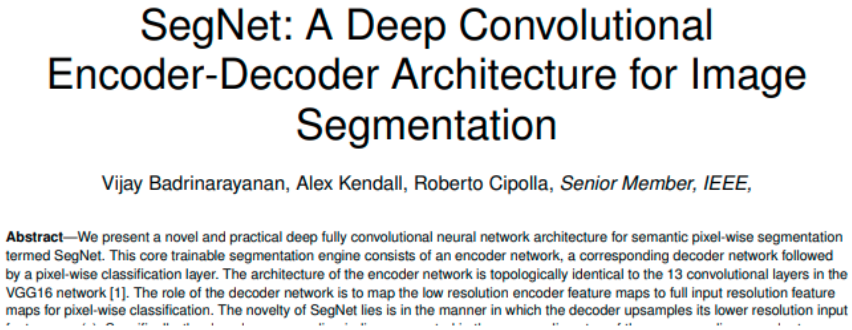
Introduction to Semantic Segmentation

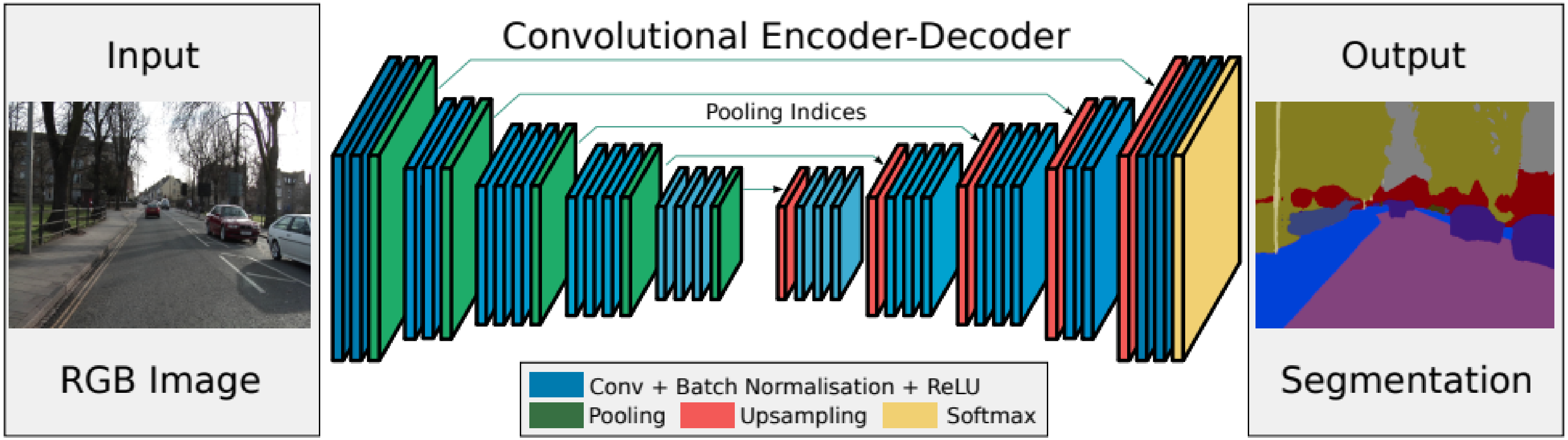
https://arxiv.org/pdf/1511.00561.pdf



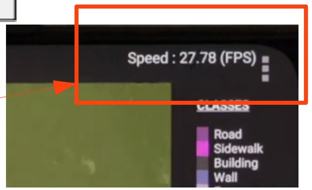
SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, *Senior Member, IEEE*,

Abstract—We present a novel and practical deep fully convolutional neural network architecture for semantic pixel-wise segmentation termed SegNet. This core trainable segmentation engine consists of an encoder network, a corresponding decoder network followed by a pixel-wise classification layer. The architecture of the encoder network is topologically identical to the 13 convolutional layers in the VGG16 network [1]. The role of the decoder network is to map the low resolution encoder feature maps to full input resolution feature maps for pixel-wise classification. The novelty of SegNet lies is in the manner in which the decoder upsamples its lower resolution input



Qualcomm sample video: https://www.youtube.com/watch?v=hGrJ3zuuvRQ



DeepLab for Dense Pixel Labeling Semantic Image Segmentation

https://github.com/google-research/deeplab2 and older one https://github.com/tensorflow/models/tree/master/research/deeplab

DeepLab2: (1) a TensorFlow library for deep labeling for a unified and state-of-the-art TensorFlow codebase for dense pixel labeling, including, but not limited to semantic segmentation, instance segmentation, panoptic segmentation, depth estimation, or even video panoptic segmentation.
(2) Deep labeling assigns a predicted value for each pixel.

