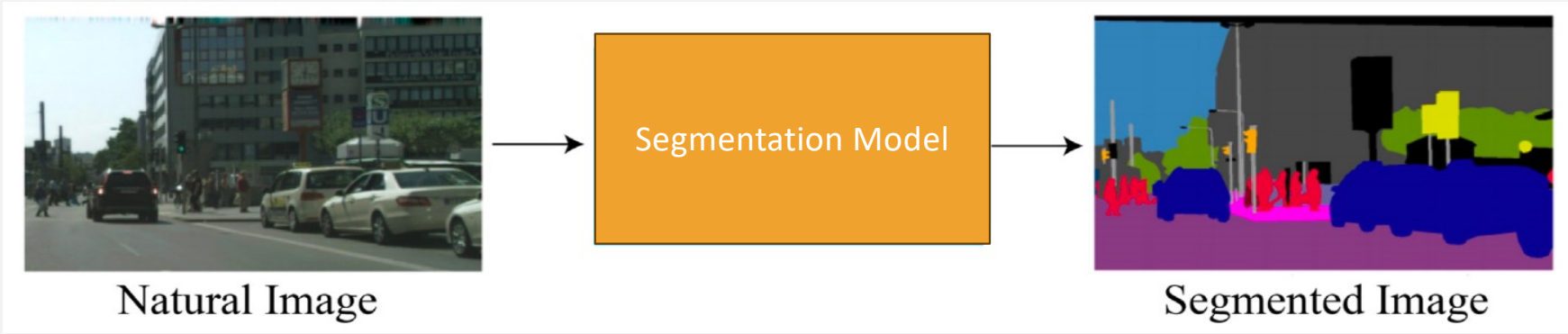
LECTURE 12 IMAGE SEGMENTATION II

Punnarai Siricharoen, Ph.D.

FROM TRADITION TO MODERN

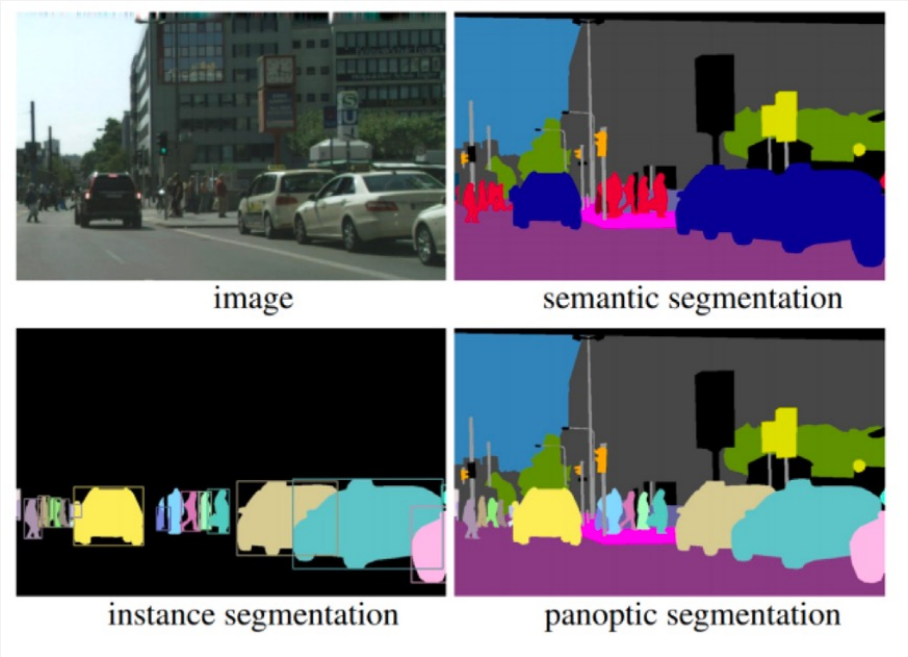
- Thresholding, Region growing, edge based detection
- Hand-crafted features (HOG & SIFT) and machine learning
- Deep learning (CNN)



Sultana, Farhana, Abu Sufian, and Paramartha Dutta. 2020. “Evolution of Image Segmentation Using Deep Convolutional Neural Network: A Survey.” Knowledge-Based Systems 201–202: 106062. https://doi.org/10.1016/j.knosys.2020.106062.

MODERN IMAGE SEGMENTATION

- Different types of image segmentation



Semantic segmentation - describes the process of associating each pixel of an image with a class label.

Instance segmentation - masks each instance of an object contained in an image independently

Panoptic segmentation - the combination of semantic segmentation and instance segmentation. The output of a panoptic segmentation model will contain

(1) pixel's label (semantic segmentation) and

(2) predicting each pixel instance (instance segmentation).

Sultana, Farhana, Abu Sufian, and Paramartha Dutta. 2020. "Evolution of Image Segmentation Using Deep Convolutional Neural Network: A Survey." Knowledge-Based Systems 201–202: 106062. https://doi.org/10.1016/j.knosys.2020.106062.

RECENT PROGRESS IN IMAGE SEGMENTATION

- Semantic image segmentation, also called pixel-level classification, is the task of clustering parts of image together which belong to the same object class

- PASCAL challenge
  - Segmentation Competition - Generating pixel-wise segmentations giving the class of the object visible at each pixel, or "background" otherwise

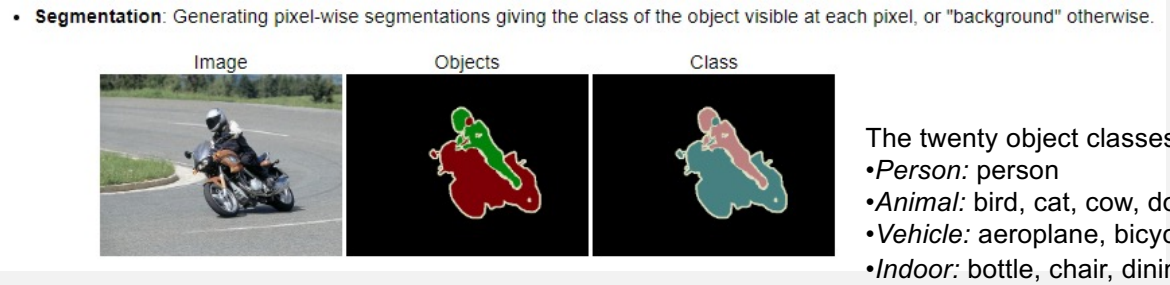


Fig. 1 Motorcycle racing image

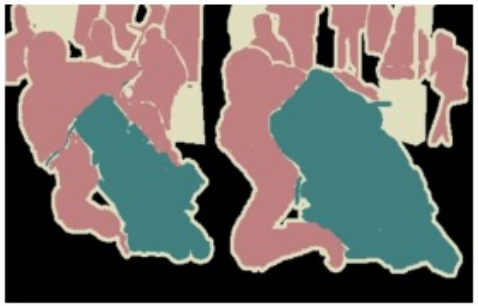


Fig. 2 Segmentation for motorcycle racing image

The twenty object classes that have been selected are:
- *Person:* person
- *Animal:* bird, cat, cow, dog, horse, sheep
- *Vehicle:* aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor:* bottle, chair, dining table, potted plant, sofa, tv/monitor

http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html

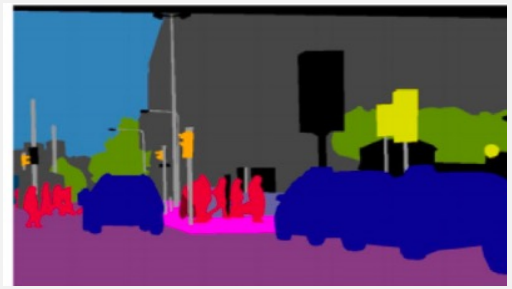
DATASET AVAILABILITY

	Number of Class	Number of Images	Description
Cityscapes	30 / 8 category	25K	semantic understanding of urban street scenes with large number of dynamic objects, varying scene layout, and varying background. 50 cities,5000 fine annotated images and 20000 coarse annotated ones. (flat surfaces, humans, vehicles, constructions, objects, nature, sky, and void)
ADE20K	150	20K	scene-centric images including stuffs like sky, road, grass, and discrete objects like person, car, bed
KITTI	11	323 (max)	traffic scenarios recorded with a variety of sensor modalities, including high-resolution RGB, grayscale stereo cameras, and a 3D laser scanner (building, tree, sky, car, sign, road, pedestrian, fence, pole, sidewalk, and bicyclist)
CamVid	32	701 (frames)	a road/driving scene understanding database which was originally captured as five video sequences with a 960×720 resolution camera mounted on the dashboard of a car. 32 classes include void, building, wall, tree, vegetation, fence, sidewalk, parking block, etc.
PASCAL VOC	20	3K	20 object categories including vehicles, household, animals, and other: aeroplane, bicycle, boat, bus, car, motorbike, train, etc.
COCO-Stuff	172	164k	scene understanding tasks like semantic segmentation, object detection and image captioning. 172 categories, 80 things, 91 stuff, and 1 unlabeled class

FROM TRADITION TO MODERN

- Training & Testing process

image

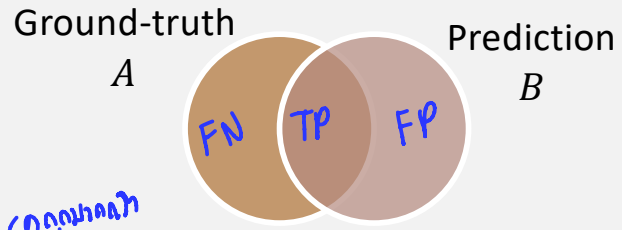


Label

Reference: http://www.theobjects.com/dragonfly/dfhelp/2020-1/Content/Artificial%20Intelligence/Deep%20Learning%20Tool/Workflow%20and%20Data%20Preparation.htm

SEGMENTATION PERFORMANCE

- Intersection over Union (IoU) - Intersection over Union indicates the overlap of the predicted bounding box coordinates to the ground truth box. Higher IoU indicates the predicted bounding box coordinates closely resembles the ground truth box coordinates.



