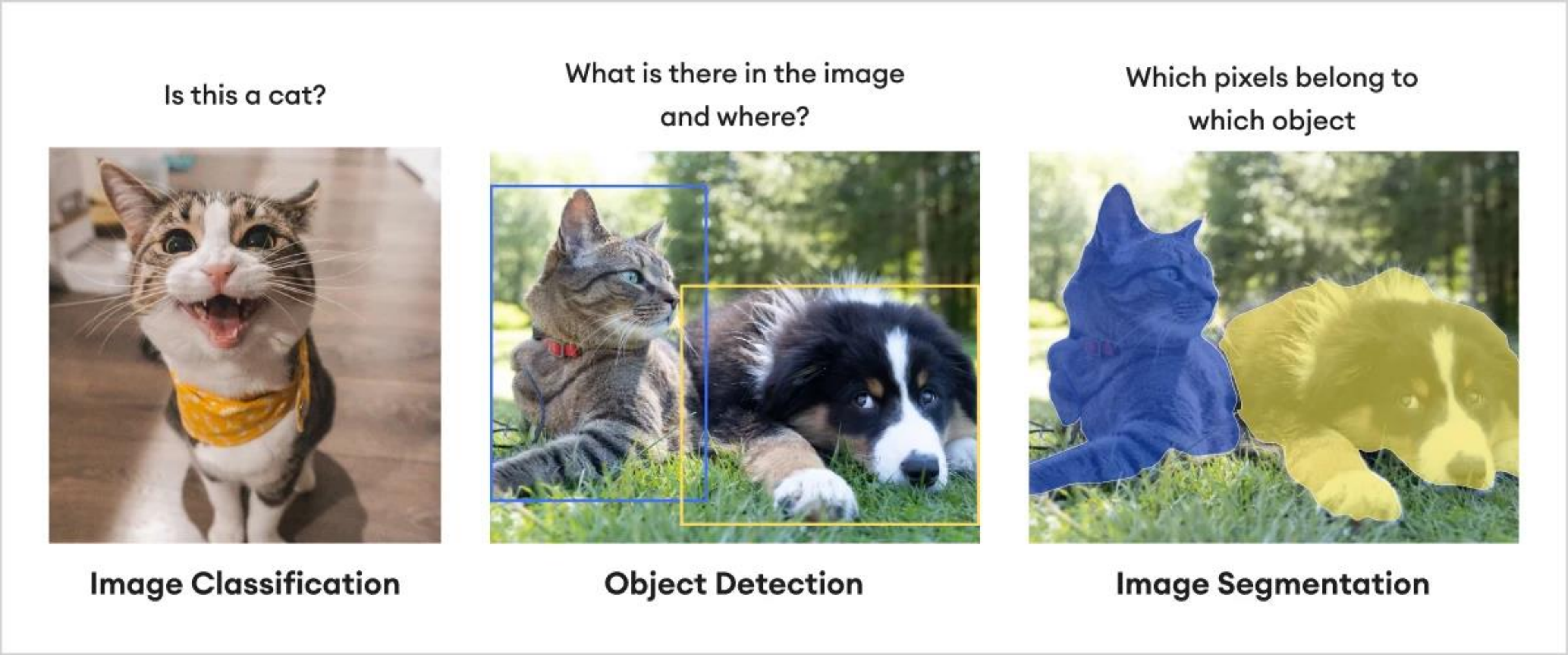
Image segmentation using deep learning


Is this a cat?
What is there in the image and where?
Which pixels belong to which object
Image Classification
Object Detection
Image Segmentation

What is image segmentation?

Computer vision has evolved to not only detect objects in a given visual and label them, but also to **outline their entire form in a precise manner**, regardless of the unique shape, all due to image segmentation.

With that, we're able to **expand the scope of accuracy and precision** significantly among image annotation tasks and apply it to innovative technological advancements.

Such instances include

- medical imaging
- agriculture
- satellite images
- AI for autonomous vehicles
- and more

What is image segmentation? - cont.

Image segmentation is a vital computer vision task that expands on the premise of object detection.

Image segmentation is essentially segmenting an image into fragments and assigning a label to each of those.

This occurs on a **pixel level** to define the precise outline of an object within its frame and class.

Those outlines — otherwise known as the output — are highlighted with either one or more colors, depending on the type of segmentation.

To streamline image segmentation in machine learning:

- the system is trained with already segmented sets of data, which is manually segmented or open-source datasets to accurately identify segments
- segment new set of images.

[https://www.superannotate.com/blog/image-segmentation-for-machine-learning]

Types of image segmentation

- semantic segmentation
- instance segmentation
- panoptic segmentation.

Foundational definition of image segmentation:

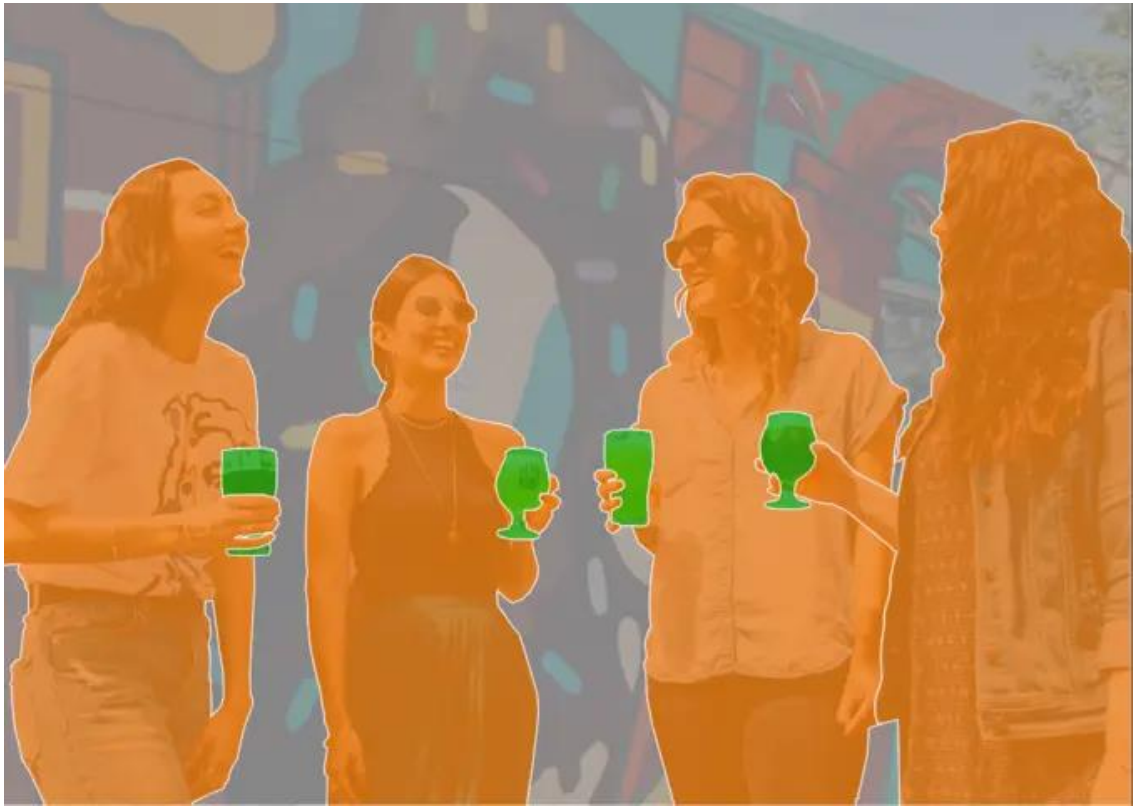
identification, grouping, and labeling of pixels in visuals that form a whole object.

During those tasks for the given image, a so-called **segmentation mask** is created, which is the **label of all the pixels** in the image.

Semantic segmentation

Segmentation masks represent fully labeled images.
All pixels in the image should belong to some category, whether they belong to the same instance or not.

The semantic segmentation is used, for example, to **distinguish between the background and category**.



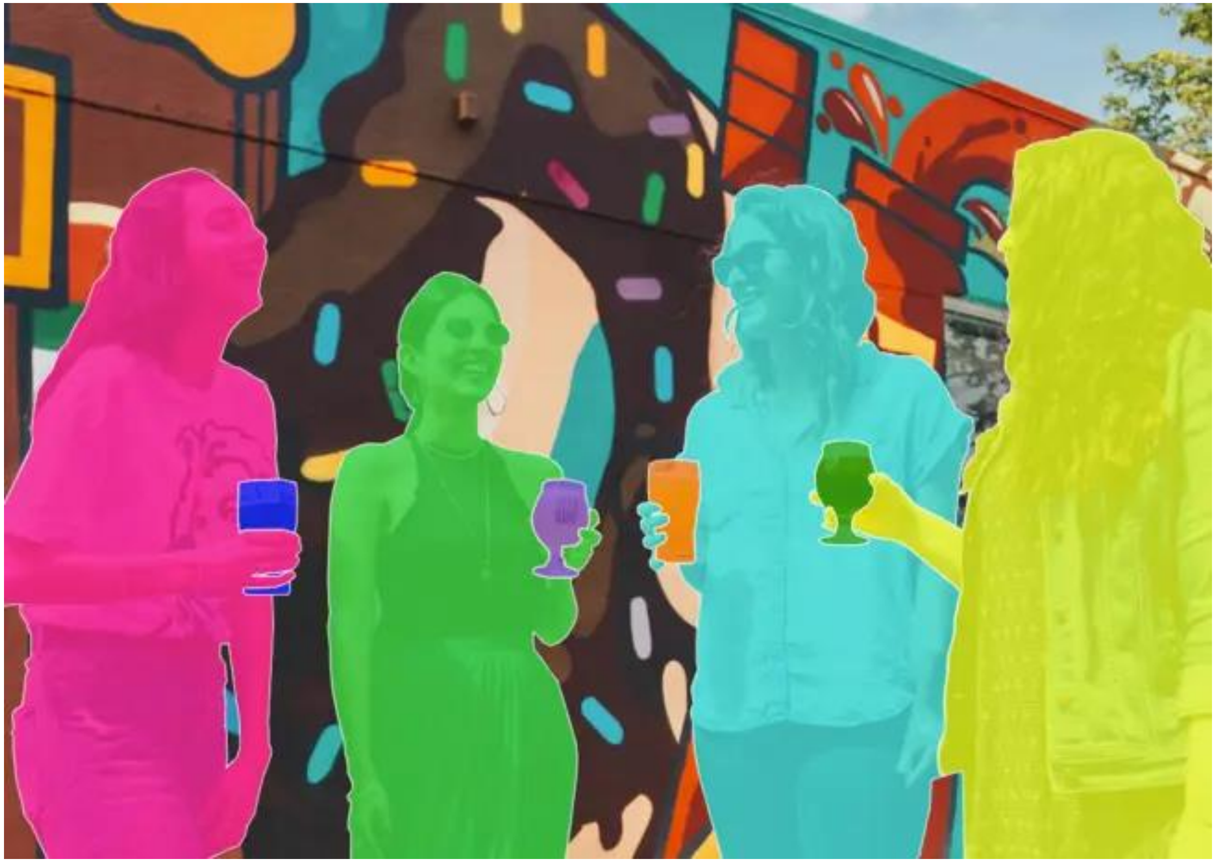
All pixels with the same category are represented as a single segment.
If two pixels are categorized as "people" then segmentation mask pixel values will be the same for both.