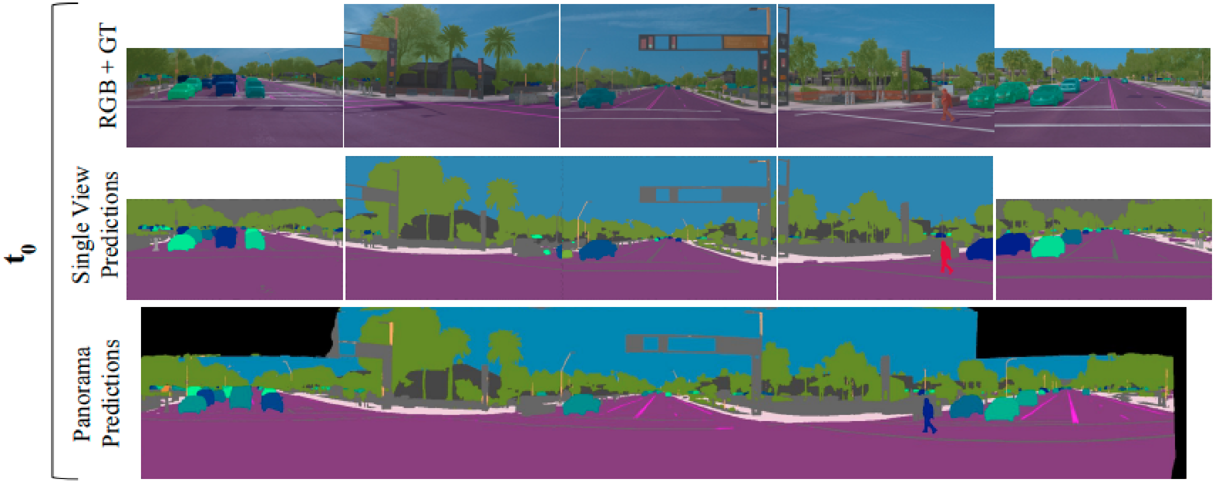
Waymo Open Dataset: Panoramic Video Panoptic Segmentation

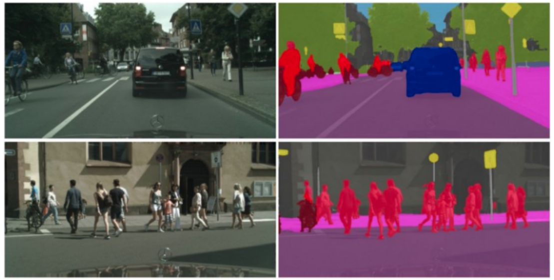

Waymo Open Dataset:
Panoramic Video Panoptic Segmentation

Jieru Mei[1*] Alex Zihao Zhu[2] Xinchen Yan[2] Hang Yan[2]
Siyuan Qiao[3] Yukun Zhu[3] Liang-Chieh Chen[3]
Henrik Kretzschmar[2] Dragomir Anguelov[2]

[1]Johns Hopkins University [2]Waymo LLC [3]Google Research

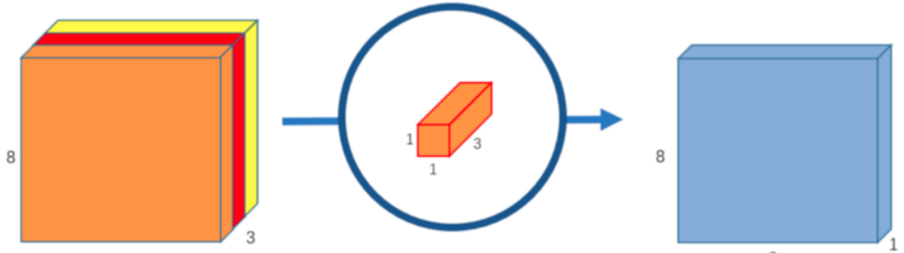
[cs.CV] 15 Jun 2022



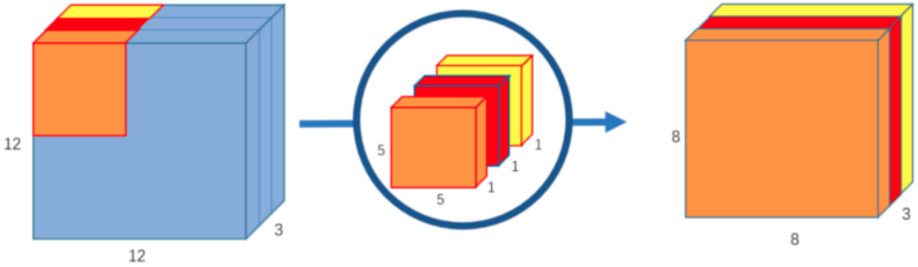


Semantic Decoder Python

1. Pointwise convolution, e.g., 1x1xk convolution;
2. Depthwise convolution: Atrous convolution;

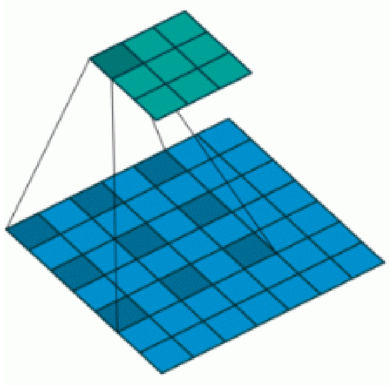


Pointwise convolution

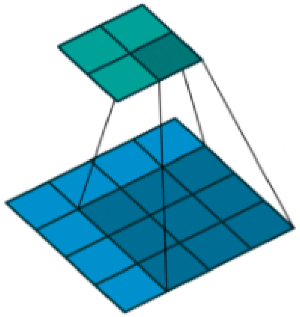


Depthwise convolution

https://www.analyticsvidhya.com/blog/2019/02/tutorial-semantic-segmentation-google-deeplab/