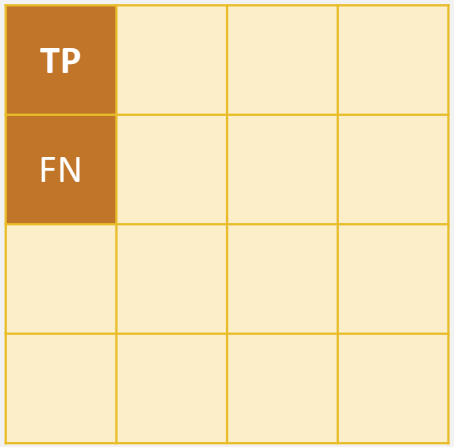
$$IoU = \frac{A \cap B}{A \cup B}$$

- Dice score – is very similar to IoU

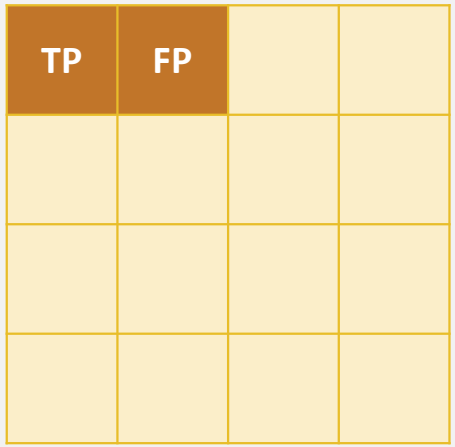
$$Dice\ score = 2 \cdot \frac{A \cap B}{|A| + |B|} = 2 \cdot \frac{TP}{2\ TP + FP + FN}$$

- Pixel Accuracy / Precision / Recall
- Object detection / Instance segmentation
  - mAP, see https://www.v7labs.com/blog/mean-average-precision / Precision / Recall (Object level)

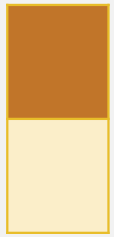
EVALUATION – PIXEL ACCURACY / PRECISION / RECALL



Ground-truth



Prediction



Class A

Background

Confusion matrix

Actual \ prediction	Class A	BG
Class A	TP 1	FN 1
BG	FP 1	TN 13

$$IoU_A = \frac{1}{1+1+1} = 33.3\% \qquad DSC_A = \frac{2*1}{2*1+1+1} = 50\%$$

$$mIoU = \frac{1}{2}\{IoU_A + IoU_{bg}\} = \frac{1}{2}\left(\frac{1}{3} + \frac{13}{15}\right) = 60\%$$

Class	Precision	Recall	F1
Class A	50%	50%	50%
BG	92.8%	92.8%	92.8%

$$Acc = \frac{correctness}{all} = \frac{14}{16} = 87.5\%$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Macro-F1 = 71.4% : No weight

Micro-F1 = 87.45% : with weight

MODERN METHODS (DEEP LEARNING)

20 classes

Accuracies of segmentation models on the PASCAL VOC test set

Method	Backbone	mIoU
FCN [30]	VGG-16	62.2
CRF-RNN [38]	-	72.0
CRF-RNN* [38]	-	74.7
BoxSup* [119]	-	75.1
Piecewise* [39]	-	78.0
DPN* [40]	-	77.5
DeepLab-CRF [76]	ResNet-101	79.7
GCN* [120]	ResNet-152	82.2
Dynamic Routing [142]	-	84.0
RefineNet [117]	ResNet-152	84.2
Wide ResNet [121]	WideResNet-38	84.9
PSPNet [54]	ResNet-101	85.4
DeeplabV3 [13]	ResNet-101	85.7
PSANet [98]	ResNet-101	85.7
EncNet [116]	ResNet-101	85.9
DFN* [99]	ResNet-101	86.2
Exfuse [122]	ResNet-101	86.2
SDN* [43]	DenseNet-161	86.6
DIS [125]	ResNet-101	86.8
APC-Net* [58]	ResNet-101	87.1
EMANet [95]	ResNet-101	87.7
DeeplabV3+ [81]	Xception-71	87.8
Exfuse [122]	ResNeXt-131	87.9
MSCI [59]	ResNet-152	88.0
EMANet [95]	ResNet-152	88.2
DeeplabV3+* [81]	Xception-71	89.0
EfficientNet+NAS-FPN [137]	-	90.5

30 classes

Accuracies of segmentation models on the Cityscapes dataset

Method	Backbone	mIoU
SegNet [25]	-	57.0
FCN-8s [30]	-	65.3
DPN [40]	-	66.8
Dilation10 [77]	-	67.1
DeeplabV2 [76]	ResNet-101	70.4
RefineNet [117]	ResNet-101	73.6
FoveaNet [126]	ResNet-101	74.1
Ladder DenseNet [127]	Ladder DenseNet-169	73.7
GCN [120]	ResNet-101	76.9
DUC-HDC [78]	ResNet-101	77.6
Wide ResNet [121]	WideResNet-38	78.4
PSPNet [54]	ResNet-101	85.4
BiSeNet [128]	ResNet-101	78.9
DFN [99]	ResNet-101	79.3
PSANet [98]	ResNet-101	80.1
DenseASPP [79]	DenseNet-161	80.6
Dynamic Routing [142]	-	80.7
SPGNet [129]	2xResNet-50	81.1
DANet [91]	ResNet-101	81.5
CCNet [96]	ResNet-101	81.4
DeeplabV3 [13]	ResNet-101	81.3
IPC [141]	ResNet-101	81.8
AC-Net [131]	ResNet-101	82.3
OCR [42]	ResNet-101	82.4
ResNeSt200 [93]	ResNeSt-200	82.7
GS-CNN [130]	WideResNet	82.8
HA-Net [94]	ResNext-101	83.2
HRNetV2+OCR [42]	HRNetV2-W48	83.7
Hierarchical MSA [139]	HRNet-OCR	85.1

91 class

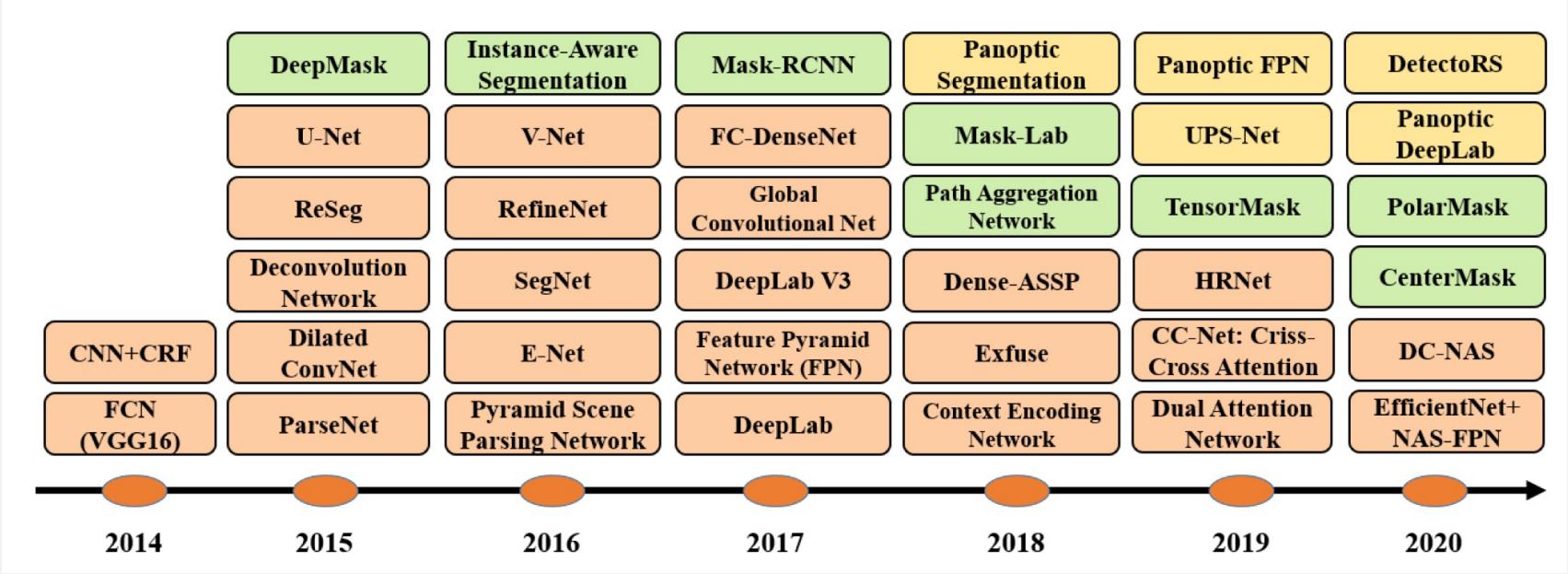
Accuracies of segmentation models on the MS COCO stuff dataset

Method	Backbone	mIoU
RefineNet [117]	ResNet-101	33.6
CCN [57]	Ladder DenseNet-101	35.7
DANet [91]	ResNet-50	37.9
DSSPN [132]	ResNet-101	37.3
EMA-Net [95]	ResNet-50	37.5
SGR [133]	ResNet-101	39.1
OCR [42]	ResNet-101	39.5
DANet [91]	ResNet-101	39.7
EMA-Net [95]	ResNet-50	39.9
AC-Net [131]	ResNet-101	40.1
OCR [42]	HRNetV2-W48	40.5

Instance segmentation model performance on COCO test-dev 2017

Method	Backbone	FPS	AP
YOLACT-550 [74]	R-101-FPN	33.5	29.8
YOLACT-700 [74]	R-101-FPN	23.8	31.2
RetinaMask [172]	R-101-FPN	10.2	34.7
TensorMask [67]	R-101-FPN	2.6	37.1
SharpMask [173]	R-101-FPN	8.0	37.4
Mask-RCNN [62]	R-101-FPN	10.6	37.9
CenterMask [72]	R-101-FPN	13.2	38.3

MODERN METHODS (DEEP LEARNING)



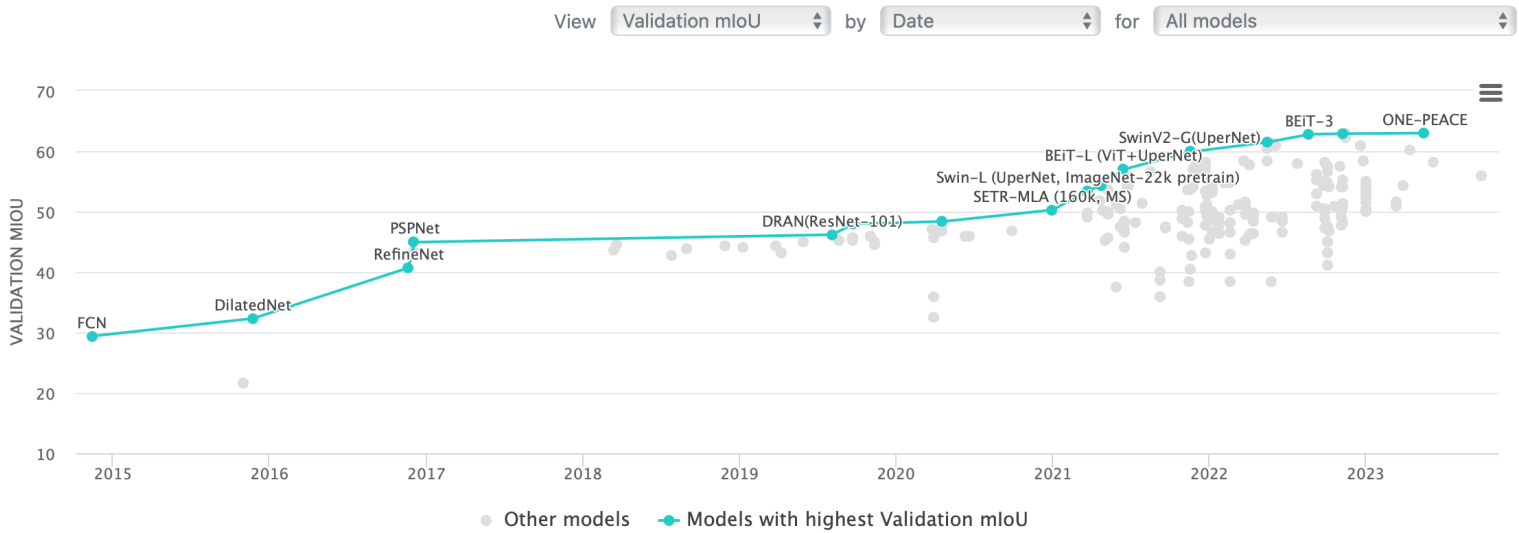
Timeline of representative DL-based image segmentation algorithms. **Orange**, **green**, and **yellow** blocks indicate semantic, instance, and panoptic segmentation algorithms, respectively.