- all persons in the image have the same (orange) color label
- all glasses/cups have the same (green) color label
- grey color is labeled as background.

Instance segmentation

Objects with their bounds are detected in the image.
Each new object is labeled as a different instance, even within the same category.

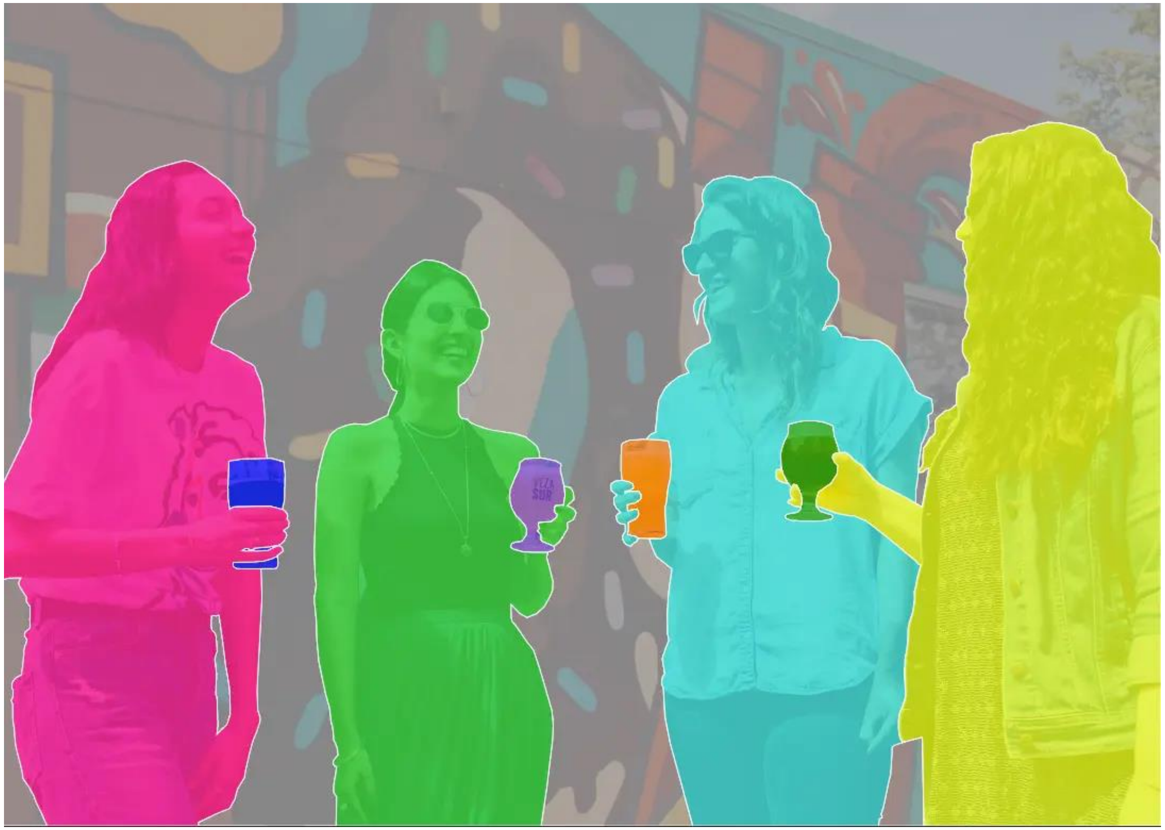
Image segmentation techniques are used, for example, to identify the **exact shape of each instance** in the image.



All peoples are human, but all of them are different individuals with varying heights, races, ages, gender, and so on.
During instance segmentation tasks, all of them should be identified as general category "people", but as different instances.
During the training, each of the segments labeled as the "people" category will be possible to select as a separate instance for neural network training.
All persons in the image have different color labels and all glasses/cups also have different color labels.

Panoptic segmentation

A combination of instance segmentation and semantic segmentation



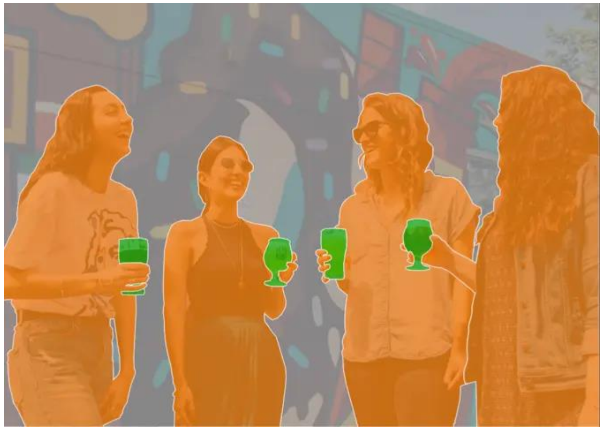
The entire image is labeled, and pixels from different instances should have different values even if they have the same category.

Panoptic segmentation is a complex computer vision task that solves both instance segmentation and semantic segmentation problems together.

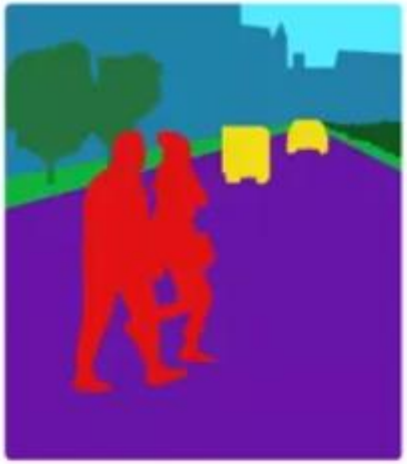
It is widely used, for example, in autonomous vehicles because the cameras on them should provide complete information around them.

Types of image segmentation

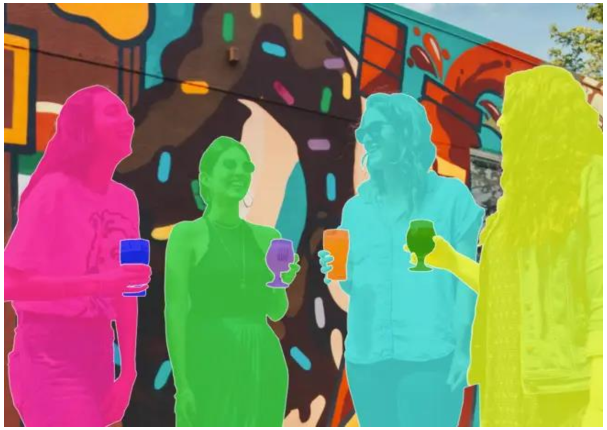
Semantic segmentation



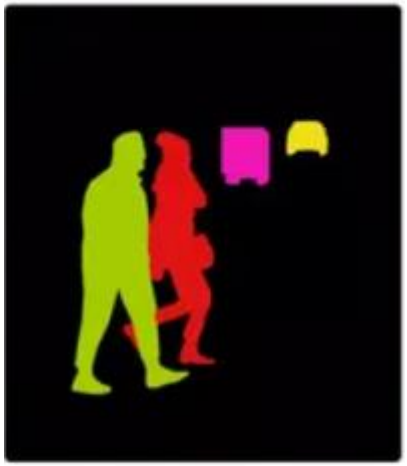
All pixels with the same category are represented as a single segment



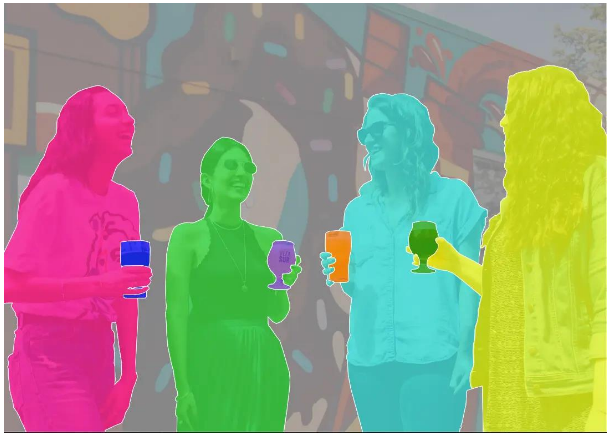
Instance segmentation



Each object is labeled as a different instance, even within the same category



Panoptic segmentation



The entire image is labeled. Pixels from different instances should have different values even if they have the same category

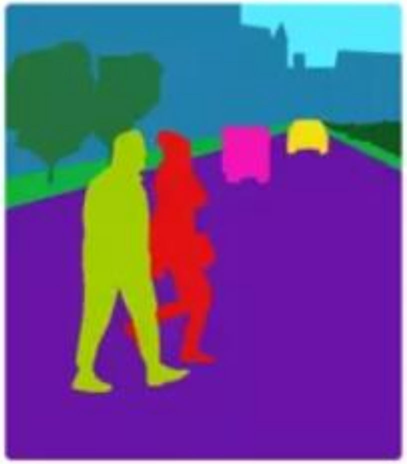


Image segmentation vs object detection

- **Image classification** — A single class is assigned to an image, commonly to the main object that is portrayed in the image. If an image contains a cat, the image will be classified as a 'cat'. With image classification, we do not know the precise location of the cat in the image nor can identify its limits on the visual compared to object localization, detection, or segmentation.