Dilated Convolution

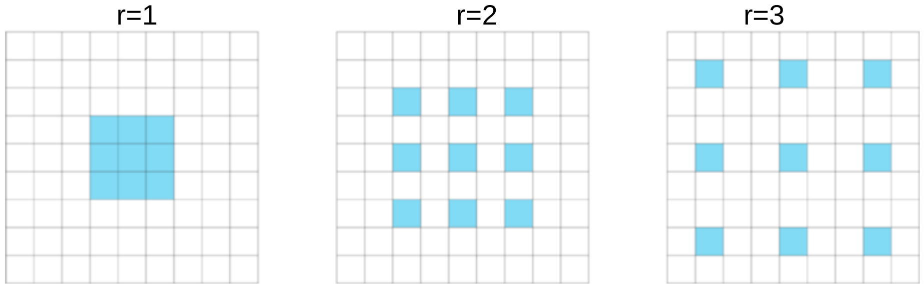


Standard Convolution

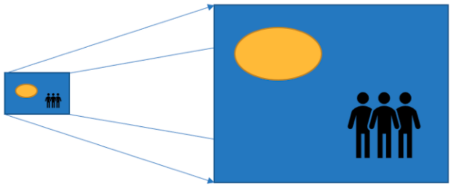
Atrous convolutions

$$y[i] = \sum_{k} x[i + r \cdot k]w[k]$$

DeepLab uses atrous convolution with rates 6, 12 and 18.

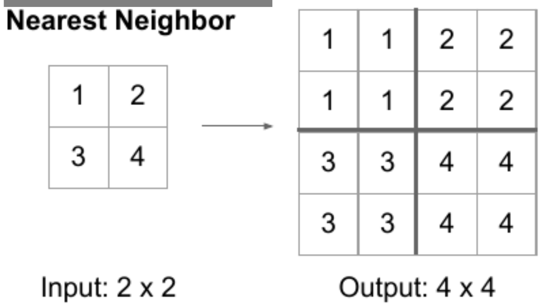


Up-sampling Techniques

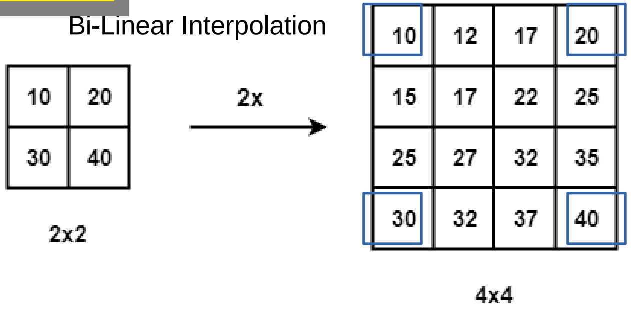


https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0

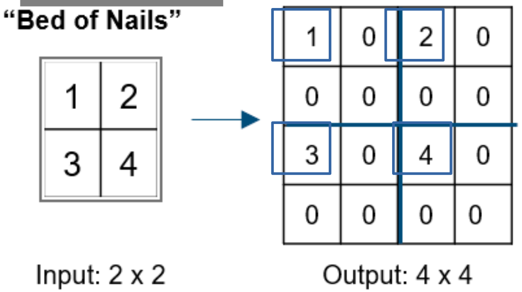
1. Nearest Neighbor



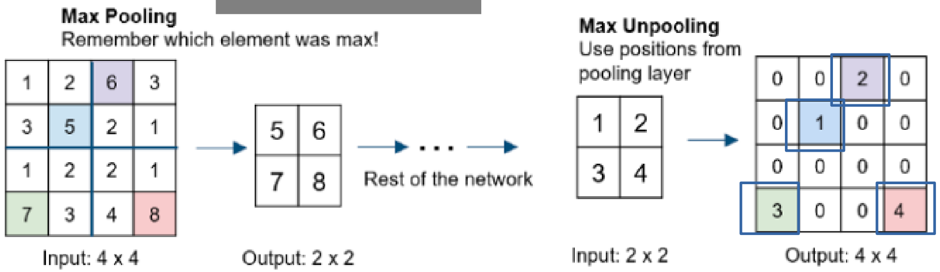
2. Bilinear



3 BoN Anchor



4. Max Unpooling



https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Upsampling + convolution is better than transpose convolution: https://distill.pub/2016/deconv-checkerboard/
https://distill.pub/2016/deconv-checkerboard/

Transposed Convolution Up-sampling

Credit of the example illustration:
https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

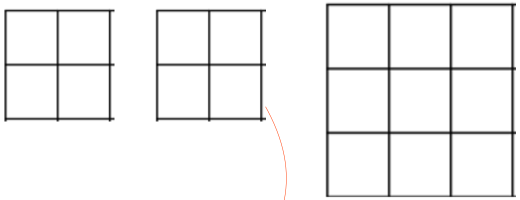
https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0