References: Minaee S. et al., Image segmentation using Deep Learning : Survey, PAMI 2022

MODERN METHODS (DEEP LEARNING)

Semantic Segmentation on ADE20K

Leaderboard Dataset



Filter: Transformer | CNN | Swin-Transformer | Neighborhood Attention | ResNet | NAT Transformer | multiscale | ViT-Giant | Ensemble | Focal-Transformer | HRNet | ResNet-101 | DeepLab v3+ | DCN | Deformable Convolution | untagged

Edit Leaderboard

Rank	Model	Validation mIoU	Test Score
1	**ONE-PEACE**	63.0	
2	**InternImage-H**	62.9	
3	**M3I Pre-training** (InternImage-H)	62.9	
4	**BEiT-3**	62.8	
5	**EVA**	62.3	

https://paperswithcode.com/sota/semantic-segmentation-on-ade20k

FCN – FULLY CONVOLUTIONAL NETWORKS

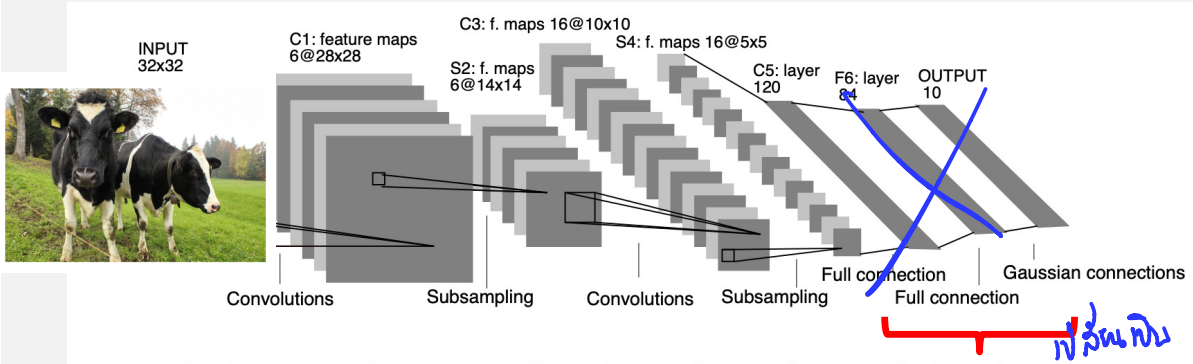


Fig. 1. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units who rained to be identical.

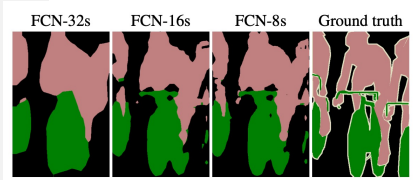
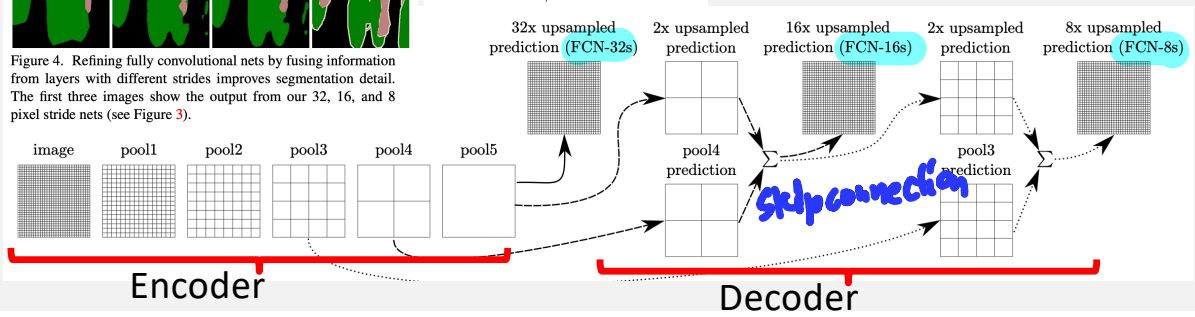


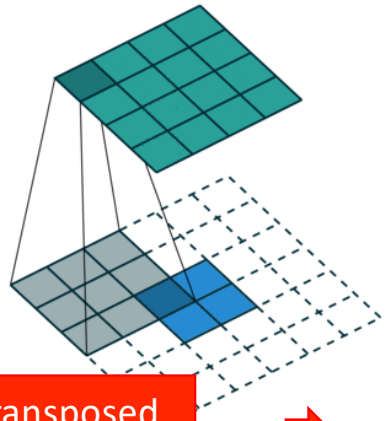
Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

	pixel acc.	mean acc.	mean IU
FCN-32s-fixed	83.0	59.7	45.4
FCN-32s	89.1	73.3	59.4
FCN-16s	90.0	75.7	62.4
FCN-8s	**90.3**	**75.9**	**62.7**



1x1 convolution → (num_class, kernel_size = 1)

Transposed convolution



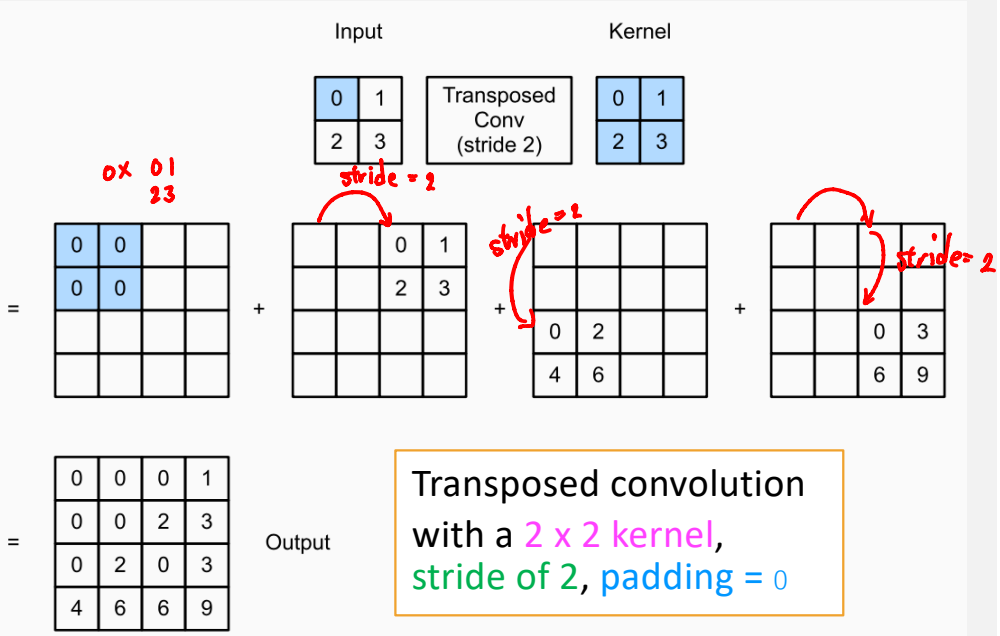
```
nn.ConvTranspose2d
(in_c, out_c,
kernel_size=2,
stride=2,
padding=0)
```

- Transposed convolution / upsampling / deconvolution – output is larger (upsampling, deconvolution) by padding the image until kernel corner can just barely reach the corner of the input.

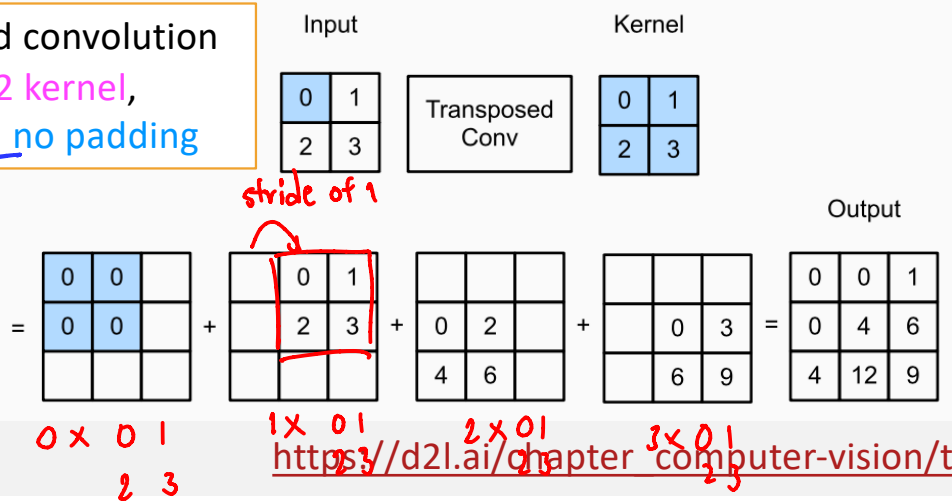
https://colab.research.google.com/drive/1K6-olgVRWCkSE-_KSZu7_xElUyUlThEZ?usp=sharing

FCN – FULLY CONVOLUTIONAL NETWORKS

- Transposed convolution
  - **Kernel** - size : learnable params
  - **Stride** – number of move for the output (not input)
  - **Padding** – applied to input

