Abdul Qayyum

Lecturer at University of Burgundy, France

- Postdoc in Electrical and Informatics Engineering
- PhD in Electrical & Electronics Engineering
- Masters in Electronics Engineering
- Bachelor in Computer Engineering

Collaborations & Expertise:

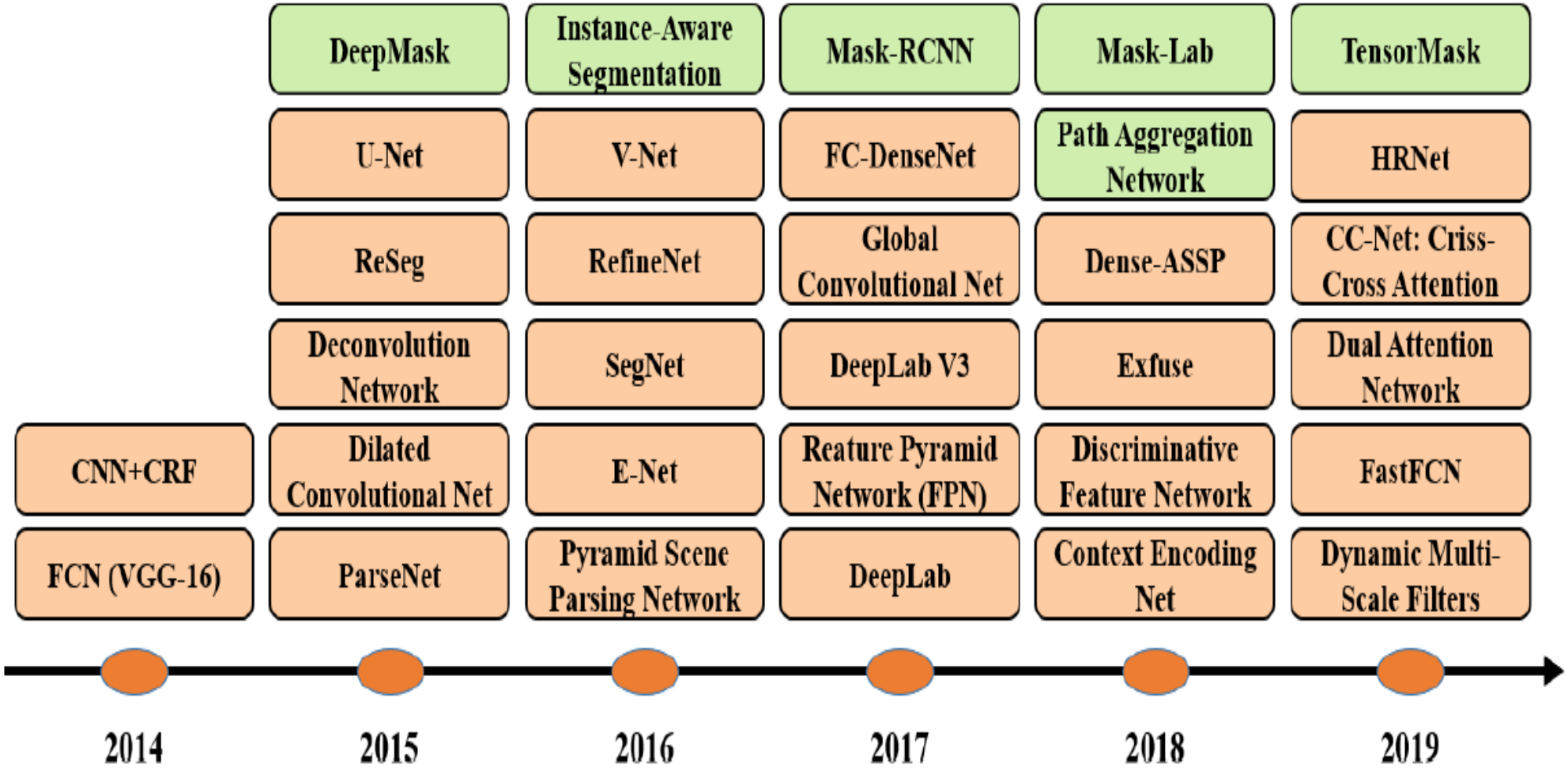
Topic: **Deep Learning Segmentation Models**

Instructor: Abdul Qayyum, PhD

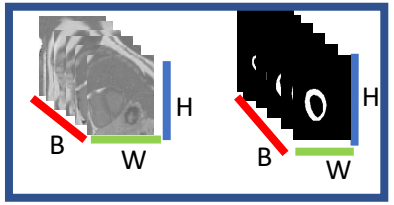
Class: MSCV

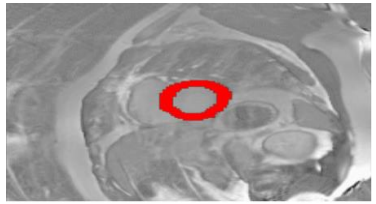
University of Burgundy, France

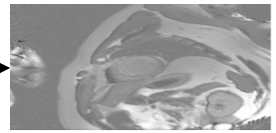
Segmentation DL Models

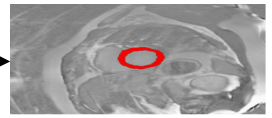


Introduction & Overview









LGE MRI dataset

Data-Preprocessing and conversion

Convert Dataset into Deep Learning Format

Training dataset (images and Masks)

Testing dataset (images and Masks)

Deep Learning Models

Save and Load Models

Test images

GT or test Masks

Introduction & Overview

What is Semantic Segmentation?

Image classification

Image Classification: A core task in Computer Vision

This image by Nikita is licensed under CC-BY 2.0

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

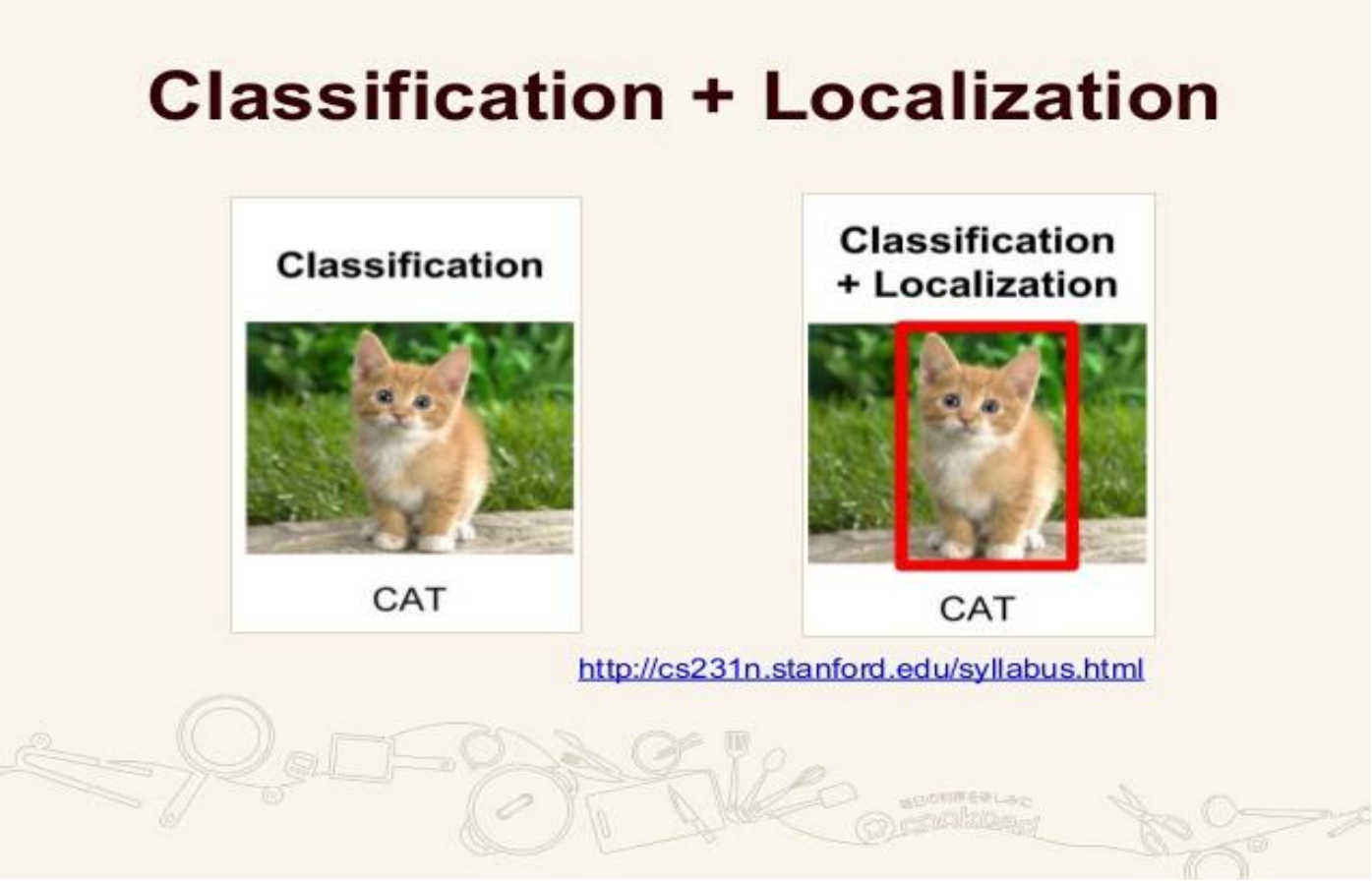
→ cat

The most fundamental building block in Computer Vision is the Image classification problem where given an image, we expect the computer to output a discrete label, which is the main object in the image. In image classification we assume that there is only one (and not multiple) object in the image.

Introduction & Overview

What is Semantic Segmentation?

b. Classification with Localization

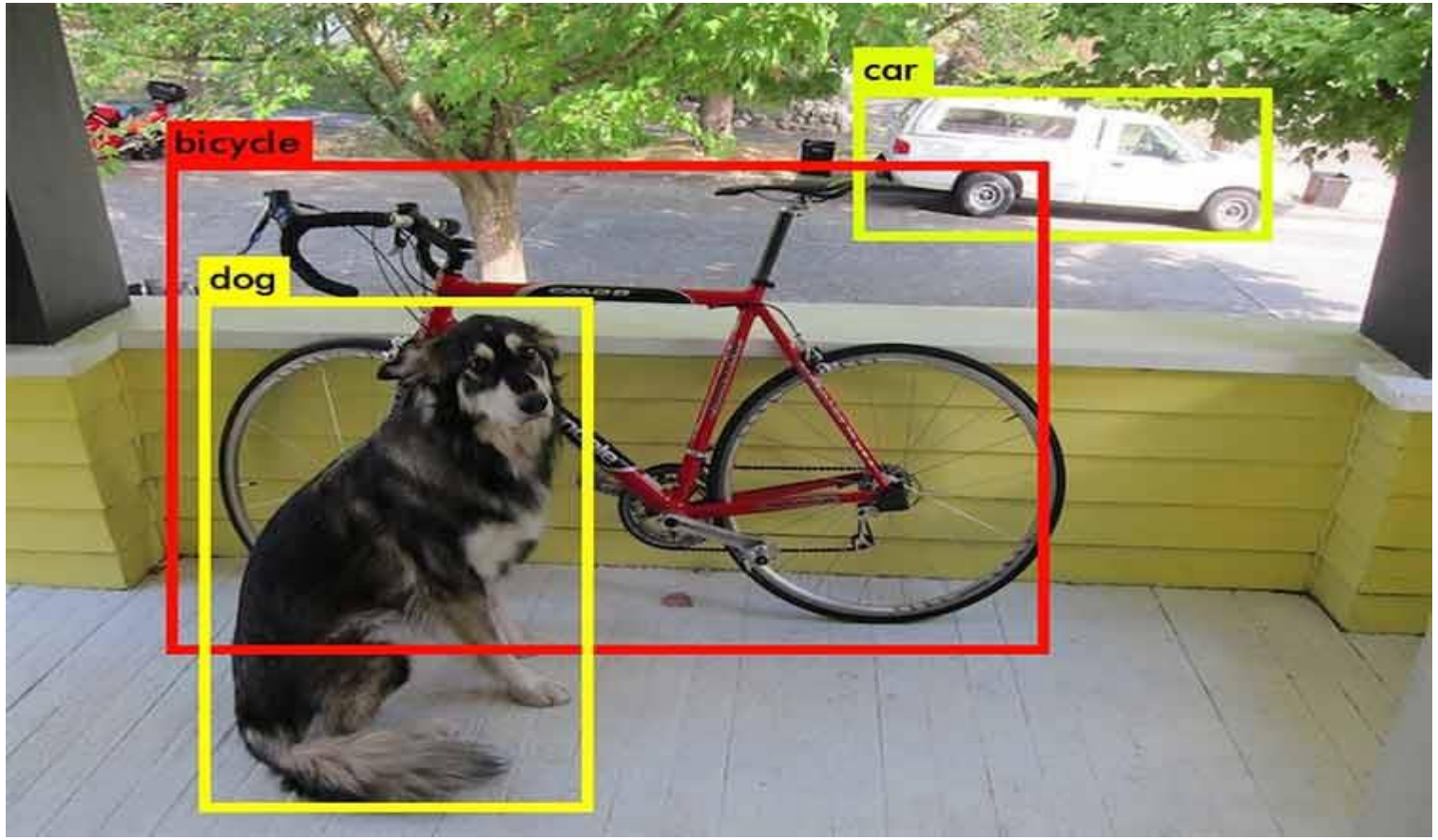


In localization along with the discrete label, we also expect the compute to localize where exactly the object is present in the image. This localization is typically implemented using a bounding box which can be identified by some numerical parameters with respect to the image boundary. Even in this case, the assumption is to have only one object per image.

Introduction & Overview

What is Semantic Segmentation?

c. Object Detection

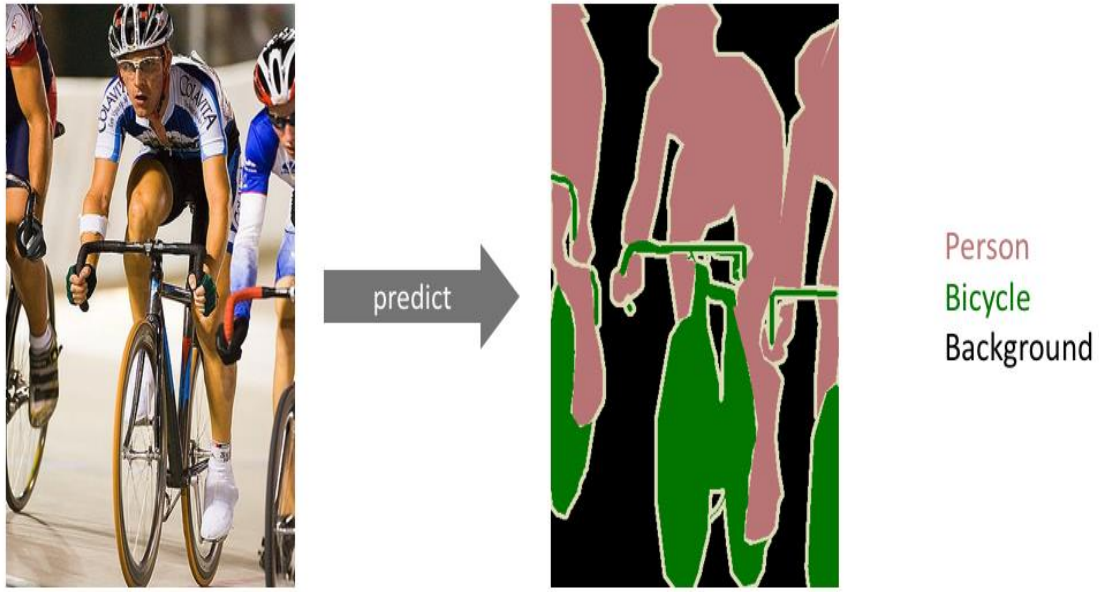


Object Detection extends localization to the next level where now the image is not constrained to have only one object, but can contain multiple objects. The task is to classify and localize all the objects in the image. Here again the localization is done using the concept of bounding box.

Introduction & Overview

What is Semantic Segmentation?

d. Semantic Segmentation



One important thing to note is that we're not separating instances of the same class; **we only care about the category of each pixel.**
In other words, if you have two objects of the same category in your input image, the segmentation map does not inherently distinguish these as separate objects. There exists a different class of models, known as instance segmentation models, which do distinguish between separate objects of the same class.

An example of semantic segmentation, where the goal is to predict class labels for each pixel in the image. (Source)

https://www.jeremyjordan.me/semantic-segmentation/

Introduction & Overview

What is Semantic Segmentation?

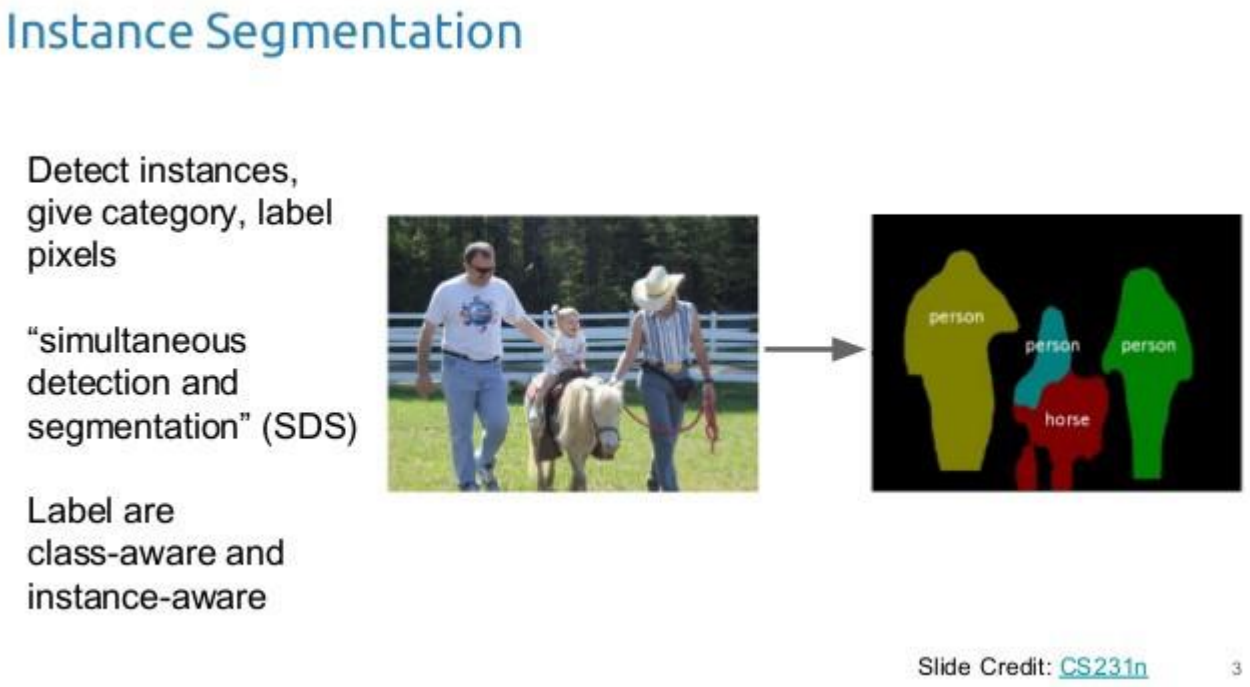
d. Semantic Segmentation

The goal of semantic image segmentation is to label each **pixel** of an image with a corresponding **class** of what is being represented. Because we're predicting for every pixel in the image, this task is commonly referred to as **dense prediction**.
Note that unlike the previous tasks, the expected output in semantic segmentation are not just labels and bounding box parameters. **The output itself is a high resolution image (typically of the same size as input image) in which each pixel is classified to a particular class. Thus it is a pixel level image classification.**

Introduction & Overview

What is Semantic Segmentation?

e. Instance segmentation

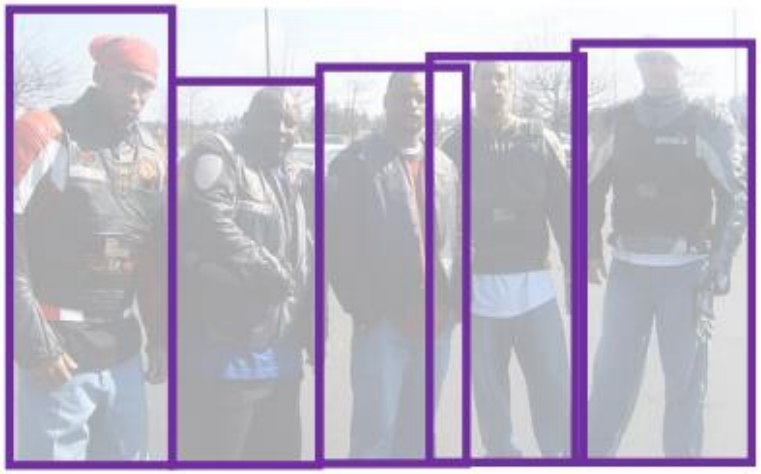


Instance segmentation is one step ahead of semantic segmentation wherein along with pixel level classification, we expect the computer to classify each instance of a class separately. For example in the image above there are 3 people, technically 3 instances of the class "Person". All the 3 are classified separately (in a different color). But semantic segmentation does not differentiate between the instances of a particular class.

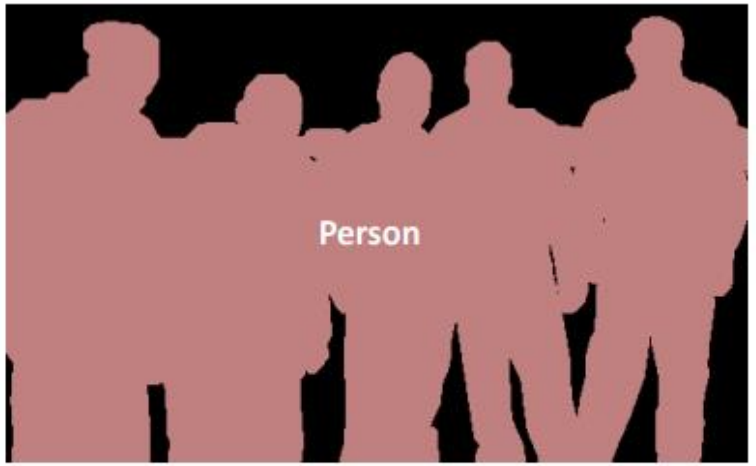
Introduction & Overview

What is Semantic Segmentation?

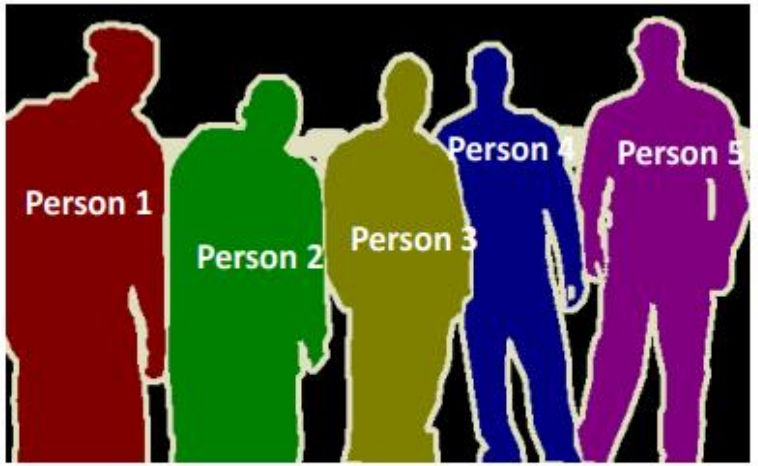
e. Instance segmentation



Object Detection



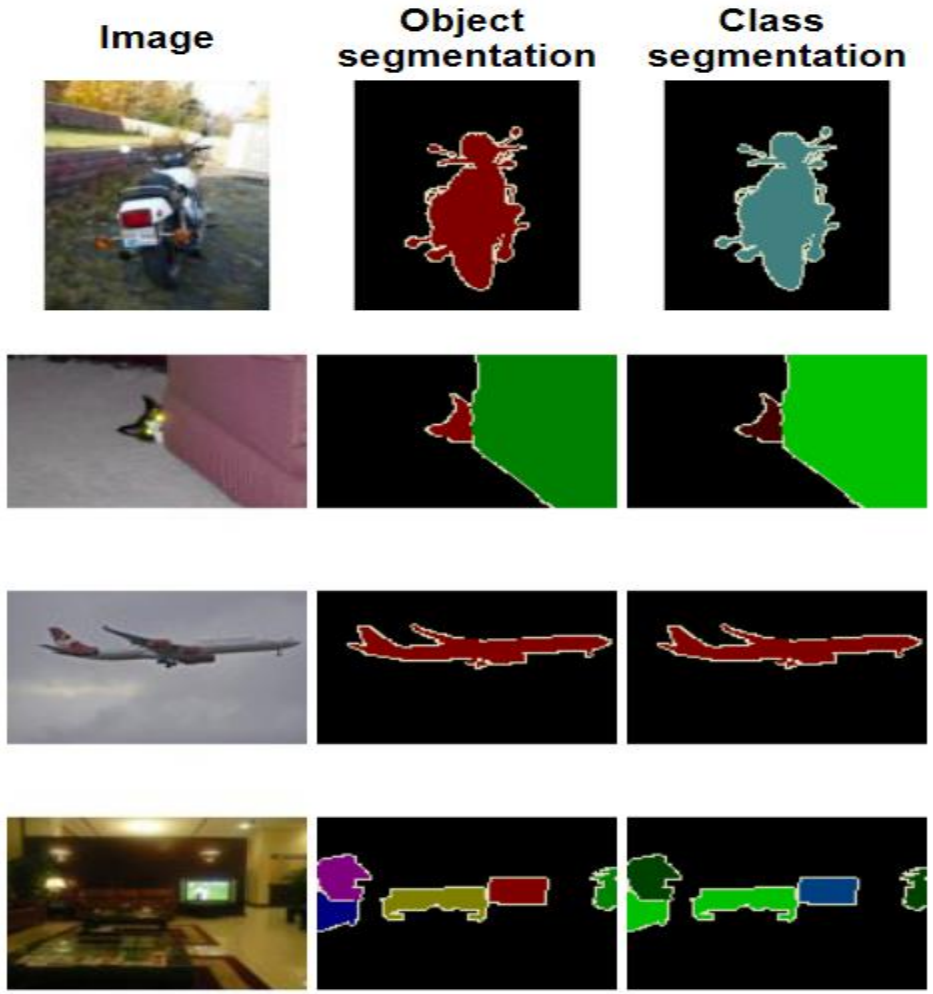
Semantic Segmentation



Instance Segmentation

Introduction & Overview

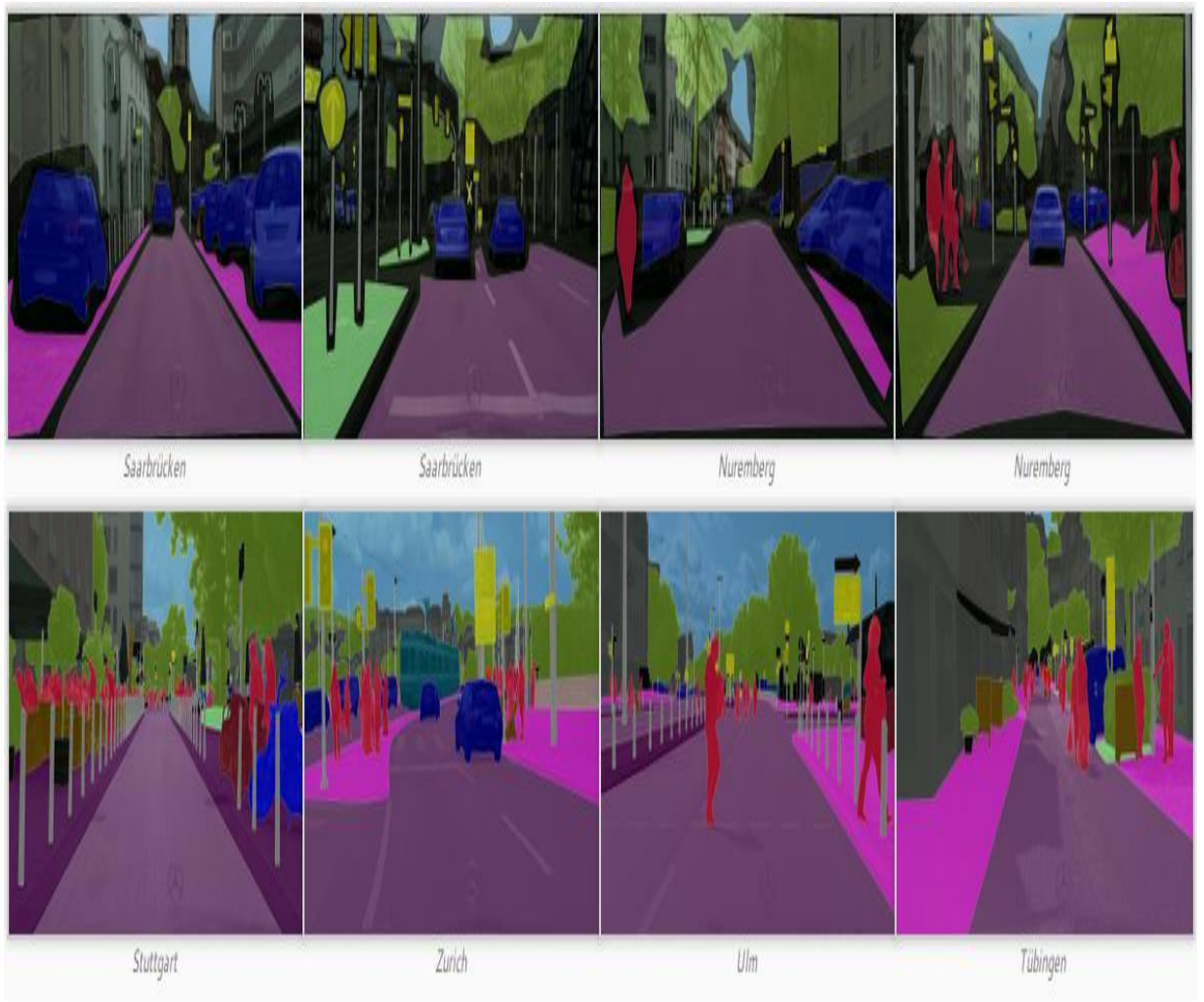
Examples of the COCO dataset for stuff segmentation.
Souce: http://cocodataset.org/



Examples of the 2012 PASCAL VOC dataset for image segmentation.Source: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html

Introduction & Overview

Example of the PASCAL-Context dataset.
Source: https://cs.stanford.edu/~roozbeh/pascal-context/

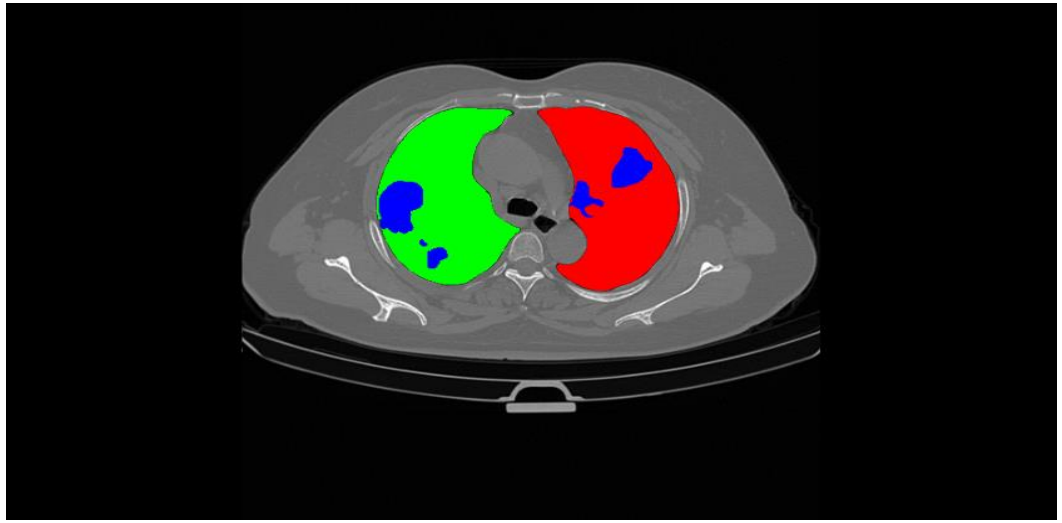


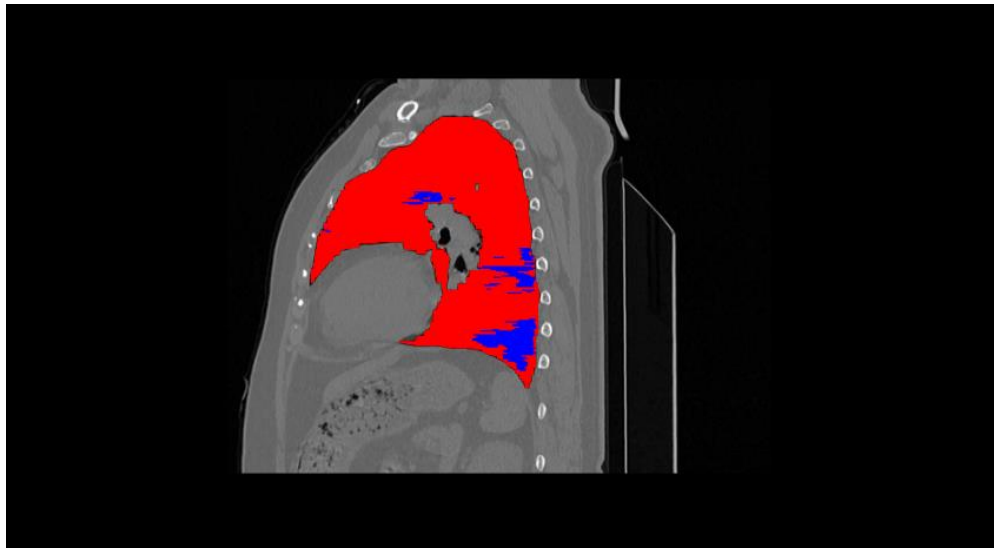
Examples of the Cityscapes dataset. Top: coarse annotations. Bottom: fine annotation. Source: https://www.cityscapes-dataset.com/

Medical Segmentation

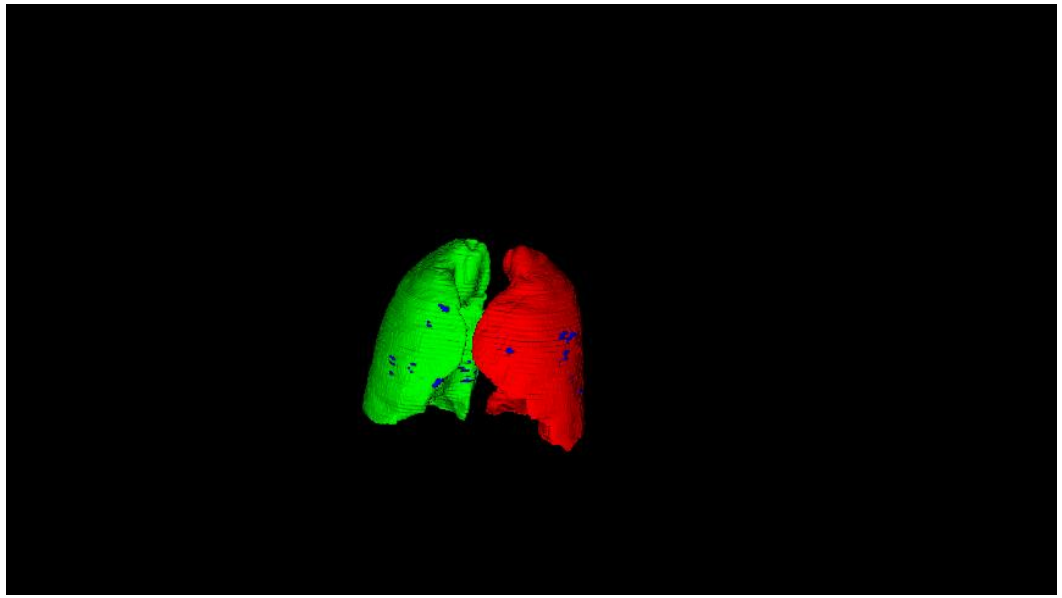
Introduction & Overview

https://gitee.com/junma11/COVID-19-CT-Seg-Benchmark

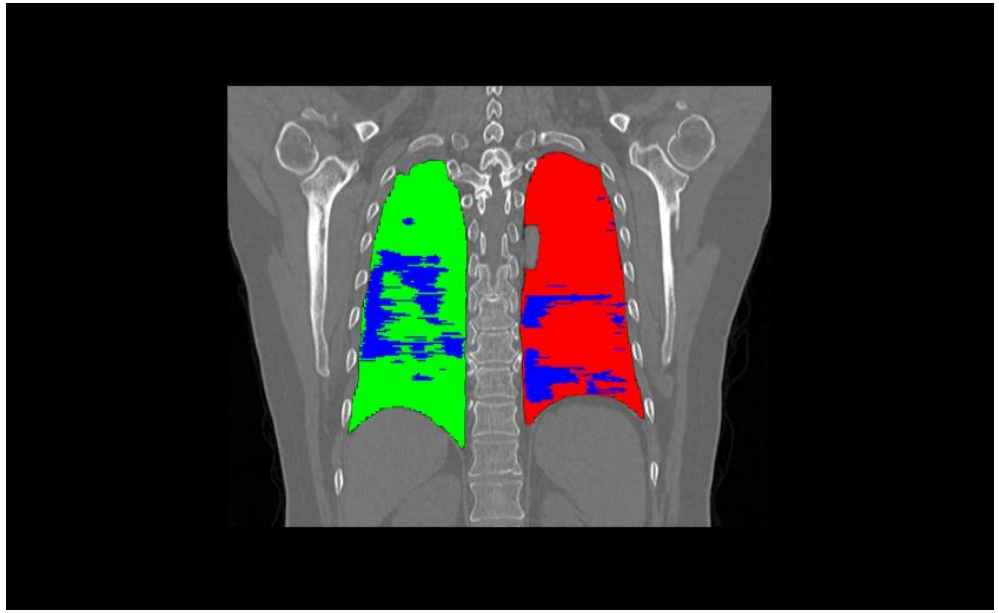




(green: left lung, red: right lung, blue: disease)

