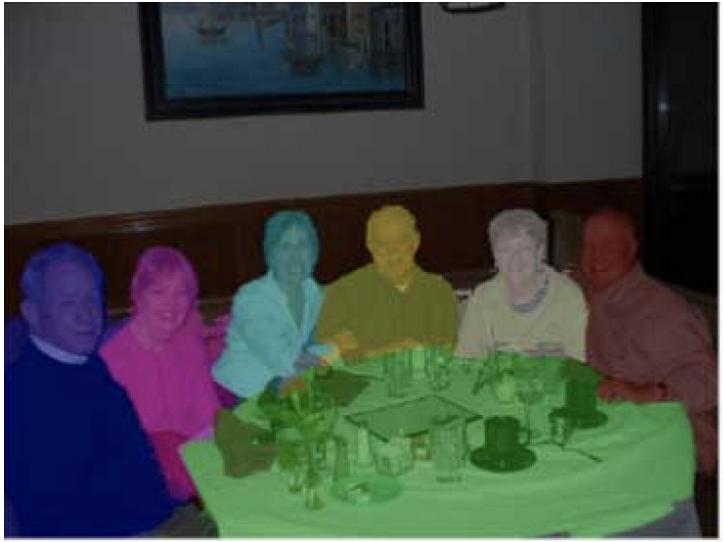
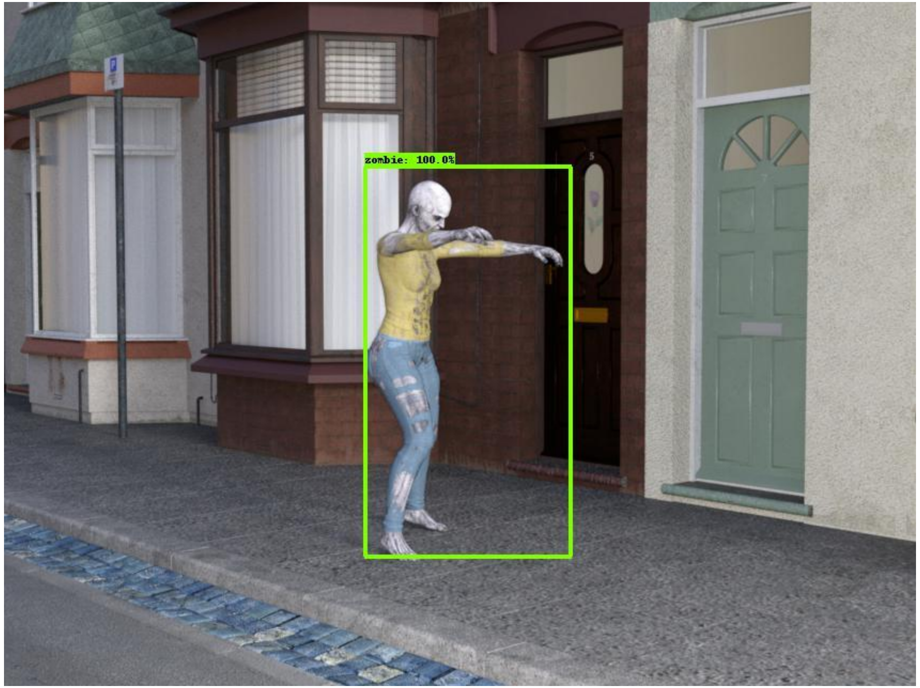
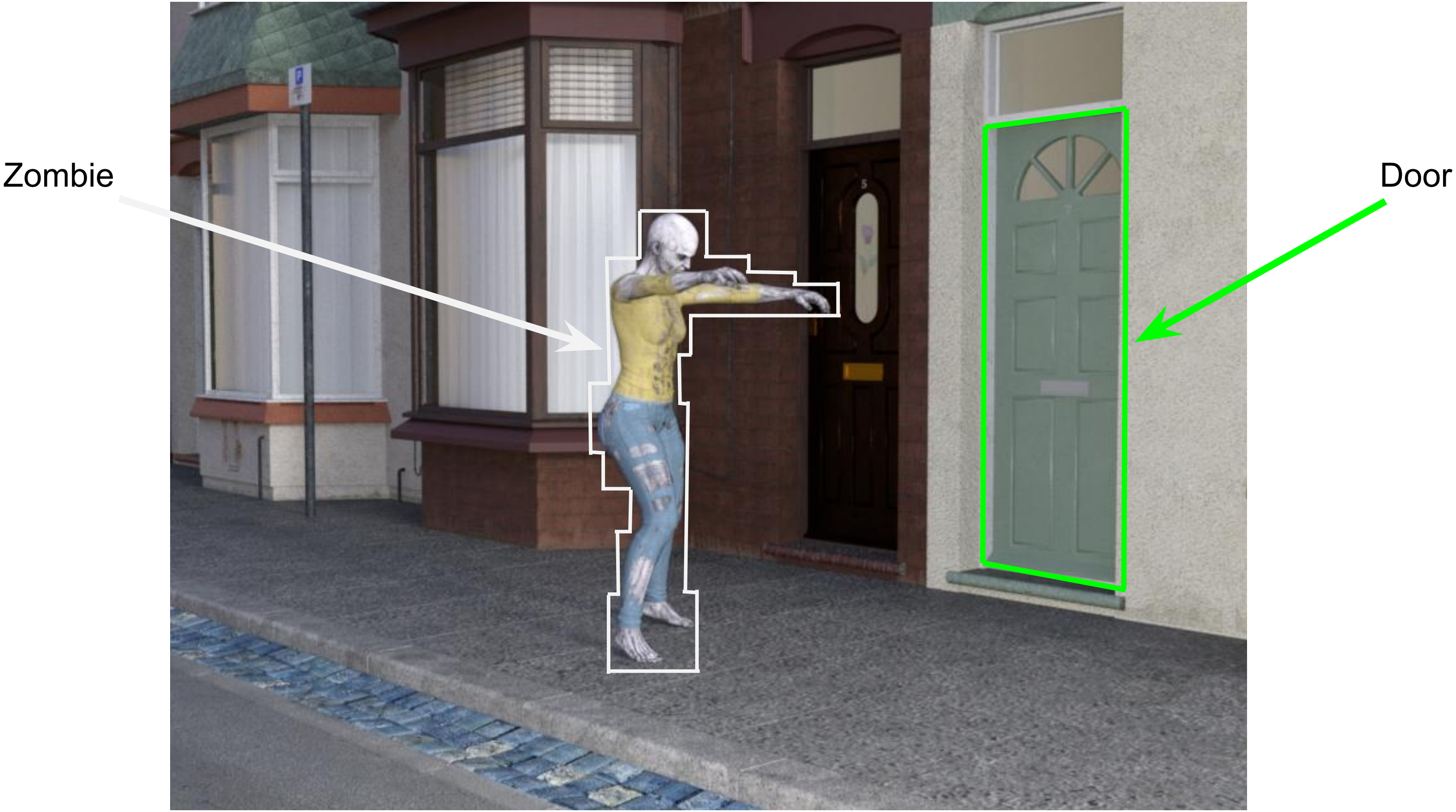
Copyright Notice

Image Segmentation



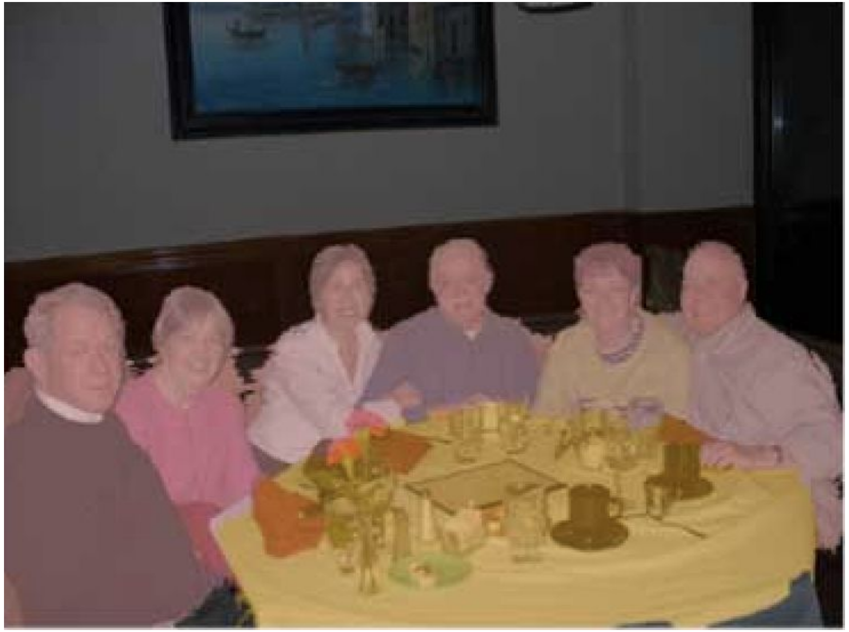
source


zombie: 100.0%

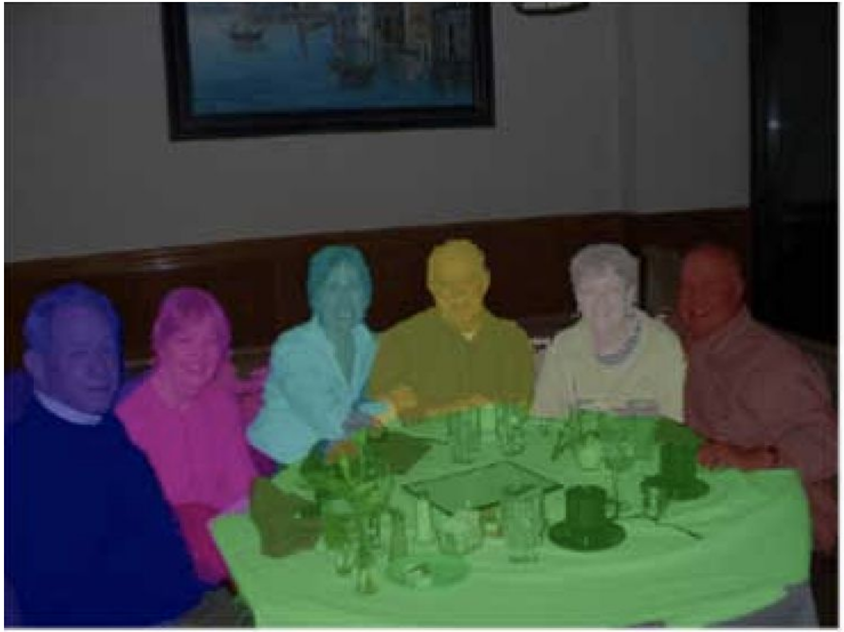

Zombie
Door

Zombie

Door



Semantic Segmentation



Instance Segmentation

Classes indices = [0 = Background 1 = People]

Image Segmentation Basic Architecture

224 x 224 x 3 (RGB image)

Encoder
(Feature Extractor, Down samples Image)

→ Downsampled Feature Map →

Decoder
(Up samples feature map, generates pixel-wise label map)

224 x 224 x n (Pixel map)
n = number of classes

Image Segmentation Basic Architecture

224 x 224 x 3 (RGB image)

Encoder
(Feature Extractor, Down samples Image)

Downsampled Feature Map

Decoder
(Up samples feature map, generates pixel-wise label map)

224 x 224 x n (Pixel map)
n = number of classes

Image Segmentation Basic Architecture

224 x 224 x 3 (RGB image)

Encoder
(Feature Extractor, Down samples Image)

Downsampled Feature Map

Decoder
(Up samples feature map, generates pixel-wise label map

224 x 224 x n (Pixel map)
n = number of classes

Image Segmentation Basic Architecture

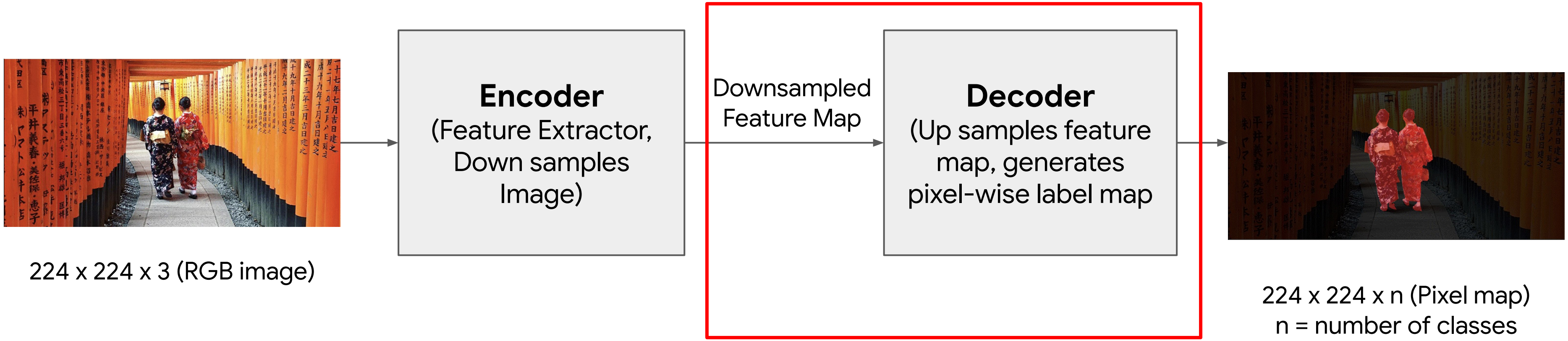


Image Segmentation Basic Architecture

224 x 224 x 3 (RGB image)

Encoder
(Feature Extractor, Down samples Image)

→ Downsampled Feature Map →

Decoder
(Up samples feature map, generates pixel-wise label map)

224 x 224 x n (Pixel map)
n = number of classes

Image Segmentation

- Encoder
  - CNN without fully connected layers
  - Aggregates low level features to high level features
- Decoder
  - Replaces fully connected layers in a CNN
  - Up samples image to original size to generate a pixel mask

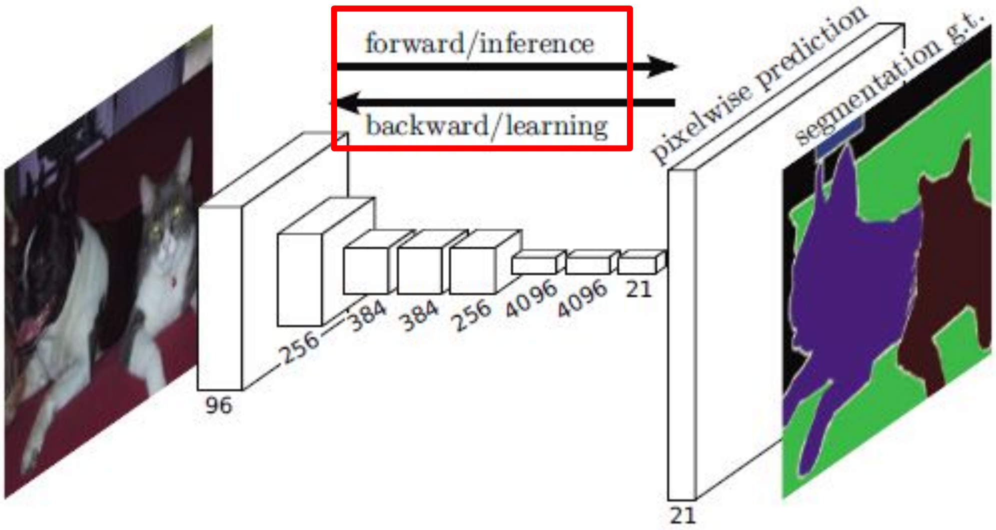
Popular Architectures

- Fully Convolutional Neural Networks.
    - SegNet
    - UNet
    - PSPNet
    - Mask-RCNN

Fully Convolutional Neural Networks

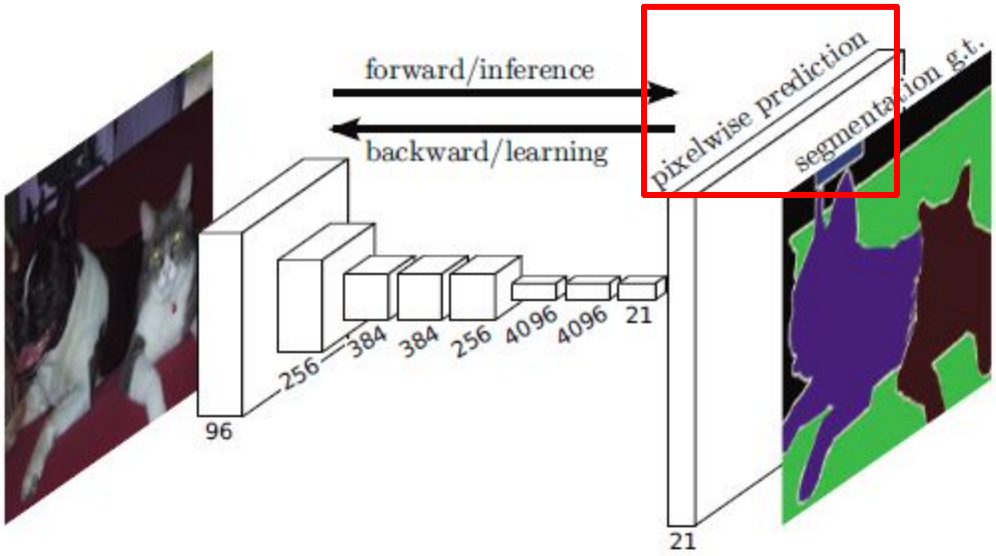
- "Fully Convolutional Networks for Semantic Segmentation"
  https://arxiv.org/abs/1411.4038
- Replace the fully connected layers with convolutional layers
- Earlier conv layers: Feature extraction and down sampling
- Later conv layers: up sample and pixel-wise labelmap.

Fully Convolutional Neural Networks



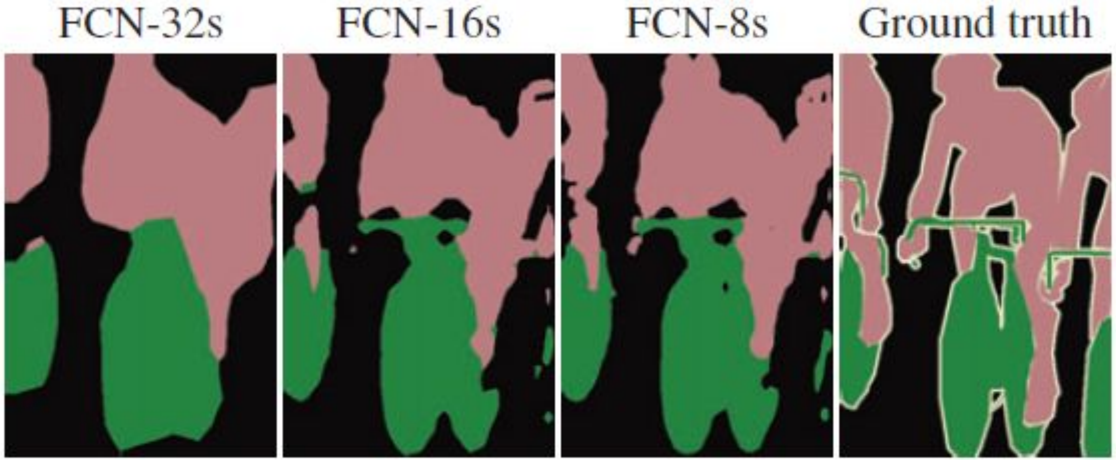
https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Fully Convolutional Neural Networks



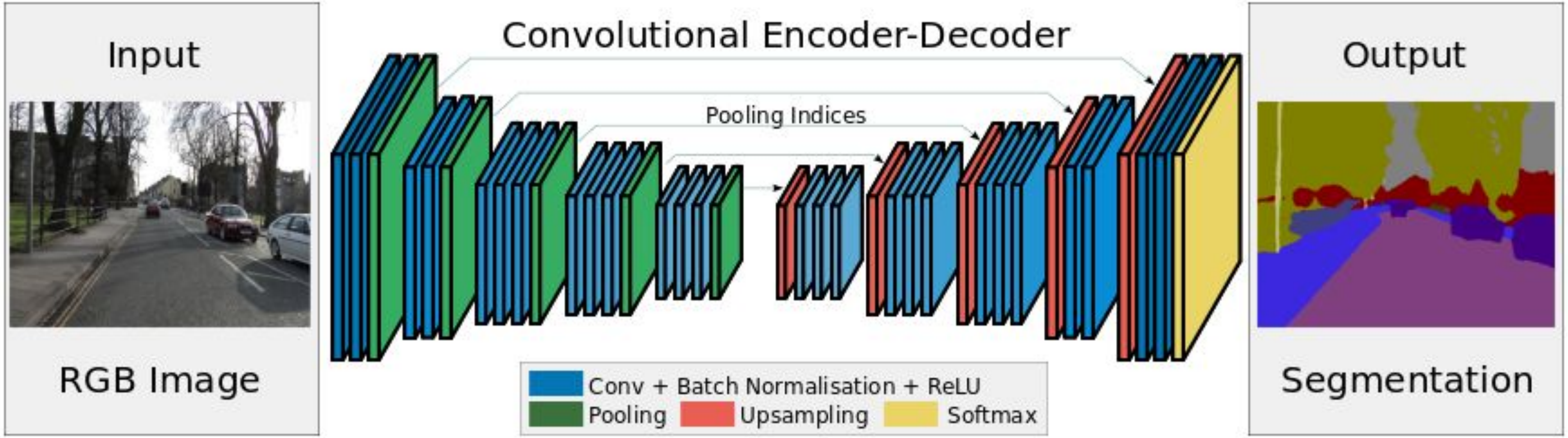
https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Comparison of Different FCNs

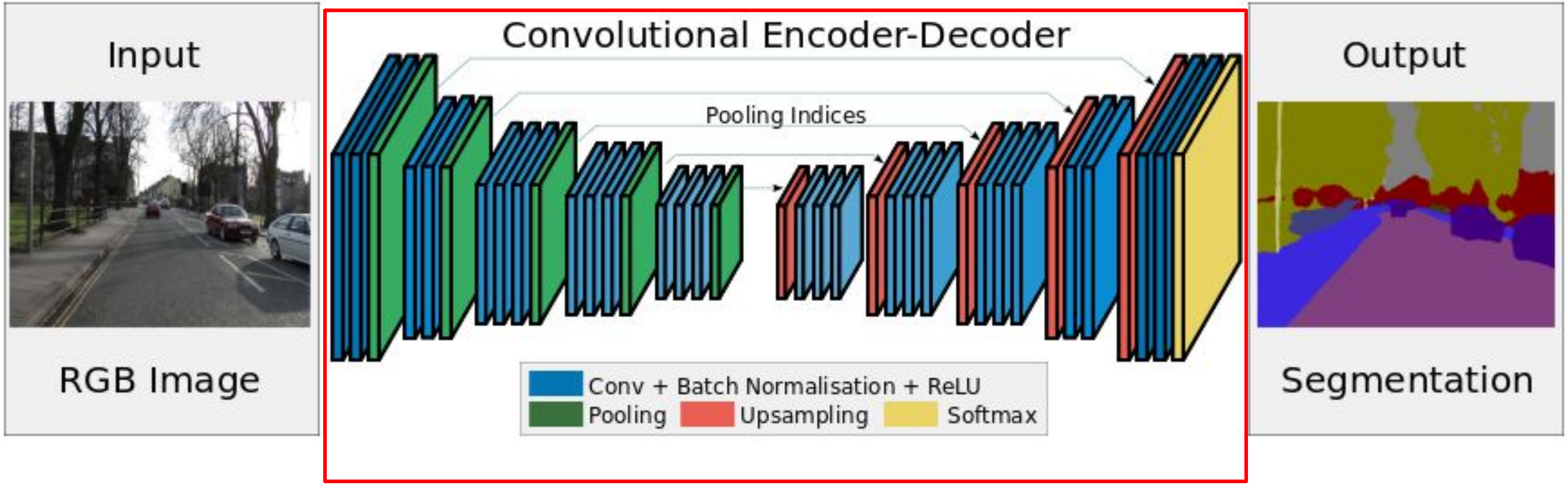


https://arxiv.org/pdf/1411.4038.pdf

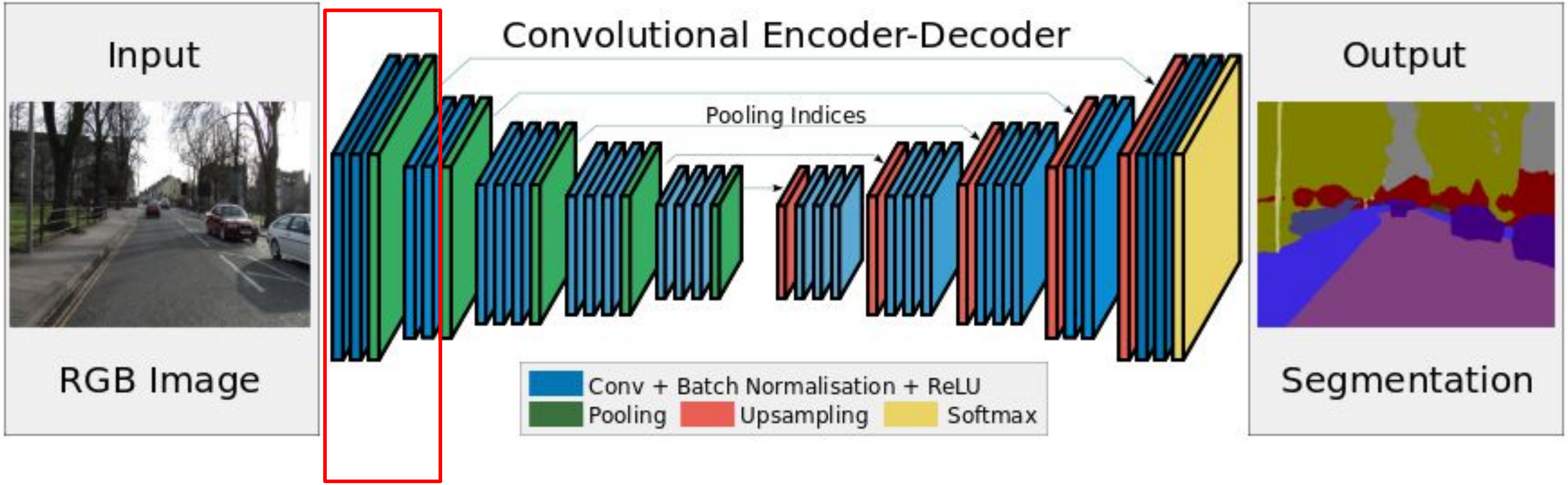
SegNet


Input
RGB Image
Convolutional Encoder-Decoder
Pooling Indices
Conv + Batch Normalisation + ReLU
Pooling
Upsampling
Softmax
Output
Segmentation