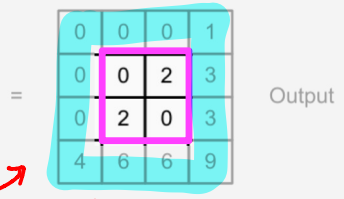


Transposed convolution with a 2 x 2 kernel, stride of 1, no padding



Transposed convolution with a 2 x 2 kernel, stride of 2, padding = 0

Transposed convolution with a 2 x 2 kernel, stride of 2, padding = 1



https://d2l.ai/chapter_computer-vision/transposed-conv.html

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 10 classes
        self.out = nn.Linear(32 * 7 * 7, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        output = F.log_softmax(self.out(x), dim=1)
        return output
```

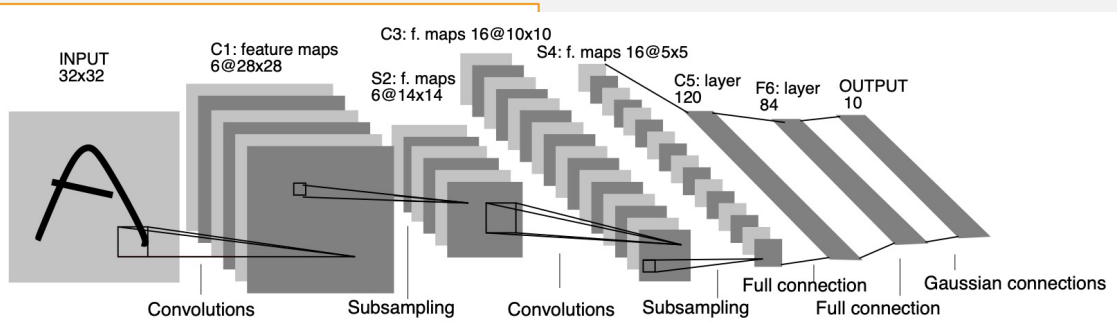


Fig. 1. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

REVISE : CNN

https://colab.research.google.com/drive/1K6-olgVRWCkSE-_KSZu7_xElUyUlThEZ?usp=sharing

U-NET

- U-Net is a U-shaped encoder-decoder network architecture, which consists of four encoder blocks and four decoder blocks that are connected via a bridge.

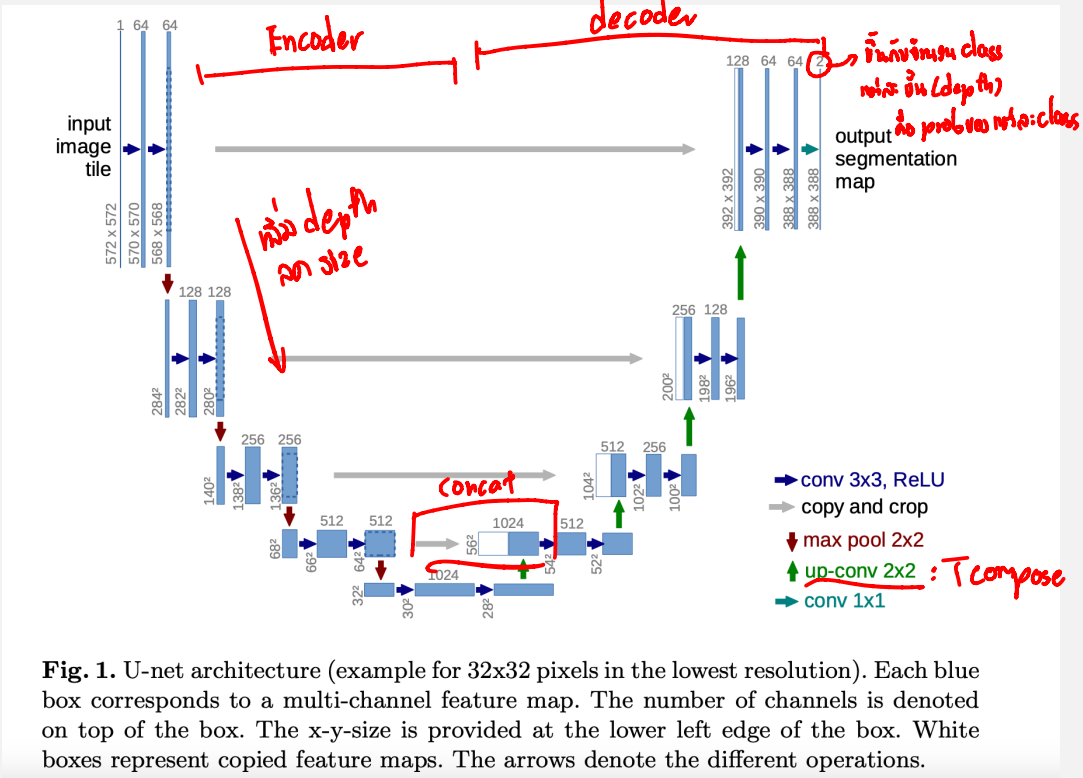


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

U-NET

- U-Net is a U-shaped encoder-decoder network architecture, which consists of four encoder blocks and four decoder blocks that are connected via a bridge.

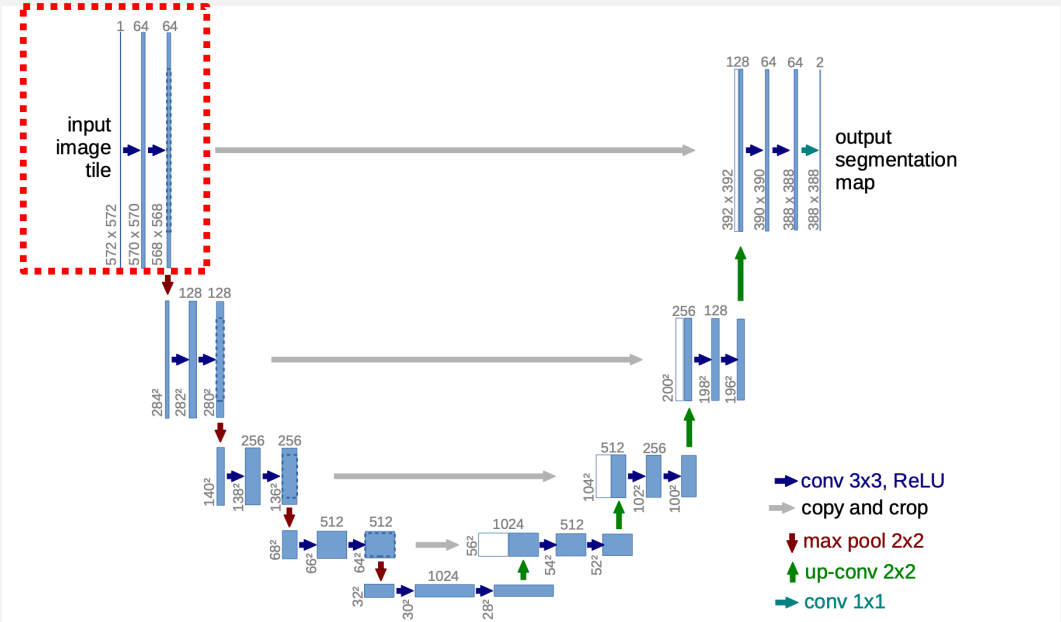


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

```
class conv_block(nn.Module):
  def __init__(self, in_c, out_c):
    super().__init__()
    self.conv1 = nn.Conv2d(in_c, out_c,
        kernel_size=3, padding=1)
    self.bn1 = nn.BatchNorm2d(out_c)
    self.conv2 = nn.Conv2d(out_c, out_c,
        kernel_size=3, padding=1)
    self.bn2 = nn.BatchNorm2d(out_c)
    self.relu = nn.ReLU()
  def forward(self, inputs):
    x = self.conv1(inputs)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.conv2(x)
    x = self.bn2(x)
    x = self.relu(x)
    return x
```

U-NET

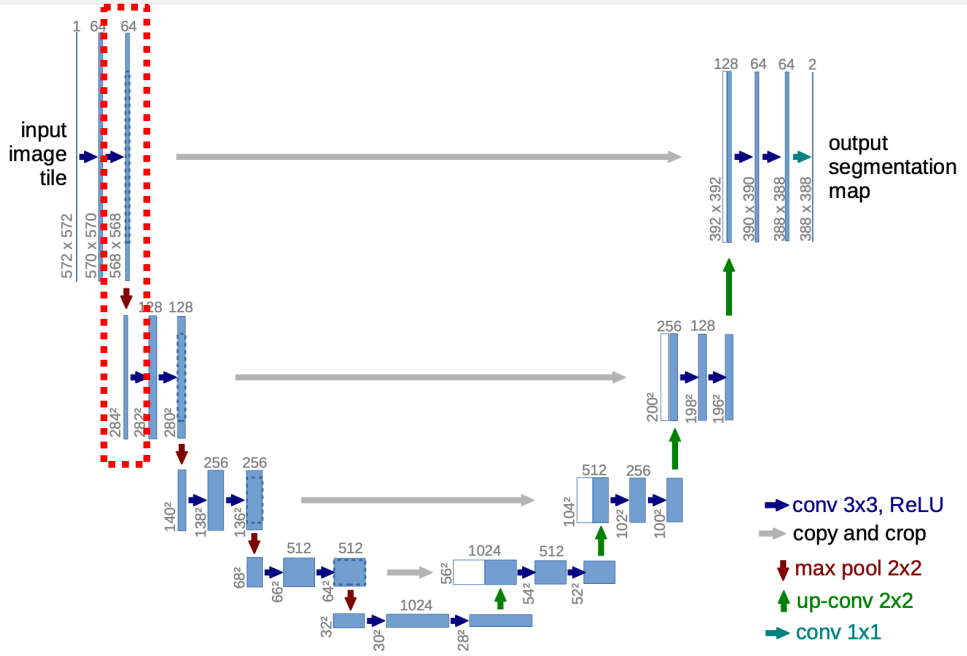


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

- The encoder network - half the spatial dimensions and double the number of filters (feature channels) at each encoder block.

```
class encoder_block(nn.Module):
  def __init__(self, in_c, out_c):
    super().__init__()
    self.conv = conv_block(in_c, out_c)
    self.pool = nn.MaxPool2d((2, 2))
  def forward(self, inputs):
    x = self.conv(inputs)
    p = self.pool(x)
    return x, p
```

U-NET

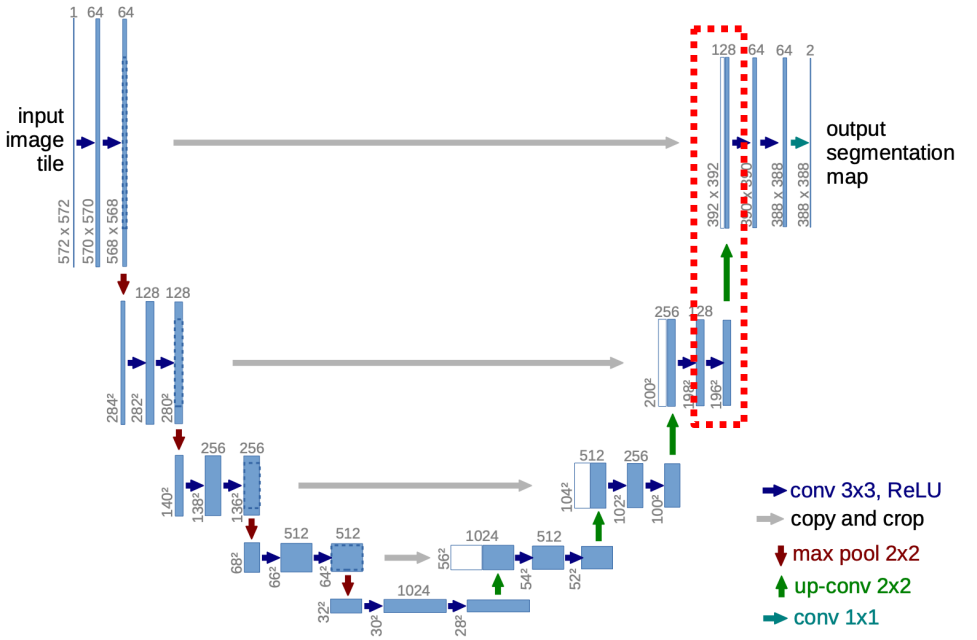


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

- The decoder network doubles the spatial dimensions and half the number of feature channels.