1. Consider a 2x2 encoded feature map which needs to be upsampled to a 3x3 feature map.

Input image Feature map: 2x2

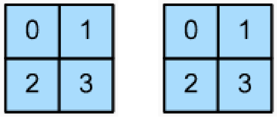
Kernel 2x2

Upsampled output image: 3x3



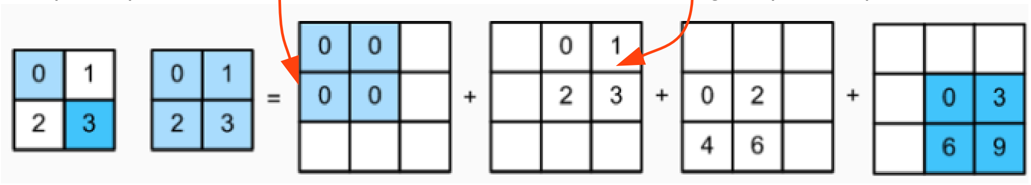
kernel of size 2x2 with unit stride and zero padding.

Example:

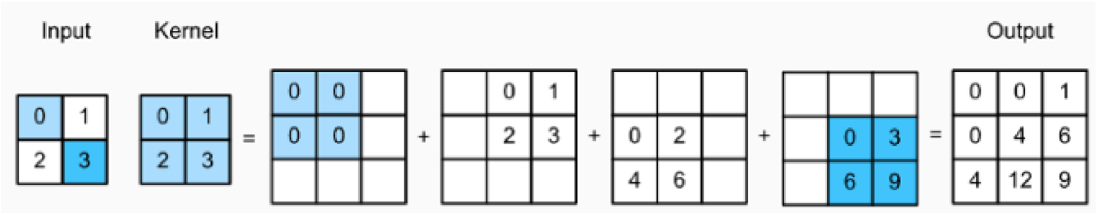


Step 1. Feature map and the kernel

Step 2. Transposed convolution for each pixel in the feature map: take 0 from the image, then multiply each coefficient of the kernel and place the result back to its corresponding location in the bigger output map, so 0x0, 0x1, 0x2, 0x3; then next take 1 from the image, repeat the process



Step 3. Add output at each pixel location together to form upsampled image

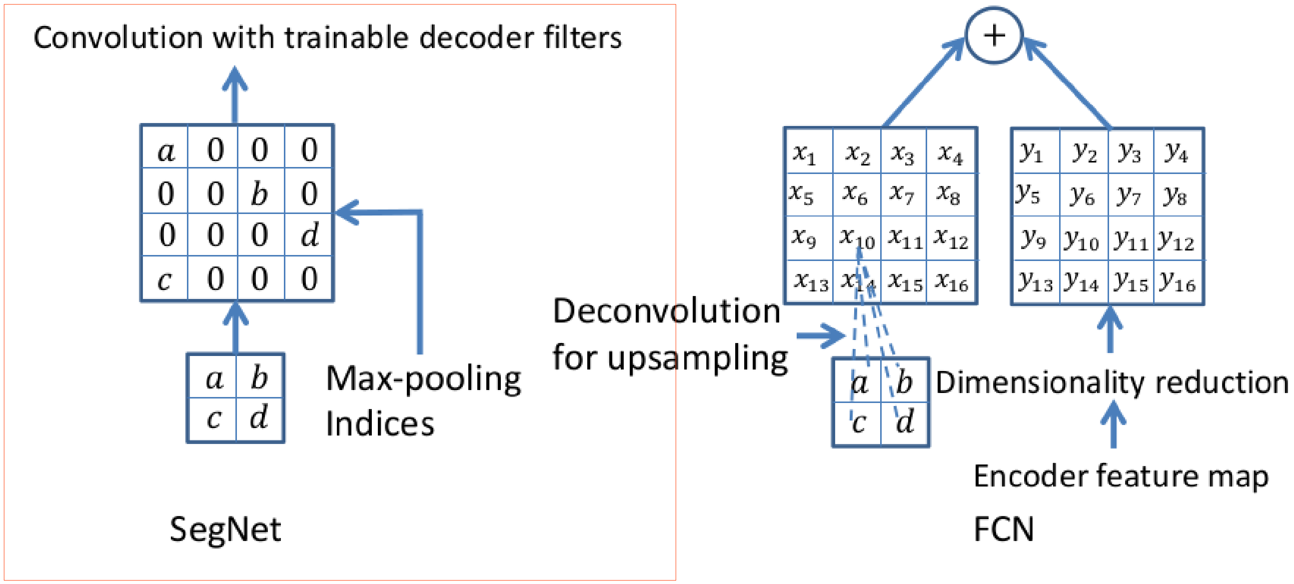


Animated tutorial on transpose convolution
https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6

```
>>> # With square kernels and equal stride
>>> m = nn.ConvTranspose2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.ConvTranspose2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> input = torch.randn(20, 16, 50, 100)
>>> output = m(input)
>>> # exact output size can be also specified as an argument
>>> input = torch.randn(1, 16, 12, 12)
>>> downsample = nn.Conv2d(16, 16, 3, stride=2, padding=1)
>>> upsample = nn.ConvTranspose2d(16, 16, 3, stride=2, padding=1)
>>> h = downsample(input)
>>> h.size()
torch.Size([1, 16, 6, 6])
>>> output = upsample(h, output_size=input.size())
>>> output.size()
torch.Size([1, 16, 12, 12])
```

Transposed Convolution Up-sampling

Upsampling Example (Left), source:
https://arxiv.org/pdf/1511.00561.pdf



pp. 6

Fig. 3. An illustration of SegNet and FCN [2] decoders. a, b, c, d correspond to values in a feature map. SegNet uses the max pooling indices to upsample (without learning) the feature map(s) and convolves with a trainable decoder filter bank. FCN upsamples by learning to deconvolve the input feature map and adds the corresponding encoder feature map to produce the decoder output. This feature map is the output of the max-pooling layer (includes sub-sampling) in the corresponding encoder. Note that there are no trainable decoder filters in FCN.