SegNet

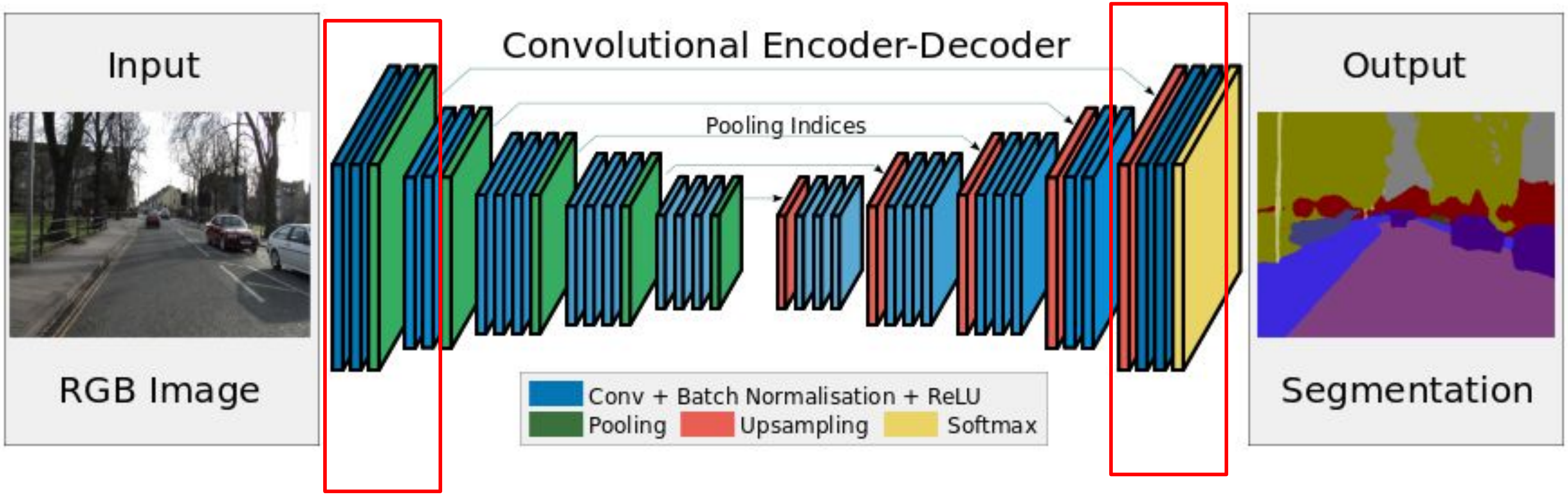

Input
RGB Image
Convolutional Encoder-Decoder
Pooling Indices
Conv + Batch Normalisation + ReLU
Pooling
Upsampling
Softmax
Output
Segmentation

SegNet



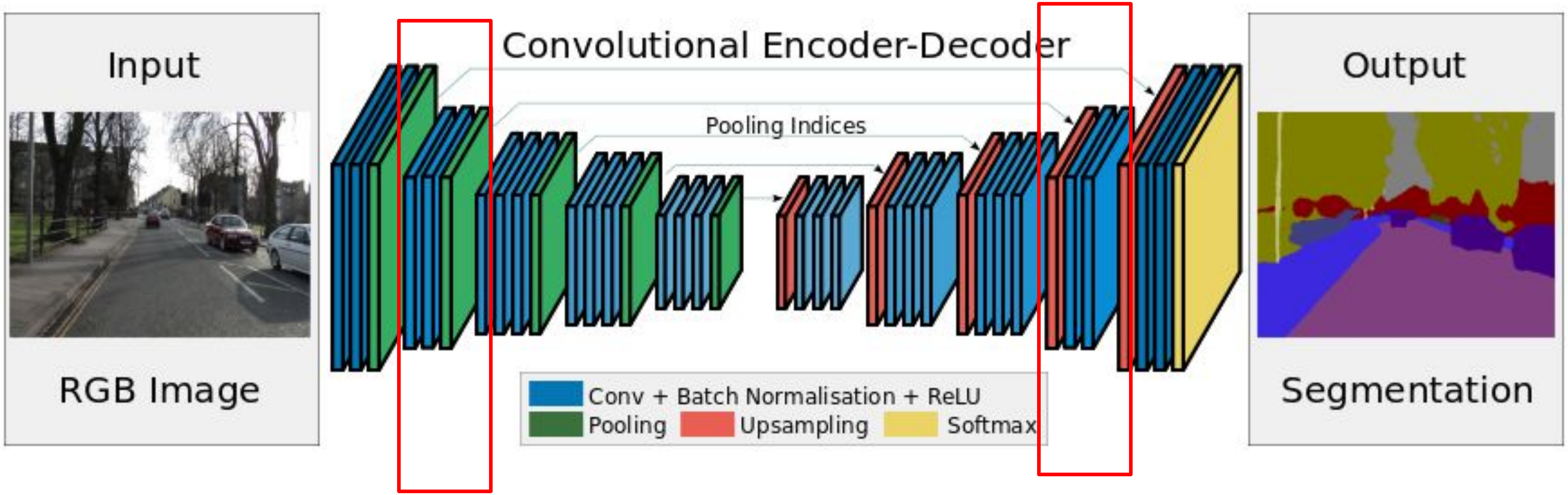
Input
RGB Image
Convolutional Encoder-Decoder
Pooling Indices
Output
Segmentation
Conv + Batch Normalisation + ReLU
Pooling
Upsampling
Softmax

SegNet

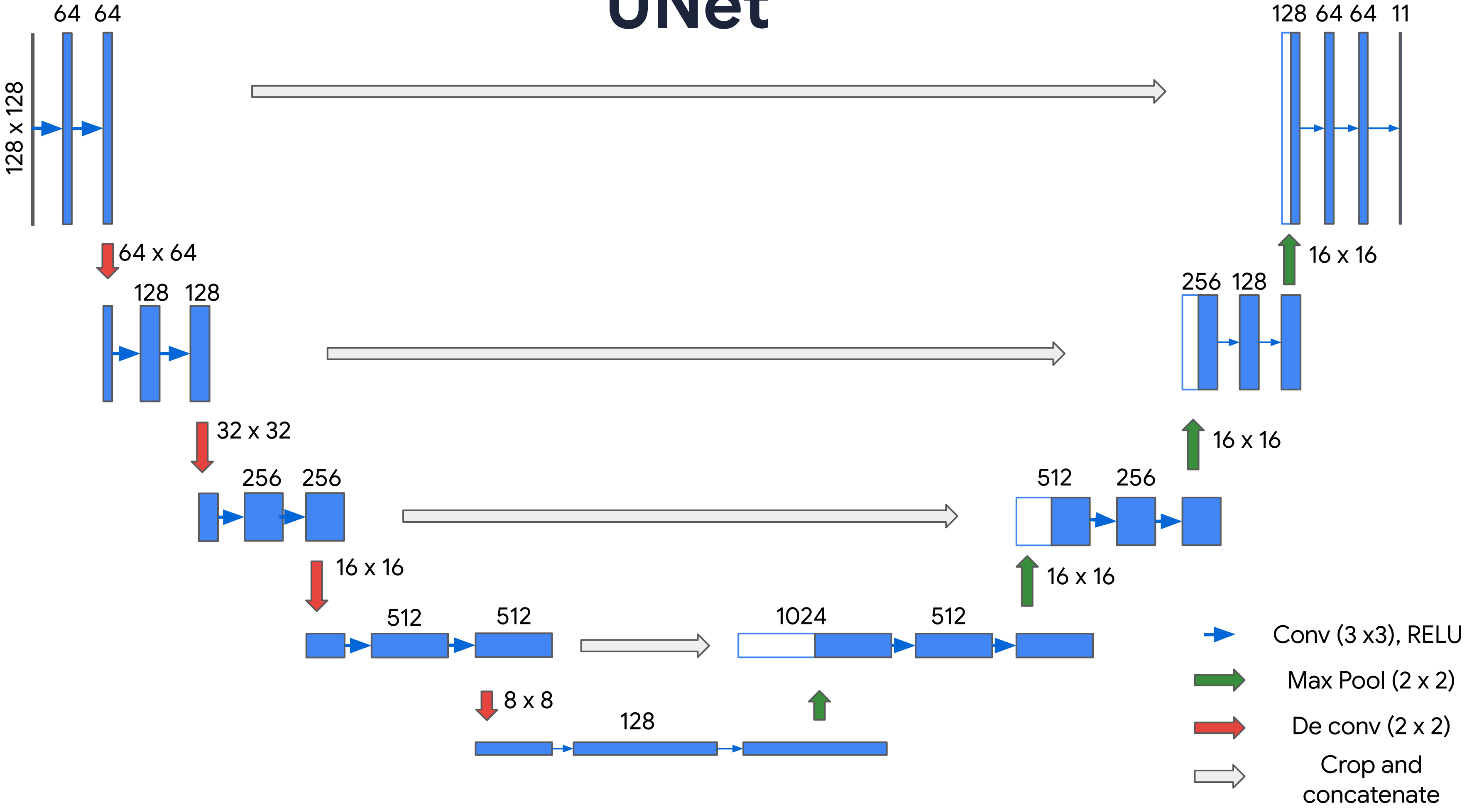


Input
RGB Image
Convolutional Encoder-Decoder
Pooling Indices
Output
Segmentation
Conv + Batch Normalisation + ReLU
Pooling
Upsampling
Softmax

SegNet

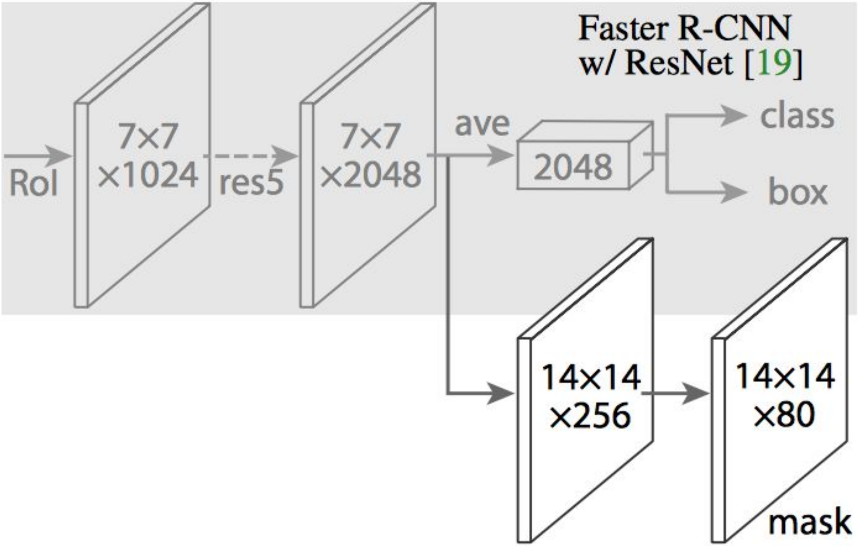

Input
RGB Image
Convolutional Encoder-Decoder
Pooling Indices
Conv + Batch Normalisation + ReLU
Pooling
Upsampling
Softmax
Output
Segmentation

UNet



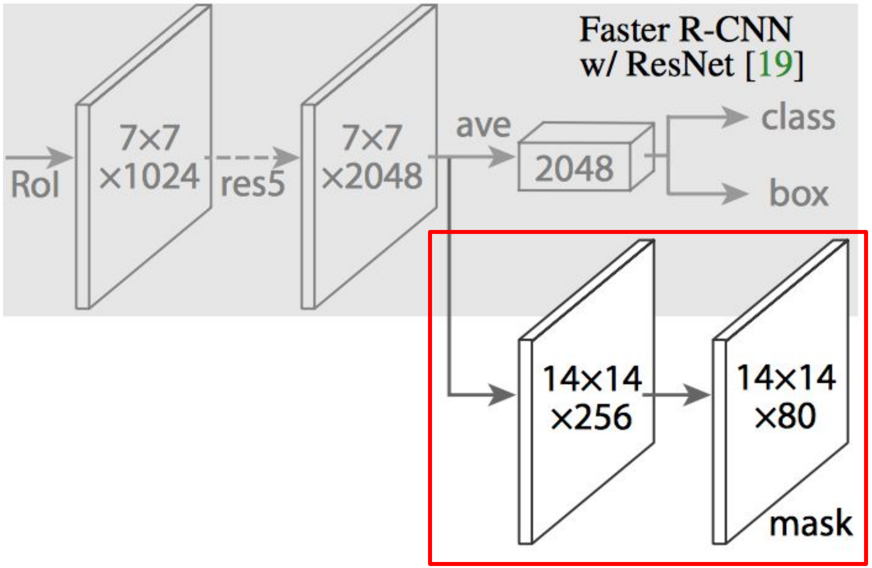
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
128
1024 512
16 x 16
512 256
16 x 16
256 128
16 x 16
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

Mask R-CNN



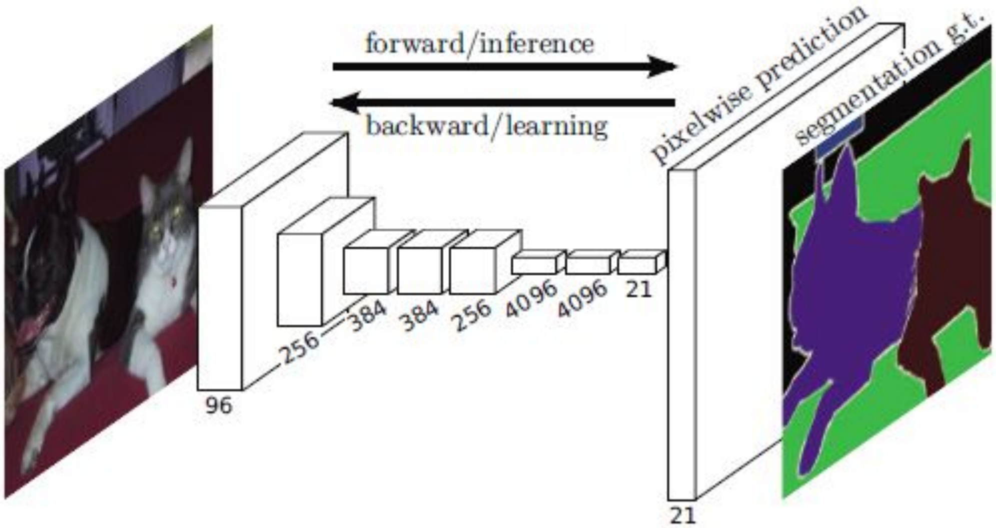
https://arxiv.org/abs/1703.06870

Mask R-CNN



https://arxiv.org/abs/1703.06870

Fully Convolutional Neural Networks

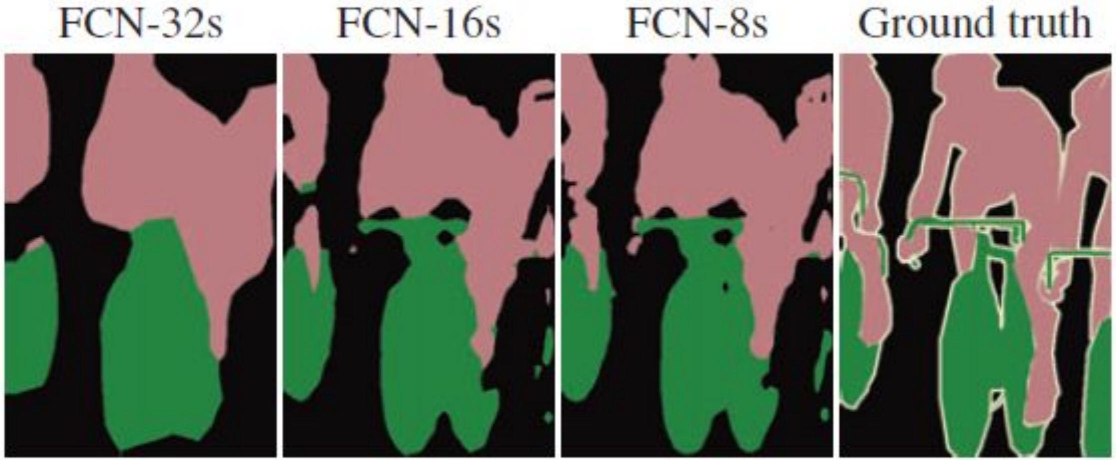


Fully Convolutional Networks for Semantic Segmentation
By Jonathan Long, Evan Shelhamer, Trevor Darrell
https://arxiv.org/pdf/1411.4038.pdf

Encoder

- Popular encoder architectures:
  - VGG-16
  - ResNet-50
  - MobileNet
- Reuse convolutional layers for feature extraction.
  - Do not reuse fully connected layers

Decoder



Image

1	2
2	1
0	1
0	1

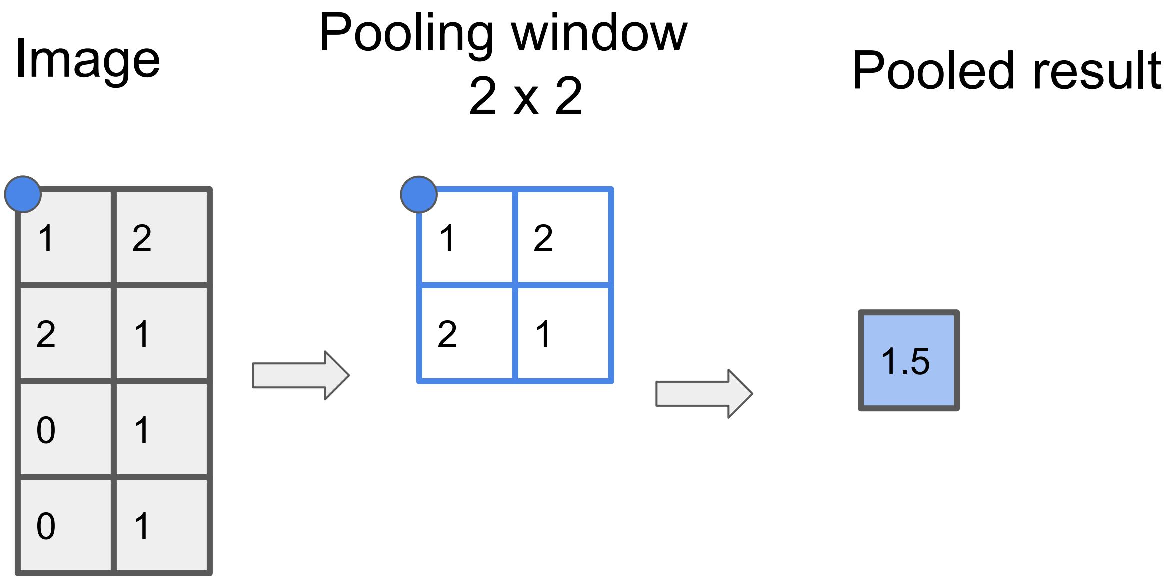


Image
1
2
2
1
0
1
0
1
Pooling window
2 x 2
1
2
2
1
Pooled result
1.5

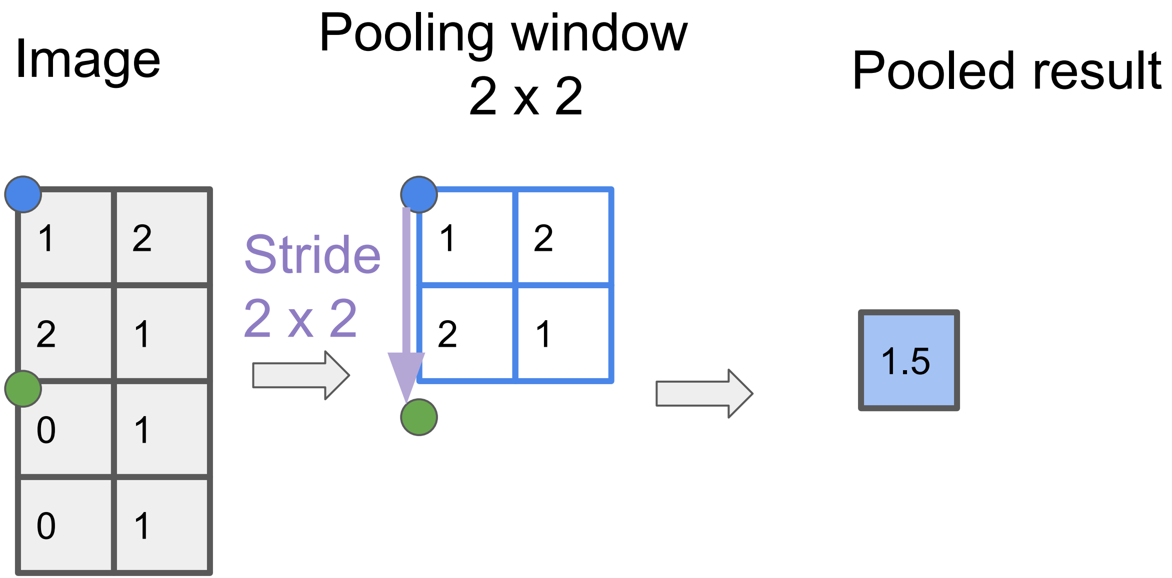


Image
Pooling window
2 x 2
Pooled result
1
2
2
1
0
1
0
1
Stride
2 x 2
1
2
2
1
1.5

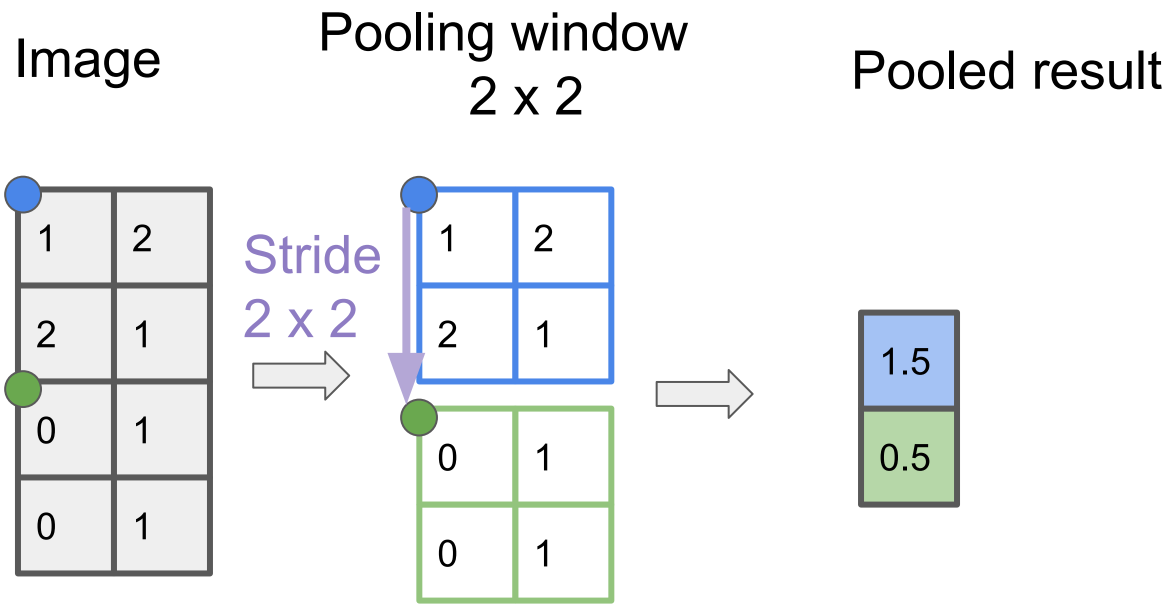


Image
Pooling window
2 x 2
Pooled result
1
2
2
1
0
1
0
1
Stride
2 x 2
1
2
2
1
0
1
0
1
1.5
0.5

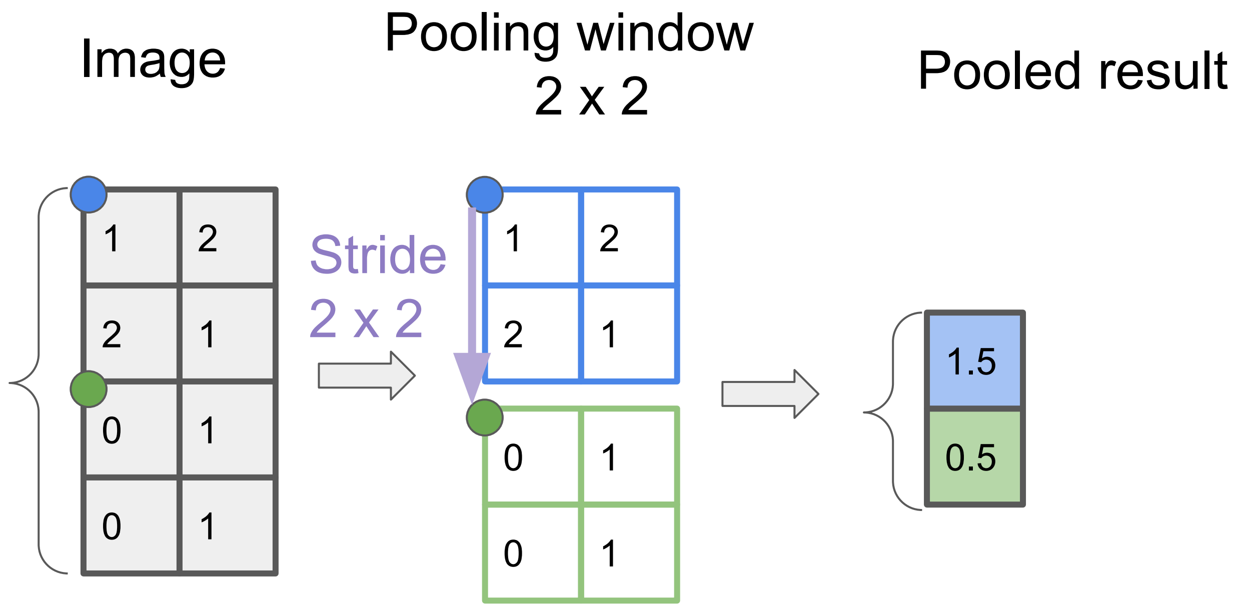


Image
Pooling window
2 x 2
Pooled result
1
2
2
1
0
1
0
1
Stride
2 x 2
1
2
2
1
0
1
0
1
1.5
0.5

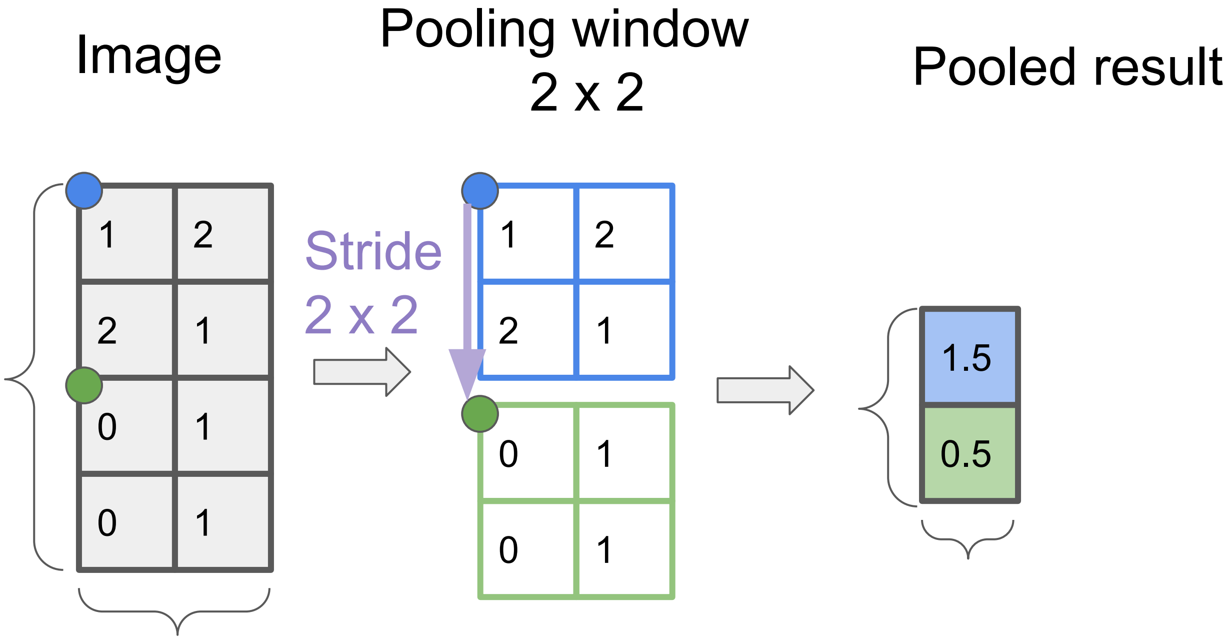


Image
Pooling window
2 x 2
Pooled result
1
2
2
1
0
1
0
1
Stride
2 x 2
1
2
2
1
0
1
0
1
1.5
0.5