```
class decoder_block(nn.Module):
 def __init__(self, in_c, out_c):
    super().__init__()
    self.up = nn.ConvTranspose2d(in_c, out_c,
    kernel_size=2, stride=2, padding=0)
    self.conv = conv_block(out_c+out_c, out_c)
 def forward(self, inputs, skip):
    x = self.up(inputs)
    x = torch.cat([x, skip], axis=1)
    x = self.conv(x)
    return x
```

U-NET

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
       BatchNorm2d-2         [-1, 64, 224, 224]             128
              ReLU-3         [-1, 64, 224, 224]               0
            Conv2d-4         [-1, 64, 224, 224]          36,928
       BatchNorm2d-5         [-1, 64, 224, 224]             128
              ReLU-6         [-1, 64, 224, 224]               0
        conv_block-7         [-1, 64, 224, 224]               0
         MaxPool2d-8         [-1, 64, 112, 112]               0
  encoder_block-9  [[-1, 64, 224, 224], [-1, 64, 112, 112]]               0
           Conv2d-10        [-1, 128, 112, 112]          73,856
      BatchNorm2d-11        [-1, 128, 112, 112]             256
             ReLU-12        [-1, 128, 112, 112]               0
           Conv2d-13        [-1, 128, 112, 112]         147,584
      BatchNorm2d-14        [-1, 128, 112, 112]             256
             ReLU-15        [-1, 128, 112, 112]               0
       conv_block-16        [-1, 128, 112, 112]               0
        MaxPool2d-17          [-1, 128, 56, 56]               0
 encoder_block-18  [[-1, 128, 112, 112], [-1, 128, 56, 56]]               0
           Conv2d-19          [-1, 256, 56, 56]         295,168
      BatchNorm2d-20          [-1, 256, 56, 56]             512
             ReLU-21          [-1, 256, 56, 56]               0
           Conv2d-22          [-1, 256, 56, 56]         590,080
      BatchNorm2d-23          [-1, 256, 56, 56]             512
             ReLU-24          [-1, 256, 56, 56]               0
       conv_block-25          [-1, 256, 56, 56]               0
        MaxPool2d-26          [-1, 256, 28, 28]               0
 encoder_block-27  [[-1, 256, 56, 56], [-1, 256, 28, 28]]               0
```

...

```
  ConvTranspose2d-62        [-1, 128, 112, 112]         131,200
           Conv2d-63        [-1, 128, 112, 112]         295,040
      BatchNorm2d-64        [-1, 128, 112, 112]             256
             ReLU-65        [-1, 128, 112, 112]               0
           Conv2d-66        [-1, 128, 112, 112]         147,584
      BatchNorm2d-67        [-1, 128, 112, 112]             256
             ReLU-68        [-1, 128, 112, 112]               0
       conv_block-69        [-1, 128, 112, 112]               0
  decoder_block-70        [-1, 128, 112, 112]               0
  ConvTranspose2d-71         [-1, 64, 224, 224]          32,832
           Conv2d-72         [-1, 64, 224, 224]          73,792
      BatchNorm2d-73         [-1, 64, 224, 224]             128
             ReLU-74         [-1, 64, 224, 224]               0
           Conv2d-75         [-1, 64, 224, 224]          36,928
      BatchNorm2d-76         [-1, 64, 224, 224]             128
             ReLU-77         [-1, 64, 224, 224]               0
       conv_block-78         [-1, 64, 224, 224]               0
  decoder_block-79         [-1, 64, 224, 224]               0
           Conv2d-80          [-1, 1, 224, 224]              65
      build_unet-81          [-1, 1, 224, 224]               0
================================================================
Total params: 31,043,521
Trainable params: 31,043,521
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 26122122.03
Params size (MB): 118.42
Estimated Total Size (MB): 26122241.03
----------------------------------------------------------------
```

```python
class build_unet(nn.Module):
    def __init__(self):
        super().__init__()
        """ Encoder """
        self.e1 = encoder_block(3, 64)
        self.e2 = encoder_block(64, 128)
        self.e3 = encoder_block(128, 256)
        self.e4 = encoder_block(256, 512)
        """ Bottleneck """
        self.b = conv_block(512, 1024)
        """ Decoder """
        self.d1 = decoder_block(1024, 512)
        self.d2 = decoder_block(512, 256)
        self.d3 = decoder_block(256, 128)
        self.d4 = decoder_block(128, 64)
        """ Classifier """
        self.outputs = nn.Conv2d(64, 1, kernel_size=1, padding=0)
    def forward(self, inputs):
        """ Encoder """
        s1, p1 = self.e1(inputs)
        s2, p2 = self.e2(p1)
        s3, p3 = self.e3(p2)
        s4, p4 = self.e4(p3)
        """ Bottleneck """
        b = self.b(p4)
        """ Decoder """
        d1 = self.d1(b, s4)
        d2 = self.d2(d1, s3)
        d3 = self.d3(d2, s2)
        d4 = self.d4(d3, s1)
        """ Classifier """
        outputs = self.outputs(d4)
        return outputs
```

U-NET

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
       BatchNorm2d-2         [-1, 64, 224, 224]             128
              ReLU-3         [-1, 64, 224, 224]               0
            Conv2d-4         [-1, 64, 224, 224]          36,928
       BatchNorm2d-5         [-1, 64, 224, 224]             128
              ReLU-6         [-1, 64, 224, 224]               0
        conv_block-7         [-1, 64, 224, 224]               0
         MaxPool2d-8         [-1, 64, 112, 112]               0
  encoder_block-9  [[-1, 64, 224, 224], [-1, 64, 112, 112]]              0
           Conv2d-10        [-1, 128, 112, 112]          73,856
      BatchNorm2d-11        [-1, 128, 112, 112]             256
             ReLU-12        [-1, 128, 112, 112]               0
           Conv2d-13        [-1, 128, 112, 112]         147,584
      BatchNorm2d-14        [-1, 128, 112, 112]             256
             ReLU-15        [-1, 128, 112, 112]               0
       conv_block-16        [-1, 128, 112, 112]               0
        MaxPool2d-17          [-1, 128, 56, 56]               0
 encoder_block-18  [[-1, 128, 112, 112], [-1, 128, 56, 56]]             0
           Conv2d-19          [-1, 256, 56, 56]         295,168
      BatchNorm2d-20          [-1, 256, 56, 56]             512
             ReLU-21          [-1, 256, 56, 56]               0
           Conv2d-22          [-1, 256, 56, 56]         590,080
      BatchNorm2d-23          [-1, 256, 56, 56]             512
             ReLU-24          [-1, 256, 56, 56]               0
       conv_block-25          [-1, 256, 56, 56]               0
        MaxPool2d-26          [-1, 256, 28, 28]               0
 encoder_block-27  [[-1, 256, 56, 56], [-1, 256, 28, 28]]              0
                                  ...
  ConvTranspose2d-62        [-1, 128, 112, 112]         131,200
           Conv2d-63        [-1, 128, 112, 112]         295,040
      BatchNorm2d-64        [-1, 128, 112, 112]             256
             ReLU-65        [-1, 128, 112, 112]               0
           Conv2d-66        [-1, 128, 112, 112]         147,584
      BatchNorm2d-67        [-1, 128, 112, 112]             256
             ReLU-68        [-1, 128, 112, 112]               0
       conv_block-69        [-1, 128, 112, 112]               0
  decoder_block-70        [-1, 128, 112, 112]               0
  ConvTranspose2d-71         [-1, 64, 224, 224]          32,832
           Conv2d-72         [-1, 64, 224, 224]          73,792
      BatchNorm2d-73         [-1, 64, 224, 224]             128
             ReLU-74         [-1, 64, 224, 224]               0
           Conv2d-75         [-1, 64, 224, 224]          36,928
      BatchNorm2d-76         [-1, 64, 224, 224]             128
             ReLU-77         [-1, 64, 224, 224]               0
       conv_block-78         [-1, 64, 224, 224]               0
  decoder_block-79         [-1, 64, 224, 224]               0
           Conv2d-80          [-1, 1, 224, 224]              65
      build_unet-81          [-1, 1, 224, 224]               0
================================================================
Total params: 31,043,521
Trainable params: 31,043,521
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 26122122.03
Params size (MB): 118.42
Estimated Total Size (MB): 26122241.03
----------------------------------------------------------------
```

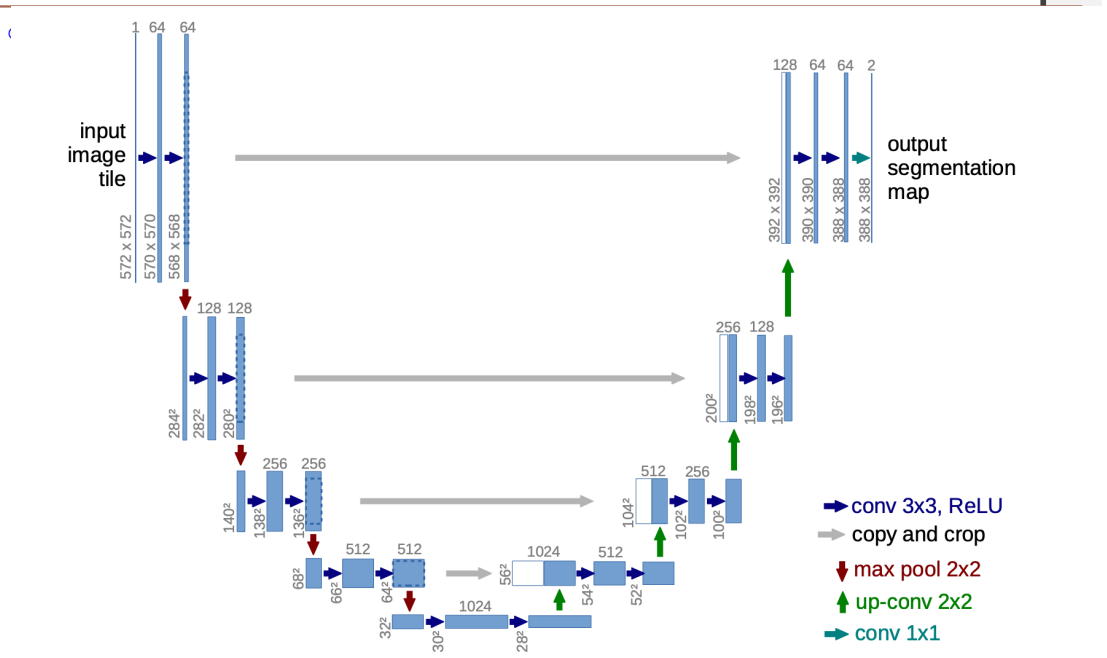


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