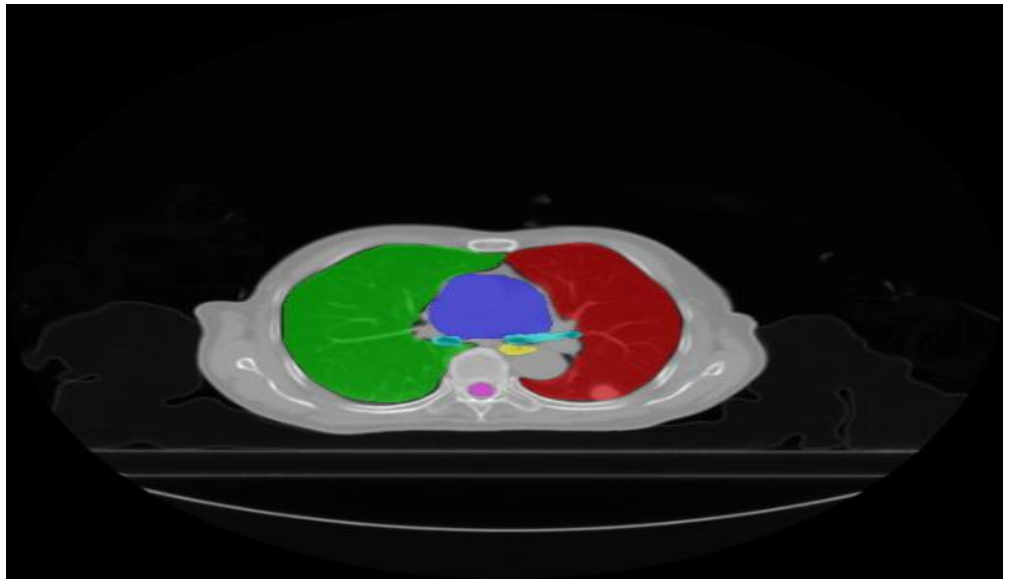


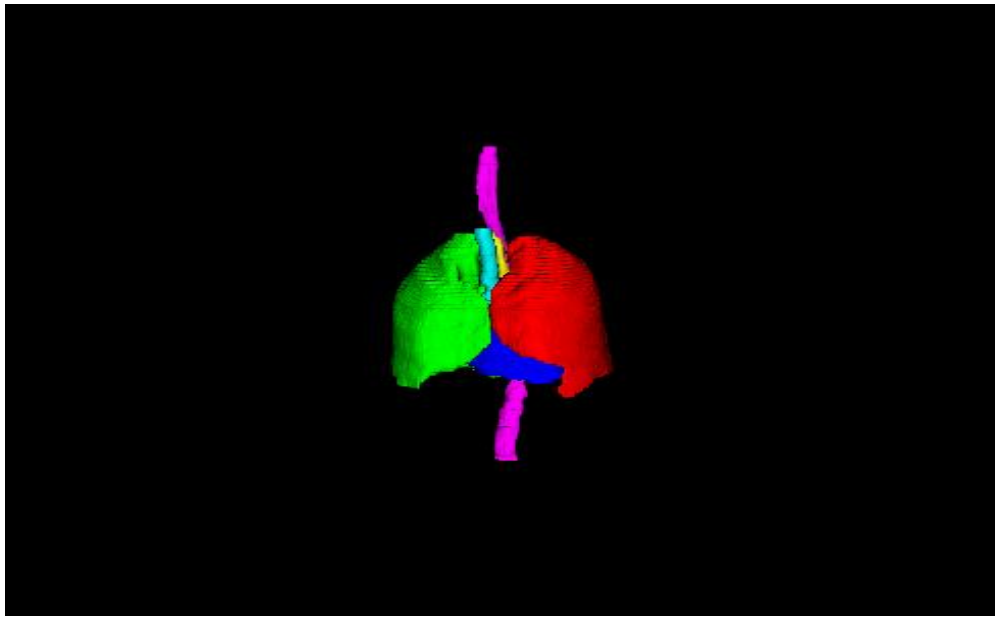
3D view

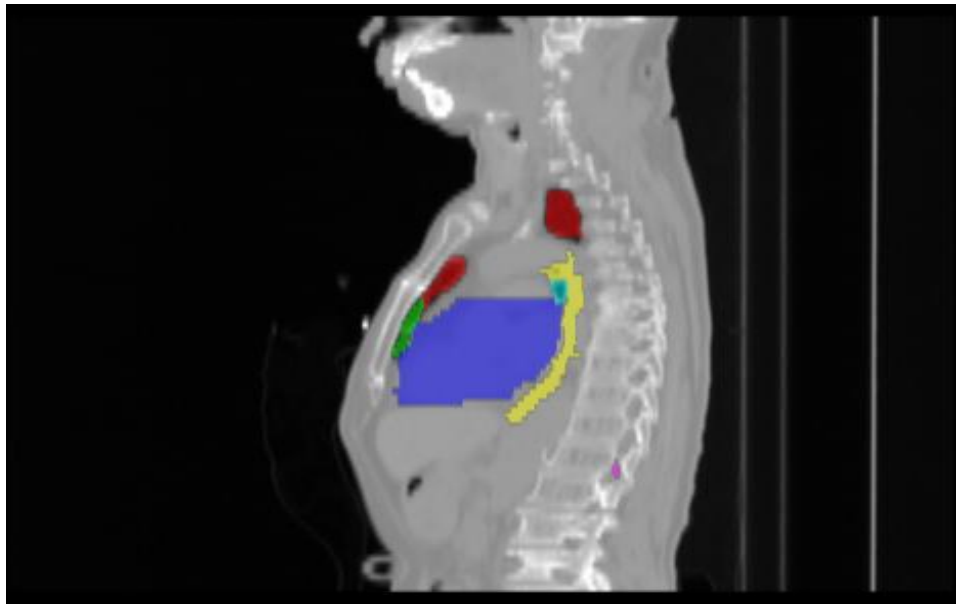


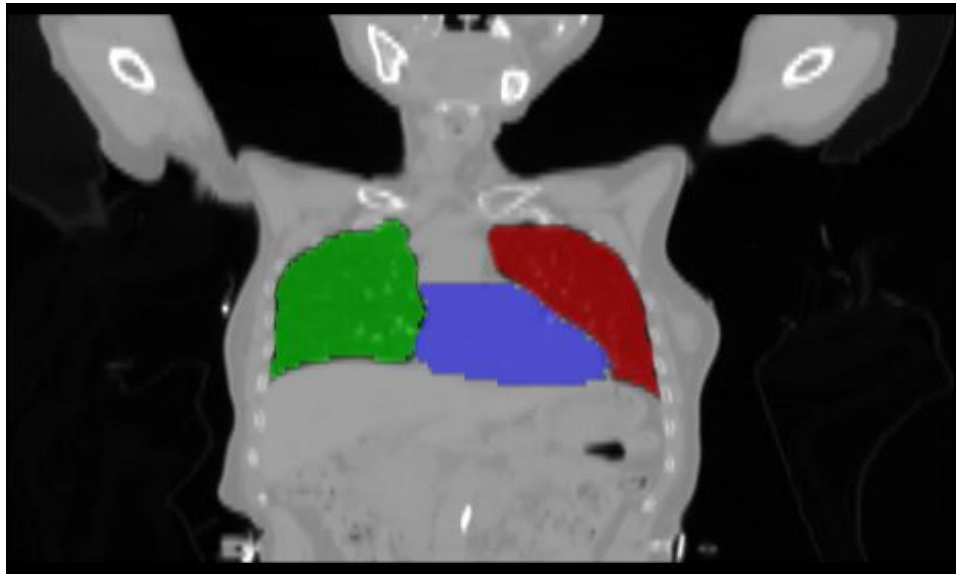
Introduction & Overview

https://gitee.com/junma11/COVID-19-CT-Seg-Benchmark

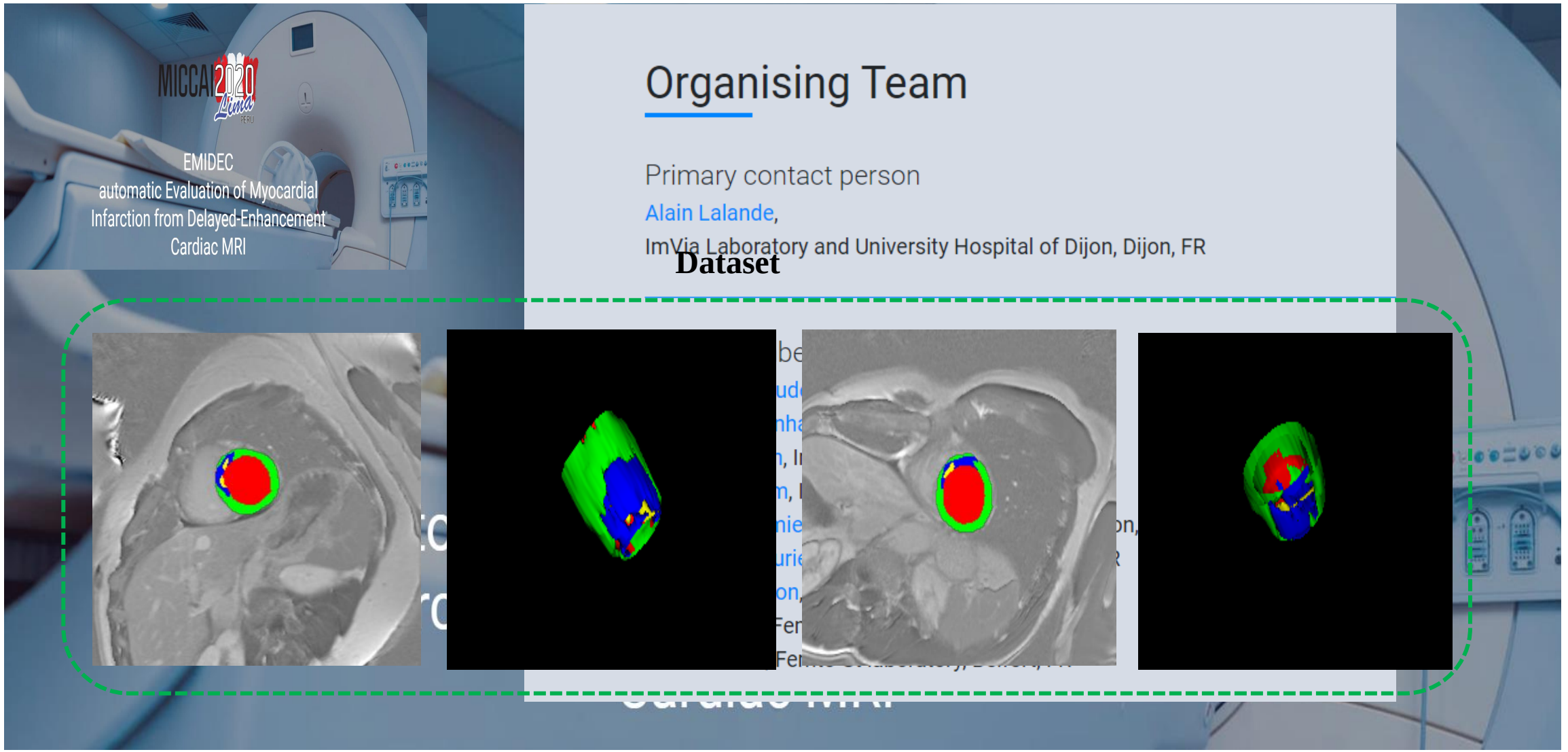






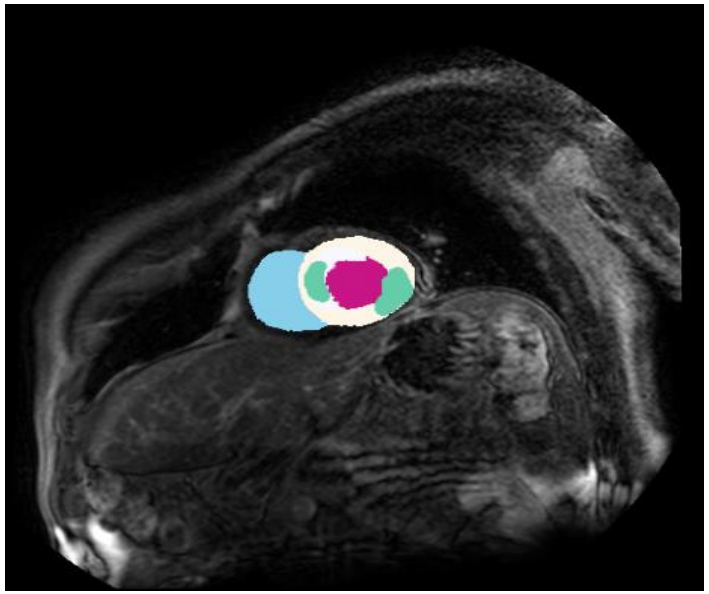
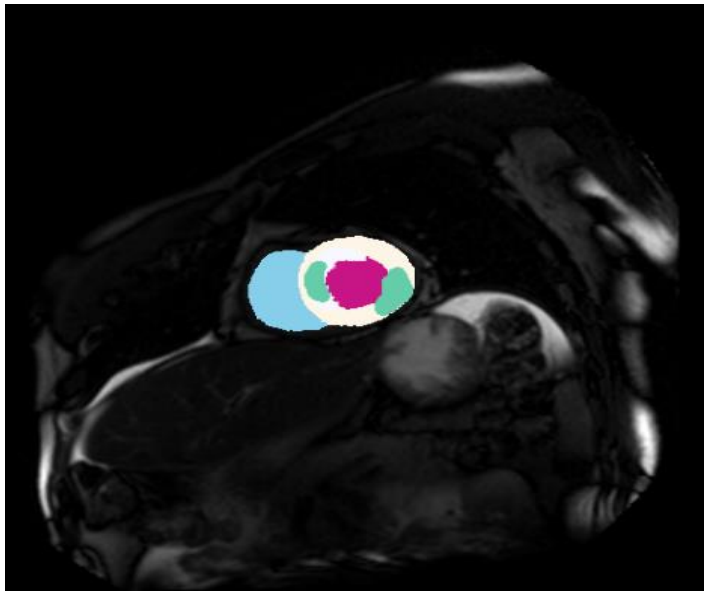
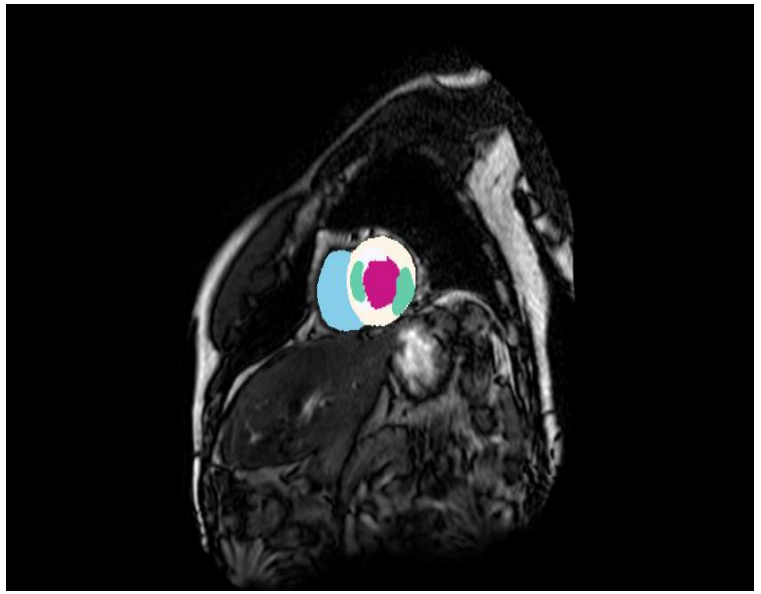
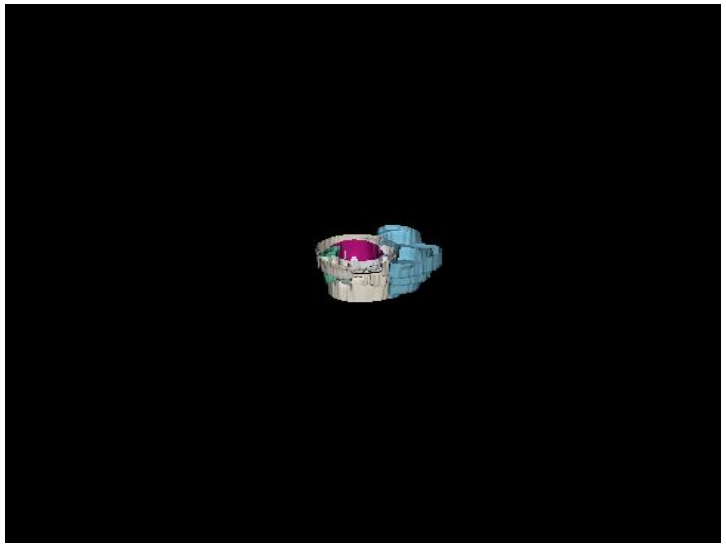


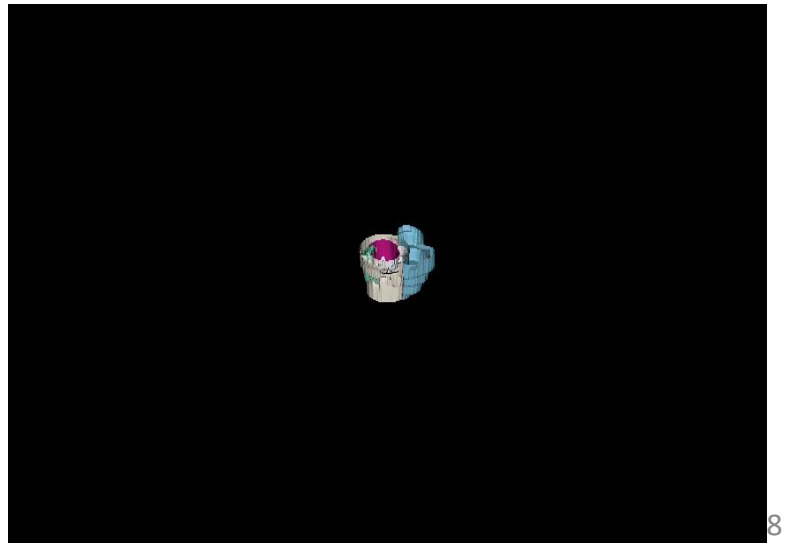
Dataset Launch for MICCAI 2020 Challenge



Introduction & Overview

http://202.120.224.125/urlfilter/urlfilter.html?url=www.sdspeople.fudan.edu.cn/zhuangxiahai/0/MyoPS20/&type=%E6%95%99%E8%82%B2



Introduction & Overview

https://www.miccai2020.org/en/MICCAI-2020-CHALLENGES.html

CHALLENGES

MAIN PAGE >> PROGRAM >> SATELLITE EVENTS

Name	Date	Time	Contact	DOI
2nd Retinal Fundus Glaucoma Challenge	8 October 2020	AM	Huazhu Fu / hzfu(at)ieee.org	10.5281/zenodo.3714946
3D Head and Neck Tumor Segmentation in PET/CT	4 October 2020	AM	Vincent Andrearczyk / vincent.andrearczyk(at)hevs.ch	10.5281/zenodo.3714956
Anatomical Brain Barriers to Cancer Spread: Segmentation from CT and MR Images	4 October 2020	PM	Nadya Shusharina / NSHUSHARINA(at)mgh.harvard.edu	10.5281/zenodo.3714981
Automated Segmentation Of Coronary Arteries	8 October 2020	AM	Ramtin Gharleghi / r.gharleghi(at)student.unsw.edu.au	10.5281/zenodo.3714985
Automatic Evaluation of Myocardial Infarction	4 October	Full Day	Alain Lalande / alain.lalande(at)u-bourgogne.fr	10.5281/zenodo.3714997

Introduction & Overview

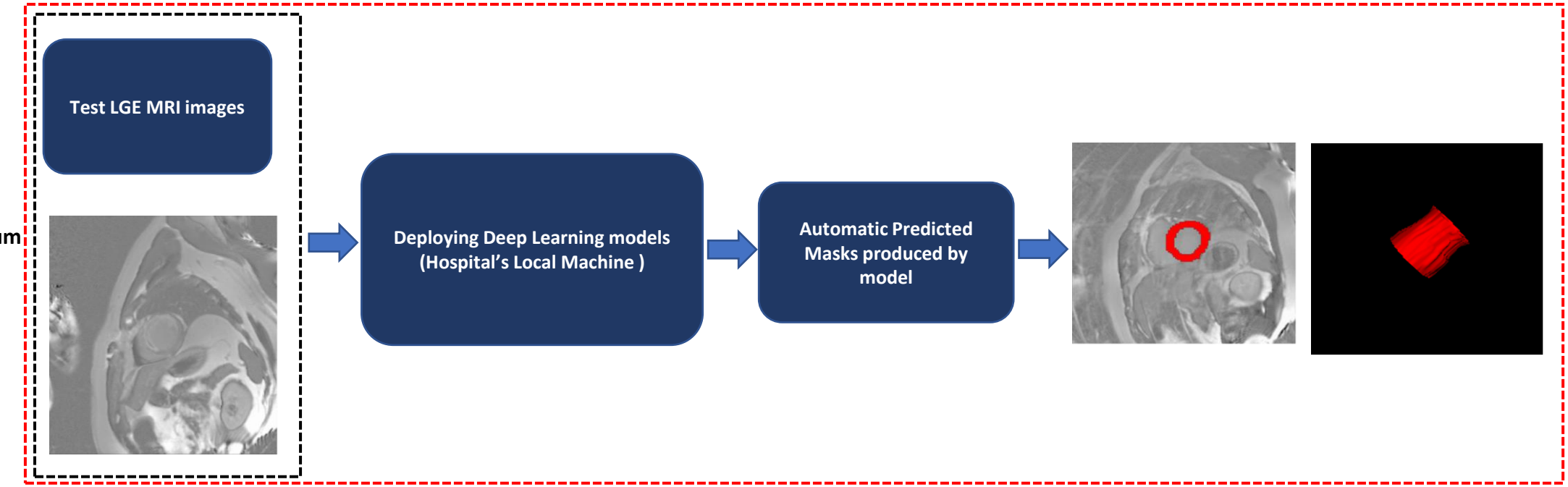
https://www.miccai2020.org/en/MICCAI-2020-CHALLENGES.html

Intracranial Aneurysm Detection and Segmentation Challenge	8 October 2020	AM	Kimberley Timmins / k.m.timmins(at)umcutrecht.nl	10.5281/zenodo.3715847
Large Scale Vertebrae Segmentation Challenge	4 October 2020	AM	Anjany Sekuboyina / anjany.sekuboyina(at)tum.de	10.5281/zenodo.3715865
Learn2Reg – The Challenge	8 October 2020	Full Day	Mattias Heinrich / heinrich(at)imi.uni-luebeck.de	10.5281/zenodo.3715651
Medical Out-of-Distribution Analysis Challenge	8 October 2020	AM	David Zimmerer / d.zimmerer(at)dkfz.de	10.5281/zenodo.3715869
MICCAI Brain Tumor Segmentation (BraTS) 2020 Benchmark: "Prediction of Survival and Pseudoprogression"	4 October 2020	PM	Spyridon Bakas / sbakas(at)upenn.edu	10.5281/zenodo.3718903
Multi-Centre, Multi-Vendor & Multi-Disease Cardiac Image Segmentation Challenge	4 October 2020	PM	Víctor M. Campello /victor.campello(at)ub.edu	10.5281/zenodo.3715889

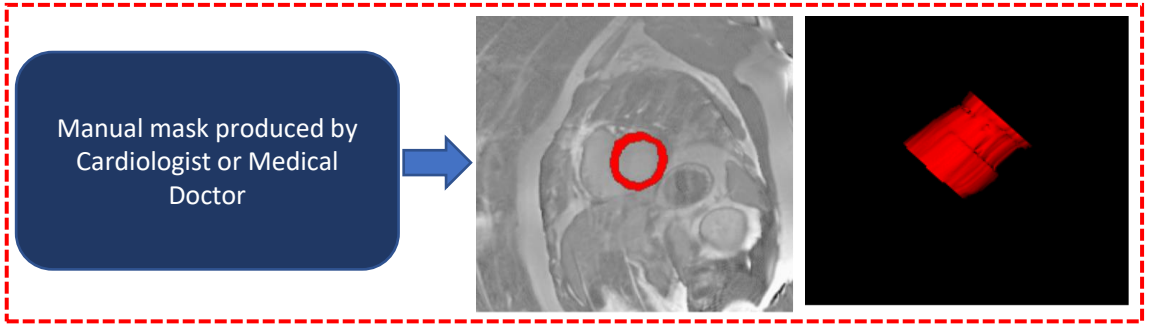
Deployment of Proposed Model

Deployment of deep learning model in local or remote machine Example:

Automatic Myocardium Segmentation



Manual Myocardium Segmentation



Segmentation Basic Representation

Representing the task

Simply, our goal is to take either a RGB color image (height×width×3) or a grayscale image (height×width×1) and output a segmentation map where each pixel contains a class label represented as an integer (height×width×1).

Input

segmented

1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

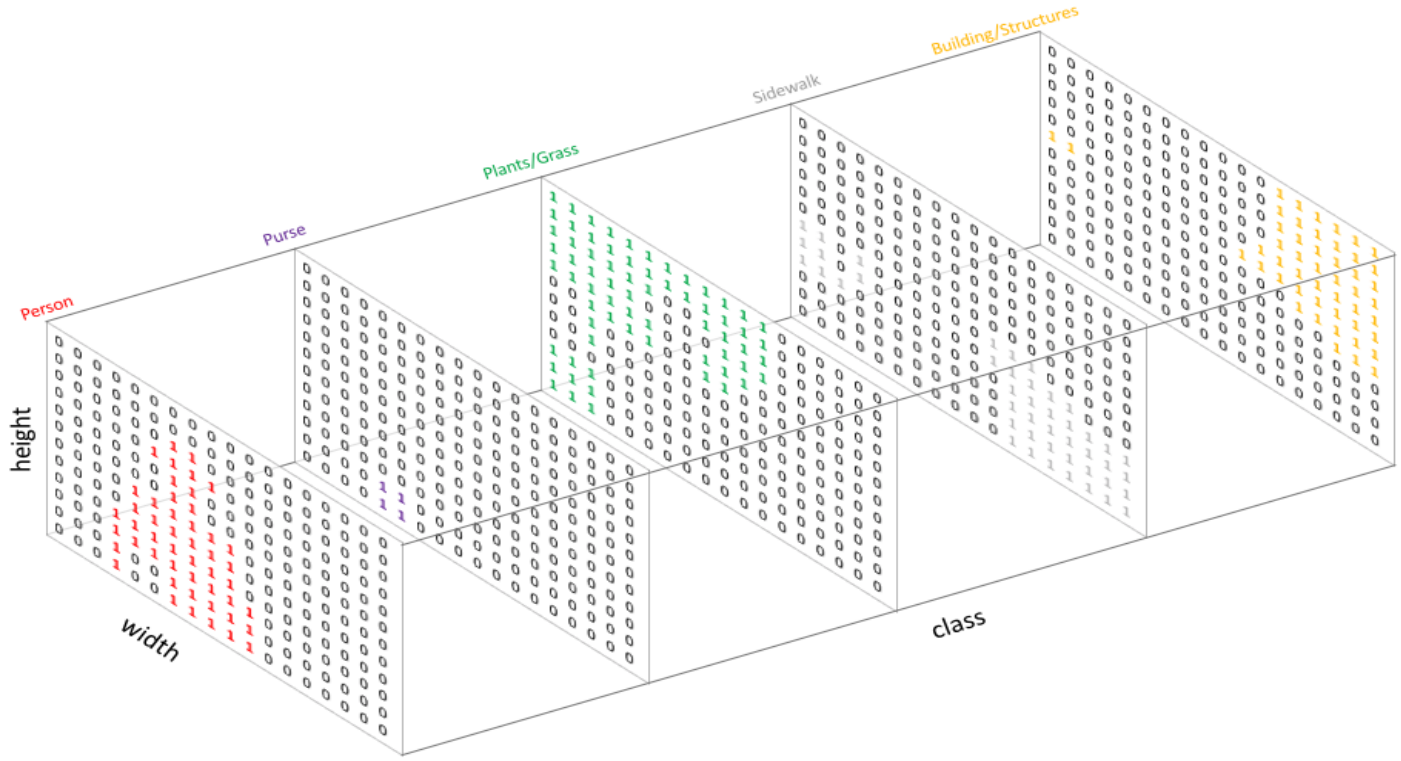
Semantic Labels

https://www.jeremyjordan.me/semantic-segmentation/

Segmentation Basic Representation

Representing the task

Similar to how we treat standard categorical values, we'll create our **target** by one-hot encoding the class labels - essentially creating an output channel for each of the possible classes.



https://www.jeremyjordan.me/semantic-segmentation/

Segmentation Basic Representation

Representing the task

When we overlay a single channel of our target (or prediction), we refer to this as a mask which illuminates the regions of an image where a specific class is present.

https://www.jeremyjordan.me/semantic-segmentation/

Segmentation Basic Representation

Representing the task

A prediction can be collapsed into a segmentation map (as shown in the first image) by taking the argmax of each depth-wise pixel vector.

0: Background/Unknown
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

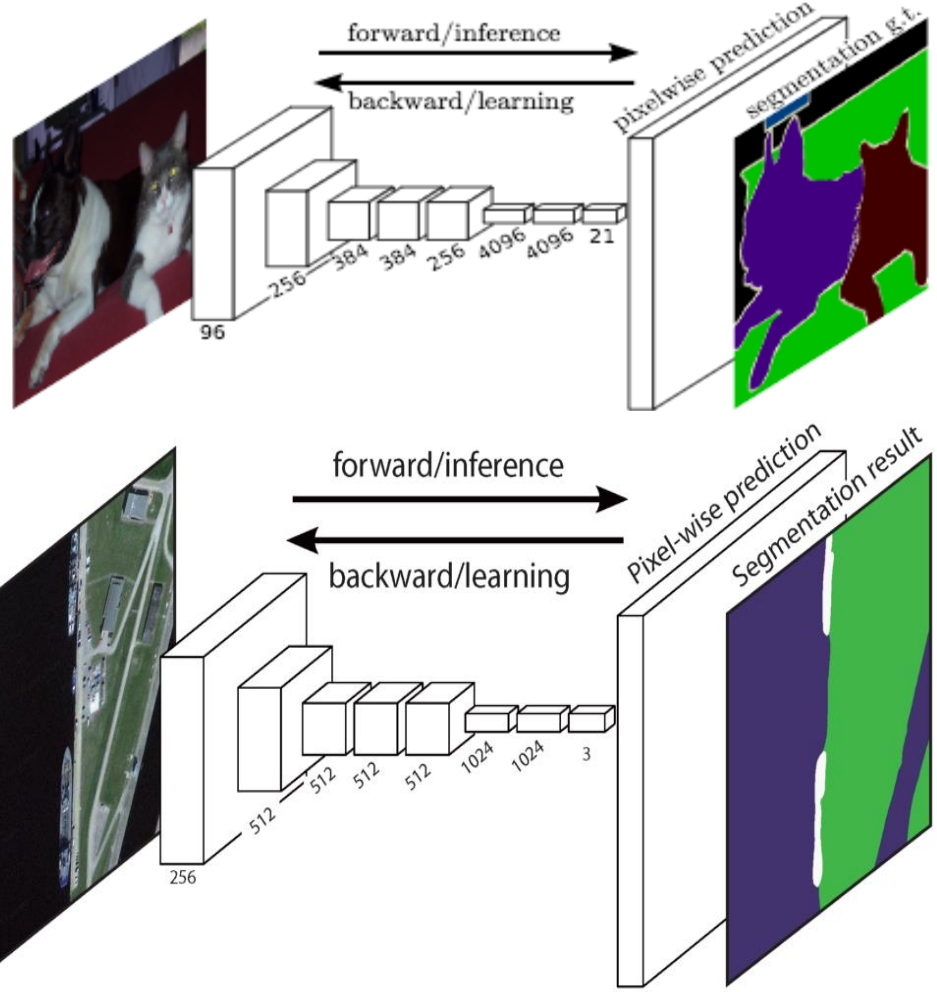
When we overlay a single channel of our target (or prediction), we refer to this as a mask which illuminates the regions of an image where a specific class is present.

https://www.jeremyjordan.me/semantic-segmentation/

Segmentation DL models

Fully Convolutional Networks for Semantic Segmentation (PAMI, 2016)

The model proposed in this paper achieves a performance of 67.2% mean IU on PASCAL VOC 2012.



One issue in this specific FCN is that by propagating through several alternated convolutional and pooling layers, the resolution of the output feature maps is down sampled. Therefore, the direct predictions of FCN are typically in low resolution, resulting in relatively fuzzy object boundaries.

A variety of more advanced FCN-based approaches have been proposed to address this issue, including **SegNet, DeepLab-CRF, and Dilated Convolutions.**

https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc

Segmentation DL models

U-Net: Convolutional Networks for Biomedical Image Segmentation (MICCAI, 2015)

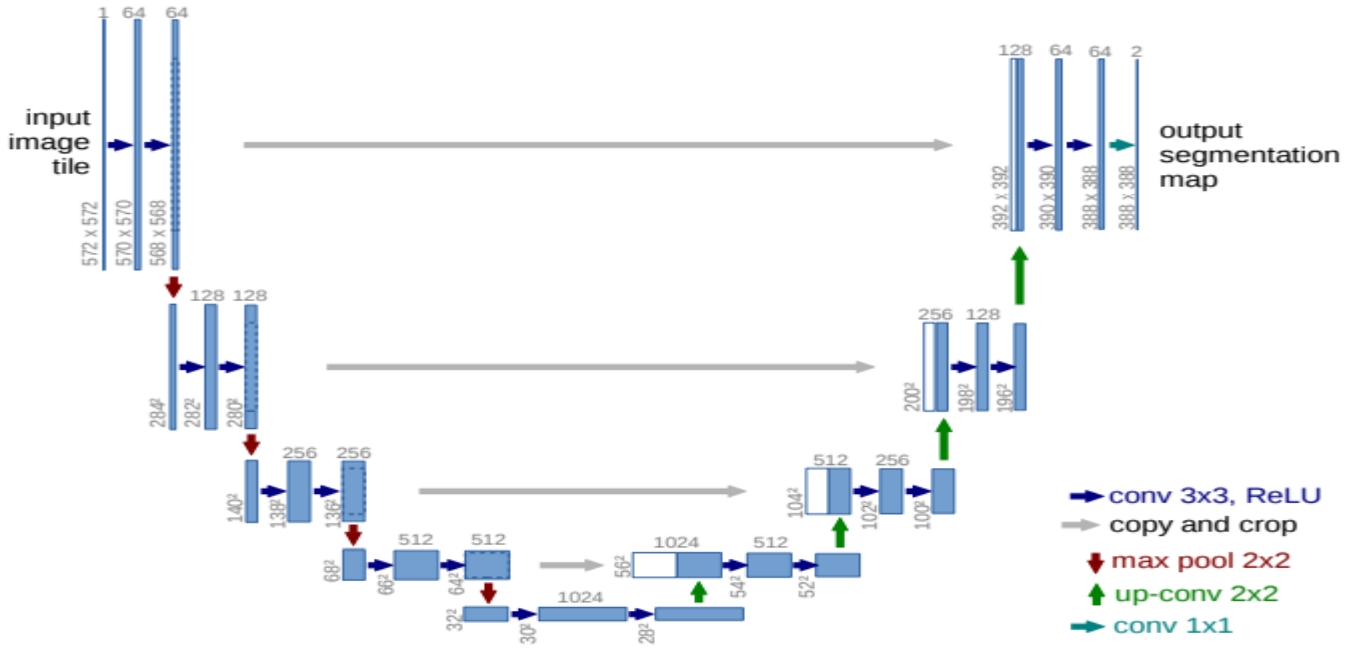


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows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

https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc

Segmentation DL models

The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation (2017)

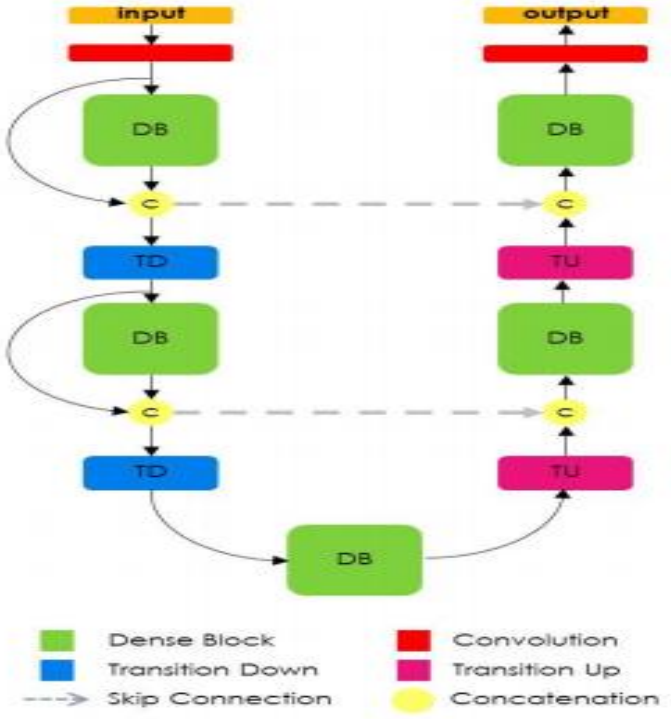
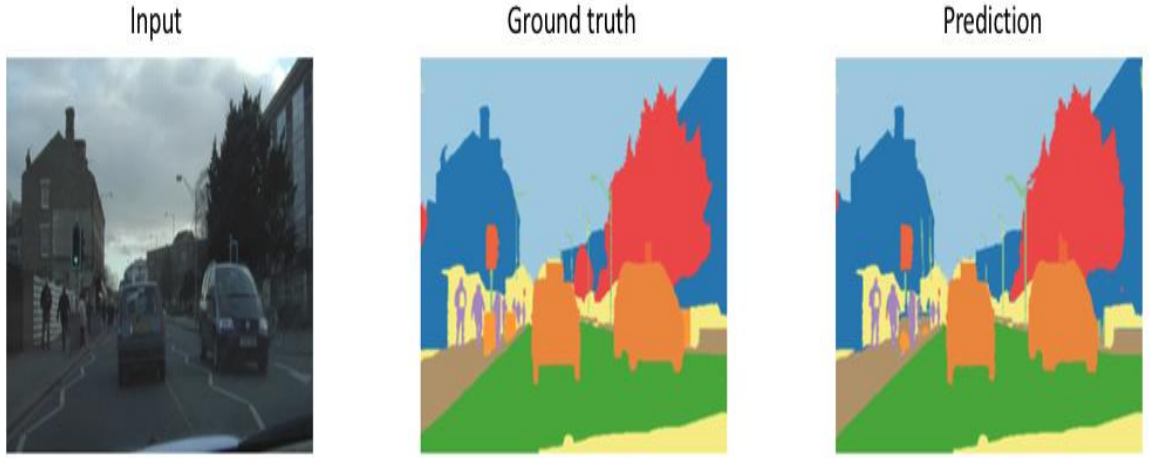


Figure 1. Diagram of our architecture for semantic segmentation.



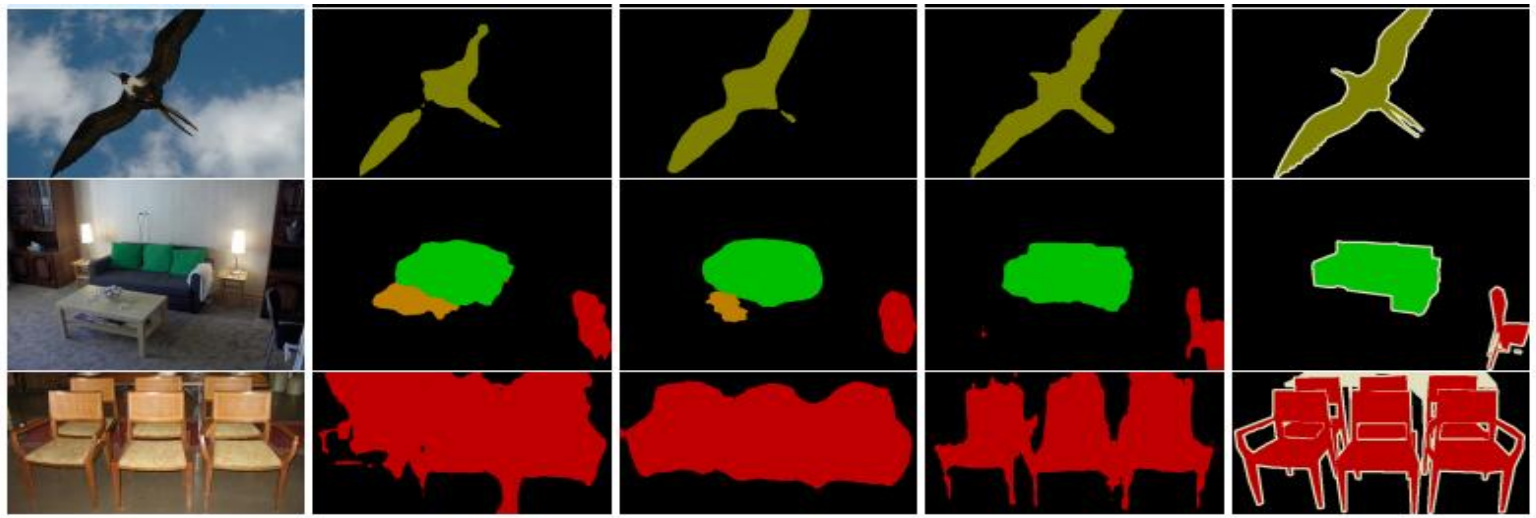
The FC-DenseNet103 model acheives state of the art results (Oct 2017) on the CamVid dataset.

https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc

Segmentation DL models

Multi-Scale Context Aggregation by Dilated Convolutions (ICLR, 2016)

The module was tested on the Pascal VOC 2012 dataset. It proves that adding a context module to existing semantic segmentation architectures improves their accuracy.