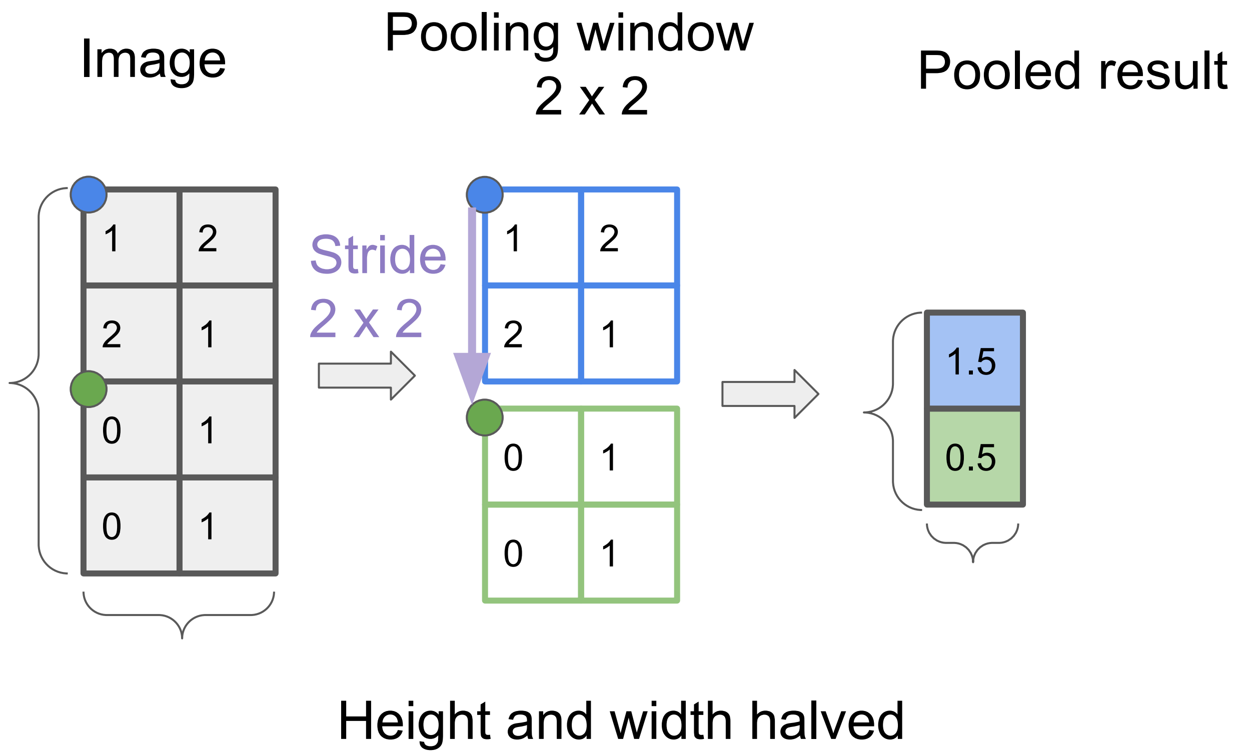


Image
Pooling window
2 x 2
Pooled result
1
2
2
1
0
1
0
1
Stride
2 x 2
1
2
2
1
0
1
0
1
1.5
0.5
Height and width halved

FCN-32

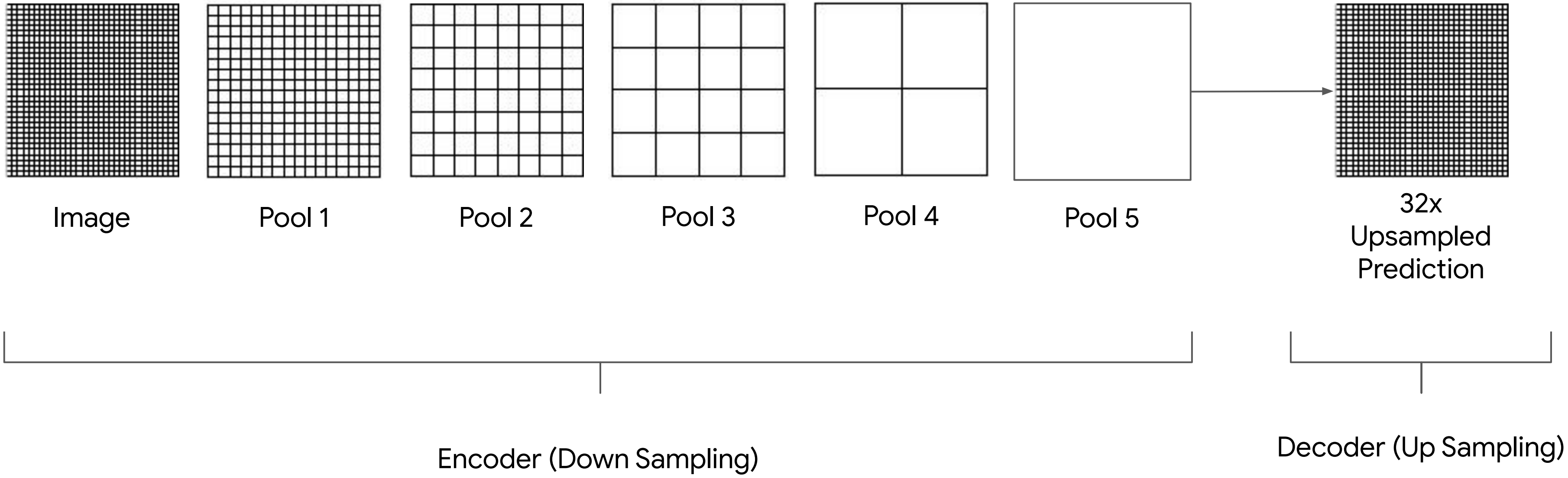
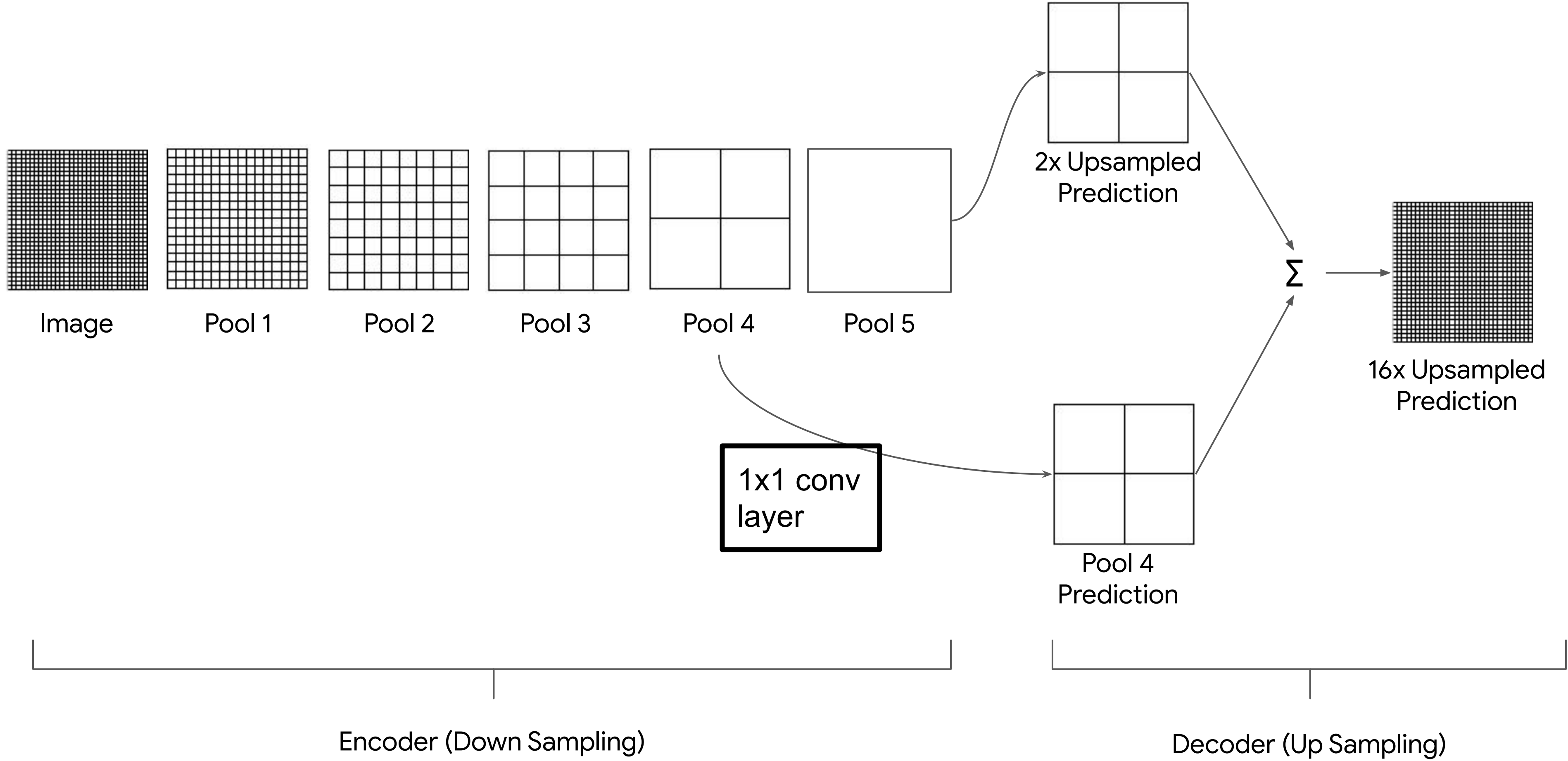


Image
Pool 1
Pool 2
Pool 3
Pool 4
Pool 5
32x
Upsampled
Prediction
Encoder (Down Sampling)
Decoder (Up Sampling)

FCN-16



2x Upsampled Prediction
Image
Pool 1
Pool 2
Pool 3
Pool 4
Pool 5
Σ
16x Upsampled Prediction
1x1 conv layer
Pool 4 Prediction
Encoder (Down Sampling)
Decoder (Up Sampling)

FCN-8

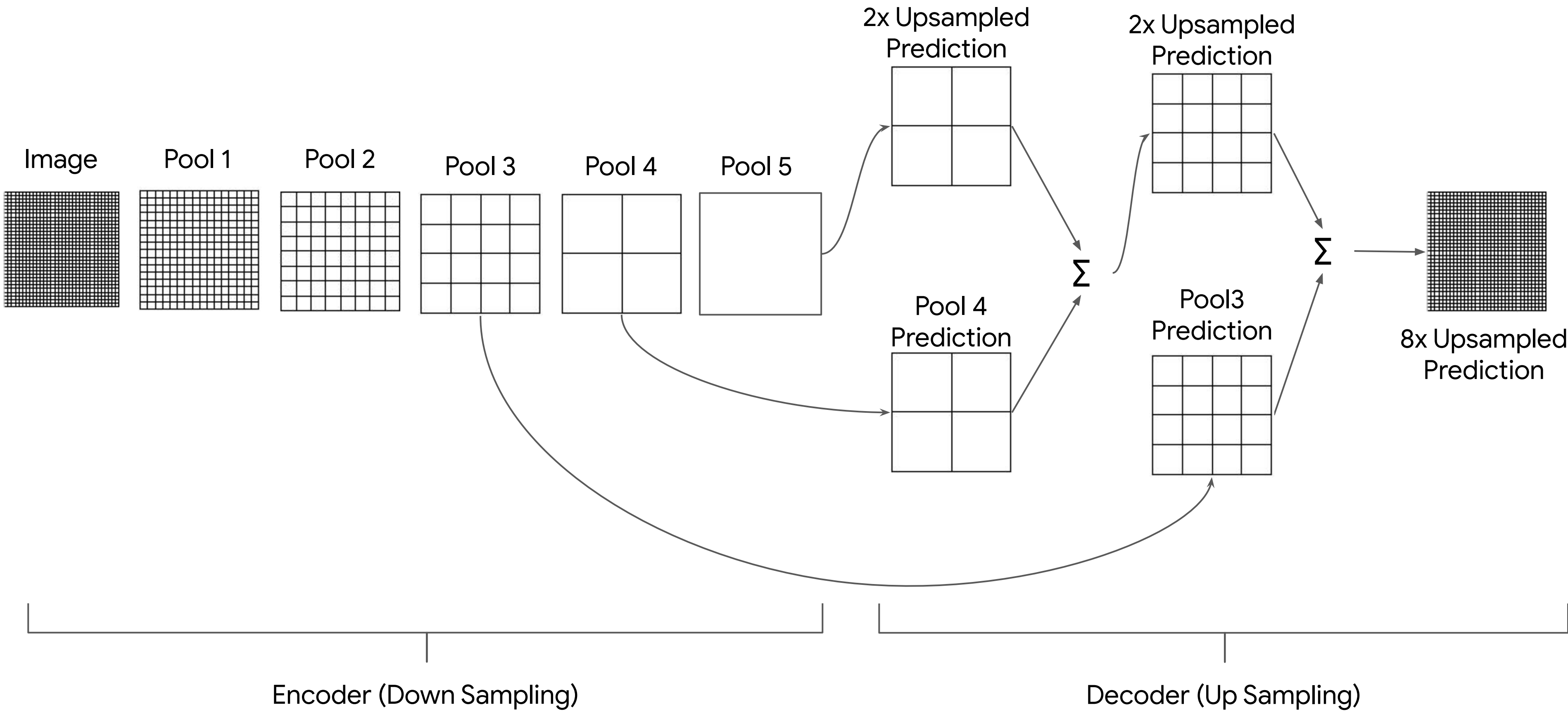
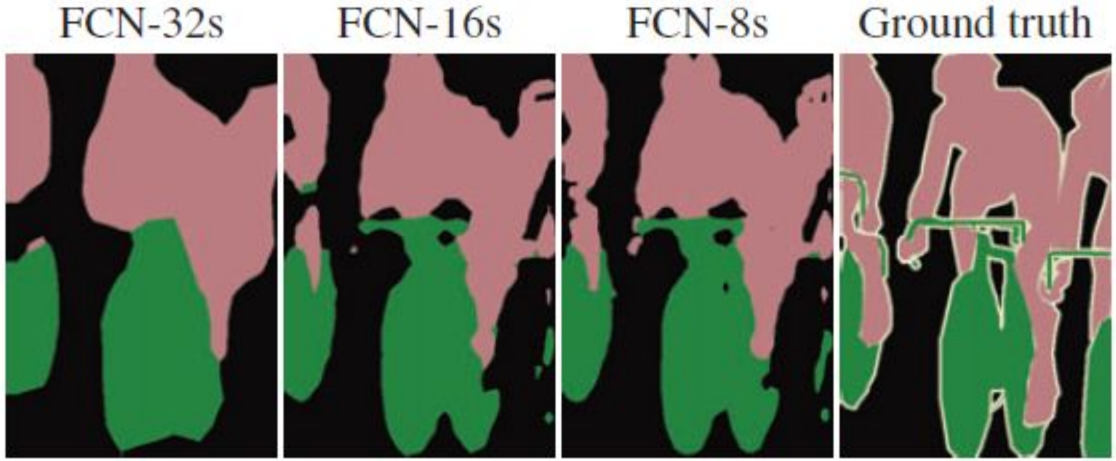


Image
Pool 1
Pool 2
Pool 3
Pool 4
Pool 5
2x Upsampled Prediction
Pool 4 Prediction
Σ
2x Upsampled Prediction
Pool3 Prediction
Σ
8x Upsampled Prediction
Encoder (Down Sampling)
Decoder (Up Sampling)

Comparison of Different FCNs



Upsampling

- Upsampling is increasing height and width of the feature map.
- Two types of layers used in TensorFlow:
  - Simple Scaling - UpSampling2D
  - Transposed Convolution(Deconvolution) - Conv2DTranspose

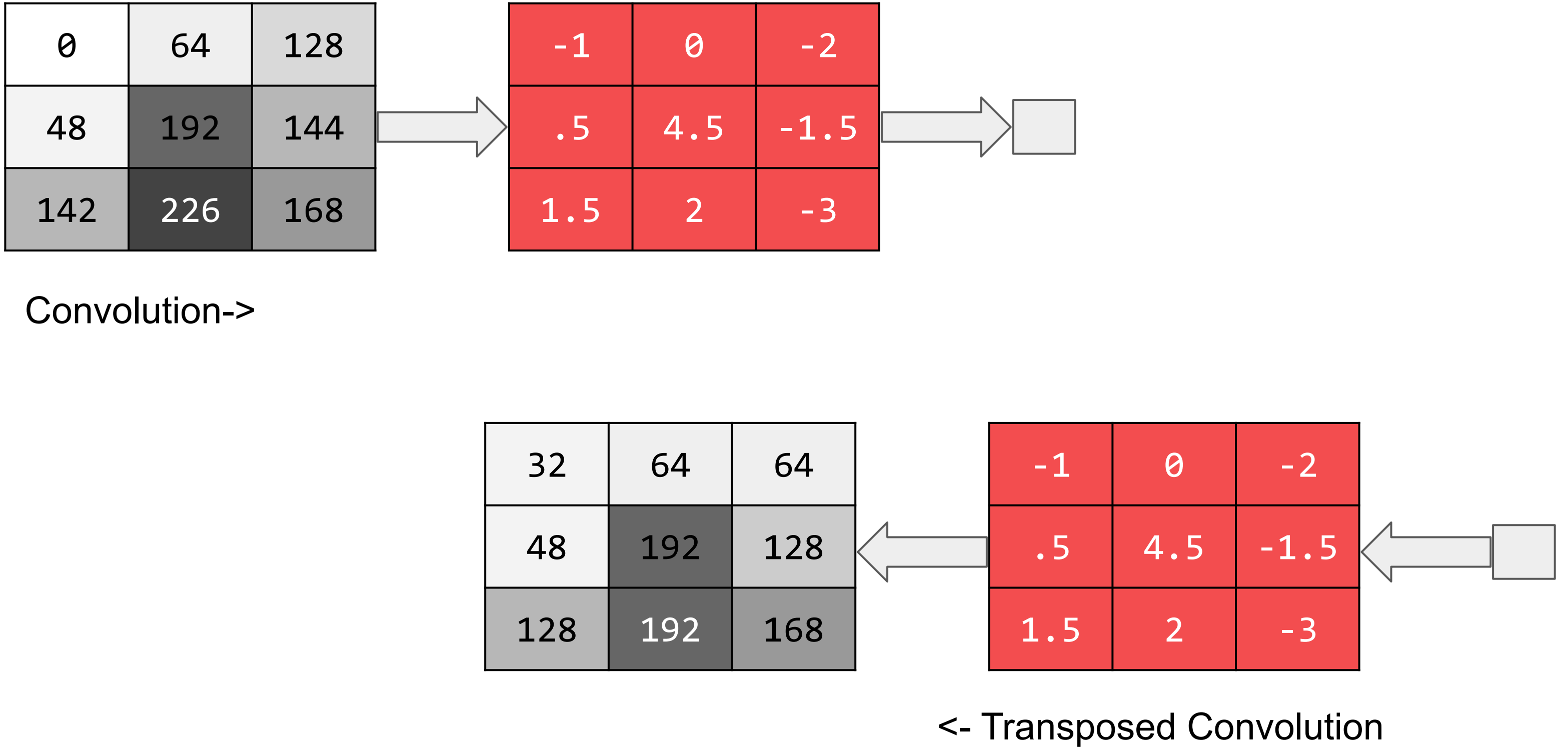
Simple Scaling - UpSampling2D

- Upsampling2D scales up the image
- Two Types of scaling:
    - Nearest
        - Copies value from nearest pixel.
    - Bilinear
        - linear interpolation from nearby pixels.

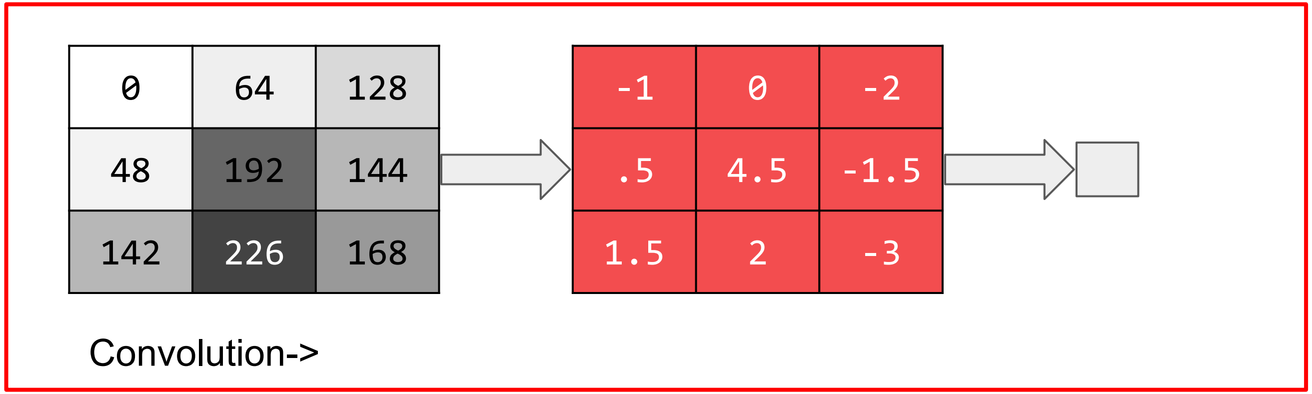
UpSampling2D - Usage

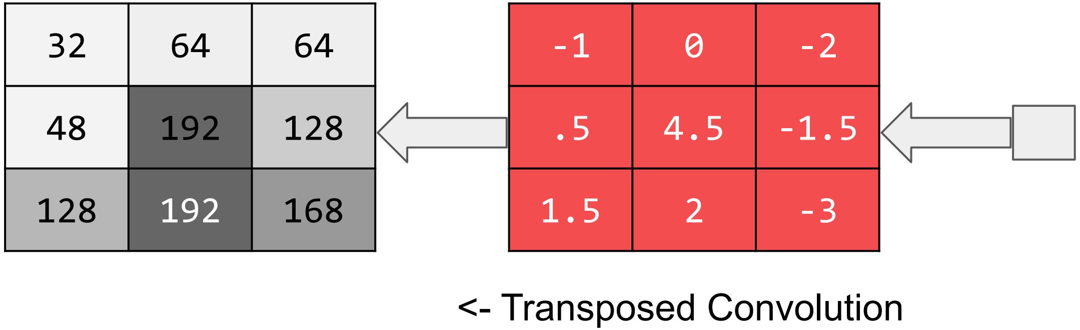
```
x = UpSampling2D(
    size=(2, 2),
    data_format=None,
    interpolation='nearest')(x)

size: int or tuple of two ints
data_format: 'channels_first', 'channels_last' or None
interpolation: 'nearest' or 'bilinear'
```

Transposed Convolution

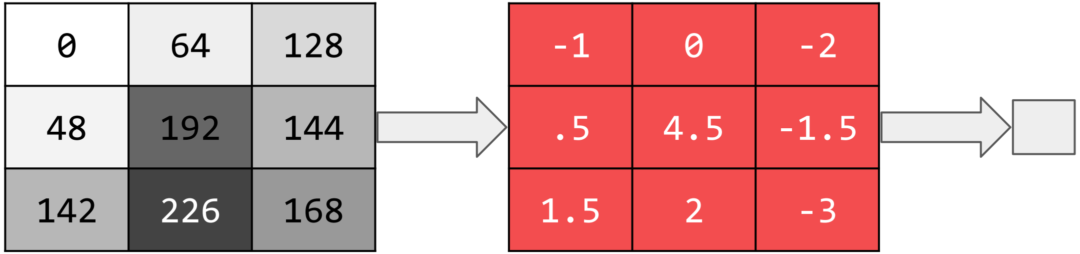


Transposed Convolution

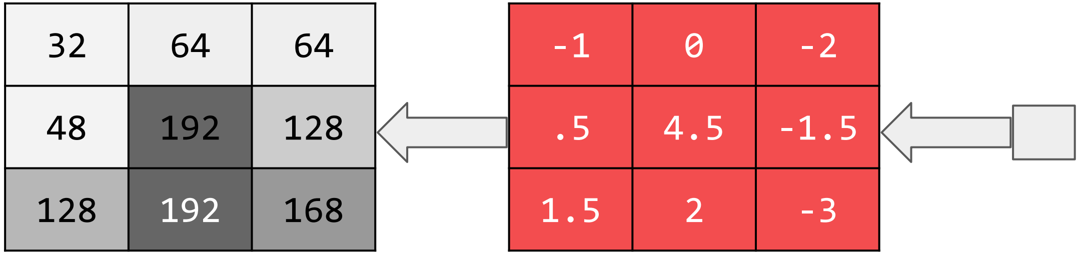




Transposed Convolution



Convolution->



<- Transposed Convolution

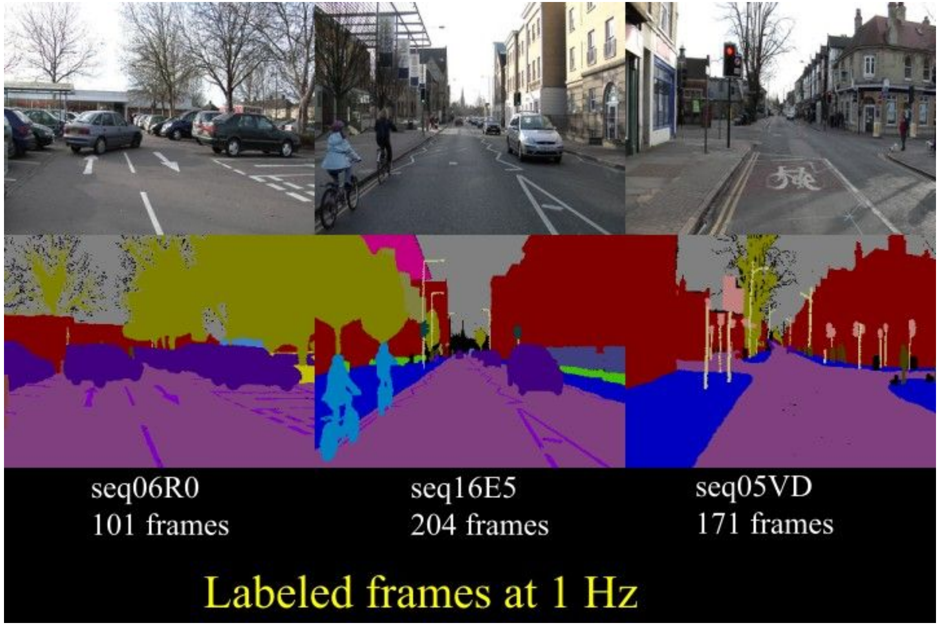
Conv2DTranspose

- Reverse of Convolution.
- Applied to output of a convolution operation.
- Uses a kernel of a specified size and stride in order to recreate the original input before the convolution operation.