Transposed Convolution Up-sampling with Python

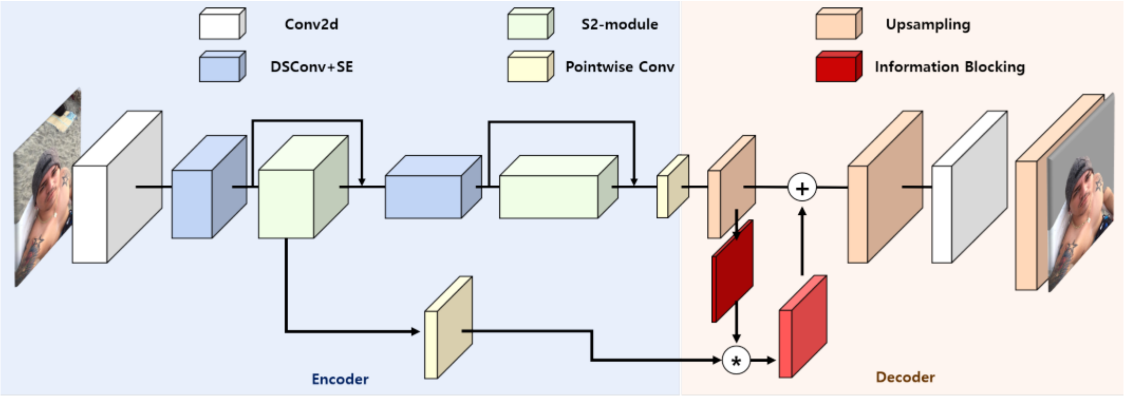
Python Reference Code (Untested)

```
def apply_conv(data, kernel, conv):
"""
Args:
data (NDArray): input data.
kernel (NDArray): convolution's kernel parameters.
conv (Block): convolutional layer.
Returns:
NDArray: output data (after applying convolution).
"""
# add dimensions for batch and channels if necessary
while data.ndim < len(conv.weight.shape):
data = data.expand_dims(0)
# add dimensions for channels and in_channels if necessary
while kernel.ndim < len(conv.weight.shape):
kernel = kernel.expand_dims(0)
# check if transpose convolution
if type(conv).__name__.endswith("Transpose"):
in_channel_idx = 0
else:
in_channel_idx = 1
# initialize and set weight
conv._in_channels = kernel.shape[in_channel_idx]
conv.initialize()
conv.weight.set_data(kernel)
return conv(data)
```

https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

SiNet Architecture for the Fast Portrait Segmentation With CPU

An encoder-decoder structure is the most commonly
used structure for segmentation.



DSConv: Efficient Convolution Operator https://arxiv.org/abs/1901.01928

We introduce DSConv, a flexible quantized convolu-
tion operator that replaces single-precision operations with
their far less expensive integer counterparts

Pointwise Convolution: convolution that uses a 1x1 kernel. This
kernel has a depth.

SINet: Extreme Lightweight Portrait Segmentation Networks with Spatial Squeeze Modules and Information Blocking Decoder

Hyojin Park
Seoul National University
wolfrun@snu.ac.kr

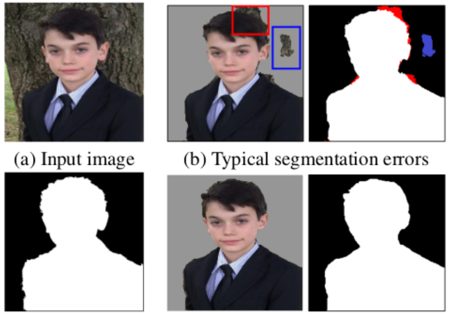
Lars Lowe Sjösund
Clova AI, NAVER Corp
lars.sjosund@navercorp.com