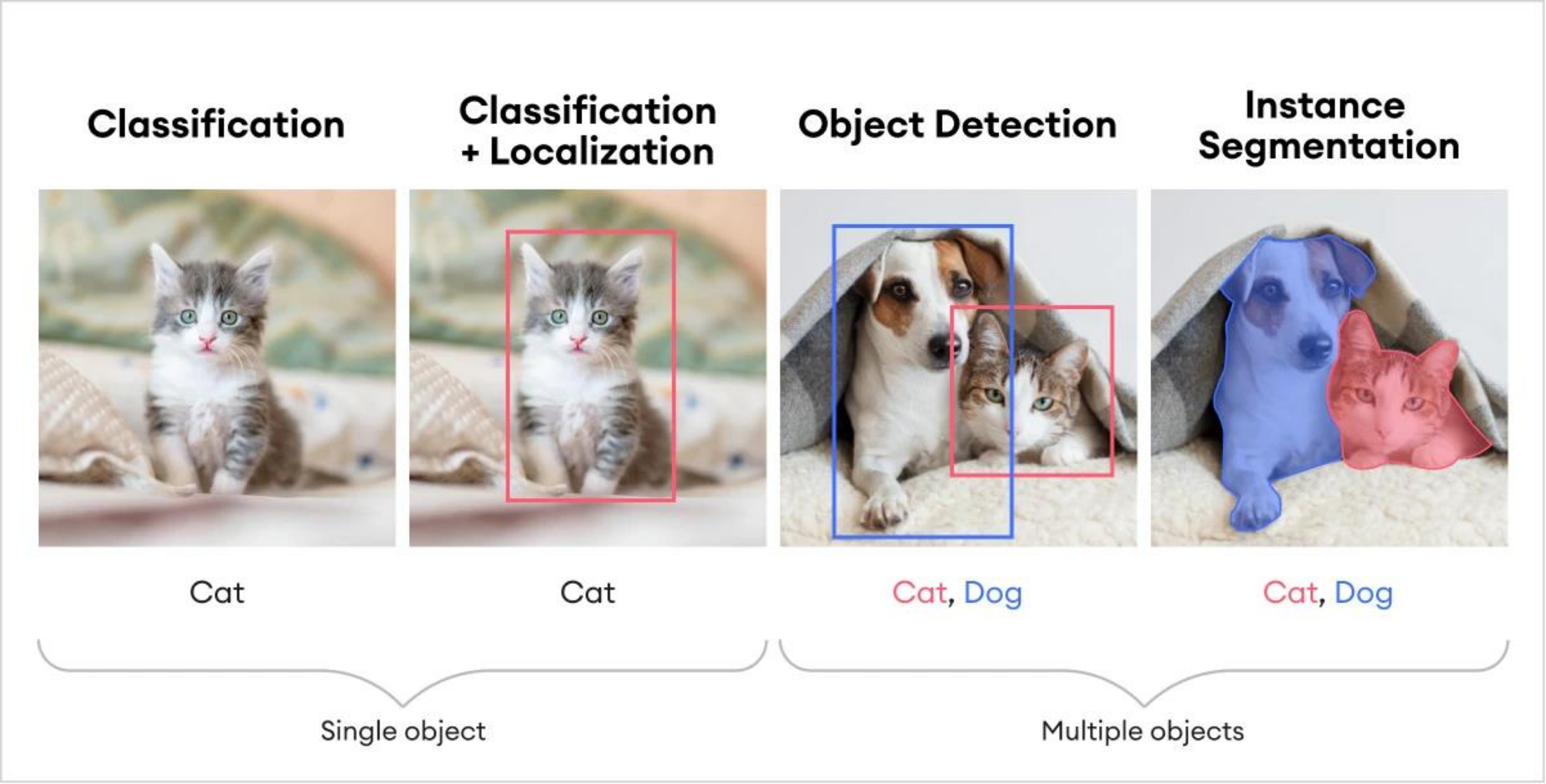
- **Object detection** — The objects within an image or a video are detected, marked with a bounding box, and then labeled.

The **primary difference** between object detection and image segmentation is the **final output.**

- With object **detection**, the key signifier is the **bounding box** that draws a square or rectangle around the limits of each object.
- With image **segmentation**, the **entire outline of the object** is considered, without the presence of any background content.

- **Localization** — With image/object localization, we are able to identify the location of the main subject of an image. However, image localization does not typically assign classes to the localized object as it considers the main subject instead of all of the present objects in a given frame.

Image segmentation vs object detection



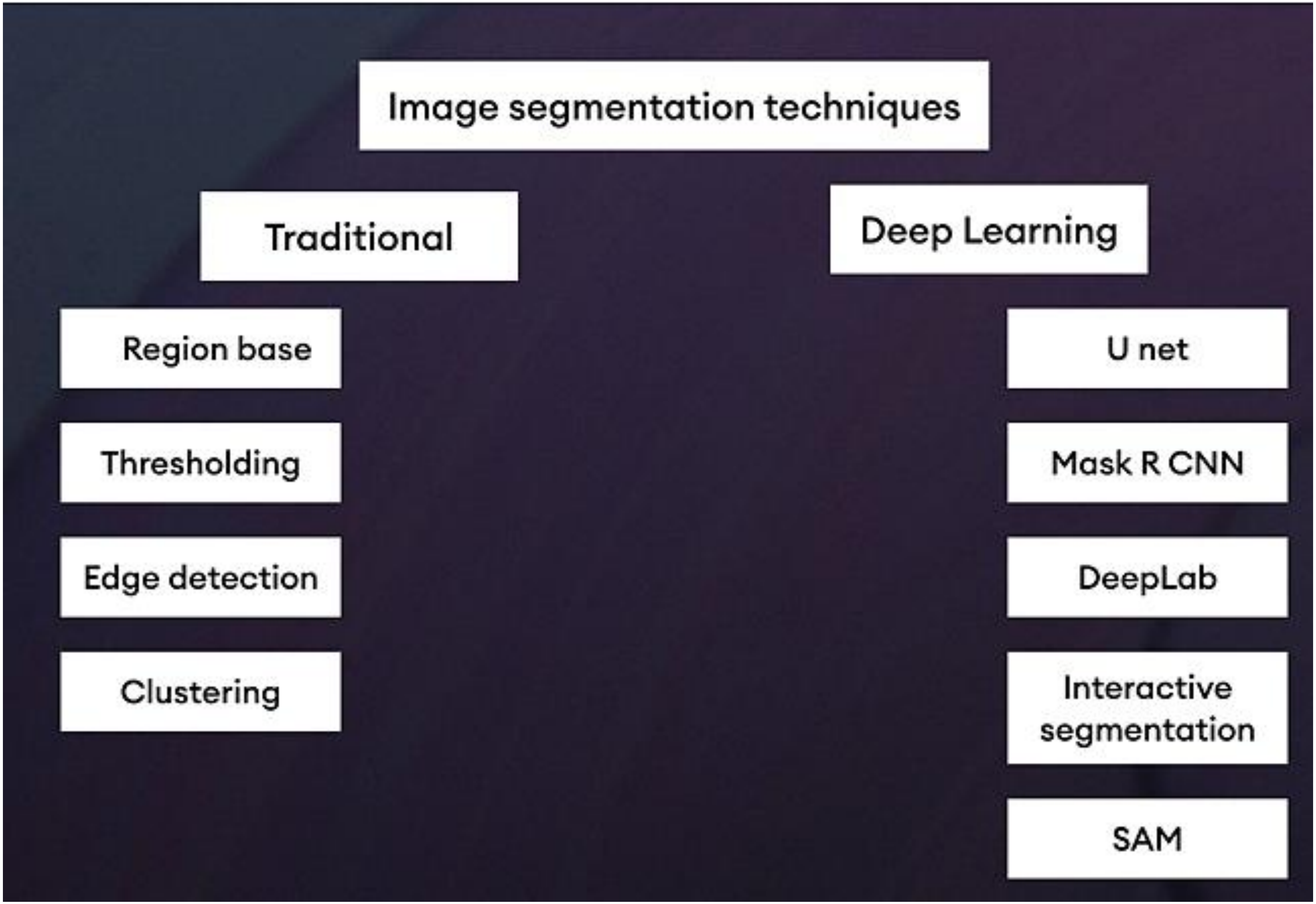


Image segmentation techniques

Traditional

- Region base
- Thresholding
- Edge detection
- Clustering

Deep Learning

- U net
- Mask R CNN
- DeepLab
- Interactive segmentation
- SAM

Image segmentation with deep learning

Image segmentation or **semantic segmentation** is the task of assigning **a class to each pixel** in an image.
Models are trained using **segmentation maps as target variables**.

