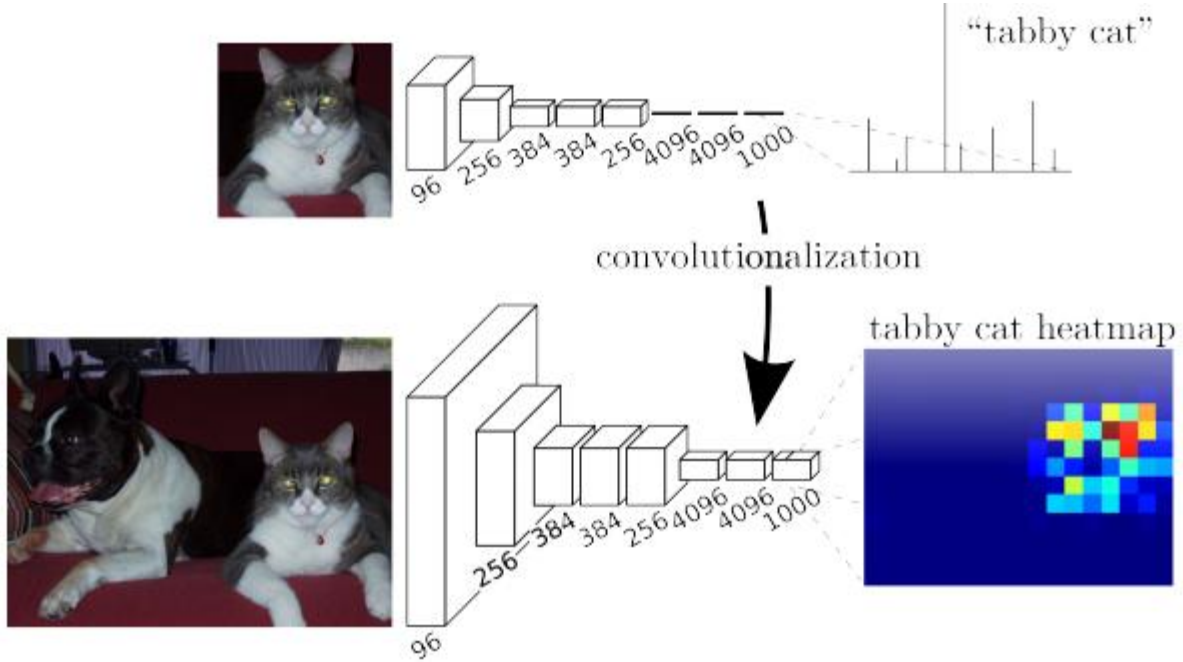
FCN分割网络

FCN是一种端到端的语义分割模型，主要创新点如下：

1. 全卷积化(Fully Convolutional)：用于解决逐像素(pixel-wise)的预测问题。通过将基础网络(例如VGG)最后面几个全连接层换成卷积层，可实现任意大小的图像输入，并且输出图像大小与输入相对应；
2. 反卷积(deconvolution)：上采样操作，用于恢复图片尺寸，方便后续逐像素预测；
3. 跳跃结构(skip architecture)：用于融合高低层特征信息。通过跨层连接的结构，结合了网络浅层的细(fine-grain)粒度信息信息以及深层的粗糙(coarse)信息，以实现精准的分割任务。

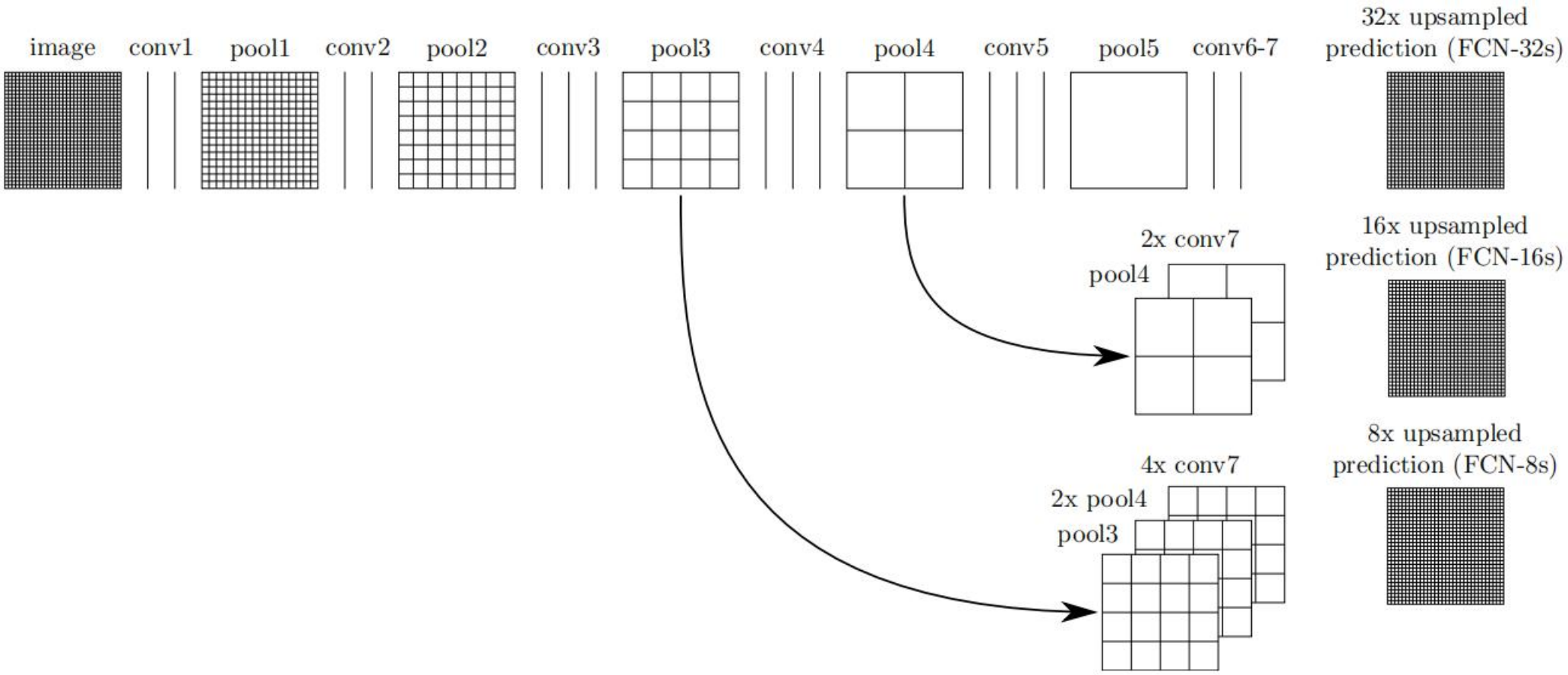
模型思想

对于一般的分类CNN网络，如VGG和Resnet，都会在网络的最后加入一些全连接层，经过softmax后就可以获得类别概率信息。但是这个概率信息是1维的，即只能标识整个图片的类别，不能标识每个像素点的类别，所以这种全连接方法不适用于图像分割。但实际上，全连接层也可以看做是卷积层。如图所示，上图中有两个全连接层，大小均为4096，还有一个概率输出层，大小为1000。如果我们考虑把这些一维向量看成是特征图，只不过特征图的size是1x1大小，而向量的长度为通道数(如下图所示)，那么实际上，全连接层也可以看作是卷积层。因此，基于这么考虑，可以将这些传统的CNN网络转为全卷积网络。



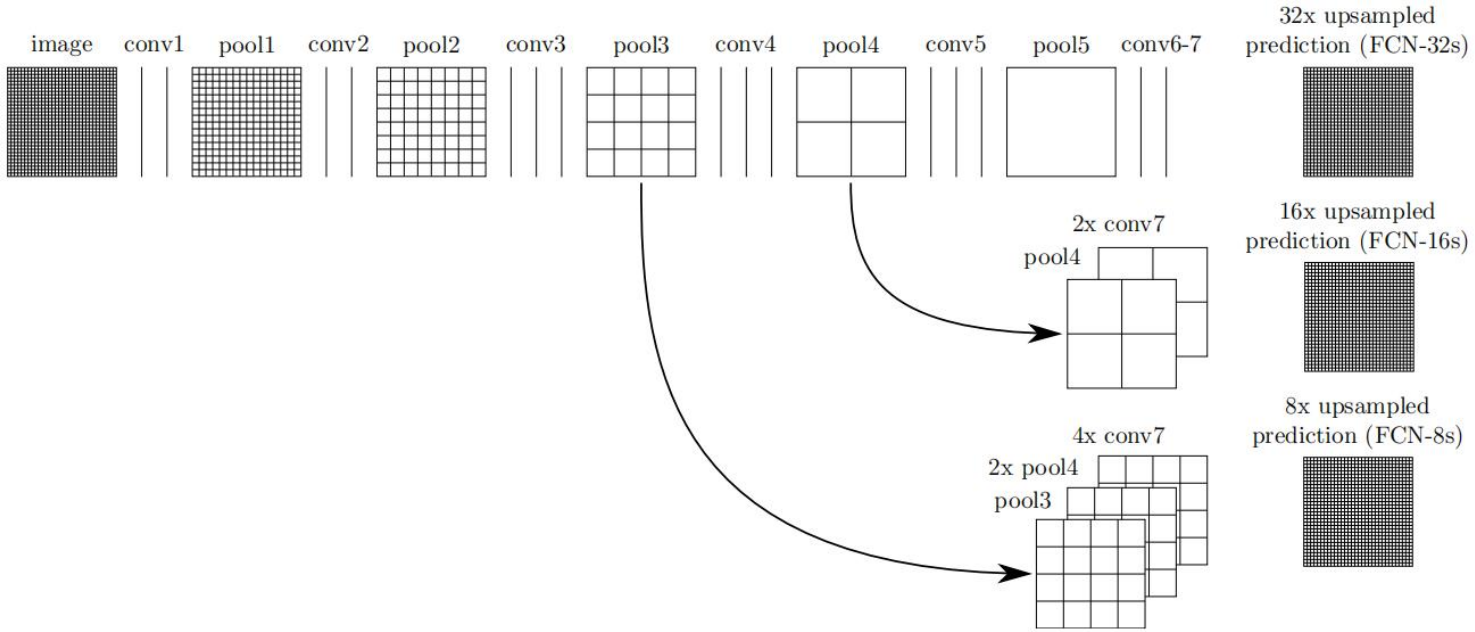
模型思想

FCN提出一种全新的跨层连接结构 (skip architecture)，将低层的目标位置信息强但语义信息弱的特征图和高层目标位置信息弱但语义信息强的特征图进行融合，以此来提升语义分割性能。



模型思想

- 对于FCN-32s(s是strides缩写)：直接对pool5进行32倍上采样获得32x upsampled feature，再对32x upsampled feature每个点做softmax prediction，获得32x upsampled prediction(即分割图)。
- 对于FCN-16s：首先对pool5进行2倍上采样获得2x upsampled feature（大小和pool4一样），然后将其与pool4逐点相加，然后对相加的feature进行16倍上采样，并softmax prediction，获得16x upsampled feature prediction。
- 对于FCN-8s：首先对pool5 feature进行2倍上采样获得2x upsampled feature，再把pool4 feature和2x upsampled feature逐点相加，又得到2x upsampled feature，继续pool3 feature和2x upsampled feature进行逐点相加。然后对相加的feature进行8倍上采样，并softmax prediction，获得8x upsampled feature prediction。



损失函数和训练过程

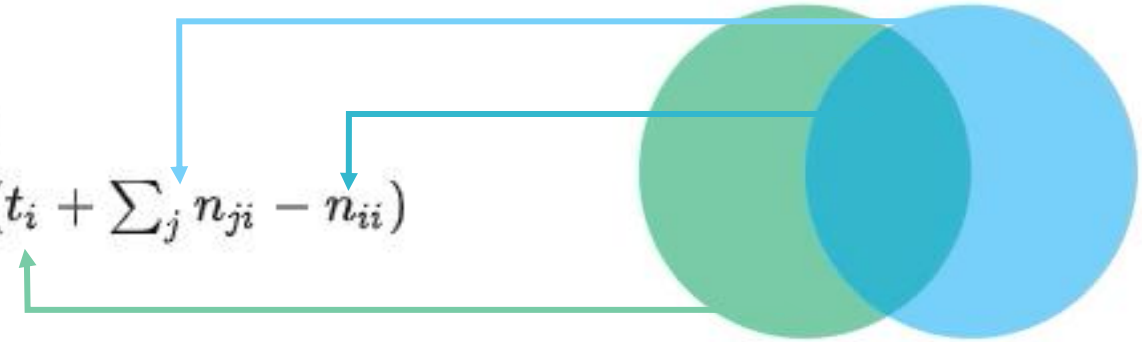
FCN采用的是简单的常规softmax分类损失，对于PASCAL VOC2012数据而言，其包括20个类别(加背景一共21个类别)，假设输入图片是W x H，那么最后输出的尺度是 W x H x 21，每个像素位置对应21个通道，softmax损失函数计算就是基于像素点的。

基础网络是AlexNet，VGG16，GoogLeNet，FCN论文主要以VGG16进行说明。具体实验细节是：
(1) 第一阶段。以经典的分类网络(imagnet)为初始化，最后两级是全连接，弃去不用；
(2)第二阶段。从特征小图(16x16x4096)预测分割小图(16x16x21)，之后直接升采样为大图。反卷积的步长为32，这个网络称为FCN-32s。
(3) 第三阶段。在第二次升采样前，把第4个pooling层的预测结果融合进来。使用跳级结构提升精确性，第二次反卷积步长为16，这个网络称为FCN-16s。
(4) 第四阶段。进一步融合了第3个pooling层的预测结果，第三次反卷积步长为8，记为FCN-8s。

模型评价指标

FCN使用了4种评价指标对模型进行全面评估，具体包括：

- pixel accuracy: $\sum_i n_{ii} / \sum_i t_i$
- mean accuracy: $(1/n_{cl}) \sum_i n_{ii}/t_i$
- mean IU: $(1/n_{cl}) \sum_i n_{ii}/(t_i + \sum_j n_{ji} - n_{ii})$
- frequency weighted IU : $(\sum_k t_k)^{-1} \sum_i t_i n_{ii}/(t_i + \sum_j n_{ji} - n_{ii})$



n_{ij} ：类别i被预测成类别j的像素个数

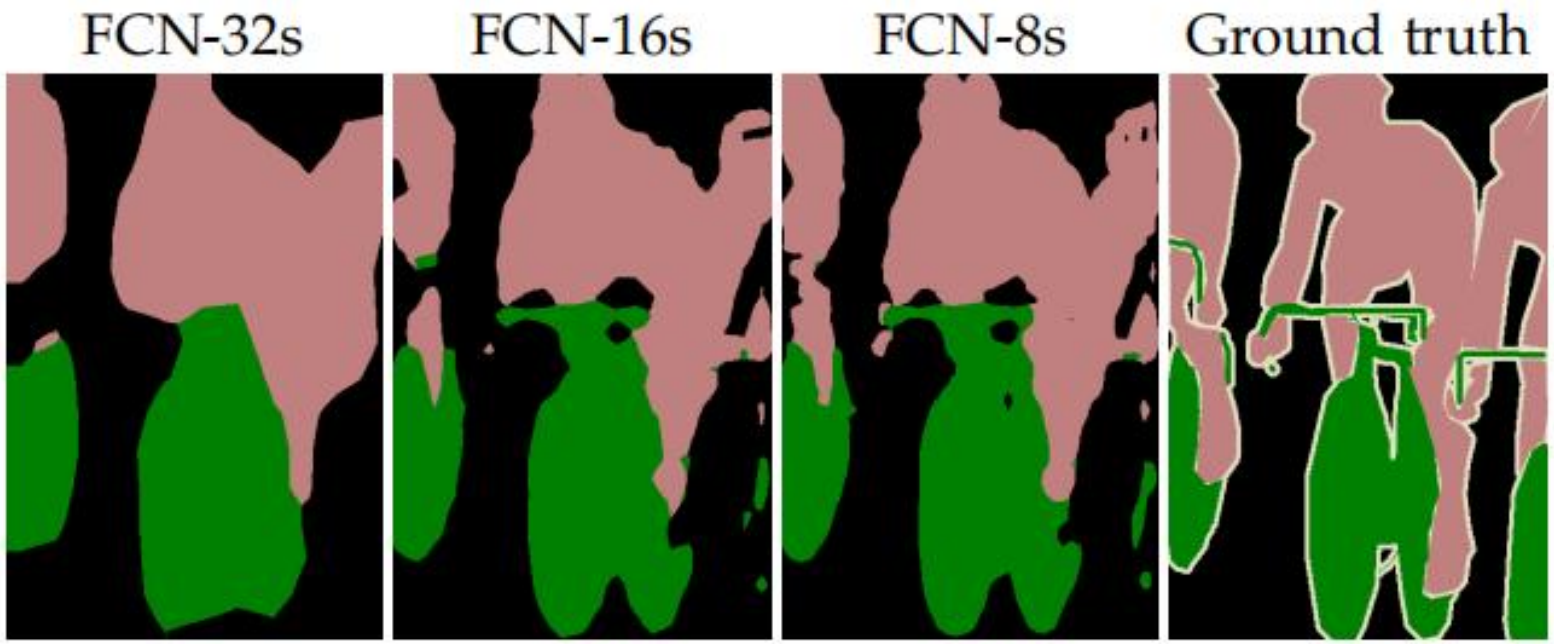
n_{cl} ：目标类别个数(包含背景)

$t_i = \sum_j n_{ij}$ ：目标类别i的总像素个数(真实标签)

pixel accuracy表示整张图片的像素中，预测正确像素个数所占比例；
mean accuracy是对pixel accuracy结果除以类别个数；
mean IU是平均像素交并比；
frequency weighted IU是mean IU的改进，其考虑了类别出现的频率，将其作为权重因子。

结果分析

对于FCN-32s，FCN-16s和FCN-8s的直观效果对比：



FCN 示例

- **Train and Deploy Fully Convolutional Networks for Semantic Segmentation**

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

This example shows how to train a fully convolutional semantic segmentation network.

A semantic segmentation network classifies every pixel in an image, resulting in an image that is segmented by class. Applications for semantic segmentation include road segmentation for autonomous driving and cancer cell segmentation for medical diagnosis.

To illustrate the training procedure, this example trains FCN-8s, one type of convolutional neural network (CNN) designed for semantic image segmentation.

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

This example uses the CamVid dataset from the University of Cambridge for training. This data set is a collection of images containing street-level views obtained while driving. The data set provides pixel-level labels for 32 semantic classes including car, pedestrian, and road.

```
imageURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/files/701_StillsRaw_full.zip';
labelURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/LabeledApproved_full.zip';
outputFolder = fullfile(pwd);
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder)
    labelsZip = fullfile(outputFolder,'labels.zip');   % 16 MB CamVid dataset labels
    imagesZip = fullfile(outputFolder,'images.zip');       % 557 MB CamVid dataset images
    websave(labelsZip, labelURL);
    unzip(labelsZip, fullfile(outputFolder,'labels'));
    websave(imagesZip, imageURL);
    unzip(imagesZip, fullfile(outputFolder,'images'));
end
```

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Load CamVid Images

Use imageDatastore to load CamVid images. The imageDatastore enables you to efficiently load a large collection of images onto a disk.

```
imgDir = fullfile(outputFolder,'images','701_StillsRaw_full');
imds = imageDatastore(imgDir);
```

Display one of the images.

```
I = readimage(imds,25);
I = histeq(I);
imshow(I)
```

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Load CamVid Pixel-Labeled Images

Use pixelLabelDatastore to load CamVid pixel label image data. A pixelLabelDatastore encapsulates the pixel label data and the label ID to a class name mapping.