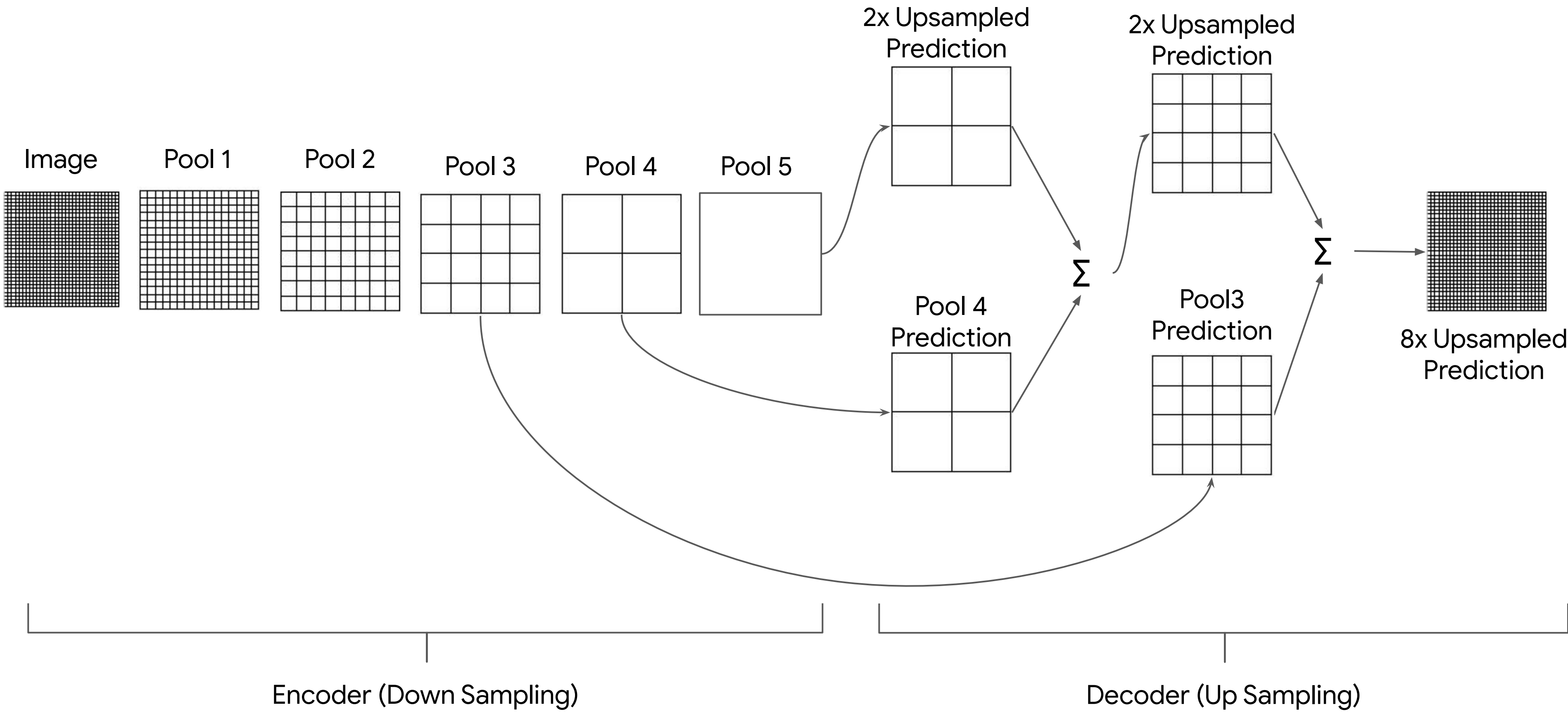
Conv2DTranspose - Usage

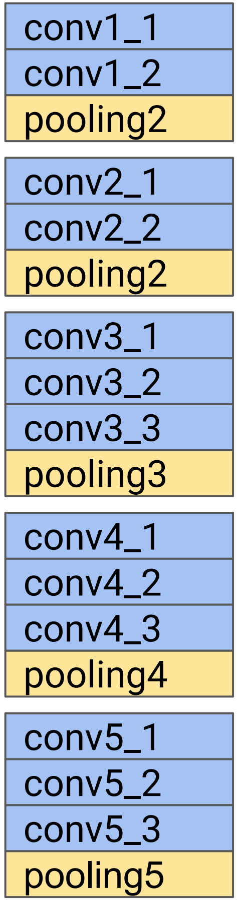
```
Conv2DTranspose(
    filters=32,
    kernel_size=(3, 3)
)
```

- The Cambridge Driving, labelled video database (aka CamVid) contains 10 minutes of 30fps video, segmented and labelled with 32 classes
- GitHub account [divamgupta] has taken a subsample of the CamVid dataset to create a smaller dataset.

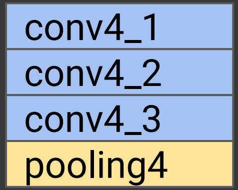


FCN-8

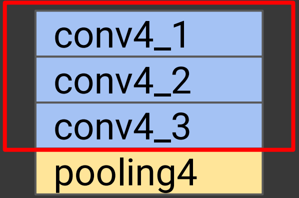

2x Upsampled Prediction
2x Upsampled Prediction
Image
Pool 1
Pool 2
Pool 3
Pool 4
Pool 5
Σ
Σ
Pool 4 Prediction
Pool3 Prediction
8x Upsampled Prediction
Encoder (Down Sampling)
Decoder (Up Sampling)


conv1_1
conv1_2
pooling2
conv2_1
conv2_2
pooling2
conv3_1
conv3_2
conv3_3
pooling3
conv4_1
conv4_2
conv4_3
pooling4
conv5_1
conv5_2
conv5_3
pooling5

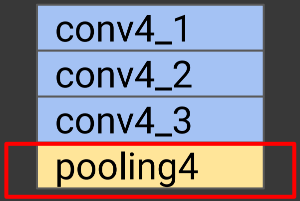
```
def block(x, n_convs, filters, kernel_size, activation, pool_size, pool_stride, block_name):
  for i in range(n_convs):
      x = tf.keras.layers.Conv2D(filters=filters,
                                 kernel_size=kernel_size, activation=activation,
                                 padding='same',
                                 name="{}_conv{}".format(block_name, i + 1))(x)

  x = tf.keras.layers.MaxPooling2D(pool_size=pool_size, strides=pool_stride,
                                    name="{}_pool{}".format(block_name, i+1 ))(x)

  return x
```



```python
def block(x, n_convs, filters, kernel_size, activation, pool_size, pool_stride, block_name):
  for i in range(n_convs):
      x = tf.keras.layers.Conv2D(filters=filters,
                                 kernel_size=kernel_size, activation=activation,
                                 padding='same',
                                 name="{}_conv{}".format(block_name, i + 1))(x)

  x = tf.keras.layers.MaxPooling2D(pool_size=pool_size, strides=pool_stride,
                                   name="{}_pool{}".format(block_name, i+1 ))(x)


  return x
```

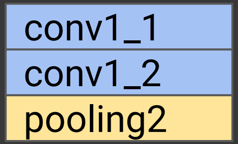


```python
def block(x, n_convs, filters, kernel_size, activation, pool_size, pool_stride, block_name):
  for i in range(n_convs):
      x = tf.keras.layers.Conv2D(filters=filters,
                                 kernel_size=kernel_size, activation=activation,
                                 padding='same',
                                 name="{}_conv{}".format(block_name, i + 1))(x)

  x = tf.keras.layers.MaxPooling2D(pool_size=pool_size, strides=pool_stride,
                                   name="{}_pool{}".format(block_name, i+1 ))(x)

  return x
```

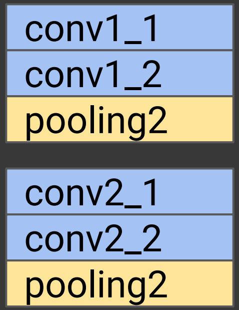


```python
def VGG_16(image_input):
  x = block(image_input, n_convs=2, filters=64, kernel_size=(3,3),
           activation='relu',pool_size=(2,2), pool_stride=(2,2),
           block_name='block1')
  p1= x

  x = block(x, n_convs=2, filters=128, kernel_size=(3,3),
           activation='relu',pool_size=(2,2), pool_stride=(2,2),
           block_name='block2')
  p2 = x

  ...
```

```
def VGG_16(image_input):
    x = block(image_input, n_convs=2, filters=64, kernel_size=(3,3),
              activation='relu',pool_size=(2,2), pool_stride=(2,2),
              block_name='block1')
    p1= x

    x = block(x, n_convs=2, filters=128, kernel_size=(3,3),
              activation='relu',pool_size=(2,2), pool_stride=(2,2),
              block_name='block2')
    p2 = x

    ...
```



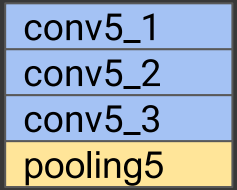
```python
def VGG_16(image_input):
  x = block(image_input, n_convs=2, filters=64, kernel_size=(3,3),
                activation='relu',pool_size=(2,2), pool_stride=(2,2),
                block_name='block1')
  p1= x

  x = block(x, n_convs=2, filters=128, kernel_size=(3,3),
                activation='relu',pool_size=(2,2), pool_stride=(2,2),
                block_name='block2')
  p2 = x

  ...
```



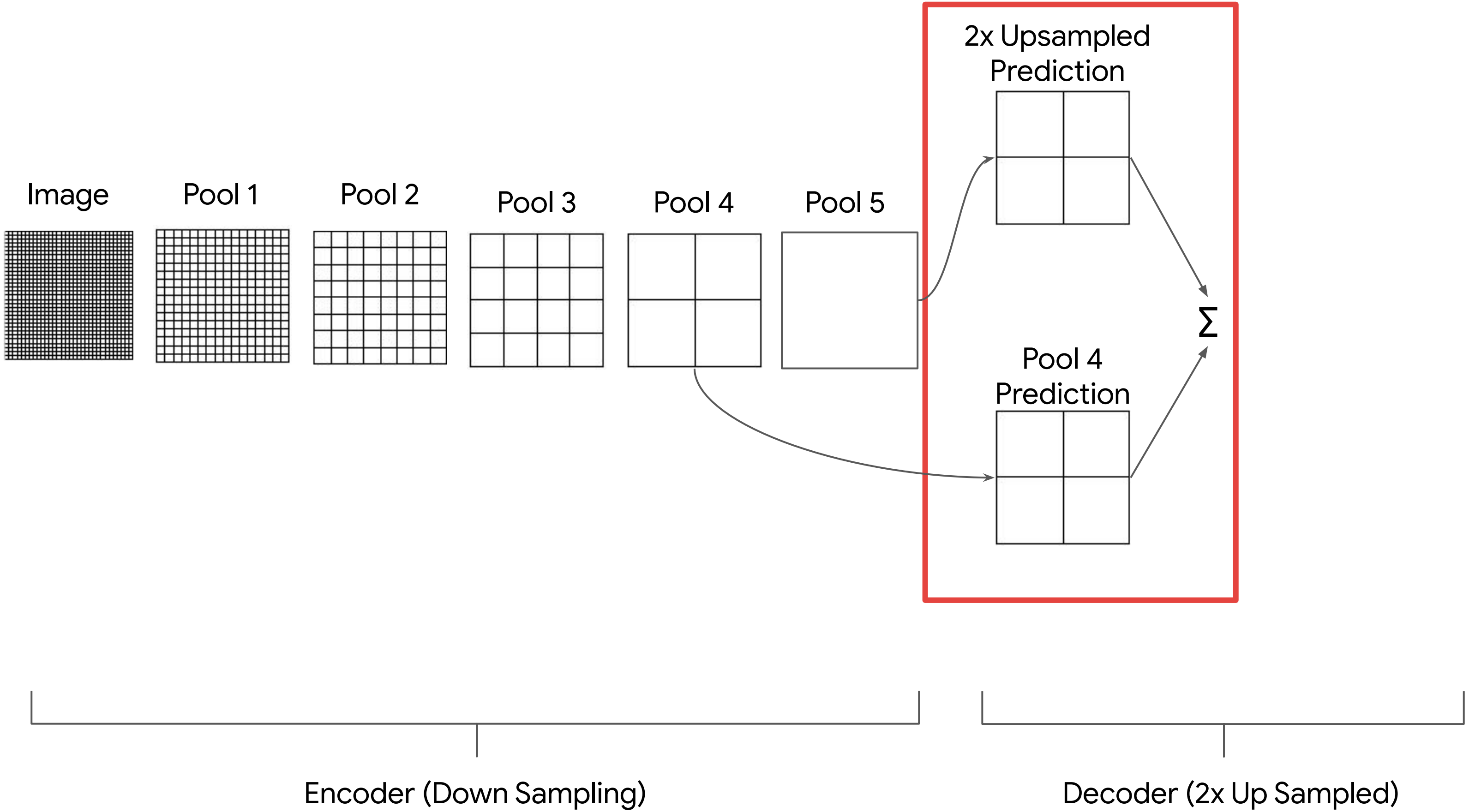
```
 x = block(x, n_convs=3, filters=256, kernel_size=(3,3), activation='relu',pool_size=(2,2),
           pool_stride=(2,2), block_name='block3')
p3 = x
```


```
x = block(x, n_convs=3, filters=512, kernel_size=(3,3), activation='relu',pool_size=(2,2),
          pool_stride=(2,2), block_name='block4')
p4 = x
```


```
x = block(x, n_convs=3, filters=512, kernel_size=(3,3), activation='relu',pool_size=(2,2),
          pool_stride=(2,2), block_name='block5')
p5 = x

...
```



2x Upsampling



Define Decoder - 2x UpSampling

```
def fcn8_decoder(convs, n_classes):
  f1, f2, f3, f4, f5 = convs

  o = tf.keras.layers.Conv2DTranspose(n_classes, kernel_size=(4,4),
                                      strides=(2,2), use_bias=False )(f5)

  o = tf.keras.layers.Cropping2D(cropping=(1,1))(o)

  o2 = f4
  o2 = ( tf.keras.layers.Conv2D(n_classes, (1,1),
                              activation='relu', padding='same'))(o2)

  o = tf.keras.layers.Add()([o, o2])
  ...
```

Define Decoder - 2x UpSampling

```
def fcn8_decoder(convs, n_classes):
  f1, f2, f3, f4, f5 = convs

  o = tf.keras.layers.Conv2DTranspose(n_classes, kernel_size=(4,4),
                                      strides=(2,2), use_bias=False )(f5)

  o = tf.keras.layers.Cropping2D(cropping=(1,1))(o)

  o2 = f4
  o2 = ( tf.keras.layers.Conv2D(n_classes, (1,1),
                              activation='relu', padding='same'))(o2)

  o = tf.keras.layers.Add()([o, o2])
  ...
```

Define Decoder - 2x UpSampling

```
def fcn8_decoder(convs, n_classes):
  f1, f2, f3, f4, f5 = convs

  o = tf.keras.layers.Conv2DTranspose(n_classes, kernel_size=(4,4),
                                      strides=(2,2), use_bias=False )(f5)

  o = tf.keras.layers.Cropping2D(cropping=(1,1))(o)

  o2 = f4
  o2 = ( tf.keras.layers.Conv2D(n_classes, (1,1),
                               activation='relu', padding='same'))(o2)

  o = tf.keras.layers.Add()([o, o2])
  ...
```

Define Decoder - 2x UpSampling

```
def fcn8_decoder(convs, n_classes):
  f1, f2, f3, f4, f5 = convs

  o = tf.keras.layers.Conv2DTranspose(n_classes, kernel_size=(4,4),
                                      strides=(2,2), use_bias=False )(f5)

  o = tf.keras.layers.Cropping2D(cropping=(1,1))(o)

  o2 = f4
  o2 = ( tf.keras.layers.Conv2D(n_classes, (1,1),
                                activation='relu', padding='same'))(o2)

  o = tf.keras.layers.Add()([o, o2])
  ...
```

Define Decoder - 2x UpSampling

```
def fcn8_decoder(convs, n_classes):
  f1, f2, f3, f4, f5 = convs

  o = tf.keras.layers.Conv2DTranspose(n_classes, kernel_size=(4,4),
                                      strides=(2,2), use_bias=False )(f5)

  o = tf.keras.layers.Cropping2D(cropping=(1,1))(o)

  o2 = f4
  o2 = ( tf.keras.layers.Conv2D(n_classes, (1,1),
                               activation='relu', padding='same'))(o2)

  o = tf.keras.layers.Add()([o, o2])
  ...
```

1 x 1 Convolutions

(B, F, H, W) - *B = # batches; F= # filters, H, W = Height/Width*

1 x 1 Convolutions

(B, F, H, W) - *B = # batches; F= # filters, H, W = Height/Width*

Apply a layer with N 1x1 Convolutions with stride of 1:

1 x 1 Convolutions

(B, F, H, W) - *B = # batches; F= # filters, H, W = Height/Width*

Apply a layer with N 1x1 Convolutions with stride of 1:

(B, N, H, W) - *B = # batches; N= # filters, H, W = Height/Width*

1 x 1 Convolutions

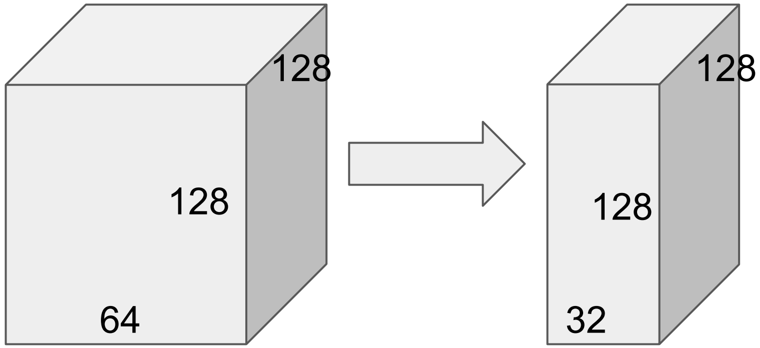
(B, F, H, W) - *B = # batches; F= # filters, H, W = Height/Width*

(B, N, H, W) - *B = # batches; N= # filters, H, W = Height/Width*

1 x 1 Convolutions

(B, F, H, W) - *B = # batches; F= # filters, H, W = Height/Width*

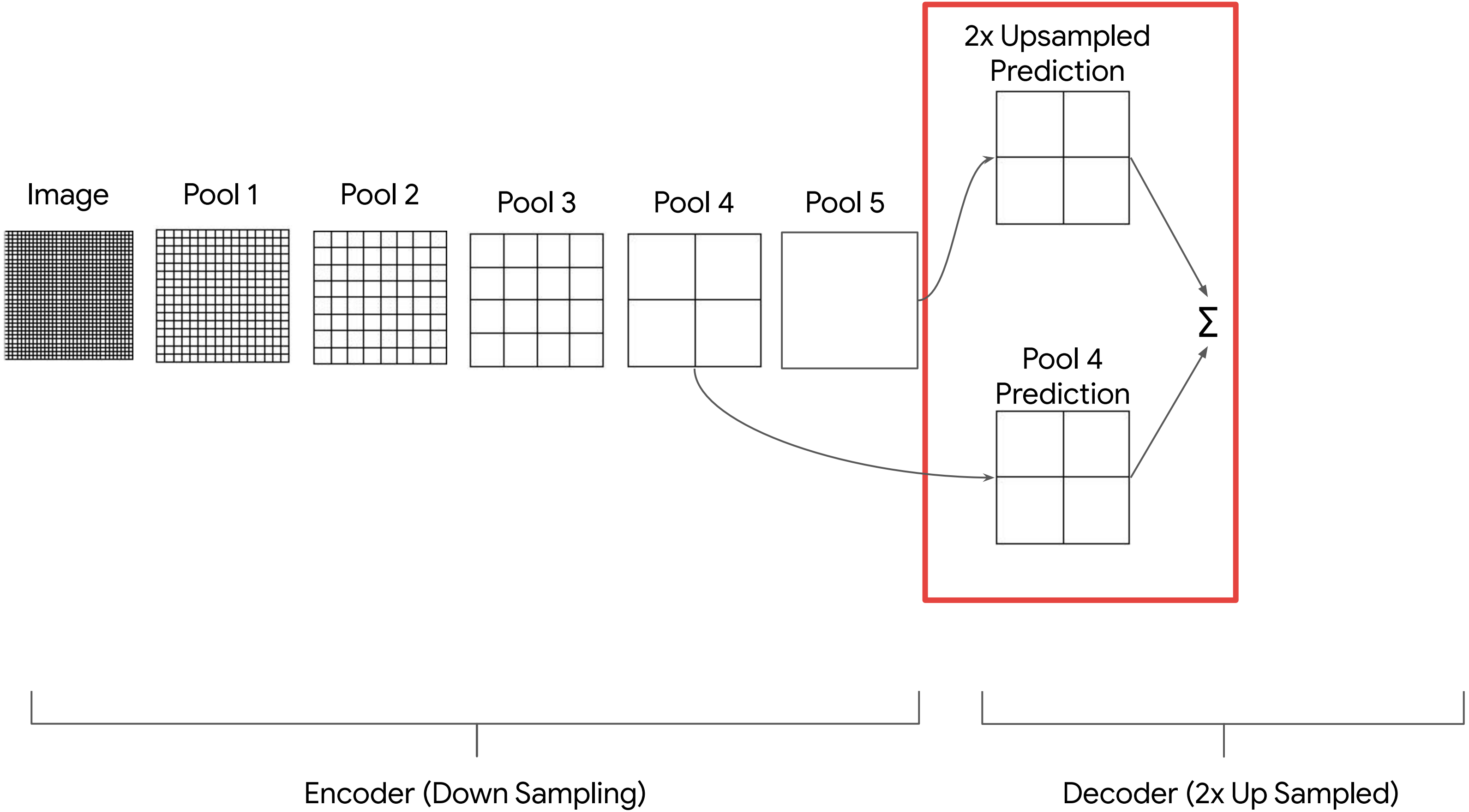
(B, N, H, W) - *B = # batches; N= # filters, H, W = Height/Width*



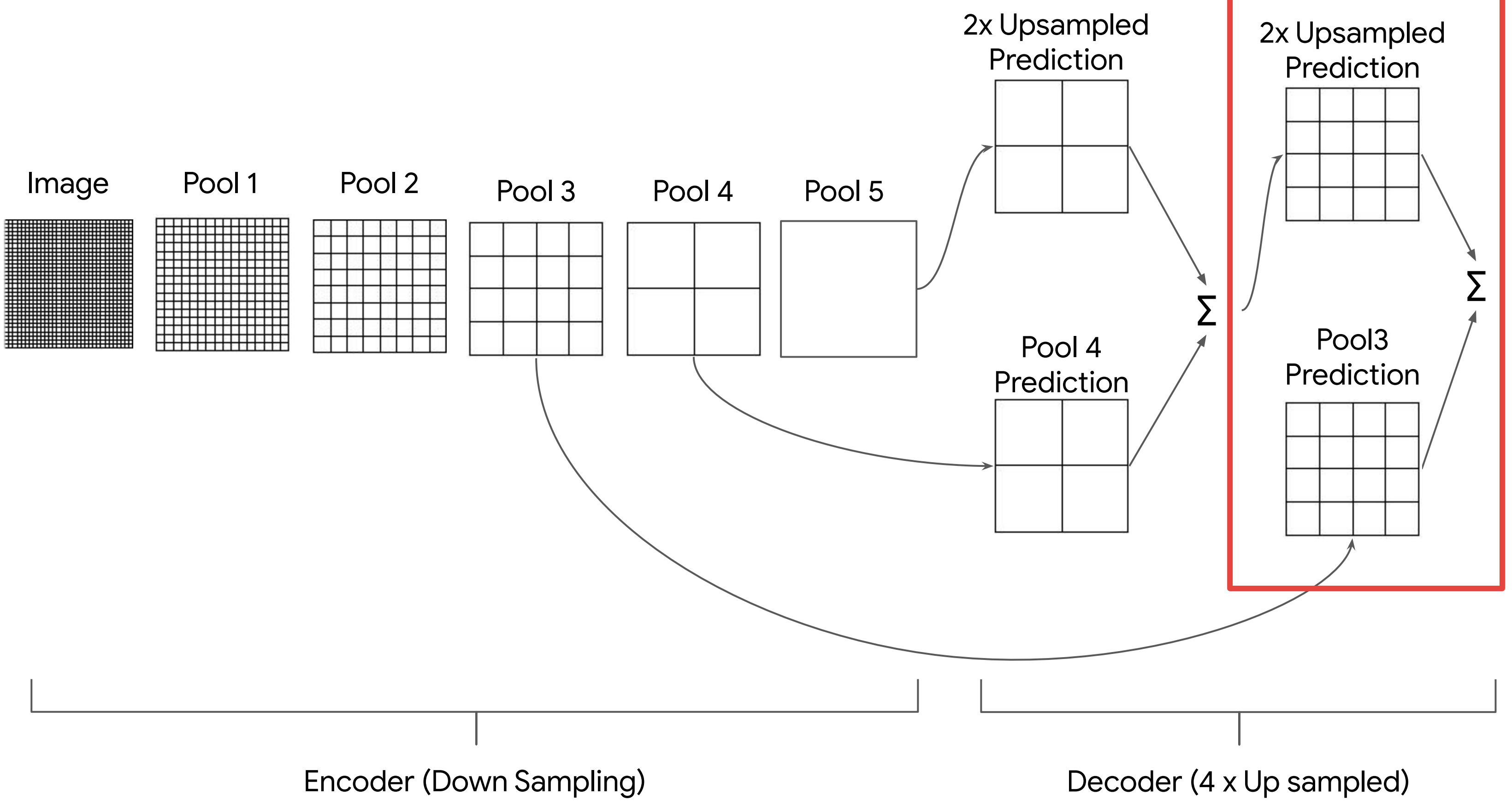
Define Decoder - 2x UpSampling

```
def fcn8_decoder(convs, n_classes):
  f1, f2, f3, f4, f5 = convs

  o = tf.keras.layers.Conv2DTranspose(n_classes, kernel_size=(4,4),
                                       strides=(2,2), use_bias=False )(f5)

  o = tf.keras.layers.Cropping2D(cropping=(1,1))(o)

  o2 = f4
  o2 = ( tf.keras.layers.Conv2D(n_classes, (1,1),
                                 activation='relu', padding='same'))(o2)

  o = tf.keras.layers.Add()([o, o2])
  ...
```

2x Upsampling



2 x 2 Upsampled



```python
def fcn8_decoder(convs, n_classes):
  ...
  o = (tf.keras.layers.Conv2DTranspose( n_classes, kernel_size=(4,4),
                                        strides=(2,2))(o)

  o = tf.keras.layers.Cropping2D(cropping=(1, 1))(o)

  o2 = ( tf.keras.layers.Conv2D(n_classes,(1,1), activation='relu',
                                padding='same'))(f3)


  o = tf.keras.layers.Add()([o, o2])

  ...
```