YoungJoon Yoo
Clova AI, NAVER Corp
youngjoon.yoo@navercorp.com

Nicolas Monet
NAVER LABS Europe
nicolas.monet@naverlabs.com

Jihwan Bang
Search Solutions, Inc
jihwan.bang@navercorp.com

Nojun Kwak
Seoul National University
nojunk@snu.ac.kr



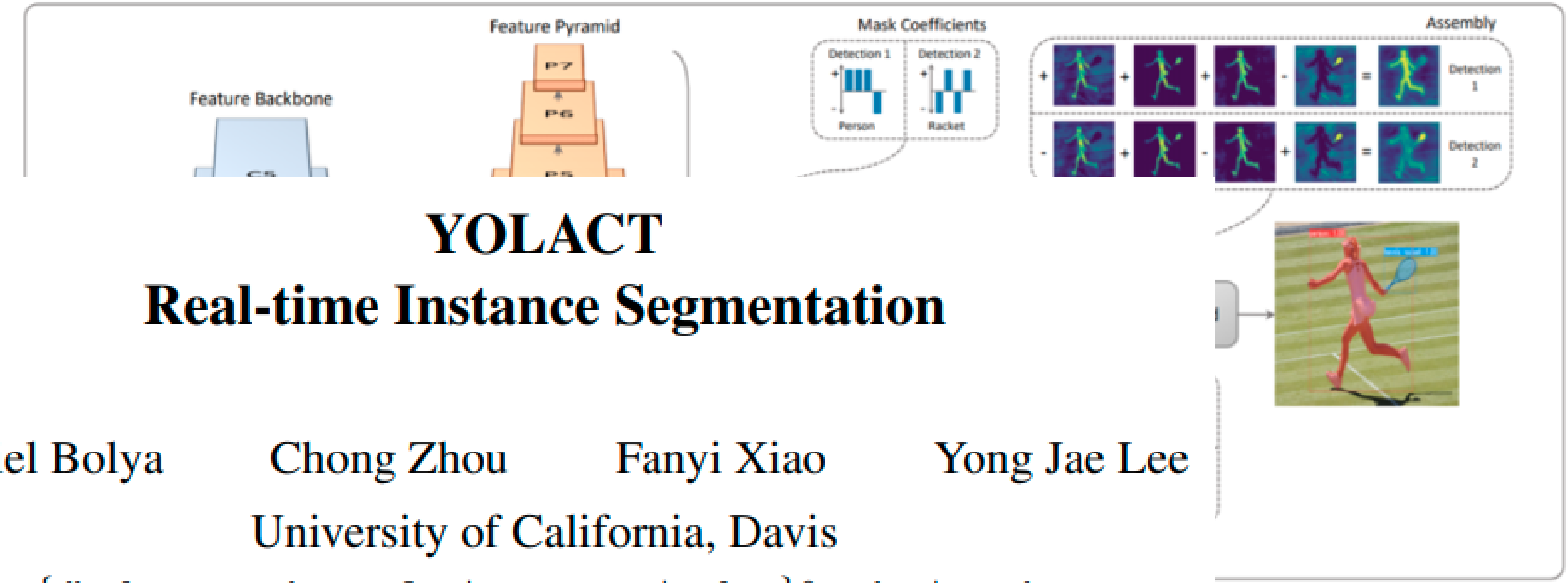
(a) Input image (b) Typical segmentation errors

(c) Ground truth (d) Example of Ours

segmentation models. Our method reduces the number of parameters from 2.1M to 86.9K (around 95.9% reduction), while maintaining the accuracy under an 1% margin from the state-of-the-art portrait segmentation method. We also show our model is successfully executed on a real mobile device with 100.6 FPS. In addition, we demonstrate that our

Use github Tensorflow 2.x Yolact Sample Code

1. The source from github:
https://github.com/anshkumar/yolact.git
Contributors:
leohsuofnthuÂ HSU CHIH-CHAO
anshkumarÂ vedanshu



YOLACT
Real-time Instance Segmentation

Daniel Bolya   Chong Zhou   Fanyi Xiao   Yong Jae Lee
University of California, Davis
{dbolya, cczhou, fyxiao, yongjaelee}@ucdavis.edu

odes indicate functions
ResNet-101 + FPN.

Introduction to Yolact as Implementation Example (for Its Speed)

1. The source from github:
https://github.com/anshkumar/yolact.git
Contributors:
leohsuofnthuÂ HSU CHIH-CHAO
anshkumarÂ vedanshu

2. readme document for the github code installation and testing.

https://github.com/hualili/opencv/blob/master/deep-learning-2022s/2022F-107-%23102n-1a-README-YOLACT-GPU-v1-YY-2022-9-12.pdf

3. Reference paper:
https://arxiv.org/pdf/1904.02689.pdf

https://arxiv.org/pdf/1904.02689.pdf

YOLACT
Real-time Instance Segmentation

Daniel Bolya Chong Zhou Fanyi Xiao Yong Jae Lee
University of California, Davis
{dbolya, cczhou, fyxiao, yongjaelee}@ucdavis.edu