Image segmentation with deep learning is arguably becoming the most accurate approach for the task in recent years.

To achieve image segmentation for machine learning, it is unnecessary to delve too deeply into complex deep learning segmentation architectures, it is more beneficial to understand the basic principles of how they work.

Key elements of the image segmentation architecture consist of the **encoder and decoder**.

Through this, parts of the image are extracted via filters in the convolution and pooling layers and then a final output is received with a segmentation mask.

This is also known as the **convolutional encoder-decoder architecture**.

The UNET architecture was developed to address the limitations and overcome the challenges faced by traditional approaches to image segmentation.

End-to-End Learning: UNET takes an end-to-end learning technique, which means it learns to segment images directly from input-output pairs without user annotation. UNET can automatically extract key features and execute accurate segmentation by training on a large labeled dataset, removing the need for labor-intensive manual annotation.

Fully Convolutional Architecture: UNET is based on a fully convolutional architecture, which implies that it is entirely made up of convolutional layers and does not include any fully connected layers. This architecture enables UNET to function on input images of any size, increasing its flexibility and adaptability to various segmentation tasks and input variations.

U-shaped Architecture with Skip Connections: The network's characteristic architecture includes an encoding path (contracting path) and a decoding path (expanding path), allowing it to collect local information and global context. Skip connections bridge the gap between the encoding and decoding paths, maintaining critical information from previous layers and allowing for more precise segmentation.

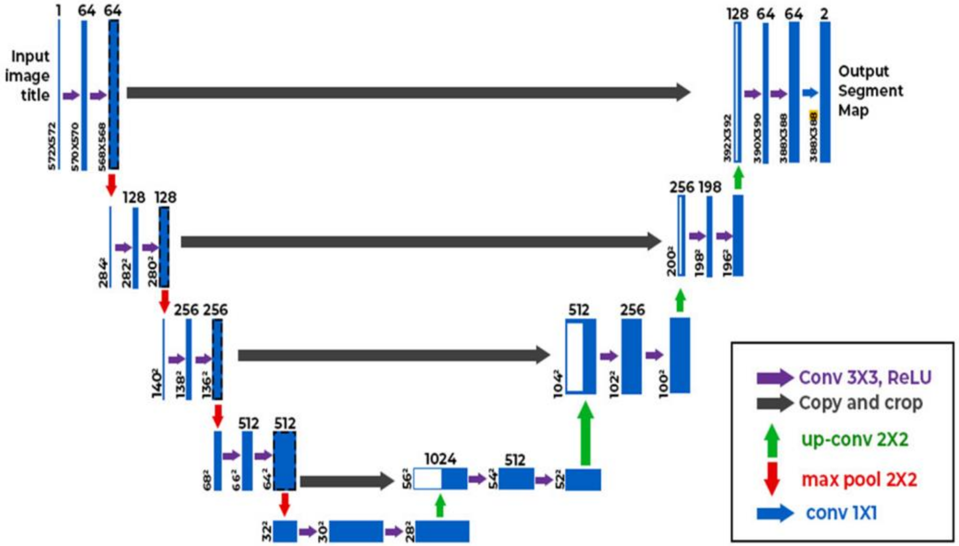
https://www.analyticsvidhya.com/blog/2023/08/unet-architecture-mastering-image-segmentation/#h-image-segmentation

U-Net – semantic segmenatation

U-Net model resembles a 'U' shape when the architecture is visualized.

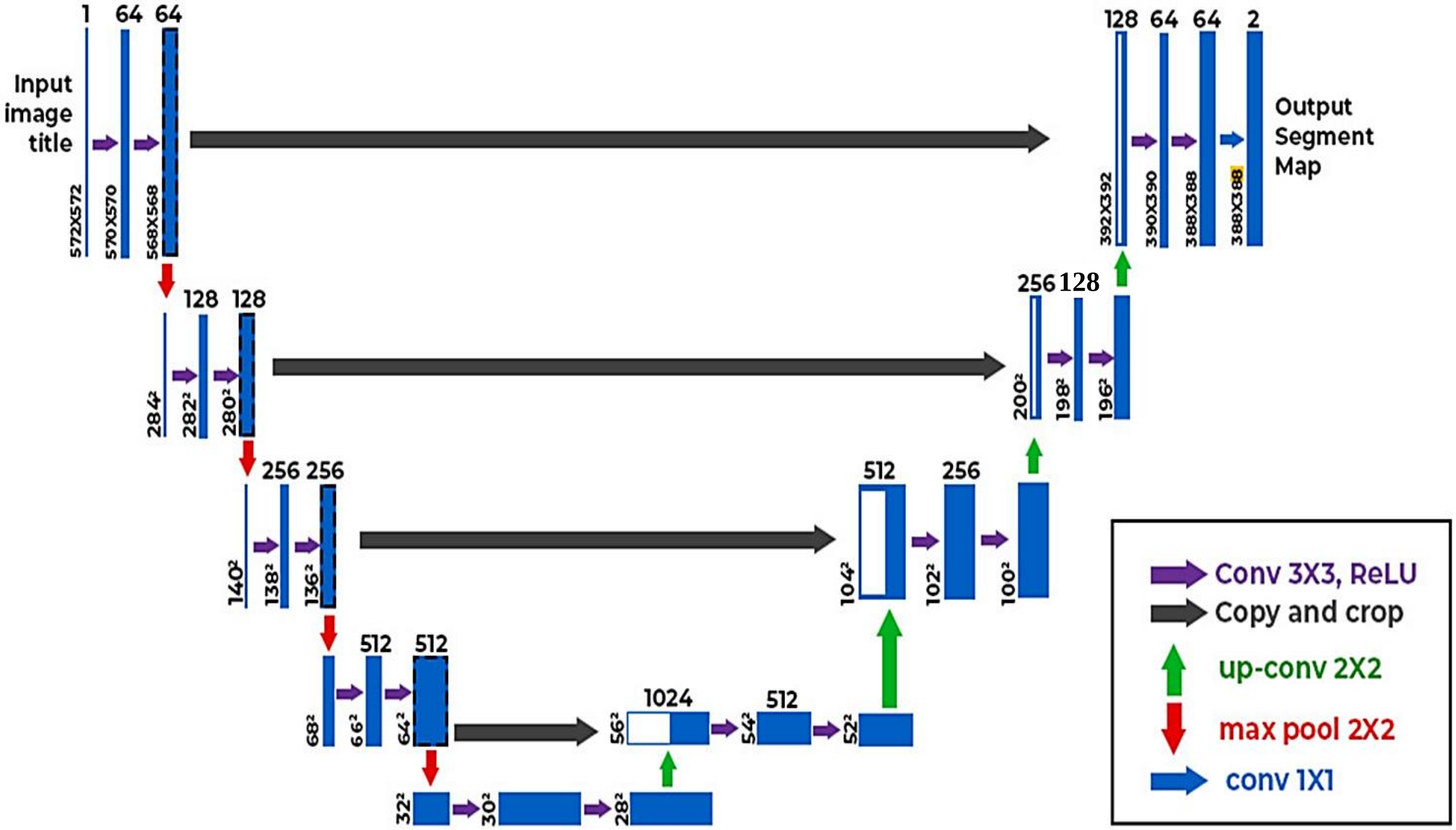
❑ It is composed of two parts:

- ➢ Downsampling (the contracting path)
- ➢ Upnsampling (expanding part)



The significance of U-Net is the accuracy and speed it achieves for image segmentation by repurposing the same feature maps that are initially used to expand a vector into a fully segmented output image.