Group the 32 original classes in CamVid to 11 classes. Specify these classes.

```
classes = [
    "Sky"
    "Building"
    "Pole"
    "Road"
    "Pavement"
    "Tree"
    "SignSymbol"
    "Fence"
    "Car"
    "Pedestrian"
    "Bicyclist"
    ];
```

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Load CamVid Pixel-Labeled Images

To reduce 32 classes into 11 classes, multiple classes from the original data set are grouped together. For example, "Car" is a combination of "Car", "SUVPickupTruck", "Truck_Bus", "Train", and "OtherMoving". Return the grouped label IDs by using the camvidPixelLabelIDs supporting function.

```
labelIDs = camvidPixelLabelIDs();
```

Use the classes and label IDs to create the pixelLabelDatastore.

```
labelDir = fullfile(outputFolder,'labels');
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);
```

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Load CamVid Pixel-Labeled Images

Read and display one of the pixel-labeled images by overlaying it on top of an image.

```
C = readimage(pxds,25);
cmap = camvidColorMap;
B = labeloverlay(I,C,'ColorMap',cmap);
imshow(B)
pixelLabelColorbar(cmap,classes);
```

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Analyze Data Set Statistics

To see the distribution of class labels in the CamVid dataset, use countEachLabel. This function counts the number of pixels by class label.

```
tbl = countEachLabel(pxds)
```

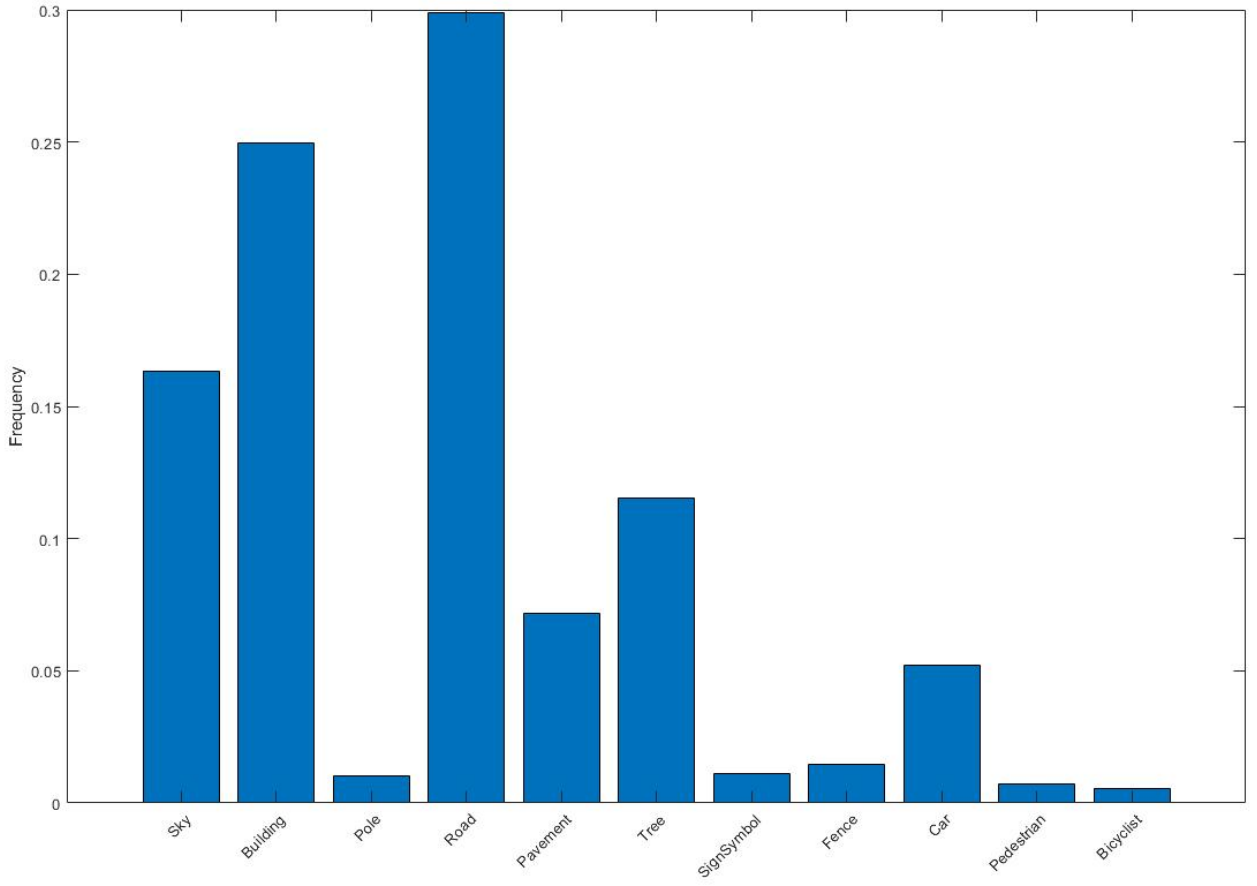
	Name	PixelCount	ImagePixelCount
1	'Sky'	76801167	483148800
2	'Building'	117373718	483148800
3	'Pole'	4798742	483148800
4	'Road'	140535728	484531200
5	'Pavement'	33614414	472089600
6	'Tree'	54258673	447897600
7	'SignSymbol'	5224247	468633600
8	'Fence'	6921061	251596800
9	'Car'	24436957	483148800
10	'Pedestrian'	3402909	444441600
11	'Bicyclist'	2591222	261964800

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Visualize the pixel counts by class.

```
frequency = tbl.PixelCount/sum(tbl.PixelCount);

bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
```



Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Resize CamVid Data

The images in the CamVid data set are 720-by-960. To reduce training time and memory usage, resize the images and pixel label images to 360-by-480 by using the resizeCamVidImages and resizeCamVidPixelLabels supporting functions.

```
imageFolder = fullfile(outputFolder,'imagesResized',filesep);
imds = resizeCamVidImages(imds,imageFolder);

labelFolder = fullfile(outputFolder,'labelsResized',filesep);
pxds = resizeCamVidPixelLabels(pxds,labelFolder);
```

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Prepare Training and Test Sets

Network is trained by using 60% of the images from the dataset. The rest of the images are used for testing. The following code randomly splits the image and pixel label data into a training set and a test set.

```
[imdsTrain,imdsTest,pxdsTrain,pxdsTest] =
partitionCamVidData(imds,pxds);
```

The 60/40 split results in the following number of training and test images:

```
numTrainingImages = numel(imdsTrain.Files)   % 421
numTestingImages = numel(imdsTest.Files)     % 280
```

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Create Network

Use fcnLayers to create fully convolutional network layers initialized by using VGG-16 weights. The fcnLayers function performs the network transformations to transfer the weights from VGG-16 and adds the additional layers required for semantic segmentation. The output of the fcnLayers function is a LayerGraph object representing FCN. A LayerGraph object encapsulates the network layers and the connections between the layers.

```
imageSize = [360 480];
numClasses = numel(classes);
lgraph = fcnLayers(imageSize,numClasses);
```

The image size is selected based on the size of the images in the dataset. The number of classes is selected based on the classes in CamVid.

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Balance Classes by Using Class Weighting

The classes in CamVid are not balanced. To improve training, you can use the pixel label counts computed earlier by the countEachLabel function and calculate the median frequency class weights.

```
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq;
```

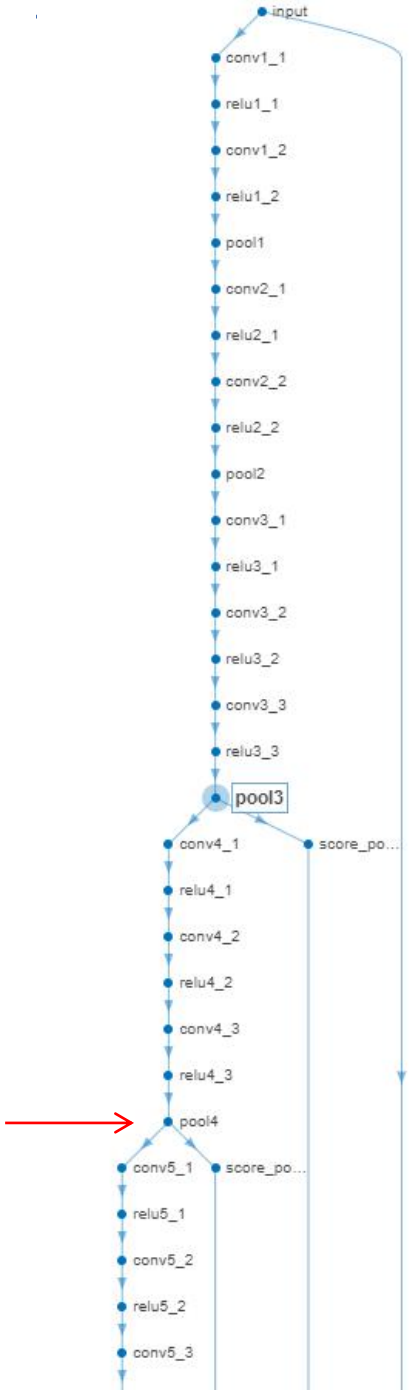
Specify the class weights by using a pixelClassificationLayer.

```
pxLayer =
pixelClassificationLayer('Name','labels','Classes',tbl.Name,'ClassWeights',classWeights)
```

Balance Classes by Using Class Weighting

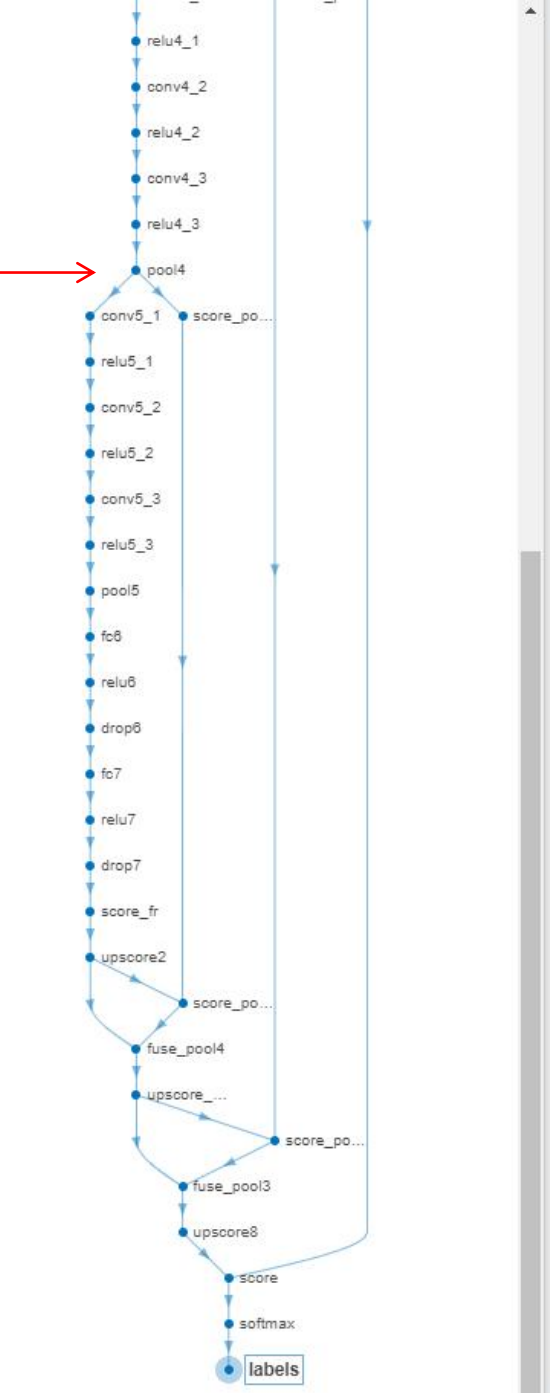
Update the network that has the new pixelClassificationLayer by removing the current pixelClassificationLayer and adding the new layer. The current pixelClassificationLayer is named 'pixelLabels'. Remove it by using the removeLayers function, add the new one by using the addLayers function, and connect the new layer to the rest of the network by using the connectLayers function.

```
lgraph = removeLayers(lgraph,'pixelLabels');
lgraph = addLayers(lgraph, pxLayer);
lgraph = connectLayers(lgraph,'softmax','labels');
```



ANALYSIS RESULT

	Name	Type	Activations	Learnable Properties
1	input 360×480×3 images with 'zerocenter' normalization	Image Input	360(S) × 480(S) × 3(C) × 1(B)	-
2	conv1_1 64 3×3×3 convolutions with stride [1 1] and padding [100 100 100 100]	Convolution	558(S) × 678(S) × 64(C) × 1(B)	Weights 3 × 3 × 3 × 64 Bias 1 × 1 × 64
3	relu1_1 ReLU	ReLU	558(S) × 678(S) × 64(C) × 1(B)	-
4	conv1_2 64 3×3×64 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	558(S) × 678(S) × 64(C) × 1(B)	Weights 3 × 3 × 64 × 64 Bias 1 × 1 × 64
5	relu1_2 ReLU	ReLU	558(S) × 678(S) × 64(C) × 1(B)	-
6	pool1 2×2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	279(S) × 339(S) × 64(C) × 1(B)	-
7	conv2_1 128 3×3×64 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	279(S) × 339(S) × 128(C) × 1(…	Weights 3 × 3 × 64 × 128 Bias 1 × 1 × 128
8	relu2_1 ReLU	ReLU	279(S) × 339(S) × 128(C) × 1(…	-
9	conv2_2 128 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	279(S) × 339(S) × 128(C) × 1(…	Weights 3 × 3 × 128 × 128 Bias 1 × 1 × 128
10	relu2_2 ReLU	ReLU	279(S) × 339(S) × 128(C) × 1(…	-
11	pool2 2×2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	139(S) × 169(S) × 128(C) × 1(…	-
12	conv3_1 256 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	139(S) × 169(S) × 256(C) × 1(…	Weights 3 × 3 × 128 × 256 Bias 1 × 1 × 256
13	relu3_1 ReLU	ReLU	139(S) × 169(S) × 256(C) × 1(…	-
14	conv3_2 256 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	139(S) × 169(S) × 256(C) × 1(…	Weights 3 × 3 × 256 × 256 Bias 1 × 1 × 256
15	relu3_2 ReLU	ReLU	139(S) × 169(S) × 256(C) × 1(…	-
16	conv3_3 256 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	139(S) × 169(S) × 256(C) × 1(…	Weights 3 × 3 × 256 × 256 Bias 1 × 1 × 256
17	relu3_3 ReLU	ReLU	139(S) × 169(S) × 256(C) × 1(…	-
18	pool3 2×2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	69(S) × 84(S) × 256(C) × 1(B)	-
19	conv4_1 512 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	69(S) × 84(S) × 512(C) × 1(B)	Weights 3 × 3 × 256 × 512 Bias 1 × 1 × 512
20	relu4_1 ReLU	ReLU	69(S) × 84(S) × 512(C) × 1(B)	-
21	conv4_2 512 3×3×512 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	69(S) × 84(S) × 512(C) × 1(B)	Weights 3 × 3 × 512 × 512 Bias 1 × 1 × 512
22	relu4_2 ReLU	ReLU	69(S) × 84(S) × 512(C) × 1(B)	-
23	conv4_3 512 3×3×512 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	69(S) × 84(S) × 512(C) × 1(B)	Weights 3 × 3 × 512 × 512 Bias 1 × 1 × 512
24	relu4_3 ReLU	ReLU	69(S) × 84(S) × 512(C) × 1(B)	-



ANALYSIS RESULT

	Name	Type	Activations	Learnable Properties
28	conv5_2 512 3×3×512 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	34(S) × 42(S) × 512(C) × 1(B)	Weights 3 × 3 × 512 × 512 Bias 1 × 1 × 512
29	relu5_2 ReLU	ReLU	34(S) × 42(S) × 512(C) × 1(B)	-
30	conv5_3 512 3×3×512 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	34(S) × 42(S) × 512(C) × 1(B)	Weights 3 × 3 × 512 × 512 Bias 1 × 1 × 512
31	relu5_3 ReLU	ReLU	34(S) × 42(S) × 512(C) × 1(B)	-
32	pool5 2×2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	17(S) × 21(S) × 512(C) × 1(B)	-
33	fc6 4096 7×7×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	11(S) × 15(S) × 4096(C) × 1(B)	Weigh… 7 × 7 × 512 × 40… Bias 1 × 1 × 4096
34	relu6 ReLU	ReLU	11(S) × 15(S) × 4096(C) × 1(B)	-
35	drop6 50% dropout	Dropout	11(S) × 15(S) × 4096(C) × 1(B)	-
36	fc7 4096 1×1×4096 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	11(S) × 15(S) × 4096(C) × 1(B)	Weigh… 1 × 1 × 4096 × 40… Bias 1 × 1 × 4096
37	relu7 ReLU	ReLU	11(S) × 15(S) × 4096(C) × 1(B)	-
38	drop7 50% dropout	Dropout	11(S) × 15(S) × 4096(C) × 1(B)	-
39	score_fr 11 1×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	11(S) × 15(S) × 11(C) × 1(B)	Weights 1 × 1 × 4096 × 11 Bias 1 × 1 × 11
40	upscore2 11 4×4×11 transposed convolutions with stride [2 2] and cropping [0 0 0 0]	Transposed Convol…	24(S) × 32(S) × 11(C) × 1(B)	Weights 4 × 4 × 11 × 11 Bias 1 × 1 × 11
41	score_pool4 11 1×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	34(S) × 42(S) × 11(C) × 1(B)	Weights 1 × 1 × 512 × 11 Bias 1 × 1 × 11
42	score_pool4c center crop	Crop 2D	24(S) × 32(S) × 11(C) × 1(B)	-
43	fuse_pool4 Element-wise addition of 2 inputs	Addition	24(S) × 32(S) × 11(C) × 1(B)	-
44	upscore_pool4 11 4×4×11 transposed convolutions with stride [2 2] and cropping [0 0 0 0]	Transposed Convol…	50(S) × 66(S) × 11(C) × 1(B)	Weights 4 × 4 × 11 × 11 Bias 1 × 1 × 11
45	score_pool3 11 1×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	69(S) × 84(S) × 11(C) × 1(B)	Weights 1 × 1 × 256 × 11 Bias 1 × 1 × 11
46	score_pool3c center crop	Crop 2D	50(S) × 66(S) × 11(C) × 1(B)	-
47	fuse_pool3 Element-wise addition of 2 inputs	Addition	50(S) × 66(S) × 11(C) × 1(B)	-
48	upscore8 11 16×16×11 transposed convolutions with stride [8 8] and cropping [0 0 0 0]	Transposed Convol…	408(S) × 536(S) × 11(C) × 1(B)	Weights 16 × 16 × 11 × 11 Bias 1 × 1 × 11
49	score center crop	Crop 2D	360(S) × 480(S) × 11(C) × 1(B)	-
50	softmax softmax	Softmax	360(S) × 480(S) × 11(C) × 1(B)	-
51	labels Class weighted cross-entropy loss with 'Sky', 'Building', and 9 other classes	Pixel Classification …	360(S) × 480(S) × 11(C) × 1(B)	-

Select Training Options

The optimization algorithm for training is Adam, which is derived from adaptive moment estimation. Use the trainingOptions function to specify the hyperparameters used for Adam.

```
options = trainingOptions('adam', ...
    'InitialLearnRate',1e-3, ...
    'MaxEpochs',100, ...
    'MiniBatchSize',4, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2);
```

'CheckpointPath' is set to a temporary location. This name-value pair enables the saving of network checkpoints at the end of every training epoch. If training is interrupted due to a system failure or power outage, you can resume training from the saved checkpoint.

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Data Augmentation

Data augmentation is used to improve network accuracy by randomly transforming the original data during training. By using data augmentation, you can add more variety to the training data without increasing the number of labeled training samples. To apply the same random transformation to both image and pixel label data use datastore combine and transform. First, combine imdsTrain and pxdsTrain.

```
dsTrain = combine(imdsTrain, pxdsTrain);
```

Next, use datastore transform to apply the desired data augmentation defined in the supporting function augmentImageAndLabel. Here, random left/right reflection and random X/Y translation of +/- 10 pixels is used for data augmentation.

```
xTrans = [-10 10];
yTrans = [-10 10];
dsTrain = transform(dsTrain, @(data)augmentImageAndLabel(data,xTrans,yTrans));
```

Note that data augmentation is not applied to the test and validation data. Ideally, test and validation data should be representative of the original data and is left unmodified for unbiased evaluation.

Train and Deploy Fully Convolutional Networks for Semantic Segmentation

Start Training

Start training using trainNetwork if the doTraining flag is true. Otherwise, load a pretrained network.

The training was verified on an NVIDIA™ Titan Xp with 12 GB of GPU memory. If your GPU has less memory, you might run out of memory. If you do not have enough memory in your system, try lowering the MiniBatchSize property in trainingOptions to 1. Training this network takes about 5 hours or longer depending on your GPU hardware.

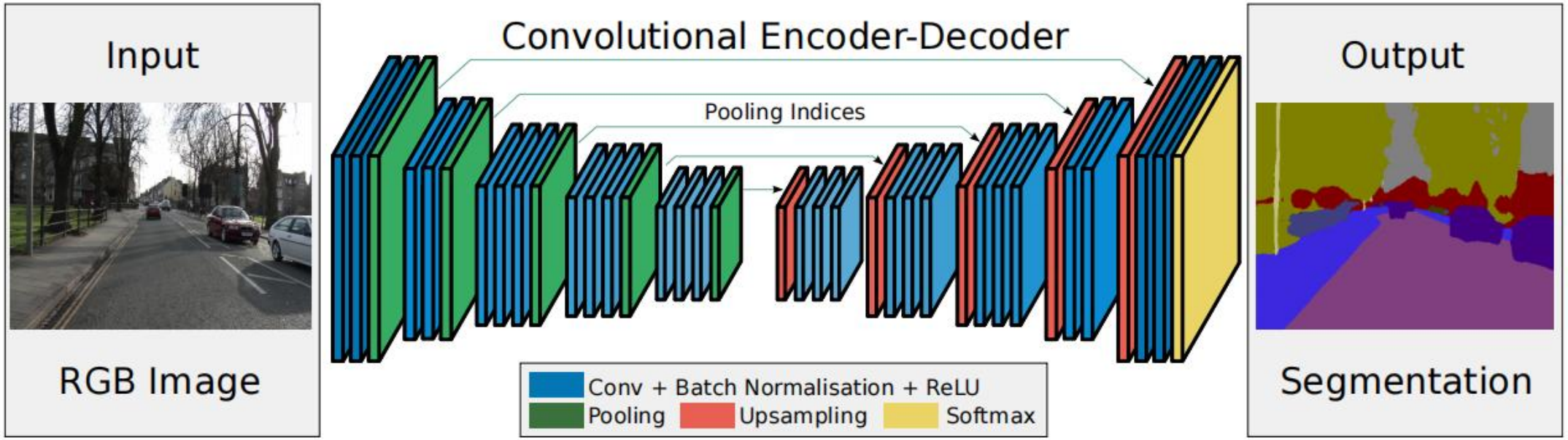
```
doTraining = false;
if doTraining
    [net, info] = trainNetwork(dsTrain,lgraph,options);
    save('FCN8sCamVid.mat','net');
end
```

Save the DAG network object as a MAT-file named FCN8sCamVid.mat.