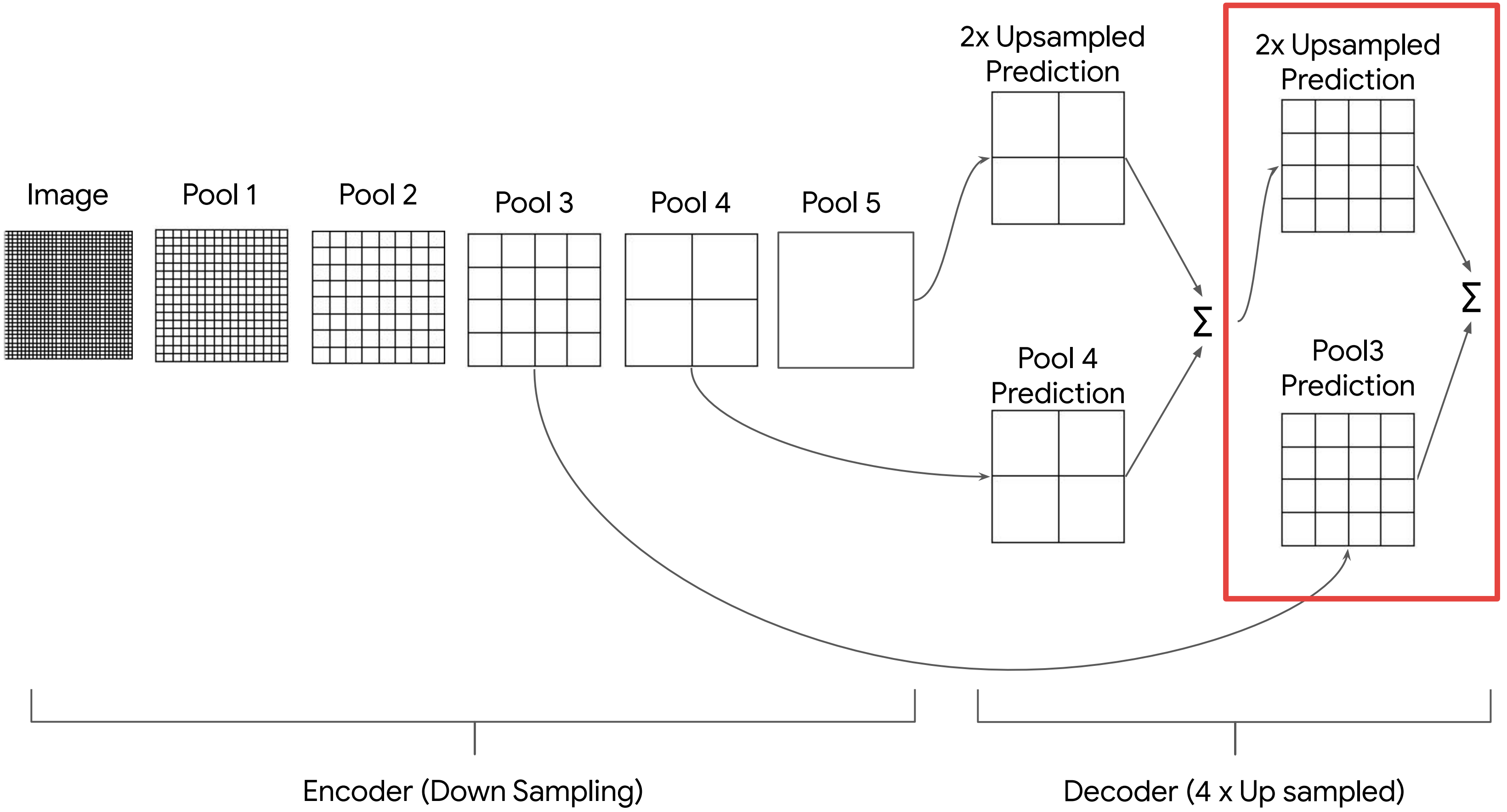
```python
def fcn8_decoder(convs, n_classes):
  ...
  o = (tf.keras.layers.Conv2DTranspose( n_classes, kernel_size=(4,4),
                                        strides=(2,2))(o)

  o = tf.keras.layers.Cropping2D(cropping=(1, 1))(o)

  o2 = ( tf.keras.layers.Conv2D(n_classes,(1,1), activation='relu',
                               padding='same'))(f3)


  o = tf.keras.layers.Add()([o, o2])

  ...
```

```python
def fcn8_decoder(convs, n_classes):
  ...
  o = (tf.keras.layers.Conv2DTranspose( n_classes, kernel_size=(4,4),
                                         strides=(2,2))(o)

  o = tf.keras.layers.Cropping2D(cropping=(1, 1))(o)

  o2 = ( tf.keras.layers.Conv2D(n_classes,(1,1), activation='relu',
                                 padding='same'))(f3)


  o = tf.keras.layers.Add()([o, o2])

  ...
```

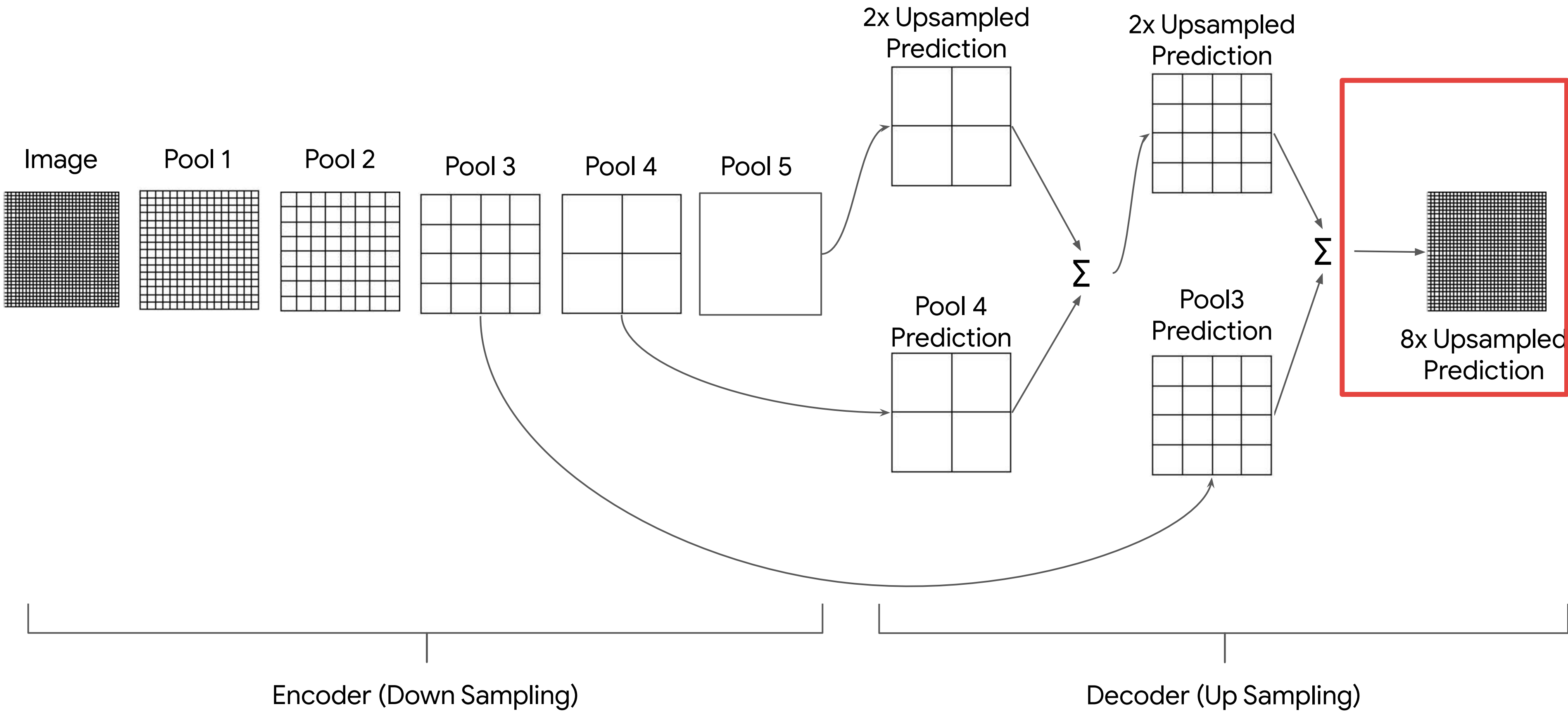
```python
def fcn8_decoder(convs, n_classes):
  ...
  o = (tf.keras.layers.Conv2DTranspose( n_classes, kernel_size=(4,4),
                                       strides=(2,2))(o)

  o = tf.keras.layers.Cropping2D(cropping=(1, 1))(o)

  o2 = ( tf.keras.layers.Conv2D(n_classes,(1,1), activation='relu',
                               padding='same'))(f3)


  o = tf.keras.layers.Add()([o, o2])

  ...
```

```python
def fcn8_decoder(convs, n_classes):
  ...
  o = (tf.keras.layers.Conv2DTranspose( n_classes, kernel_size=(4,4),
                                       strides=(2,2))(o)

  o = tf.keras.layers.Cropping2D(cropping=(1, 1))(o)

  o2 = ( tf.keras.layers.Conv2D(n_classes,(1,1), activation='relu',
                               padding='same'))(f3)


  o = tf.keras.layers.Add()([o, o2])

  ...
```

2 x 2 Upsampled



2 x 2 x 8 Up Sampled



Define Decoder

```
def fcn8_decoder(convs, n_classes):
  …

  o = tf.keras.layers.Conv2DTranspose(n_classes , kernel_size=(8,8),
                                      strides=(8,8))(o)

  o = (tf.keras.layers.Activation('softmax'))(o)

  return o
```

Define Decoder

```
def fcn8_decoder(convs, n_classes):
  …

  o = tf.keras.layers.Conv2DTranspose(n_classes , kernel_size=(8,8),
                                                   strides=(8,8))(o)

  o = (tf.keras.layers.Activation('softmax'))(o)

  return o
```

Define Decoder

```
def fcn8_decoder(convs, n_classes):
  …

  o = tf.keras.layers.Conv2DTranspose(n_classes , kernel_size=(8,8),
                                      strides=(8,8))(o)

  o = (tf.keras.layers.Activation('softmax'))(o)

  return o
```

Define Final Model

```
def segmentation_model():
  inputs = tf.keras.layers.Input(shape=(224,224,3,))
  convs = VGG_16(image_input=inputs)
  outputs = fcn8_decoder(convs, 12)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)
  return model

model = segmentation_model()
```

Define Final Model

```
def segmentation_model():
  inputs = tf.keras.layers.Input(shape=(224,224,3,))
  convs = VGG_16(image_input=inputs)
  outputs = fcn8_decoder(convs, 12)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)
  return model

model = segmentation_model()
```

Define Final Model

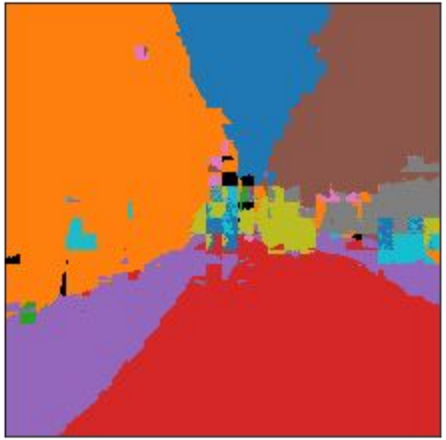
```
def segmentation_model():
  inputs = tf.keras.layers.Input(shape=(224,224,3,))
  convs = VGG_16(image_input=inputs)
  outputs = fcn8_decoder(convs, 12)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)
  return model

model = segmentation_model()
```

Define Final Model

```
def segmentation_model():
  inputs = tf.keras.layers.Input(shape=(224,224,3,))
  convs = VGG_16(image_input=inputs)
  outputs = fcn8_decoder(convs, 12)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)
  return model

model = segmentation_model()
```

Sample Visualization of Predicted Segments

Original Image

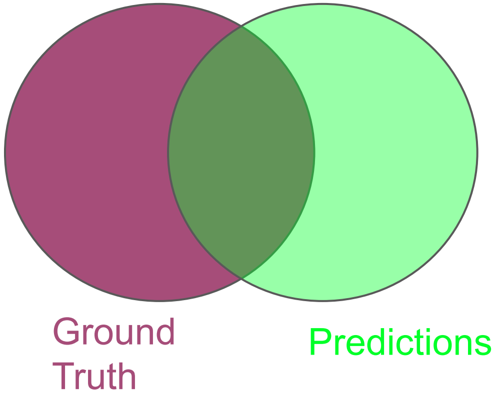


Predicted Segments

Ground Truth Segments

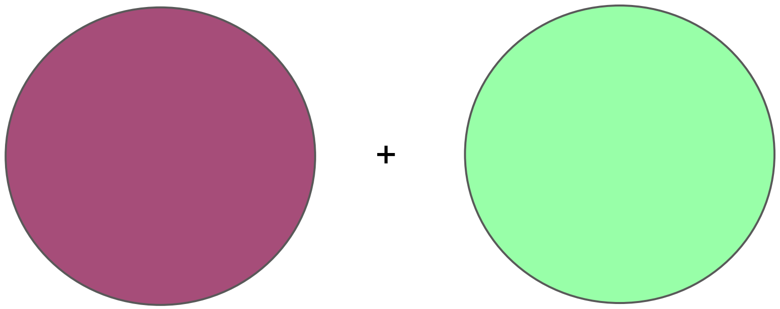
Area of Overlap

Area of Overlap = sum(True Positives)



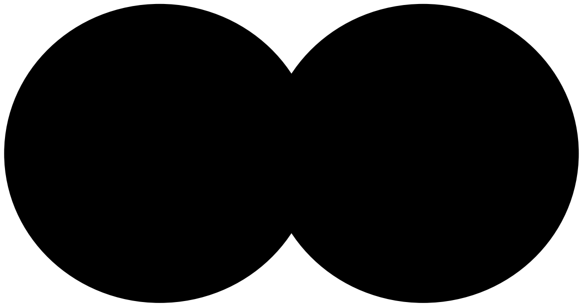
Combined Area

Combined Area = Total Pixels in predicted segmentation mask + Total Pixels in True Segmentation mask



Area of Union

Area of Union = Total Pixels in predicted segmentation mask + Total Pixels in True Segmentation mask - Area of Overlap



Calculate Areas

```
def class_wise_metrics(y_true, y_pred):
 ...
  smoothening_factor = 0.00001

  for i in range(n_classes):
    intersection = np.sum((y_pred == i) * (y_true == i))
    y_true_area = np.sum((y_true == i))
    y_pred_area = np.sum((y_pred == i))
    combined_area = y_true_area + y_pred_area
    union_area = combined_area - intersection
  ...
```

Calculate Areas

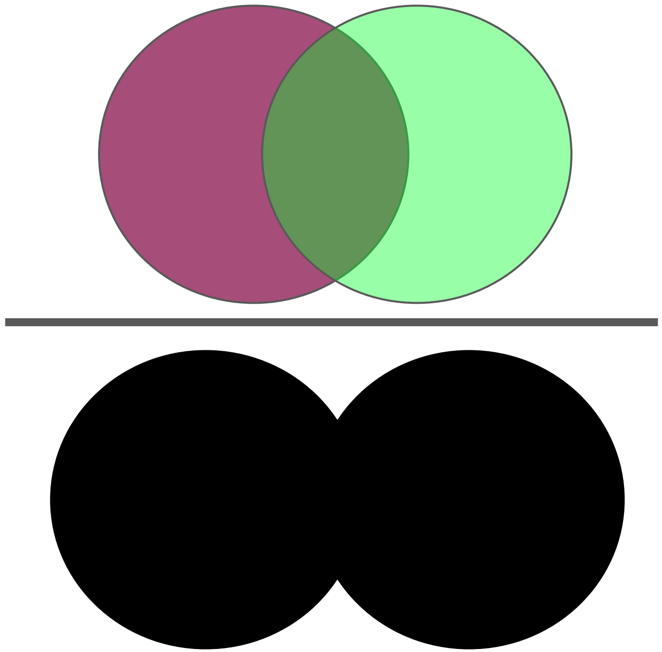
```
def class_wise_metrics(y_true, y_pred):
 ...
  smoothening_factor = 0.00001

  for i in range(n_classes):
    intersection = np.sum((y_pred == i) * (y_true == i))
    y_true_area = np.sum((y_true == i))
    y_pred_area = np.sum((y_pred == i))
    combined_area = y_true_area + y_pred_area
    union_area = combined_area - intersection
  ...
```

Calculate Areas

```
def class_wise_metrics(y_true, y_pred):
 ...
  smoothening_factor = 0.00001

  for i in range(n_classes):
    intersection = np.sum((y_pred == i) * (y_true == i))
    y_true_area = np.sum((y_true == i))
    y_pred_area = np.sum((y_pred == i))
    combined_area = y_true_area + y_pred_area
    union_area = combined_area - intersection
   ...
```

Intersection Over Union

$$\text{IOU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

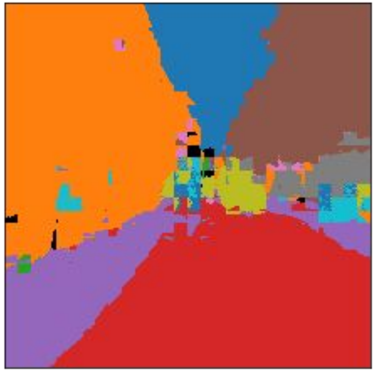


Calculate IOU

```
def class_wise_metrics(y_true, y_pred):
    class_wise_iou = []

    ...
    for i in range(n_classes):
        ...
        iou = (intersection) / (union_area)
        class_wise_iou.append(iou)

    return class_wise_iou
```

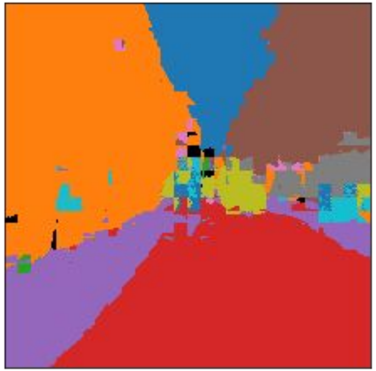
IOU Results

sky	0.8779669959482955
building	0.7570989578412737
column/pole	4.57875457665808e-10
road	0.915543155822588
side walk	0.7235628237658467
vegetation	0.7664541807647628
traffic light	3.0202657798187055e-05
fence	0.006380242448568188
vehicle	0.2950299461448835
pedestrian	0.000126433276608086
byciclist	0.023621930993270864
void	0.16456276759816527



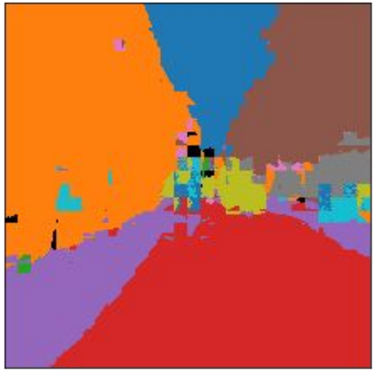
IOU Results

sky	0.8779669959482955
building	0.7570989578412737
column/pole	4.57875457665808e-10
road	0.915543155822588
side walk	0.7235628237658467
vegetation	0.7664541807647628
traffic light	3.0202657798187055e-05
fence	0.006380242448568188
vehicle	0.2950299461448835
pedestrian	0.0001264333276608086
byciclist	0.023621930993270864
void	0.16456276759816527



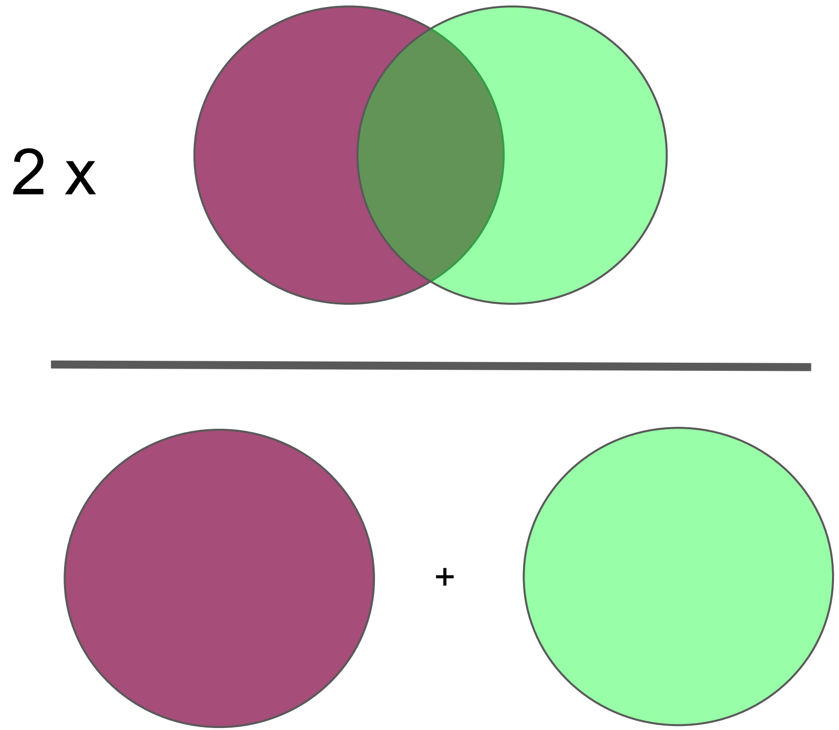
IOU Results

sky	0.8779669959482955
building	0.7570989578412737
column/pole	4.578754576658O8e-10
road	0.915543155822588
side walk	0.7235628237658467
vegetation	0.7664541807647628
traffic light	3.0202657798187055e-05
fence	0.006380242448568188
vehicle	0.2950299461448835
pedestrian	0.000126433276608086
byciclist	0.023621930993270864
void	0.16456276759816527



Dice Score

$$\textbf{Dice Score} = 2 \times \frac{\text{Area of Overlap}}{\text{Combined Area}}$$

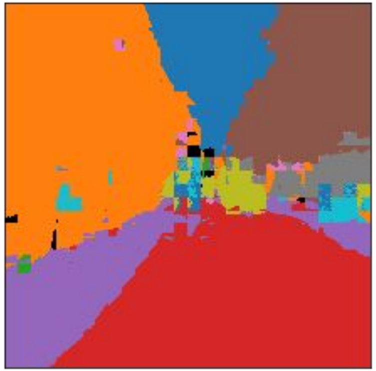


Calculate Dice Score

```
def class_wise_metrics(y_true, y_pred):
  class_wise_dice_score = []

  ...
  for i in range(n_classes):
    ...
    dice_score = 2 * (intersection) / (combined_area)
    class_wise_dice_score.append(dice_score)

  return class_wise_dice_score
```

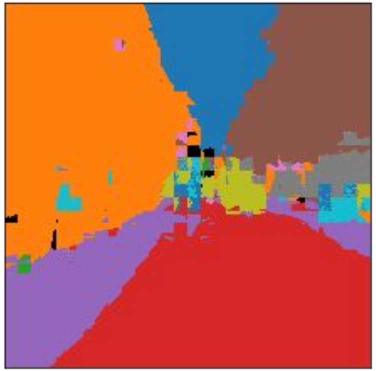
Dice Score Results

sky	0.9350185576821789
building	0.861760180856767
column/pole	9.157509153316 16e-10
road	0.955909714740 2678
side walk	0.839612938734 6395
vegetation	0.867788351509 2748
traffic light	6.04034912686 4078e-05
fence	0.01267958606 5167121
vehicle	0.45563416820 437186
pedestrian	0.00025283468 86971326
byciclist	0.04615362426 586659
void	0.282617257200 9191



Dice Score Results

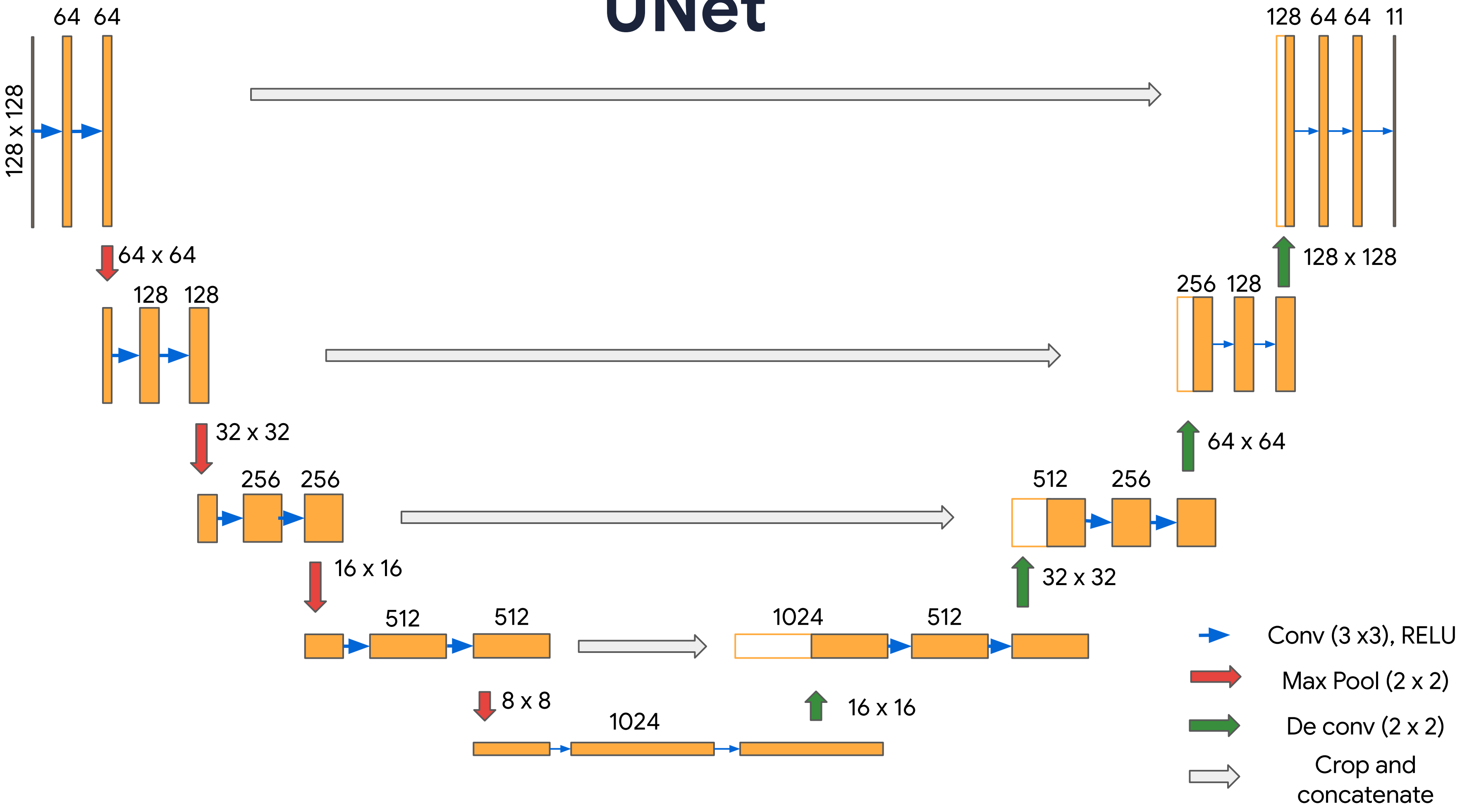
sky	0.9350185576821789
building	0.861760180856767
column/pole	9.15750915331616e-10
road	0.9559097147402678
side walk	0.8396129387346395
vegetation	0.8677883515092748
traffic light	6.040349126864078e-05
fence	0.012679586065167121
vehicle	0.45563416820437186
pedestrian	0.0002528346886971326
byciclist	0.04615362426586659
void	0.2826172572009191