(a) Image (b) FCN-8s (c) DeepLab (d) Our front end (e) Ground truth

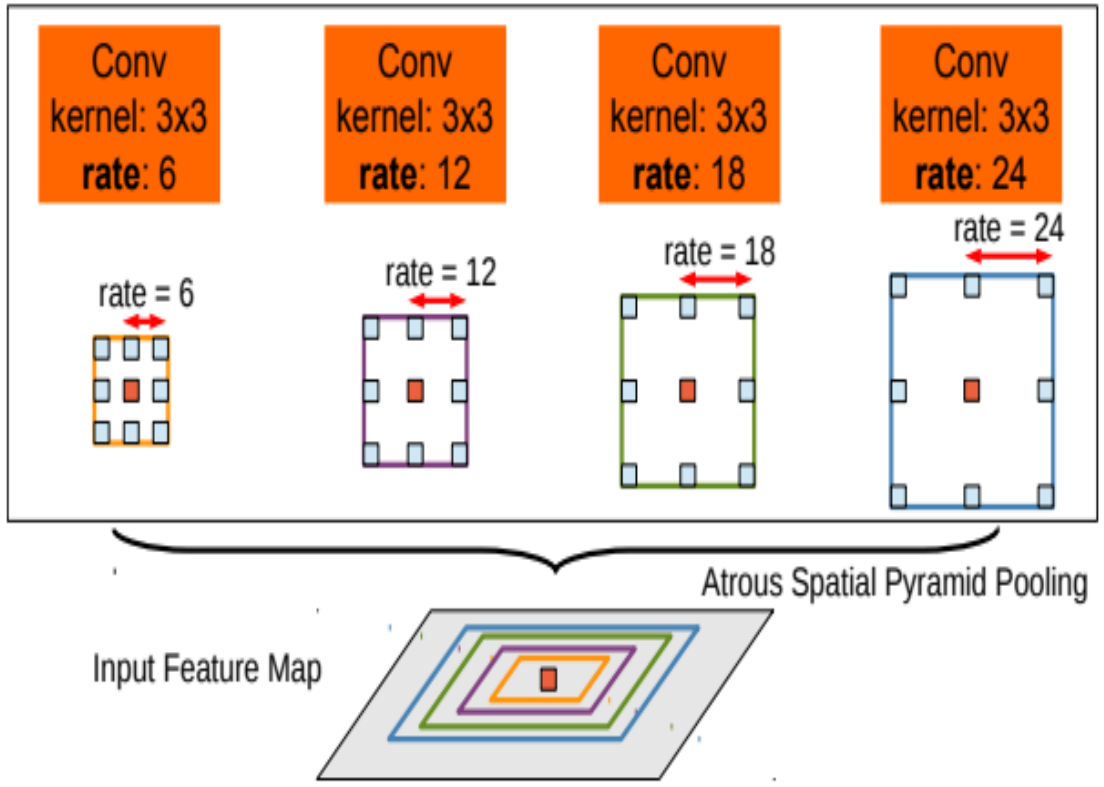
Figure 2: Semantic segmentations produced by different adaptations of the VGG-16 classification network. From left to right: (a) input image, (b) prediction by FCN-8s (Long et al., 2015), (c) prediction by DeepLab (Chen et al., 2015a), (d) prediction by our simplified front-end module, (e) ground truth.

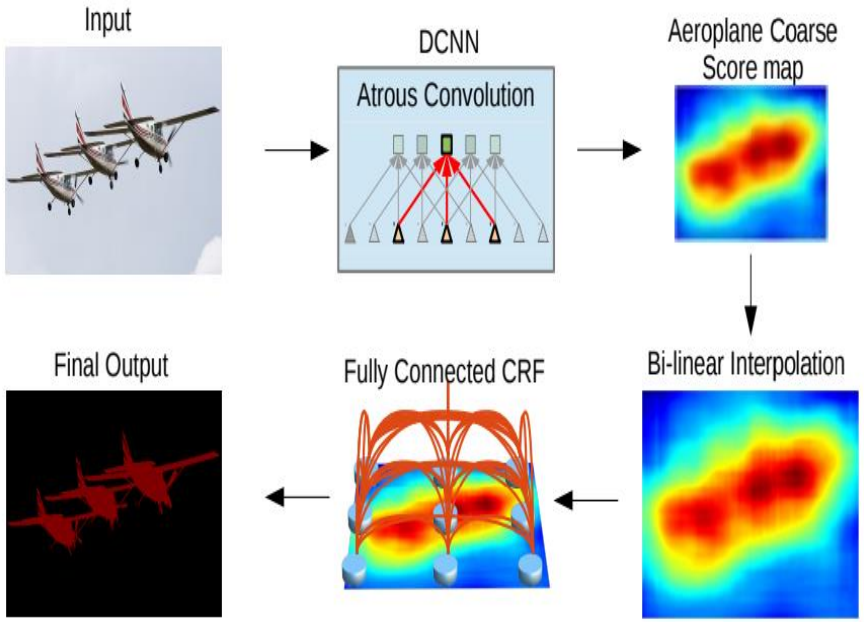
https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc

Segmentation DL models

DeepLab, DeepLabv3 and DeepLabv3+

https://medium.com/@arthur_ouaknine/review-of-deep-learning-algorithms-for-image-semantic-segmentation-509a600f7b57



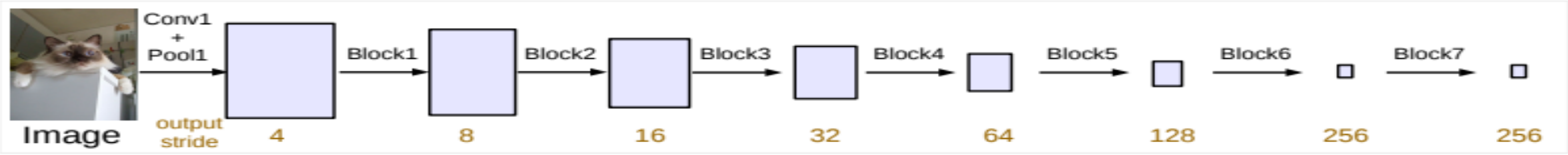


Atrous Spatial Pyramid Pooling (ASPP) exploiting multiple scale of objects to classify the pixel in the center.
Source: L.-C. Chen et al. (2017)

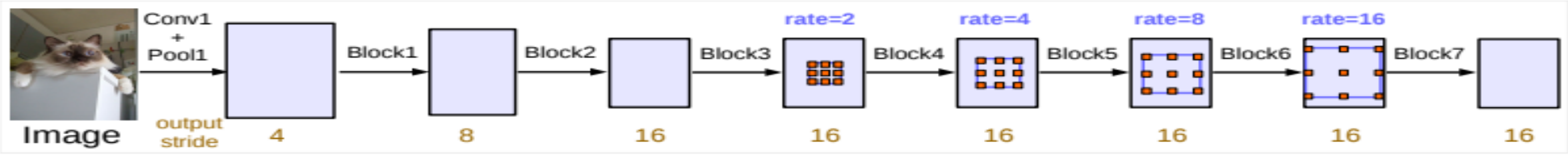
Segmentation DL models

DeepLabv3

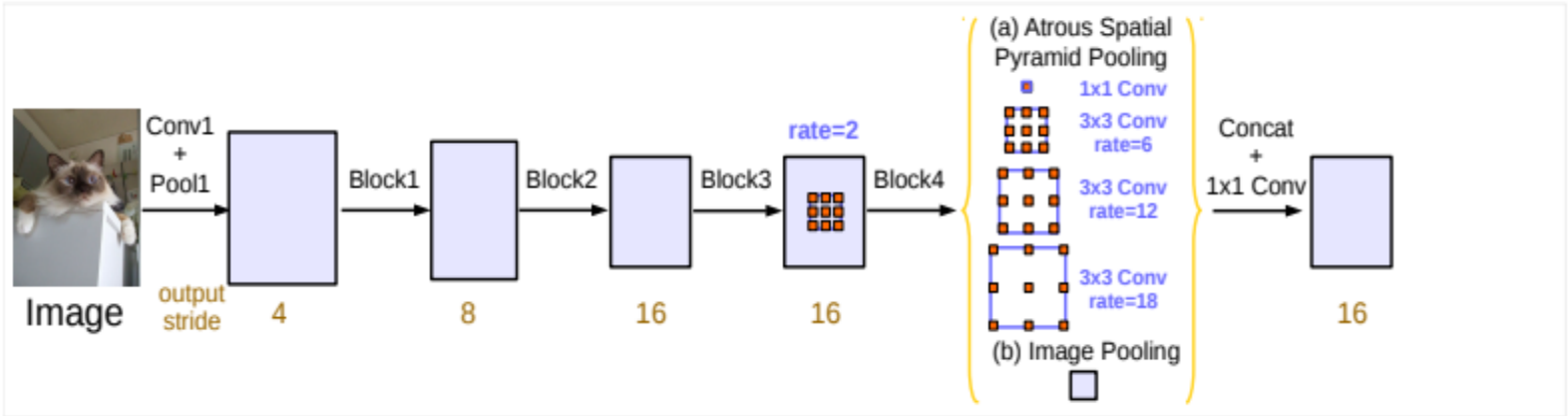
https://medium.com/@arthur_ouaknine/review-of-deep-learning-algorithms-for-image-semantic-segmentation-509a600f7b57



(a) Going deeper without atrous convolution.



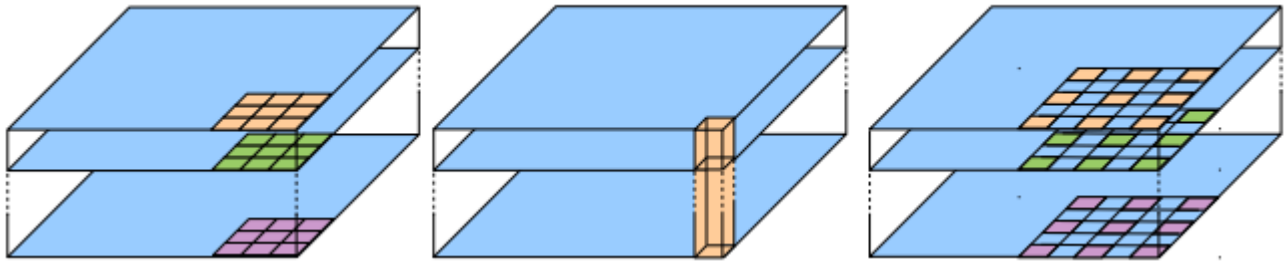
(b) Going deeper with atrous convolution. Atrous convolution with $rate > 1$ is applied after block3 when $output\_stride = 16$.

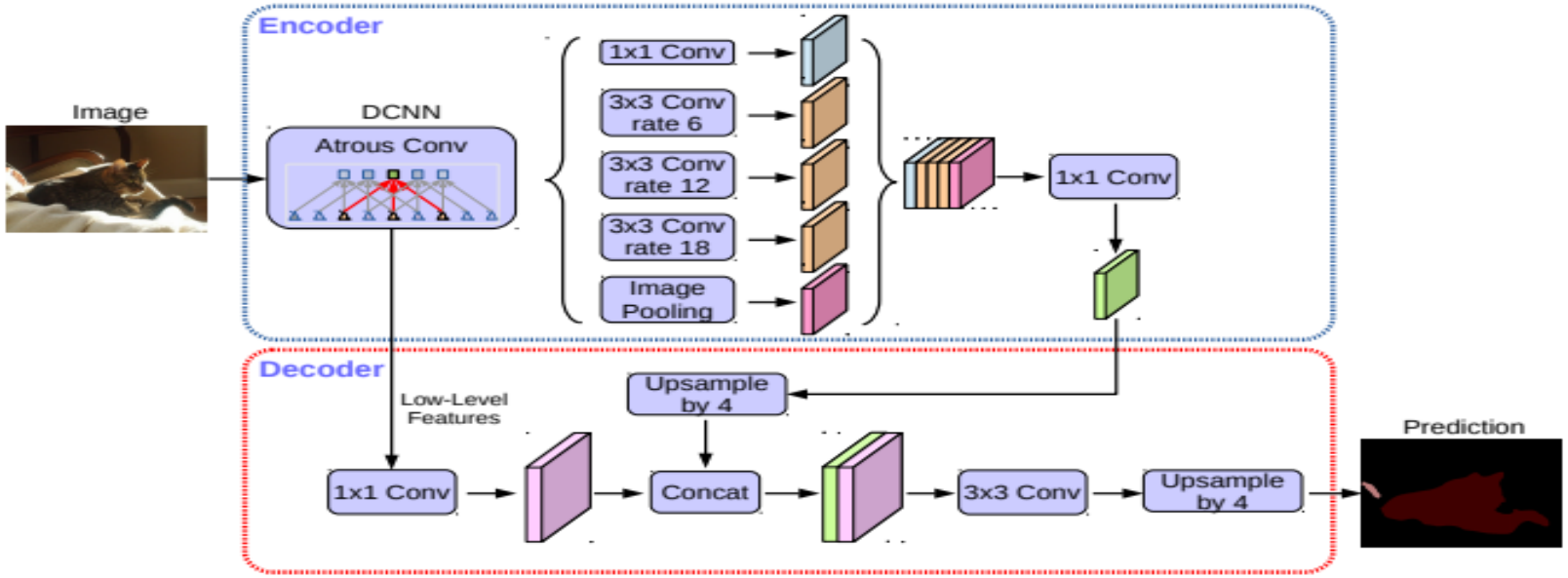


Segmentation DL models

DeepLabv3+

https://medium.com/@arthur_ouaknine/review-of-deep-learning-algorithms-for-image-semantic-segmentation-509a600f7b57





Segmentation DL models

DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs (TPAMI, 2017)

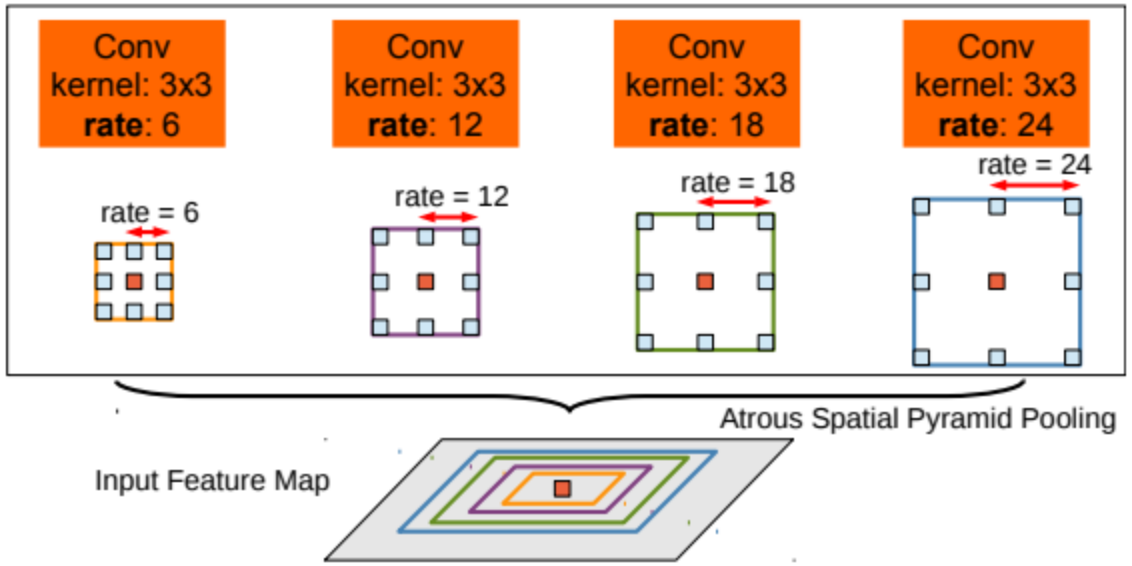
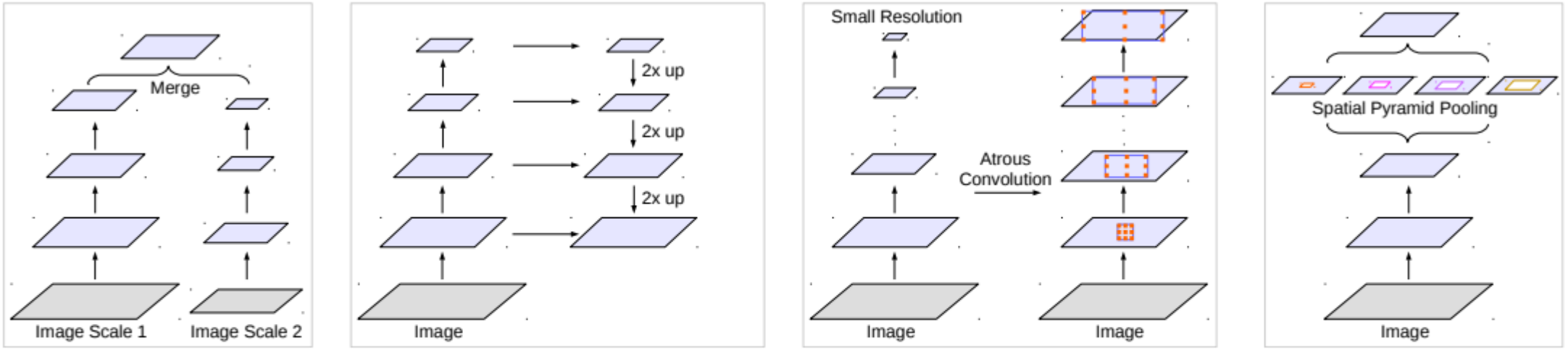


Fig. 4: Atrous Spatial Pyramid Pooling (ASPP). To classify the center pixel (orange), ASPP exploits multi-scale features by employing multiple parallel filters with different rates. The effective Field-Of-Views are shown in different colors.

https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc

Segmentation DL models

Rethinking Atrous Convolution for Semantic Image Segmentation (2017)



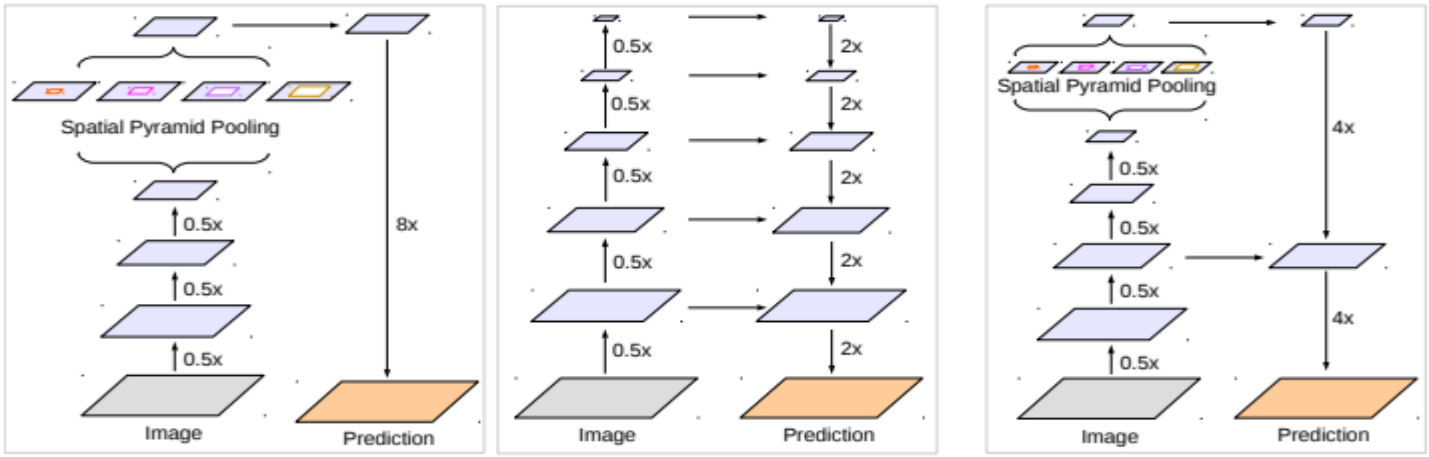
(a) Image Pyramid (b) Encoder-Decoder (c) Deeper w. Atrous Convolution (d) Spatial Pyramid Pooling

Figure 2. Alternative architectures to capture multi-scale context.

https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc

Segmentation DL models

Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation (ECCV, 2018)



(a) Spatial Pyramid Pooling (b) Encoder-Decoder (c) Encoder-Decoder with Atrous Conv

Fig. 1. We improve DeepLabv3, which employs the spatial pyramid pooling module (a), with the encoder-decoder structure (b). The proposed model, DeepLabv3+, contains rich semantic information from the encoder module, while the detailed object boundaries are recovered by the simple yet effective decoder module. The encoder module allows us to extract features at an arbitrary resolution by applying atrous convolution.

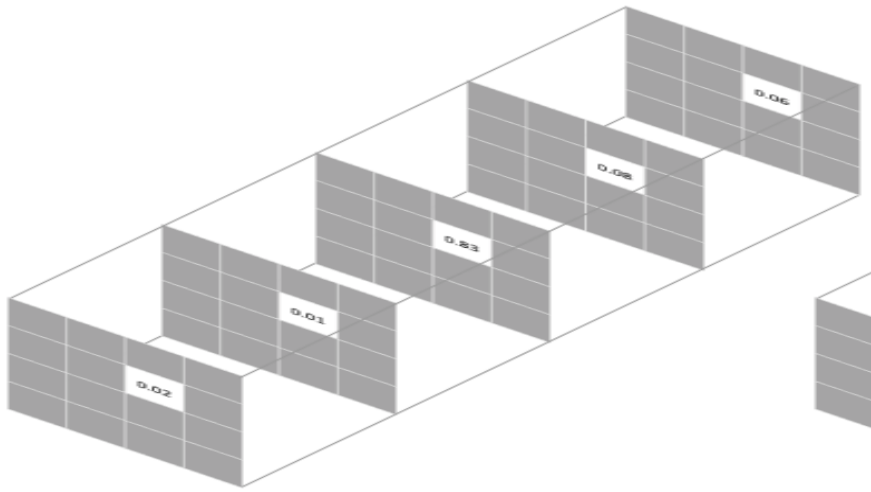
https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc

Loss Computation Segmentation DL models

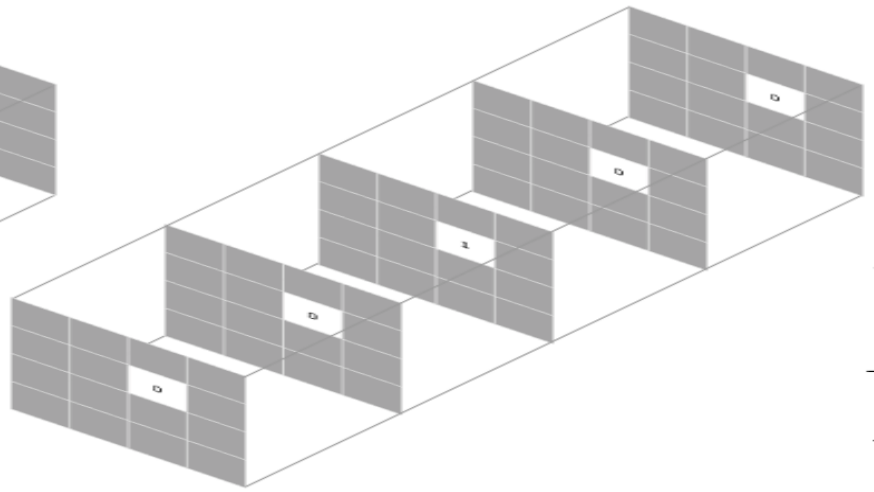
Representing the task

Defining a loss function

The most commonly used loss function for the task of image segmentation is a **pixel-wise cross entropy loss**. This loss examines *each pixel individually*, comparing the class predictions (depth-wise pixel vector) to our one-hot encoded target vector.



Prediction for a selected pixel



Target for the corresponding pixel

Pixel-wise loss is calculated as the log loss, summed over all possible classes

$$-\sum_{classes} y_{true} \log\left(y_{pred}\right)$$

This scoring is repeated over all **pixels** and averaged

https://www.jeremyjordan.me/semantic-segmentation/

Loss Computation Segmentation DL models

Representing the task

Another popular loss function for image segmentation tasks is based on the Dice coefficient, which is essentially a measure of overlap between two samples. This measure ranges from 0 to 1 where a Dice coefficient of 1 denotes perfect and complete overlap. The Dice coefficient was originally developed for binary data, and can be calculated as:

$$Dice = 2|A \cap B| / |A| + |B|$$

where $|A \cap B|$ represents the common elements between sets A and B, and $|A|$ represents the number of elements in set A (and likewise for set B).

For the case of evaluating a Dice coefficient on predicted segmentation masks, we can approximate $|A \cap B|$ as the element-wise multiplication between the prediction and target mask, and then sum the resulting matrix.

https://www.jeremyjordan.me/semantic-segmentation/

Loss Computation Segmentation DL models

Representing the task

Dice=2|A∩B||A|+|B|
where |A∩B| represents the common elements between sets A and B, and |A| represents the number of elements in set A (and likewise for set B).

For the case of evaluating a Dice coefficient on predicted segmentation masks, we can approximate |A∩B| as the element-wise multiplication between the prediction and target mask, and then sum the resulting matrix.

$$|A \cap B| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \xrightarrow{\text{element-wise multiply}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

prediction target

https://www.jeremyjordan.me/semantic-segmentation/

Loss Computation Segmentation DL models

Representing the task

$$|A \cap B| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \xrightarrow{\text{element-wise multiply}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

prediction target

Because our target mask is binary, we effectively zero-out any pixels from our prediction which are not "activated" in the target mask. For the remaining pixels, we are essentially penalizing low-confidence predictions; a higher value for this expression, which is in the numerator, leads to a better Dice coefficient.

https://www.jeremyjordan.me/semantic-segmentation/

Loss Computation Segmentation DL models

Representing the task

In order to quantify |A| and |B|, some researchers use the simple sum whereas other researchers prefer to use the squared sum for this calculation. I don't have the practical experience to know which performs better empirically over a wide range of tasks, so I'll leave you to try them both and see which works better.

$$|A| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix}^{2\ (optional)} \xrightarrow{\text{sum}} 7.82$$

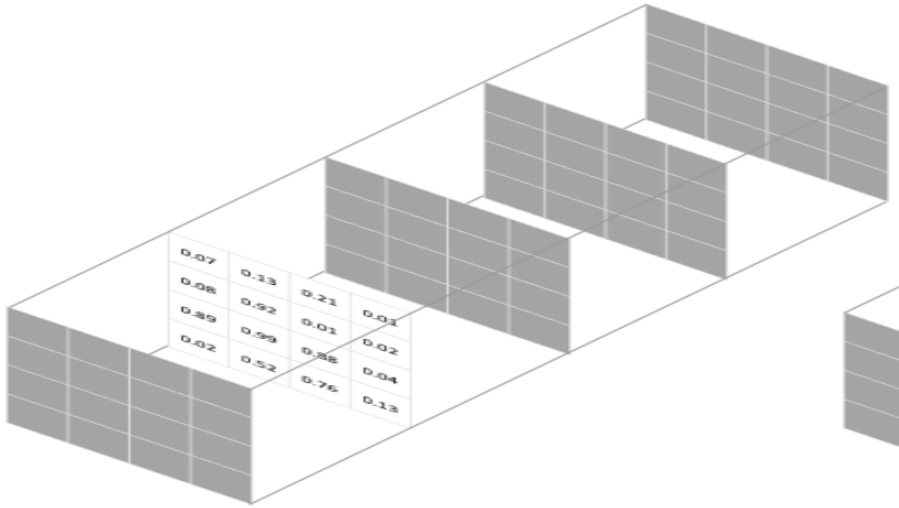
$$|B| = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{2\ (optional)} \xrightarrow{\text{sum}} 8$$

https://www.jeremyjordan.me/semantic-segmentation/

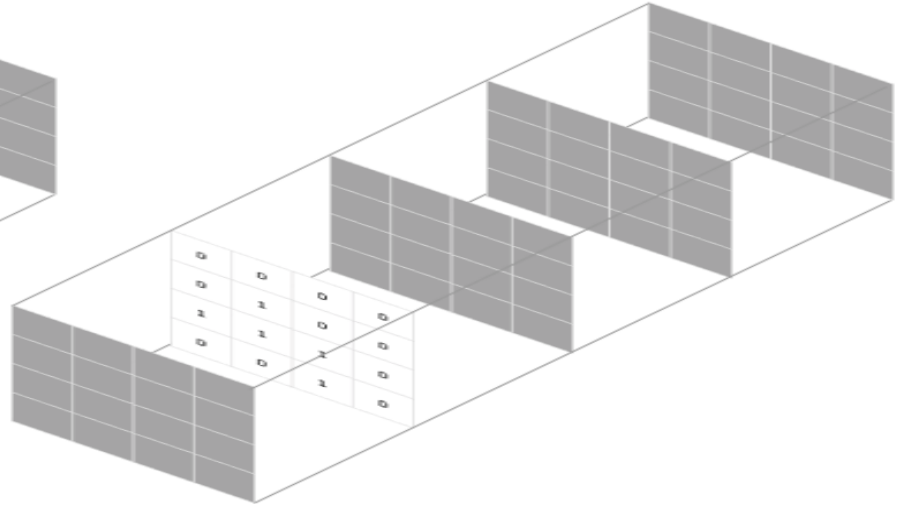
Loss Computation Segmentation DL models

Representing the task

In order to formulate a loss function which can be minimized, we'll simply use 1−Dice. This loss function is known as the soft Dice loss because we directly use the predicted probabilities instead of thresholding and converting them into a binary mask.



Prediction for a selected class



Target for the corresponding class

Soft Dice coefficient is calculated for each class mask

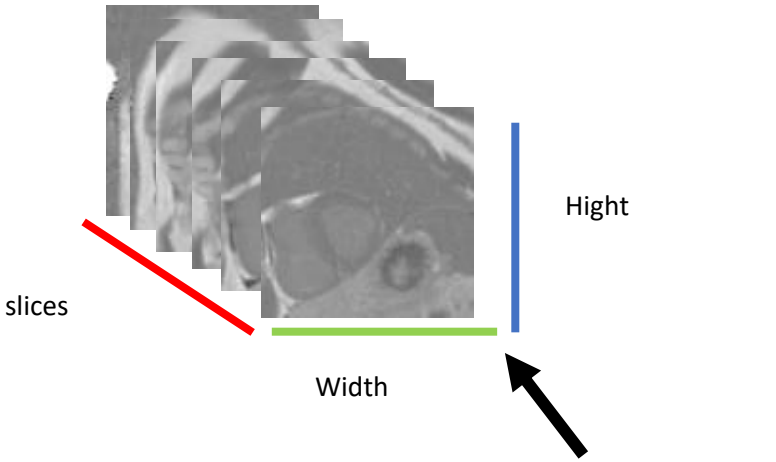
$$1 - \frac{2\sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2}$$

This scoring is repeated over all **classes** and averaged

https://www.jeremyjordan.me/semantic-segmentation/

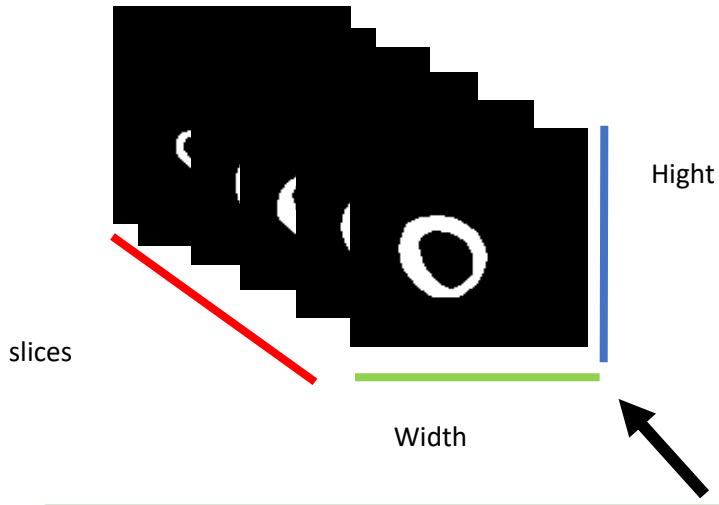
Dataset preprocessing & Training and Testing

Training and testing dataset for deep learning medical segmentation model:



Training data=(1800,224,224,1)

Samples Width Hight Channels



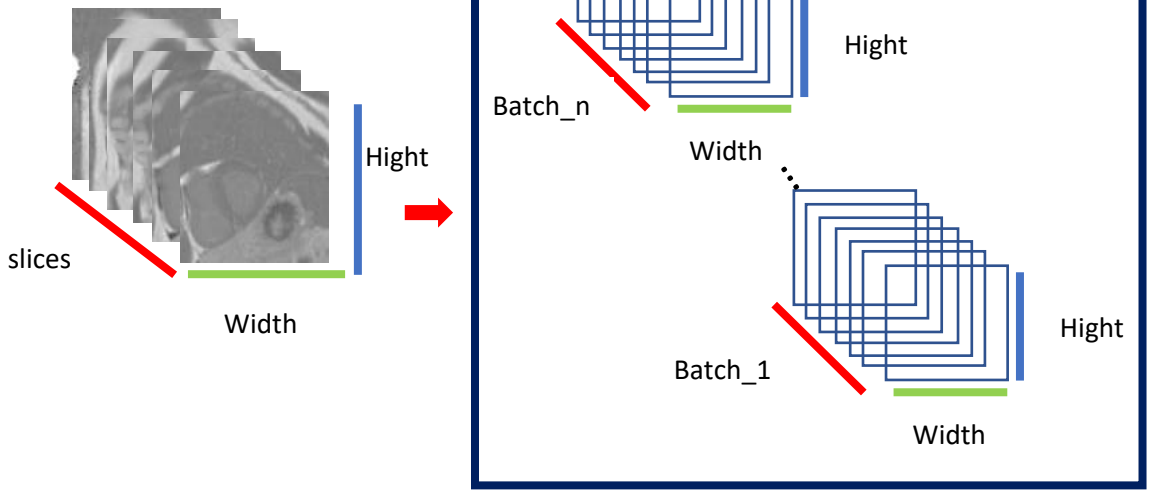
Training labels=(1800,224,224,1)

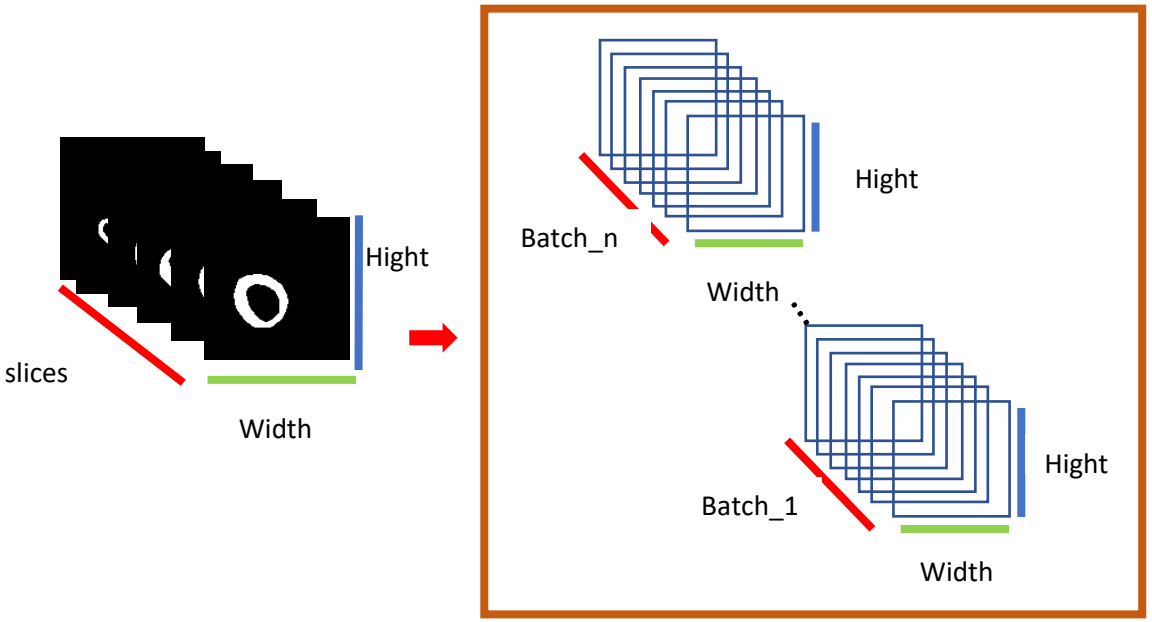
Samples Width Hight classes

Dataset preprocessing & Training and Testing

Dataset Tensor Formation for Deep Learning Models:

Training data generator provided the dataset into different number of batches for deep learning models





Training data=(10, 224, 224, 1)

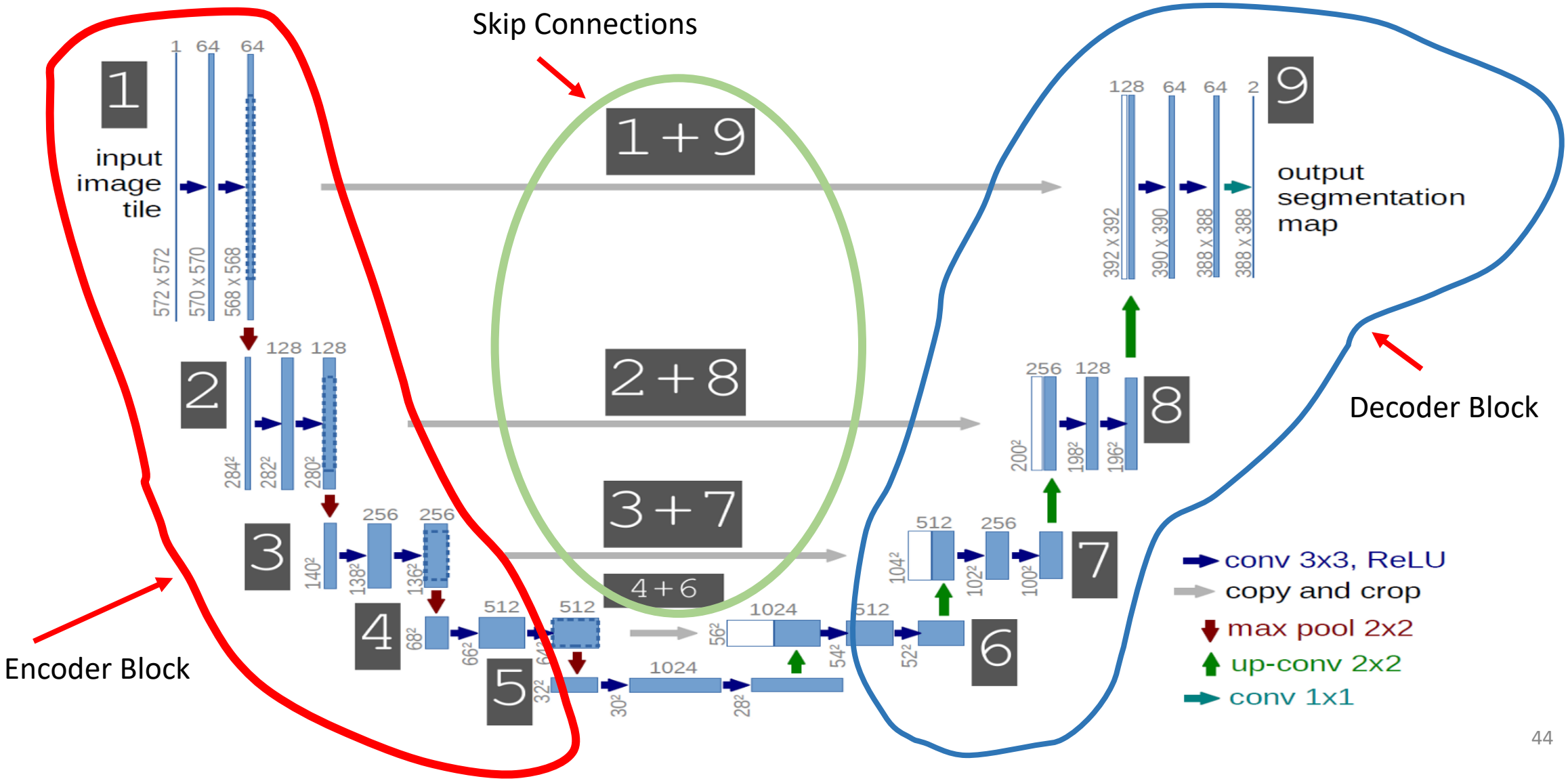
Batch_size | Width | Hight | Channels

Training Labels=(10, 224, 224, 1)