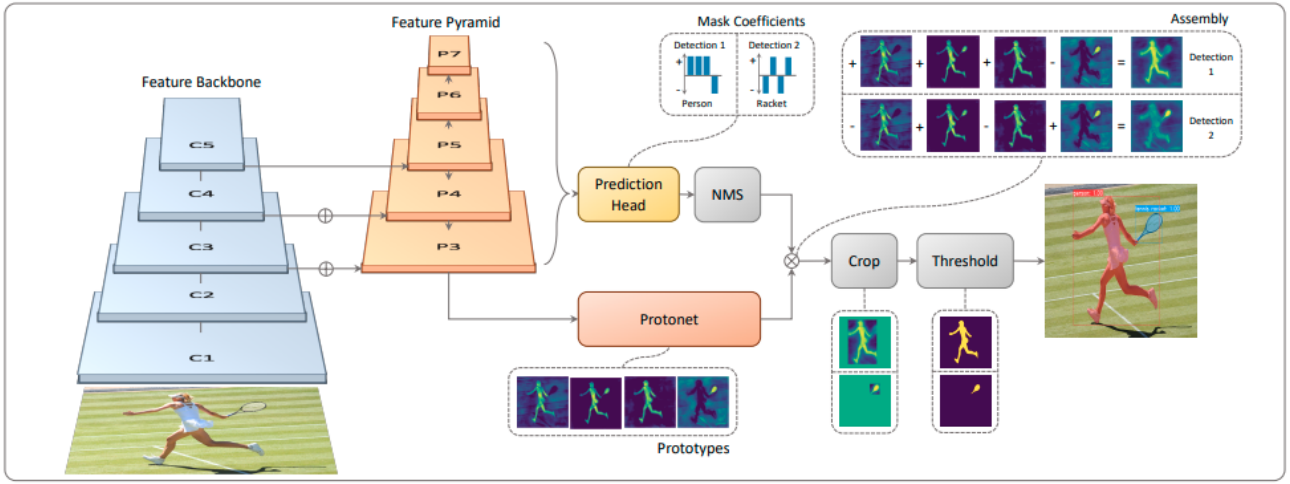
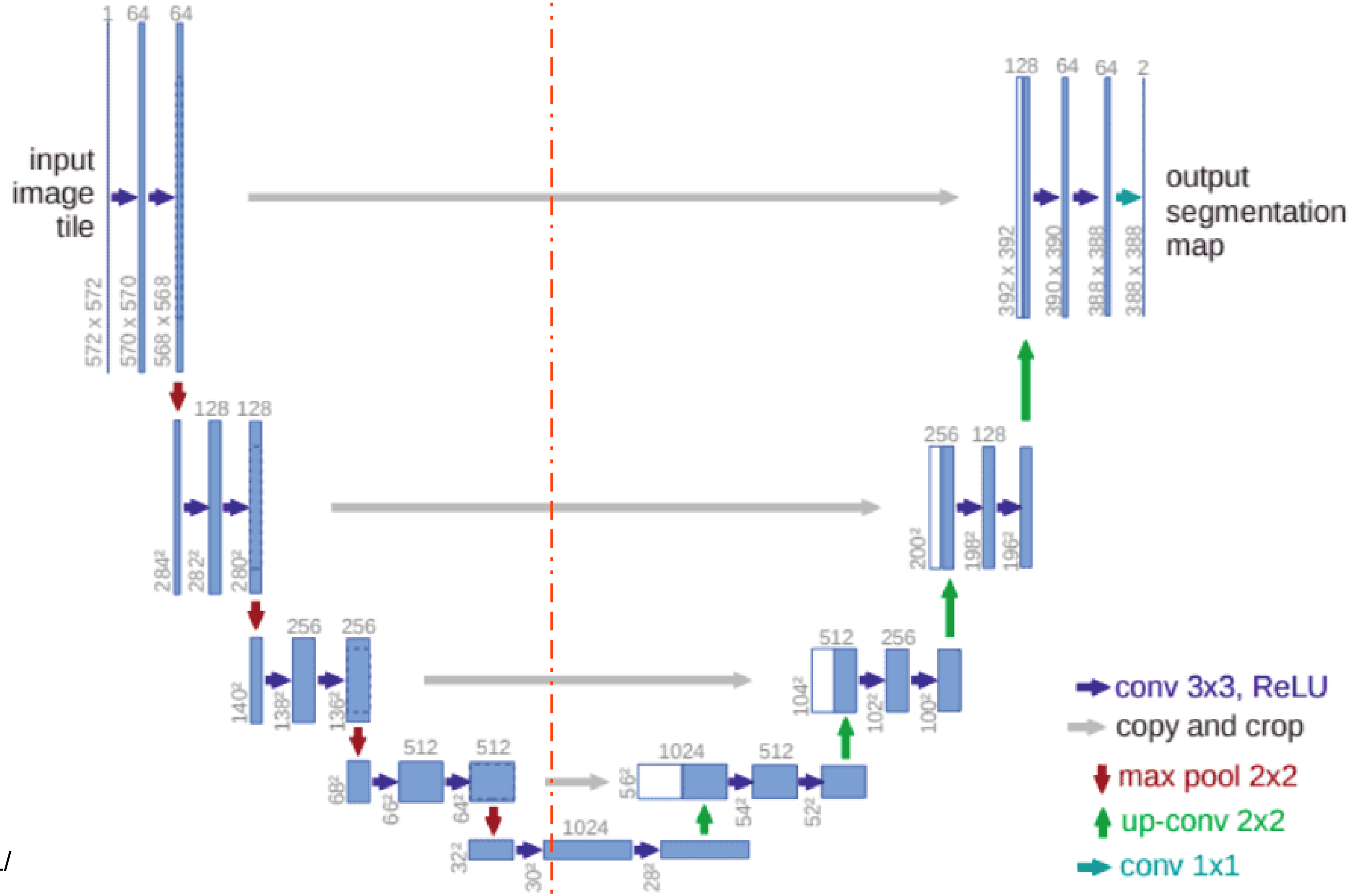


Figure 2: YOLACT Architecture Blue/yellow indicates low/high values in the prototypes, gray nodes indicate functions that are not trained, and k = 4 in this example. We base this architecture off of RetinaNet [27] using ResNet-101 + FPN.