SegNet

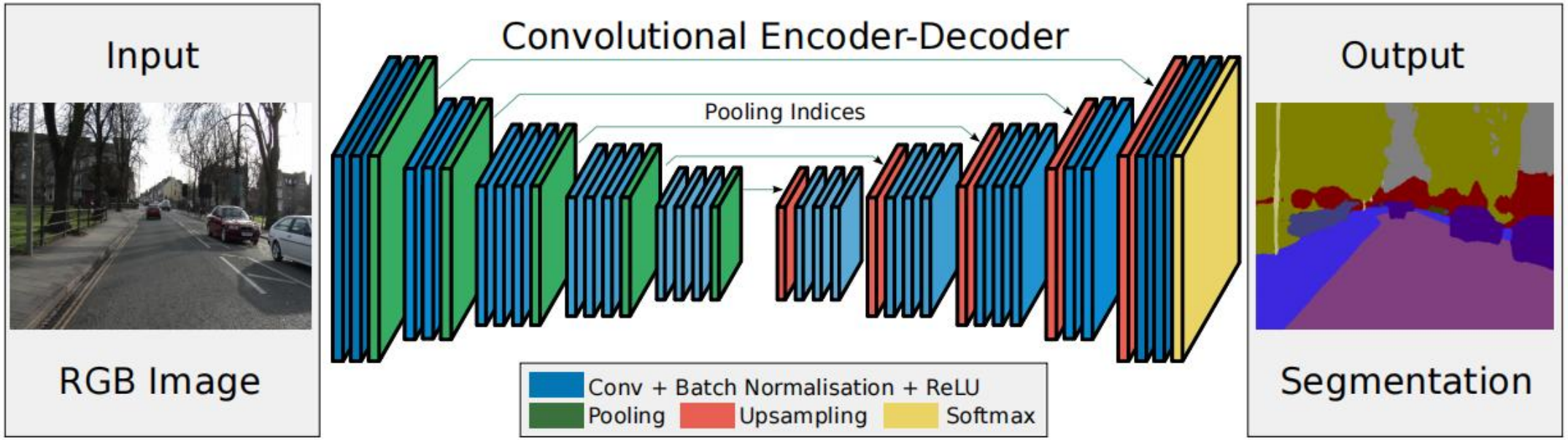
2015 年，剑桥大学的 Badrinarayanan 等人提出了 SegNet 语义分割模型，沿用了 FCN 的分割思想，是一种有监督的编解码结构的对称网络模型。

SegNet 模型可以处理任意尺寸的输入图像，编码部分通过最大池化逐阶段缩小输入图像的尺寸和降低参数量，并且记录图像中的池化索引位置。为了使输入图像与输出图像分辨率一致，解码部分通过上采样逐阶段恢复图像信息，最后通过Softmax 分类器输出语义分割结果。



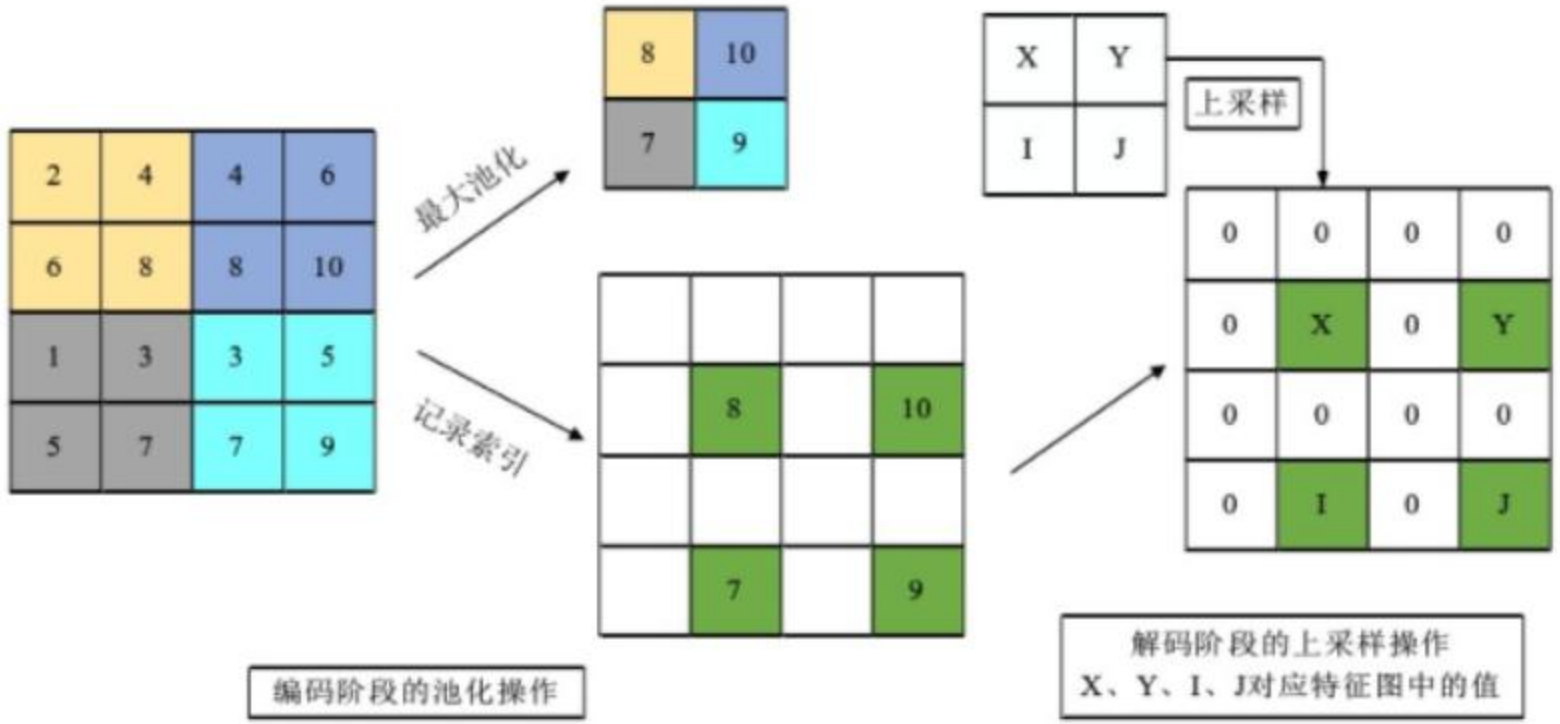
SegNet

SegNet 采用去除了全连接层的 VGG16 作为编码部分来实现特征提取，编码部分包含 5 个阶段共 13 个卷积层，每个阶段包含卷积层和池化层，每经过一个阶段将特征图尺寸减半。解码部分与编码部分对称，也包含 5 个阶段共 13 个卷积层，每个阶段包含上采样层和卷积层，每经过一个阶段将特征图尺寸翻倍。模型所有卷积层后接 BN 层和 ReLU 激活函数，使网络可以更快收敛，提升网络的非线性表达能力。



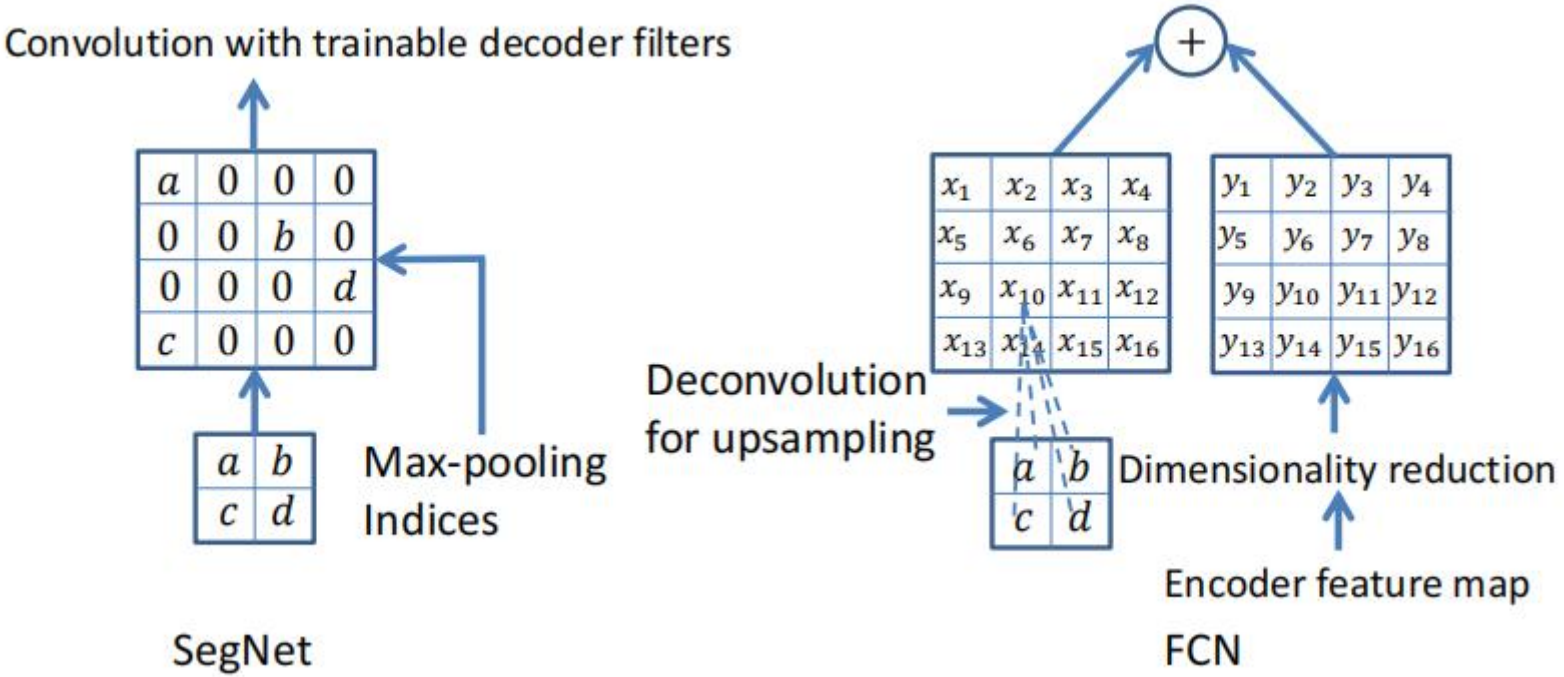
SegNet

SegNet 最大的优势就是利用最大池化的位置索引进行上采样，SegNet 在编码器进行最大池化时记录最大值所在的位置索引，在解码器中利用对应的池化索引来辅助上采样，不仅能够减少计算量，还能更好地保留图像边缘信息，使上采样不需要训练学习，大大提升了网络的运行速度。

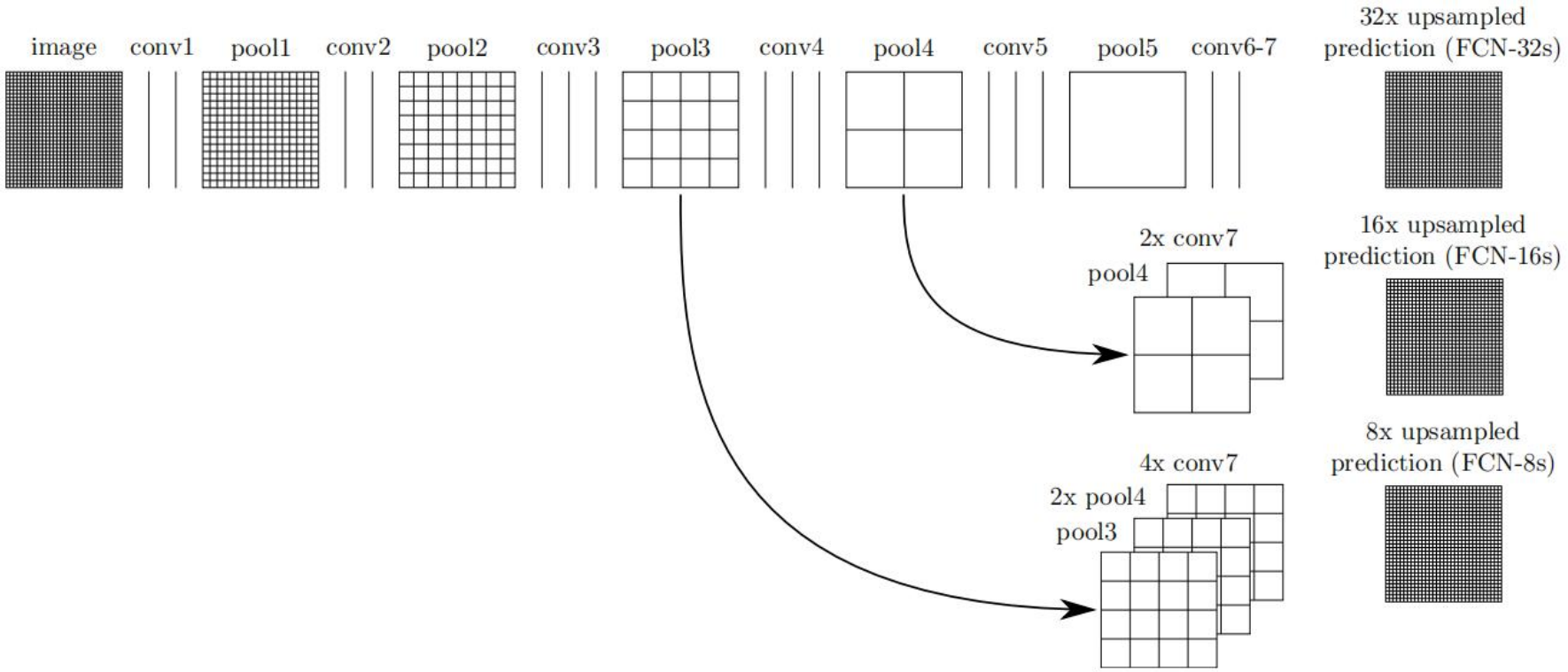


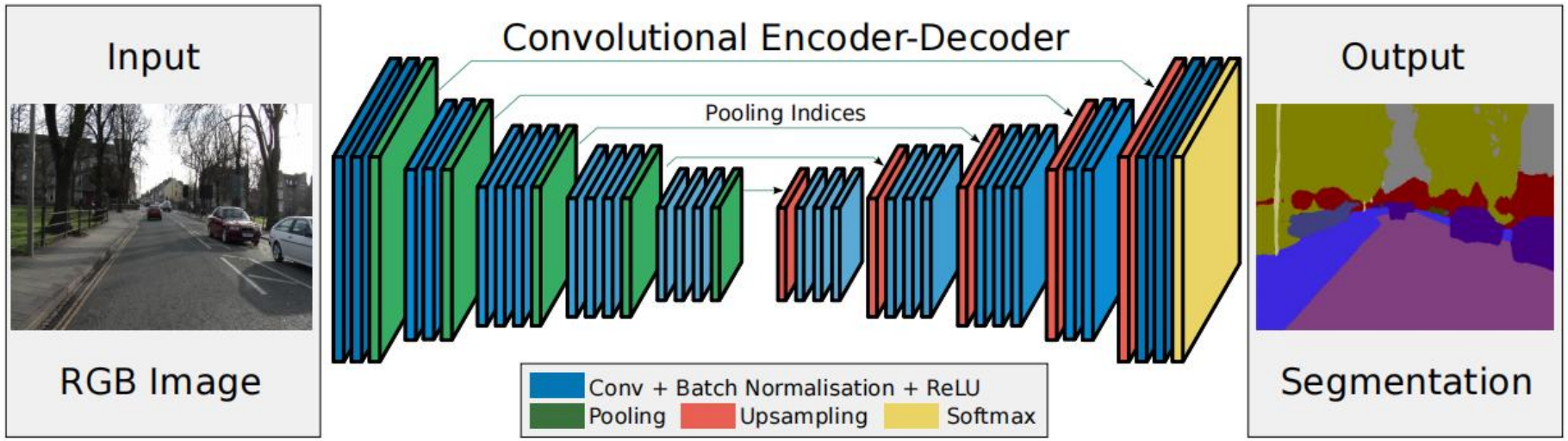
SegNet

SegNet和FCN解码器的对比：a, b, c, d分别对应特征图中的值。SegNet使用最大池索引对特征图进行上采样(不学习)，并与可训练的解码器滤波器组进行卷积。FCN通过学习反卷积来提高样本输入特征映射并添加相应的编码器特征映射以产生解码器输出。编码器特征映射来自于相应编码器中的最大池化层。FCN中没有可训练的解码器过滤器。



SegNet vs FCN





SegNet 示例

- **Train SegNet**
- **Create SegNet With Custom Encoder-Decoder Depth**

Train SegNet

Load training images and pixel labels.

```
dataSetDir = fullfile(toolboxdir('vision'),'visiondata','triangleImages');
imageDir = fullfile(dataSetDir,'trainingImages');
labelDir = fullfile(dataSetDir,'trainingLabels');
```

Create an image datastore holding the training images.

```
imds = imageDatastore(imageDir);
```

Train SegNet

Define the class names and their associated label IDs.

```
classNames = ["triangle", "background"];
labelIDs   = [255 0];
```

Create a pixel label datastore holding the ground truth pixel labels for the training images.

```
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
```

Train SegNet

Create SegNet layers.

```
imageSize = [32 32];
numClasses = 2;
lgraph = segnetLayers(imageSize,numClasses,2)
```

Combine image and pixel label data for training a semantic segmentation network.

```
ds = combine(imds,pxds);
```

ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope…
			size 1(S) × 4(S) × 1(S) × 32(S) ×…	
9	encoder2_conv1 64 3×3×64 convolutions with stride [1 1] …	Convolution	16(S) × 16(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
10	encoder2_bn_1 Batch normalization	Batch Normalization	16(S) × 16(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
11	encoder2_relu_1 ReLU	ReLU	16(S) × 16(S) × 64(C) × 1(B)	-
12	encoder2_conv2 64 3×3×64 convolutions with stride [1 1] …	Convolution	16(S) × 16(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
13	encoder2_bn_2 Batch normalization	Batch Normalization	16(S) × 16(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
14	encoder2_relu_2 ReLU	ReLU	16(S) × 16(S) × 64(C) × 1(B)	-
15	encoder2_maxpool 2×2 max pooling with stride [2 2] and pa…	Max Pooling	out 8(S) × 8(S) × 64(C) × 1(B) ind… 4096(S) × 1(S) × 1(C) × 1(B) size 1(S) × 4(S) × 1(S) × 16(S) ×…	-
16	decoder2_unpool Max Unpooling	Max Unpooling	16(S) × 16(S) × 64(C) × 1(B)	-
17	decoder2_conv2 64 3×3×64 convolutions with stride [1 1] …	Convolution	16(S) × 16(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
18	decoder2_bn_2 Batch normalization	Batch Normalization	16(S) × 16(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
19	decoder2_relu_2 ReLU	ReLU	16(S) × 16(S) × 64(C) × 1(B)	-
20	decoder2_conv1 64 3×3×64 convolutions with stride [1 1] …	Convolution	16(S) × 16(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
21	decoder2_bn_1 Batch normalization	Batch Normalization	16(S) × 16(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
22	decoder2_relu_1 ReLU	ReLU	16(S) × 16(S) × 64(C) × 1(B)	-
23	decoder1_unpool Max Unpooling	Max Unpooling	32(S) × 32(S) × 64(C) × 1(B)	-
24	decoder1_conv2 64 3×3×64 convolutions with stride [1 1] …	Convolution	32(S) × 32(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
25	decoder1_bn_2 Batch normalization	Batch Normalization	32(S) × 32(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
26	decoder1_relu_2 ReLU	ReLU	32(S) × 32(S) × 64(C) × 1(B)	-
27	decoder1_conv1 2 3×3×64 convolutions with stride [1 1] a…	Convolution	32(S) × 32(S) × 2(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 2
28	decoder1_bn_1 Batch normalization	Batch Normalization	32(S) × 32(S) × 2(C) × 1(B)	Offset 1 × 1 × 2 Scale 1 × 1 × 2
29	decoder1_relu_1 ReLU	ReLU	32(S) × 32(S) × 2(C) × 1(B)	-
30	softmax softmax	Softmax	32(S) × 32(S) × 2(C) × 1(B)	-
31	pixelLabels Cross-entropy loss	Pixel Classification …	32(S) × 32(S) × 2(C) × 1(B)	-

Train SegNet

Set up training options.

```
options = trainingOptions('sgdm','InitialLearnRate',1e-3, ...
      'MaxEpochs',20,'VerboseFrequency',10);
```

Train the network.

```
net = trainNetwork(ds,lgraph,options)
```

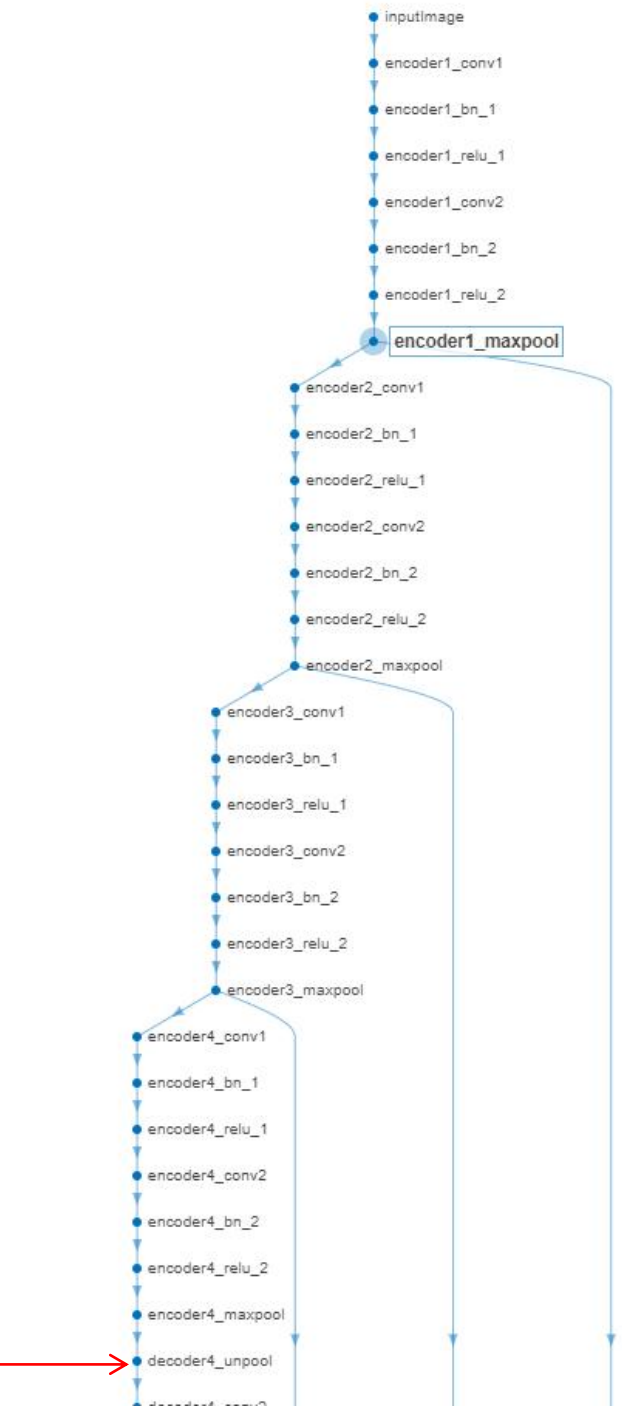
```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:08 |       52.15% |       0.7380 |          0.0010 |
|      10 |          10 |       00:00:20 |       63.51% |       0.7131 |          0.0010 |
|      20 |          20 |       00:00:33 |       71.58% |       0.6722 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.
net =
  DAGNetwork with properties:

         Layers: [31×1 nnet.cnn.layer.Layer]
    Connections: [34×2 table]
     InputNames: {'inputImage'}
    OutputNames: {'pixelLabels'}
```