Batch_size | Width | Hight | Classes

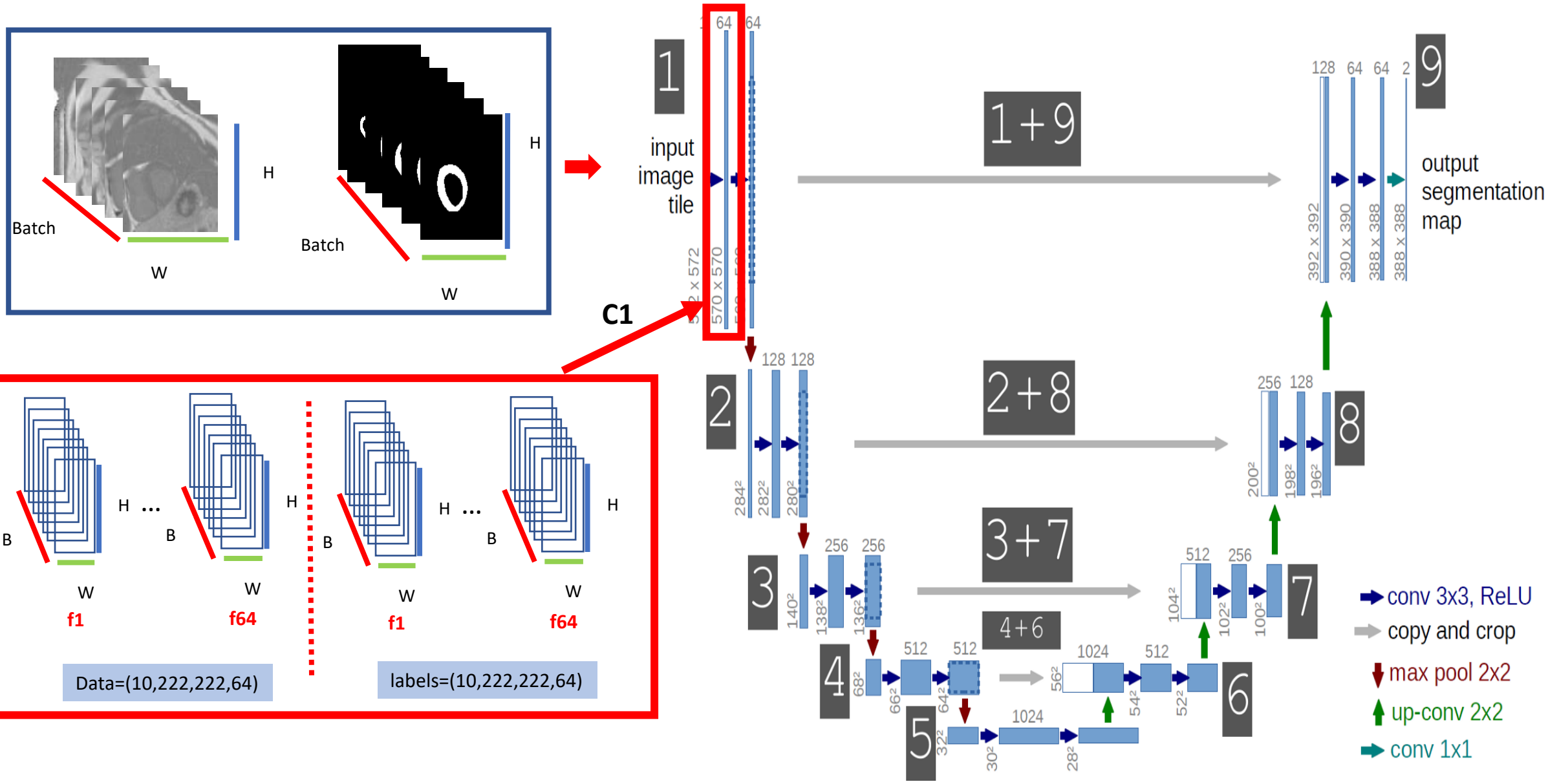
Unet Segmentation Deep Learning Model

State-of-the-art deep learning model used in medical image segmentation:



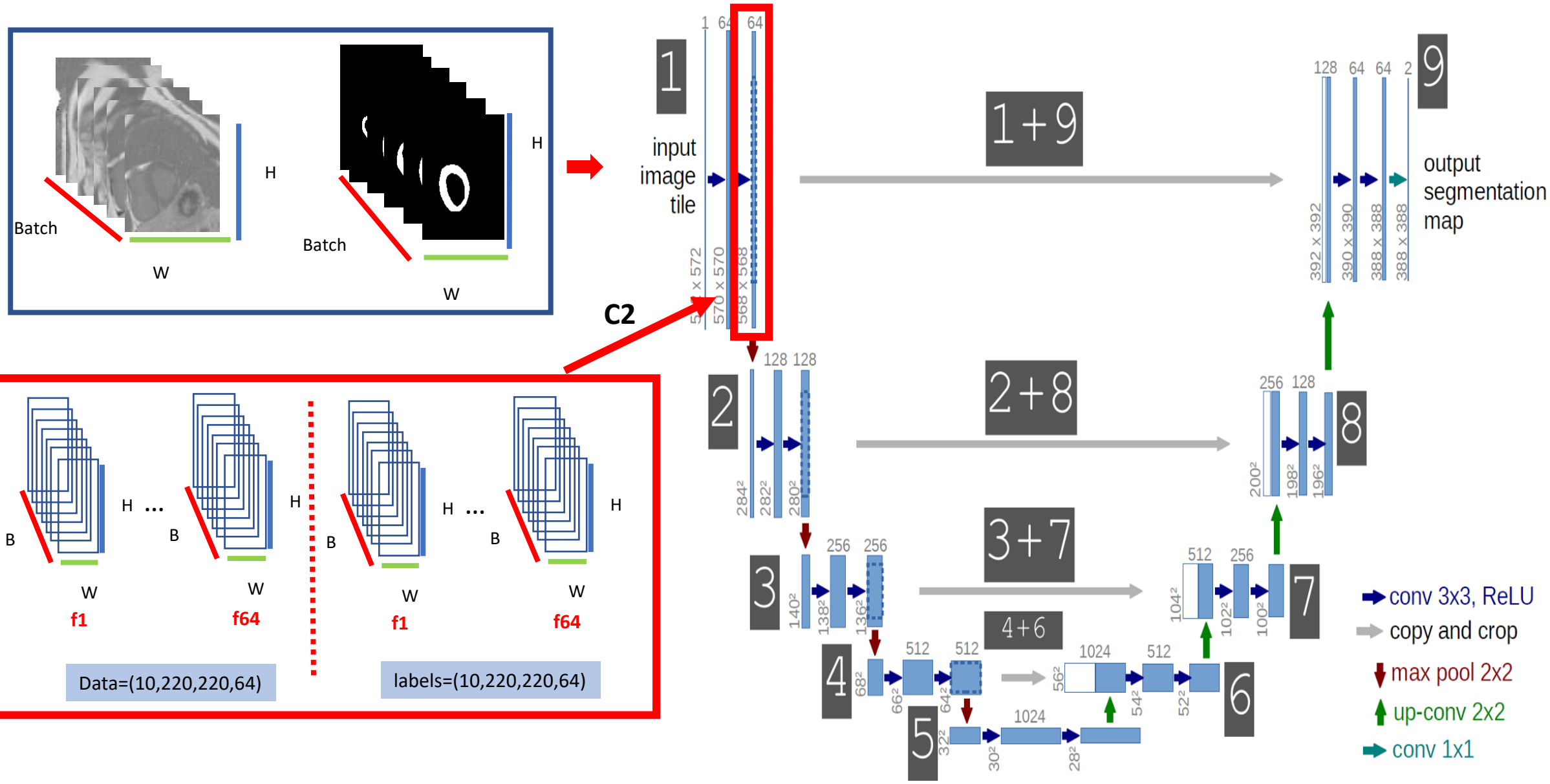
Unet Segmentation Deep Learning Model

State-of-the-art deep learning model used in medical image segmentation:



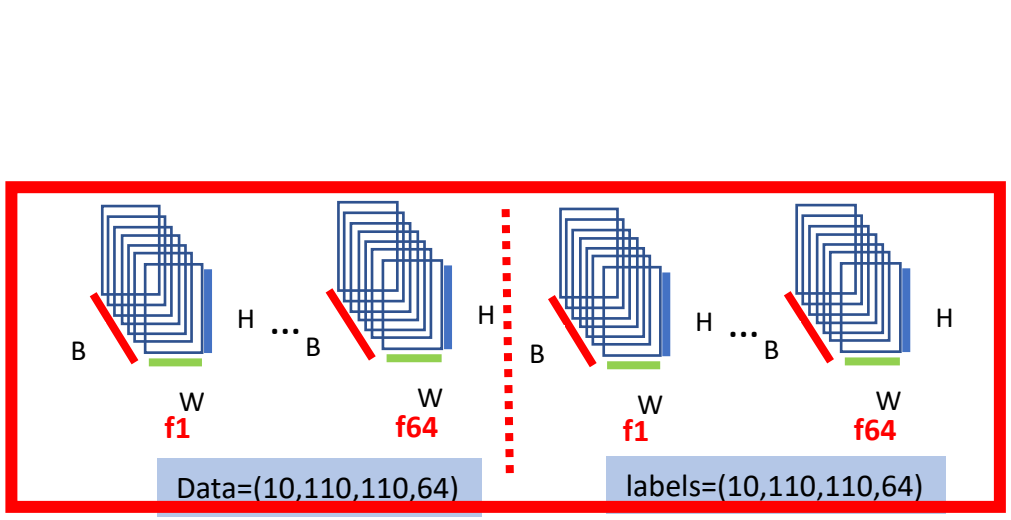
Unet Segmentation Deep Learning Model

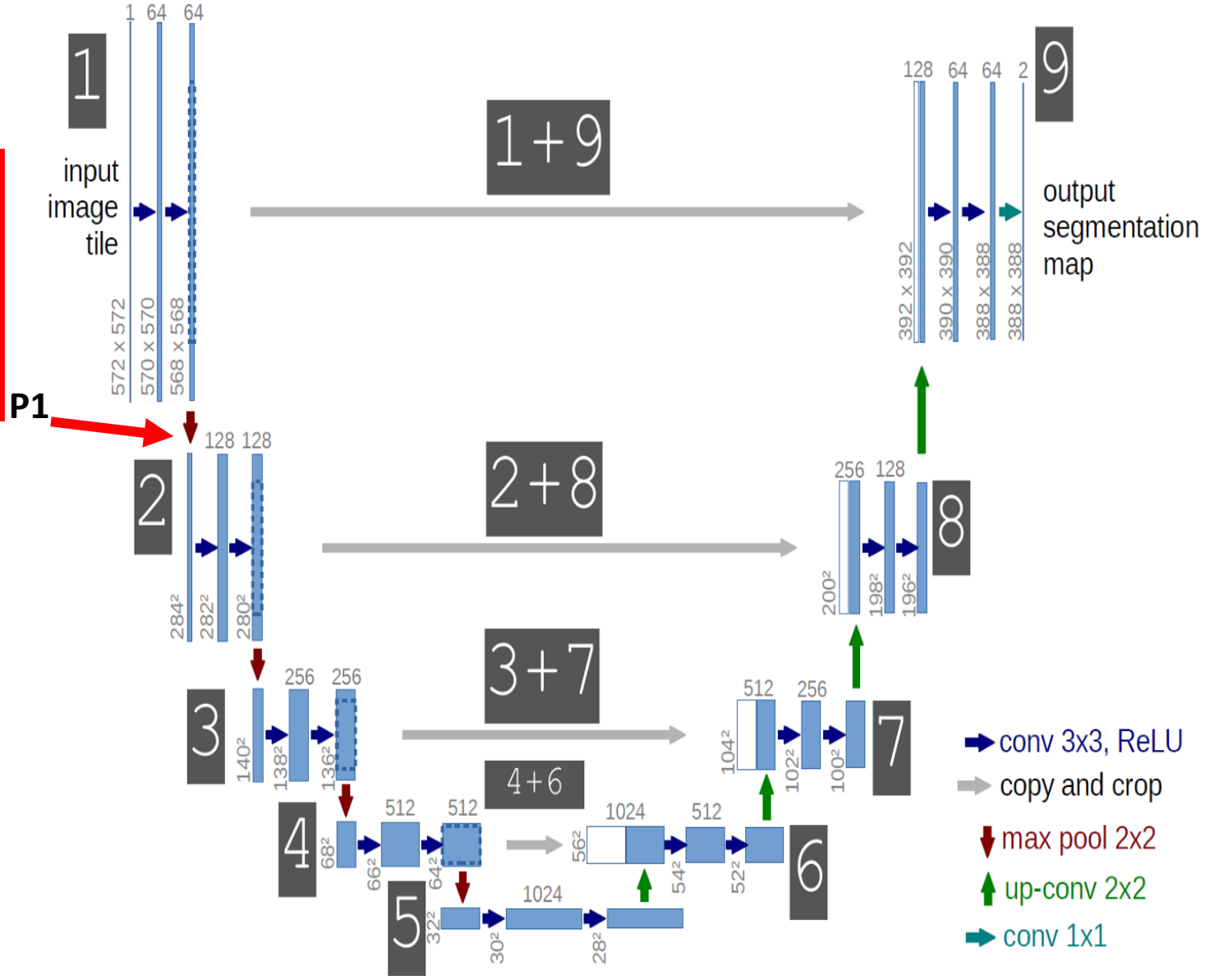
State-of-the-art deep learning model used in medical image segmentation:



Unet Segmentation Deep Learning Model

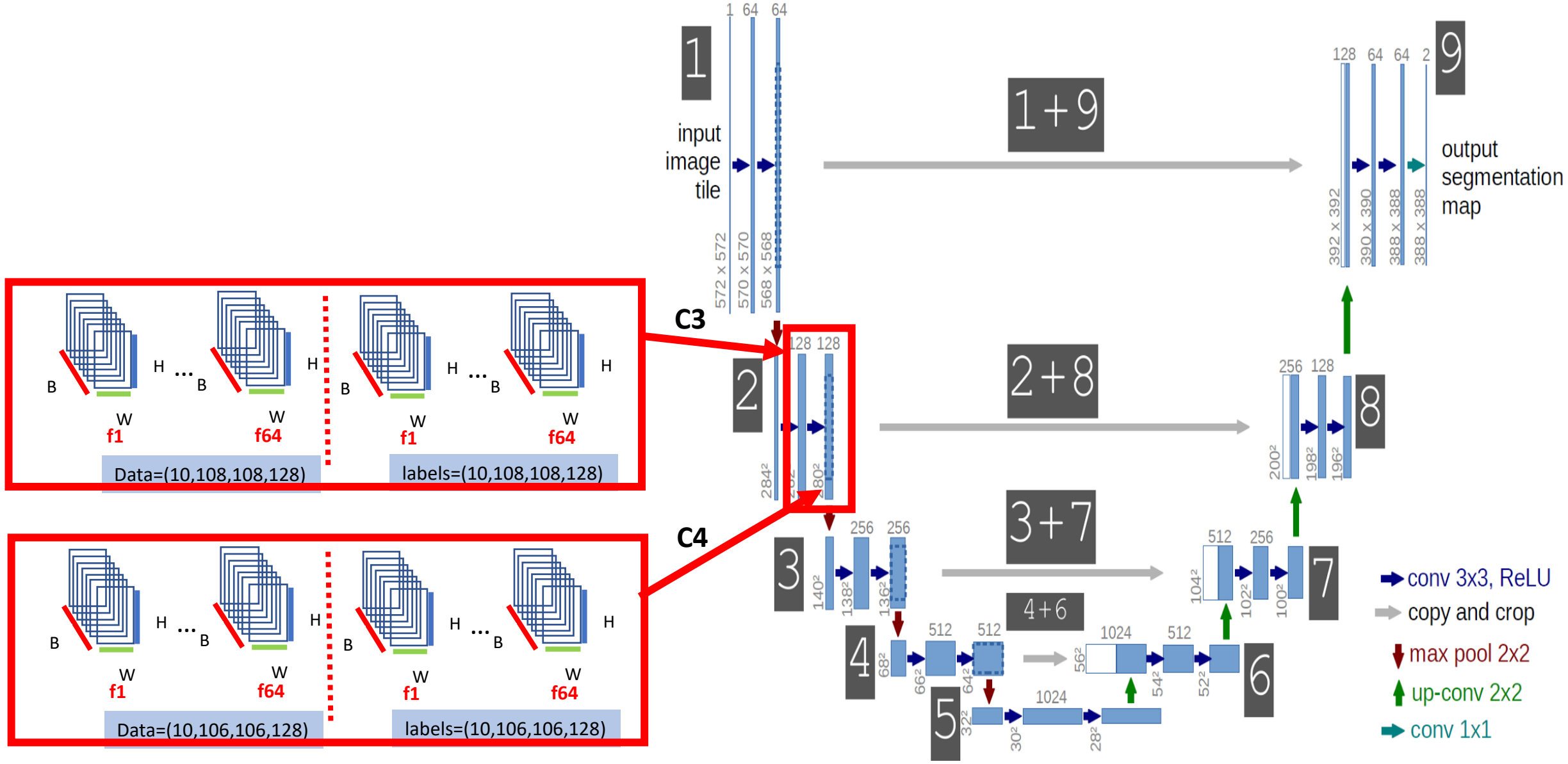
State-of-the-art deep learning model used in medical image segmentation:





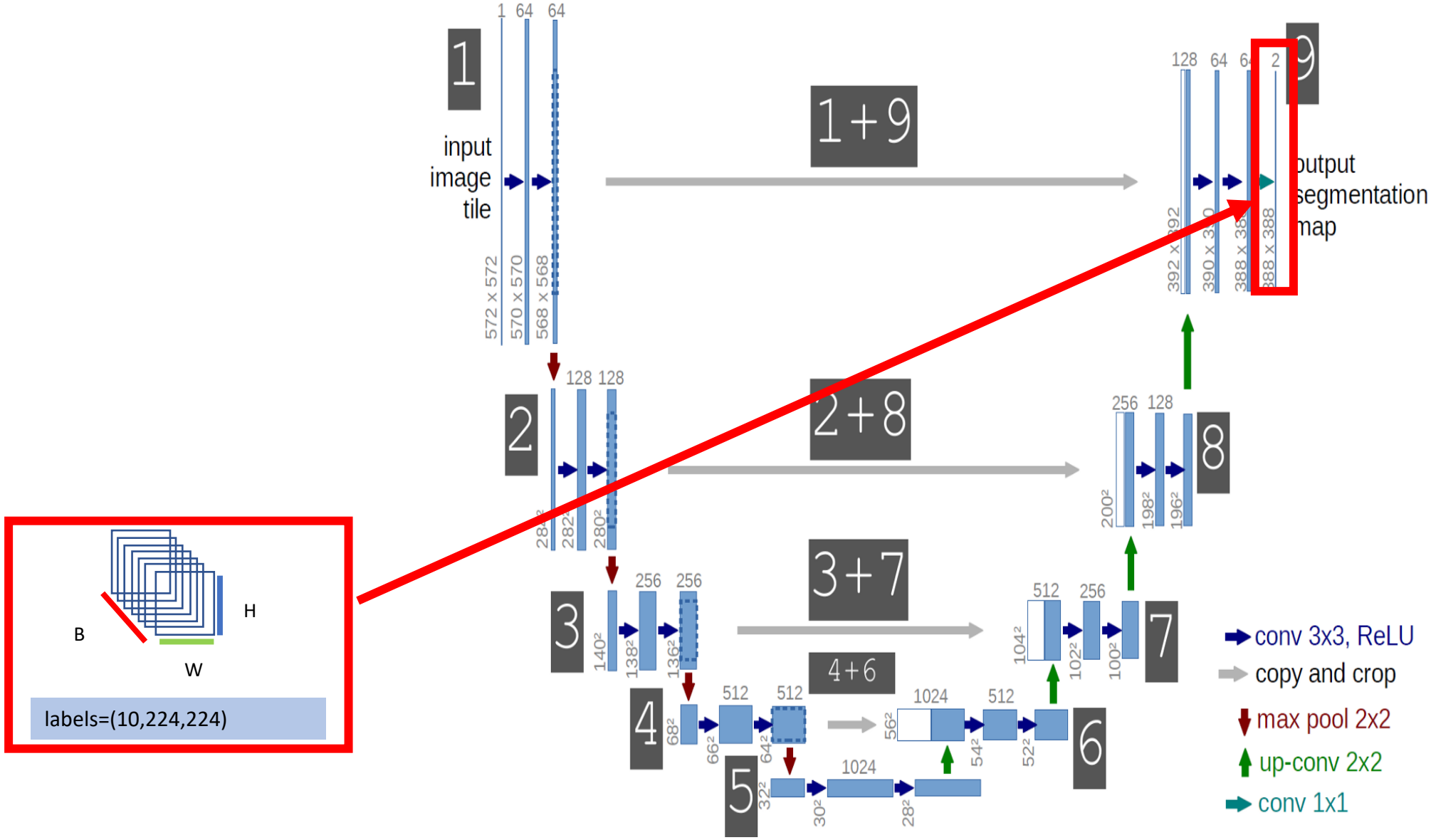
Unet Segmentation Deep Learning Model

State-of-the-art deep learning model used in medical image segmentation:



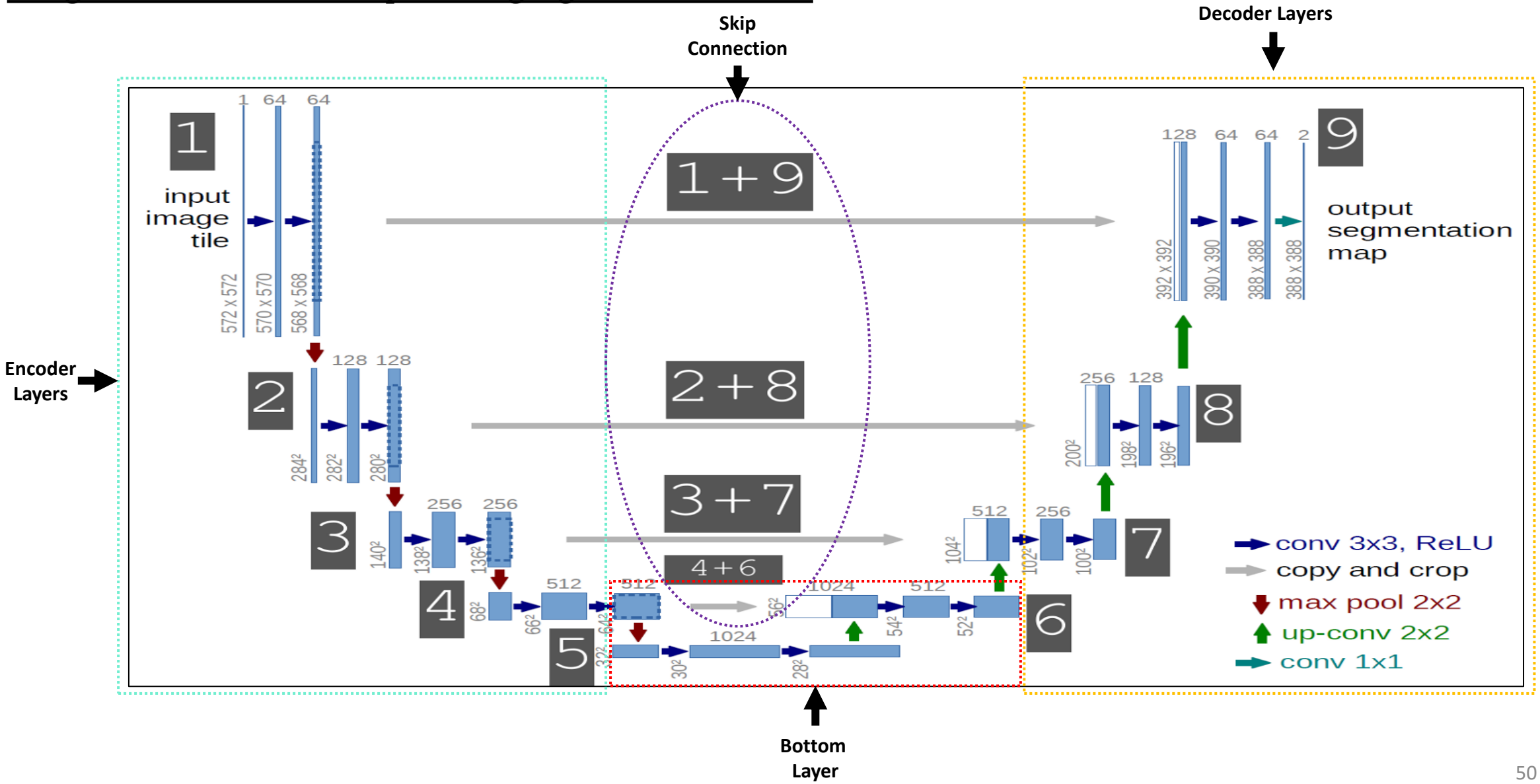
Unet Segmentation Deep Learning Model

State-of-the-art deep learning model used in medical image segmentation:

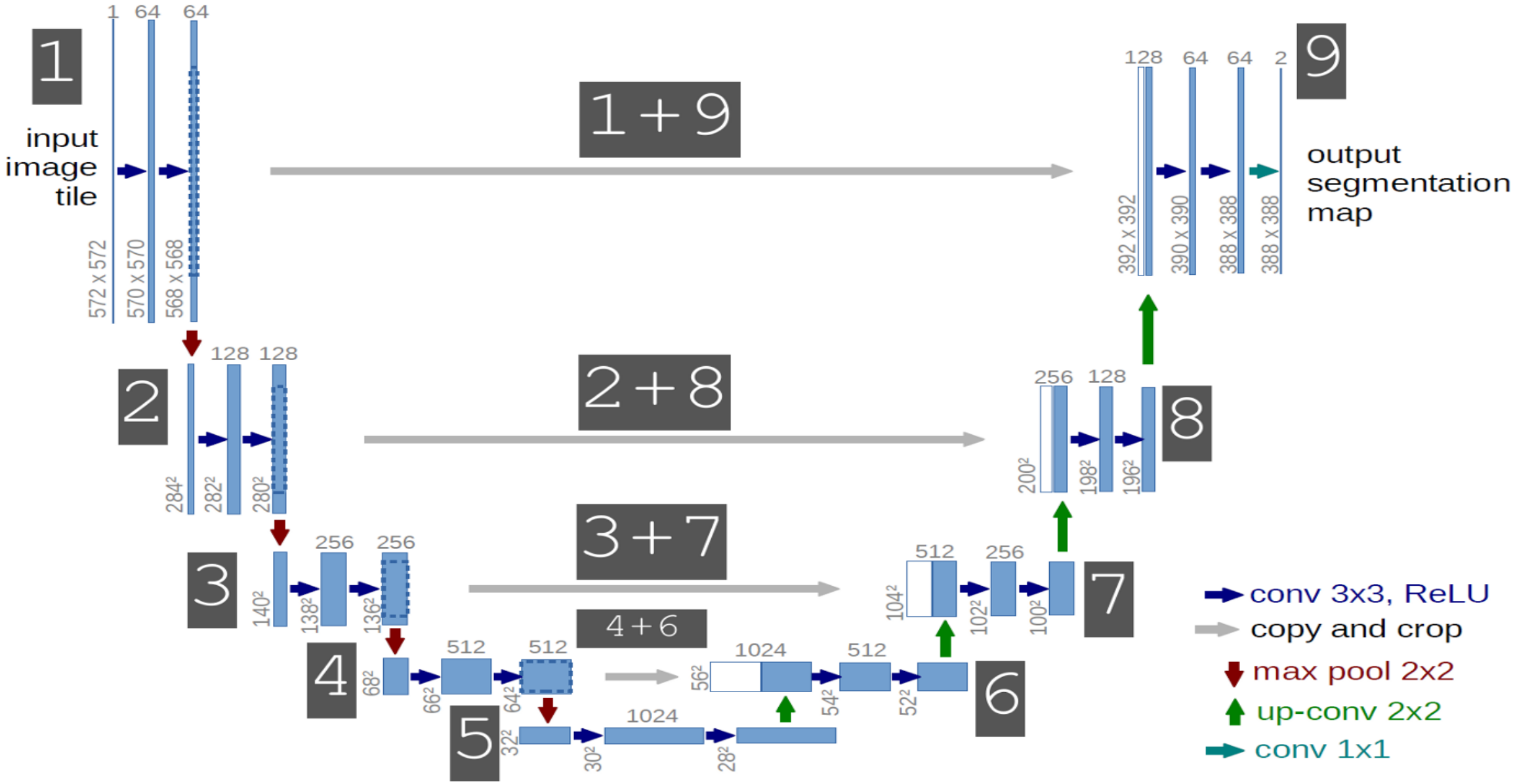


Unet Segmentation Deep Learning Model

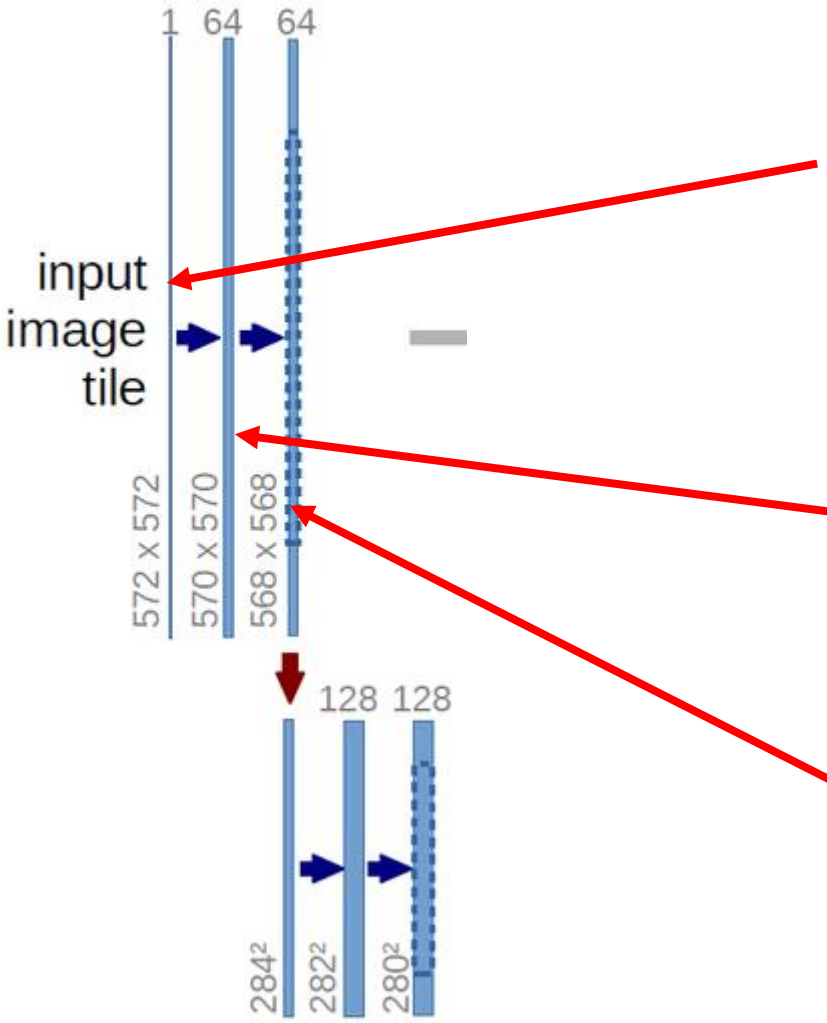
Design consideration in deep learning segmentation model:



Unet Model


1
input image tile
1 64 64
572 x 572
570 x 570
568 x 568
1+9
9
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
2
128 128
284²
282²
280²
2+8
8
256 128
200²
198²
196²
3
256 256
140²
138²
136²
3+7
7
512 256
104²
102²
100²
4
512 512
68²
66²
64²
4+6
6
1024 512
56²
54²
52²
5
1024
32²
30²
28²
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Unet Model

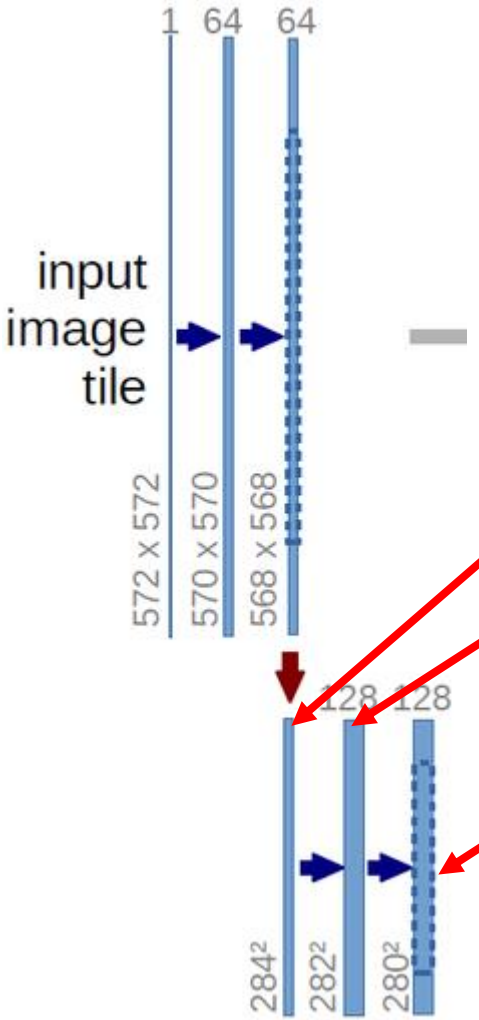



```
import keras
input=keras.Input((572,572,3)) # this layer is in cor
e layer of keras documentation
print(input.shape)
(None, 572, 572, 3)
```

```
x=input
C1= keras.layers.Conv2D(64, 3, activation='relu')(x)
print(C1.shape)
(None, 570, 570, 64)
```

```
C2= keras.layers.Conv2D(64, 3, activation='relu')(C1)
print(C2.shape)
(None, 568, 568, 64)
```

Unet Model

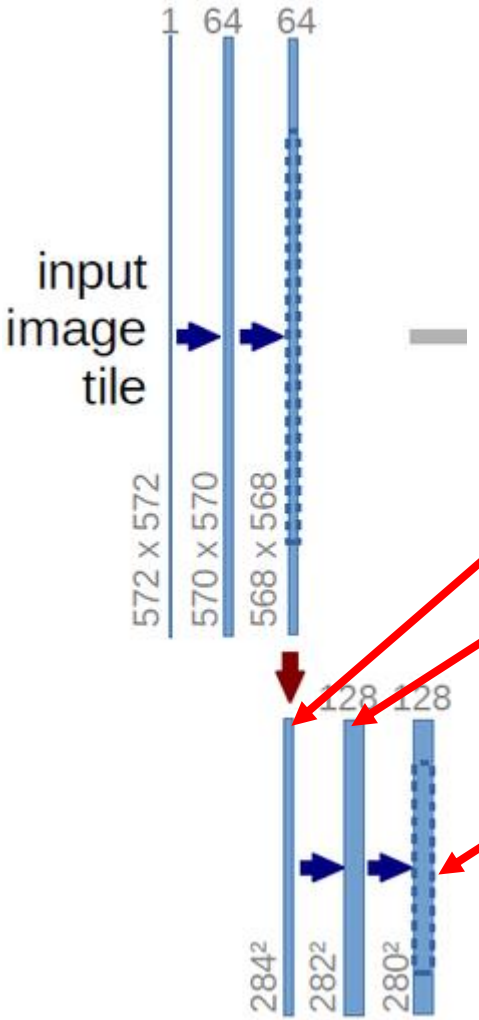



```
Pool1=keras.layers.MaxPool2D(pool_size=(2,2),
padding='valid')(C2)
print(Pool1.shape)
(None, 284, 284, 64)
```

```
P_C1=keras.layers.Conv2D(128,3,activation='relu')(Pool1)
print(P_C1.shape)
(None, 282, 282, 128)
```

```
P_C2=keras.layers.Conv2D(128,3,activation='relu')(P_C1)
print(P_C2.shape)
(None, 280, 280, 128)
```

Unet Model

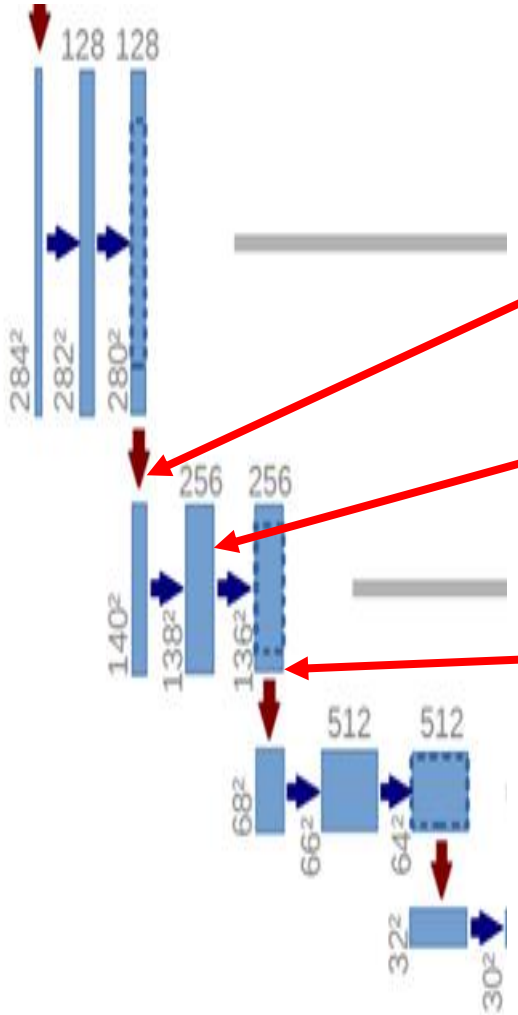



```
Pool1=keras.layers.MaxPool2D(pool_size=(2,2),
padding='valid')(C2)
print(Pool1.shape)
(None, 284, 284, 64)
```

```
P_C1=keras.layers.Conv2D(128,3,activation='relu')(Pool1)
print(P_C1.shape)
(None, 282, 282, 128)
```

```
P_C2=keras.layers.Conv2D(128,3,activation='relu')(P_C1)
print(P_C2.shape)
(None, 280, 280, 128)
```