```
        """ Decoder """
        d1 = self.d1(b, s4)
        d2 = self.d2(d1, s3)
        d3 = self.d3(d2, s2)
        d4 = self.d4(d3, s1)
        """ Classifier """
        outputs = self.outputs(d4)
        return outputs
```

U-NET – LOSS FUNCTIONS

- **Dice loss** – we use *1 – Dice score* to maximize the overlap between two sets
- **Binary cross-entropy with Logitsloss** - This loss combines a *Sigmoid* layer and the *BCELoss* in one single class. This version is more numerically stable than using a plain *Sigmoid* followed by a *BCELoss* as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.
  - It's possible to trade off recall and precision by adding weights to positive examples. In the case of multi-label classification the loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right]$$ for binary-class

$$L_c = \{l_{1,c}, \dots, l_{N,c}\}^\top, \quad l_{n,c} = -w_{n,c} \left[p_c y_{n,c} \cdot \log \sigma(x_{n,c}) + (1 - y_{n,c}) \cdot \log(1 - \sigma(x_{n,c})) \right]$$ for multi-class

https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

U-NET

- Training model using two losses.

```
class SegmentationModel(nn.Module):
  def __init__(self):
    super(SegmentationModel,self).__init__()
    self.arc = build_unet()

  def forward(self, images, masks = None):
    logits = self.arc(images)

    if masks != None:
      loss1 = DiceLoss(mode="binary")(logits,masks)
      loss2 = nn.BCEWithLogitsLoss()(logits,masks)
      return logits, loss1 + loss2

    return logits
```

PRE-TRAINED MODELS

pretrained backbone to extract features of different spatial resolution

- Fine-tuning the pretrained model with your own dataset.
- Select the backbone : https://segmentation-modelspytorch.readthedocs.io/en/latest/

Encoders

Encoder	Weights
resnet18	imagenet
resnet34	imagenet
resnet50	imagenet
resnet101	imagenet
resnet152	imagenet
resnext50_32x4d	imagenet
resnext101_32x8d	imagenetinstagram
resnext101_32x16d	instagram
resnext101_32x32d	instagram
resnext101_32x48d	instagram
dpn68	imagenet

Decoder - depends on models architecture (Unet/Linknet/PSPNet/FPN)

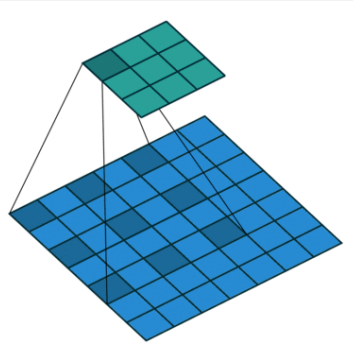
```
ENCODER = 'vgg16'
WEIGHTS = 'imagenet'
```

```
import segmentation_models_pytorch as smp

class SegmentationModel(nn.Module):
  def __init__(self):
    super(SegmentationModel,self).__init__()

    self.arc = smp.Unet(
        encoder_name = ENCODER, :classification
        encoder_weights = WEIGHTS,
        in_channels = 3,
        classes = 1,
        activation = None
    )
  def forward(self, images, masks = None):
    logits = self.arc(images)

    if masks != None:
      loss1 = DiceLoss(mode="binary")(logits,masks)
      loss2 = nn.BCEWithLogitsLoss()(logits,masks)
      return logits, loss1 + loss2