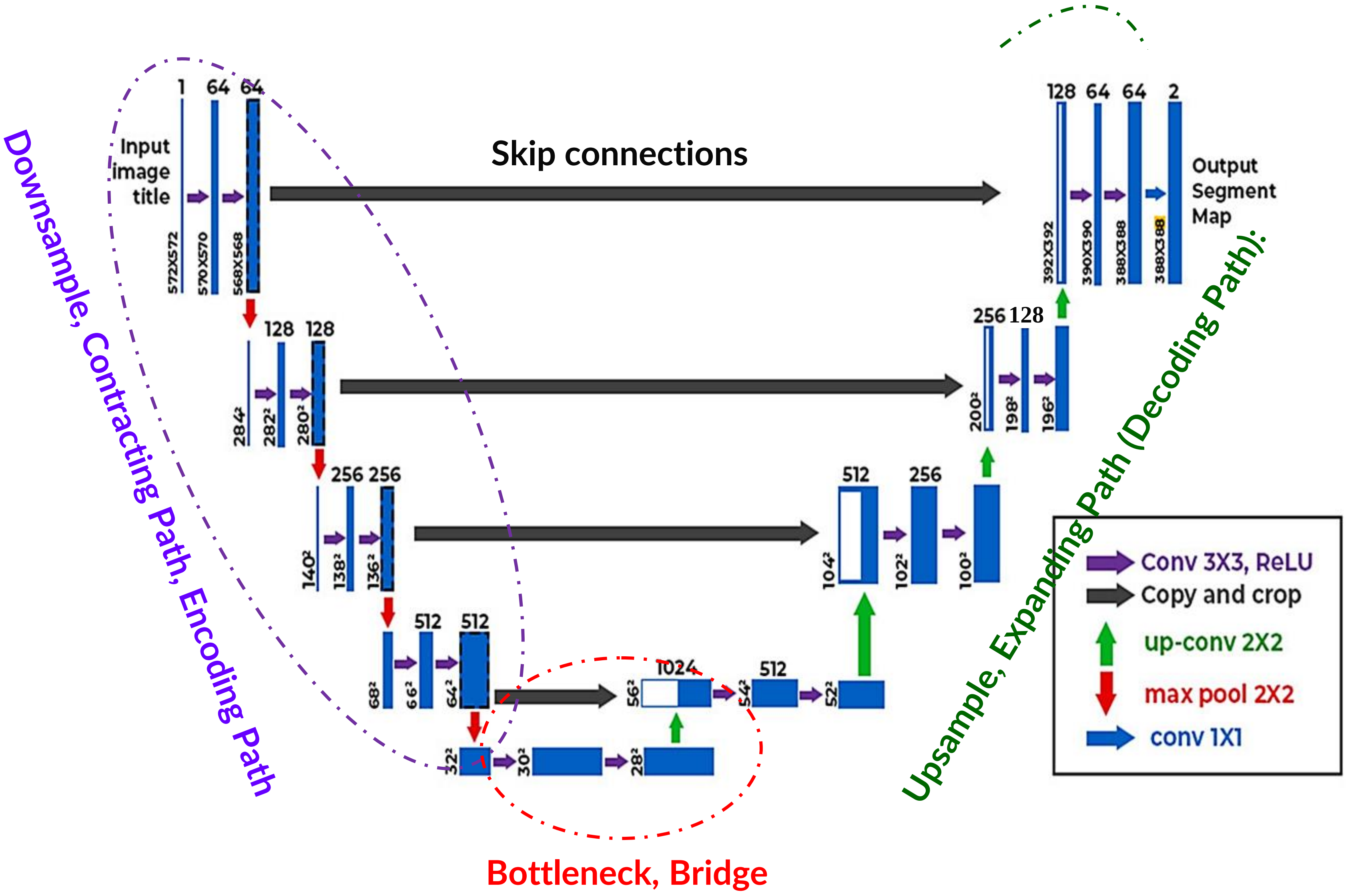
U-Net Architecture



Example: U-Net network converts a grayscale input image of size 572×572×1 into a binary segmented output map of size 388×388×2

https://www.geeksforgeeks.org/u-net-architecture-explained/

Olaf Ronneberger, Philipp Fischer, and Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, 18 May 2015, https://arxiv.org/pdf/1505.04597.pdf?ref=machinelearningnuggets.com



Skip connections

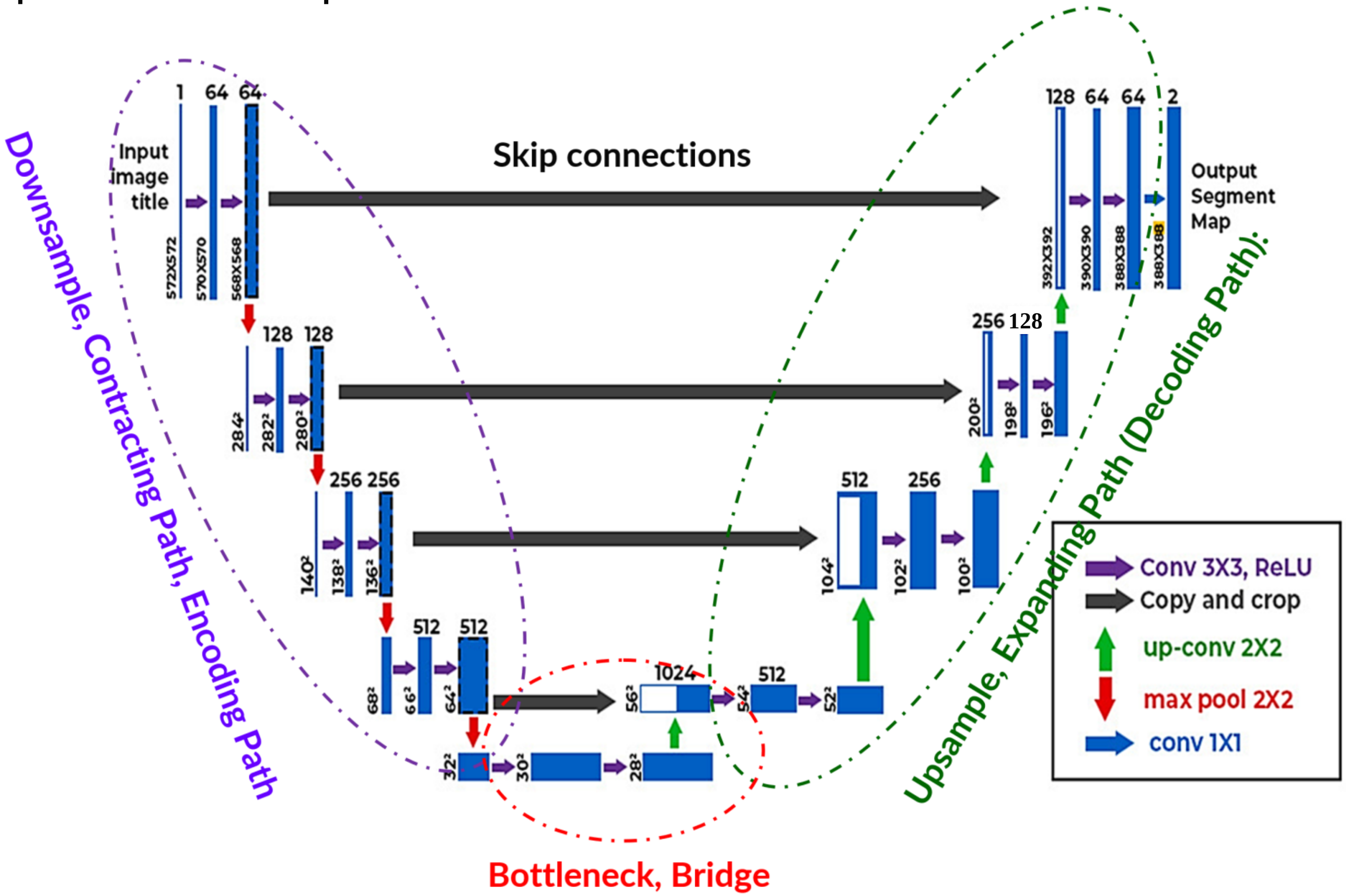
Downsample, Contracting Path, Encoding Path

Upsample, Expanding Path (Decoding Path):

Bottleneck, Bridge

During the **contracting path**, the input image is progressively reduced in height and width but increased in the number of channels. This increase in channels allows the network to capture high-level features as it progresses down the path.

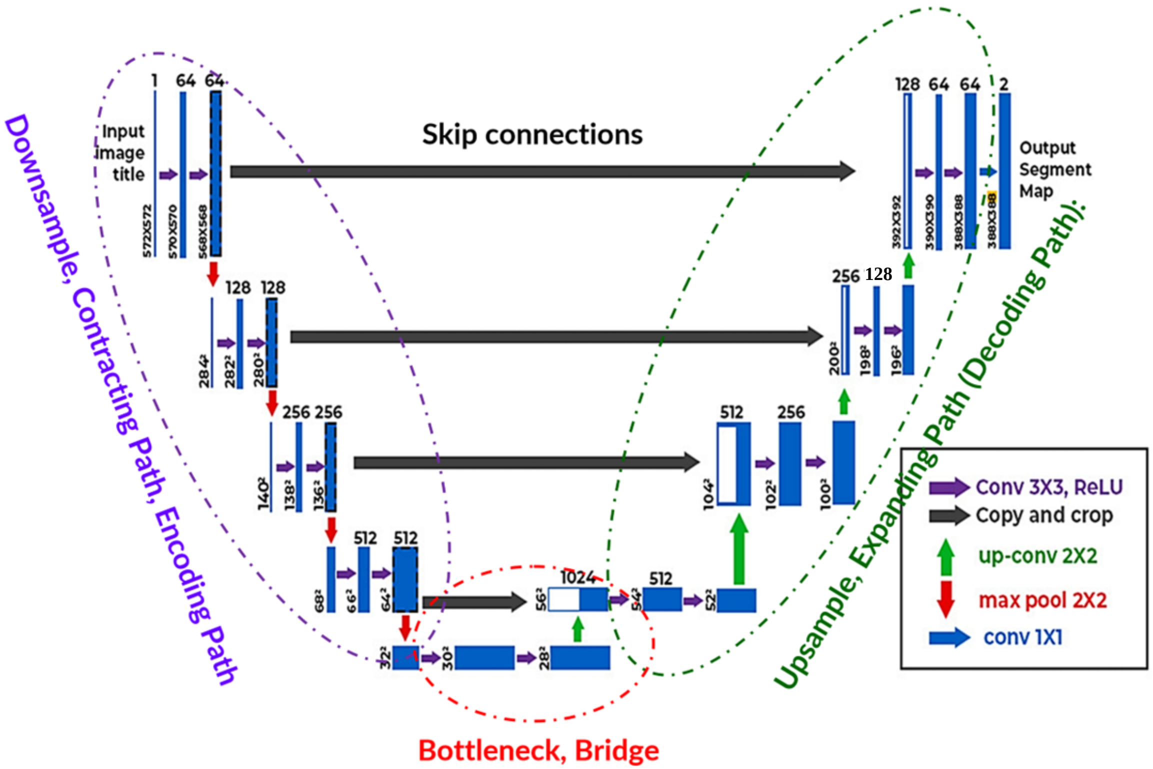
At the bottleneck, a final convolution operation is performed to generate a 30×30×1024 shaped feature map.



The expansive path then takes the feature map from the bottleneck and converts it back into an image of the same size as the original input.
This is done using upsampling layers, which increase the spatial resolution of the feature map while reducing the number of channels.