Unet Model

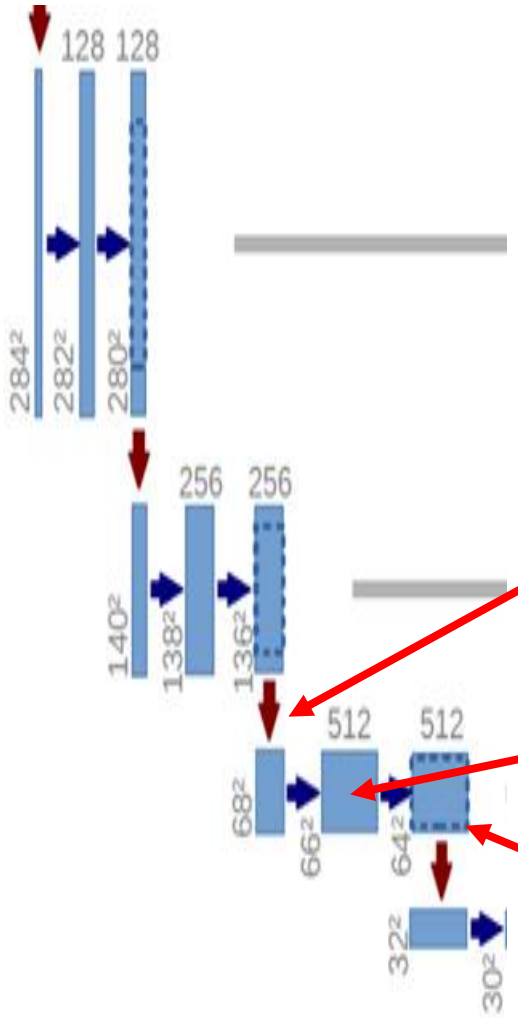



```
P2=keras.layers.MaxPool2D(pool_size=(2,2),pad
ding='valid')(P_C2)
print(P2.shape)
(None, 140, 140, 128)
```

```
P2_C1=keras.layers.Conv2D(256,3,activation='relu')(P2)
print(P2_C1.shape)
(None, 138, 138, 256)
```

```
P2_C2=keras.layers.Conv2D(256,3,activation='relu')(P2_C1)
print(P2_C2.shape)
(None, 136, 136, 256)
```

Unet Model

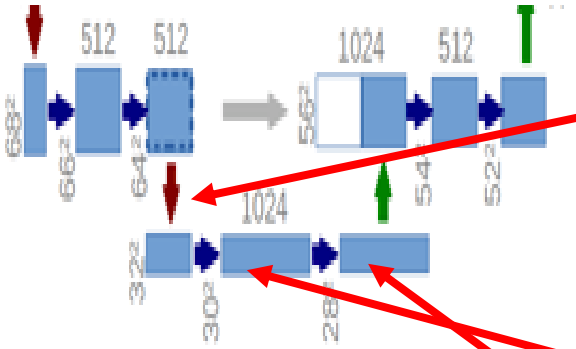



```
P3=keras.layers.MaxPool2D(pool_size=(2,2),pad
ding='valid')(P2_C2)
print(P3.shape)
(None, 68, 68, 256)
```

```
P3_C1=keras.layers.Conv2D(512,3,activation='relu')(P3)
print(P3_C1.shape)
(None, 66, 66, 512)
```

```
P3_C2=keras.layers.Conv2D(512,3,activation='relu')(P3_C1)
print(P3_C2.shape)
(None, 64, 64, 512)
```

Unet Model

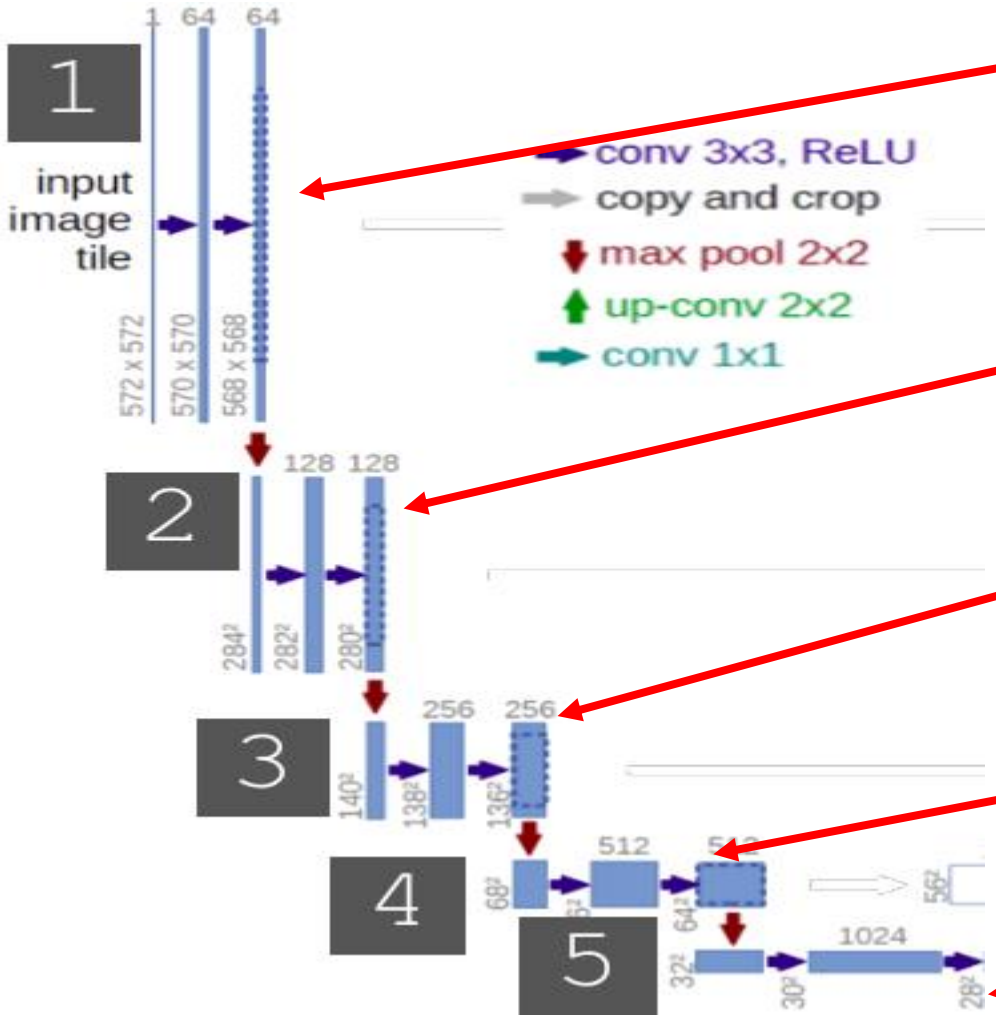



```
P4=keras.layers.MaxPool2D(pool_size=(2,2),pad
ding='valid')(P3_C2)
print(P4.shape)
(None, 32, 32, 512)
```

```
P4_C1=keras.layers.Conv2D(1024,3, activation=
'relu')(P4)
print(P4_C1.shape)
(None, 30, 30, 1024)
```

```
P4_C2=keras.layers.Conv2D(1024,3, activation=
'relu')(P4_C1)
print(P4_C2.shape)
(None, 28, 28, 1024)
```

Unet Model



Input=(572x572x3)
C1=(570x570x64)
C2=(568x568x64)

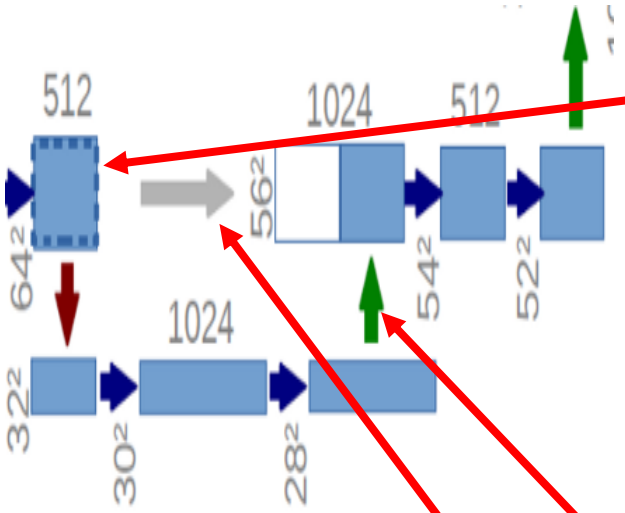
Pool1=(284x284x64)
P_C1=(282x282x128)
P_C2=(280x280x128)

P2=(140x140x128)
P2_C1=(138x138x256)
P2_C2=(136x136x256)

P3=(68x68x256)
P3_C1=(66x66x512)
P3_C2=(64x64x512)

P4=(32x32x512)
P4_C1=(30x30x1024)
P4_C2=(28x28x1024)

Unet Model



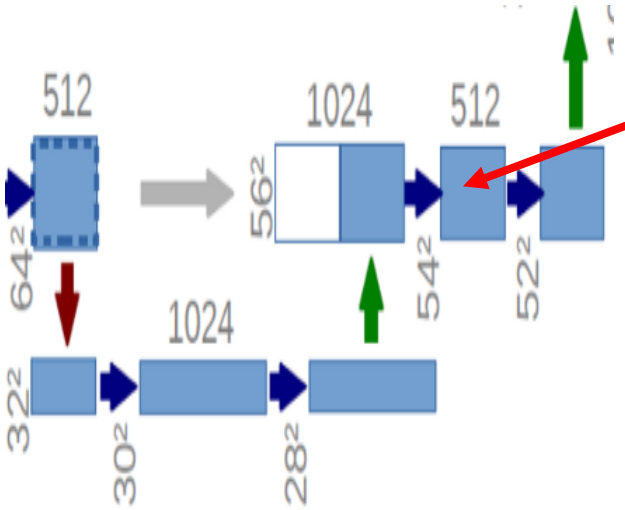

```
P3_C2=keras.layers.Conv2D(512,3,activation='relu')(P3_C1)
print(P3_C2.shape)
(None, 64, 64, 512)
```

```
ch, cw = get_crop_shape(P3_C2, Up_1)
crop_P3_C2 = tf.keras.layers.Cropping2D(cropping=(ch,cw))(
P3_C2)
print(crop_P3_C2.shape)
(None, 56, 56, 512)
```

```
Up_1 = tf.keras.layers.Conv2DTranspose(512, (2, 2), stride
s=(2, 2), padding='same') (P4_C2)
print(up_1.shape)
(None, 56, 56, 512)
```

```
ConCat1=tf.keras.layers.Concatenate()([up_1,crop_P3_C2])
print(ConCat1.shape)
(None, 56, 56, 1024)
```

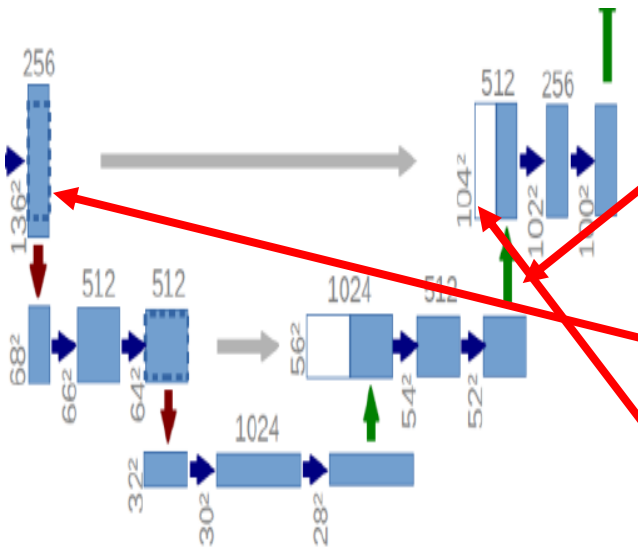
Unet Model




```
Up1_C1=keras.layers.Conv2D(512,3,activation='relu')(ConCa
t1)
print(Up1_C1.shape)
(None, 54, 54, 512)
```

```
Up1_C2=keras.layers.Conv2D(512,3,activation='relu')(Up1_C
1)
print(Up1_C2.shape)
(None, 52, 52, 512)
```

Unet Model

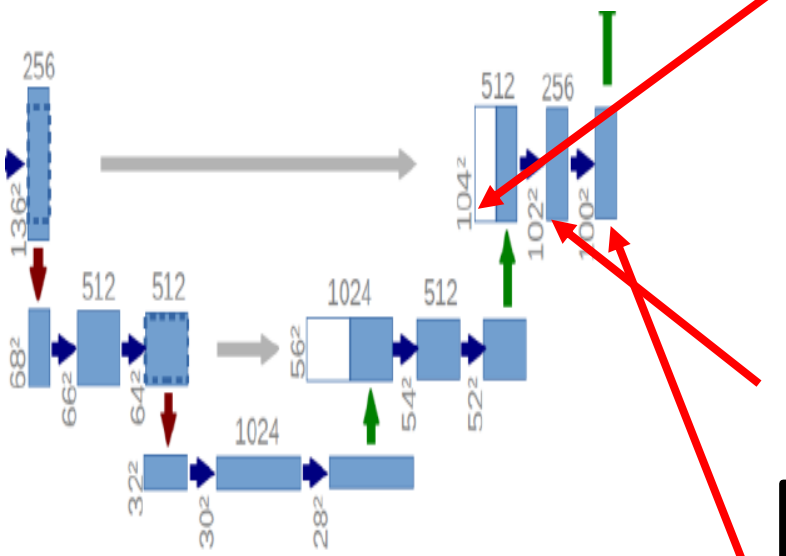



```
up_2=tf.keras.layers.Conv2DTranspose(256,(2,2),strides=(2
,2),padding='same')(Up1_C2)
print(up_2.shape)
(None, 104, 104, 256)
```

```
ch, cw = get_crop_shape(P2_C2, up_2)
crop_P2_C2 = tf.keras.layers.Cropping2D(cropping=(ch,cw))
(P2_C2)
print(crop_P2_C2.shape)
(None, 104, 104, 256)
```

```
ConCat2=tf.keras.layers.Concatenate()([up_2,crop_P2_C2])
print(ConCat2.shape)
(None, 104, 104, 512)
```

Unet Model

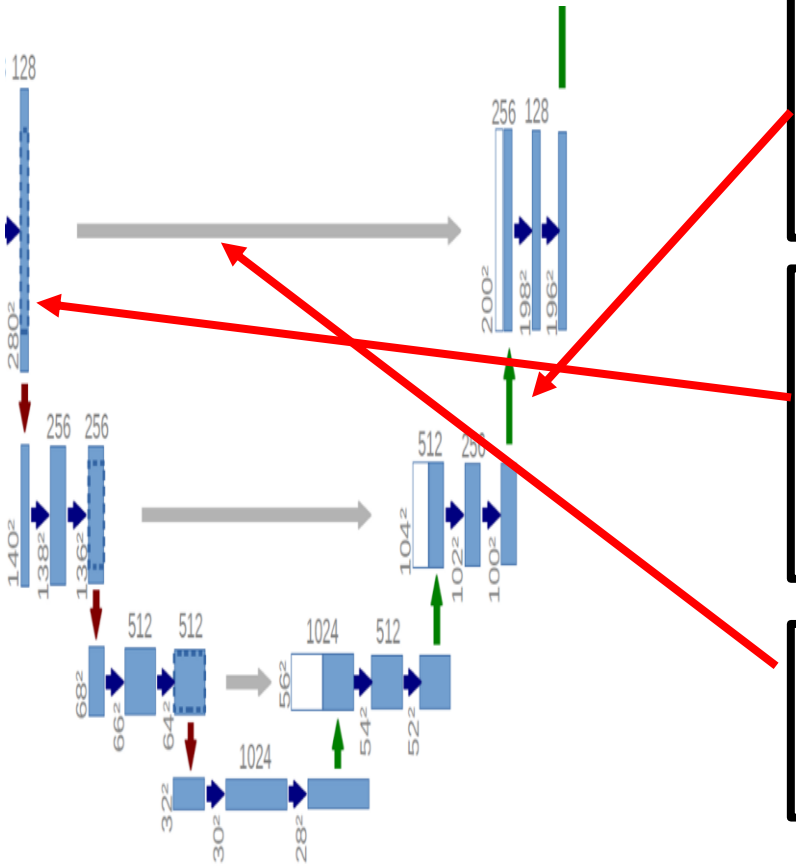



```
ConCat2=tf.keras.layers.Concatenate()([up_2,crop_P2_C2])
print(ConCat2.shape)
(None, 104, 104, 512)
```

```
Up2_c1=tf.keras.layers.Conv2D(256,3, activation='relu')(C
onCat2)
print(Up2_c1.shape)
(None, 102, 102, 256)
```

```
Up2_c2=tf.keras.layers.Conv2D(256,3, activation='relu')(Up
2_c1)
print(Up2_c2.shape)
(None, 100, 100, 256)
```

Unet Model

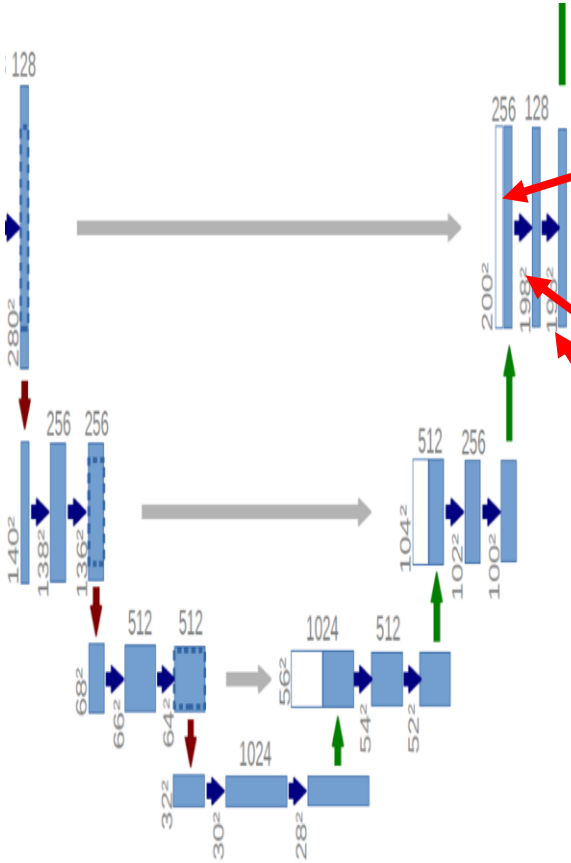



```
up_3=tf.keras.layers.Conv2DTranspose(128,kernel_size=(2,2),strides=(2,2),padding='same')(Up2_c2)
print(up_3.shape)
(None, 200, 200, 128)
```

```
ch, cw = get_crop_shape(P_C2, up_3)
crop_P_C2 = tf.keras.layers.Cropping2D(cropping=(ch,cw))(P_C2)
print(crop_P_C2.shape)
(None, 200, 200, 128)
```

```
ConCat3=tf.keras.layers.Concatenate()([up_3,crop_P_C2])
print(ConCat3.shape)
(None, 200, 200, 256)
```

Unet Model

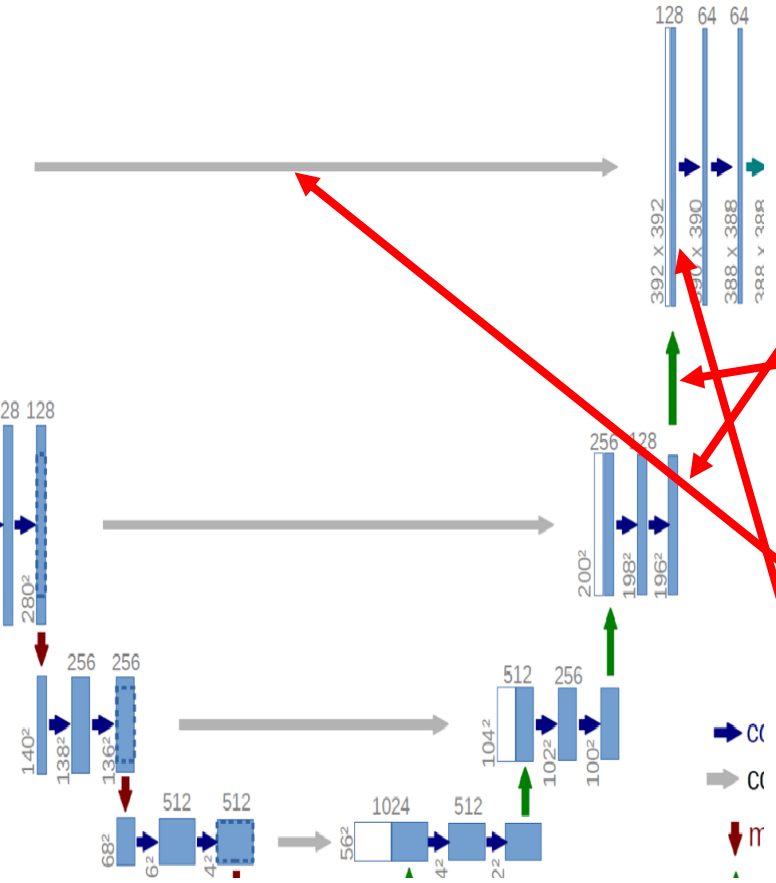



```
ConCat3=tf.keras.layers.Concatenate()([up_3,crop_P_C2])
print(ConCat3.shape)
(None, 200, 200, 256)
```

```
Up3_c1=tf.keras.layers.Conv2D(128,3,activation='relu')(ConC
at3)
print(Up3_c1.shape)
(None, 198, 198, 128)
```

```
Up3_c2=tf.keras.layers.Conv2D(128,3, activation='relu')(Up3
_c1)
print(Up3_c2.shape)
(None, 196, 196, 128)
```

Unet Model



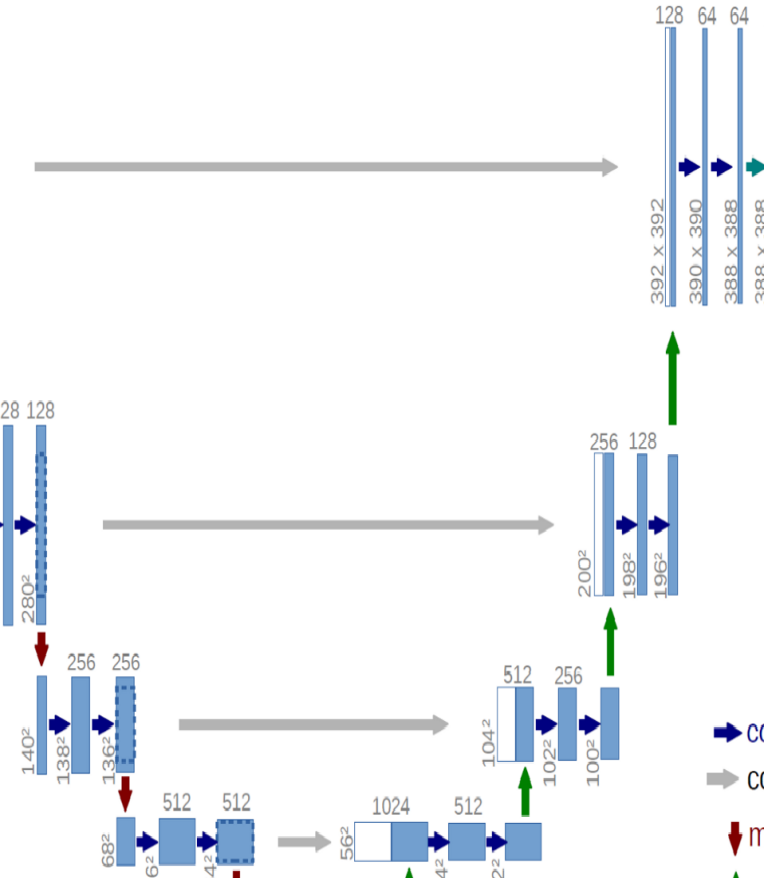

```
Up3_c2=tf.keras.layers.Conv2D(128,3, activation='relu')(Up3
_c1)
print(Up3_c2.shape)
(None, 196, 196, 128)
```

```
up_4=tf.keras.layers.Conv2DTranspose(64,kernel_size=(2,2
),strides=(2,2),padding='same')(Up3_c2)
print(up_4.shape)
```

```
ch, cw = get_crop_shape(C2, up_4)
crop_C2 = tf.keras.layers.Cropping2D(cropping=(ch,cw))(C
2)
print(crop_C2.shape)
(None, 392, 392, 64)
```

```
ConCat4=tf.keras.layers.Concatenate()([crop_C2,up_4])
print(ConCat4.shape)
(None, 392, 392, 128)
```

Unet Model

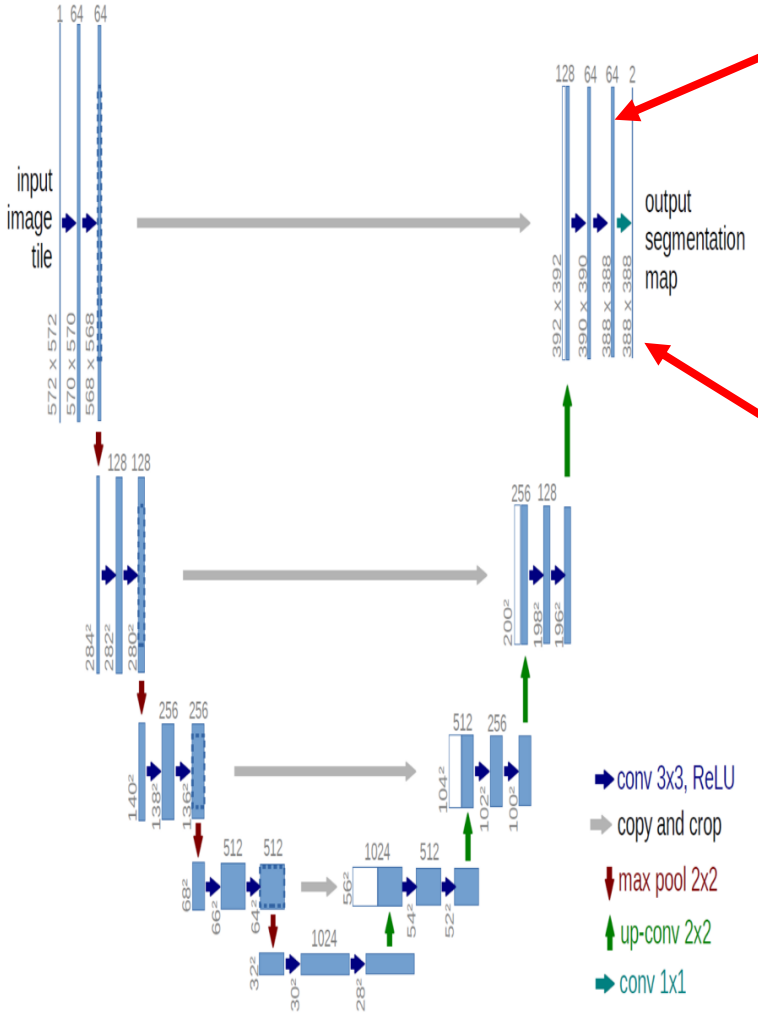



```
ConCat4=tf.keras.layers.Concatenate()([crop_C2,up_4])
print(ConCat4.shape)
(None, 392, 392, 128)
```

```
Up4_c1=tf.keras.layers.Conv2D(64,3,activation='relu')(C
onCat4)
print(Up4_c1.shape)
(None, 390, 390, 64)
```

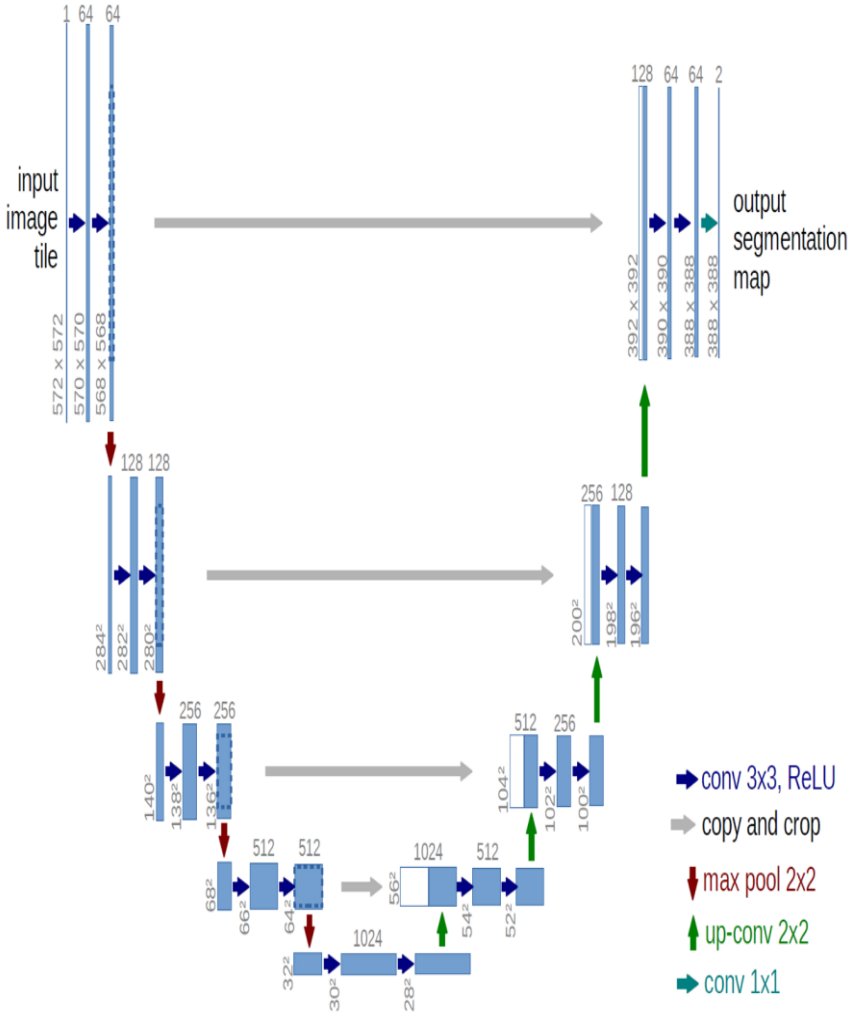
```
Up4_c2=tf.keras.layers.Conv2D(64,3,activation='relu')(Up
4_c1)
print(Up4_c2.shape)
(None, 388, 388, 64)
```

Unet Model




```
Up4_c2=tf.keras.layers.Conv2D(64,3,activation='relu')(Up
4_c1)
print(Up4_c2.shape)
(None, 388, 388, 64)
```

```
c_1x1=tf.keras.layers.Conv2D(1,1,activation='softmax',pa
dding='same')(Up4_c2)
print(c_1x1.shape)
(None, 388, 388, 1)
```



C1x1=(388x388x1)

Input=(572x572x3)
C1=(570x570x64)
C2=(568x568x64)

Pool1=(284x284x64)
P_C1=(282x282x128)
P_C2=(280x280x128)

P2=(140x140x128)
P2_C1=(138x138x256)
P2_C2=(136x136x256)

P3=(68x68x256)
P3_C1=(66x66x512)
P3_C2=(64x64x512)

P4=(32x32x512)
P4_C1=(30x30x1024)
P4_C2=(28x28x1024)

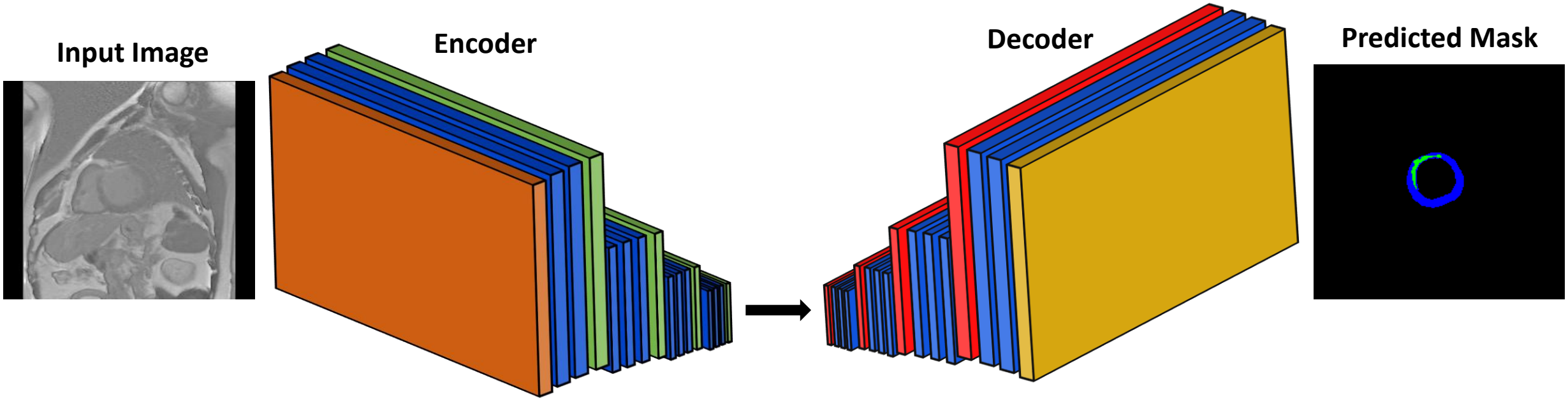
Up4_c2=(388x388x64)
Up4_c1=(390x390x64)
ConcatC4=(392x392x128)
up4=(392x392x64)

Up3_c2=(196x196x128)
Up3_c1=(198x198x128)
ConcatC3=(200x200x256)
up3=(200x200x128)

Up2_c2=(100x100x256)
Up2_c1=(102x102x256)
ConcatC2=(104x104x512)
up2=(104x104x256)

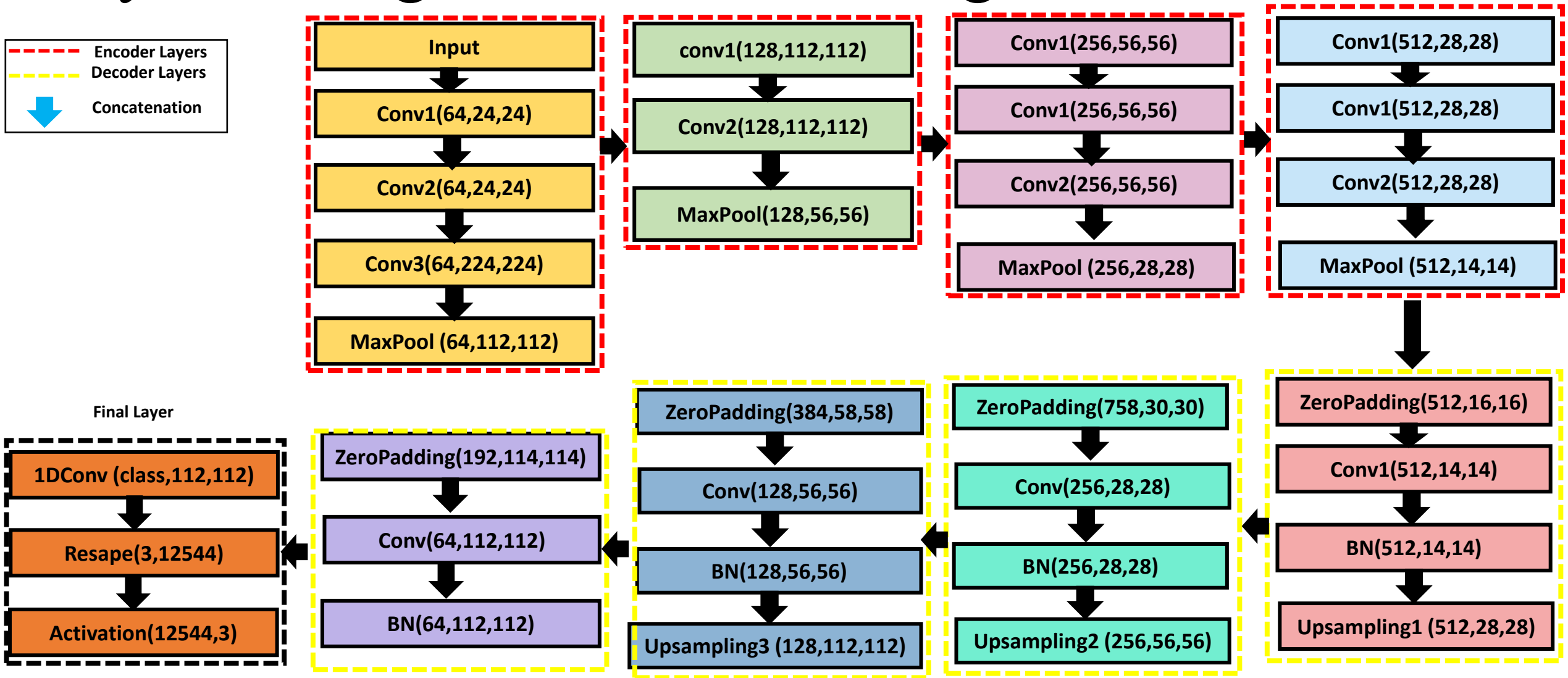
Up1_c2=(52x52x512)
Up1_c1=(54x54x512)
ConcatC1=(56x56x1024)
up1=(56x56x512)
P4_C2=(28x28x1024)