    return logits
```

DEEPLAB

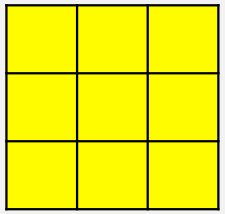
- Developed and open-sourced by Google
- The dense prediction is achieved by simply up-sampling the output of the last convolution layer and computing pixel-wise loss
- Deeplab applies **atrous convolution** for up-sample to compute feature maps in higher sampling rate.
- capture fine details by employing a fully connected Conditional Random Field (CRF)

DEEPLAB

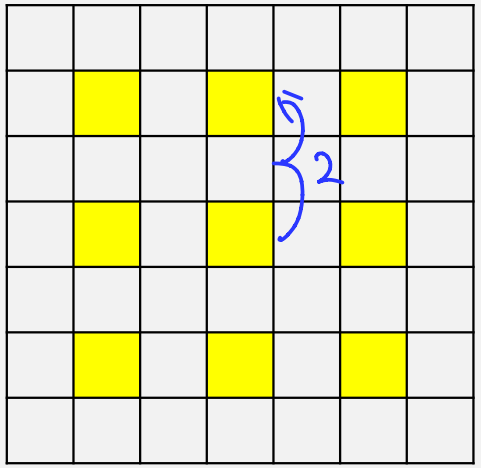


https://towardsdatascience.com/a-primer-on-atrous-convolutions-and-depth-wise-separable-convolutions-443b106919f5

- DeepLabv1 – Dense Coord-Convolution Network (DCNN) followed by Atrous convolution
- Atrous / Dilated Convolution
  - Max Pool & Stride reduces spatial resolution of the feature map
  - Expands receptive field (size of the region of the input feature map – filter size) without downsampling
  - Allow conv filter to look larger area in input w/o decrease in spatial resolution or increase kernel size
  - Maintains input spatial dimensions
  - Captures large context in high-res segmentation
  - Parameter : dilation rate (r)



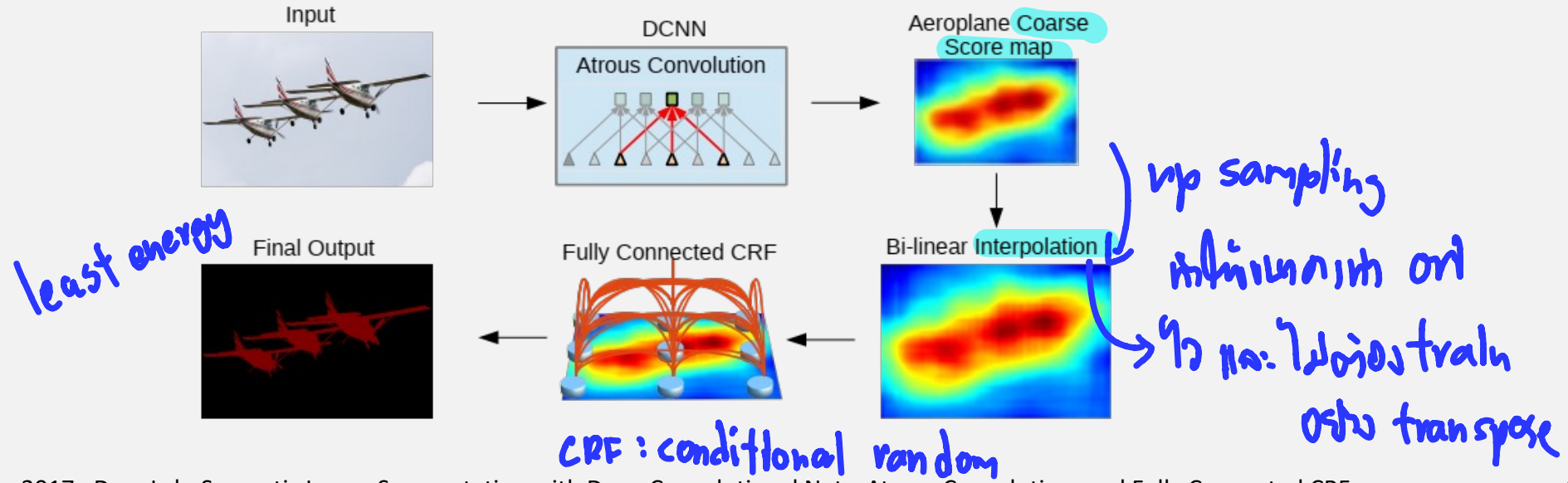
3x3 Kernel



3x3 Dilated kernel with Dilation rate = 2

DEEPLAB

- DeepLabv1 - up-sampled to the original size of the image, using bi-linear interpolation. Finally, to improve the segmentation result fully connected CRF is applied

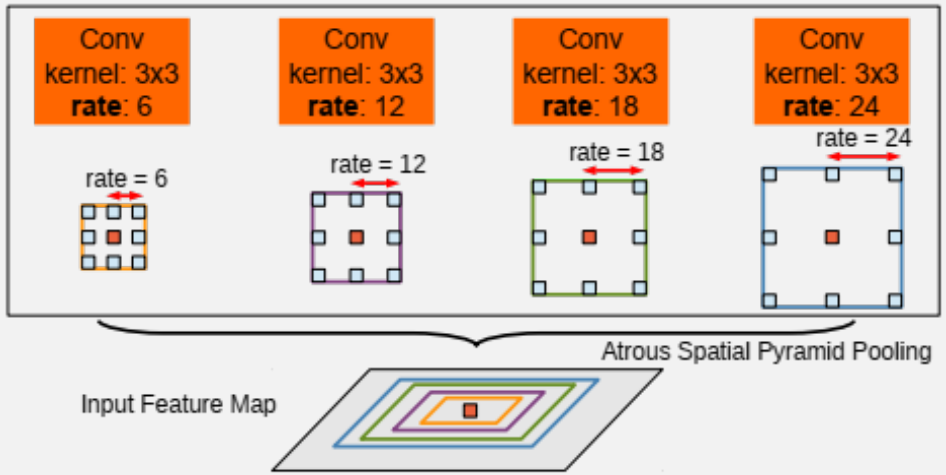


Chen et al., 2017 : DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs

DEEPLAB

- DeepLabv2 - Use of **Atrous Spatial Pyramid Pooling (ASPP)** to handle object in multiple scales
  - **Parallel Atrous Convolutions** with different dilation rates.
  - **Global Average Pooling** to capture global context, followed by upsampling.
  - **Concatenate** the outputs of all layers

  along the channel dimension. This concatenated output has features from each atrous convolution rate, capturing a wide range of context for more precise segmentation



Chen et al., 2017 : DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs

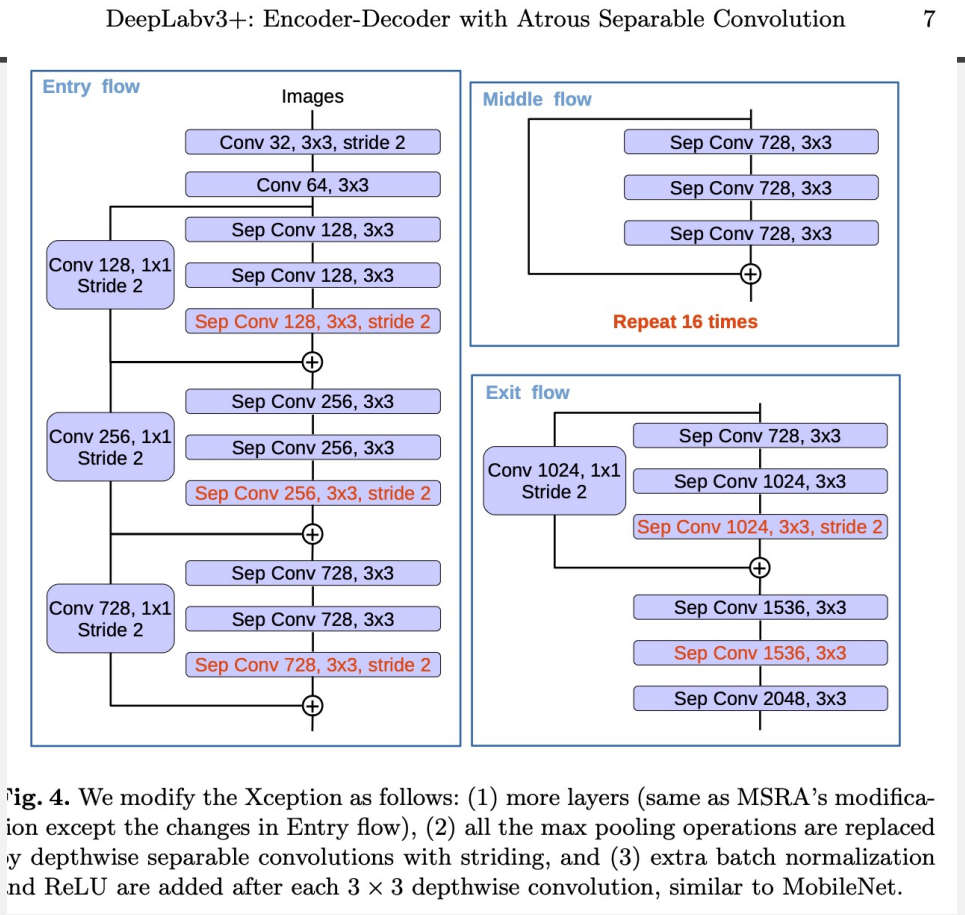
DEEPLAB

- DeepLabv3
  - The encoder-decoder model is able to obtain sharp object boundaries.
    - An encoder module that gradually reduces the feature maps and captures higher semantic information.
    - A decoder module that gradually recovers the spatial information.
    - **depth-wise separable convolution** to increases computational efficiency - a depth-wise convolution followed by a point-wise convolution (i.e., 1 × 1 convolution).

Chen et al., 2017 : DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs

DEEPLAB

- DeepLabv3+
  - Adding a simple but effective decoder module to further refine the segmentation results especially along object boundaries.
  - Modify the Aligned Xception instead of ResNet-101 as its main feature extractor (encoder) to be deeper. All max pooling operations are replaced by depth-wise separable convolution.

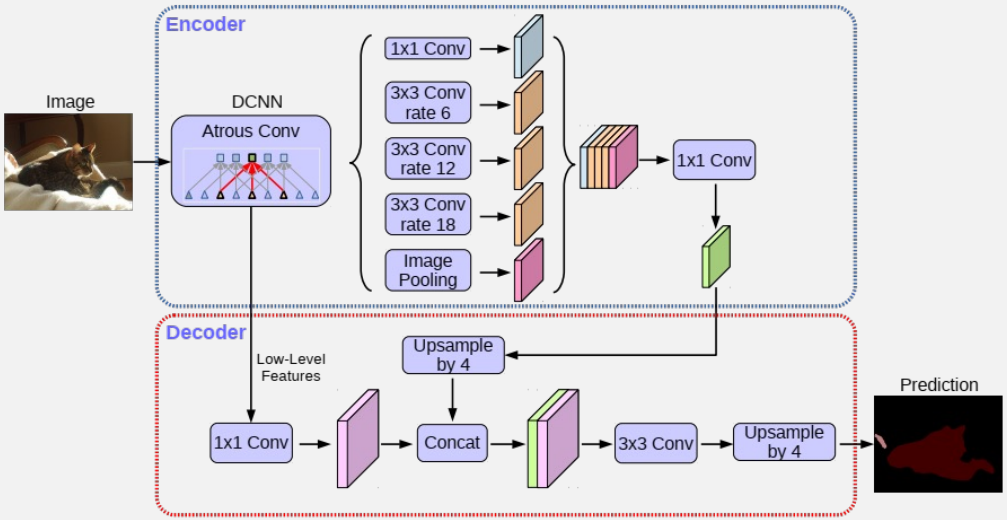


ig. 4. We modify the Xception as follows: (1) more layers (same as MSRA's modifica-ion except the changes in Entry flow), (2) all the max pooling operations are replaced y depthwise separable convolutions with striding, and (3) extra batch normalization nd ReLU are added after each 3×3 depthwise convolution, similar to MobileNet.

Chen, et al. (2018), Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation

DEEPLAB

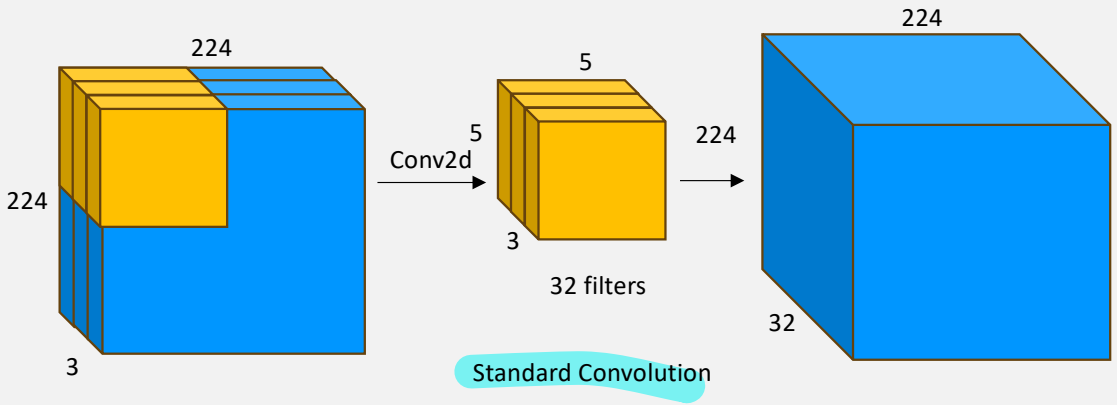
- DeepLabv3+
  - Adding a simple but effective decoder module to further refine the segmentation results especially along object boundaries.
  - Modify the Aligned Xception instead of ResNet-101 as its main feature extractor (encoder) to be deeper. All max pooling operations are replaced by depth-wise separable convolution.



Chen, et al. (2018), Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation

DEPTH-WISE SEPARABLE CONVOLUTION

- Depth-wise separable Convolution (used in MobileNet & Xception Architecture)
  - The filters are applied to each channel, so the output of the depth-wise convolution has the same channels as the input
  - Efficiency and Parameter Reduction
  - Combining Spatial and Cross-Channel Information



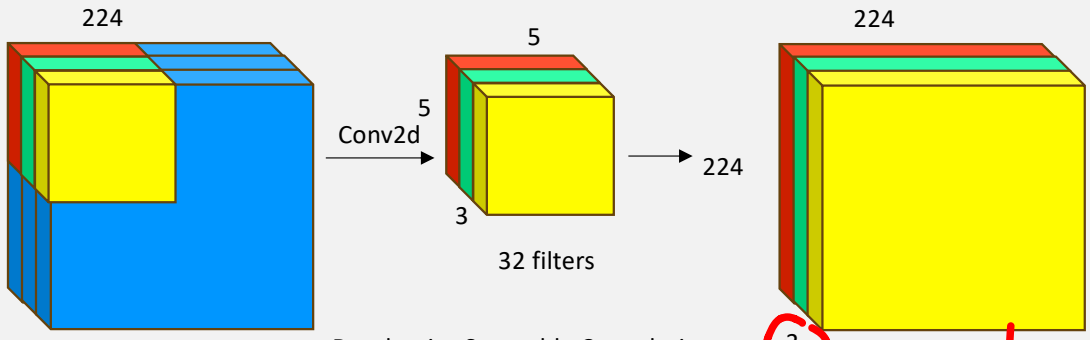
RGB

Parameters : 3 x 5 x 5 x 32 = 2,400
(Ignore bias)

DEPTH-WISE SEPARABLE CONVOLUTION

- Depth-wise separable Convolution (used in MobileNet & Xception Architecture)
  - The filters are applied to each channel, so the output of the depth-wise convolution has the same channels as the input
  - Efficiency and Parameter Reduction
  - Combining Spatial and Cross-Channel Information

1 kernel / 1 depth



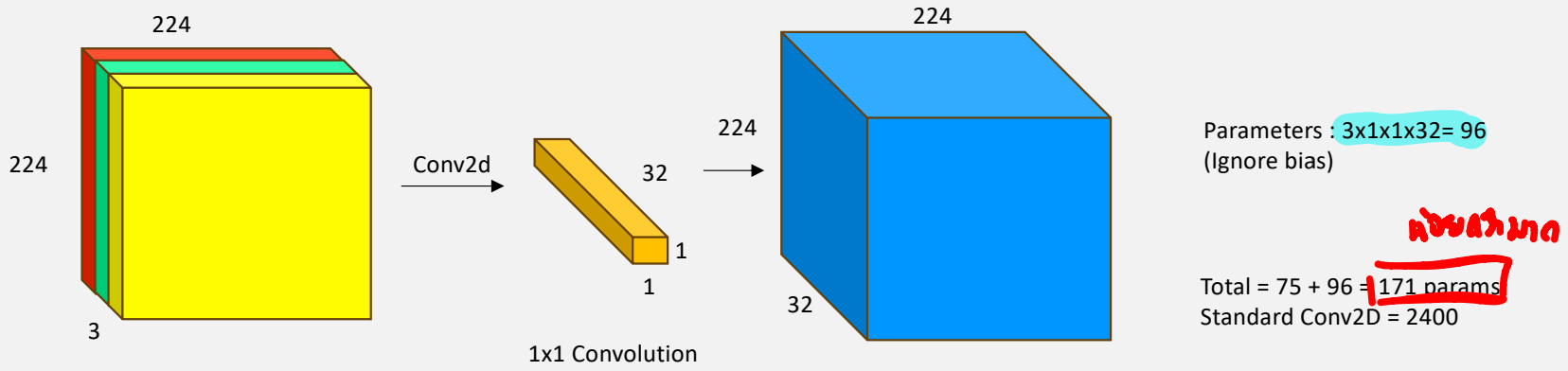
Parameters : 3 x 5 x 5 = 75 x1
(Ignore bias)

depth ยังไม่ได้

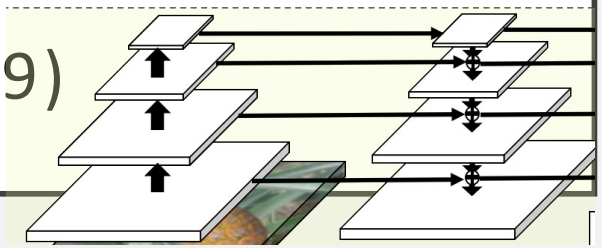
↳ แปลว่า depth ต้อง 1x1 conv ด้วย

DEPTH-WISE SEPARABLE CONVOLUTION

- Depth-wise separable Convolution (used in MobileNet & Xception Architecture)
  - Typically followed by a 1x1 convolution (also known as a pointwise convolution) - Inter-channel mixing is achieved by convolving the output of depth-wise convolution with a 1x1 kernel of required number of output channels