Create SegNet With Custom Encoder-Decoder Depth

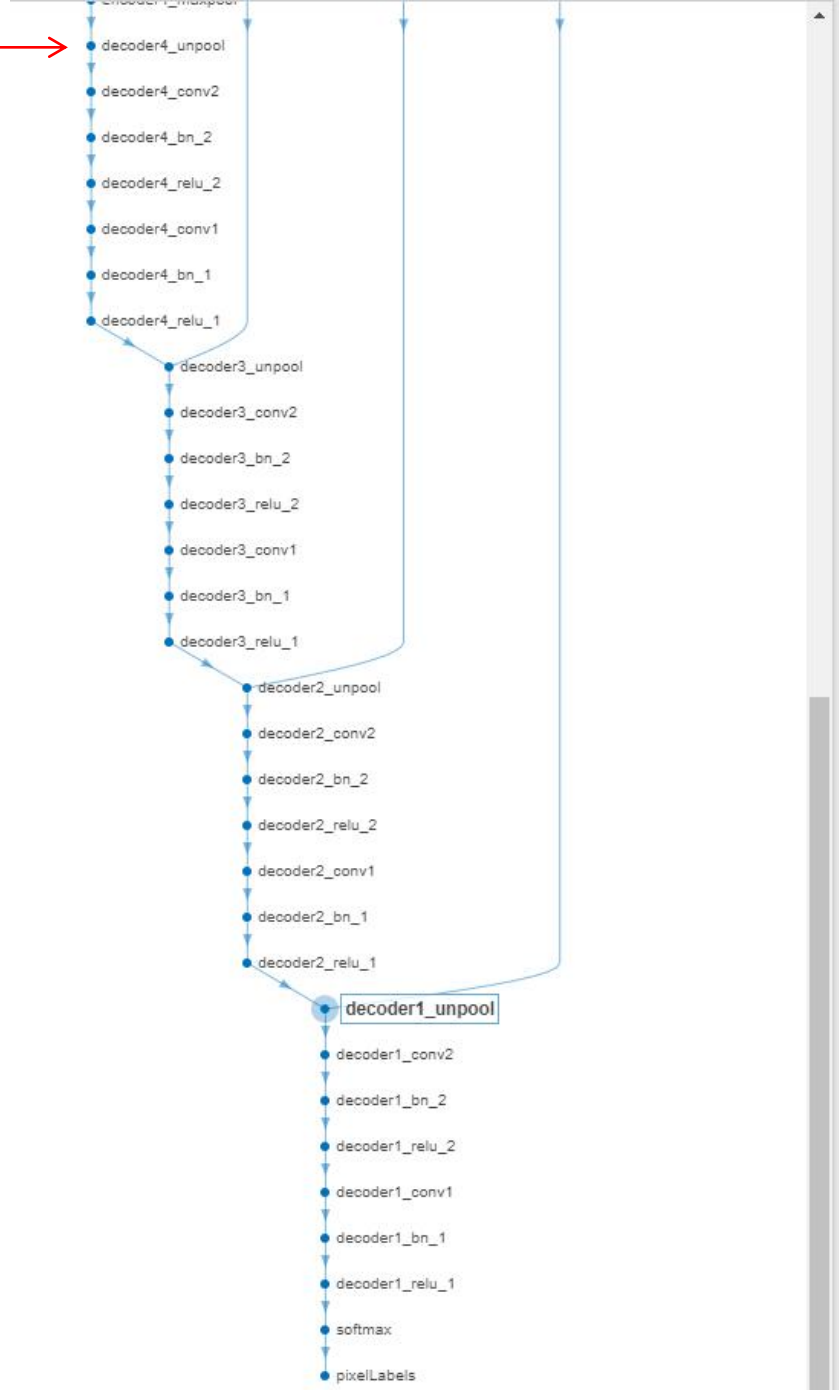
Create SegNet layers with an encoder/decoder depth of 4.

```
imageSize = [480 640 3];
numClasses = 5;
encoderDepth = 4;
lgraph = segnetLayers(imageSize,numClasses,encoderDepth)
```



ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
1	inputImage 480×640×3 images with 'zerocenter' nor...	Image Input	480(S) × 640(S) × 3(C) × 1(B)	-
2	encoder1_conv1 64 3×3×3 convolutions with stride [1 1] a...	Convolution	480(S) × 640(S) × 64(C) × 1(B)	Weig… 3 × 3 × 3 … Bias 1 × 1 × 64
3	encoder1_bn_1 Batch normalization	Batch Normalization	480(S) × 640(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
4	encoder1_relu_1 ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
5	encoder1_conv2 64 3×3×64 convolutions with stride [1 1] ...	Convolution	480(S) × 640(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
6	encoder1_bn_2 Batch normalization	Batch Normalization	480(S) × 640(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
7	encoder1_relu_2 ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
8	encoder1_maxpool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	out 240(S) × 320(S) × 64(C) × 1(… ind… 4915200(S) × 1(S) × 1(C) × 1… size 1(S) × 4(S) × 1(S) × 480(S) …	-
9	encoder2_conv1 64 3×3×64 convolutions with stride [1 1] ...	Convolution	240(S) × 320(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
10	encoder2_bn_1 Batch normalization	Batch Normalization	240(S) × 320(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
11	encoder2_relu_1 ReLU	ReLU	240(S) × 320(S) × 64(C) × 1(B)	-
12	encoder2_conv2 64 3×3×64 convolutions with stride [1 1] ...	Convolution	240(S) × 320(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
13	encoder2_bn_2 Batch normalization	Batch Normalization	240(S) × 320(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
14	encoder2_relu_2 ReLU	ReLU	240(S) × 320(S) × 64(C) × 1(B)	-
15	encoder2_maxpool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	out 120(S) × 160(S) × 64(C) × 1(… ind… 1228800(S) × 1(S) × 1(C) × 1… size 1(S) × 4(S) × 1(S) × 240(S) …	-
16	encoder3_conv1 64 3×3×64 convolutions with stride [1 1] ...	Convolution	120(S) × 160(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
17	encoder3_bn_1 Batch normalization	Batch Normalization	120(S) × 160(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
18	encoder3_relu_1 ReLU	ReLU	120(S) × 160(S) × 64(C) × 1(B)	-
19	encoder3_conv2 64 3×3×64 convolutions with stride [1 1] ...	Convolution	120(S) × 160(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
20	encoder3_bn_2 Batch normalization	Batch Normalization	120(S) × 160(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
21	encoder3_relu_2 ReLU	ReLU	120(S) × 160(S) × 64(C) × 1(B)	-
22	encoder3_maxpool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	out 60(S) × 80(S) × 64(C) × 1(B) ind… 307200(S) × 1(S) × 1(C) × 1(… size 1(S) × 4(S) × 1(S) × 120(S) …	-
23	encoder4_conv1 64 3×3×64 convolutions with stride [1 1] ...	Convolution	60(S) × 80(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64



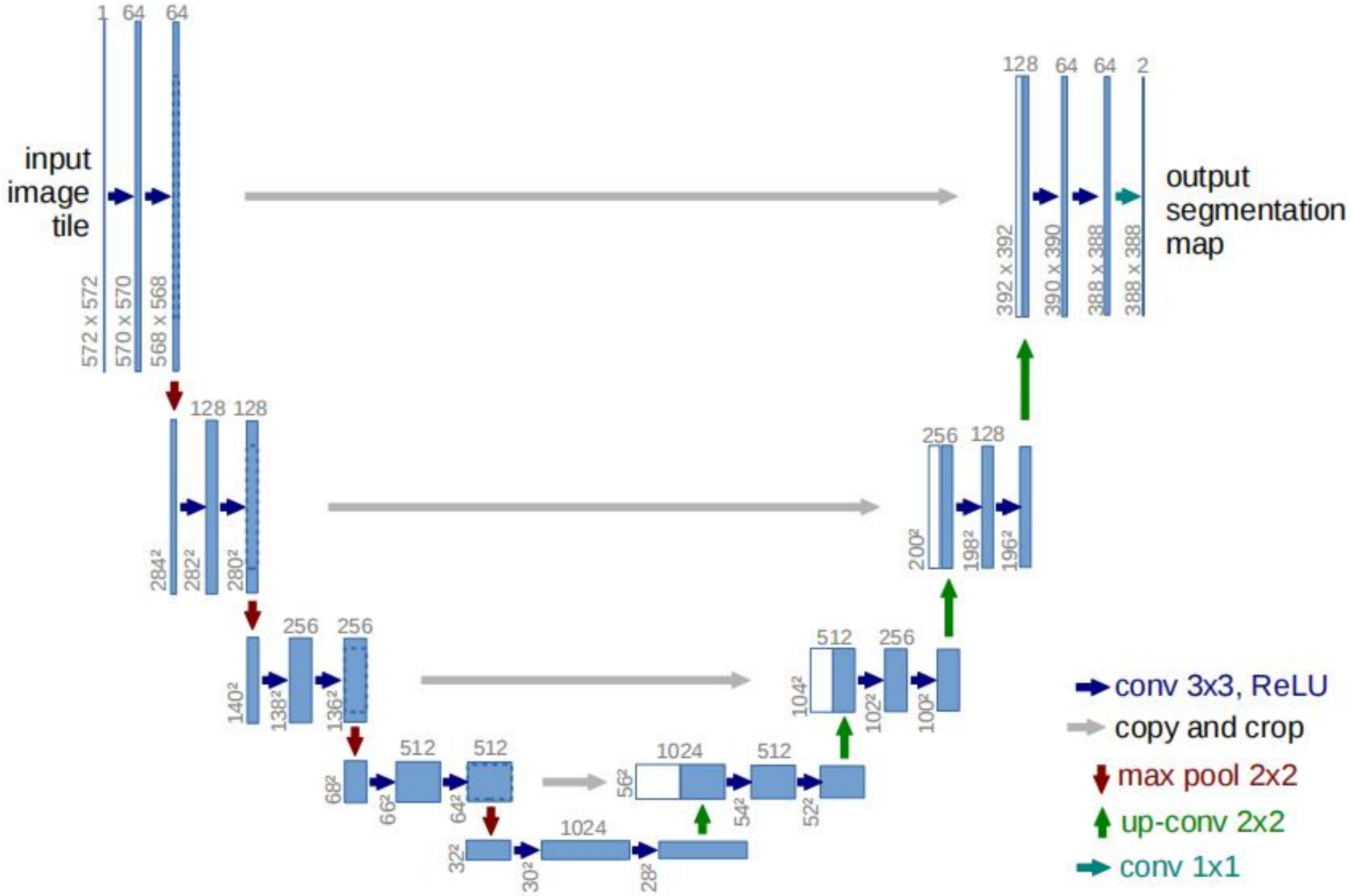
ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
36	decoder4_relu_1 ReLU	ReLU	60(S) × 80(S) × 64(C) × 1(B)	-
37	decoder3_unpool Max Unpooling	Max Unpooling	120(S) × 160(S) × 64(C) × 1(B)	-
38	decoder3_conv2 64 3×3×64 convolutions with stride [1 1] ...	Convolution	120(S) × 160(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
39	decoder3_bn_2 Batch normalization	Batch Normalization	120(S) × 160(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
40	decoder3_relu_2 ReLU	ReLU	120(S) × 160(S) × 64(C) × 1(B)	-
41	decoder3_conv1 64 3×3×64 convolutions with stride [1 1] ...	Convolution	120(S) × 160(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
42	decoder3_bn_1 Batch normalization	Batch Normalization	120(S) × 160(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
43	decoder3_relu_1 ReLU	ReLU	120(S) × 160(S) × 64(C) × 1(B)	-
44	decoder2_unpool Max Unpooling	Max Unpooling	240(S) × 320(S) × 64(C) × 1(B)	-
45	decoder2_conv2 64 3×3×64 convolutions with stride [1 1] ...	Convolution	240(S) × 320(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
46	decoder2_bn_2 Batch normalization	Batch Normalization	240(S) × 320(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
47	decoder2_relu_2 ReLU	ReLU	240(S) × 320(S) × 64(C) × 1(B)	-
48	decoder2_conv1 64 3×3×64 convolutions with stride [1 1] ...	Convolution	240(S) × 320(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
49	decoder2_bn_1 Batch normalization	Batch Normalization	240(S) × 320(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
50	decoder2_relu_1 ReLU	ReLU	240(S) × 320(S) × 64(C) × 1(B)	-
51	decoder1_unpool Max Unpooling	Max Unpooling	480(S) × 640(S) × 64(C) × 1(B)	-
52	decoder1_conv2 64 3×3×64 convolutions with stride [1 1] ...	Convolution	480(S) × 640(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
53	decoder1_bn_2 Batch normalization	Batch Normalization	480(S) × 640(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
54	decoder1_relu_2 ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
55	decoder1_conv1 5 3×3×64 convolutions with stride [1 1] a...	Convolution	480(S) × 640(S) × 5(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 5
56	decoder1_bn_1 Batch normalization	Batch Normalization	480(S) × 640(S) × 5(C) × 1(B)	Offset 1 × 1 × 5 Scale 1 × 1 × 5
57	decoder1_relu_1 ReLU	ReLU	480(S) × 640(S) × 5(C) × 1(B)	-
58	softmax softmax	Softmax	480(S) × 640(S) × 5(C) × 1(B)	-
59	pixelLabels Cross-entropy loss	Pixel Classification ...	480(S) × 640(S) × 5(C) × 1(B)	-

U-Net

2015 年，德国弗赖堡大学的 Ronneberger 等人提出了 U-Net 语义分割模型，Unet 的初衷是为了解决生物医学图像方面（细胞分割）的问题，由于效果很好后来广泛的应用在语义分割的各个方向，比如卫星图像分割，工业瑕疵检测等。UNet遵循FCN的原理，并进行了相应的改进，使其适应小样本的简单分割问题。
Unet 跟 FCN 都是 Encoder-Decoder 结构，结构简单有效。Encoder 负责特征提取，可以将自己熟悉的各种特征提取网络放在这个位置。由于医学样本收集较为困难，为解决这个问题，U-Net应用了图像增强方法，在数据集有限情况下获得了不错的精度。

U-Net


input image tile
572 x 572
570 x 570
568 x 568
1 64 64
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
512
56²
54²
52²
256
104²
102²
100²
128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

U-Net

U-Net网络前半部分（左边）作用是特征提取，后半部分（右边）是上采样。是一种典型的 Encoder-Decoder 结构。因为此网络整体结构类似于大写的英文字母 U，故得名U-Net。U-Net 网络的特点是：U型网络结构和Skip Connection跳层连接，是一种对称的网络结构。

Skip Connection对应中间四条灰色的平行线，Skip Connection就是在上采样的过程中，融合下采样过过程中的特征图。Skip Connection用到的融合的操作也很简单，就是将特征图的通道进行叠加，俗称拼接（Concat）。

U-Net

对于特征图，一个大小为256*256*64的特征图，即特征图的宽为256，高为256，通道数为64。和一个大小为256*256*32的特征图进行拼接融合，就会得到一个大小为256*256*96的特征图。

在实际使用中，拼接融合的两个特征图的大小不一定相同，例如256*256*64的特征图和240*240*32的特征图进行拼接，有两种拼接办法：

第一种：将大256*256*64的特征图进行裁剪，裁剪为240*240*64的特征图，上下左右各舍弃8 pixel，裁剪后再进行拼接，得到240*240*96的特征图。

第二种：将小240*240*32的特征图进行padding操作，padding为256*256*32的特征图，上下左右各补8 pixel，padding后再进行拼接，得到256*256*96的特征图。

U-Net

U-Net的特点：

- 使用多个 pooling layer实现网络对图像特征的多尺度特征提取；
- 上采样部分融合特征提取部分的输出，这样做可以将多尺度特征融合在了一起；
- U-Net采用了与FCN不同的特征融合方式：拼接（Concat），U-Net采用将特征在通道维度拼接在一起，形成更厚的特征。而FCN融合时使用的对应点相加，并不形成更厚的特征。
- U-Net在得到融合特征后继续进行3层卷积得到输入，而FCN直接（通过转置卷积）上采样得到输出。
- U-Net网络依靠数据增强可以有效使用有限的标注数据，已成为大多做医疗影像语义分割任务的基线模型。

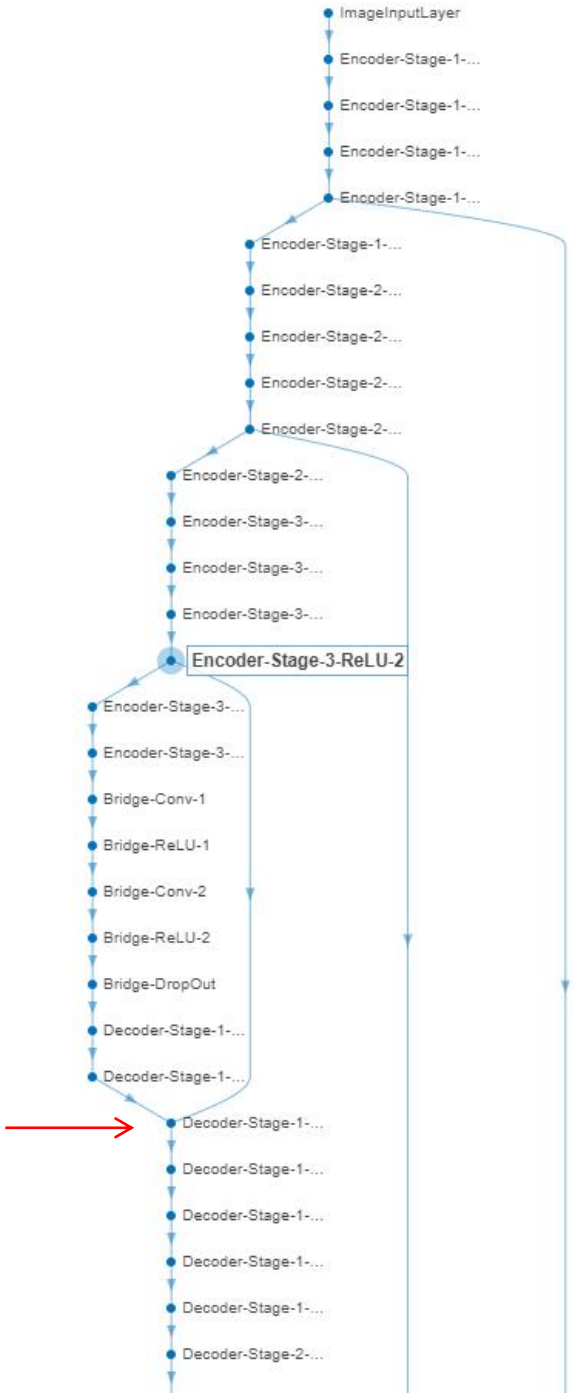
U-Net 示例

- **Create U-Net Network with Custom Encoder-Decoder Depth**
- **Train U-Net Network for Semantic Segmentation**

Create U-Net Network with Custom Encoder-Decoder Depth

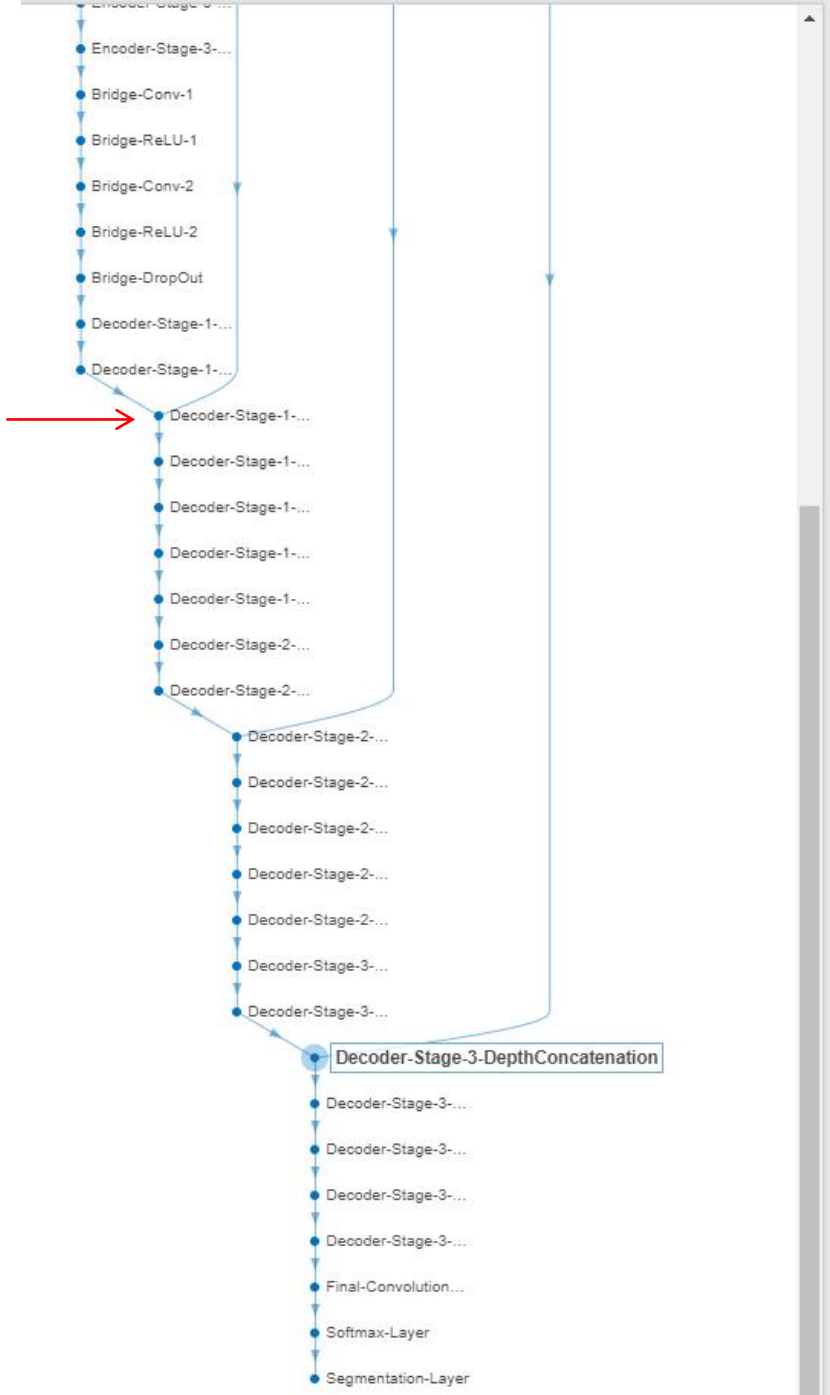
Create a U-Net network with an encoder-decoder depth of 3.

```
imageSize = [480 640 3];
numClasses = 5;
encoderDepth = 3;
lgraph = unetLayers(imageSize,numClasses,'EncoderDepth',encoderDepth)
```



ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
1	ImageInputLayer 480×640×3 images with 'zerocenter' nor...	Image Input	480(S) × 640(S) × 3(C) × 1(B)	-
2	Encoder-Stage-1-Conv-1 64 3×3 convolutions with stride [1 1] and...	Convolution	480(S) × 640(S) × 64(C) × 1(B)	Weig… 3 × 3 × 3 … Bias 1 × 1 × 64
3	Encoder-Stage-1-ReLU-1 ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
4	Encoder-Stage-1-Conv-2 64 3×3 convolutions with stride [1 1] and...	Convolution	480(S) × 640(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
5	Encoder-Stage-1-ReLU-2 ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
6	Encoder-Stage-1-MaxPool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	240(S) × 320(S) × 64(C) × 1(B)	-
7	Encoder-Stage-2-Conv-1 128 3×3 convolutions with stride [1 1] an...	Convolution	240(S) × 320(S) × 128(C) × 1(B)	Weig… 3 × 3 × 64 … Bias 1 × 1 × 128
8	Encoder-Stage-2-ReLU-1 ReLU	ReLU	240(S) × 320(S) × 128(C) × 1(B)	-
9	Encoder-Stage-2-Conv-2 128 3×3 convolutions with stride [1 1] an...	Convolution	240(S) × 320(S) × 128(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 128
10	Encoder-Stage-2-ReLU-2 ReLU	ReLU	240(S) × 320(S) × 128(C) × 1(B)	-
11	Encoder-Stage-2-MaxPool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	120(S) × 160(S) × 128(C) × 1(B)	-
12	Encoder-Stage-3-Conv-1 256 3×3 convolutions with stride [1 1] an...	Convolution	120(S) × 160(S) × 256(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 256
13	Encoder-Stage-3-ReLU-1 ReLU	ReLU	120(S) × 160(S) × 256(C) × 1(B)	-
14	Encoder-Stage-3-Conv-2 256 3×3 convolutions with stride [1 1] an...	Convolution	120(S) × 160(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
15	Encoder-Stage-3-ReLU-2 ReLU	ReLU	120(S) × 160(S) × 256(C) × 1(B)	-
16	Encoder-Stage-3-DropOut 50% dropout	Dropout	120(S) × 160(S) × 256(C) × 1(B)	-
17	Encoder-Stage-3-MaxPool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	60(S) × 80(S) × 256(C) × 1(B)	-
18	Bridge-Conv-1 512 3×3 convolutions with stride [1 1] an...	Convolution	60(S) × 80(S) × 512(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 512
19	Bridge-ReLU-1 ReLU	ReLU	60(S) × 80(S) × 512(C) × 1(B)	-
20	Bridge-Conv-2 512 3×3 convolutions with stride [1 1] an...	Convolution	60(S) × 80(S) × 512(C) × 1(B)	Wei… 3 × 3 × 512… Bias 1 × 1 × 512
21	Bridge-ReLU-2 ReLU	ReLU	60(S) × 80(S) × 512(C) × 1(B)	-
22	Bridge-DropOut 50% dropout	Dropout	60(S) × 80(S) × 512(C) × 1(B)	-
23	Decoder-Stage-1-UpConv 256 2×2 transposed convolutions with st...	Transposed Convol...	120(S) × 160(S) × 256(C) × 1(B)	Wei… 2 × 2 × 256… Bias 1 × 1 × 256
24	Decoder-Stage-1-UpReLU ReLU	ReLU	120(S) × 160(S) × 256(C) × 1(B)	-



ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
23	Decoder-Stage-1-UpConv 256 2×2 transposed convolutions with st...	Transposed Convol...	120(S) × 160(S) × 256(C) × 1(B)	Wei… 2 × 2 × 256… Bias 1 × 1 × 256
24	Decoder-Stage-1-UpReLU ReLU	ReLU	120(S) × 160(S) × 256(C) × 1(B)	-
25	Decoder-Stage-1-DepthConcaten... Depth concatenation of 2 inputs	Depth concatenation	120(S) × 160(S) × 512(C) × 1(B)	-
26	Decoder-Stage-1-Conv-1 256 3×3 convolutions with stride [1 1] an...	Convolution	120(S) × 160(S) × 256(C) × 1(B)	Wei… 3 × 3 × 512… Bias 1 × 1 × 256
27	Decoder-Stage-1-ReLU-1 ReLU	ReLU	120(S) × 160(S) × 256(C) × 1(B)	-
28	Decoder-Stage-1-Conv-2 256 3×3 convolutions with stride [1 1] an...	Convolution	120(S) × 160(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
29	Decoder-Stage-1-ReLU-2 ReLU	ReLU	120(S) × 160(S) × 256(C) × 1(B)	-
30	Decoder-Stage-2-UpConv 128 2×2 transposed convolutions with st...	Transposed Convol...	240(S) × 320(S) × 128(C) × 1(B)	Wei… 2 × 2 × 128… Bias 1 × 1 × 128
31	Decoder-Stage-2-UpReLU ReLU	ReLU	240(S) × 320(S) × 128(C) × 1(B)	-
32	Decoder-Stage-2-DepthConcaten... Depth concatenation of 2 inputs	Depth concatenation	240(S) × 320(S) × 256(C) × 1(B)	-
33	Decoder-Stage-2-Conv-1 128 3×3 convolutions with stride [1 1] an...	Convolution	240(S) × 320(S) × 128(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 128
34	Decoder-Stage-2-ReLU-1 ReLU	ReLU	240(S) × 320(S) × 128(C) × 1(B)	-
35	Decoder-Stage-2-Conv-2 128 3×3 convolutions with stride [1 1] an...	Convolution	240(S) × 320(S) × 128(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 128
36	Decoder-Stage-2-ReLU-2 ReLU	ReLU	240(S) × 320(S) × 128(C) × 1(B)	-
37	Decoder-Stage-3-UpConv 64 2×2 transposed convolutions with stri...	Transposed Convol...	480(S) × 640(S) × 64(C) × 1(B)	Weig… 2 × 2 × 64 … Bias 1 × 1 × 64
38	Decoder-Stage-3-UpReLU ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
39	Decoder-Stage-3-DepthConcaten... Depth concatenation of 2 inputs	Depth concatenation	480(S) × 640(S) × 128(C) × 1(B)	-
40	Decoder-Stage-3-Conv-1 64 3×3 convolutions with stride [1 1] and...	Convolution	480(S) × 640(S) × 64(C) × 1(B)	Weig… 3 × 3 × 128… Bias 1 × 1 × 64
41	Decoder-Stage-3-ReLU-1 ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
42	Decoder-Stage-3-Conv-2 64 3×3 convolutions with stride [1 1] and...	Convolution	480(S) × 640(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
43	Decoder-Stage-3-ReLU-2 ReLU	ReLU	480(S) × 640(S) × 64(C) × 1(B)	-
44	Final-ConvolutionLayer 5 1×1 convolutions with stride [1 1] and ...	Convolution	480(S) × 640(S) × 5(C) × 1(B)	Weig… 1 × 1 × 64… Bias 1 × 1 × 5
45	Softmax-Layer softmax	Softmax	480(S) × 640(S) × 5(C) × 1(B)	-
46	Segmentation-Layer Cross-entropy loss	Pixel Classification ...	480(S) × 640(S) × 5(C) × 1(B)	-

Train U-Net Network for Semantic Segmentation

Load training images and pixel labels into the workspace.

```
dataSetDir = fullfile(toolboxdir('vision'),'visiondata','triangleImages');
imageDir = fullfile(dataSetDir,'trainingImages');
labelDir = fullfile(dataSetDir,'trainingLabels');
```

Create an imageDatastore object to store the training images.

```
imds = imageDatastore(imageDir);
```

Define the class names and their associated label IDs.

```
classNames = ["triangle","background"];
labelIDs   = [255 0];
```

Create a pixelLabelDatastore object to store ground truth pixel labels for training images.

```
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
```

Train U-Net Network for Semantic Segmentation

Create the U-Net network.

```
imageSize = [32 32];
numClasses = 2;
lgraph = unetLayers(imageSize, numClasses)
```

Create a datastore for training the network.

```
ds = combine(imds,pxds);
```

Set training options.

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',1e-3, ...
    'MaxEpochs',20, ...
    'VerboseFrequency',10);
```

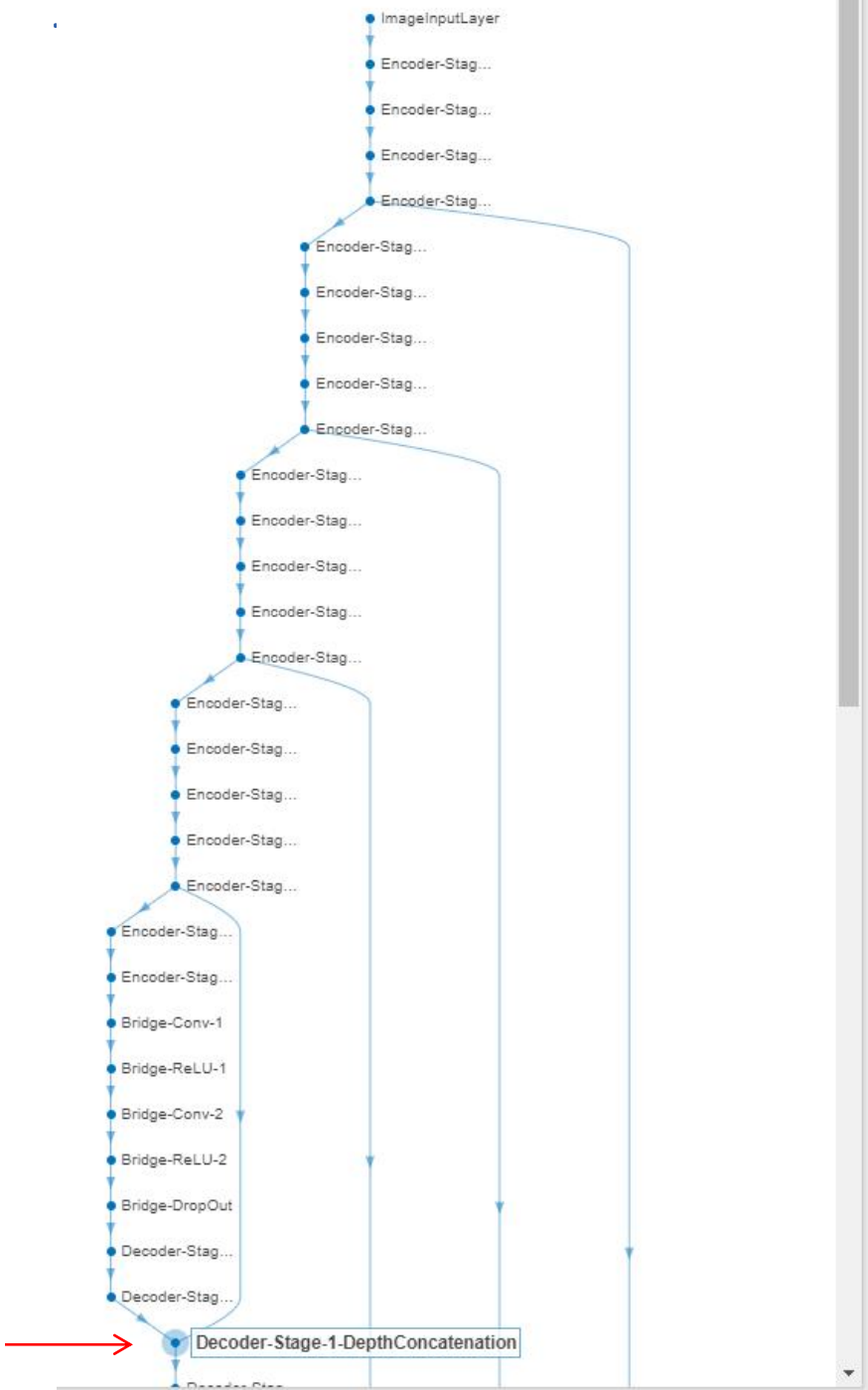
Train U-Net Network for Semantic Segmentation

Train the network.

```
net = trainNetwork(ds,lgraph,options)
```

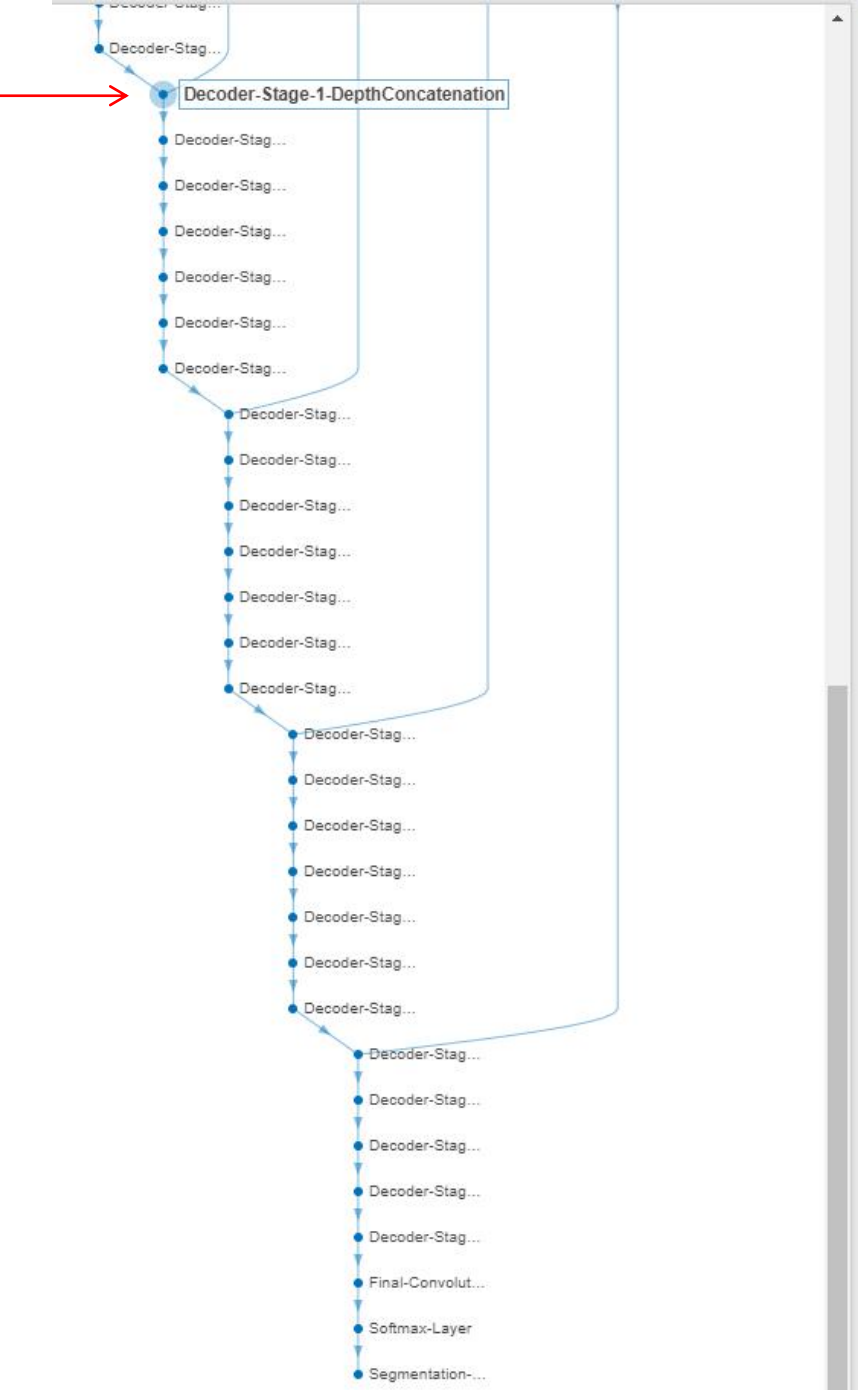
```
Training on single GPU.
Initializing input data normalization.
|========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|========================================================================================|
|       1 |           1 |       00:00:03 |       17.68% |       8.7782 |          0.0010 |
|      10 |          10 |       00:00:24 |       95.38% |       0.5892 |          0.0010 |
|      20 |          20 |       00:00:48 |       96.98% |       0.2099 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.
net =
  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```



ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
7	Encoder-Stage-2-Conv-1 128 3×3 convolutions with stride [1 1] an...	Convolution	16(S) × 16(S) × 128(C) × 1(B)	Weig… 3 × 3 × 64 … Bias 1 × 1 × 128
8	Encoder-Stage-2-ReLU-1 ReLU	ReLU	16(S) × 16(S) × 128(C) × 1(B)	-
9	Encoder-Stage-2-Conv-2 128 3×3 convolutions with stride [1 1] an...	Convolution	16(S) × 16(S) × 128(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 128
10	Encoder-Stage-2-ReLU-2 ReLU	ReLU	16(S) × 16(S) × 128(C) × 1(B)	-
11	Encoder-Stage-2-MaxPool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	8(S) × 8(S) × 128(C) × 1(B)	-
12	Encoder-Stage-3-Conv-1 256 3×3 convolutions with stride [1 1] an...	Convolution	8(S) × 8(S) × 256(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 256
13	Encoder-Stage-3-ReLU-1 ReLU	ReLU	8(S) × 8(S) × 256(C) × 1(B)	-
14	Encoder-Stage-3-Conv-2 256 3×3 convolutions with stride [1 1] an...	Convolution	8(S) × 8(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
15	Encoder-Stage-3-ReLU-2 ReLU	ReLU	8(S) × 8(S) × 256(C) × 1(B)	-
16	Encoder-Stage-3-MaxPool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	4(S) × 4(S) × 256(C) × 1(B)	-
17	Encoder-Stage-4-Conv-1 512 3×3 convolutions with stride [1 1] an...	Convolution	4(S) × 4(S) × 512(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 512
18	Encoder-Stage-4-ReLU-1 ReLU	ReLU	4(S) × 4(S) × 512(C) × 1(B)	-
19	Encoder-Stage-4-Conv-2 512 3×3 convolutions with stride [1 1] an...	Convolution	4(S) × 4(S) × 512(C) × 1(B)	Wei… 3 × 3 × 512… Bias 1 × 1 × 512
20	Encoder-Stage-4-ReLU-2 ReLU	ReLU	4(S) × 4(S) × 512(C) × 1(B)	-
21	Encoder-Stage-4-DropOut 50% dropout	Dropout	4(S) × 4(S) × 512(C) × 1(B)	-
22	Encoder-Stage-4-MaxPool 2×2 max pooling with stride [2 2] and pa...	Max Pooling	2(S) × 2(S) × 512(C) × 1(B)	-
23	Bridge-Conv-1 1024 3×3 convolutions with stride [1 1] a...	Convolution	2(S) × 2(S) × 1024(C) × 1(B)	Wei… 3 × 3 × 512… Bias 1 × 1 × 1024
24	Bridge-ReLU-1 ReLU	ReLU	2(S) × 2(S) × 1024(C) × 1(B)	-
25	Bridge-Conv-2 1024 3×3 convolutions with stride [1 1] a...	Convolution	2(S) × 2(S) × 1024(C) × 1(B)	Wei… 3 × 3 × 102… Bias 1 × 1 × 1024
26	Bridge-ReLU-2 ReLU	ReLU	2(S) × 2(S) × 1024(C) × 1(B)	-
27	Bridge-DropOut 50% dropout	Dropout	2(S) × 2(S) × 1024(C) × 1(B)	-
28	Decoder-Stage-1-UpConv 512 2×2 transposed convolutions with st...	Transposed Convol...	4(S) × 4(S) × 512(C) × 1(B)	Wei… 2 × 2 × 512… Bias 1 × 1 × 512
29	Decoder-Stage-1-UpReLU ReLU	ReLU	4(S) × 4(S) × 512(C) × 1(B)	-
30	Decoder-Stage-1-DepthConcaten... Depth concatenation of 2 inputs	Depth concatenation	4(S) × 4(S) × 1024(C) × 1(B)	-



ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
35	Decoder-Stage-2-UpConv 256 2×2 transposed convolutions with st...	Transposed Convol...	8(S) × 8(S) × 256(C) × 1(B)	Wei… 2 × 2 × 256… Bias 1 × 1 × 256
36	Decoder-Stage-2-UpReLU ReLU	ReLU	8(S) × 8(S) × 256(C) × 1(B)	-
37	Decoder-Stage-2-DepthConcaten... Depth concatenation of 2 inputs	Depth concatenation	8(S) × 8(S) × 512(C) × 1(B)	-
38	Decoder-Stage-2-Conv-1 256 3×3 convolutions with stride [1 1] an...	Convolution	8(S) × 8(S) × 256(C) × 1(B)	Wei… 3 × 3 × 512… Bias 1 × 1 × 256
39	Decoder-Stage-2-ReLU-1 ReLU	ReLU	8(S) × 8(S) × 256(C) × 1(B)	-
40	Decoder-Stage-2-Conv-2 256 3×3 convolutions with stride [1 1] an...	Convolution	8(S) × 8(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
41	Decoder-Stage-2-ReLU-2 ReLU	ReLU	8(S) × 8(S) × 256(C) × 1(B)	-
42	Decoder-Stage-3-UpConv 128 2×2 transposed convolutions with st...	Transposed Convol...	16(S) × 16(S) × 128(C) × 1(B)	Wei… 2 × 2 × 128… Bias 1 × 1 × 128
43	Decoder-Stage-3-UpReLU ReLU	ReLU	16(S) × 16(S) × 128(C) × 1(B)	-
44	Decoder-Stage-3-DepthConcaten... Depth concatenation of 2 inputs	Depth concatenation	16(S) × 16(S) × 256(C) × 1(B)	-
45	Decoder-Stage-3-Conv-1 128 3×3 convolutions with stride [1 1] an...	Convolution	16(S) × 16(S) × 128(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 128
46	Decoder-Stage-3-ReLU-1 ReLU	ReLU	16(S) × 16(S) × 128(C) × 1(B)	-
47	Decoder-Stage-3-Conv-2 128 3×3 convolutions with stride [1 1] an...	Convolution	16(S) × 16(S) × 128(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 128
48	Decoder-Stage-3-ReLU-2 ReLU	ReLU	16(S) × 16(S) × 128(C) × 1(B)	-
49	Decoder-Stage-4-UpConv 64 2×2 transposed convolutions with stri...	Transposed Convol...	32(S) × 32(S) × 64(C) × 1(B)	Weig… 2 × 2 × 64 … Bias 1 × 1 × 64
50	Decoder-Stage-4-UpReLU ReLU	ReLU	32(S) × 32(S) × 64(C) × 1(B)	-
51	Decoder-Stage-4-DepthConcaten... Depth concatenation of 2 inputs	Depth concatenation	32(S) × 32(S) × 128(C) × 1(B)	-
52	Decoder-Stage-4-Conv-1 64 3×3 convolutions with stride [1 1] and...	Convolution	32(S) × 32(S) × 64(C) × 1(B)	Weig… 3 × 3 × 128… Bias 1 × 1 × 64
53	Decoder-Stage-4-ReLU-1 ReLU	ReLU	32(S) × 32(S) × 64(C) × 1(B)	-
54	Decoder-Stage-4-Conv-2 64 3×3 convolutions with stride [1 1] and...	Convolution	32(S) × 32(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
55	Decoder-Stage-4-ReLU-2 ReLU	ReLU	32(S) × 32(S) × 64(C) × 1(B)	-
56	Final-ConvolutionLayer 2 1×1 convolutions with stride [1 1] and ...	Convolution	32(S) × 32(S) × 2(C) × 1(B)	Weig… 1 × 1 × 64… Bias 1 × 1 × 2
57	Softmax-Layer softmax	Softmax	32(S) × 32(S) × 2(C) × 1(B)	-
58	Segmentation-Layer Cross-entropy loss	Pixel Classification ...	32(S) × 32(S) × 2(C) × 1(B)	-

DeepLab系列

DeepLab系列是谷歌团队提出的一系列语义分割算法。DeepLab v1于2014年提出，随后2017到2018年又相继提出了DeepLab v2，DeepLab v3以及DeepLab v3+。

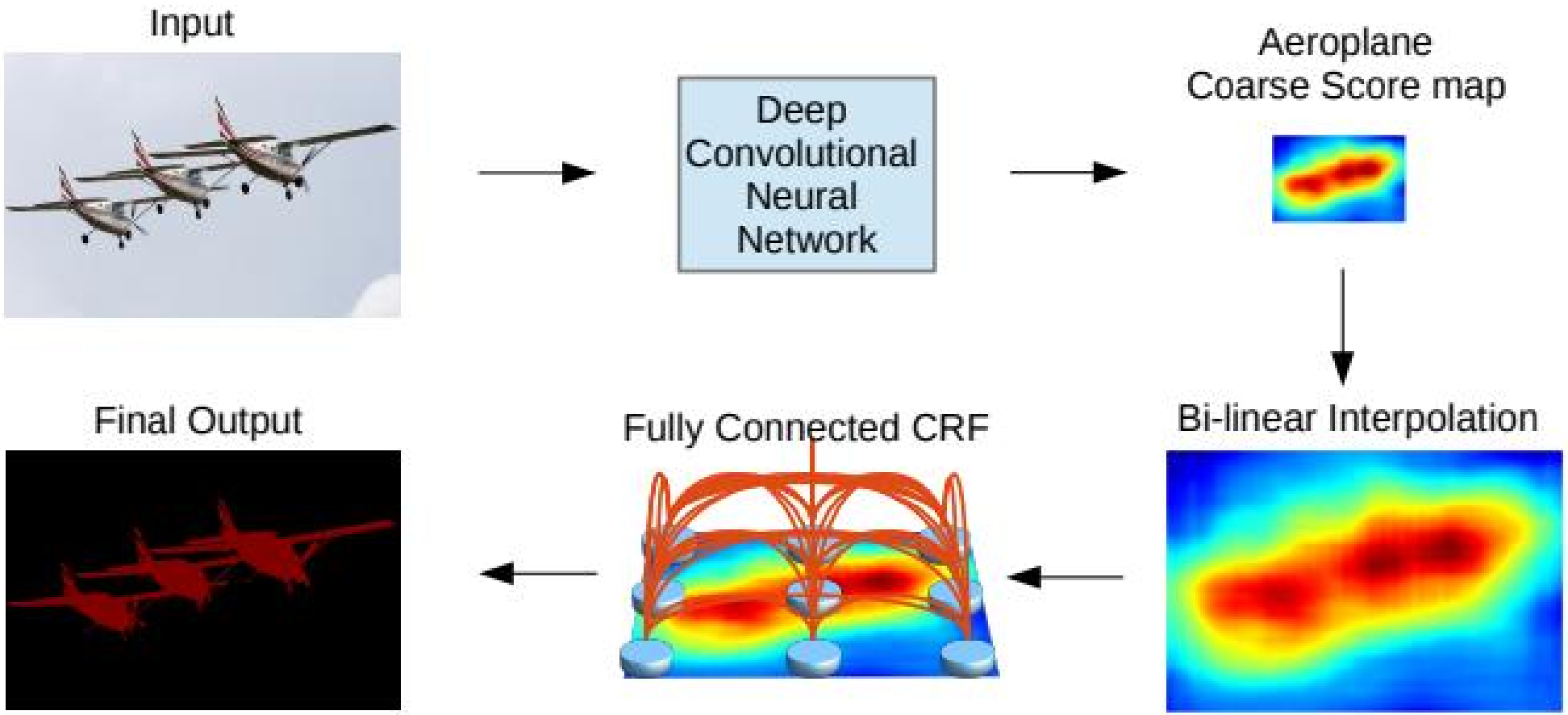
DeepLab v1的两个创新点是空洞卷积（Atros Convolution）和基于全连接条件随机场（Fully Connected CRF）。DeepLab v2的不同之处是提出了空洞空间金字塔池化（Atros Spatial Pyramid Pooling，ASPP）。DeepLab v3则是对ASPP进行了进一步的优化包括添加 卷积，BN操作等。DeepLab v3+则是仿照U-Net的结构添加了一个向上采样的解码器模块，用来优化边缘的精度。

DeepLab系列

DeepLab v1 将深度卷积神经网络（DCNN）和概率图模型结合起来，以完成逐像素的分类——即语义分割。由于DCNN本身具有空间不变性，这使得其在高级任务（图像识别等）有很好的表现，但是对于低级别的任务（语义分割等），单独使用其最后一层的输出无法准确地完成定位。作者将DCNN最后一层的输出和全连接的条件随机场（CRF）相结合，来解决DCNN定位不准确的问题。
网络主体基于VGG16，使用空洞卷积（膨胀卷积）来实现在不减小感受野的情况下，减小模型的下采样率（从VGG16的32倍下采样率到本模型的8倍下采样率），使其很好地保留了小物体的细节特征，同时在上采样时，由于仅需要上采样8倍，使得模型可以直接使用双线性插值进行上采样，而不需要用需要训练的上采样算法（比如转置卷积），加快了模型的训练/推理速度，也减少了参数量。

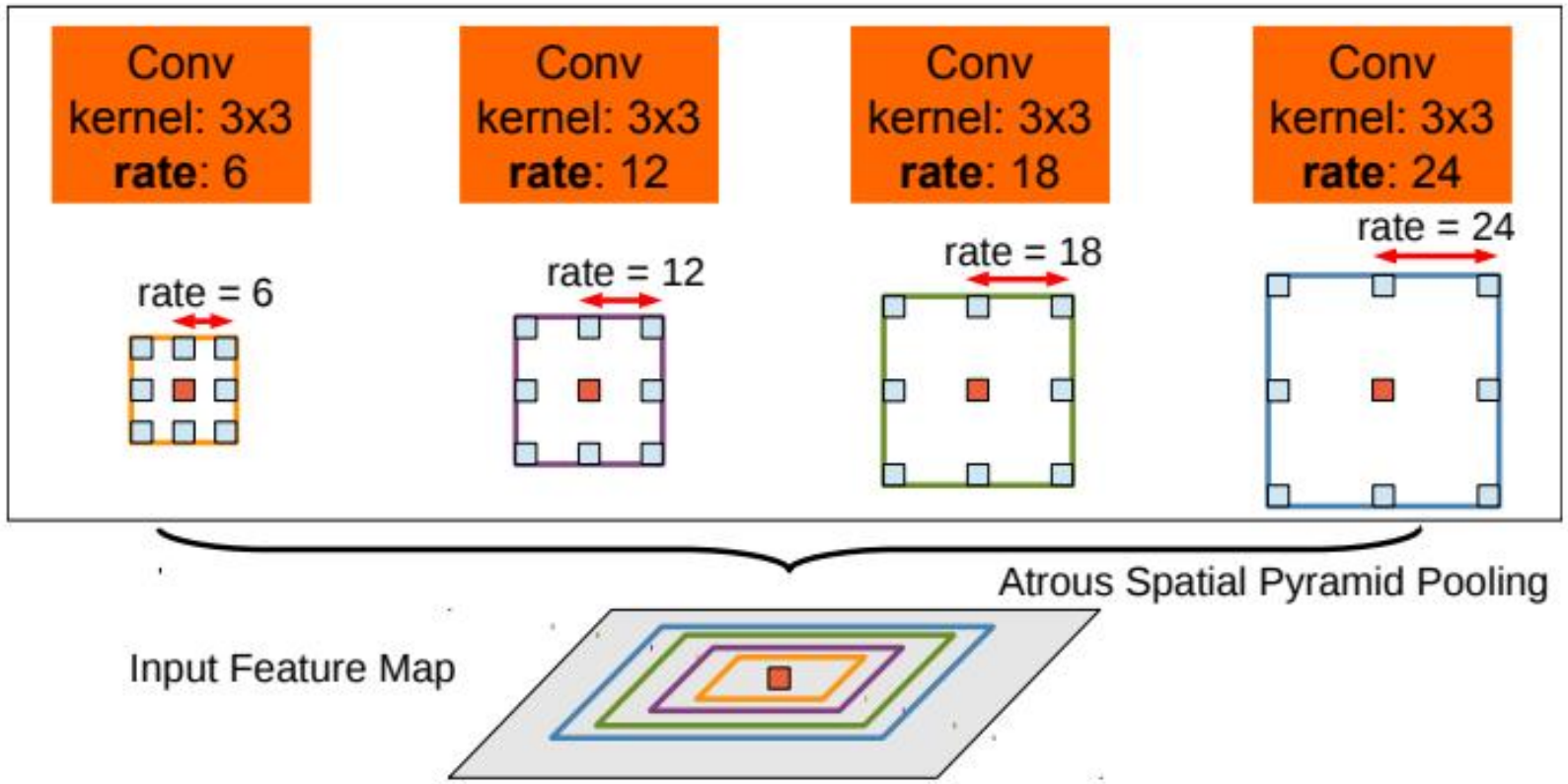
DeepLab系列

DeepLab v1有两个核心点，即：空洞卷积和CRF。它首先将VGG的普通卷积替换为空洞卷积得到分割图，在通过CRF将得到的分割图进行后处理优化



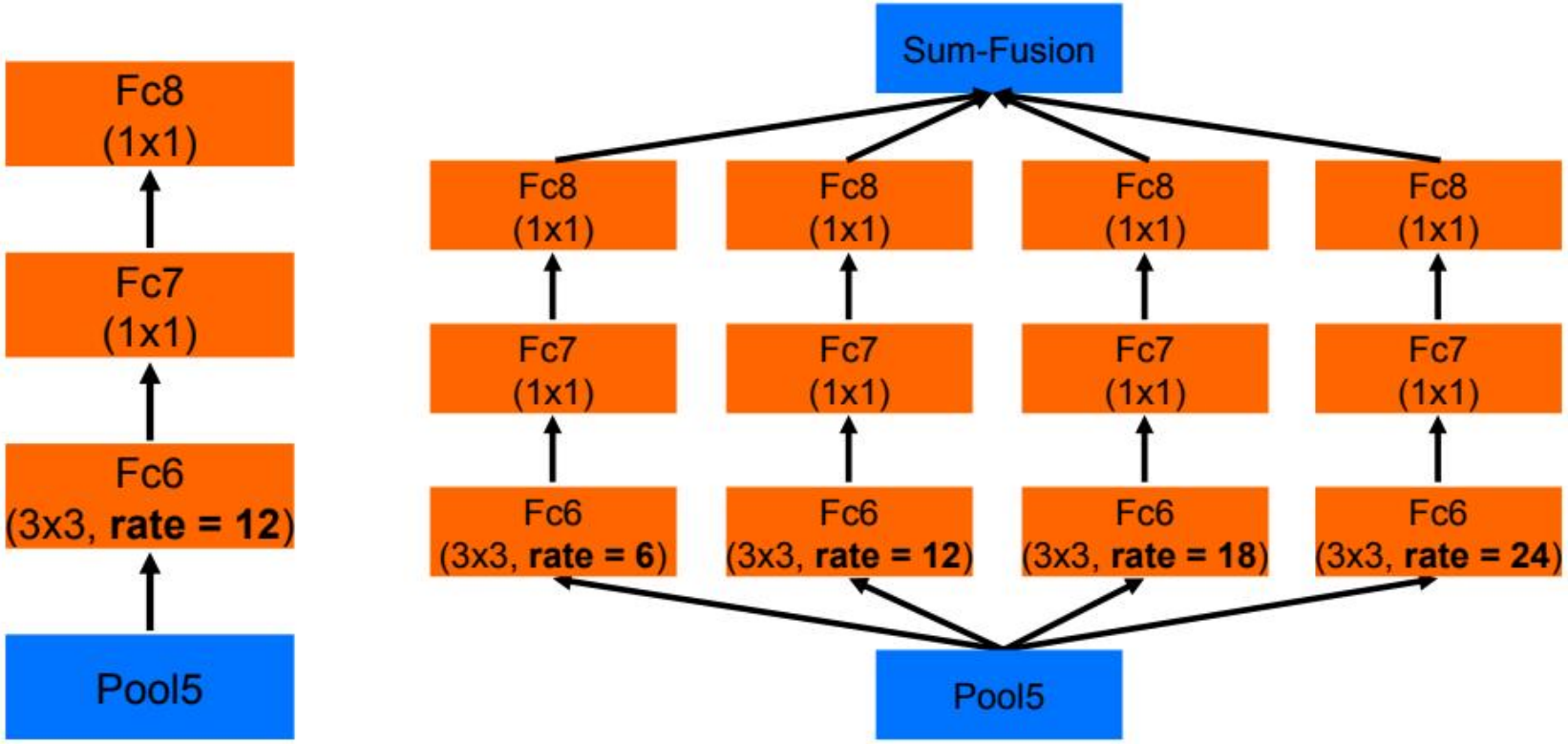
DeepLab系列

DeepLab v2 使用ResNet作为主干网络，还提出了用于适应多尺度物体的方法，一是使用多尺度的图片进行训练，二是使用ASPP（Atrous Spatial Pyramid Pooling）结构。受到RCNN中SPP（spatial pyramid pooling）的启发，作者引入了ASPP来实现对多尺度物体的检测。ASPP使用多个具有不同采样率的空洞卷积并行地提取不同感受野的特征，这些并行的特征在每个分支中经过后处理又融合在一起来产生最后结果。



DeepLab系列

DeepLab v2 依旧保持了DeepLabv1 的处理流程，即以空洞卷积和CRF为核心。
DeepLabv2 相对于 DeepLab v1 最大的改动是增加了受 SPP（Spacial Pyramid Pooling）启发得来的 ASPP（Atrous Spacial Pyramid Pooling），在模型最后进行像素分类之前增加一个类似 Inception 的结构，包含不同 rate（空洞间隔） 的 Atrous Conv（空洞卷积），增强模型识别不同尺寸的同一物体的能力。

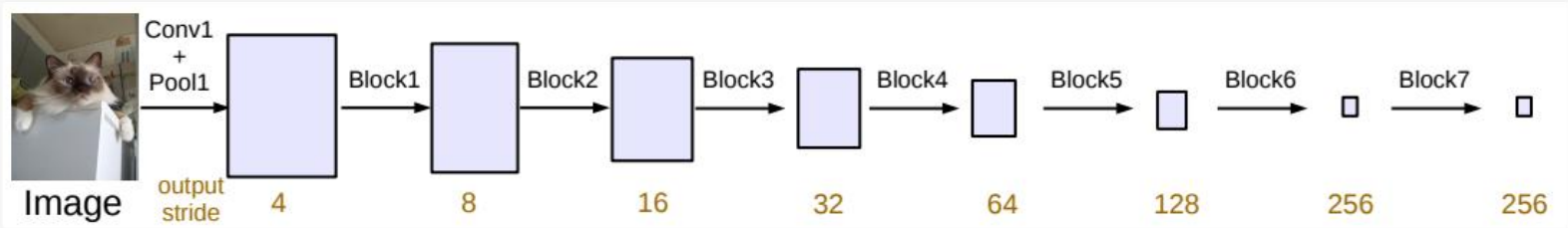


DeepLab系列

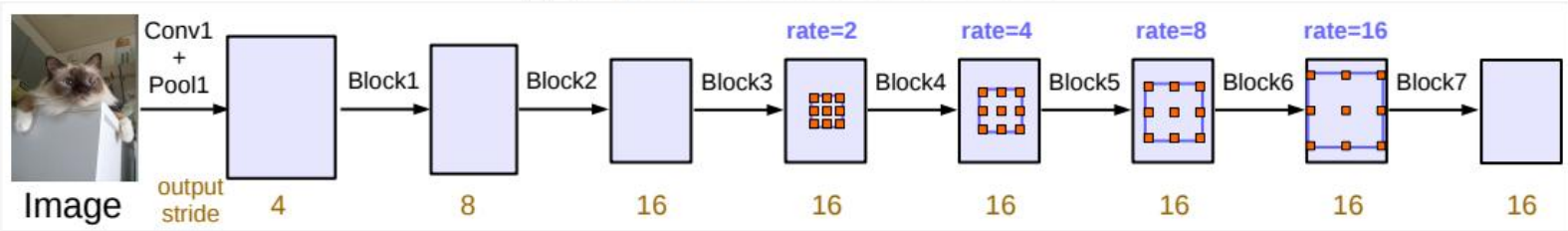
DeepLab v3 移除了CRF模块，引入了Multi-Grid策略，即多次使用空洞卷积核而不像在v1和v2中仅使用一次空洞卷积；优化ASPP的结构，包括加入BN等。DeepLab v3的Multi-Grid策略参考了HDC（hybrid dilated convolution ）的思想，在一个block中连续使用多个不同扩张率的空洞卷积。HDC的提出是为了解决空洞卷积可能会产生的gridding问题。

DeepLab系列

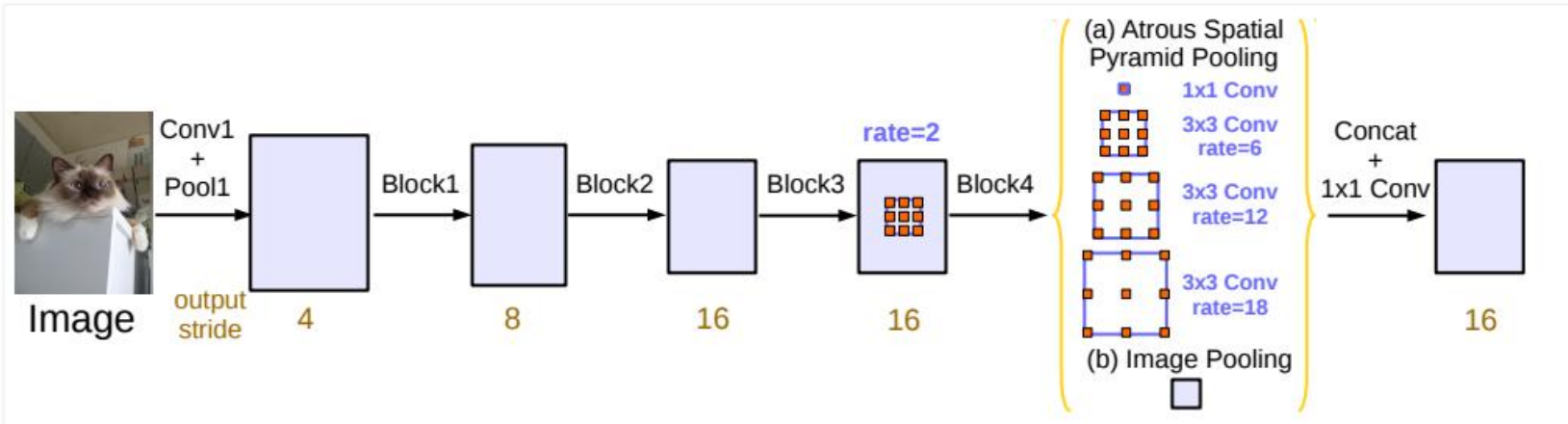
DeepLab V3将空洞卷积应用在级联模块。具体来说，取ResNet中最后一个block，在下图中为block4，并在其后面增加级联模块。



(a) Going deeper without atrous convolution.