VGG-Segnet for Multiclass Segmentation

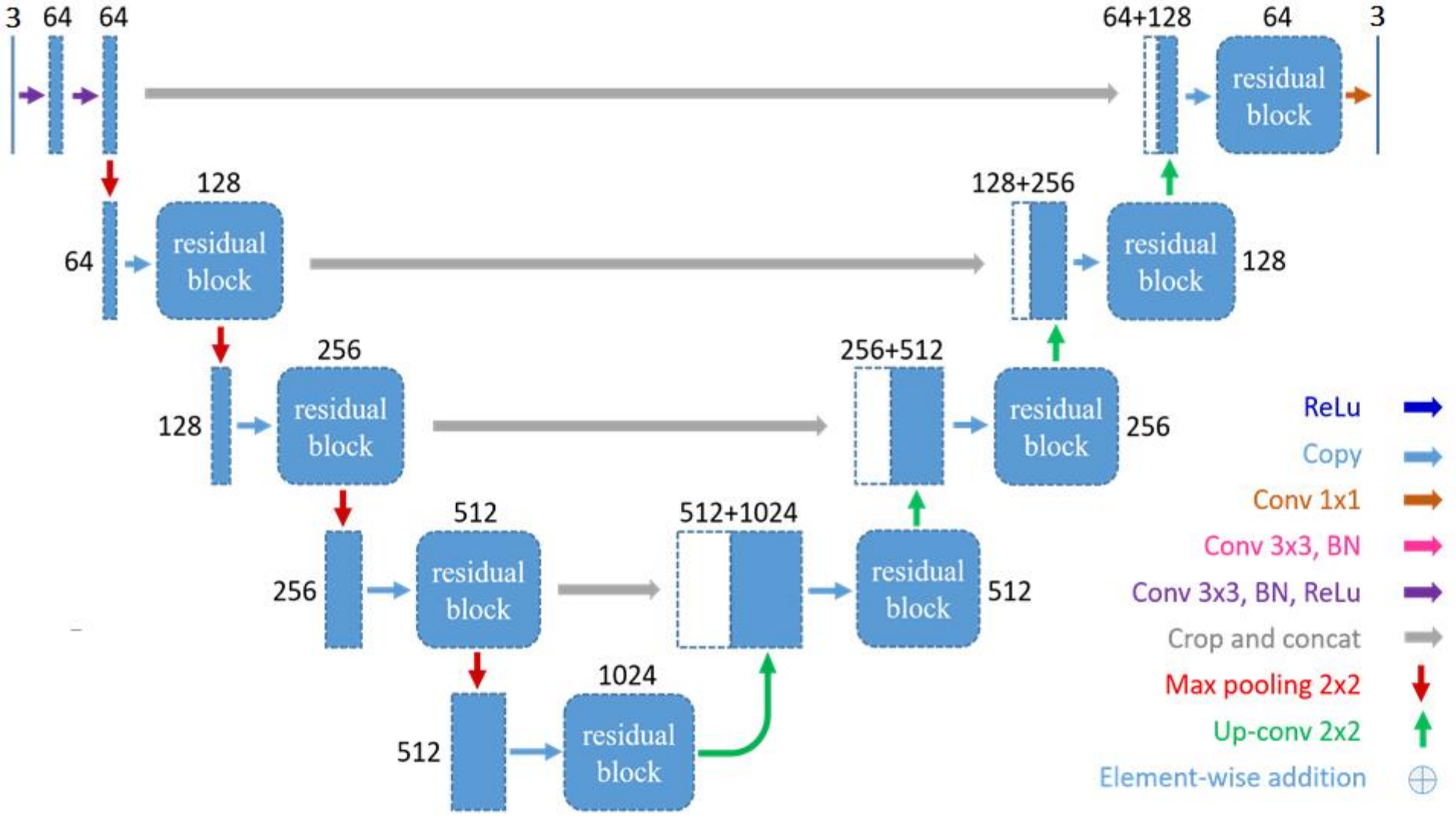


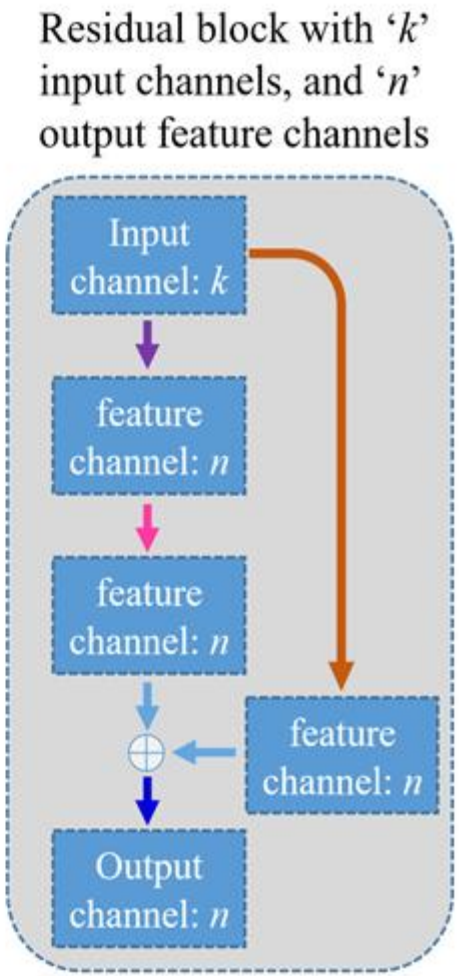
Normalization
Upsampling
Convolutional
Softmax
MaxPooling

Layer Configuration of VGGSegnet



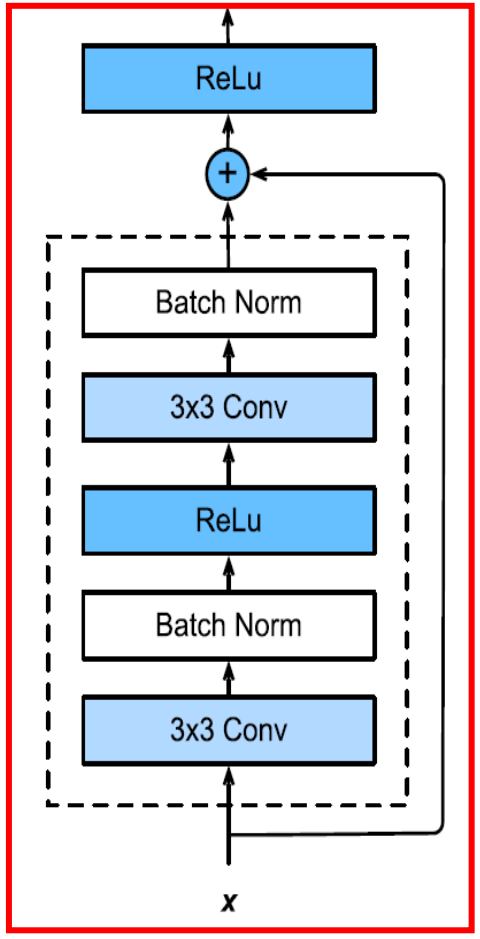
ResNet50 like Unet for Multiclass Segmentation



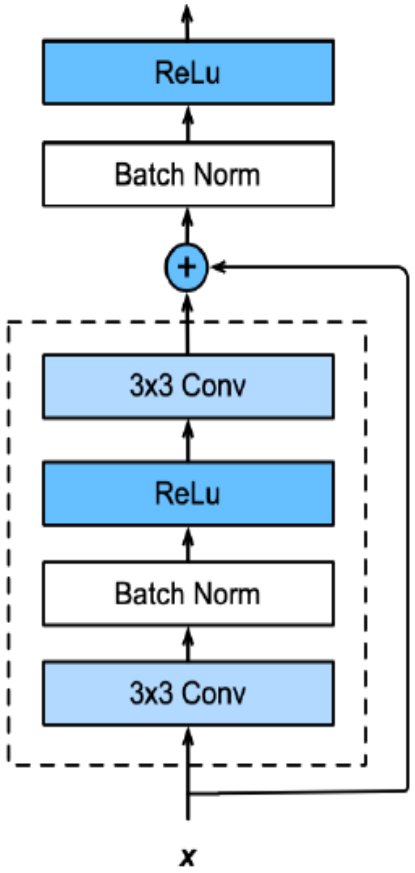


Design configuration in ResNet

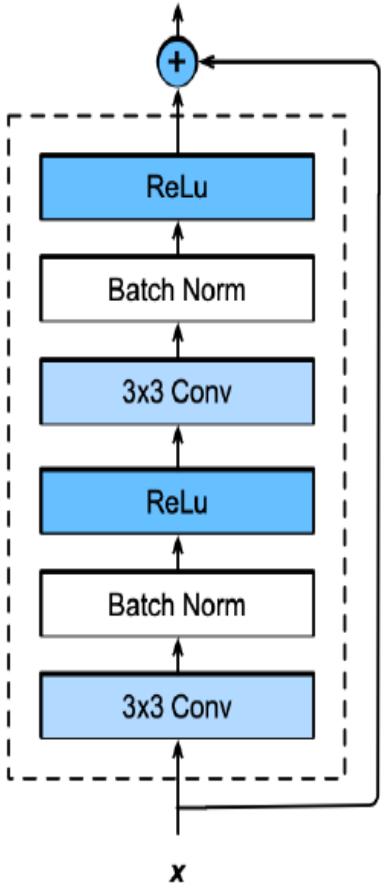
We used original configuration in our design



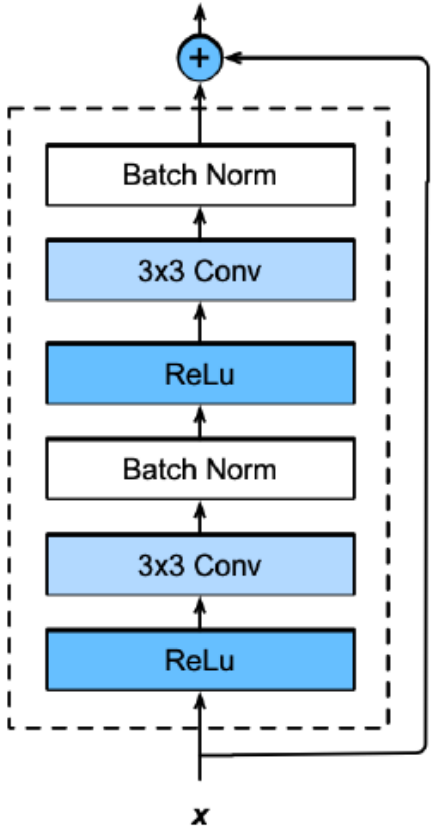
Original



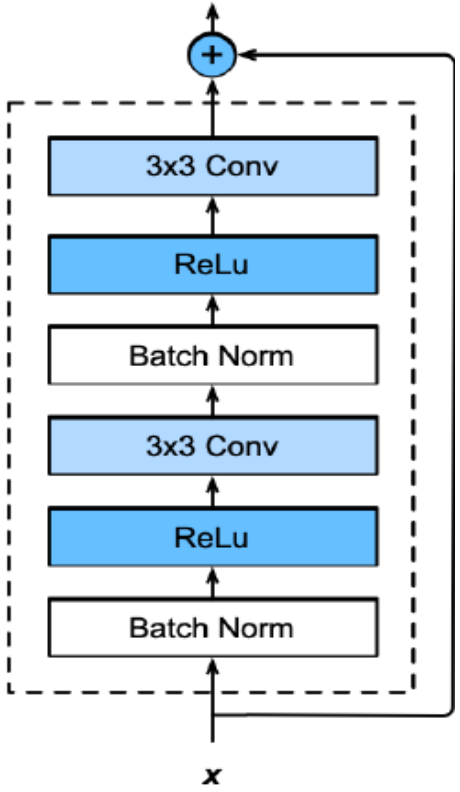
Batch norm after addition



ReLu last



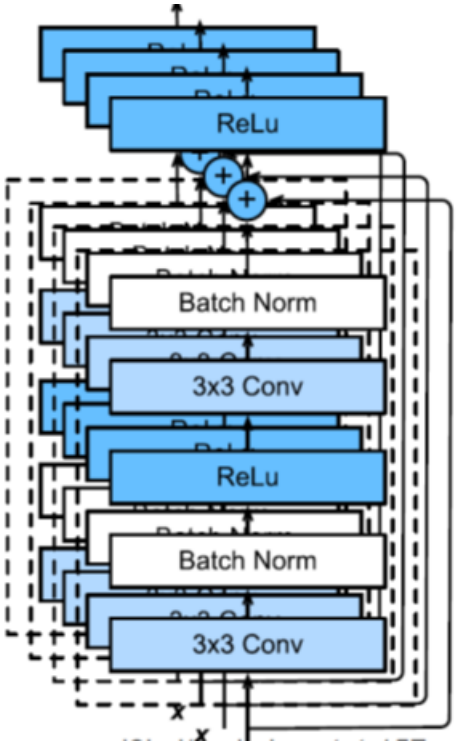
Batch Norm last



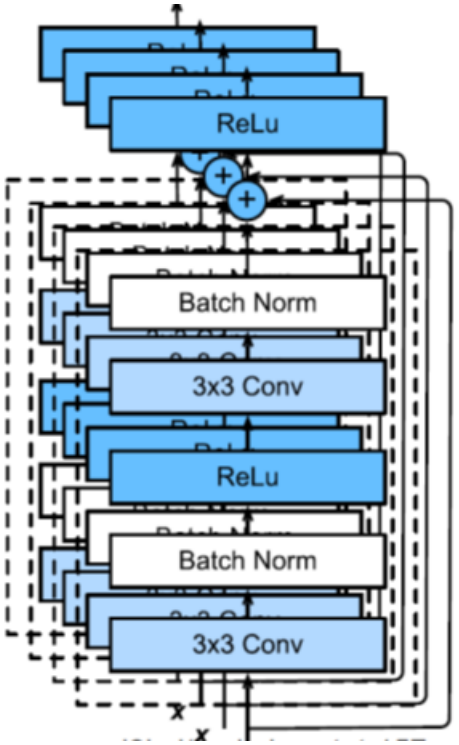
Pre activation

Residual Block Mapping

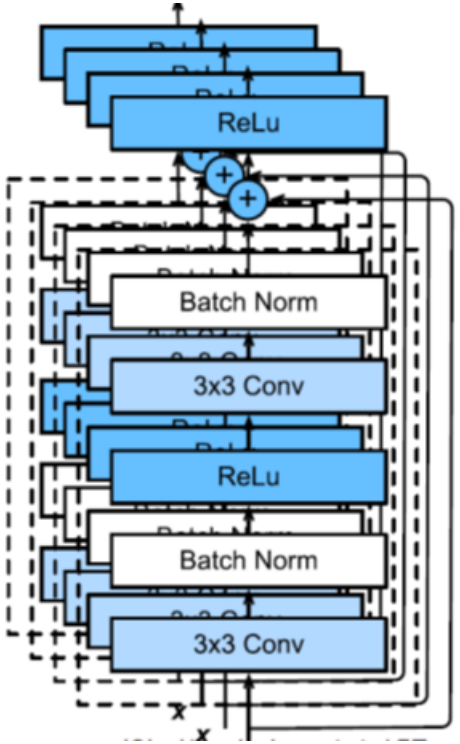
2-residual block



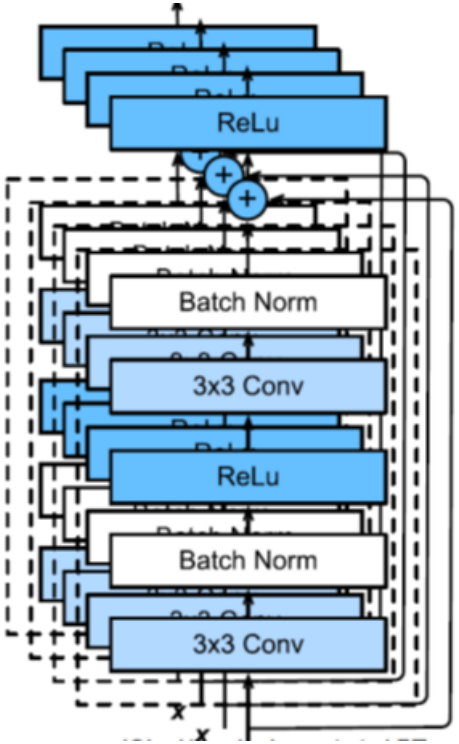
3-residual block



4-residual block



5-residual block



Layer Design of ResNet50 for segmentation

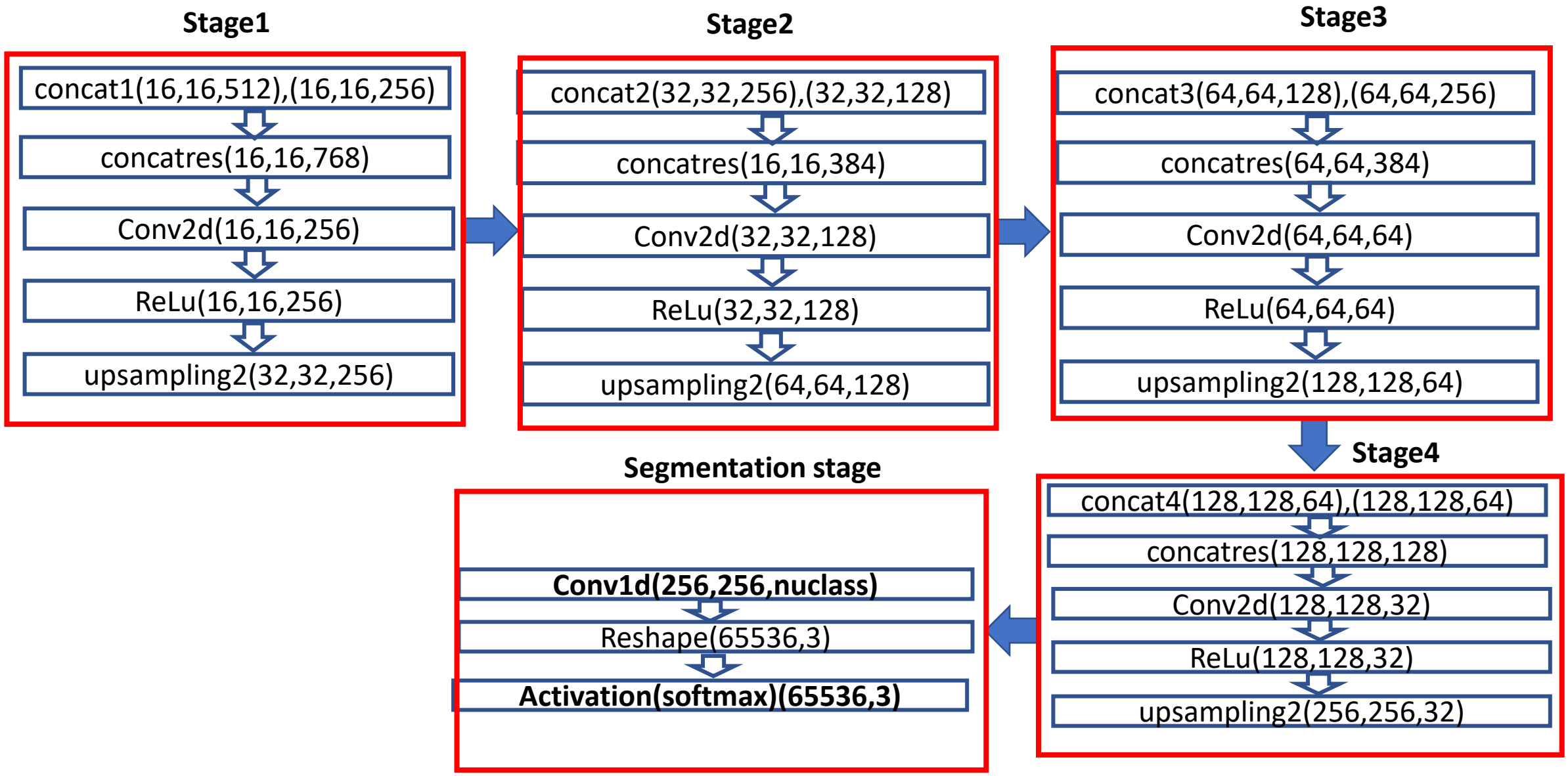
Extract Layers from Base Resnet Model

Input	256,256,3
Conv2d	128,128,64
BatchNormalization	128,128,64
activation_50 (Activation)	(None, 128, 128, 64) Block1
activation_59 (Activation)	(None, 64, 64, 256) Blcok2 2a, 2b,2c, 2b,2a,2c,2c,2a,2b
activation_61 (Activation)	None, 32, 32, 128) Blcok3 3a,2a,2b,2c,3b,2a,2b,2c,3c,2a, 2b,2c,3d,2a,2b,2c
activation_76 (Activation)	(None, 16, 16, 256) Blcok4 4a,2a,2b,2c,4b,2a,2b,2c,4c,2a, 2b,2c,4d,2a,2b,2c,4e,2a,2b,2c, 4f,2a,2b,2c
activation_93 (Activation)	(None, 8, 8, 512) Block5 5a,2a,2b,2c,5b,2a,2b,2c,5c,2a, 2b,2c

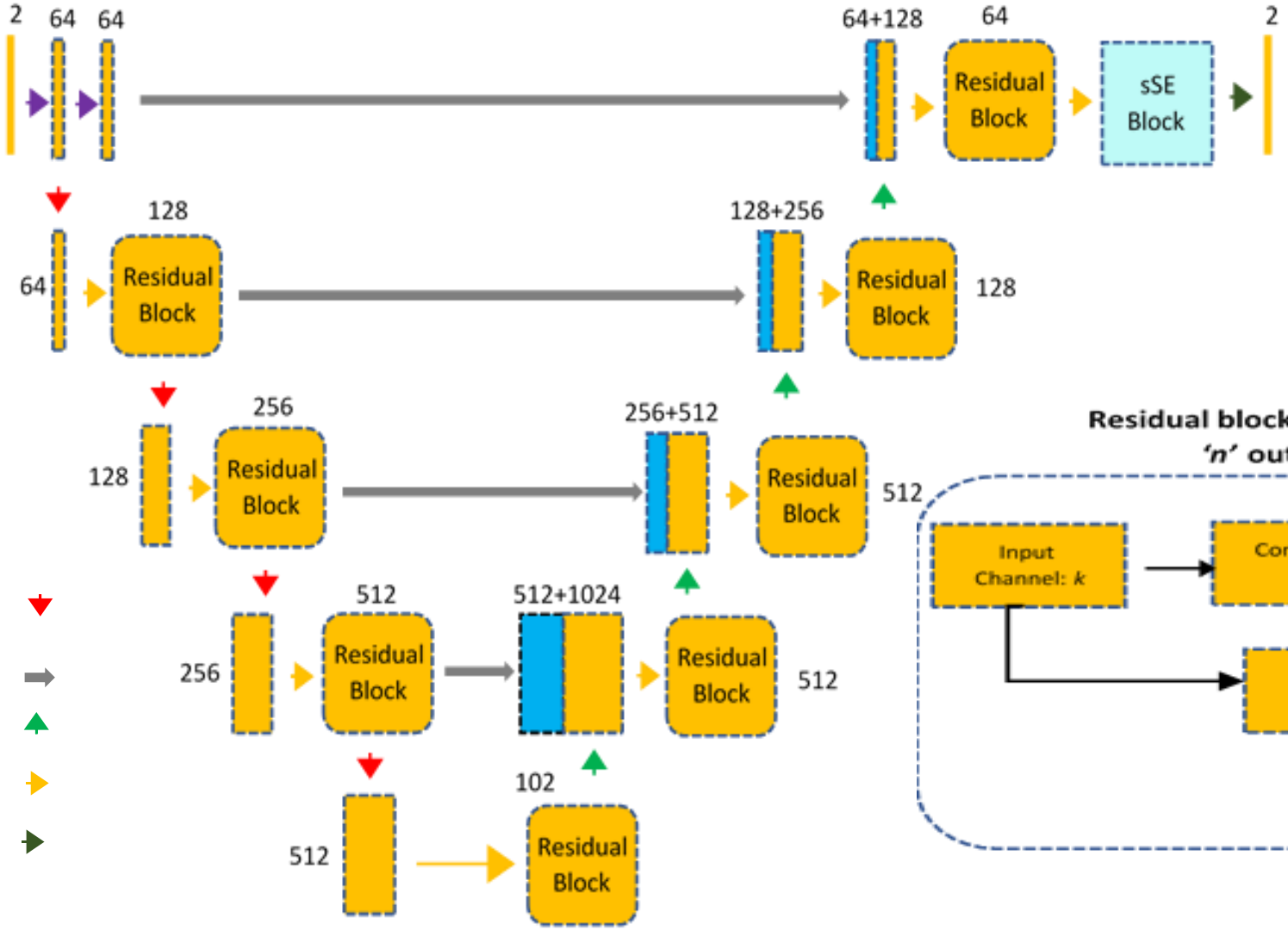
Decoder Part of Resnet Model

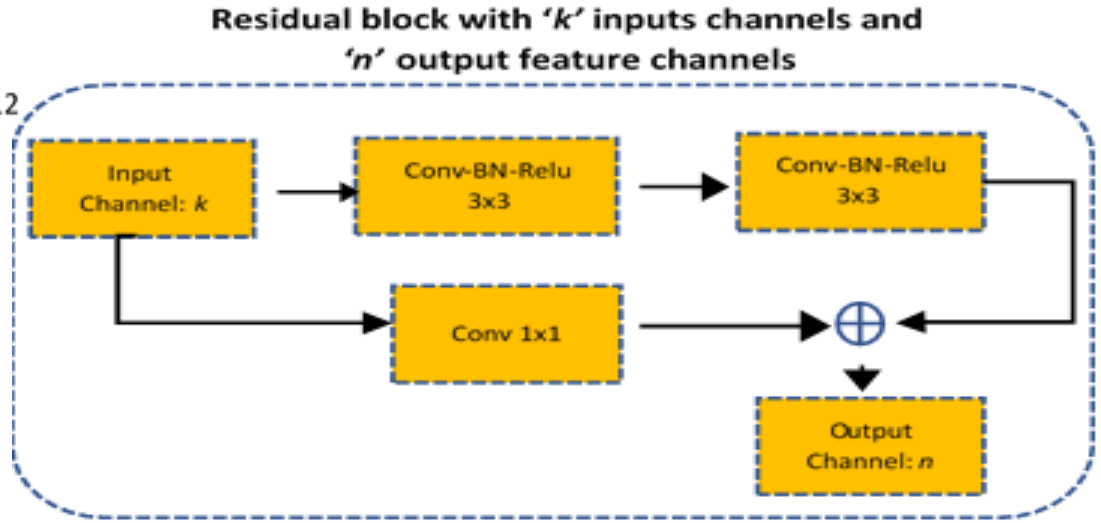
up_sampling2d_16	(None, 16, 16, 512)
concatenate_13 (Concatenate)	(None, 16, 16, 768)
conv2d_16 (Conv2D)	(None, 16, 16, 256)
up_sampling2d_17	(None, 32, 32, 256)
concatenate_14 (Concatenate	(None, 32, 32, 384)
conv2d_17 (Conv2D)	(None, 32, 32, 128)
up_sampling2d_18	(None, 64, 64, 128)
concatenate_15 (Concatenate)	(None, 64, 64, 384)
conv2d_18 (Conv2D)	(None, 64, 64, 64)
up_sampling2d_19 (UpSampling2D)	(None, 128, 128, 64)
concatenate_16 (Concatenate)	(None, 128, 128, 128)
conv2d_19 (Conv2D)	(None, 128, 128, 32)
up_sampling2d_20 (UpSampling2D)	(None, 256, 256, 32)
conv2d_20 (Conv2D)	(None, 256, 256, 3)
reshape_1 (Reshape)	(None, 3, 65536)

Decoder Part of ResNet

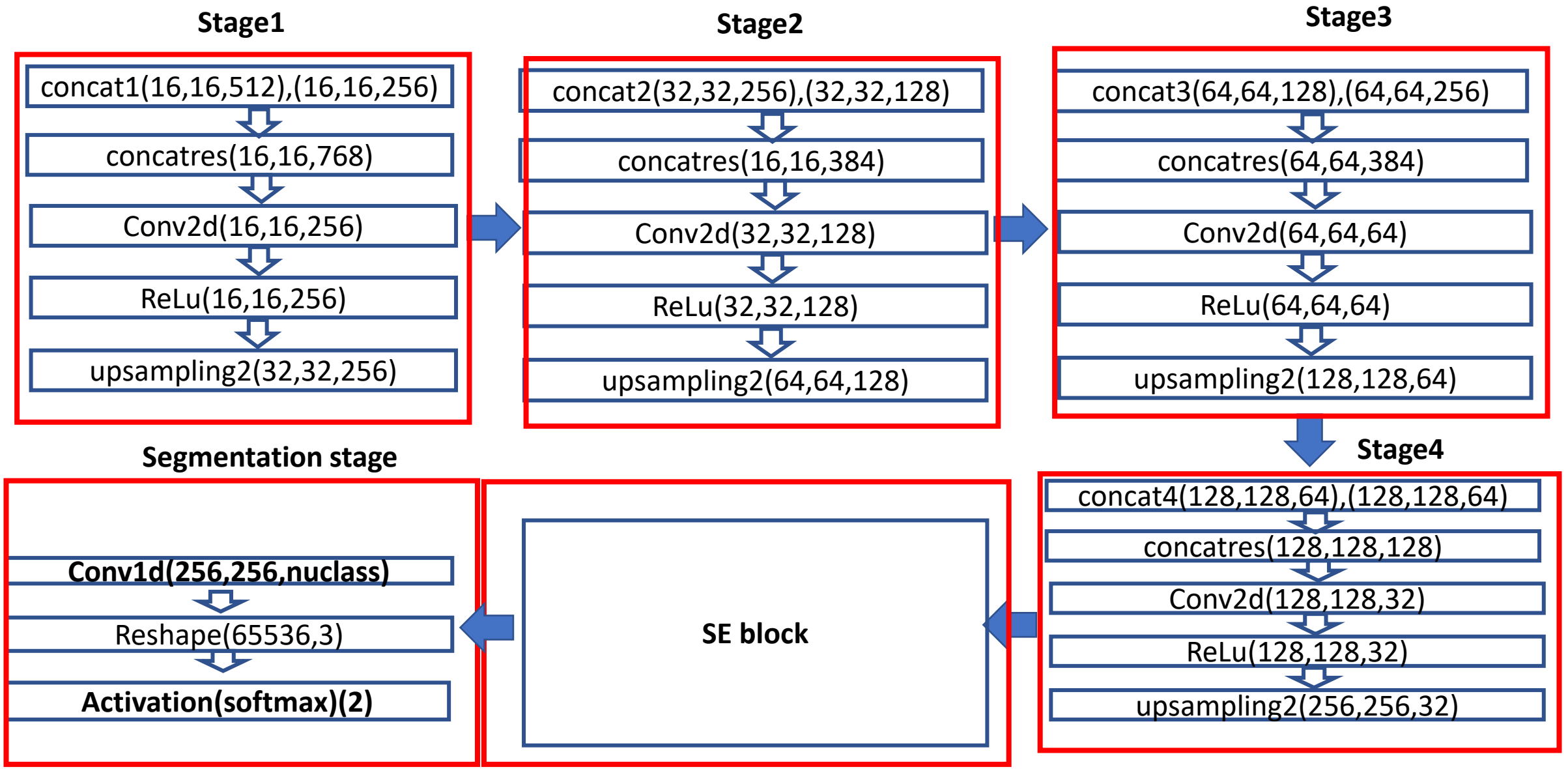


Proposed ResNet-SE network for Segmentation





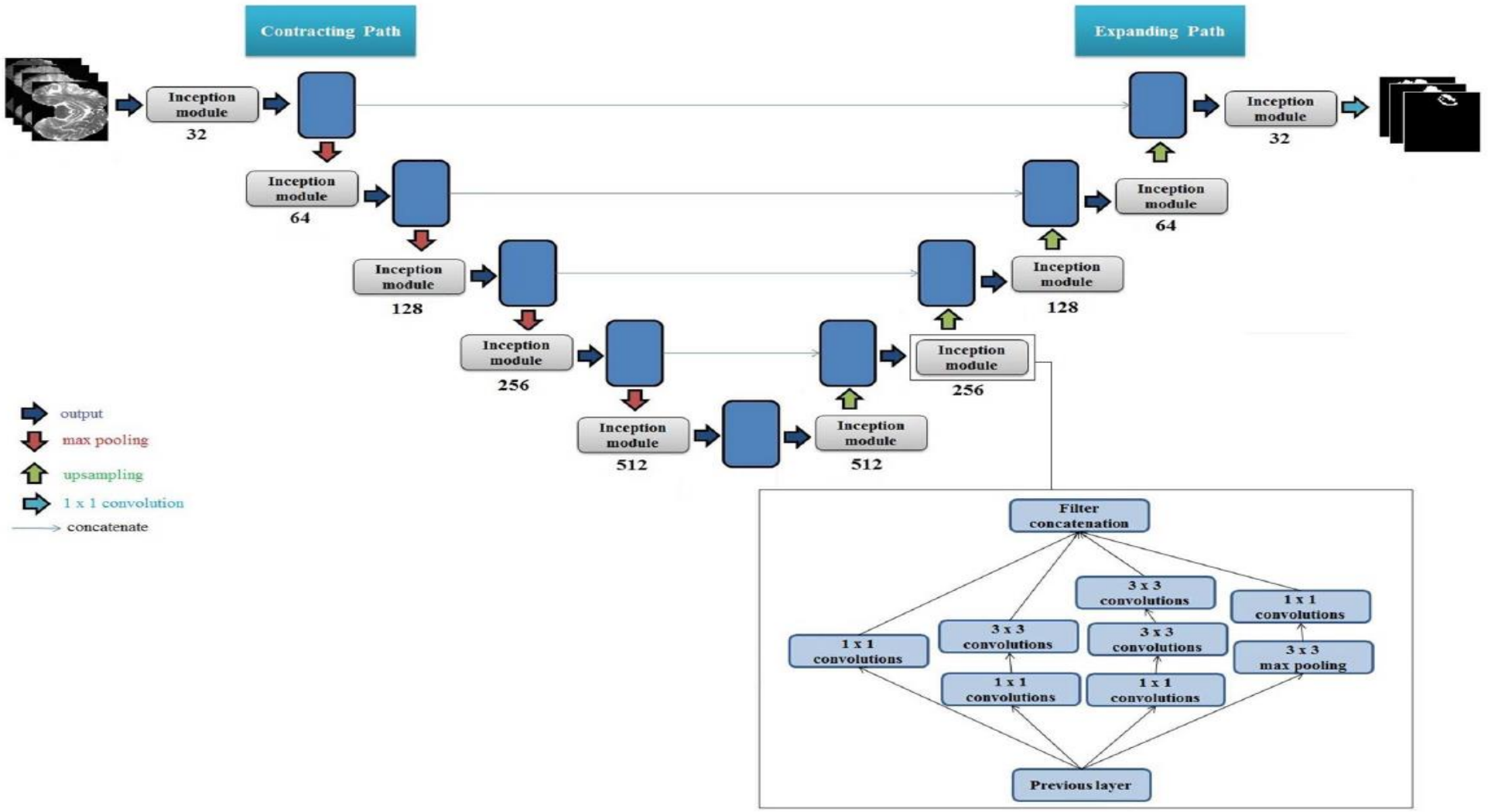
Decoder Part of ResNet



Recent Work on medical Segmentation

https://www.frontiersin.org/articles/10.3389/fncom.2019.00044/full

Inception Modules Enhance Brain Tumor Segmentation



Recent Work on medical Segmentation

A Generalized Deep Learning Framework for Whole-Slide Image Segmentation and Analysis

https://arxiv.org/abs/2001.00258

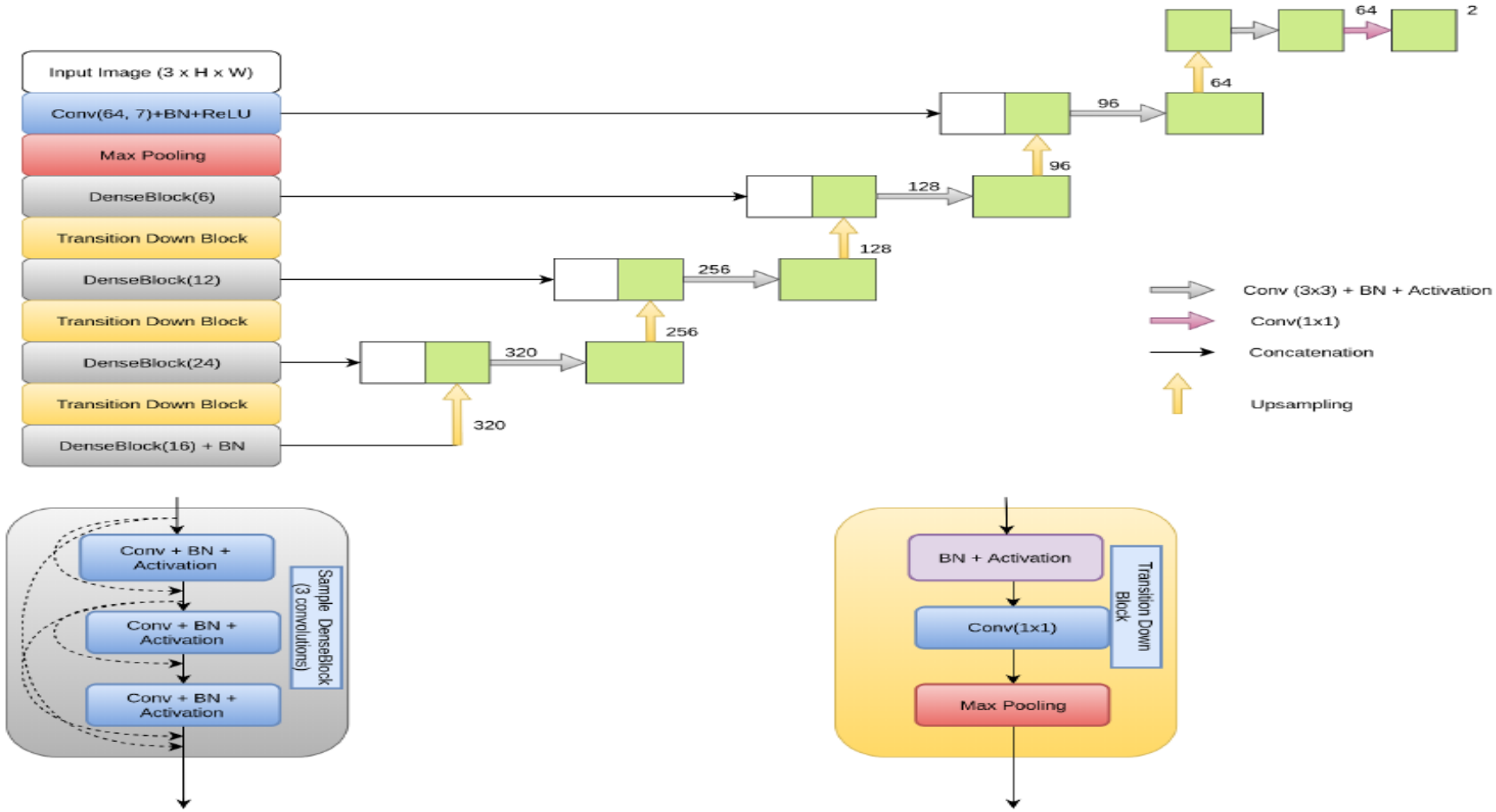


Figure 3: Densenet Architecture

Recent Work on medical Segmentation

A Generalized Deep Learning Framework for Whole-Slide Image Segmentation and Analysis

https://arxiv.org/abs/2001.00258

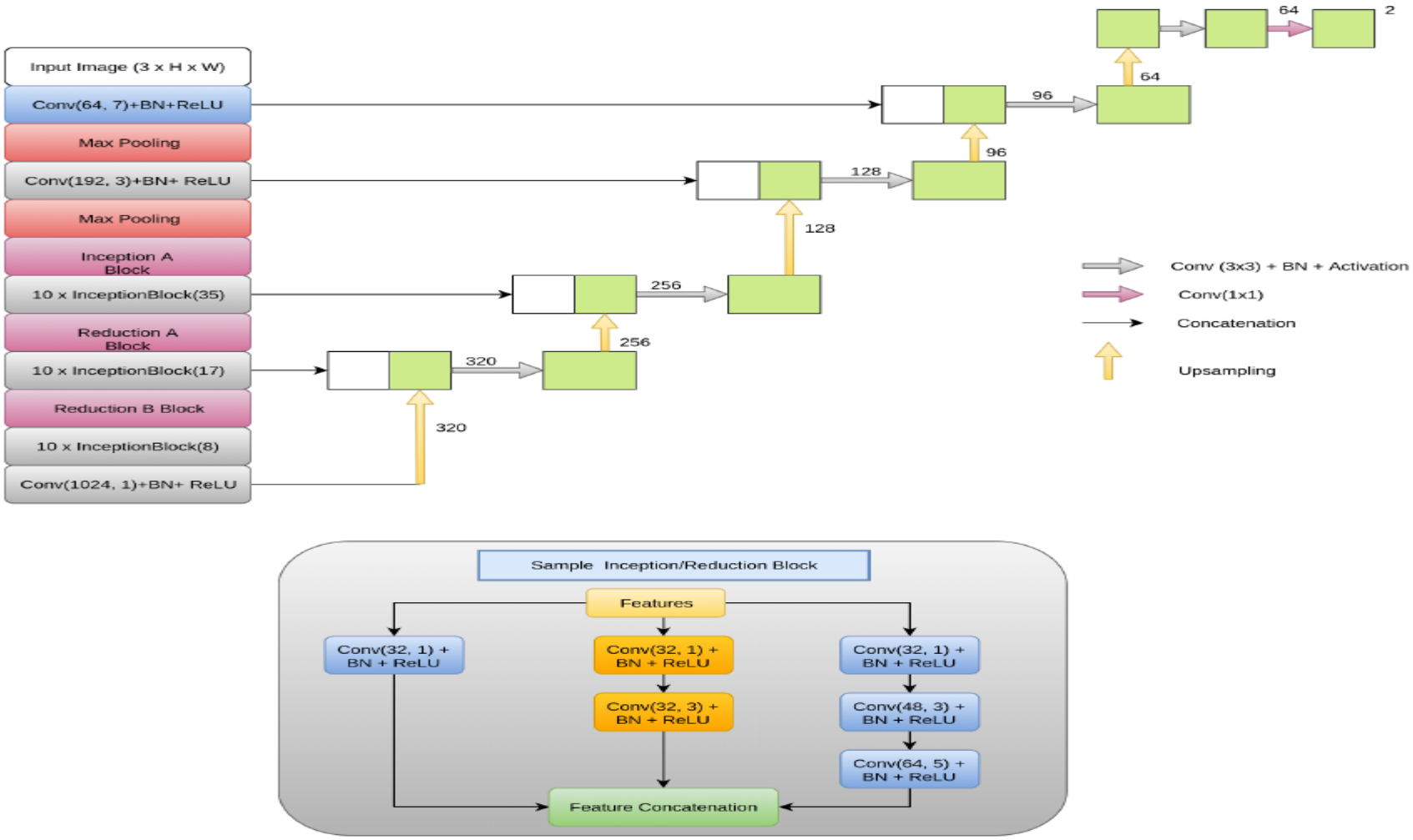


Figure 4: Inception-ResNetV2 architecture

Recent Work on medical Segmentation

A Generalized Deep Learning Framework for Whole-Slide Image Segmentation and Analysis