EFFICENTNET-NAS-FPN (2019)



- NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection
  - **Efficient Backbone (EfficientNet):** Provides a parameter-efficient and computationally effective backbone. Enables high accuracy with fewer parameters, making it suitable for real-time applications and resource-constrained devices.
  - **Optimized Architecture (Neural Architecture Search, NAS)**: Ensures that the backbone and overall architecture are optimized for the task, leveraging EfficientNet's architecture and scaling principles. Customizes the model structure automatically, achieving better performance and efficiency than manual designs.
  - **Multi-Scale Representation (FPN)**: Enhances detection and segmentation capabilities by fusing features across multiple scales. Strengthens the model's ability to detect objects at different sizes and maintain high-quality boundaries for segmentation tasks.

Feature Pyramid Network

Ghaisi, et al.(2019), GoogleBrain, NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection

HRNET

- Deep High-Resolution Representation Learning for Visual Recognition
  - High-Resolution Representations throughout the network, ideal for capturing fine spatial details.
  - Multi-Scale Fusion across multiple resolutions, allowing accurate boundary detection and detailed segmentation. Parallel Branches with Stage-Wise Design that efficiently combine information at multiple resolutions.
  - High Accuracy and Parameter Efficiency, making it suitable for real-time and resource-limited applications.

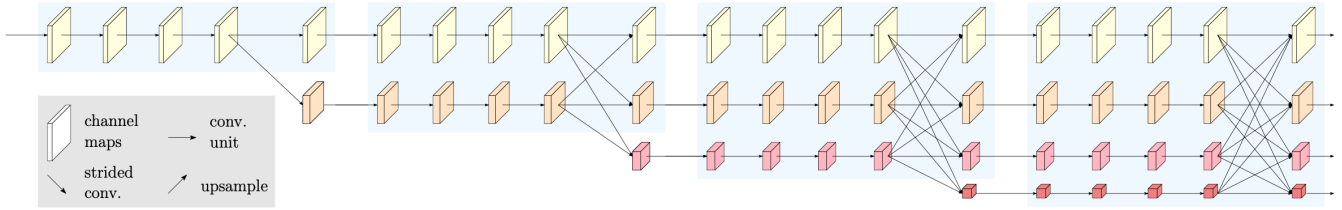


Fig. 2. An example of a high-resolution network. Only the main body is illustrated, and the stem (two stride-2 3×3 convolutions) is not included. There are four stages. The 1st stage consists of high-resolution convolutions. The 2nd (3rd, 4th) stage repeats two-resolution (three-resolution, four-resolution) blocks. The detail is given in Section 3.

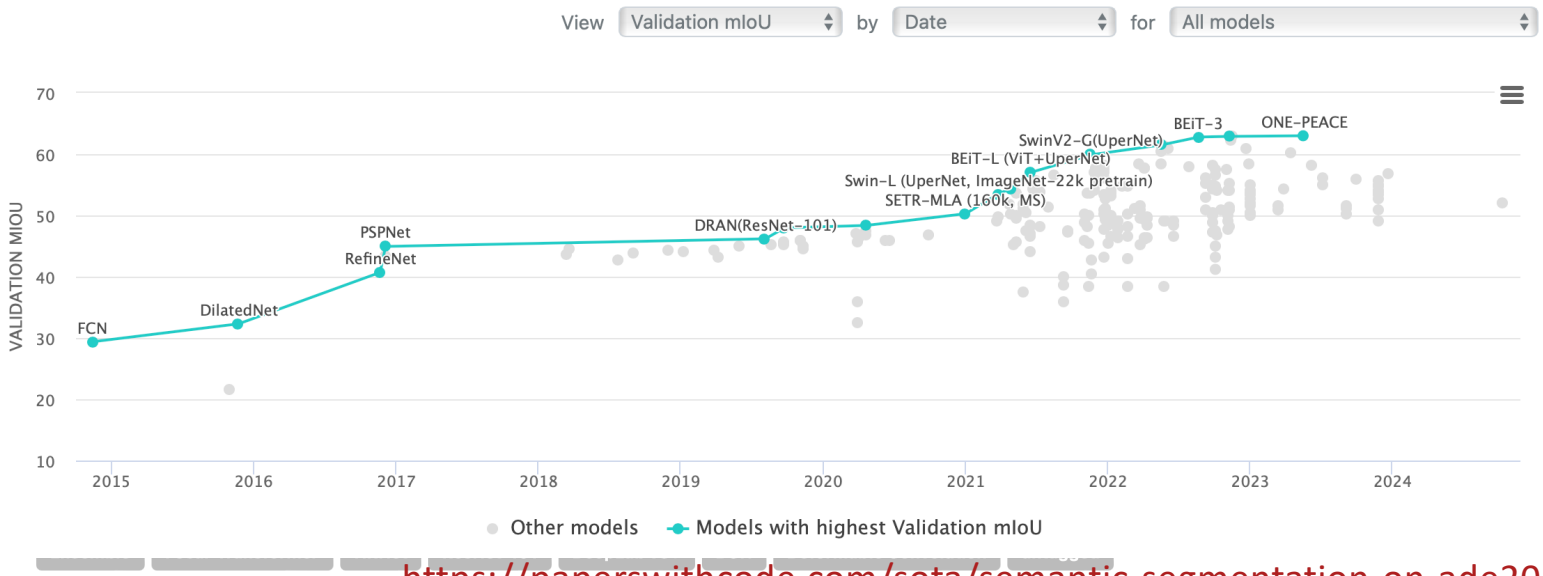
Wang, et al. (2020), PAMI, Deep High-Resolution Representation Learning for Visual Recognition

MODERN METHODS (DEEP LEARNING)

Semantic Segmentation

Semantic Segmentation on ADE20K

Leaderboard Dataset



Rank	Model	Validation mIoU	Test Score
1	**ONE-PEACE**	63.0	
2	**InternImage-H**	62.9	
3	**M3I Pre-training** (InternImage-H)	62.9	
4	**BEiT-3**	62.8	
5	**EVA**	62.3	

https://paperswithcode.com/sota/semantic-segmentation-on-ade20k

TOP MODELS FOR ADE20K (NOV 2023,2024)

Rank	Model	Validation↑ mIoU	Test Score
1	**ONE-PEACE**	63.0	
2	**InternImage-H**	62.9	
3	**M3I Pre-training** (InternImage-H)	62.9	
4	**BEiT-3**	62.8	
5	**EVA**	62.3	

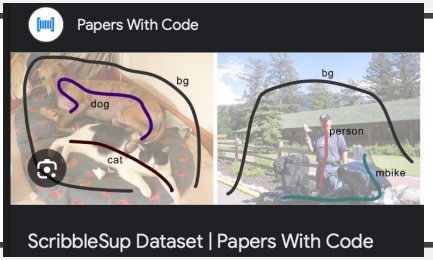
- **ONE-PEACE** – exploring One General Representation Model Toward Unlimited Modalities
  - a highly extensible model with 4B parameters
  - integrate representations across vision, audio, and language modalities
  - results on a wide range of uni-modal and multi-modal tasks,
    - image classification (ImageNet),
    - semantic segmentation (ADE20K),
    - audio-text retrieval (AudioCaps, Clotho),
    - audio classification (ESC-50, FSD50K, VGGSound),
    - audio question answering (AVQA),
    - image-text retrieval (MSCOCO, Flickr30K), and
    - visual grounding (RefCOCO/+/g)

TOP MODELS FOR ADE20K (NOV 2023,2024)

Rank	Model	Validation↑ mIoU	Test Score
1	**ONE-PEACE**	63.0	
2	**InternImage-H**	62.9	
3	**M3I Pre-training** (InternImage-H)	62.9	
4	**BEiT-3**	62.8	
5	**EVA**	62.3	

- InternImage-H
  - a new large-scale CNN-based foundation model
  - can obtain the gain from increasing parameters and training data like ViTs.
  - takes deformable convolution as the core operator, so has the large effective receptive field required for detection and segmentation, and also has the adaptive spatial aggregation conditioned by input and task information.
- M3I Pre-training
  - Towards All-in-one Pre-training via Maximizing Multi-modal Mutual Information
  - an all-in-one single-stage pre-training approach, named Maximizing Multi-modal Mutual Information Pre-training (M3I Pre-training)
- BEiT-3 - a general-purpose multimodal foundation model, Multiway Transformers for general-purpose modeling
- EVA - Limits of Masked Visual Representation Learning at Scale, a vanilla ViT pre-trained to reconstruct the masked out image-text aligned vision features conditioned on visible image patches

TRENDS



- Applications : Computer vision and image recognition, cartography and navigation, medicine and medical imaging, satellite imaging, AR/VR
- Weakly supervised learning: image-level, bounding box, point, scribble dataset
- Domain adaptation: Input-level
- Multi-modal data fusion: RGB + T (thermal), RGB + D (Depth – LiDar)
- Real-time segmentation

Mo, et al. (2022) Review the state-of-the-art technologies of semantic segmentation based on deep learning

EXERCISE #1 EXPERIMENT WITH DIFF. BACKBONE

- From U-Net model, change the backbone of the model, identify
  - Why you select a particular backbone?
  - Does it improve the performance of the model?