Upsampling Layers (Transposed Convolutions)

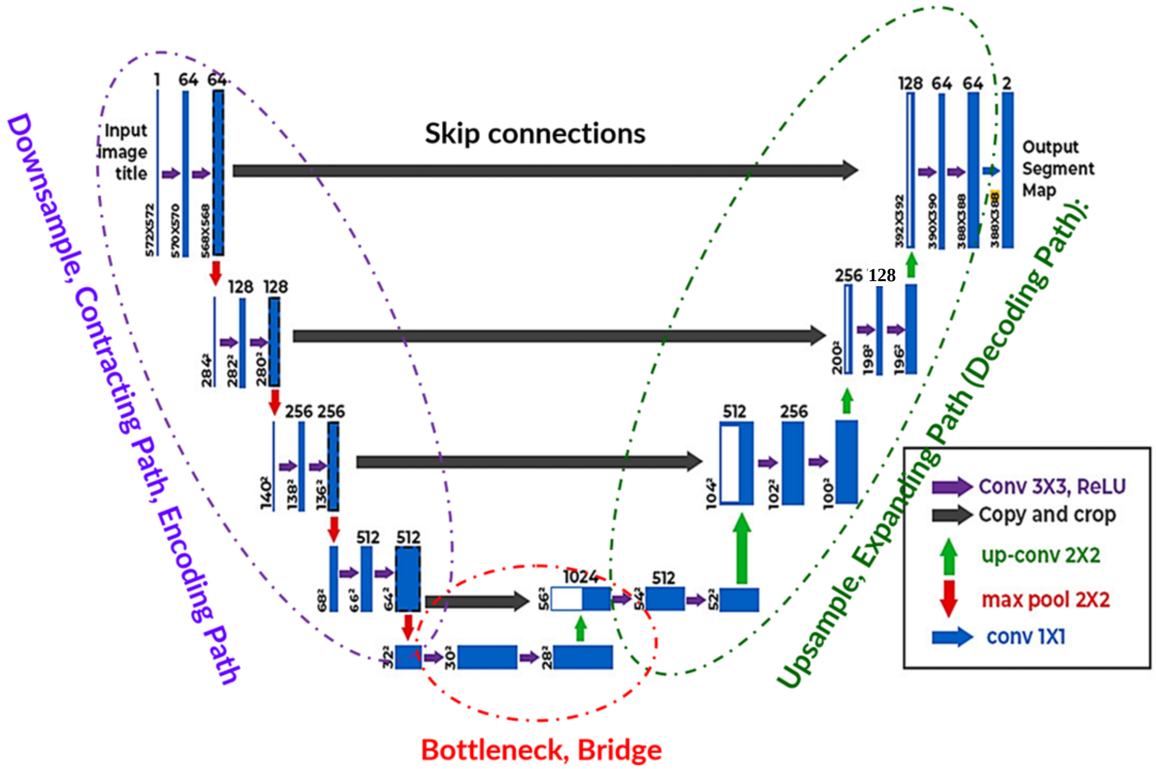
Transposed convolutions are essentially the opposite of regular convolutions. They enhance spatial dimensions rather than decrease them, allowing for upsampling.



By constructing a sparse kernel and applying it to the input feature map, transposed convolutions learn to upsample the feature maps. The network learns to fill in the gaps between the current spatial locations during this process, thus boosting the resolution of the feature maps.

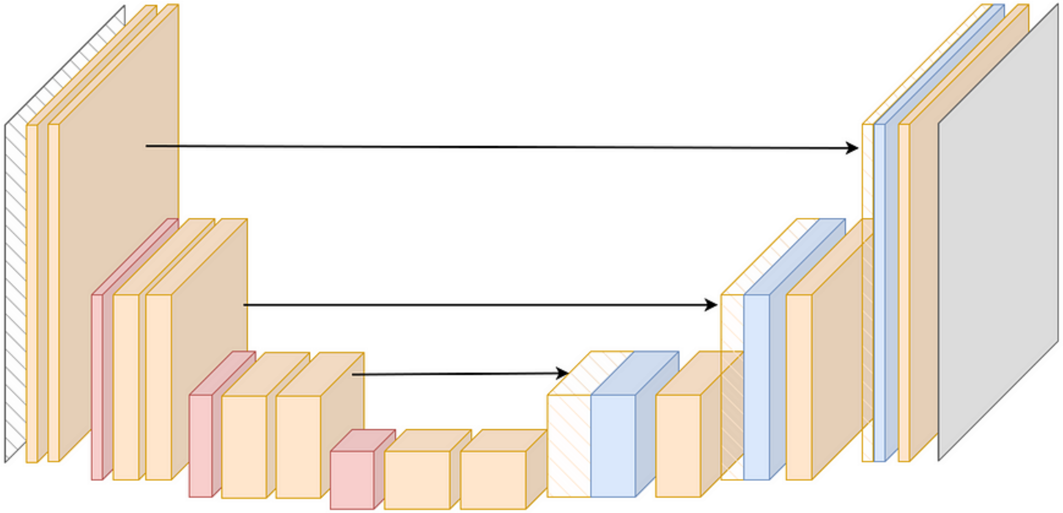
The **skip connections** from the contracting path are used to help the decoder layers locate and refine the features in the image.

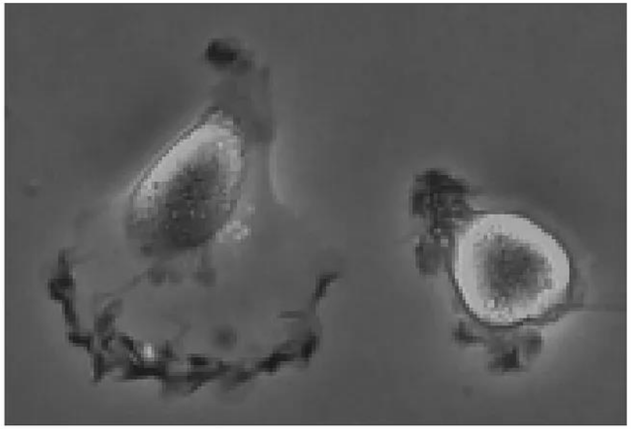
Concatenation is commonly used to implement skip connections in UNET. The feature maps from the encoding path are concatenated with the upsampled feature maps from the decoding path during the upsampling procedure. This concatenation allows the network to incorporate multi-scale information for appropriate segmentation, exploiting high-level context and low-level features.



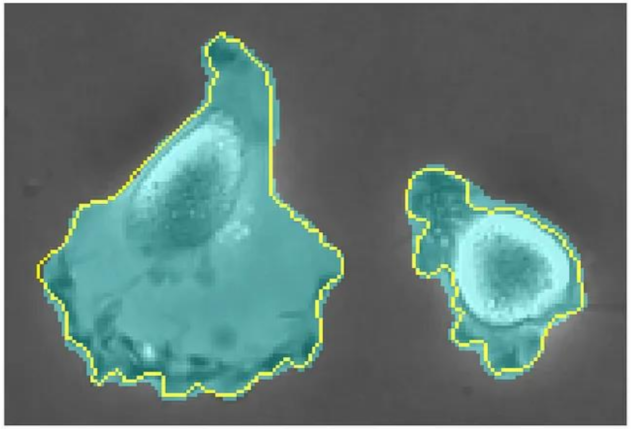
Finally, each pixel in the output image represents a label that corresponds to a particular object or class in the input image.

In our case, the output map is a binary segmentation map where each pixel represents a foreground or background region.





original image



binary segmentation map

https://towardsdatascience.com/u-net-explained-understanding-its-image-segmentation-architecture-56e4842e313a

The map separates the image into cell and non-cell pixels.

Loss Function in UNET

It is critical to select an appropriate loss function while training UNET and optimizing its parameters for picture segmentation tasks. UNET frequently employs segmentation-friendly loss functions such as the Dice coefficient or cross-entropy loss.

Dice Coefficient Loss

The Dice coefficient is a similarity statistic that calculates the overlap between the anticipated and true segmentation masks. The Dice coefficient loss, or soft Dice loss, is calculated by subtracting one from the Dice coefficient. When the anticipated and ground truth masks align well, the loss minimizes, resulting in a higher Dice coefficient.

The Dice coefficient loss is especially effective for unbalanced datasets in which the background class has many pixels. By penalizing false positives and false negatives, it promotes the network to divide both foreground and background regions accurately.

Loss Function in UNET

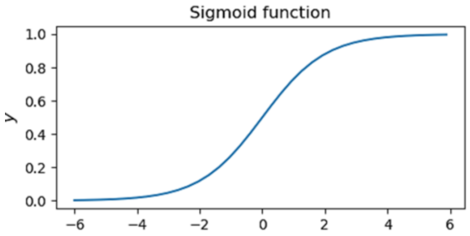
Cross-Entropy Loss

Cross-entropy loss function in image segmentation measures the dissimilarity between the predicted class probabilities and the ground truth labels.
Treat each pixel as an independent classification problem in image segmentation, and the cross-entropy loss is computed pixel-wise.

The cross-entropy loss encourages the network to assign high probabilities to the correct class labels for each pixel. It penalizes deviations from the ground truth, promoting accurate segmentation results. This loss function is effective when the foreground and background classes are balanced or when multiple classes are involved in the segmentation task.