https://arxiv.org/abs/2001.00258

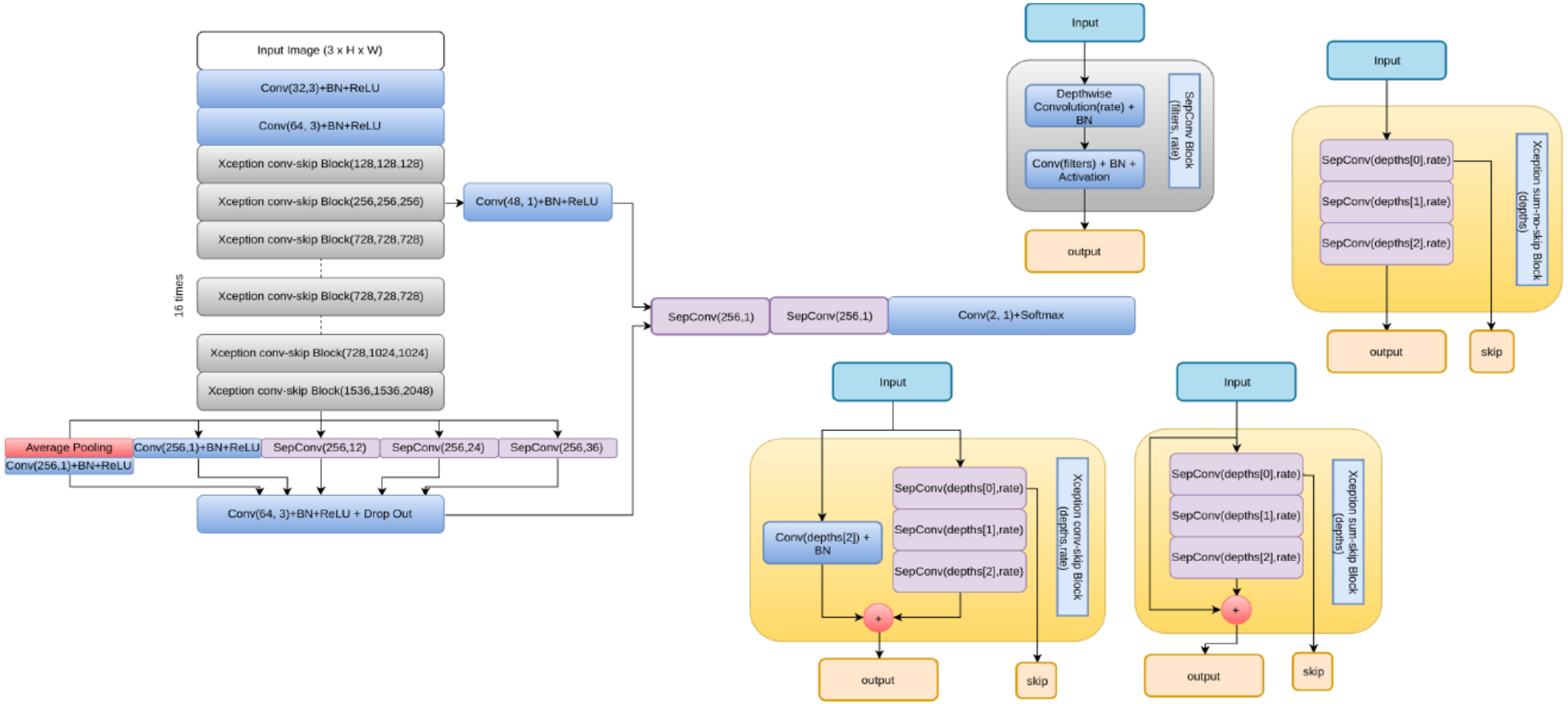


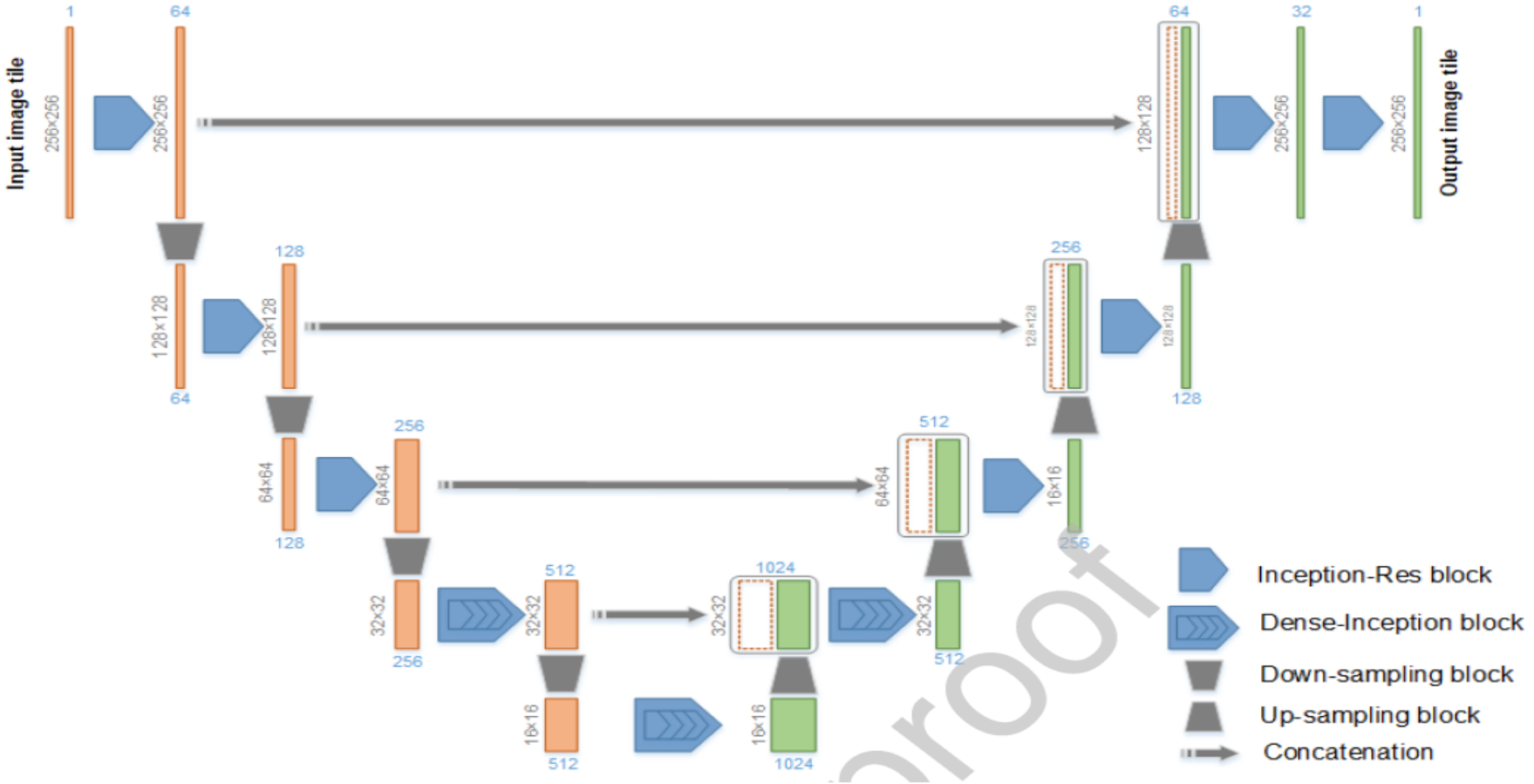
Figure 5: DeepLabV3Plus architecture

Recent Work on medical Segmentation

DENSE-INception U-net for medical image segmentation

https://www.sciencedirect.com/science/article/abs/pii/S0169260719307904?via%3Dihub

Fig. 1

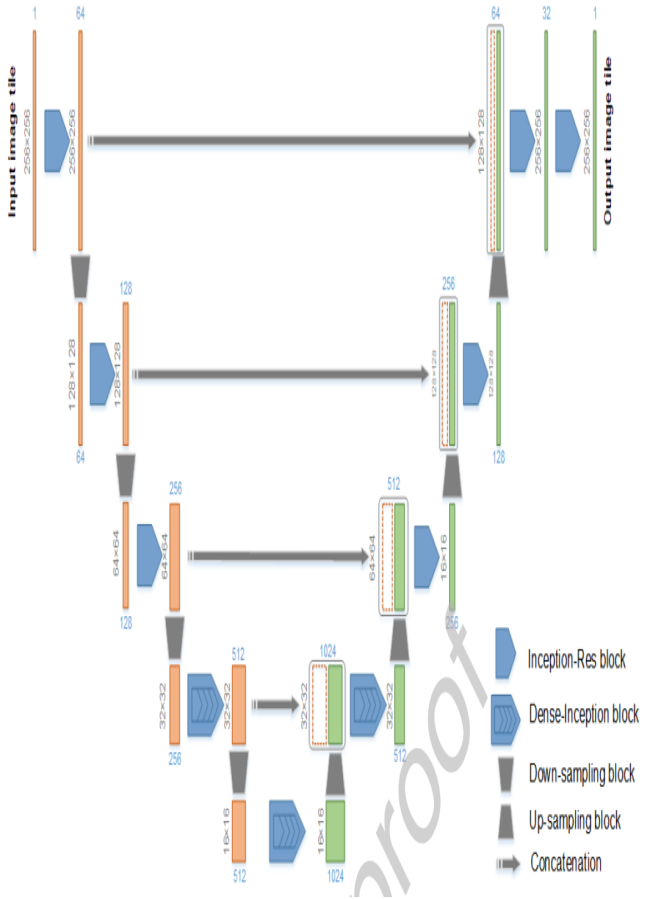


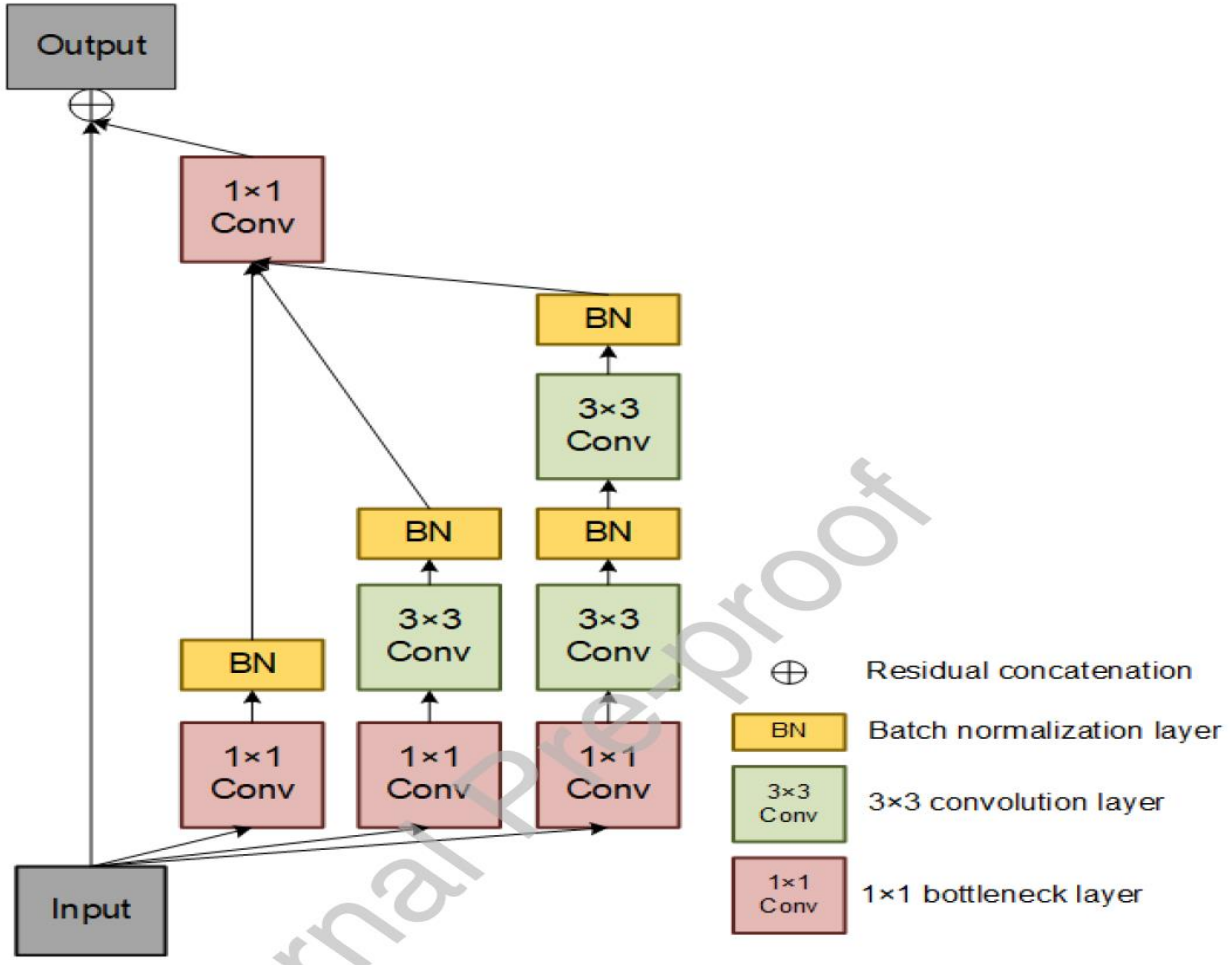
Recent Work on medical Segmentation

DENSE-INception U-net for medical image segmentation

https://www.sciencedirect.com/science/article/abs/pii/S0169260719307904?via%3Dihub

Fig. 1





Proposed ResNet-SE network for Segmentation

DENSE-INception U-net for medical image segmentation

https://www.sciencedirect.com/science/article/abs/pii/S0169260719307904?via%3Dihub

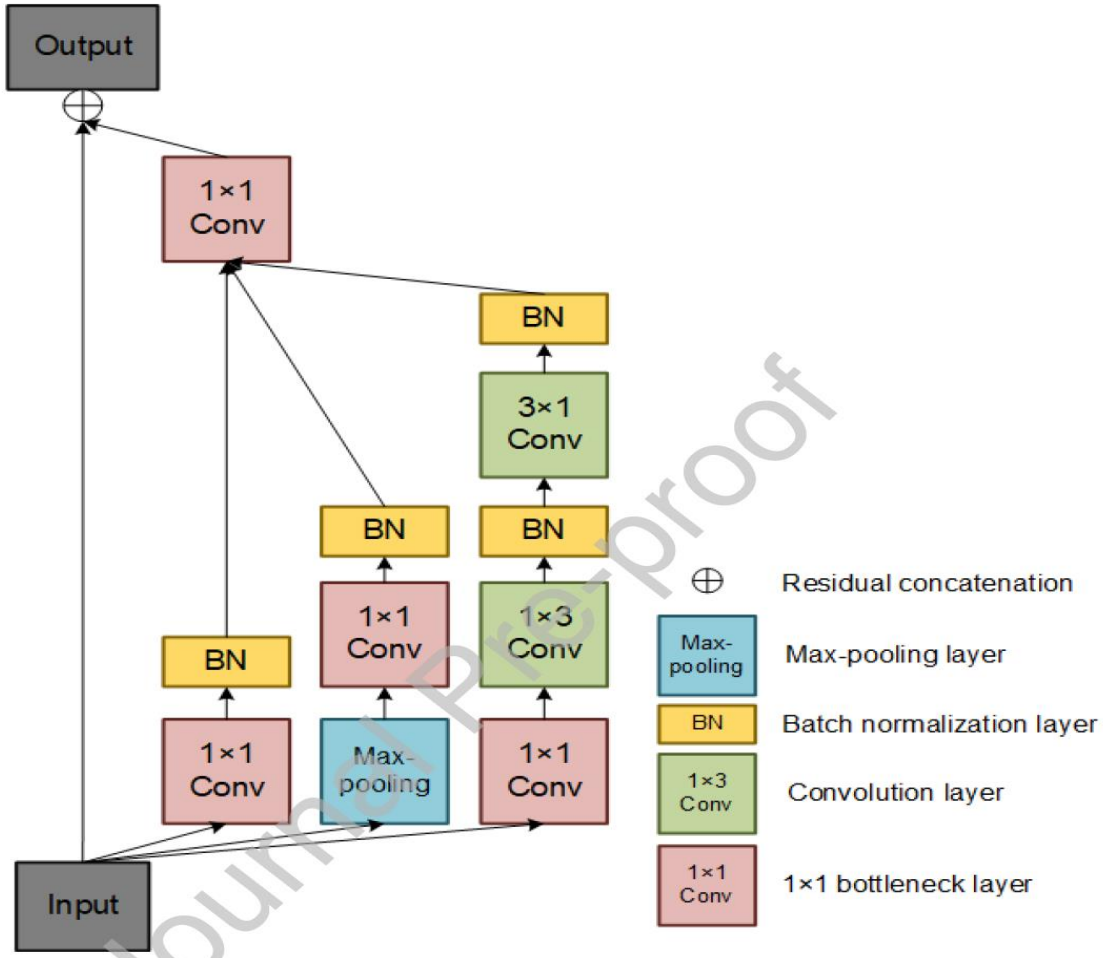
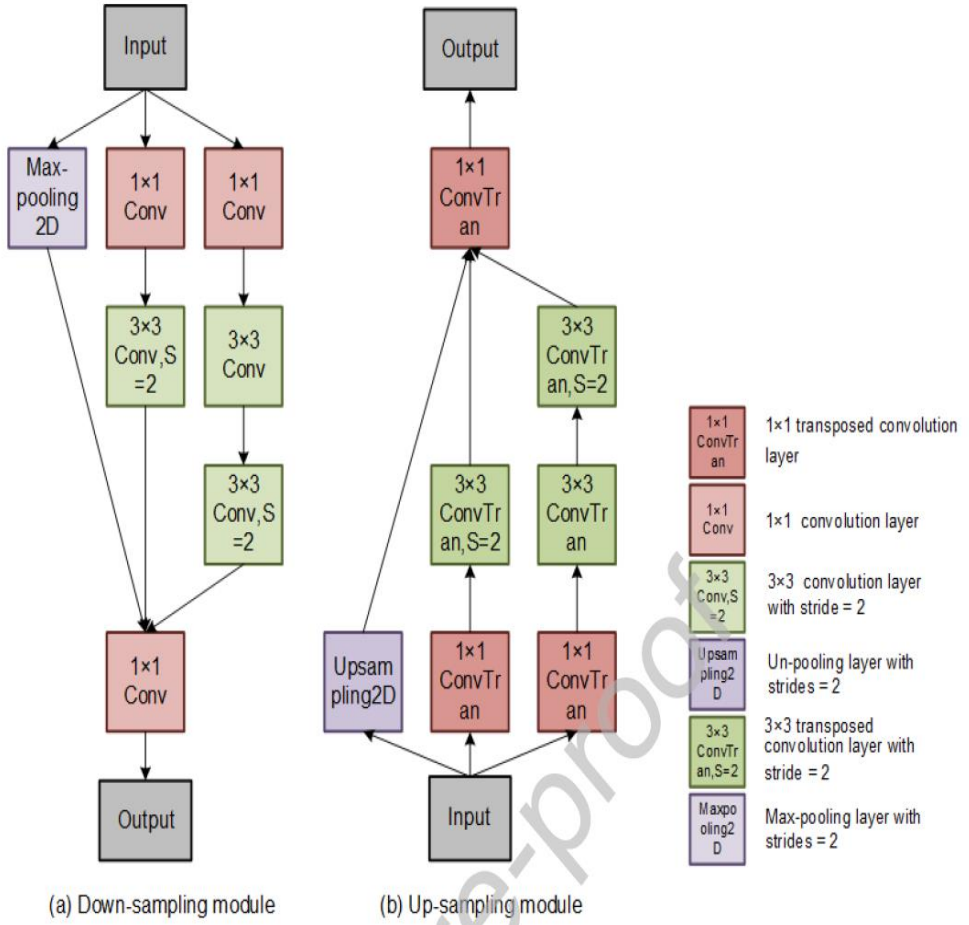


Fig 5



Recent Work on medical Segmentation

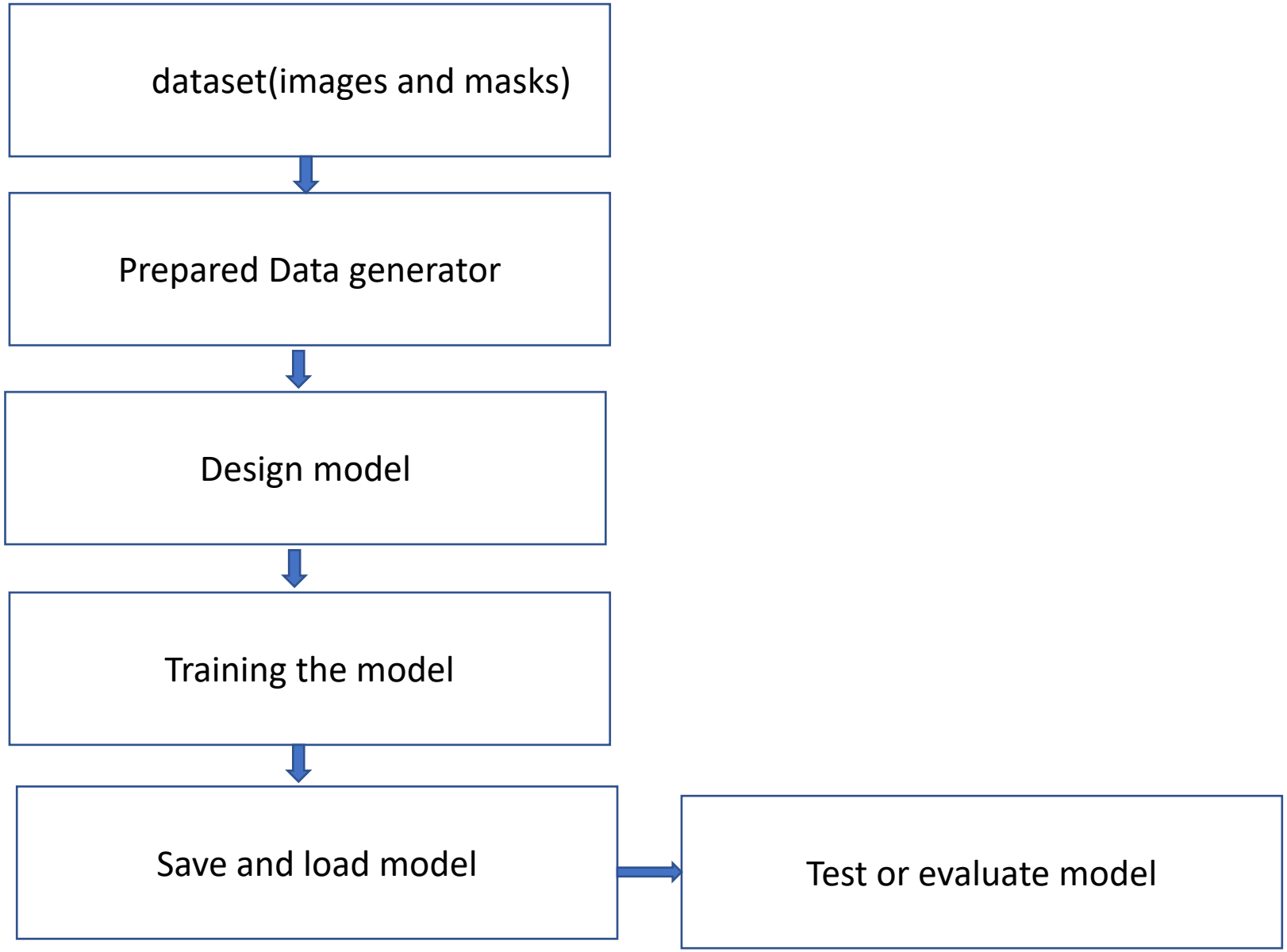
Inception U-Net Architecture for Semantic Segmentation to Identify Nuclei in Microscopy Cell Images

https://dl.acm.org/doi/fullHtml/10.1145/3376922

Table 1. Inception U-Net Layer Architecture

Path	Type	Fltr. Size/Strd./Padd.	Output Shape	1x1 Conv.	3x3 Conv.	5x5 Conv.	Hyd. Pool.	Params.
Contraction	Switch Norm.	—	128x128x3	—	—	—	—	18
	*Hyd. Pool. (0)	16x16/16/valid	8x8x3	—	—	—	—	0
	Incep. (1a)	—	128x128x51	16	16	16	2x2	1,728
	Conv. (1b)–D	1x1/1/same	128x128x16	—	—	—	—	832
	Hyd. Pool. (1c)	2x2/2/valid	64x64x16	—	—	—	—	0
	Incep. (2a)	—	64x64x112	32	32	32	2x2	18,016
	Conv. (2b)–D	1x1/1/same	64x64x32	—	—	—	—	3,616
	Hyd. Pool. (2c)	2x2/2/valid	32x32x32	—	—	—	—	0
	Incep. (3a)	—	32x32x224	64	64	64	2x2	71,872
	Conv. (3b)–D	1x1/s/same	32x32x64	—	—	—	—	14,400
	Hyd. Pool. (3c)	2x2/2/valid	16x16x64	—	—	—	—	0
	Incep. (4a)	—	16x16x448	128	128	128	2x2	280,104
	Conv. (4b)–D	1x1/1/same	16x16x128	—	—	—	—	57,472
	Hyd. Pool. (4c)	2x2/2/valid	8x8x128	—	—	—	—	0
	Incep. (5a)	—	8x8x896	256	256	256	2x2	1,147,648
	Concat. (5a,0)	—	8x8x899	—	—	—	—	0
	Conv. (5b)	1x1/1/same	8x8x256	—	—	—	—	230,400
Expansion	Conv.T (6a)	2x2/2/same	16x16x128	—	—	—	—	131,200
	Concat. (6a,4b)	—	16x16x256	—	—	—	—	0
	Incep. (6b)	—	16x16x640	128	128	128	2x2	1,147,264
	Conv. (6c)–D	1x1/1/same	16x16x128	—	—	—	—	82,048
	Conv.T (7a)	2x2/2/same	32x32x64	—	—	—	—	32,832
	Concat. (7a,3b)	—	32x32x128	—	—	—	—	0
	Incep. (7b)	—	32x32x320	64	64	64	2x2	286,912
	Conv. (7c)–D	1x1/1/same	32x32x64	—	—	—	—	20,544
	Conv.T (8a)	2x2/2/same	64x64x32	—	—	—	—	8,224
	Concat. (8a,2b)	—	64x64x64	—	—	—	—	0
	Incep. (8b)	—	64x64x160	32	32	32	2x2	71,776
	Conv. (8c)–D	1x1/1/same	64x64x32	—	—	—	—	5,152
	Conv.T (9a)	2x2/2/same	128x128x16	—	—	—	—	2,064
	Concat. (9a,1b)	—	128x128x32	—	—	—	—	0
	Incep. (9b)	—	128x128x80	16	16	16	2x2	17,968
	Conv. (9c)	1x1/1/same	128x128x16	—	—	—	—	1,296
	Conv. (O/P)	1x1/1/same	128x128x1	—	—	—	—	17

Steps in Segmentation



Steps in Segmentation

Data Preparation

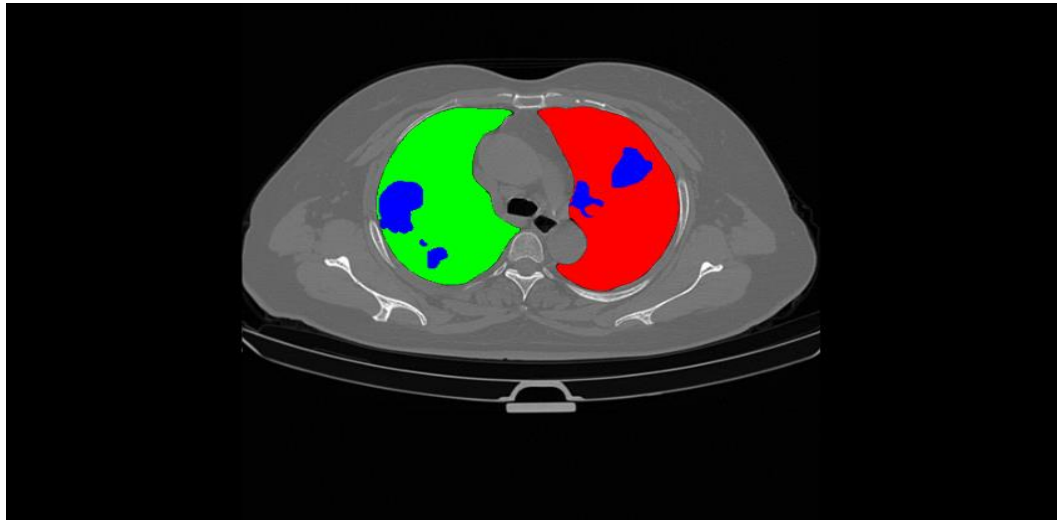
Steps in Segmentation

Data-generator for training validation and testing

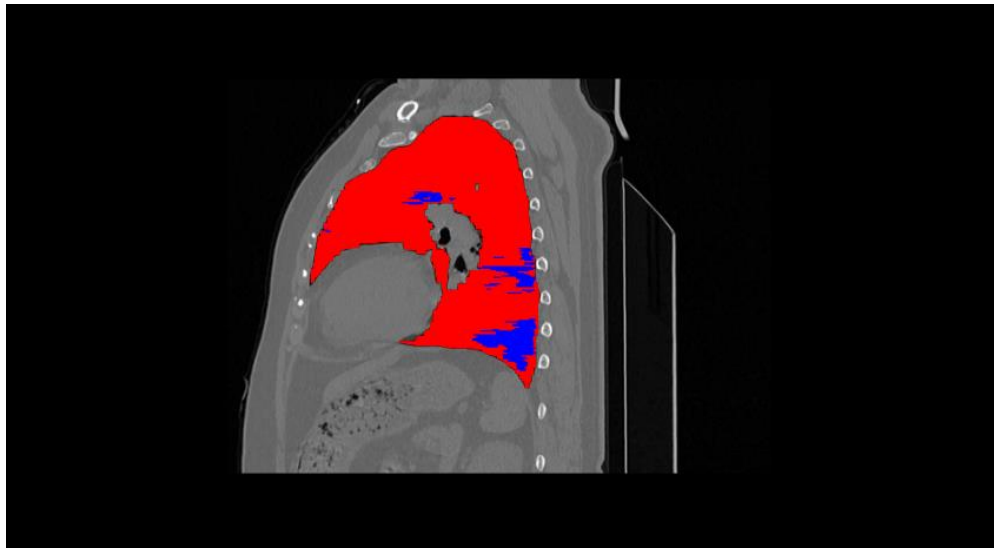
```
# Classes for data loading and preprocessing
class Dataset:
    """Dataset. Read images, apply augmentation and preprocessing tran
sformations.


    CLASSES = ['RightLung', 'LeftLung', 'Disease', 'unlabelled']

    def __init__(
            self,
            images_dir,
            masks_dir,
            classes=None,
            augmentation=None,
            preprocessing=None,
    ):
        self.ids = os.listdir(images_dir)

        # Sorted
        self.images_fps = sorted([os.path.join(images_dir, image_id) f
or image_id in self.ids])
        self.masks_fps = sorted([os.path.join(masks_dir, image_id) for
 image_id in self.ids])


        # convert str names to class values on masks
        self.class_values = [self.CLASSES.index(cls) for cls in classe
s] # cls used instead of cls.lower()

        self.augmentation = augmentation
        self.preprocessing = preprocessing
```

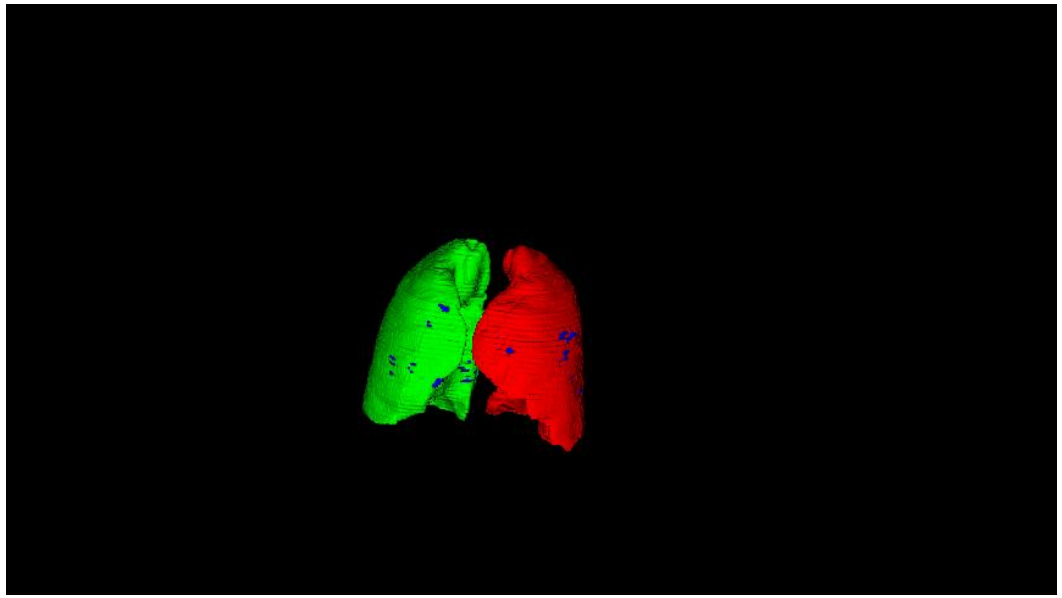
```
def __getitem__(self, i):

        # Read data
        image = cv2.imread(self.images_fps[i])
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        #image = skimage.io.imread(self.images_fps[i])


        mask = cv2.imread(self.masks_fps[i], 0)
        #mask = skimage.io.imread(self.masks_fps[i])


        # Extract certain classes from mask
        masks = [(mask == v) for v in self.class_values]
        mask = np.stack(masks, axis=-1).astype('float')

        # Add background if mask is not binary
        if mask.shape[-1] != 1:
            background = 1 - mask.sum(axis=-1, keepdims=True)
            mask = np.concatenate((mask, background), axis=-1)

        # Apply augmentations
        if self.augmentation:
            sample = self.augmentation(image=image, mask=mask)
            image, mask = sample['image'], sample['mask']

        # Apply preprocessing
        if self.preprocessing:
            sample = self.preprocessing(image=image, mask=mask)
            image, mask = sample['image'], sample['mask']

        return image, mask

    def __len__(self):
        return len(self.ids)
```

Steps in Segmentation

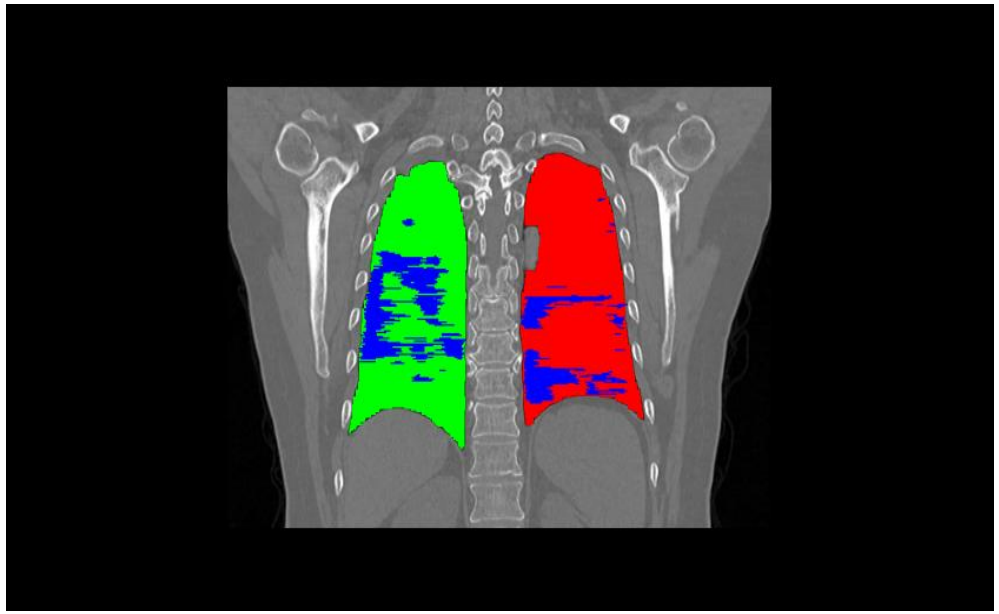


(green: left lung, red: right lung, blue: disease)