Unet For Semantic Segmentation

https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

Unet For Semantic Segmentation

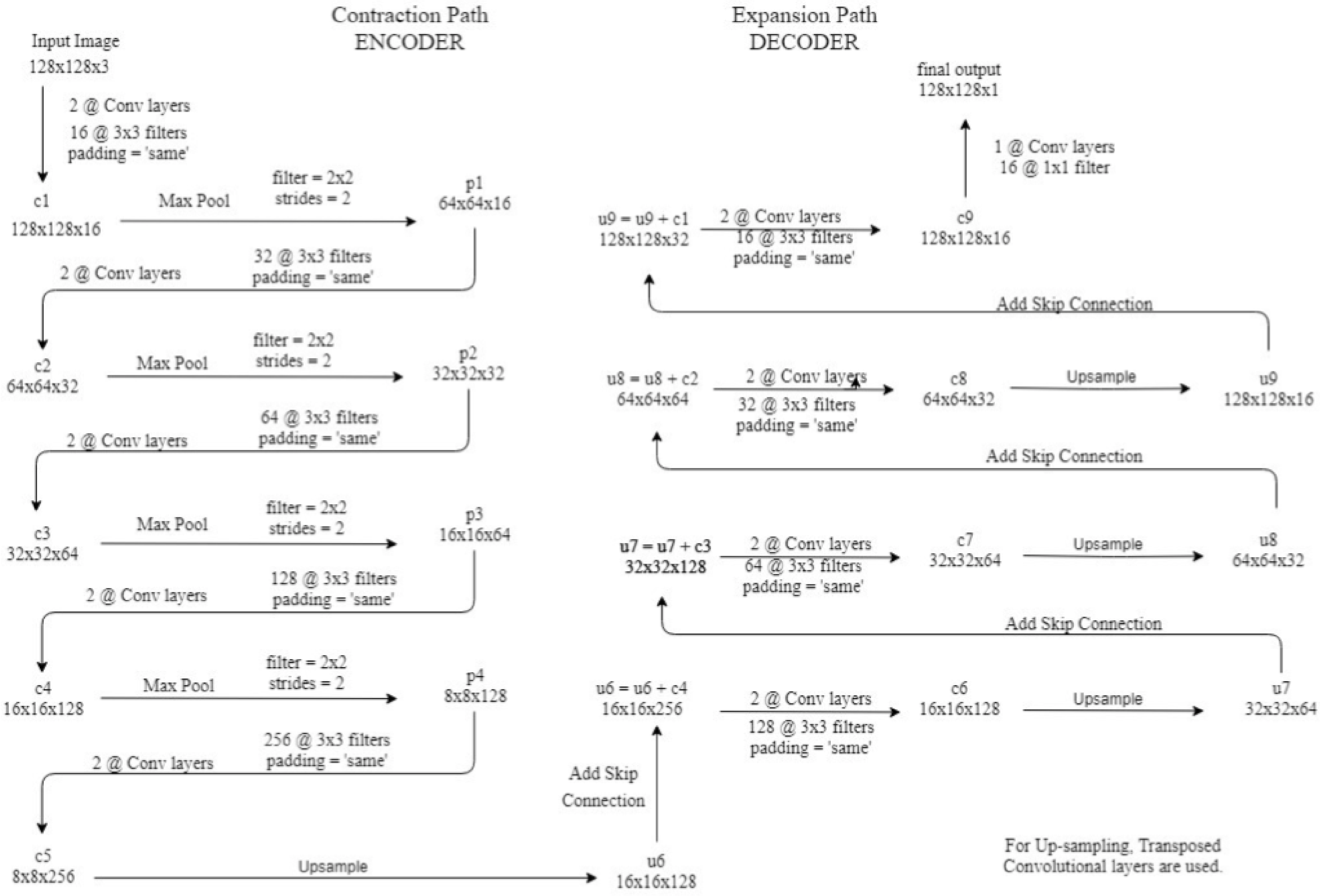


https://www.analyticsvidhya.com/blog/2019/02/tutorial-semantic-segmentation-google-deeplab/

Code sample and tutorial:
https://pyimagesearch.com/2022/02/21/u-net-image-segmentation-in-keras/

Unet For Semantic Segmentation

https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47



GIPHY and Other Tools for Annotation of Images for Semantic Segmentation

https://giphy.com/gifs/R0dnXaKJowlR2yL5CG

Labelbox

Supervisely

Fritz AI

RectLabel

Anolytics

Playment

Appen

Scale.ai

https://cnvrg.io/semantic-segmentation/

Six (6) Useful Image Segmentation Datasets And Python Usage

https://cnvrg.io/semantic-segmentation/

coco:

```
import tensorflow_datasets as tfds
(X_train, X_test), ds_info = tfds.load(
    'coco',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```

waymo:

```
import tensorflow_datasets as tfds
(X_train, X_test), ds_info = tfds.load(
    â€˜waymo_open_datasetâ€™,
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```

PASCAL:

```
import tensorflow_datasets as tfds
(X_train, X_test), ds_info = tfds.load(
    â€˜vocâ€™,
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```