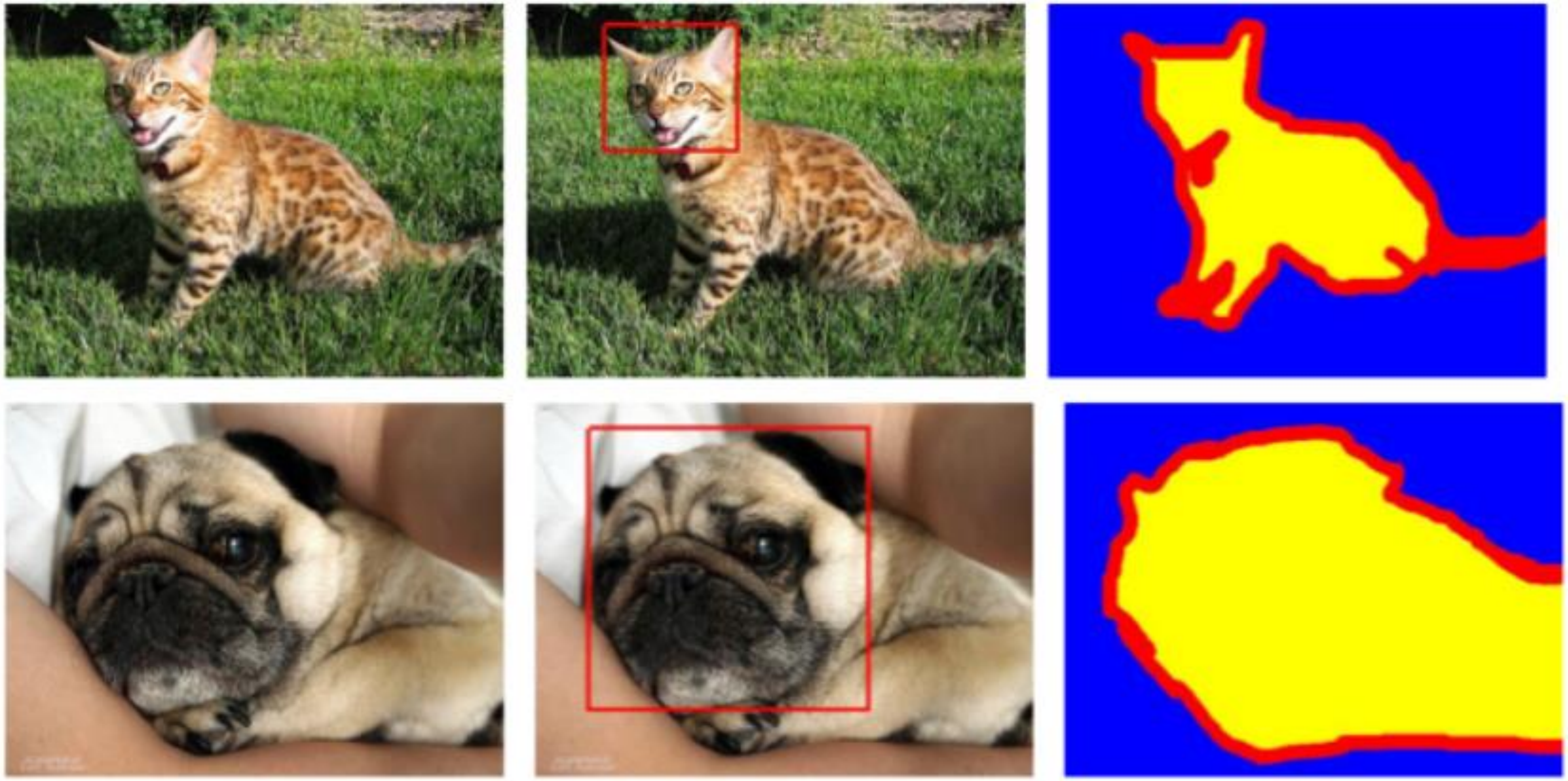
The choice between the Dice coefficient loss and cross-entropy loss depends on the segmentation task's specific requirements and the dataset's characteristics. Both loss functions have advantages and can be combined or customized based on specific needs.

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$$



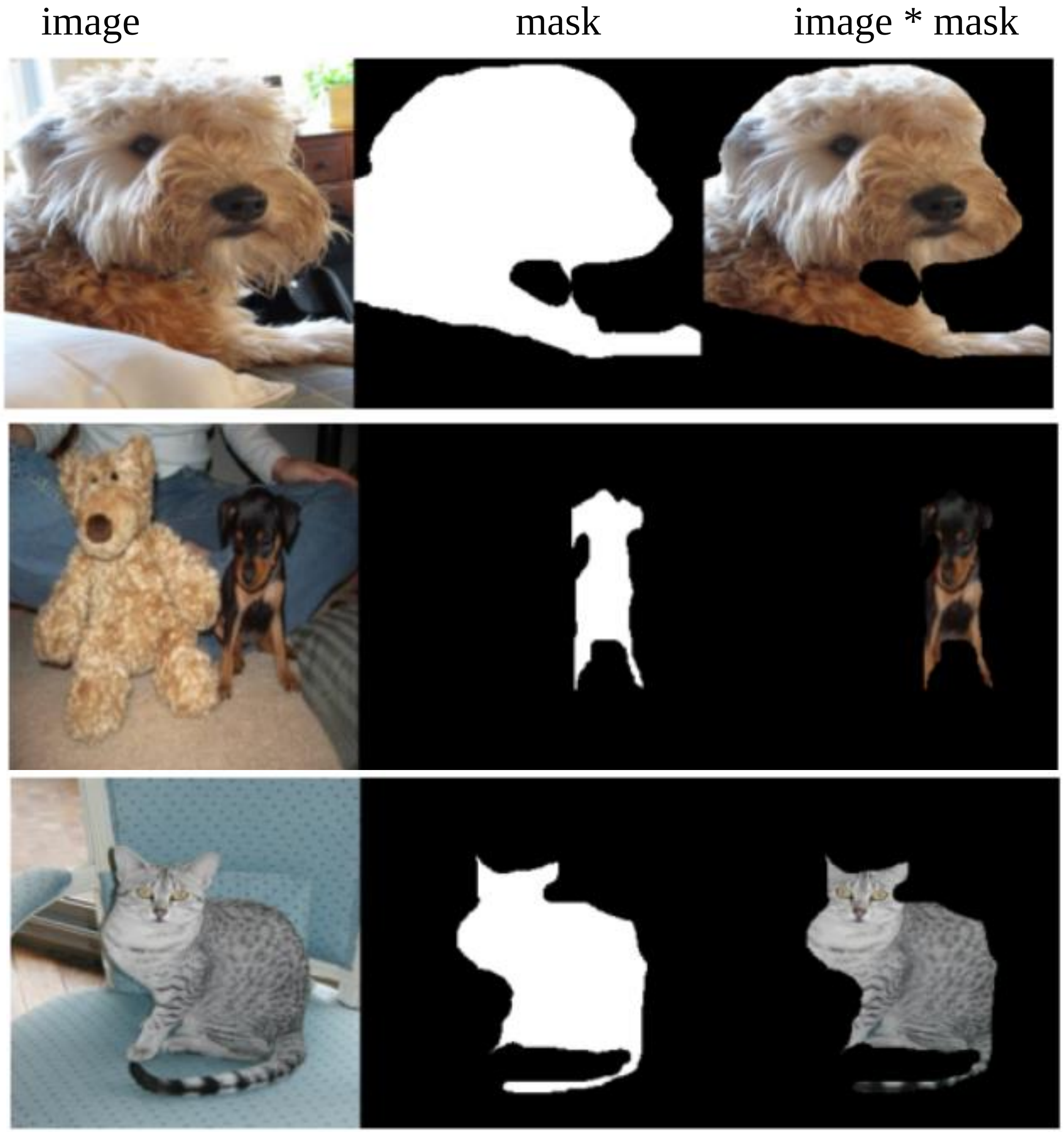
Case study
Instance segmentation using U-net

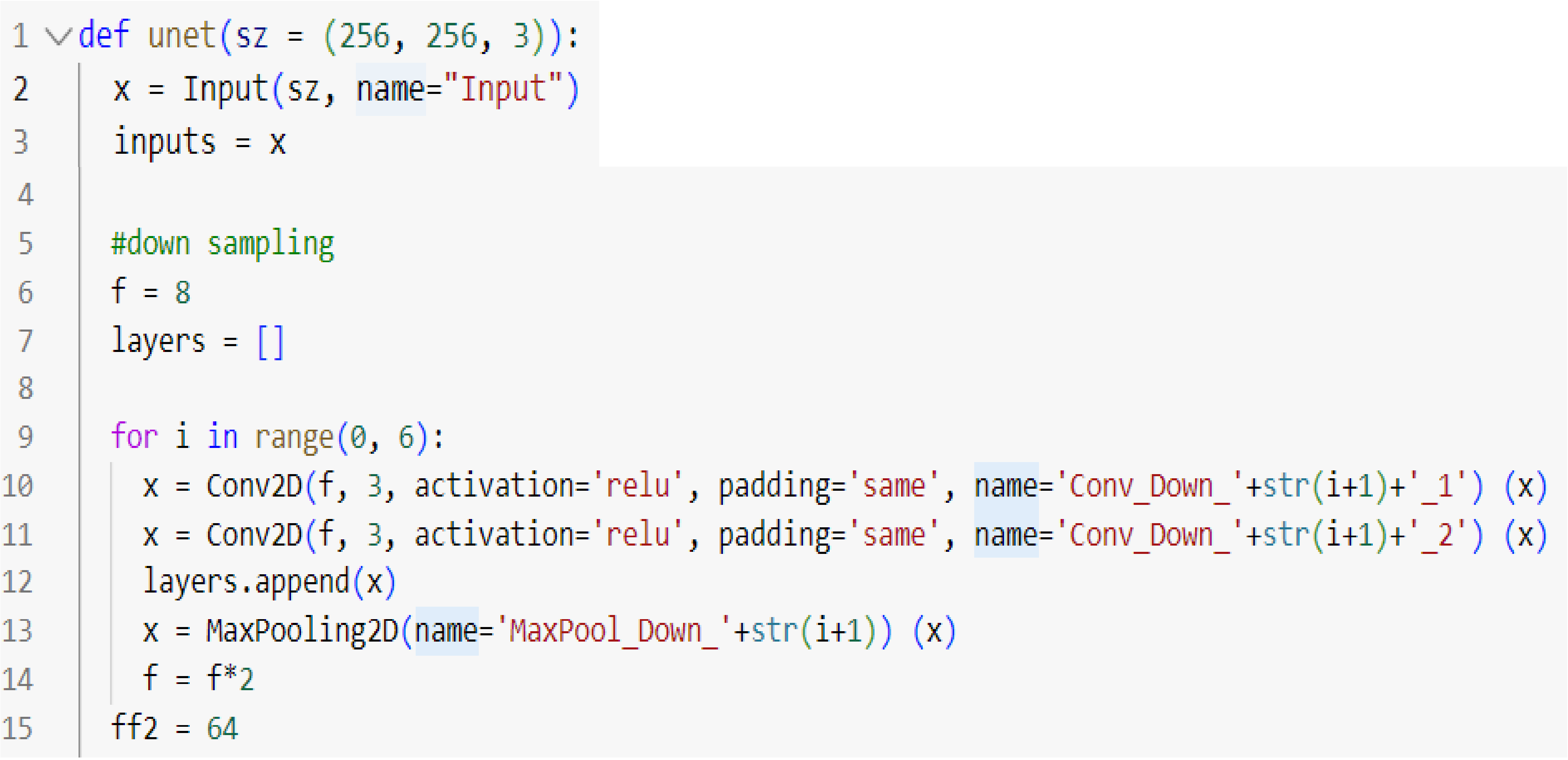
The Oxford-IIIT Pet Dataset. It contains 37 classes of dogs and cats with around 200 images per each class. The dataset contains labels as bounding boxes and segmentation masks. The total number of images in the dataset is a little more than 7K.