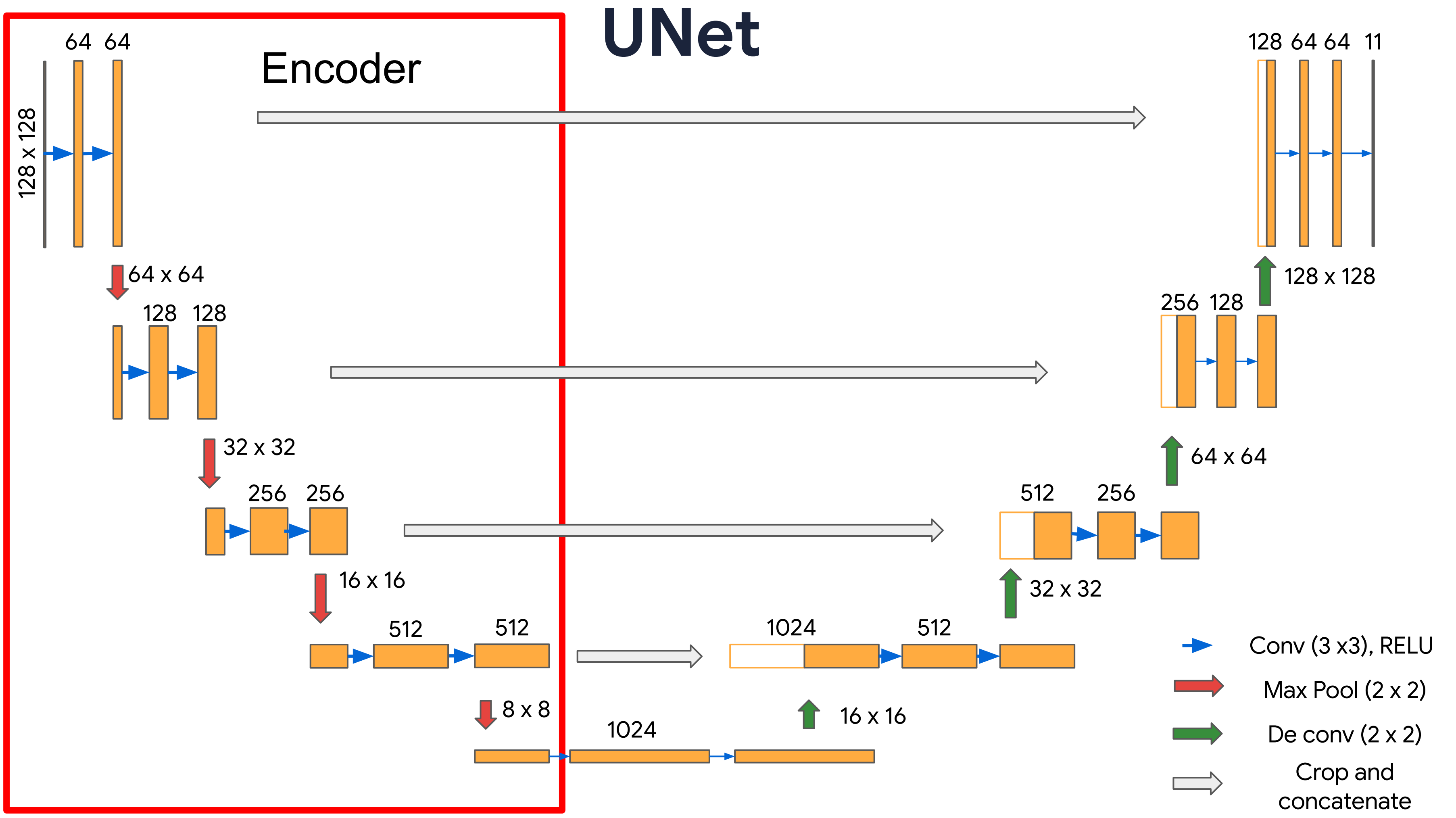
U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, Thomas Brox
https://arxiv.org/abs/1505.04597

UNet

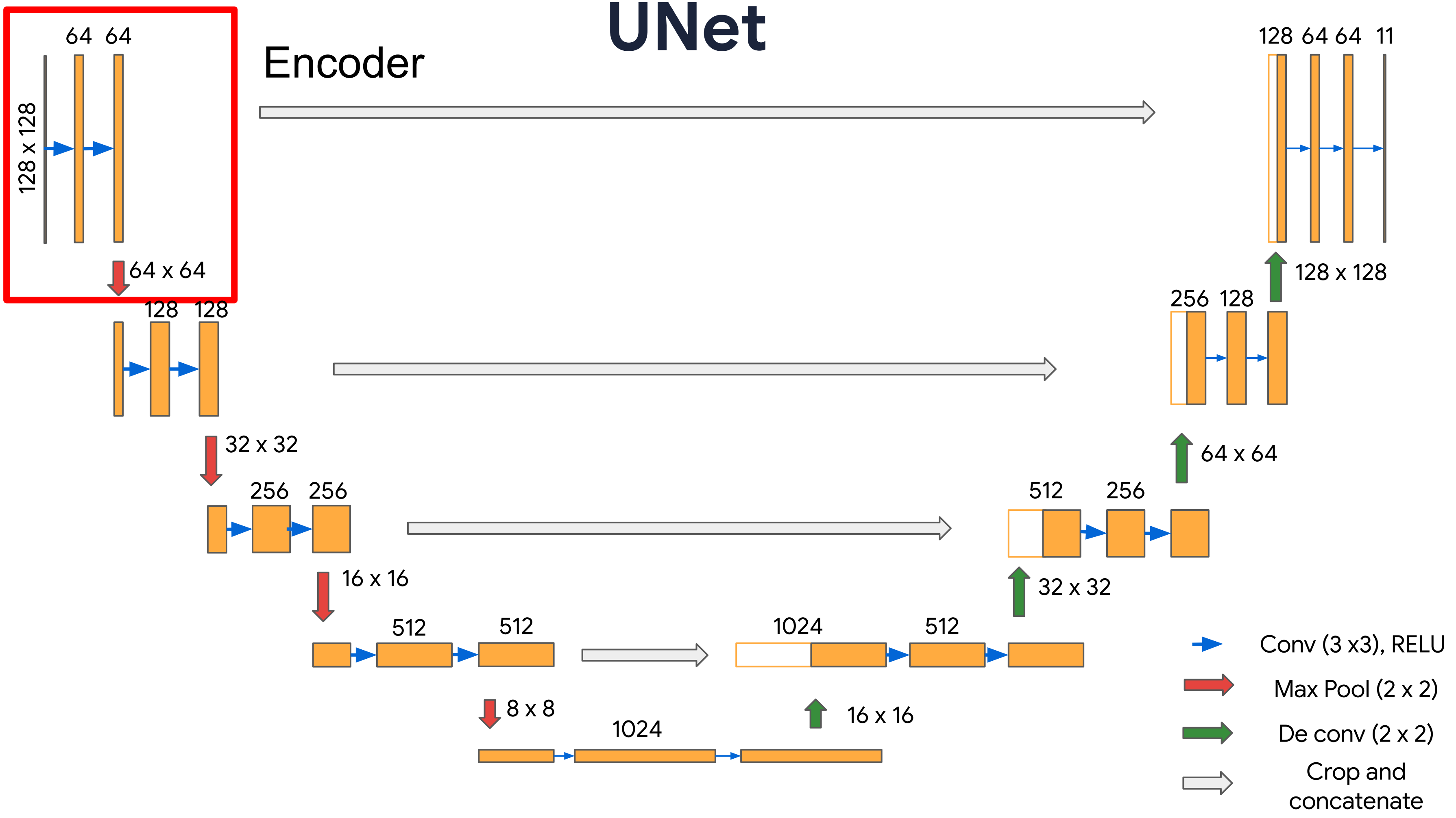


64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
1024 512
16 x 16
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

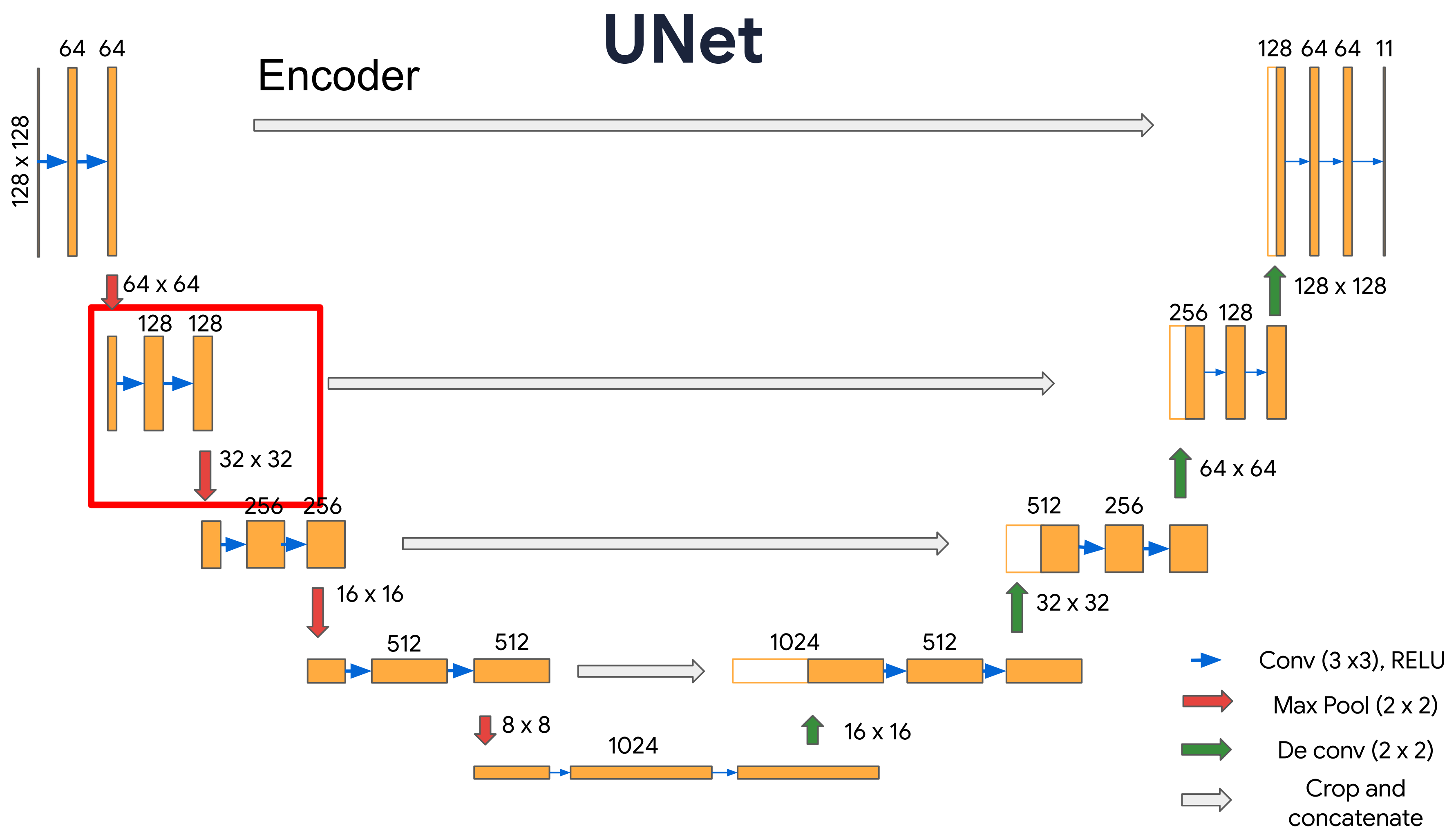


UNet
Encoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
16 x 16
1024 512
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

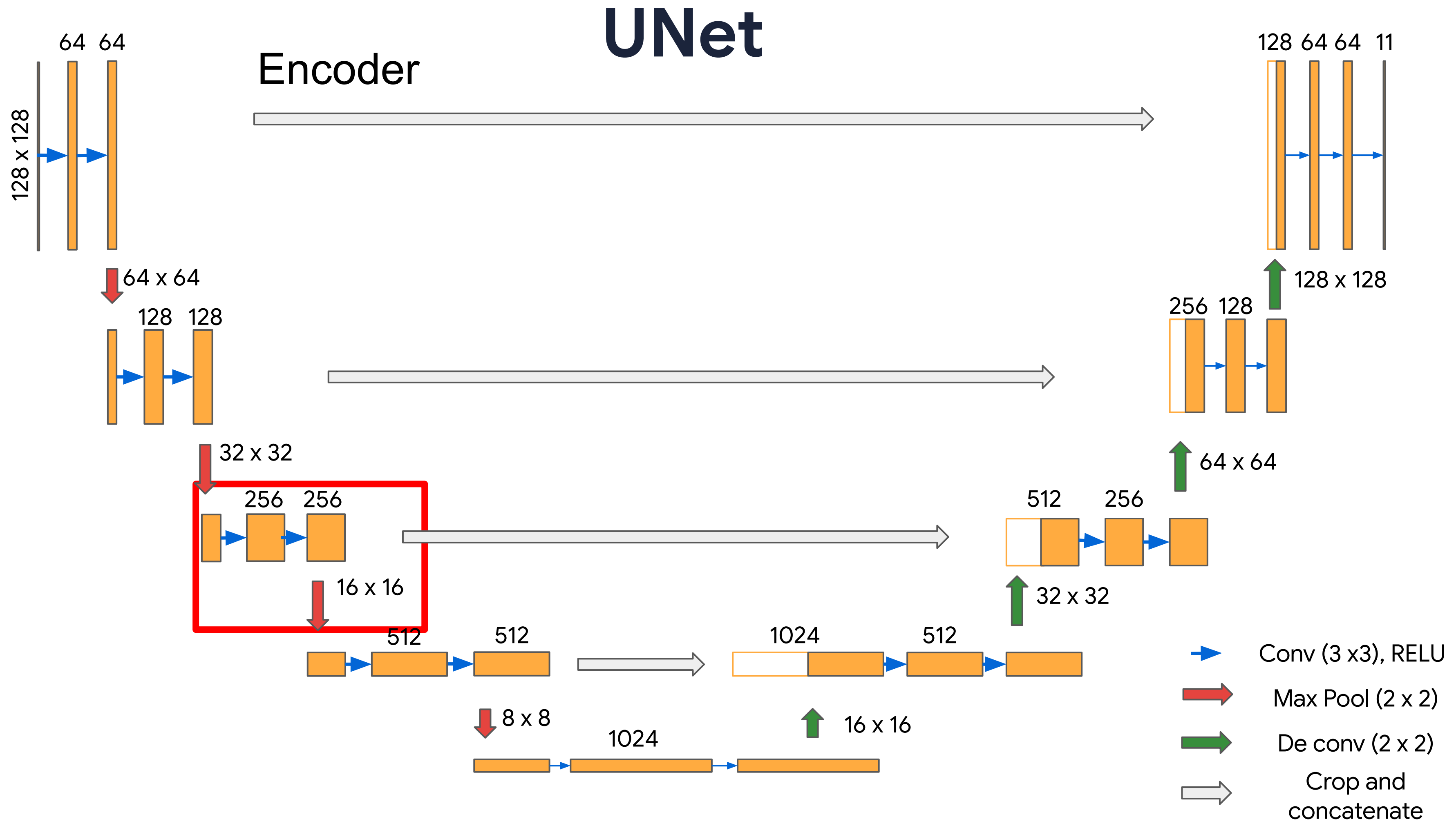
UNet



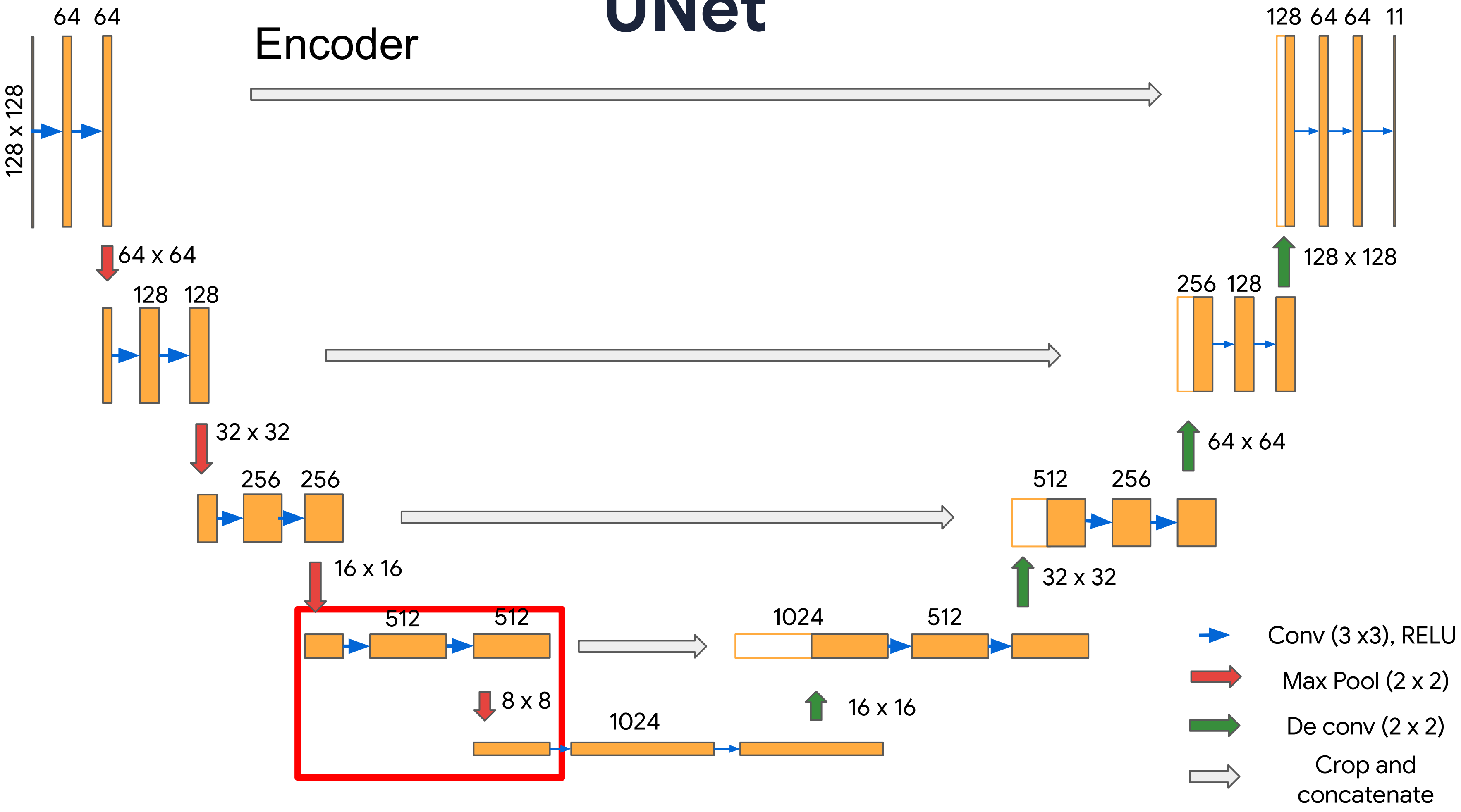
Encoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
16 x 16
1024 512
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate



UNet
Encoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
16 x 16
1024 512
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

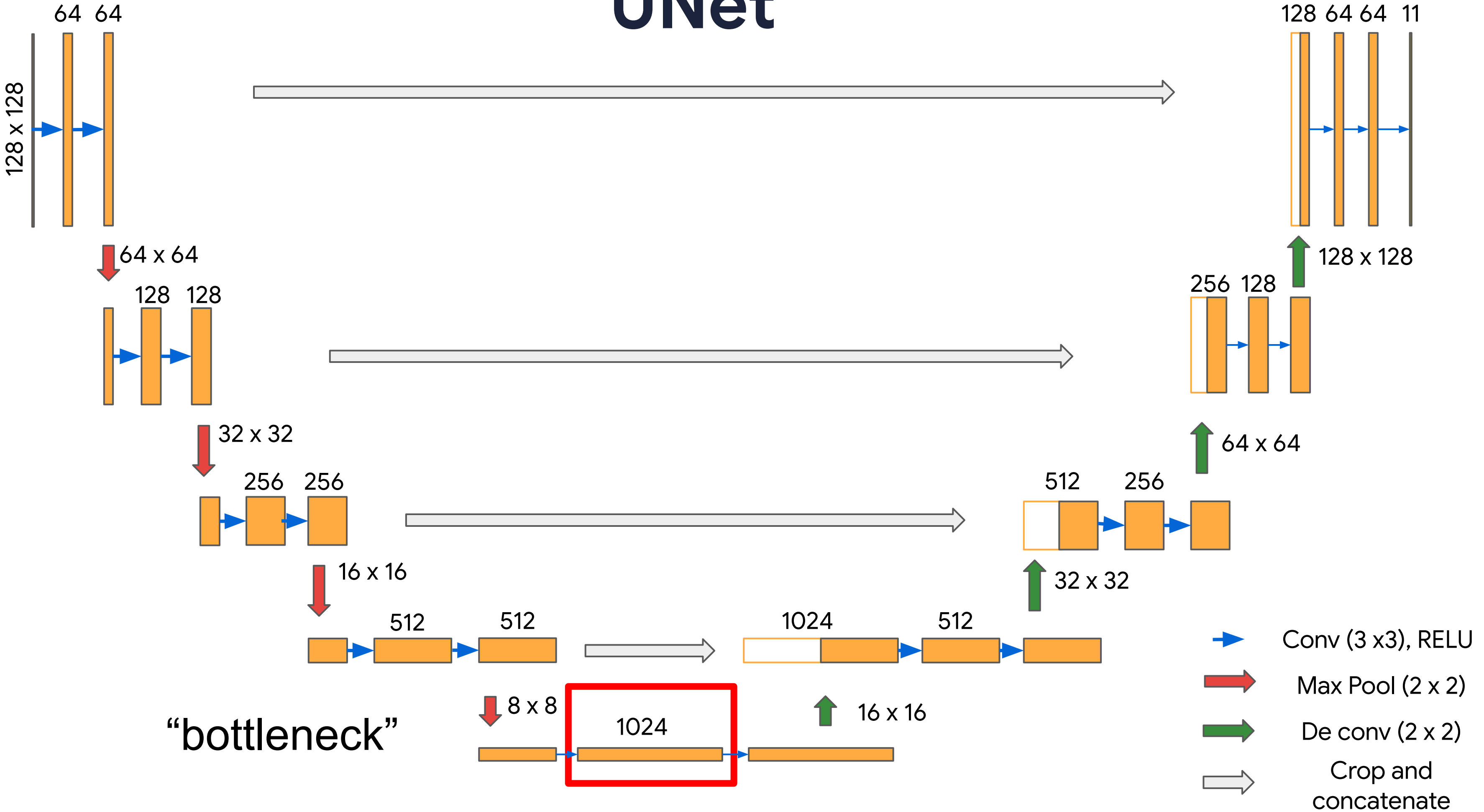

UNet
Encoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
16 x 16
1024 512
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

UNet

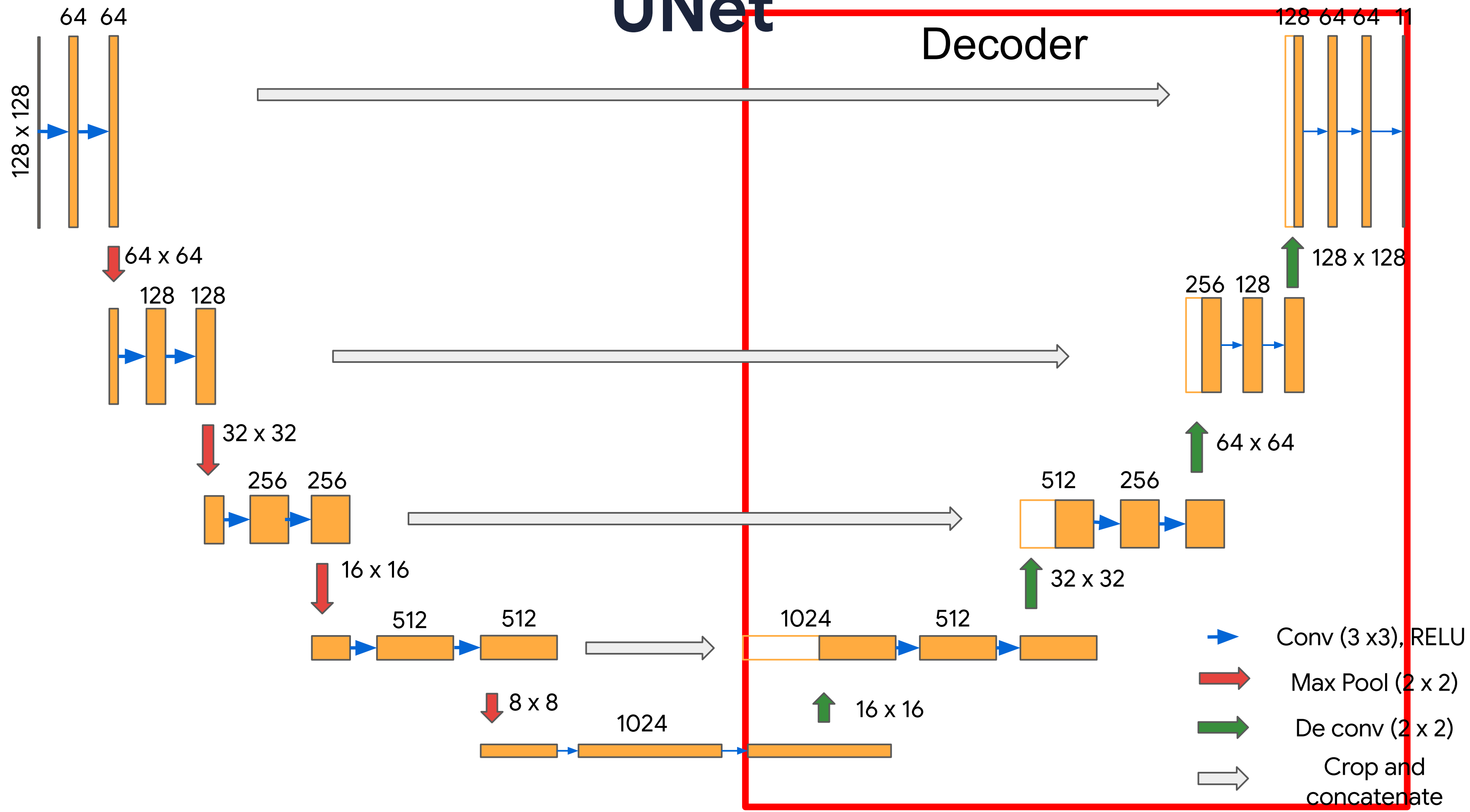


Encoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
16 x 16
1024 512
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

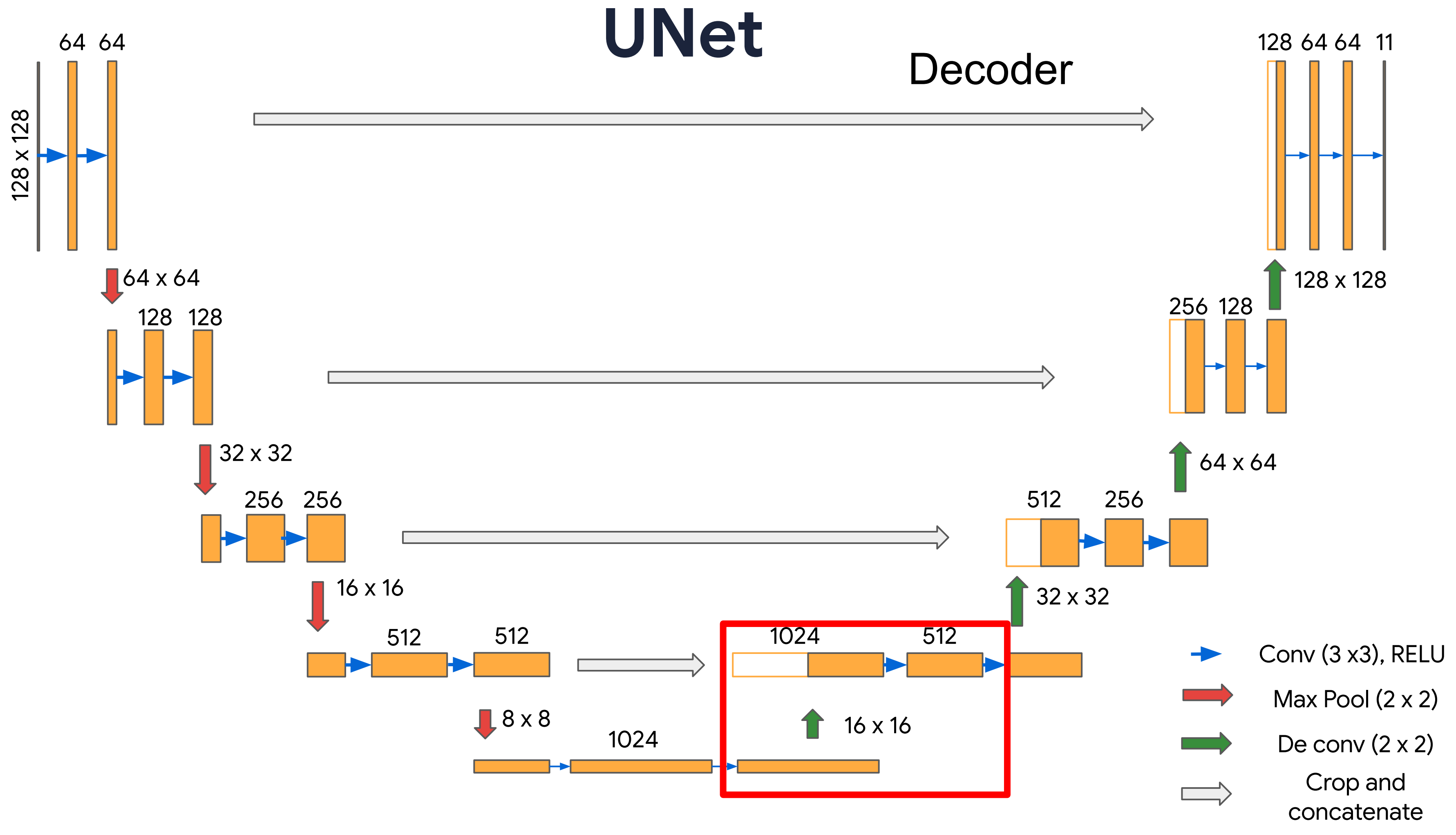
UNet


64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
"bottleneck"
1024 512
16 x 16
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