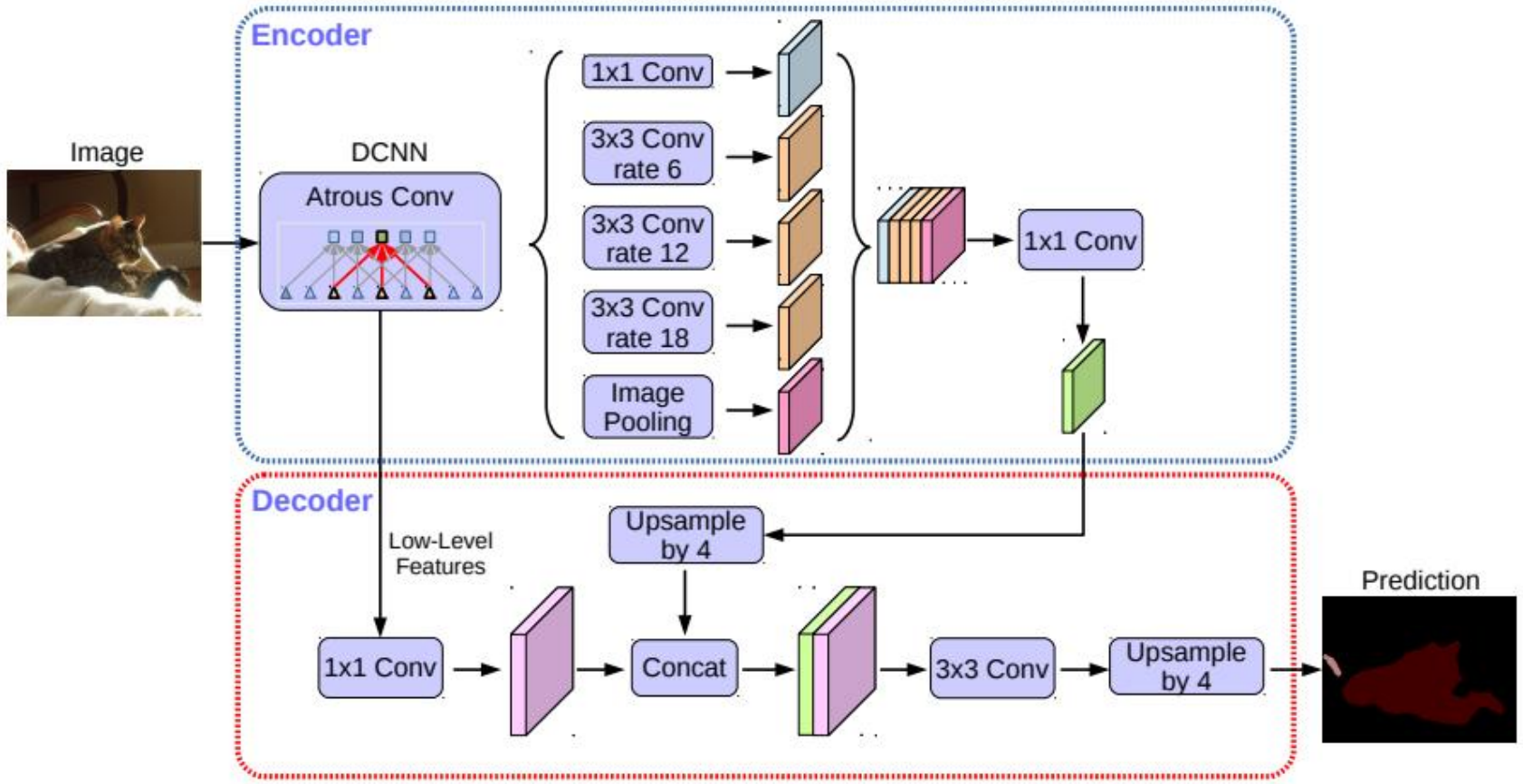


对ASPP模块做了改进：应用了BN层，使用模型最后的特征映射的全局平均池化



DeepLab系列

DeepLab v1 ~ v3 系列都是在降采样8倍的尺度上进行预测的，因该尺度特征并没有包含较多的浅层特征导致了边界效果不理想。DeepLabv3+模型的整体架构如图所示，相比DeepLab v3，DeepLab v3+引入了Decoder模块，其将底层特征与高层特征进一步融合，提升分割边界准确度。



DeepLab系列

DeepLab全系列基于空洞卷积并进行了一系列改进。DeepLab v1直接在VGG的基础上加入了空洞卷积，但是效果不理想因此使用了CRF进行了后处理优化。DeepLab v2在v1的基础上添加了ASPP模块，ASPP的引入优化了对不同尺度的目标的分割效果，但是依然需要依赖CRF进行优化。DeepLab v3的Multi-Grid的策略参考了HDC，解决了空洞卷积的Gridding问题；同时DeepLab v3对ASPP的修改也赋予了ASPP更强的表征能力，不再需要CRF的修复。DeepLab v3+参考了目标检测中常见的特征融合策略，使网络保留了比较多的浅层信息；同时它也加入了深度可分离卷积来对分割网络的速度进行优化。

DeepLab 示例

- **Semantic Segmentation Using Deep Learning**

Semantic Segmentation Using Deep Learning

This example shows how to train a semantic segmentation network using Deeplab v3+, one type of convolutional neural network (CNN) designed for semantic image segmentation.

This example creates the Deeplab v3+ network with weights initialized from a pre-trained Resnet-18 network. ResNet-18 is an efficient network that is well suited for applications with limited processing resources. Other pretrained networks such as MobileNet v2 or ResNet-50 can also be used depending on application requirements.

Semantic Segmentation Using Deep Learning

In addition, download a pretrained version of DeepLab v3+. The pretrained model allows you to run the entire example without having to wait for training to complete.

```
pretrainedURL =
'https://ssd.mathworks.com/supportfiles/vision/data/deeplabv3plusResnet18CamVid.zip';
pretrainedFolder = fullfile(pwd,"deeplabv3plusResnet18CamVid");
pretrainedNetworkZip = fullfile(pretrainedFolder,'deeplabv3plusResnet18CamVid.zip');
if ~exist(pretrainedNetworkZip,'file')
    mkdir(pretrainedFolder);
    disp('Downloading pretrained network (58 MB)...');
    websave(pretrainedNetworkZip,pretrainedURL);
end
```

Semantic Segmentation Using Deep Learning

Load CamVid Images

Use imageDatastore to load CamVid images. The imageDatastore enables you to efficiently load a large collection of images on disk.

```
imgDir = fullfile(outputFolder,'images','701_StillsRaw_full');
imds = imageDatastore(imgDir);
```

Display one of the images.

```
I = readimage(imds,559);
I = histeq(I);
imshow(I)
```

Semantic Segmentation Using Deep Learning

Load CamVid Pixel-Labeled Images

Use pixelLabelDatastore to load CamVid pixel label image data. A pixelLabelDatastore encapsulates the pixel label data and the label ID to a class name mapping.

We make training easier, we group the 32 original classes in CamVid to 11 classes. Specify these classes.

```
classes = [
    "Sky"
    "Building"
    "Pole"
    "Road"
    "Pavement"
    "Tree"
    "SignSymbol"
    "Fence"
    "Car"
    "Pedestrian"
    "Bicyclist"
    ];
```

Semantic Segmentation Using Deep Learning

To reduce 32 classes into 11, multiple classes from the original dataset are grouped together. For example, "Car" is a combination of "Car", "SUVPickupTruck", "Truck_Bus", "Train", and "OtherMoving". Return the grouped label IDs by using the supporting function camvidPixelLabelIDs, which is listed at the end of this example.

```
labelIDs = camvidPixelLabelIDs();
```

Use the classes and label IDs to create the pixelLabelDatastore.

```
labelDir = fullfile(outputFolder,'labels');
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);
```

Semantic Segmentation Using Deep Learning

Read and display one of the pixel-labeled images by overlaying it on top of an image.

```
C = readimage(pxds,559);
cmap = camvidColorMap;
B = labeloverlay(I,C,'ColorMap',cmap);
imshow(B)
pixelLabelColorbar(cmap,classes);
```

Semantic Segmentation Using Deep Learning

Analyze Dataset Statistics

To see the distribution of class labels in the CamVid dataset, use countEachLabel. This function counts the number of pixels by class label.

```
tbl = countEachLabel(pxds)
```

`tbl = 11×3 table`

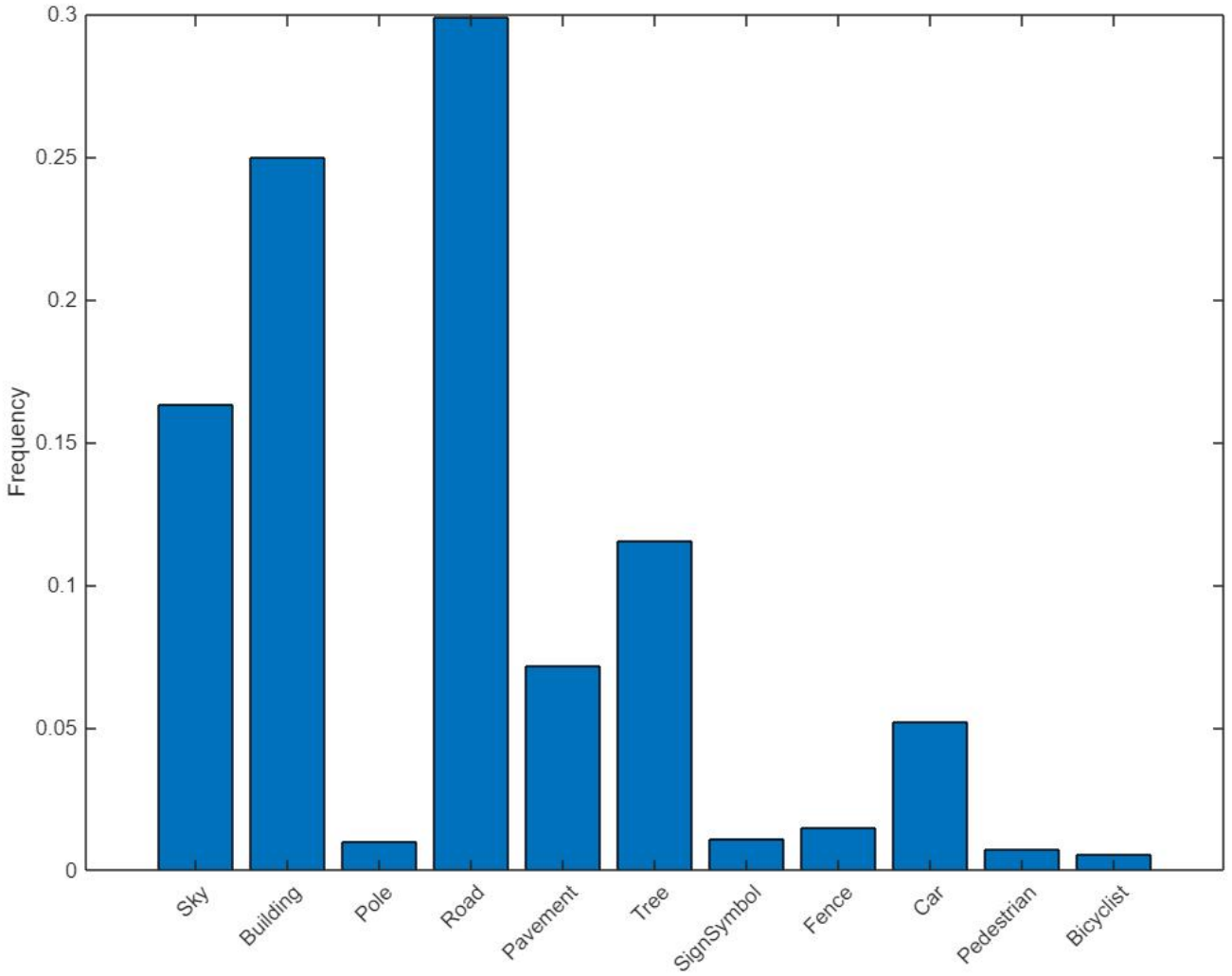
	Name	PixelCount	ImagePixelCount
3	'Pole'	4798742	483148800
4	'Road'	140535728	484531200
5	'Pavement'	33614414	472089600
6	'Tree'	54258673	447897600
7	'SignSymbol'	5224247	468633600
8	'Fence'	6921061	251596800
9	'Car'	24436957	483148800
10	'Pedestrian'	3402909	444441600
11	'Bicyclist'	2591222	261964800

Semantic Segmentation Using Deep Learning

Visualize the pixel counts by class.

```
frequency = tbl.PixelCount/sum(tbl.PixelCount);

bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
```



Prepare Training, Validation, and Test Sets

Deeplab v3+ is trained using 60% of the images from the dataset. The rest of the images are split evenly in 20% and 20% for validation and testing respectively. The following code randomly splits the image and pixel label data into a training, validation and test set.

```
[imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] =
partitionCamVidData(imds,pxds);
```

The 60/20/20 split results in the following number of training, validation and test images:

```
numTrainingImages = numel(imdsTrain.Files)
numValImages = numel(imdsVal.Files)
numTestingImages = numel(imdsTest.Files)
```

Create the Network

Use the deeplabv3plusLayers function to create a DeepLab v3+ network based on ResNet-18. Choosing the best network for your application requires empirical analysis and is another level of hyperparameter tuning. For example, you can experiment with different base networks such as ResNet-50 or MobileNet v2, or you can try other semantic segmentation network architectures such as SegNet, fully convolutional networks (FCN), or U-Net.

```
% Specify the network image size. This is typically the same as the traing image sizes.
imageSize = [720 960 3];

% Specify the number of classes.
numClasses = numel(classes);

% Create DeepLab v3+.
lgraph = deeplabv3plusLayers(imageSize, numClasses, "resnet18");
```

Balance Classes Using Class Weighting

As shown earlier, the classes in CamVid are not balanced. To improve training, you can use class weighting to balance the classes. Use the pixel label counts computed earlier with countEachLabel and calculate the median frequency class weights.

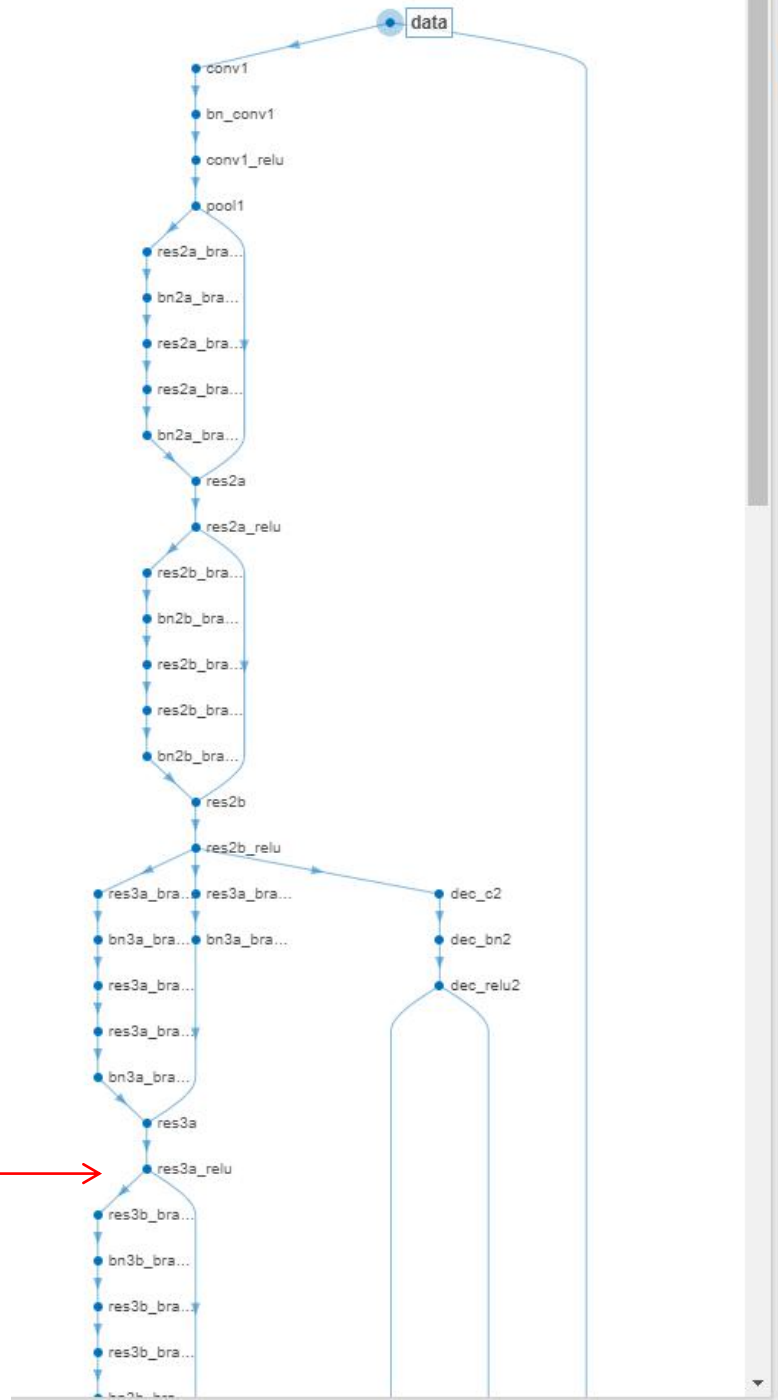
```
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq
```

```
classWeights = 11×1
    0.3182
    0.2082
    5.0924
    0.1744
    0.7103
    0.4175
    4.5371
    1.8386
    1.0000
    6.6059
      ⋮
```

Semantic Segmentation Using Deep Learning

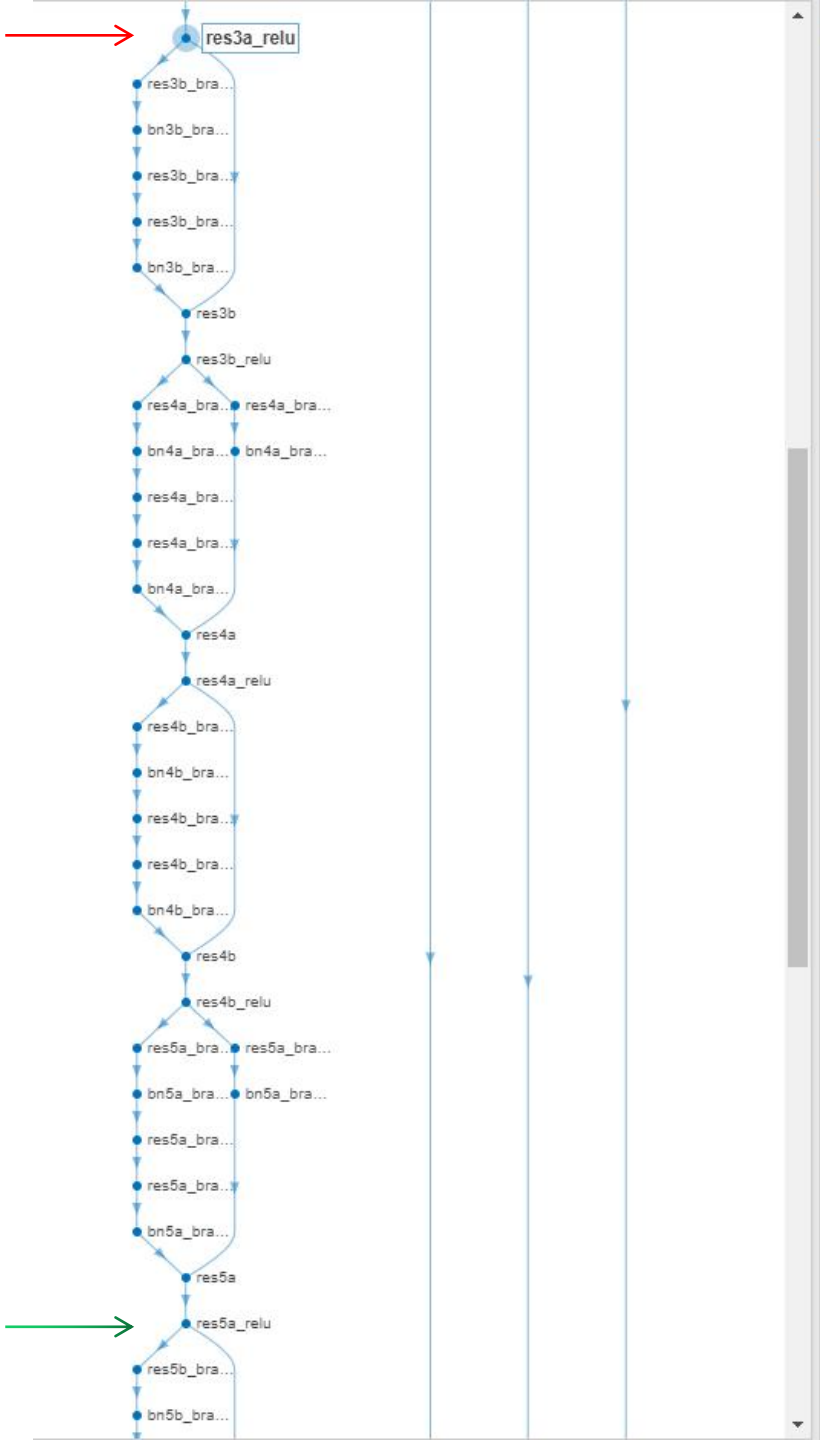
Specify the class weights using a pixelClassificationLayer.

```
pxLayer =
pixelClassificationLayer('Name','labels','Classes',tbl.Name,'ClassWeights',classWeights);
lgraph = replaceLayer(lgraph,"classification",pxLayer);
```