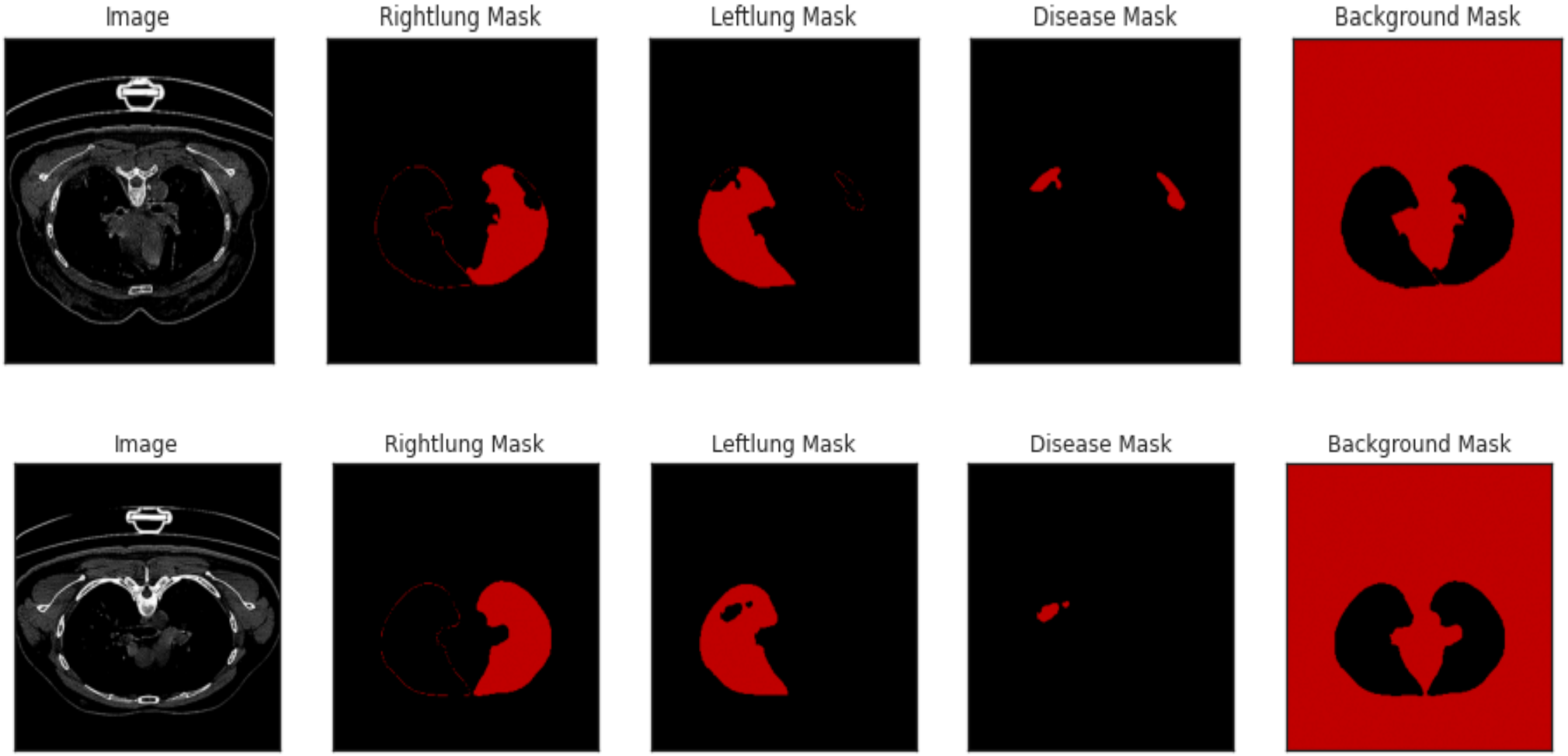
3D view



Steps in Segmentation

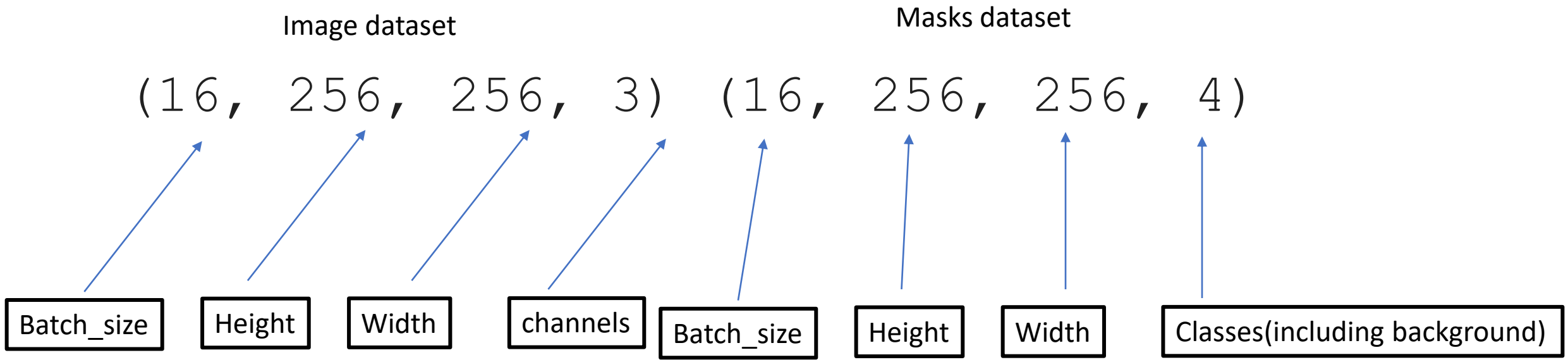
Data-generator for training, validation and testing

(256, 256, 3) (256, 256, 4)



Steps in Segmentation

Data-generator for training validation and testing



Steps in Segmentation

Model Designing

Steps in Segmentation

DensNet Base Unet model(we will see layer by layer in lab work)

Steps in Segmentation

Xception based Encoder and Resnet based decoder model (we will see layer by layer in lab work)

Steps in Segmentation

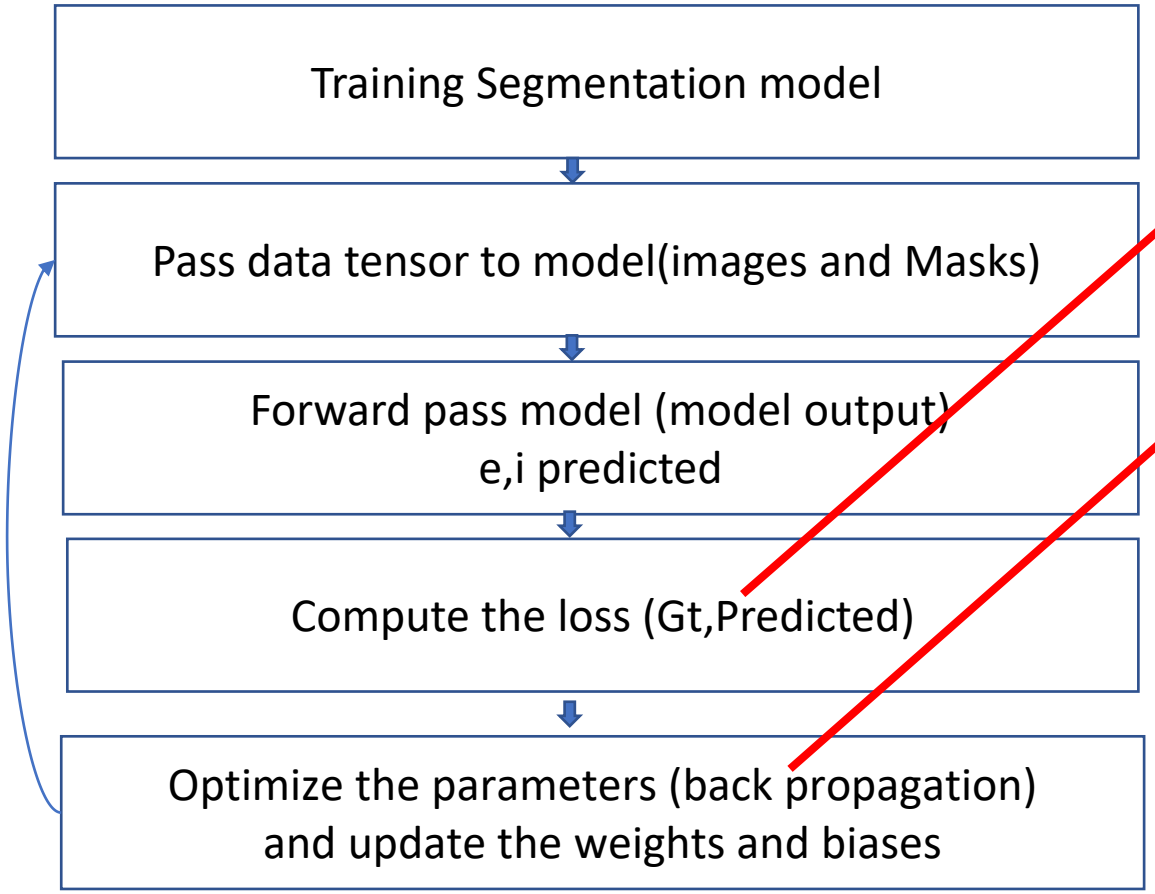
Efficient based Encoder and Resnet based decoder model (we will see layer by layer in lab work)

Steps in Segmentation

Model Training

Steps in Segmentation

Training the model for segmentation




```
model.compile(optimizer='adam',
loss=Combo_Loss, metrics=[dice_c
oef])
```

```
history = model.fit_generator(
    train_dataloader,
     steps_per_epoch=len(train_datalo
ader),
    epochs=EPOCHS,
    callbacks=callbacks,
     validation_data=valid_dataloader
,
     validation_steps=len(valid_datal
oader),  )
```

Steps in Segmentation

Training the model for segmentation
Optimizers used for training the segmentation modes are the same as for training the classification models

Optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Steps in Segmentation

https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

Training the model for segmentation

Loss function used between GT and predicted segmentation map

Lars Nieradzik

Machine learning and language enthusiast

Bremen, Germany

Email

LinkedIn

GitHub

In this post, I will implement some of the most common losses for image segmentation in Keras/TensorFlow. I will only consider the case of two classes (i.e. binary). If you know any other losses, let me know and I will add them.

16.08.2019: improved overlap measures, added CE+DL loss

Cross Entropy

Let $\mathbf{P}(Y = 0) = p$ and $\mathbf{P}(Y = 1) = 1 - p$. The predictions are given by the logistic/sigmoid function $\mathbf{P}(\hat{Y} = 0) = \frac{1}{1+e^{-x}} = \hat{p}$ and $\mathbf{P}(\hat{Y} = 1) = 1 - \frac{1}{1+e^{-x}} = 1 - \hat{p}$. Then cross entropy (CE) can be defined as follows:

$$\mathrm{CE}\left(p, \hat{p}\right) = -\left(p \log(\hat{p}) + (1-p) \log(1-\hat{p})\right)$$

In Keras, the loss function is `binary_crossentropy(y_true, y_pred)` and in TensorFlow, it is `softmax_cross_entropy_with_logits_v2`.

Steps in Segmentation

https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

Training the model for segmentation

Loss function used between GT and predicted segmentation map

Cross Entropy

Let $\mathbf{P}(Y = 0) = p$ and $\mathbf{P}(Y = 1) = 1 - p$. The predictions are given by the logistic/sigmoid function $\mathbf{P}(\hat{Y} = 0) = \frac{1}{1+e^{-x}} = \hat{p}$ and $\mathbf{P}(\hat{Y} = 1) = 1 - \frac{1}{1+e^{-x}} = 1 - \hat{p}$. Then cross entropy (CE) can be defined as follows:

$$\mathrm{CE}\left(p, \hat{p}\right) = -\left(p \log(\hat{p}) + (1-p) \log(1-\hat{p})\right)$$

In Keras, the loss function is `binary_crossentropy(y_true, y_pred)` and in TensorFlow, it is `softmax_cross_entropy_with_logits_v2`.

Steps in Segmentation

https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

Training the model for segmentation

Loss function used between GT and predicted segmentation map

Weighted cross entropy

Weighted cross entropy (WCE) is a variant of CE where all positive examples get weighted by some coefficient. It is used in the case of class imbalance. For example, when you have an image with 10% black pixels and 90% white pixels, regular CE won't work very well.

WCE can be defined as follows:

$$\mathrm{WCE}\left(p, \hat{p}\right) = -\left(\beta p \log(\hat{p}) + (1-p)\log(1-\hat{p})\right)$$

To decrease the number of false negatives, set $\beta > 1$. To decrease the number of false positives, set $\beta < 1$.

In TensorFlow, the loss function is `weighted_cross_entropy_with_logits`. In Keras, we have to

Steps in Segmentation

https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

Training the model for segmentation

Loss function used between GT and predicted segmentation map

Overlap measures

Dice Loss / F1 score

The Dice coefficient is similar to the Jaccard Index (Intersection over Union, IoU):

$$\mathrm{DC} = \frac{2TP}{2TP + FP + FN} = \frac{2|X \cap Y|}{|X| + |Y|}$$

$$\mathrm{IoU} = \frac{TP}{TP + FP + FN} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

where TP are the true positives, FP false positives and FN false negatives. We can see that $\mathrm{DC} \geq \mathrm{IoU}$.

The dice coefficient can also be defined as a loss function:

$$\mathrm{DL}\left(p, \hat{p}\right) = 1 - \frac{2p\hat{p} + 1}{p + \hat{p} + 1}$$

where $p \in \{0, 1\}$ and $0 \leq \hat{p} \leq 1$.

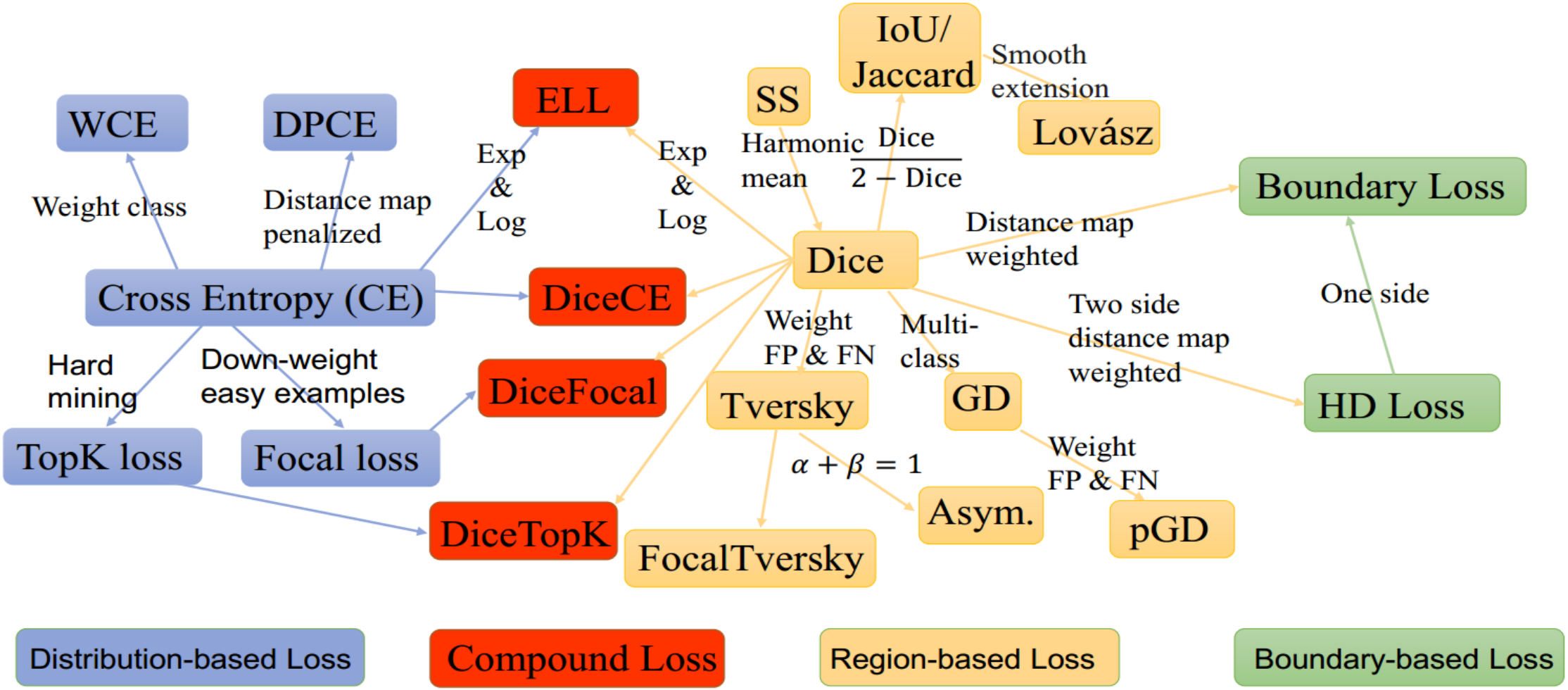
```
def dice_loss(y_true, y_pred):
  numerator = 2 * tf.reduce_sum(y_true * y_pred, axis=-1)
  denominator = tf.reduce_sum(y_true + y_pred, axis=-1)

  return 1 - (numerator + 1) / (denominator + 1)
```

Adding one to the numerator and denominator is quite important. For example, when $p = \hat{p} = 0$, the result should be 0. But without the "+1" term, we get $1 - \frac{2 \cdot 0 \cdot 0}{0+0} = 1$.

The "+1" term has two effects: (1) shift the range from $[0, 1]$ to $[0, 0.5]$, (2) prevent $\mathrm{DL}\left(p, \hat{p}\right) = 0$, when $p = 0$ and $\hat{p} > 0$. The disadvantage is when $p = 0$, we get $1 - \frac{1}{\hat{p}+1} = \frac{\hat{p}}{\hat{p}+1}$.

Steps in Segmentation

https://github.com/JunMa11/SegLoss



https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

Steps in Segmentation

Model Testing

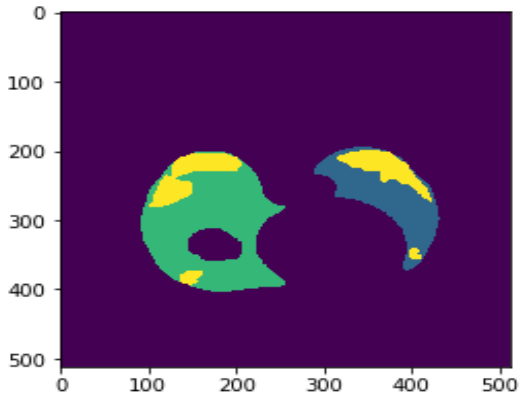
Steps in Segmentation

Validation or testing the performance of model for segmentation

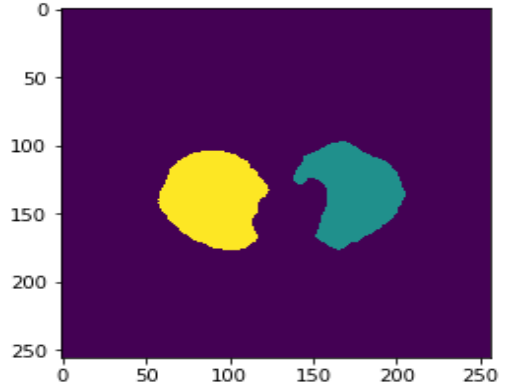
Performance Metrics:

1. $Sensitivity = TPR = \frac{TP}{TP+FN}$

2. $Specificity = TNR = \frac{TN}{TN + FP}$



GT(A)



Predicted(B)

3. Jaccard index

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

where A is ground-truth mask and B is predicted segmentation map)

4. Dice Coefficients (DC)

$$Dice(A,B) = \frac{2|A \cap B|}{|A| + |B|}$$

where A is ground-truth mask and B is predicted segmentation map)

Steps in Segmentation

Validation or testing the performance of model for segmentation

Performance Metrics:

5. Volume Overlap Error (VOE)

$$VOE(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

6. Relative Volume Difference (RVD)

$$RVD(A,B) = \frac{|B| - |A|}{|A|}$$

6. Surface Distance Metrics

Let S(A) denotes the set of surface voxels of A and the shortest distance of an arbitrary voxel v to S(A) is shown in equation

$$d(v, S(A)) = min_{S_A \epsilon S(A)} \|v - S_A\|$$

$$ASD(A,B) = \frac{1}{|S(A)| + |S(B)|}\left(\sum_{S_A \epsilon S(A)} d(S_A, S(B)) + \sum_{S_B \epsilon S(B)} d(S_B, S(A))\right)$$

7. Hausdorff Distance

Symmetric Hausdorff Distance is used to compute the (symmetric) Hausdorff Distance (HD) between the binary objects in two segmentation masks. It is defined as the maximum surface distance (MSD) between the objects.

$$MSD(A,B)$$
$$= max\{max_{S_A \epsilon S(A)} d(S_A, S(B)), max_{S_B \epsilon S(B)} d(S_B, S(A))\}$$

Steps in Segmentation

Validation or testing the performance of model for segmentation

Figure 1, for two point sets X and Y, the one-sided HD from X to Y is defined as [7]:

$$\mathrm{hd}(X,Y) = \max_{x\in X} \min_{y\in Y} \|x-y\|_2, \qquad (1)$$

and similarly for $\mathrm{hd}(Y,X)$:

$$\mathrm{hd}(Y,X) = \max_{y\in Y} \min_{x\in X} \|x-y\|_2. \qquad (2)$$

The bidirectional HD between these two sets is then:

$$\mathrm{HD}(X,Y) = \max\big(\mathrm{hd}(X,Y), \mathrm{hd}(Y,X)\big) \qquad (3)$$

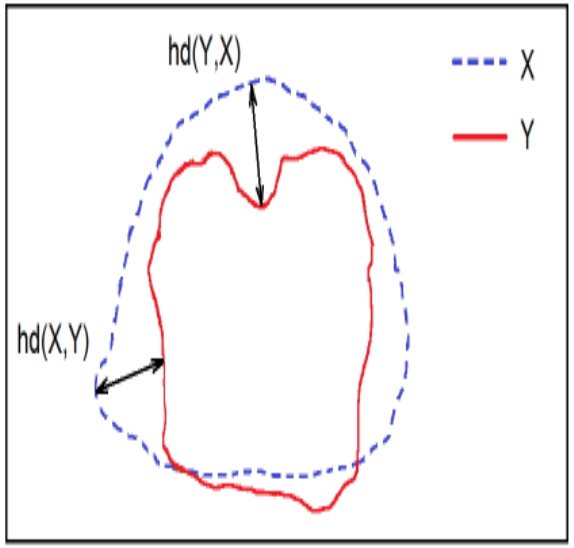


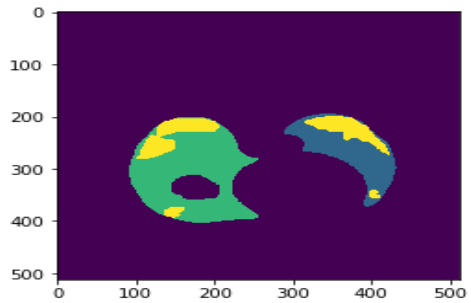
Figure 1. A schematic showing the Hausdorff Distance between points sets X and Y.

Steps in Segmentation

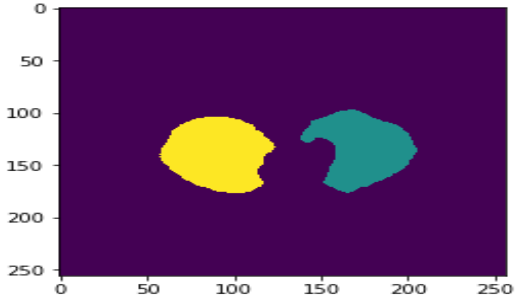
Validation or testing the performance of model for segmentation

1. *Sensitivity*
2. Specificity
3. Jaccard Index
4. Dice similarity Coefficients

Should be high



Manual Mask(GT)



Predicted by model

1. Volume Overlap Error
2. Relative Volume Difference
3. Surface Distance Metrics(Asymmetric Surface distance(ASD))
4. Hausdorff Distance(HD)

Should be low

Steps in Segmentation

Model Testing(Coding)

Steps in Segmentation

Validation or testing the performance of model for segmentation

```
# code
def dc(result, reference):
    """
    Dice coefficient

    Computes the Dice coefficient (also known as Sorensen index) between the binary
    objects in two images.

    The metric is defined as

    .. math::

        DC=\frac{2|A\cap B|}{|A|+|B|}

    , where :math:`A` is the first and :math:`B` the second set of samples (here: binary objects).

 """
    result = numpy.atleast_1d(result.astype(numpy.bool))
    reference = numpy.atleast_1d(reference.astype(numpy.bool))

    intersection = numpy.count_nonzero(result & reference)

    size_i1 = numpy.count_nonzero(result)
    size_i2 = numpy.count_nonzero(reference)

    try:
        dc = 2. * intersection / float(size_i1 + size_i2)
    except ZeroDivisionError:
        dc = 0.0

    return dc
```

Steps in Segmentation

Validation or testing the performance of model for segmentation

```
def jc(result, reference):
    """
    Jaccard coefficient

    Computes the Jaccard coefficient between the binary objects in two images.

    Parameters
    ----------
    result: array_like
            Input data containing objects. Can be any type but will be converted
            into binary: background where 0, object everywhere else.
    reference: array_like
            Input data containing objects. Can be any type but will be converted
            into binary: background where 0, object everywhere else.
        Notes
    -----
    This is a real metric. The binary images can therefore be supplied in any order.
    """
    result = numpy.atleast_1d(result.astype(numpy.bool))
    reference = numpy.atleast_1d(reference.astype(numpy.bool))

    intersection = numpy.count_nonzero(result & reference)
    union = numpy.count_nonzero(result | reference)

    jc = float(intersection) / float(union)

    return jc
```

Steps in Segmentation

```
def hd(result, reference, voxelspacing=None, connectivity=1):
    """
    Hausdorff Distance.

    Computes the (symmetric) Hausdorff Distance (HD) between the binary objects in two
    images. It is defined as the maximum surface distance between the objects.

    Parameters
    ----------
    result : array_like
        Input data containing objects. Can be any type but will be converted
        into binary: background where 0, object everywhere else.
    reference : array_like
        Input data containing objects. Can be any type but will be converted
        into binary: background where 0, object everywhere else.
    voxelspacing : float or sequence of floats, optional
        The voxelspacing in a distance unit i.e. spacing of elements
        along each dimension. If a sequence, must be of length equal to
        the input rank; if a single number, this is used for all axes. If
        not specified, a grid spacing of unity is implied.
    connectivity : int
        The neighbourhood/connectivity considered when determining the surface
        of the binary objects. This value is passed to
        `scipy.ndimage.morphology.generate_binary_structure` and should usually be :math:`> 1`.
        Note that the connectivity influences the result in the case of the Hausdorff distance.

    -----
    This is a real metric. The binary images can therefore be supplied in any order.
    """
    hd1 = __surface_distances(result, reference, voxelspacing, connectivity).max()
    hd2 = __surface_distances(reference, result, voxelspacing, connectivity).max()
    hd = max(hd1, hd2)
    return hd
```

Steps in Segmentation

Code for computing performance matrices between predicted and GT masks for test dataset

```
dc11=[]
jc11=[]
precisoion11=[]
jaccard11=[]
HD95111=[]
HD111=[]
Relativeabsolutevolumedifferenc11=[]
Averagesymmetricsurfacedistance11=[]
#import tensorflow as tf
#sys.stdout.flush()
for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):
    path = path_train
    GT_array= imread(path + '/mask11/' + id_)
    ct_array = np.array(GT_array)[:,:,1]
    p=imread(path + '/M11/' + id_)
    seg_array=p[:,:,1]
    #mask = np.expand_dims((p)[:,:,1],
    dc1=dc(ct_array, seg_array)
    jc1=jc(ct_array, seg_array)
    precisoion1=precision(ct_array,seg_array)
    HD9511=hd95(ct_array,seg_array)
    HD11=hd(ct_array,seg_array)
    Relativeabsolutevolumedifferenc=ravd(ct_array,seg_array)
    Averagesymmetricsurfacedistance1=assd(ct_array,seg_array)
    Averagesurfacedistance1=asd(ct_array,seg_array)
    sensitivity1=sensitivity(ct_array,seg_array)
    specifity1=specificity(ct_array,seg_array)
    precision1=precision(ct_array,seg_array)
    recall1=recall(ct_array,seg_array)
```

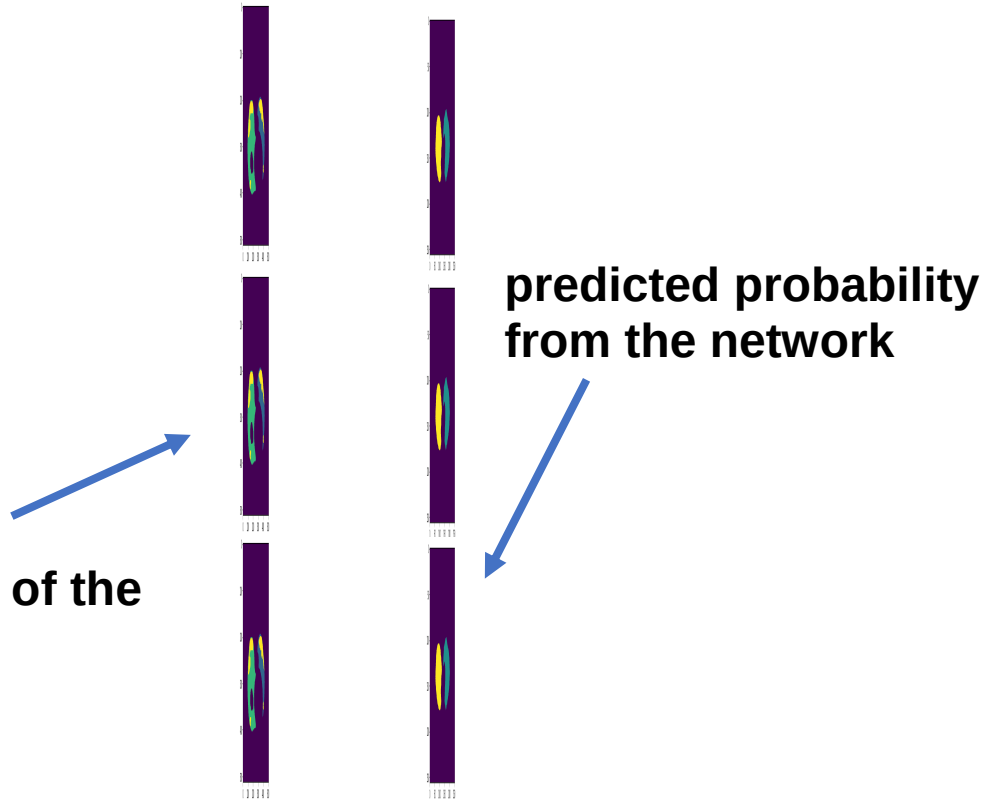
```
volume_correlationav=[volume_correlation11n,volume_correlatio
n11n1,volume_correlation11n11]
dicevalue=base(dc11)
avgbasedice=float(sum(dicevalue))/len(dicevalue)

print('dice base value==',avgbasedice)
mdice=middle(dc11)
avgmdice=float(sum(mdice))/len(mdice)
print('dice middle value==',avgmdice)

apexdice=Apex(dc11)
avgapxdice=float(sum(apexdice))/len(apexdice)
print('dice apex value==',avgapxdice)

hdbasevalue=base(HD111)
avgbasehd=float(sum(hdbasevalue))/len(hdbasevalue)
print('hd base value==',avgbasehd)

mhdvalue=middle(HD111)
avgmhdvalue=float(sum(mhdvalue))/len(mhdvalue)
print('hd middle value==',avgmhdvalue)

apexhdvalue=Apex(HD111)
avgapxhdvalue=float(sum(apexhdvalue))/len(apexhdvalue)
print('hd apex value==',avgapxhdvalue)
```

Steps in Segmentation

ROC computation for segmentation:

Each .txt file contain the total images after flattening where “true_xx.txt” contains the true values of the masks [0,1] and “predict_xx.txt” contains the predicted probability from the network. The dimension of the contents in a .txt file is "Image_Number x Rows x Columns"

```
fpr_fcn, tpr_fcn, thresholds_model =
roc_curve(np.loadtxt('true_labels.txt'),np.loadtxt('predict_labels
.txt'))

auc_model = roc_auc_score(np.loadtxt('true_labels.txt'),
                          np.loadtxt('predict_labels.txt'))
```



Steps in Segmentation

https://stackoverflow.com/questions/57830697/calculating-the-area-covered-by-the-objects-of-irregular-shapes-in-an-image

Surface area:
Compute the area of non-zero pixels and divides with total No of image height and width
Surface area=count(non_zero_pixels)/(width*height)


```
import cv2
import numpy as np
img = cv2.imread('img.jpg')
height = img.shape[0]
width = img.shape[1]
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY)
cv2.imshow("Mask", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
count = cv2.countNonZero(thresh)
area = count/(width*height)
print(area)
```

Steps in Segmentation

Surface area:
Compute the area of non-zero pixels and divides with total
No of image height and width
Surface area=count(non_zero_pixels)/(width*height)

calculate surface area of all test predicted mask produced by model
And segmentation mask produced manually

Two surface area vector obtained (predicted segmentation masks area, Gt segmentation masks area)

Predictedarea=1,…….n, where n is the length of test images

Gtarea=1,…….n, where n is the length of test images

We can used these two vectors in order to compute t-test, p-test or band-Altman plot between two surface area vectors

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

Image Segmentation: Tips and Tricks from 39 Kaggle Competitions

Posted April 7, 2020

External Data

- Use of the LUng Node Analysis Grand Challenge data because it contains detailed annotations from radiologists
- Use of the LIDC-IDRI data because it had radiologist descriptions of each tumor that they found
- Use Flickr CC, Wikipedia Commons datasets
- Use Human Protein Atlas Dataset
- Use IDRiD dataset

Data Exploration and Gaining insights

- Clustering of 3d segmentation with the 0.5 threshold
- Identify if there is a substantial difference in train/test label distributions

Preprocessing

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

Preprocessing

- Perform blob Detection using the Difference of Gaussian (DoG) method. Used the implementation available in skimage package.
- Use of patch-based inputs for training in order to reduce the time of training
- Use cudf for loading data instead of Pandas because it has a faster reader
- Ensure that all the images have the same orientation
- Apply contrast limited adaptive histogram equalization
- Use OpenCV for all general image preprocessing
- Employ automatic active learning and adding manual annotations
- Resize all images to the same resolution in order to apply the same model to scans of different thicknesses
- Convert scan images into normalized 3D numpy arrays
- Apply single Image Haze Removal using Dark Channel Prior
- Convert all data to Hounsfield units
- Find duplicate images using pair-wise correlation on RGBY
- Make labels more balanced by developing a sampler
- Apply pseudo labeling to test data in order to improve score
- Scale down images/masks to 320×480

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

Data Augmentations

- Use albumentations package for augmentations
- Apply random rotation by 90 degrees
- Use horizontal, vertical or both flips
- Attempt heavy geometric transformations: Elastic Transform, PerspectiveTransform, Piecewise Affine transforms, pincushion distortion
- Apply random HSV
- Use of loss-less augmentation for generalization to prevent loss of useful image information
- Apply channel shuffling
- Do data augmentation based on class frequency
- Apply gaussian noise
- Use lossless permutations of 3D images for data augmentation
- Rotate by a random angle from 0 to 45 degrees
- Scale by a random factor from 0.8 to 1.2
- Brightness changing
- Randomly change hue, saturation and value
- Apply D4 augmentations
- Contrast limited adaptive histogram equalization
- Use the AutoAugment augmentation strategy

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

Modeling

Architectures

- Use of a U-net based architecture. Adopted the concepts and applied them to 3D input tensors
- Employing automatic active learning and adding manual annotations
- The inception-ResNet v2 architecture for training features with different receptive fields
- Siamese networks with adversarial training
- *ResNet50*, *Xception*, *Inception ResNet v2* x 5 with Dense (FC) layer as the final layer
- Use of a global max-pooling layer which returns a fixed-length output no matter the input size
- Use of stacked dilated convolutions
- VoxelNet
- Replace plus sign in LinkNet skip connections with concat and conv1x1
- Generalized mean pooling
- Keras NASNetLarge to train the model from scratch using 224x224x3
- Use of the 3D convnet to slide over the images
- Imagenet-pre-trained ResNet152 as the feature extractor
- Replace the final fully-connected layers of ResNet by 3 fully connected layers with dropout
- Use ConvTranspose in the decoder
- Applying the VGG baseline architecture

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

- Implementing the C3D network with adjusted receptive fields and a 64 unit bottleneck layer on the end of the network
- Use of UNet type architectures with pre-trained weights to improve convergence and performance of binary segmentation on 8-bit RGB input images
- LinkNet since it's fast and memory efficient
- MASKRCNN
- BN-Inception
- Fast Point R-CNN
- Seresnext
- UNet and Deeplabv3
- Faster RCNN
- SENet154
- ResNet152
- NASNet-A-Large
- EfficientNetB4
- ResNet101
- *GAPNet*
- PNASNet-5-Large
- Densenet121

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

Loss Functions

- Dice Coefficient because it works well with imbalanced data
- Weighted boundary loss whose aim is to reduce the distance between the predicted segmentation and the ground truth
- MultiLabelSoftMarginLoss that creates a criterion that optimizes a multi-label one-versus-all loss based on max-entropy, between input and target
- Balanced cross entropy (BCE) with logit loss that involves weighing the positive and negative examples by a certain coefficient
- Lovasz that performs direct optimization of the mean intersection-over-union loss in neural networks based on the convex Lovasz extension of sub-modular losses
- FocalLoss + Lovasz obtained by summing the Focal and Lovasz losses
- Arc margin loss that incorporates margin in order to maximise face class separability
- Npairs loss that computes the npairs loss between y_true and y_pred.
- A combination of BCE and Dice loss functions
- LSEP – a pairwise ranking that is is smooth everywhere and thus is easier to optimize
- Center loss that simultaneously learns a center for deep features of each class and penalizes the distances between the deep features and their corresponding class centers
- Ring Loss that augments standard loss functions such as Softmax

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

- Hard triplet loss that trains a network to embed features of the same class at the same time maximizing the embedding distance of different classes
- *1 + BCE – Dice* that involves subtracting the BCE and DICE losses then adding 1
- Binary cross-entropy – log(dice) that is the binary cross-entropy minus the log of the dice loss
- Combinations of BCE, dice and focal
- Lovasz Loss that loss performs direct optimization of the mean intersection-over-union loss
- BCE + DICE -Dice loss is obtained by calculating smooth dice coefficient function
- Focal loss with Gamma 2 that is an improvement to the standard cross-entropy criterion
- BCE + DICE + Focal – this is basically a summation of the three loss functions
- Active Contour Loss that incorporates the area and size information and integrates the information in a dense deep learning model
- 1024 * BCE(results, masks) + BCE(cls, cls_target)
- Focal + kappa – Kappa is a loss function for multi-class classification of ordinal data in deep learning. In this case we sum it and the focal loss
- ArcFaceLoss—Additive Angular Margin Loss for Deep Face Recognition
- soft Dice trained on positives only – Soft Dice uses predicted probabilities
- 2.7 * BCE(pred_mask, gt_mask) + 0.9 * DICE(pred_mask, gt_mask) + 0.1 * BCE(pred_empty,

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

Training tips

- Try different learning rates
- Try different batch sizes
- Use SDG with momentum with manual rate scheduling
- Too much augmentation will reduce the accuracy
- Train on image crops and predict on full images
- Use of Keras's ReduceLROnPlateau() to the learning rate
- Train without augmentation until plateau then apply soft and hard augmentation to some epochs
- Freeze all layers except the last one and use 1000 images from Stage1 for tuning
- Make labels more balanced by developing a sampler
- Use of class aware sampling
- Use dropout and augmentation while tuning the last layer
- Pseudo Labeling to improve score
- Use Adam reducing LR on plateau with patience 2–4
- Use Cyclic LR with SGD
- Reduce the learning rate by a factor of two if validation loss does not improve for two consecutive epochs

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

- Reduce the learning rate by a factor of two if validation loss does not improve for two consecutive epochs
- Repeat the worst batch out of 10 batches
- Train with default UNET
- Overlap tiles so that each edge pixel is covered twice
- Hyperparameter tuning: learning rate on training, non-maximum suppression and score threshold on inference
- Remove low bounding box with low confidence score
- Train different convolutional neural networks then build an ensemble
- Stop training when the F1 score is decreasing
- Differential learning rate with gradual reducing
- Train ANNs in a stacking way using 5 folds and 30 repeats
- Track of your experiments using Neptune.

Evaluation and cross-validation

- Split on non-uniform stratified by classes
- Avoid overfitting by applying cross-validation while tuning the last layer
- 10-fold CV ensemble for classification

https://neptune.ai/blog/image-segmentation-tips-and-tricks-from-kaggle-competitions

- Avoid overfitting by applying cross-validation while tuning the last layer
- 10-fold CV ensemble for classification
- Combination of 5 10-fold CV ensembles for detection
- Sklearn's stratified K fold function
- 5 KFold Cross-Validation
- Adversarial Validation & Weighting

Ensembling methods

- Use simple majority voting for ensemble
- XGBoost on the max malignancy at 3 zoom levels, the z-location and the amount of strange tissue
- LightGBM for models with too many classes. This was done for raw data features only.
- CatBoost for a second-layer model
- Training with 7 features for the gradient boosting classifier
- Use 'curriculum learning' to speed up model training. In this technique, models are first trained on simple samples then progressively moving to hard ones.
- Ensemble with ResNet50, InceptionV3, and InceptionResNetV2
- Ensemble method for object detection

Useful Libraries for Segmentation (initial work and basic)

https://github.com/qubvel/segmentation_models

https://github.com/divamgupta/image-segmentation-keras

https://pypi.org/project/segmentation-models-pytorch/