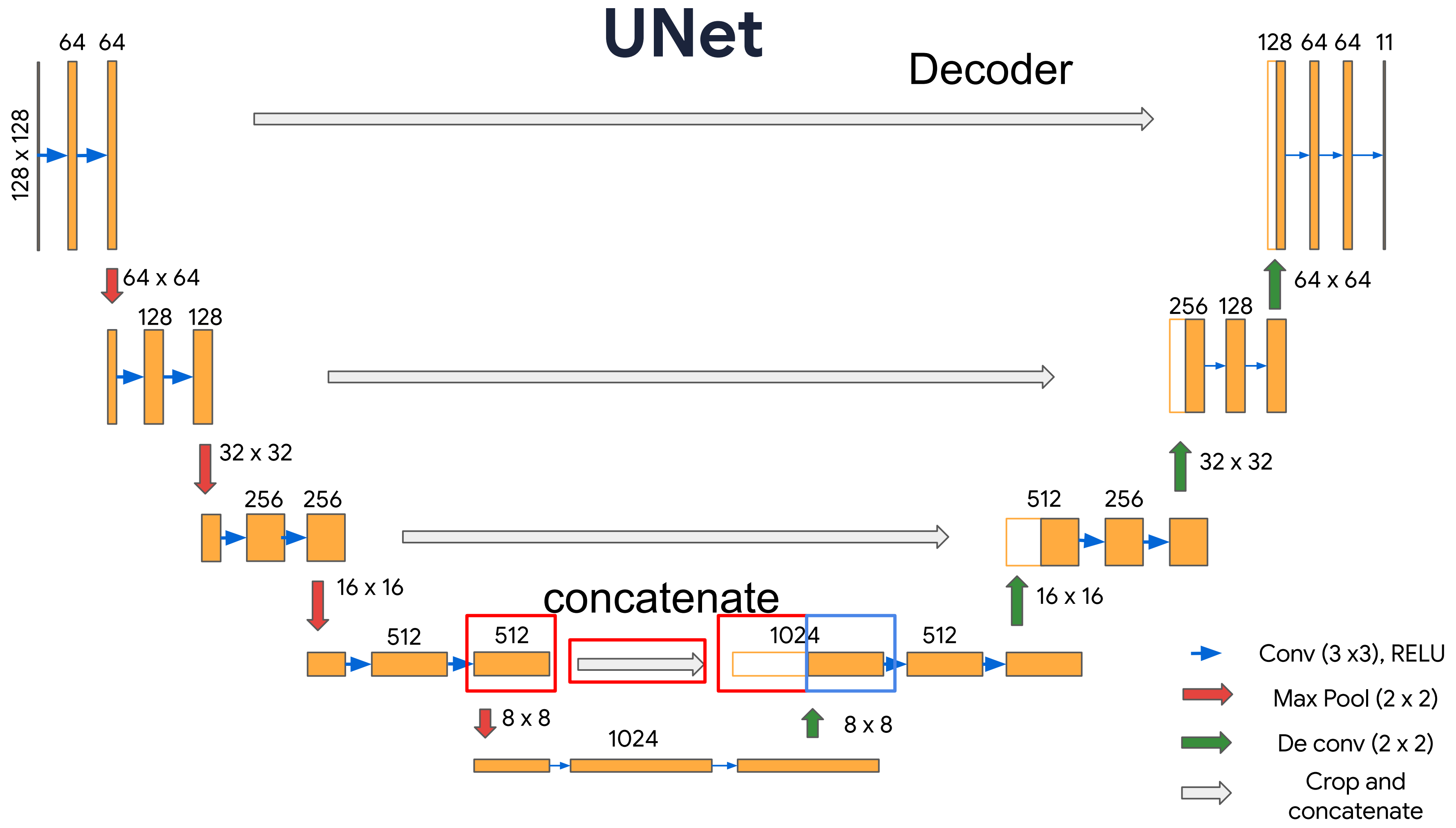
UNet



Decoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
1024 512
16 x 16
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

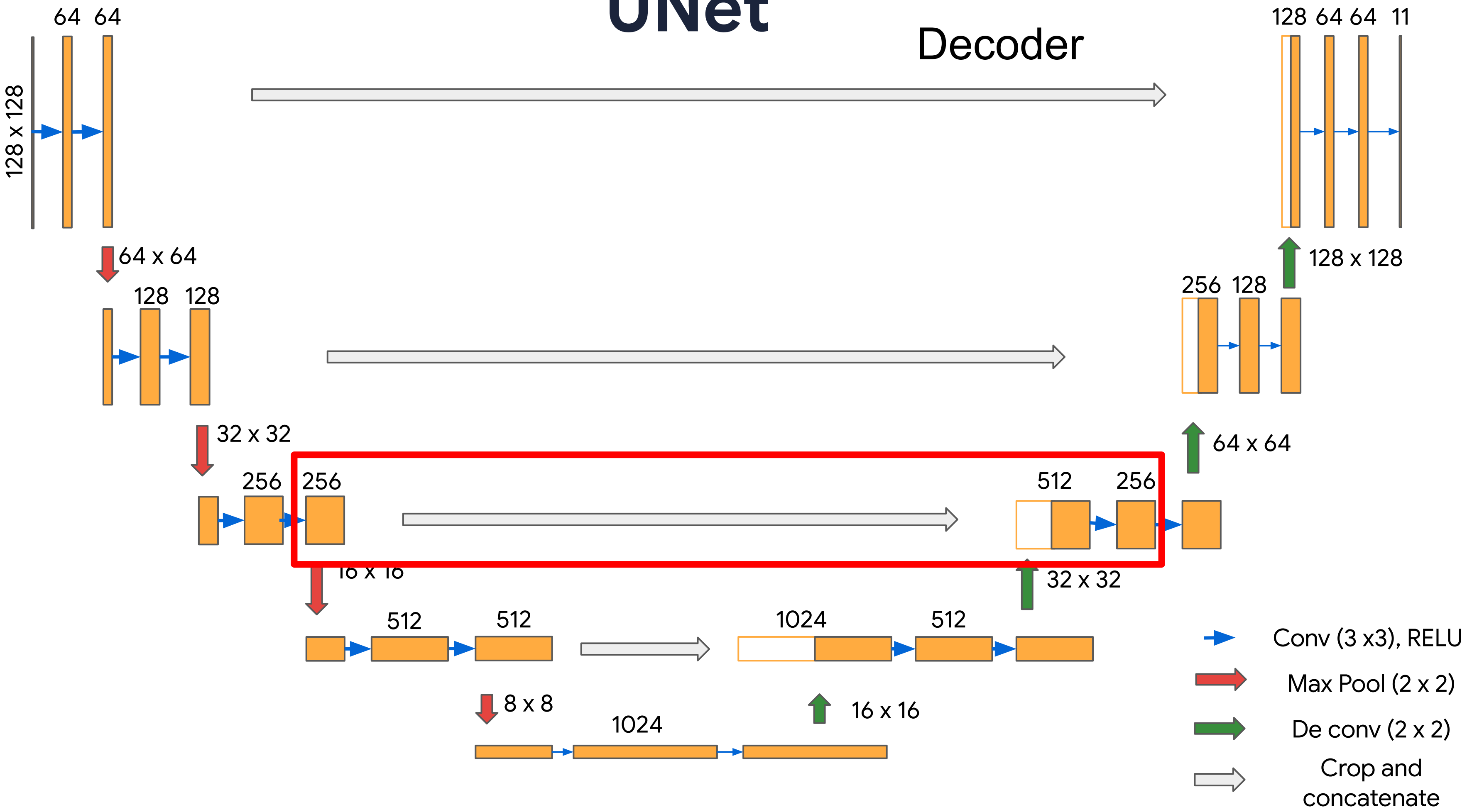


UNet
Decoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
1024 512
16 x 16
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate



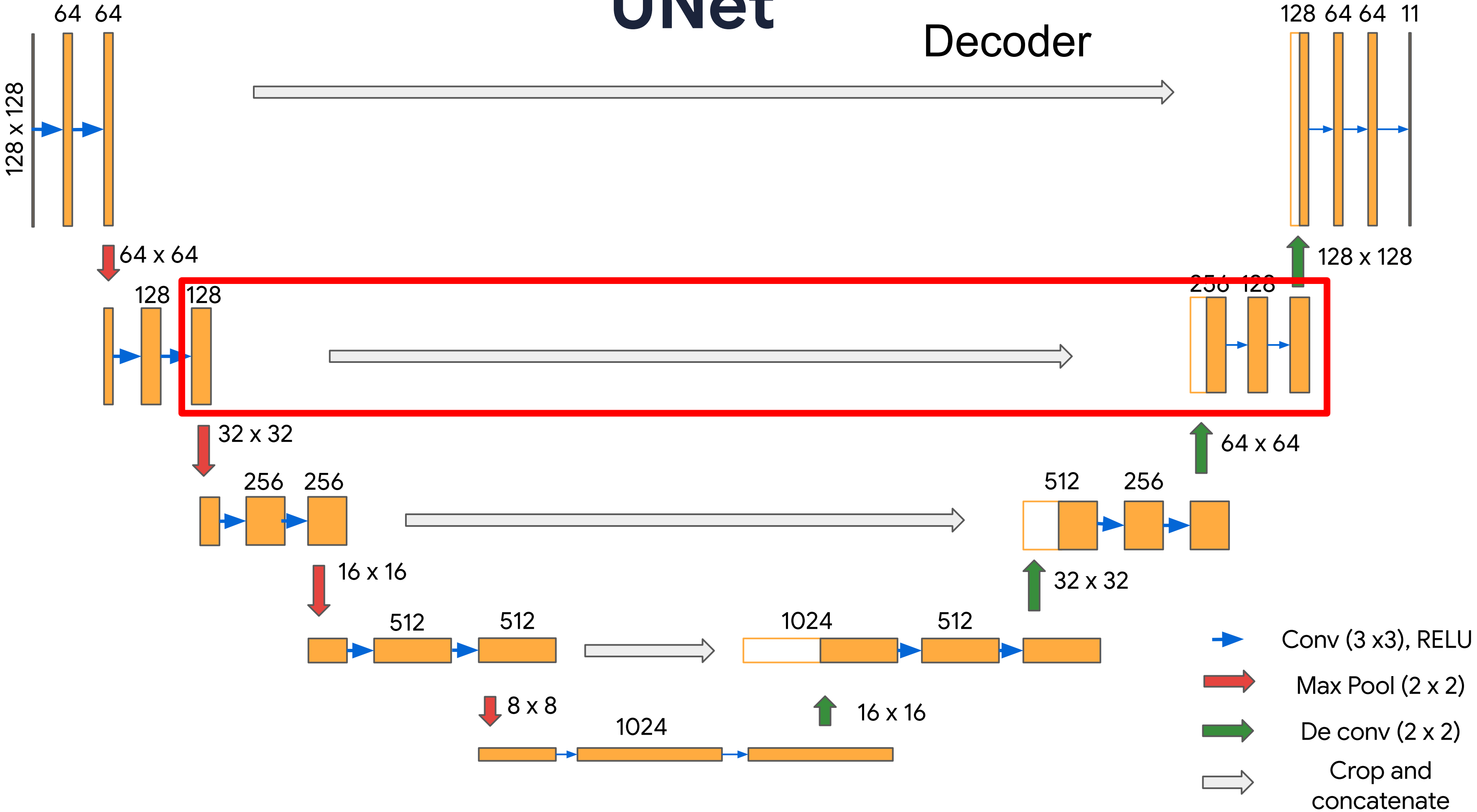
UNet
Decoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512
512
concatenate
1024
512
8 x 8
1024
8 x 8
16 x 16
512 256
32 x 32
256 128
64 x 64
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and
concatenate

UNet

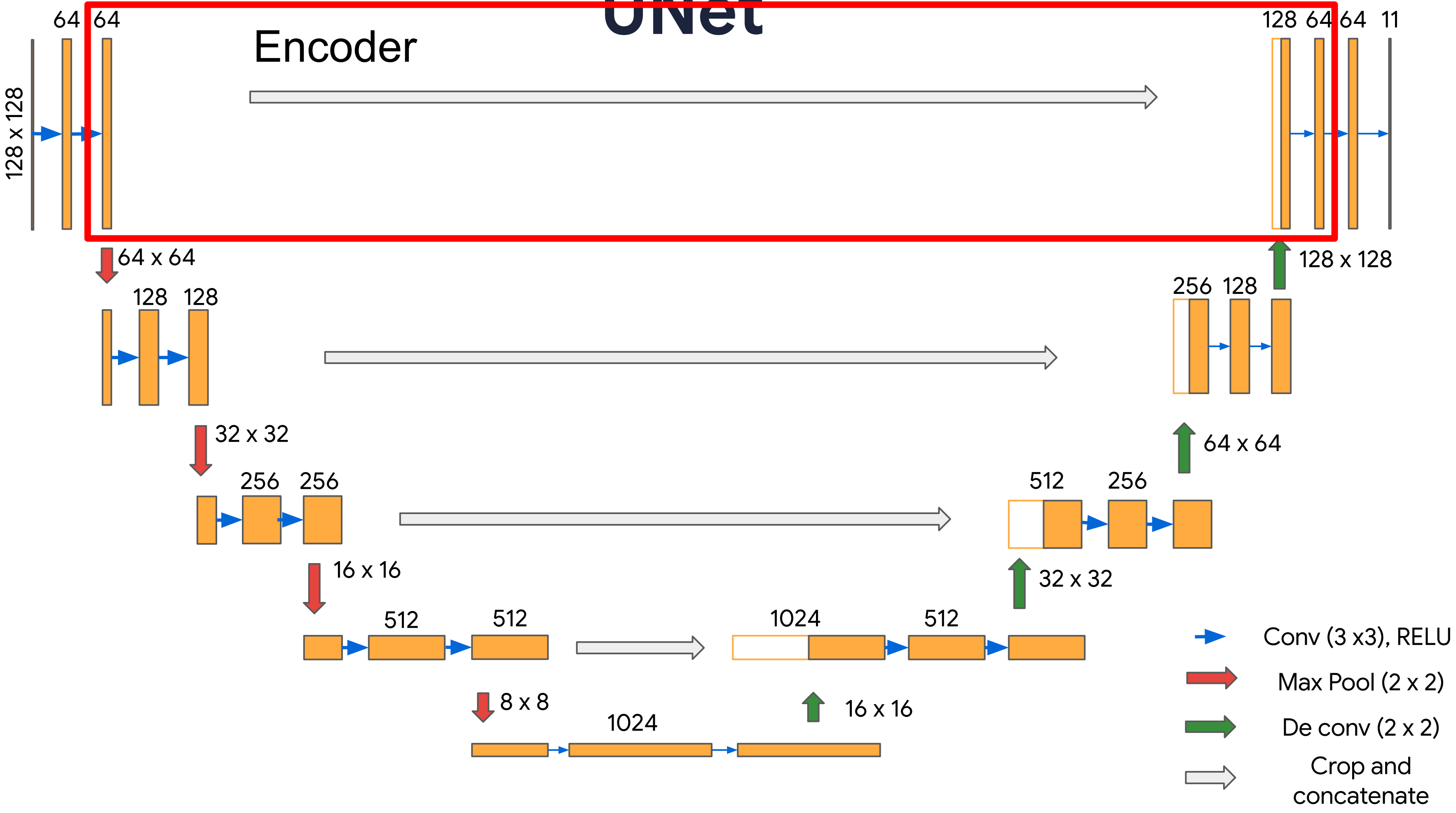


Decoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
16 x 16
1024 512
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

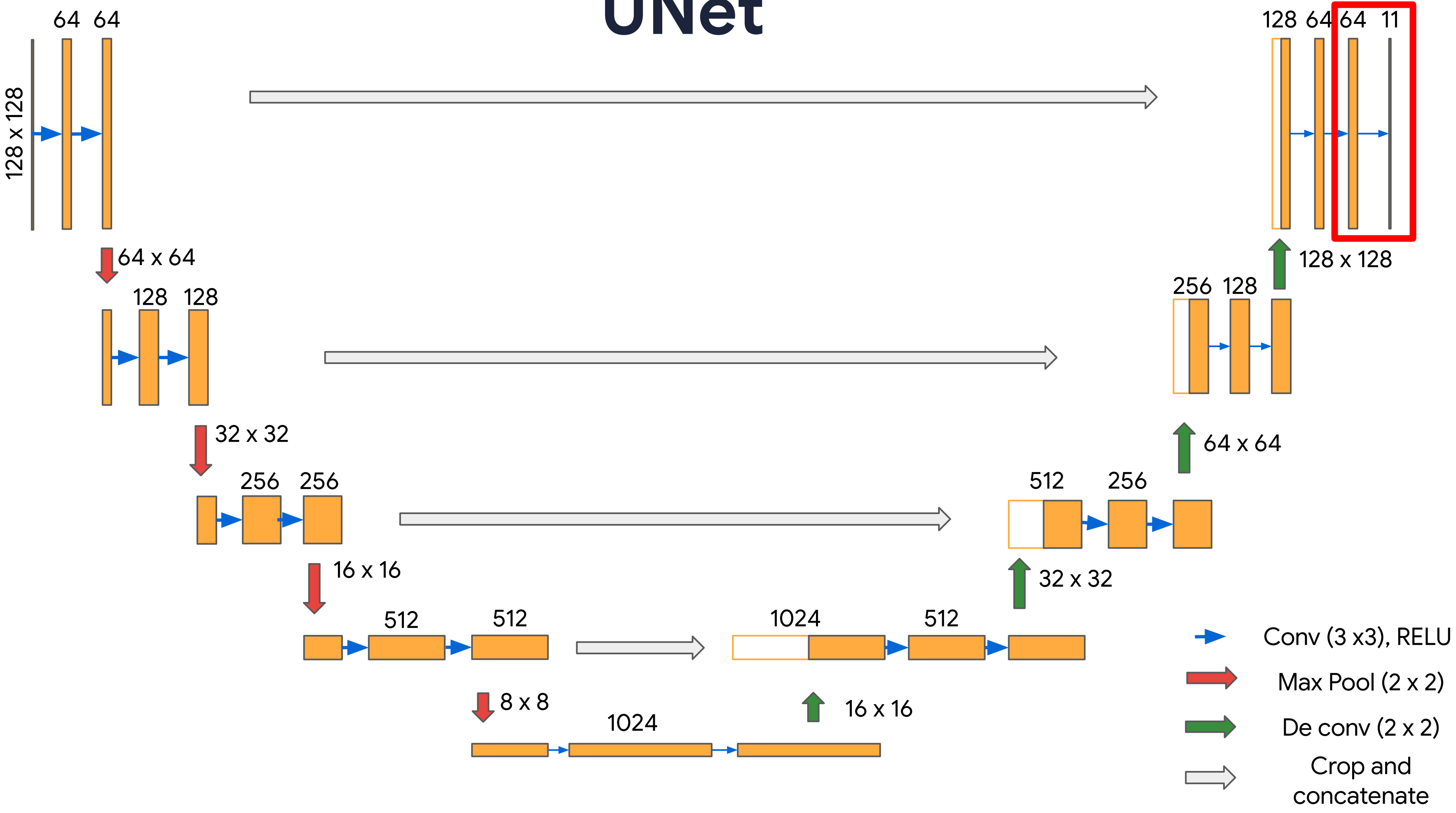
UNet


Decoder
64 64
128 x 128
64 x 64
128 128
32 x 32
256 256
16 x 16
512 512
8 x 8
1024
16 x 16
1024 512
32 x 32
512 256
64 x 64
256 128
128 x 128
128 64 64 11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

UNet


Encoder
128 x 128
64
64
64 x 64
128
128
32 x 32
256
256
16 x 16
512
512
8 x 8
1024
16 x 16
1024
512
32 x 32
512
256
64 x 64
256
128
128 x 128
128
64
64
11
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

UNet

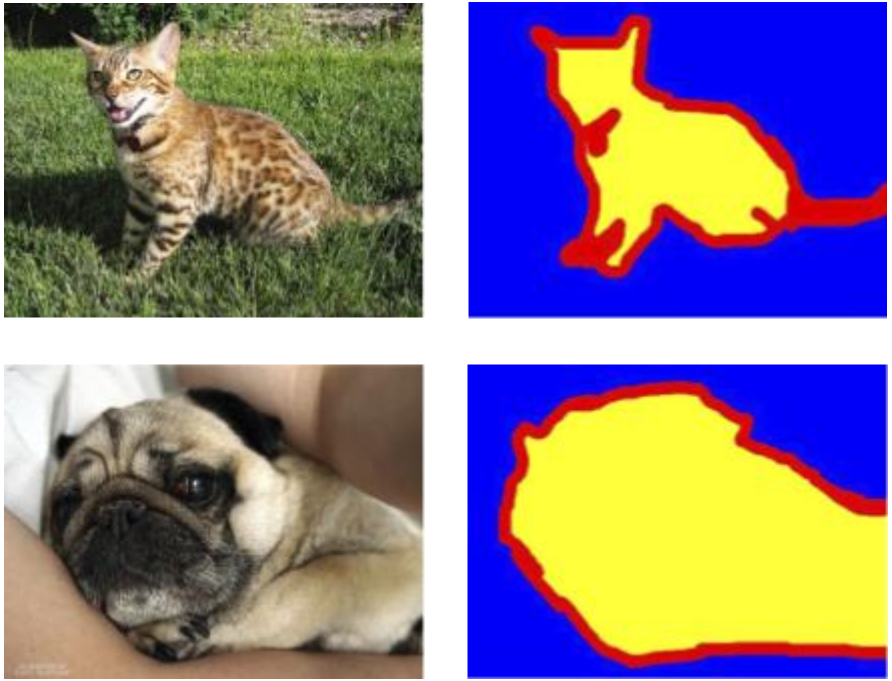


64 64
128 x 128
128 64 64 11
64 x 64
128 128
256 128
128 x 128
32 x 32
256 256
64 x 64
512 256
16 x 16
32 x 32
512 512
1024 512
8 x 8
1024
16 x 16
Conv (3 x3), RELU
Max Pool (2 x 2)
De conv (2 x 2)
Crop and concatenate

Oxford IIT Pets Dataset

- The dataset can be found at https://www.robots.ox.ac.uk/~vgg/data/pets/
- 37 category pet dataset.
- Dataset has images, associated breed, segmentation masks and head ROI.
- Segmentation Masks are labelled either {1, 2, 3}
  - 1 - Pet
  - 2 - Outline
  - 3 - Background