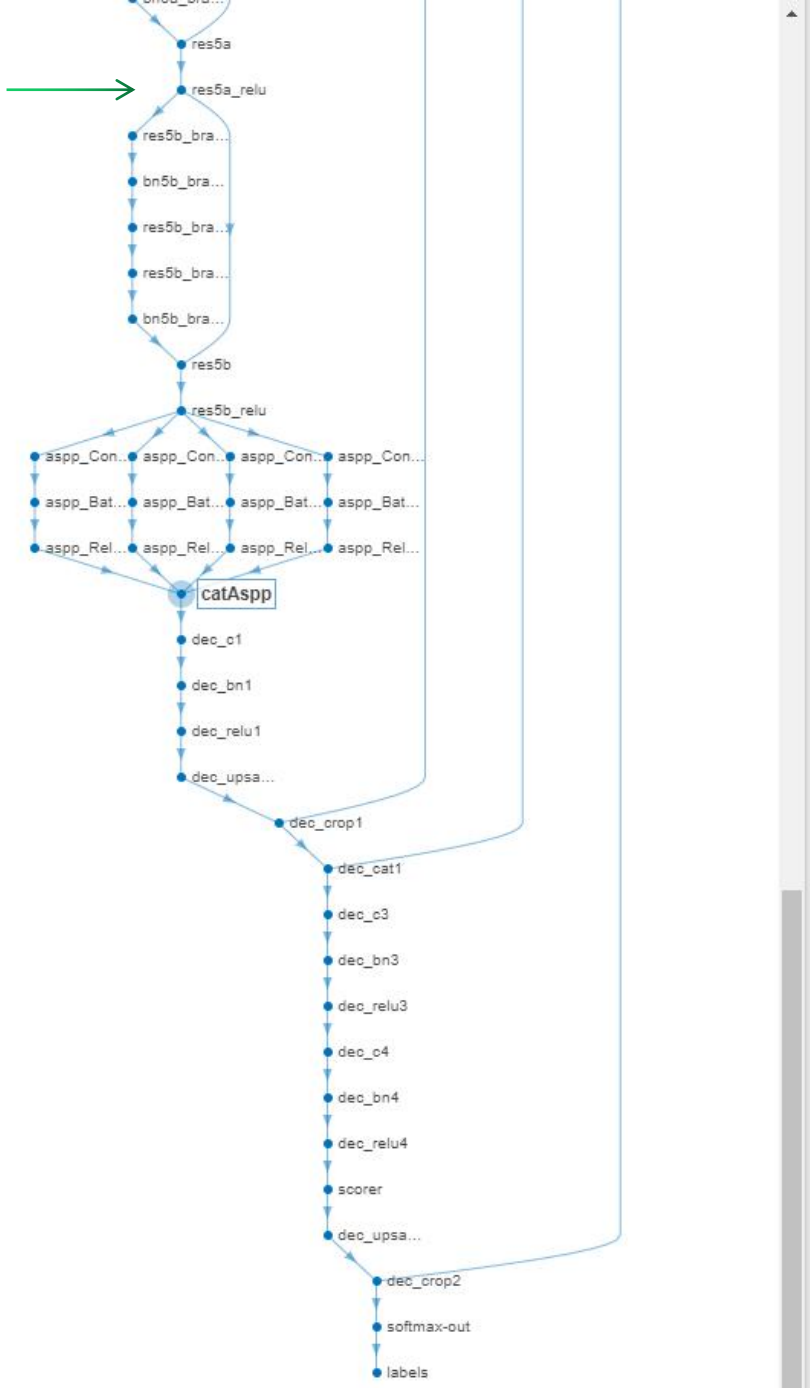
ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
1	data 720×960×3 images with 'zscore' normali...	Image Input	720(S) × 960(S) × 3(C) × 1(B)	-
2	conv1 64 7×7×3 convolutions with stride [2 2] a...	Convolution	360(S) × 480(S) × 64(C) × 1(B)	Weig… 7 × 7 × 3 … Bias 1 × 1 × 64
3	bn_conv1 Batch normalization with 64 channels	Batch Normalization	360(S) × 480(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
4	conv1_relu ReLU	ReLU	360(S) × 480(S) × 64(C) × 1(B)	-
5	pool1 3×3 max pooling with stride [2 2] and pa...	Max Pooling	180(S) × 240(S) × 64(C) × 1(B)	-
6	res2a_branch2a 64 3×3×64 convolutions with stride [1 1] ...	Convolution	180(S) × 240(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
7	bn2a_branch2a Batch normalization with 64 channels	Batch Normalization	180(S) × 240(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
8	res2a_branch2a_relu ReLU	ReLU	180(S) × 240(S) × 64(C) × 1(B)	-
9	res2a_branch2b 64 3×3×64 convolutions with stride [1 1] ...	Convolution	180(S) × 240(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
10	bn2a_branch2b Batch normalization with 64 channels	Batch Normalization	180(S) × 240(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
11	res2a Element-wise addition of 2 inputs	Addition	180(S) × 240(S) × 64(C) × 1(B)	-
12	res2a_relu ReLU	ReLU	180(S) × 240(S) × 64(C) × 1(B)	-
13	res2b_branch2a 64 3×3×64 convolutions with stride [1 1] ...	Convolution	180(S) × 240(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
14	bn2b_branch2a Batch normalization with 64 channels	Batch Normalization	180(S) × 240(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
15	res2b_branch2a_relu ReLU	ReLU	180(S) × 240(S) × 64(C) × 1(B)	-
16	res2b_branch2b 64 3×3×64 convolutions with stride [1 1] ...	Convolution	180(S) × 240(S) × 64(C) × 1(B)	Weig… 3 × 3 × 64… Bias 1 × 1 × 64
17	bn2b_branch2b Batch normalization with 64 channels	Batch Normalization	180(S) × 240(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
18	res2b Element-wise addition of 2 inputs	Addition	180(S) × 240(S) × 64(C) × 1(B)	-
19	res2b_relu ReLU	ReLU	180(S) × 240(S) × 64(C) × 1(B)	-
20	res3a_branch2a 128 3×3×64 convolutions with stride [2 2...	Convolution	90(S) × 120(S) × 128(C) × 1(B)	Weig… 3 × 3 × 64 … Bias 1 × 1 × 128
21	bn3a_branch2a Batch normalization with 128 channels	Batch Normalization	90(S) × 120(S) × 128(C) × 1(B)	Offset 1 × 1 × 128 Scale 1 × 1 × 128
22	res3a_branch2a_relu ReLU	ReLU	90(S) × 120(S) × 128(C) × 1(B)	-
23	res3a_branch2b 128 3×3×128 convolutions with stride [1 ...	Convolution	90(S) × 120(S) × 128(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 128
24	bn3a_branch2b Batch normalization with 128 channels	Batch Normalization	90(S) × 120(S) × 128(C) × 1(B)	Offset 1 × 1 × 128 Scale 1 × 1 × 128



ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
27	res3a Element-wise addition of 2 inputs	Addition	90(S) × 120(S) × 128(C) × 1(B)	-
28	res3a_relu ReLU	ReLU	90(S) × 120(S) × 128(C) × 1(B)	-
29	res3b_branch2a 128 3×3×128 convolutions with stride [1 ...	Convolution	90(S) × 120(S) × 128(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 128
30	bn3b_branch2a Batch normalization with 128 channels	Batch Normalization	90(S) × 120(S) × 128(C) × 1(B)	Offset 1 × 1 × 128 Scale 1 × 1 × 128
31	res3b_branch2a_relu ReLU	ReLU	90(S) × 120(S) × 128(C) × 1(B)	-
32	res3b_branch2b 128 3×3×128 convolutions with stride [1 ...	Convolution	90(S) × 120(S) × 128(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 128
33	bn3b_branch2b Batch normalization with 128 channels	Batch Normalization	90(S) × 120(S) × 128(C) × 1(B)	Offset 1 × 1 × 128 Scale 1 × 1 × 128
34	res3b Element-wise addition of 2 inputs	Addition	90(S) × 120(S) × 128(C) × 1(B)	-
35	res3b_relu ReLU	ReLU	90(S) × 120(S) × 128(C) × 1(B)	-
36	res4a_branch2a 256 3×3×128 convolutions with stride [2 ...	Convolution	45(S) × 60(S) × 256(C) × 1(B)	Wei… 3 × 3 × 128… Bias 1 × 1 × 256
37	bn4a_branch2a Batch normalization with 256 channels	Batch Normalization	45(S) × 60(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
38	res4a_branch2a_relu ReLU	ReLU	45(S) × 60(S) × 256(C) × 1(B)	-
39	res4a_branch2b 256 3×3×256 convolutions with stride [1 ...	Convolution	45(S) × 60(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
40	bn4a_branch2b Batch normalization with 256 channels	Batch Normalization	45(S) × 60(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
41	res4a_branch1 256 1×1×128 convolutions with stride [2 ...	Convolution	45(S) × 60(S) × 256(C) × 1(B)	Wei… 1 × 1 × 128… Bias 1 × 1 × 256
42	bn4a_branch1 Batch normalization with 256 channels	Batch Normalization	45(S) × 60(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
43	res4a Element-wise addition of 2 inputs	Addition	45(S) × 60(S) × 256(C) × 1(B)	-
44	res4a_relu ReLU	ReLU	45(S) × 60(S) × 256(C) × 1(B)	-
45	res4b_branch2a 256 3×3×256 convolutions with stride [1 ...	Convolution	45(S) × 60(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
46	bn4b_branch2a Batch normalization with 256 channels	Batch Normalization	45(S) × 60(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
47	res4b_branch2a_relu ReLU	ReLU	45(S) × 60(S) × 256(C) × 1(B)	-
48	res4b_branch2b 256 3×3×256 convolutions with stride [1 ...	Convolution	45(S) × 60(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
49	bn4b_branch2b Batch normalization with 256 channels	Batch Normalization	45(S) × 60(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
50	res4b Element-wise addition of 2 inputs	Addition	45(S) × 60(S) × 256(C) × 1(B)	-



ANALYSIS RESULT

	Name	Type	Activations	Learnable Prope...
77	aspp_Conv_4 256 3×3 convolutions with stride [1 1], di...	Convolution	45(S) × 60(S) × 256(C) × 1(B)	Wei… 3 × 3 × 512… Bias 1 × 1 × 256
78	aspp_BatchNorm_4 Batch normalization	Batch Normalization	45(S) × 60(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
79	aspp_Relu_4 ReLU	ReLU	45(S) × 60(S) × 256(C) × 1(B)	-
80	catAspp Depth concatenation of 4 inputs	Depth concatenation	45(S) × 60(S) × 1024(C) × 1(B)	-
81	dec_c1 256 1×1 convolutions with stride [1 1] an...	Convolution	45(S) × 60(S) × 256(C) × 1(B)	Wei… 1 × 1 × 102… Bias 1 × 1 × 256
82	dec_bn1 Batch normalization	Batch Normalization	45(S) × 60(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
83	dec_relu1 ReLU	ReLU	45(S) × 60(S) × 256(C) × 1(B)	-
84	dec_upsample1 256 8×8×256 transposed convolutions ...	Transposed Convol...	180(S) × 240(S) × 256(C) × 1(B)	Wei… 8 × 8 × 256… Bias 1 × 1 × 256
85	dec_c2 48 1×1 convolutions with stride [1 1] and...	Convolution	180(S) × 240(S) × 48(C) × 1(B)	Weig… 1 × 1 × 64… Bias 1 × 1 × 48
86	dec_bn2 Batch normalization	Batch Normalization	180(S) × 240(S) × 48(C) × 1(B)	Offset 1 × 1 × 48 Scale 1 × 1 × 48
87	dec_relu2 ReLU	ReLU	180(S) × 240(S) × 48(C) × 1(B)	-
88	dec_crop1 center crop	Crop 2D	180(S) × 240(S) × 256(C) × 1(B)	-
89	dec_cat1 Depth concatenation of 2 inputs	Depth concatenation	180(S) × 240(S) × 304(C) × 1(B)	-
90	dec_c3 256 3×3 convolutions with stride [1 1] an...	Convolution	180(S) × 240(S) × 256(C) × 1(B)	Wei… 3 × 3 × 304… Bias 1 × 1 × 256
91	dec_bn3 Batch normalization	Batch Normalization	180(S) × 240(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
92	dec_relu3 ReLU	ReLU	180(S) × 240(S) × 256(C) × 1(B)	-
93	dec_c4 256 3×3 convolutions with stride [1 1] an...	Convolution	180(S) × 240(S) × 256(C) × 1(B)	Wei… 3 × 3 × 256… Bias 1 × 1 × 256
94	dec_bn4 Batch normalization	Batch Normalization	180(S) × 240(S) × 256(C) × 1(B)	Offset 1 × 1 × 256 Scale 1 × 1 × 256
95	dec_relu4 ReLU	ReLU	180(S) × 240(S) × 256(C) × 1(B)	-
96	scorer 11 1×1 convolutions with stride [1 1] and...	Convolution	180(S) × 240(S) × 11(C) × 1(B)	Weig… 1 × 1 × 256… Bias 1 × 1 × 11
97	dec_upsample2 11 8×8×11 transposed convolutions with...	Transposed Convol...	720(S) × 960(S) × 11(C) × 1(B)	Weig… 8 × 8 × 11… Bias 1 × 1 × 11
98	dec_crop2 center crop	Crop 2D	720(S) × 960(S) × 11(C) × 1(B)	-
99	softmax-out softmax	Softmax	720(S) × 960(S) × 11(C) × 1(B)	-
100	labels Class weighted cross-entropy loss with '...	Pixel Classification ...	720(S) × 960(S) × 11(C) × 1(B)	-

Semantic Segmentation Using Deep Learning

Select Training Options

```
% Define validation data.
dsVal = combine(imdsVal,pxdsVal);

% Define training options.
options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',10,...
    'LearnRateDropFactor',0.3,...
    'Momentum',0.9, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'ValidationData',dsVal,...
    'MaxEpochs',30, ...
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2,...
    'Plots','training-progress',...
    'ValidationPatience', 4);
```

Semantic Segmentation Using Deep Learning

Data Augmentation

Data augmentation is used to improve network accuracy by randomly transforming the original data during training. By using data augmentation, you can add more variety to the training data without increasing the number of labeled training samples. To apply the same random transformation to both image and pixel label data use datastore combine and transform. First, combine imdsTrain and pxdsTrain.

```
dsTrain = combine(imdsTrain, pxdsTrain);
```

Next, use datastore transform to apply the desired data augmentation defined in the supporting function augmentImageAndLabel. Here, random left/right reflection and random X/Y translation of +/- 10 pixels is used for data augmentation.

```
xTrans = [-10 10];
yTrans = [-10 10];
dsTrain = transform(dsTrain, @(data)augmentImageAndLabel(data,xTrans,yTrans));
```

Note that data augmentation is not applied to the test and validation data. Ideally, test and validation data should be representative of the original data and is left unmodified for unbiased evaluation.

Semantic Segmentation Using Deep Learning

Start Training

Start training using trainNetwork if the doTraining flag is true. Otherwise, load a pretrained network.

Note: The training was verified on an NVIDIA™ Titan X with 12 GB of GPU memory. If your GPU has less memory, you may run out of memory during training. If this happens, try setting 'MiniBatchSize' to 1 in trainingOptions, or reducing the network input and resizing the training data. Training this network takes about 70 minutes. Depending on your GPU hardware, it may take longer.

```
doTraining = false;
if doTraining
    [net, info] = trainNetwork(dsTrain,lgraph,options);
else
    pretrainedNetwork = fullfile(pretrainedFolder,'deeplabv3plusResnet18CamVid.mat');
    data = load(pretrainedNetwork);
    net = data.net;
end
```

Semantic Segmentation Using Deep Learning

Test Network on One Image

As a quick sanity check, run the trained network on one test image.

```
I = readimage(imdsTest,35);
C = semanticseg(I, net);
```

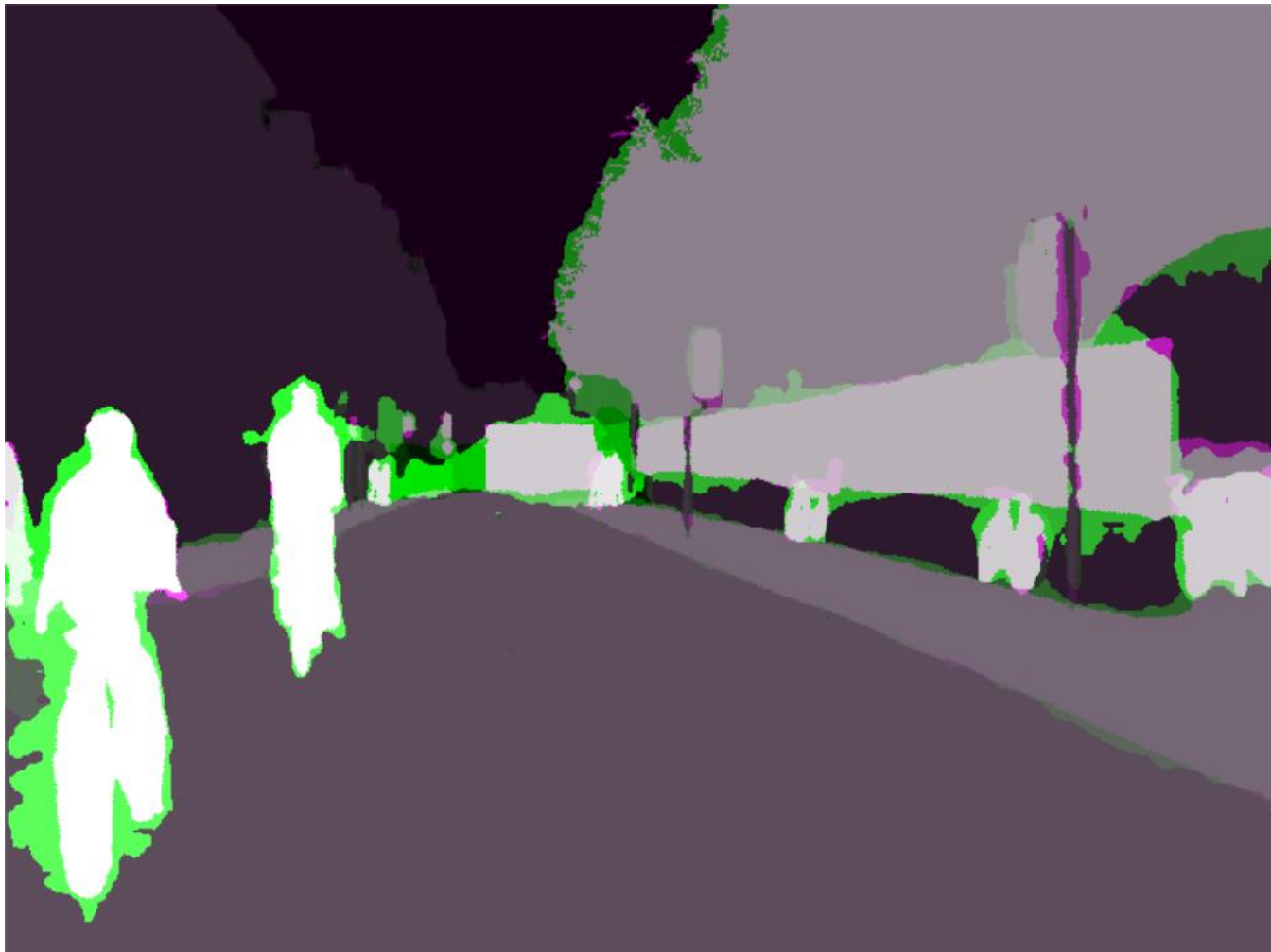
Display the results.

```
B = labeloverlay(I,C,'Colormap',cmap,'Transparency',0.4);
imshow(B)
pixelLabelColorbar(cmap, classes);
```

Semantic Segmentation Using Deep Learning

Compare the results in C with the expected ground truth stored in pxdsTest. The green and magenta regions highlight areas where the segmentation results differ from the expected ground truth.

```
expectedResult = readimage(pxdsTest,35);
actual = uint8(C);
expected = uint8(expectedResult);
imshowpair(actual, expected)
```



Semantic Segmentation Using Deep Learning

Visually, the semantic segmentation results overlap well for classes such as road, sky, and building. However, smaller objects like pedestrians and cars are not as accurate. The amount of overlap per class can be measured using the intersection-over-union (IoU) metric, also known as the Jaccard index. Use the jaccard function to measure IoU.

```
iou = jaccard(C,expectedResult);
table(classes,iou)
```

```
ans = 11×2 table
```

	classes	iou
1	"Sky"	0.9342
2	"Building"	0.8660
3	"Pole"	0.3752
4	"Road"	0.9452
5	"Pavement"	0.8542
6	"Tree"	0.9156
7	"SignSymbol"	0.6208
8	"Fence"	0.8108
9	"Car"	0.7145

Semantic Segmentation Using Deep Learning

Evaluate Trained Network

To measure accuracy for multiple test images, run semanticseg on the entire test set. A mini-batch size of 4 is used to reduce memory usage while segmenting images. You can increase or decrease this value based on the amount of GPU memory you have on your system.

```
pxdsResults = semanticseg(imdsTest,net, ...
    'MiniBatchSize',4, ...
    'WriteLocation',tempdir, ...
    'Verbose',false);
```

Semantic Segmentation Using Deep Learning

semanticseg returns the results for the test set as a pixelLabelDatastore object. The actual pixel label data for each test image in imdsTest is written to disk in the location specified by the 'WriteLocation' parameter. Use evaluateSemanticSegmentation to measure semantic segmentation metrics on the test set results.

```
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTest,'Verbose',false);
```

evaluateSemanticSegmentation returns various metrics for the entire dataset, for individual classes, and for each test image. To see the dataset level metrics, inspect metrics.DataSetMetrics .

```
metrics.DataSetMetrics
```

```
ans = 1×5 table
```

	GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
1	0.8924	0.8657	0.6635	0.8284	NaN

Semantic Segmentation Using Deep Learning

The dataset metrics provide a high-level overview of the network performance. To see the impact each class has on the overall performance, inspect the per-class metrics using metrics.ClassMetrics.

`metrics.ClassMetrics`

```
ans = 11×3 table
```

		Accuracy	IoU	MeanBFScore
1	Sky	0.9427	0.9098	NaN
2	Building	0.8149	0.7916	0.6396
3	Pole	0.7600	0.2463	NaN
4	Road	0.9395	0.9264	0.8061
5	Pavement	0.9005	0.7387	NaN
6	Tree	0.8817	0.7746	NaN
7	SignSymbol	0.7649	0.4234	NaN
8	Fence	0.8366	0.5744	NaN
9	Car	0.9259	0.7944	0.7433

Examples (ss2.m)

```
%FCL
openExample('deeplearning_shared/TrainAndDeployFullyConvolutionalNetworksExample')

%SegNet
openExample('vision/Ex66742723Example')
openExample('vision/Ex66594235Example')

%U-Net
openExample('vision/CreateUNetWithCustomEncoderDecoderDepthExample')
openExample('vision/TrainUNetExample')

%DeepLabV3
openExample('deeplearning_shared/SemanticSegmentationUsingDeepLearningExample')
```