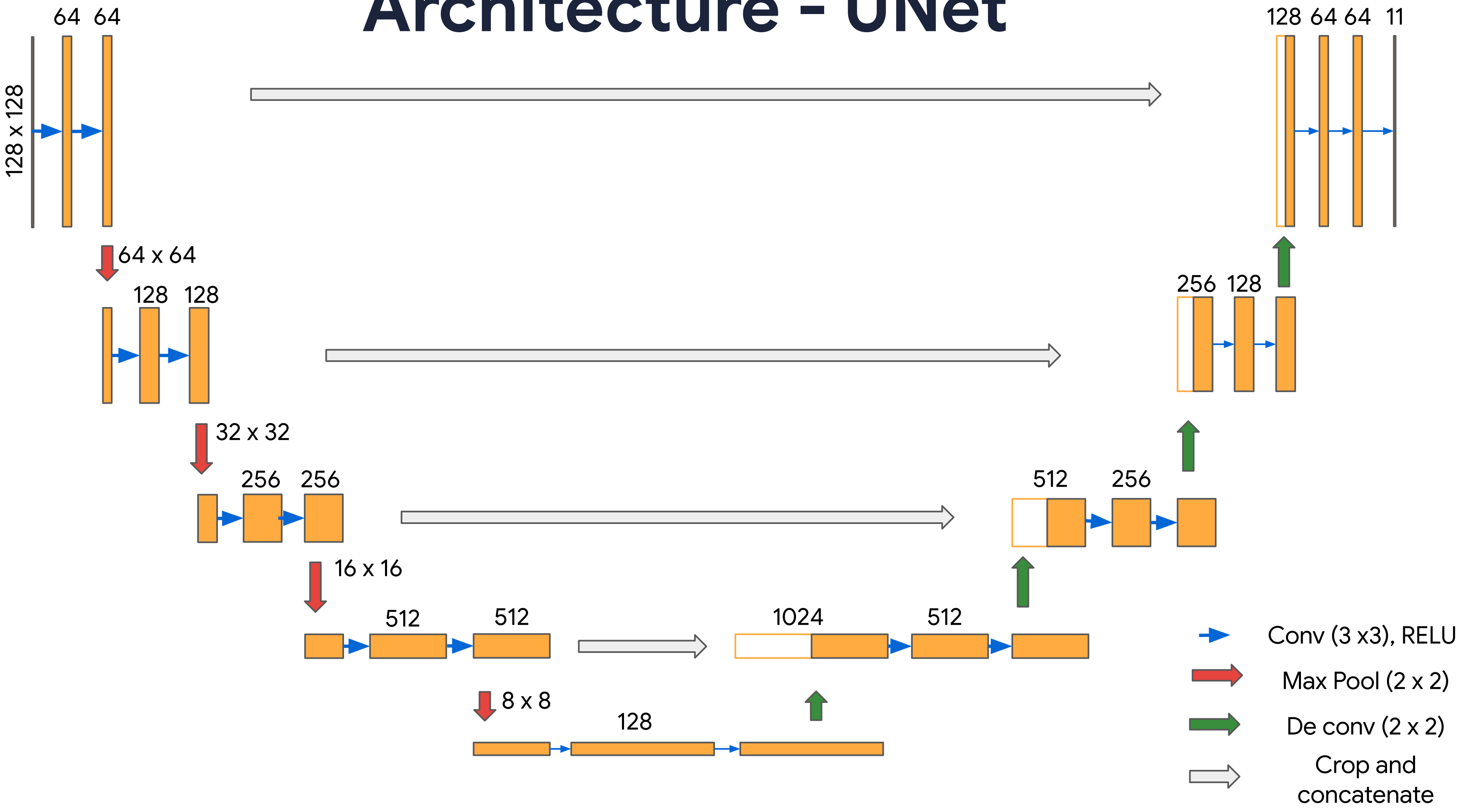
Oxford IIT Pets Dataset



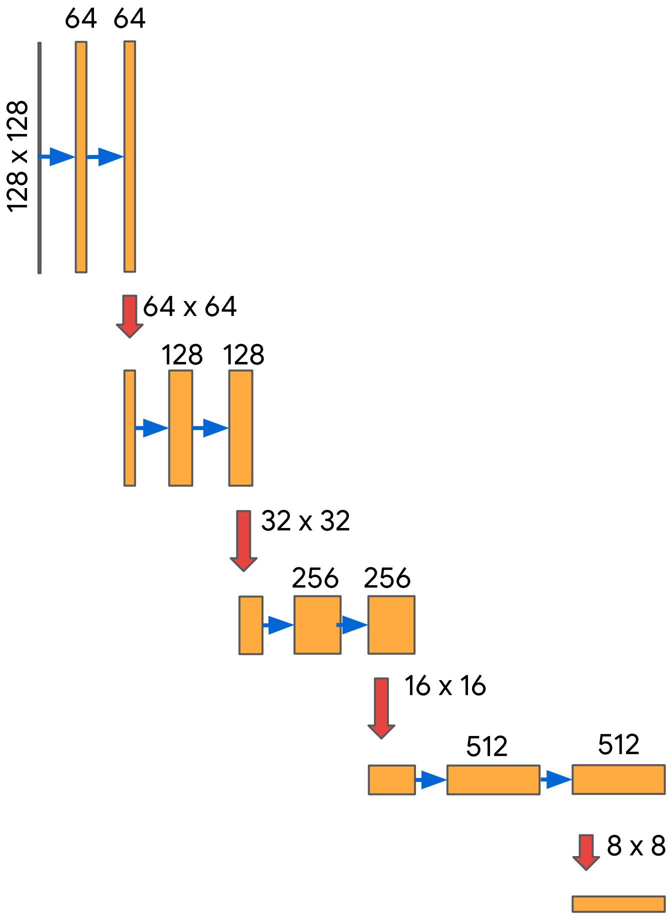
Images Segments

https://www.robots.ox.ac.uk/~vgg/data/pets/

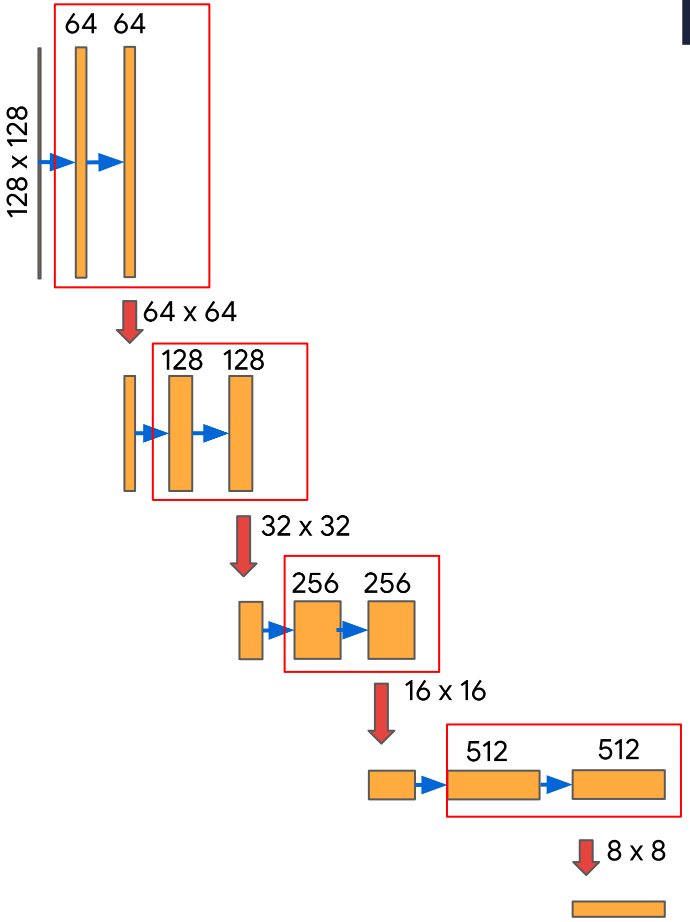
Architecture - UNet



Encoder



Encoder



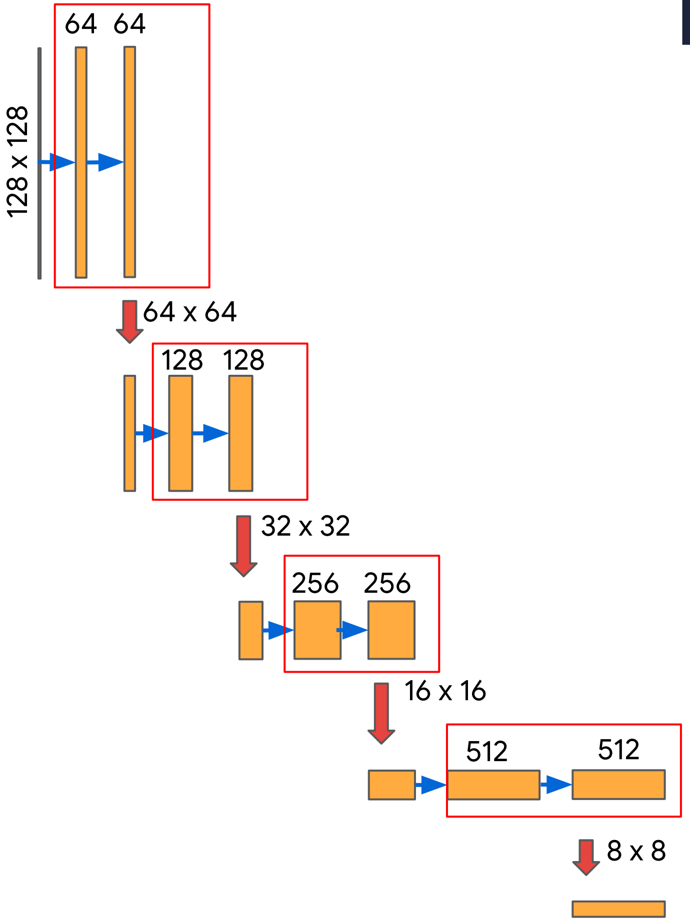
```python
def conv2d_block(input_tensor, n_filters, kernel_size = 3):
    x = input_tensor

    for i in range(2):
        x = tf.keras.layers.Conv2D(filters = n_filters,
                                   kernel_size = (kernel_size, kernel_size))(x)

        x = tf.keras.layers.Activation('relu')(x)

    return x
```

```python
def conv2d_block(input_tensor, n_filters, kernel_size = 3):
    x = input_tensor

    for i in range(2):
        x = tf.keras.layers.Conv2D(filters = n_filters,
                                   kernel_size = (kernel_size, kernel_size))(x)

        x = tf.keras.layers.Activation('relu')(x)

    return x
```

```python
def conv2d_block(input_tensor, n_filters, kernel_size = 3):
    x = input_tensor

    for i in range(2):
        x = tf.keras.layers.Conv2D(filters = n_filters,
                                   kernel_size = (kernel_size, kernel_size))(x)

        x = tf.keras.layers.Activation('relu')(x)

    return x
```

```python
def conv2d_block(input_tensor, n_filters, kernel_size = 3):
    x = input_tensor

    for i in range(2):
        x = tf.keras.layers.Conv2D(filters = n_filters,
                                   kernel_size = (kernel_size, kernel_size))(x)

        x = tf.keras.layers.Activation('relu')(x)

    return x
```

Encoder



```python
def encoder_block(inputs, n_filters, pool_size, dropout):
  f = conv2d_block(inputs, n_filters=n_filters)
  p = tf.keras.layers.MaxPooling2D(pool_size)(f)
  p = tf.keras.layers.Dropout(dropout)(p)

  return f, p
```

```python
def encoder_block(inputs, n_filters, pool_size, dropout):
  f = conv2d_block(inputs, n_filters=n_filters)
  p = tf.keras.layers.MaxPooling2D(pool_size)(f)
  p = tf.keras.layers.Dropout(dropout)(p)

  return f, p
```

```
def encoder_block(inputs, n_filters, pool_size, dropout):
  f = conv2d_block(inputs, n_filters=n_filters)
  p = tf.keras.layers.MaxPooling2D(pool_size)(f)
  p = tf.keras.layers.Dropout(dropout)(p)

  return f, p
```

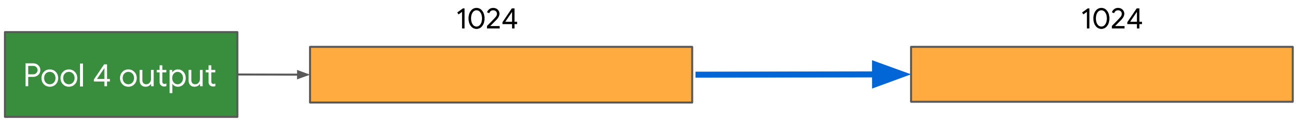
```python
def encoder_block(inputs, n_filters, pool_size, dropout):
  f = conv2d_block(inputs, n_filters=n_filters)
  p = tf.keras.layers.MaxPooling2D(pool_size)(f)
  p = tf.keras.layers.Dropout(dropout)(p)

  return f, p
```

```python
def encoder_block(inputs, n_filters, pool_size, dropout):
  f = conv2d_block(inputs, n_filters=n_filters)
  p = tf.keras.layers.MaxPooling2D(pool_size)(f)
  p = tf.keras.layers.Dropout(dropout)(p)

  return f, p
```

```python
def encoder_block(inputs, n_filters, pool_size, dropout):
  f = conv2d_block(inputs, n_filters=n_filters)
  p = tf.keras.layers.MaxPooling2D(pool_size)(f)
  p = tf.keras.layers.Dropout(dropout)(p)

  return f, p
```

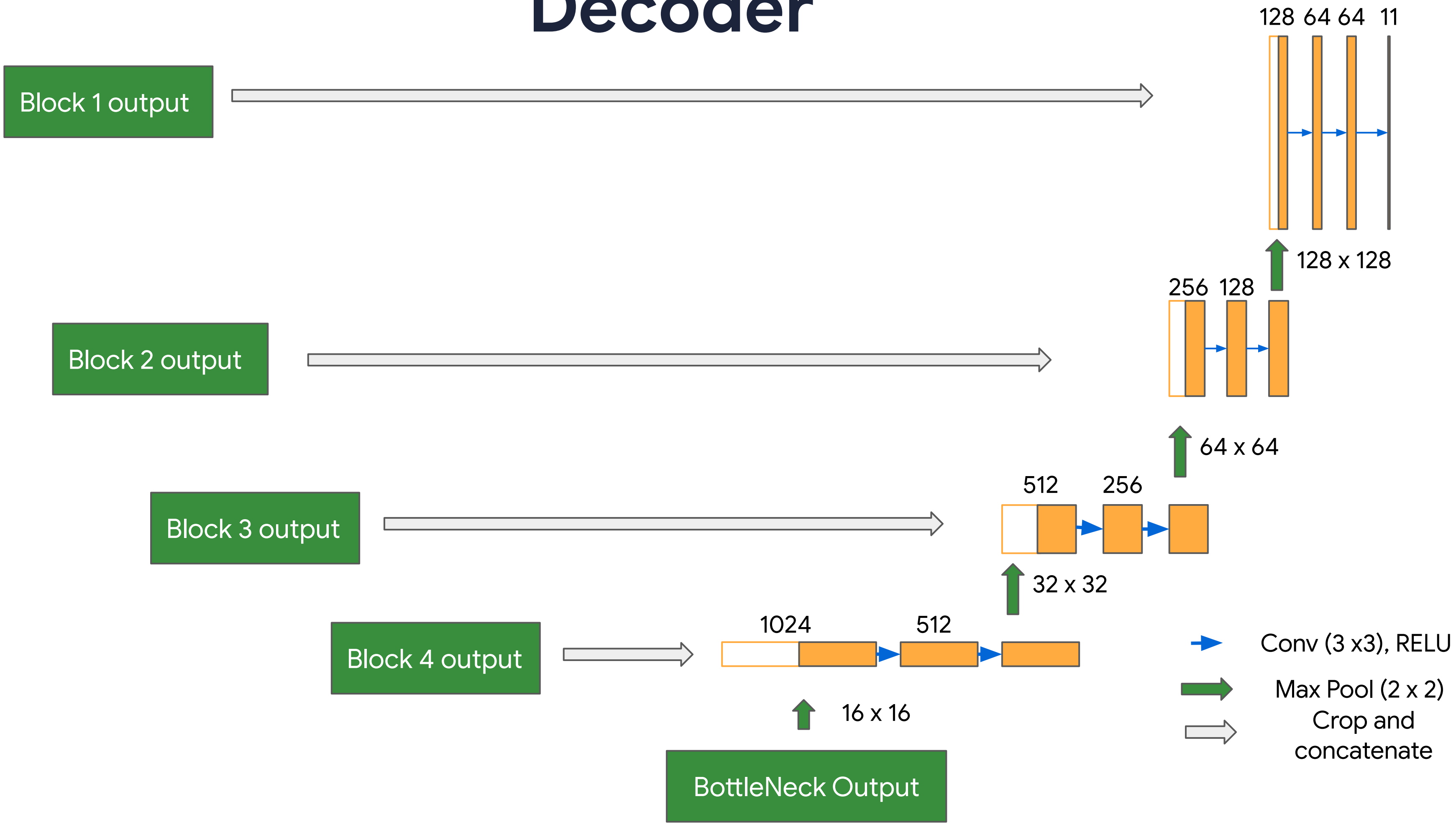
```python
def encoder(inputs):

  f1, p1 = encoder_block(inputs, n_filters=64, pool_size=(2,2), dropout=0.3)
  f2, p2 = encoder_block(p1, n_filters=128, pool_size=(2,2), dropout=0.3)
  f3, p3 = encoder_block(p2, n_filters=256, pool_size=(2,2), dropout=0.3)
  f4, p4 = encoder_block(p3, n_filters=512, pool_size=(2,2), dropout=0.3)

  return p4, (f1, f2, f3, f4)
```

Bottleneck



```python
def bottleneck(inputs):
  bottle_neck = conv2d_block(inputs, n_filters=1024)

  return bottle_neck
```

Decoder



128 64 64 11
Block 1 output
128 x 128
256 128
Block 2 output
64 x 64
512 256
Block 3 output
32 x 32
1024 512
Block 4 output
16 x 16
BottleNeck Output
Conv (3 x3), RELU
Max Pool (2 x 2)
Crop and concatenate

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                      padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                       padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                       padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                       padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                      padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                      padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                      padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
  u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides = strides,
                                       padding = 'same')(inputs)
  c = tf.keras.layers.concatenate([u, conv_output])
  c = tf.keras.layers.Dropout(dropout)(c)
  c = conv2d_block(c, n_filters, kernel_size=3)

  return c
```

```python
def decoder(inputs, convs):
  f1, f2, f3, f4 = convs

  c6 = decoder_block(inputs, f4, n_filters=512, kernel_size=(3,3), strides=(2,2), dropout=0.3)
  c7 = decoder_block(c6, f3, n_filters=256, kernel_size=(3,3), strides=(2,2), dropout=0.3)
  c8 = decoder_block(c7, f2, n_filters=128, kernel_size=(3,3), strides=(2,2), dropout=0.3)
  c9 = decoder_block(c8, f1, n_filters=64, kernel_size=(3,3), strides=(2,2), dropout=0.3)

  outputs = tf.keras.layers.Conv2D(3, (1, 1), activation='softmax')(c9)

  return outputs
```

```python
def unet():
  inputs = tf.keras.layers.Input(shape=(128,128,3))
  encoder_output, convs = encoder(inputs)

  bottle_neck = bottleneck(encoder_output)

  outputs = decoder(bottle_neck, convs)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

```python
def unet():
  inputs = tf.keras.layers.Input(shape=(128,128,3))
  encoder_output, convs = encoder(inputs)

  bottle_neck = bottleneck(encoder_output)

  outputs = decoder(bottle_neck, convs)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

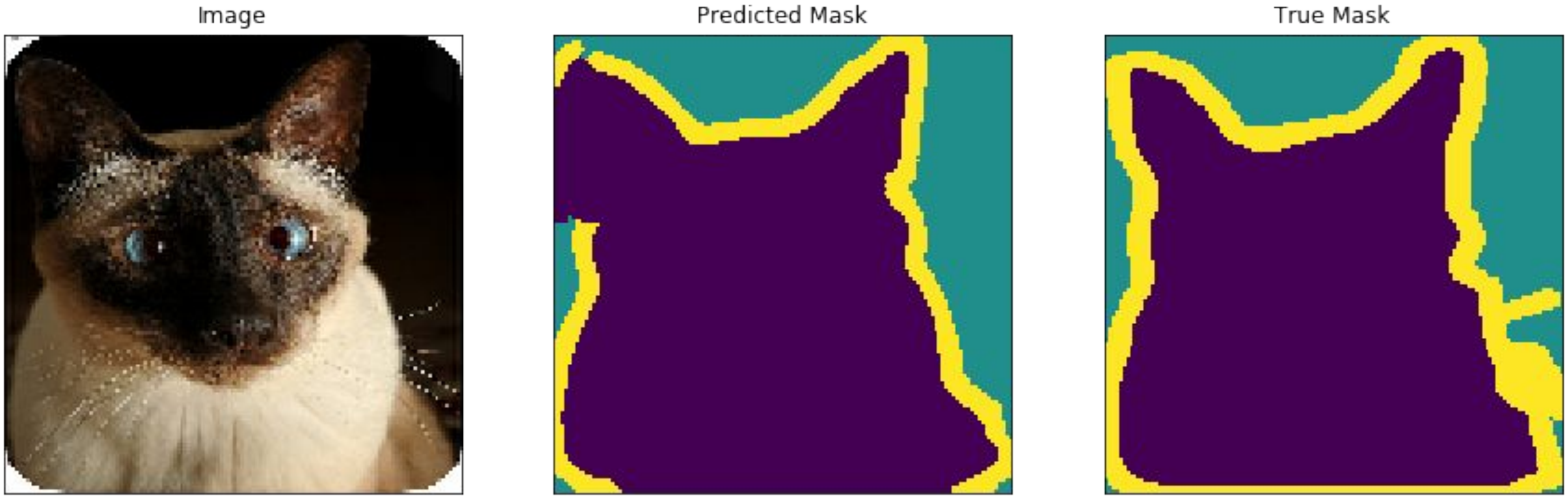
```python
def unet():
  inputs = tf.keras.layers.Input(shape=(128,128,3))
  encoder_output, convs = encoder(inputs)

  bottle_neck = bottleneck(encoder_output)

  outputs = decoder(bottle_neck, convs)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

```python
def unet():
  inputs = tf.keras.layers.Input(shape=(128,128,3))
  encoder_output, convs = encoder(inputs)

  bottle_neck = bottleneck(encoder_output)

  outputs = decoder(bottle_neck, convs)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

```python
def unet():
  inputs = tf.keras.layers.Input(shape=(128,128,3))
  encoder_output, convs = encoder(inputs)

  bottle_neck = bottleneck(encoder_output)

  outputs = decoder(bottle_neck, convs)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

```python
def unet():
  inputs = tf.keras.layers.Input(shape=(128,128,3))
  encoder_output, convs = encoder(inputs)

  bottle_neck = bottleneck(encoder_output)

  outputs = decoder(bottle_neck, convs)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

Sample Visualization of Predicted Segments



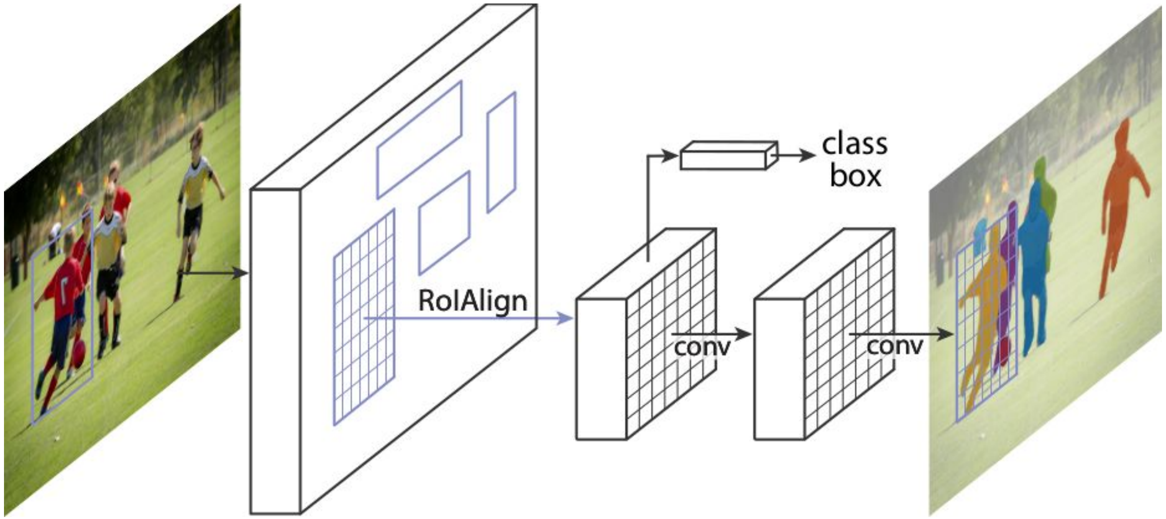
pet: IOU: 0.9084117322372148 Dice Score: 0.9520081199787308

background: IOU: 0.8237179491525691 Dice Score: 0.9033391930467228

outline: IOU: 0.5384615399787589 Dice Score: 0.7000000027777779


IJINKAN
旧パナマ領事館
公開異人館

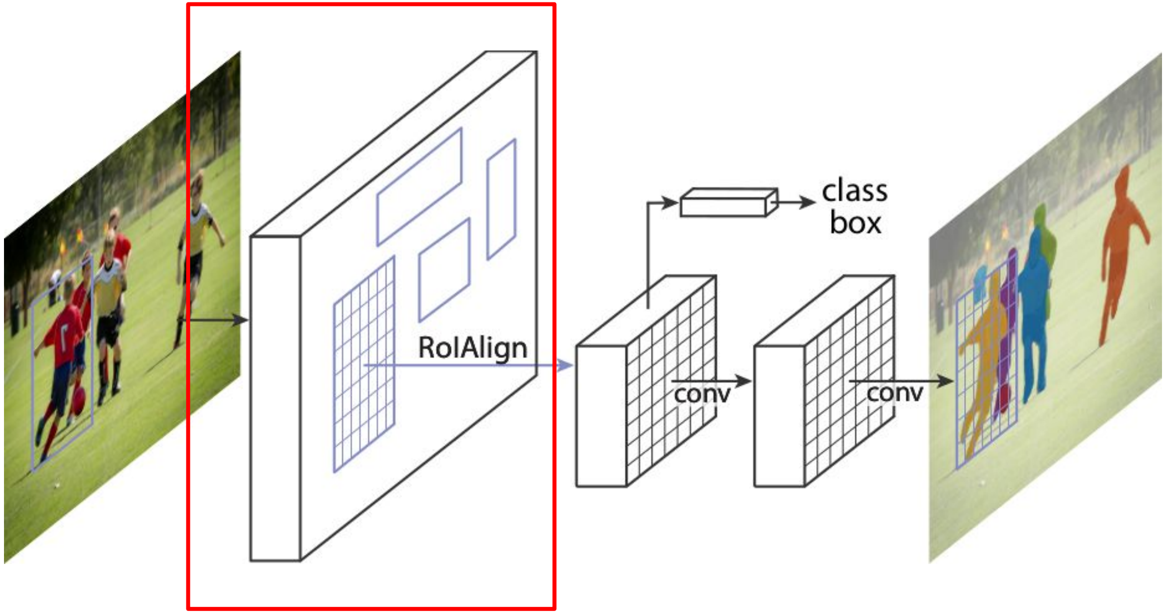
Mask R-CNN



Mask R-CNN (Facebook AI Research)
By: Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick

https://arxiv.org/abs/1703.06870

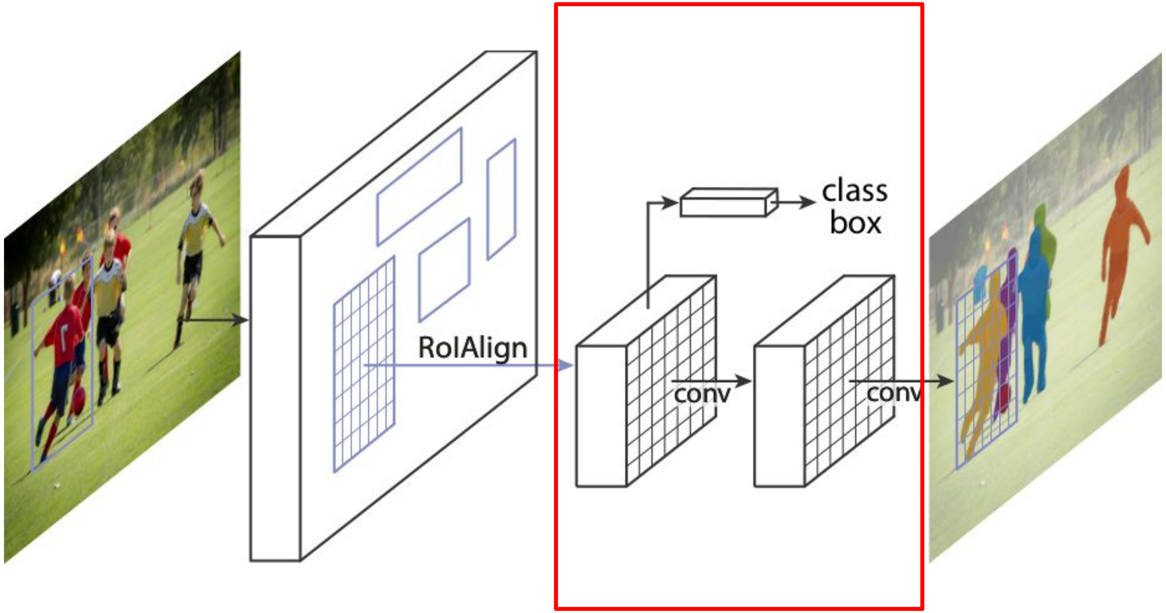
Mask R-CNN



Mask R-CNN (Facebook AI Research)

https://arxiv.org/abs/1703.06870

Mask R-CNN



Mask R-CNN (Facebook AI Research) https://arxiv.org/abs/1703.06870

Running Mask R-CNN on TPUs


```
[ ]
max_boxes_to_draw = 50    #@param {type:"
min_score_thresh = 0.1    #@param {type:

image_with_detections = visualization_ut
    np_image,
    detection_boxes,
    detection_classes,
    detection_scores,
    category_index,
    instance_masks=segmentations,
    use_normalized_coordinates=False,
    max_boxes_to_draw=max_boxes_to_draw,
    min_score_thresh=min_score_thresh)
output_image_path = 'test_results.jpg'
Image.fromarray(image_with_detections.as
display.display(display.Image(output_ima
```

max_boxes_to_draw: 50

min_score_thresh: 0.1

https://colab.research.google.com/github/tensorflow/tpu/blob/master/models/official/mask_rcnn/mask_rcnn_demo.ipynb

```python
def dict_to_tf_example(data, label_map_dict, image_dir):
    # Convert segmentation masks to run length encoded masks
    run_len_encoding = rle(data['segmentation'], image_height, image_width)
    ...
    return example
```

```python
def dict_to_tf_example(data, label_map_dict, image_dir):
    # Convert segmentation masks to run length encoded masks
    run_len_encoding = rle(data['segmentation'], image_height, image_width)
    ...
    return example
```