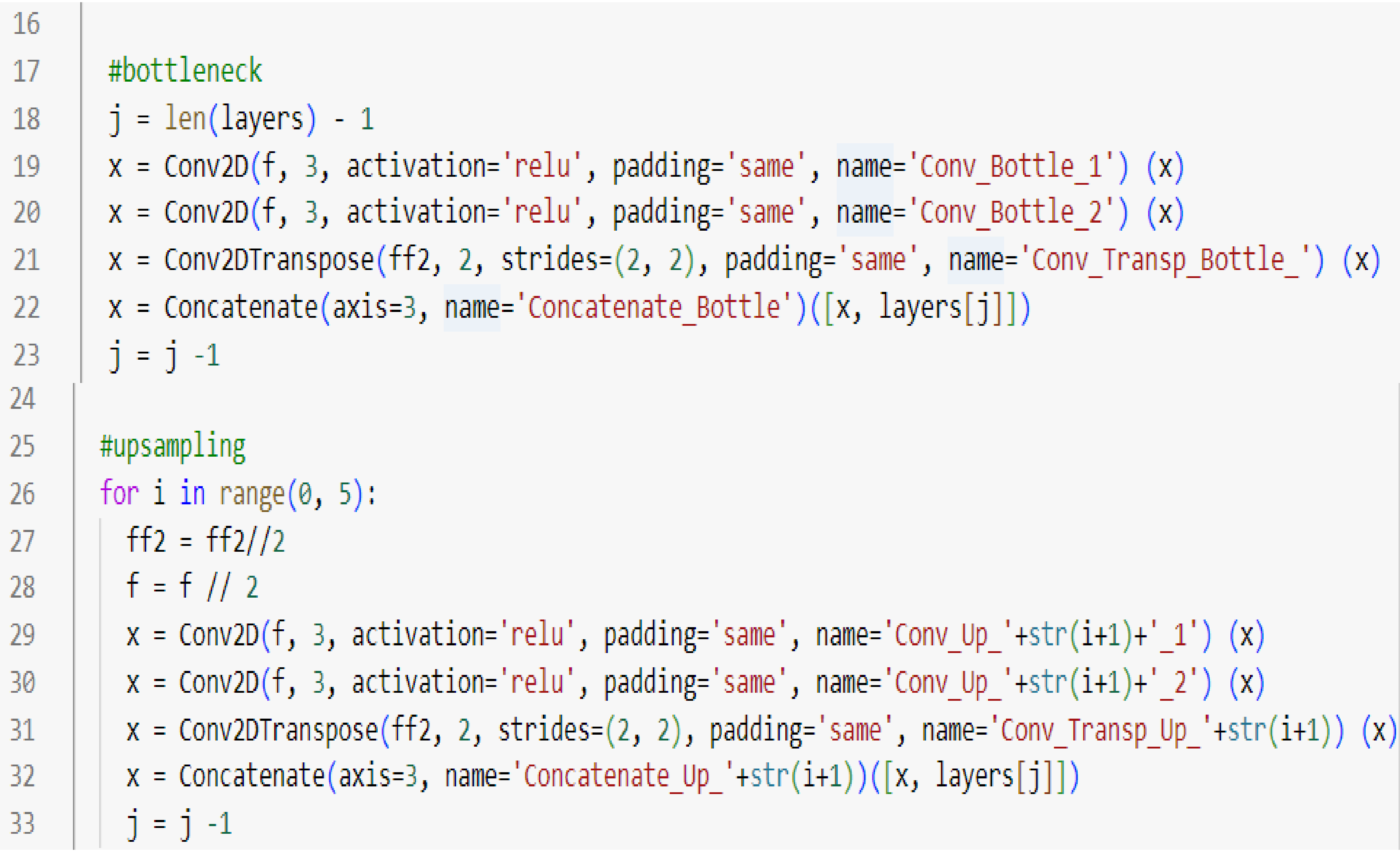
There are two folders:

- images which contains the raw images
- annotation which contains the masks as a binary folder image.

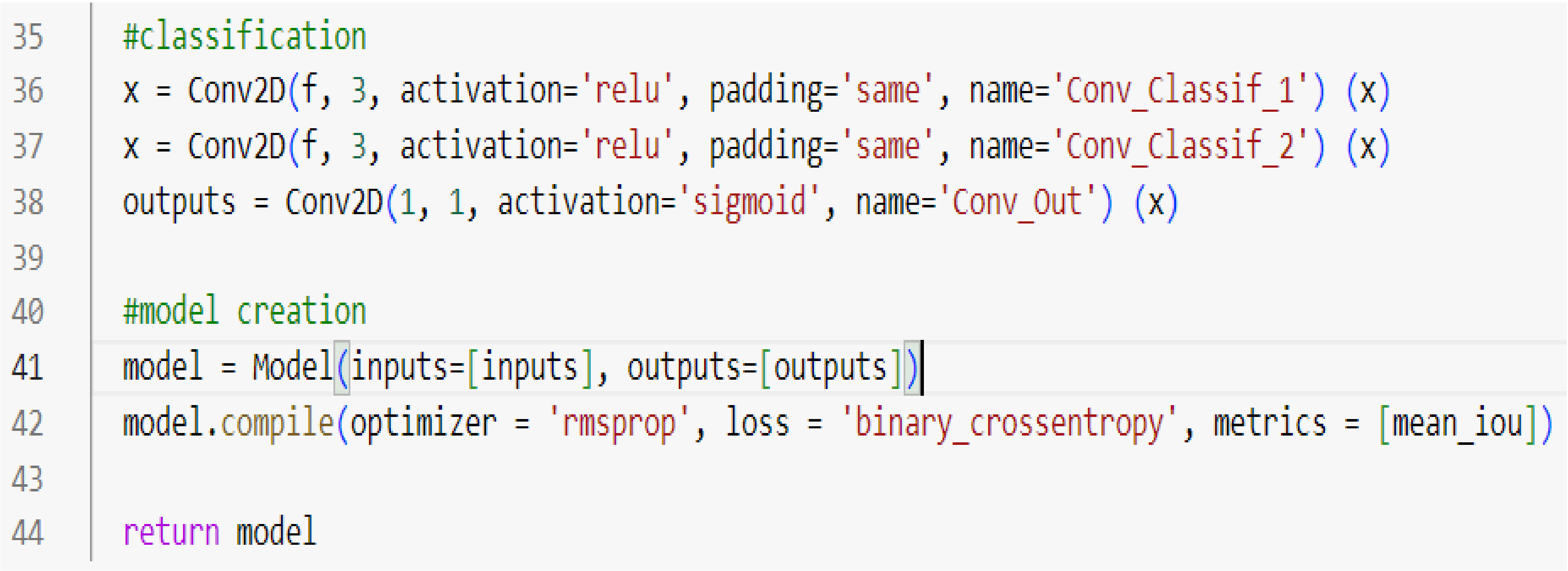




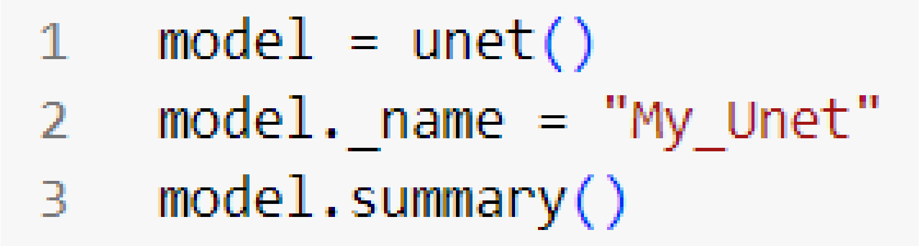
```python
def unet(sz = (256, 256, 3)):
    x = Input(sz, name="Input")
    inputs = x

    #down sampling
    f = 8
    layers = []

    for i in range(0, 6):
        x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Down_'+str(i+1)+'_1') (x)
        x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Down_'+str(i+1)+'_2') (x)
        layers.append(x)
        x = MaxPooling2D(name='MaxPool_Down_'+str(i+1)) (x)
        f = f*2
    ff2 = 64
```



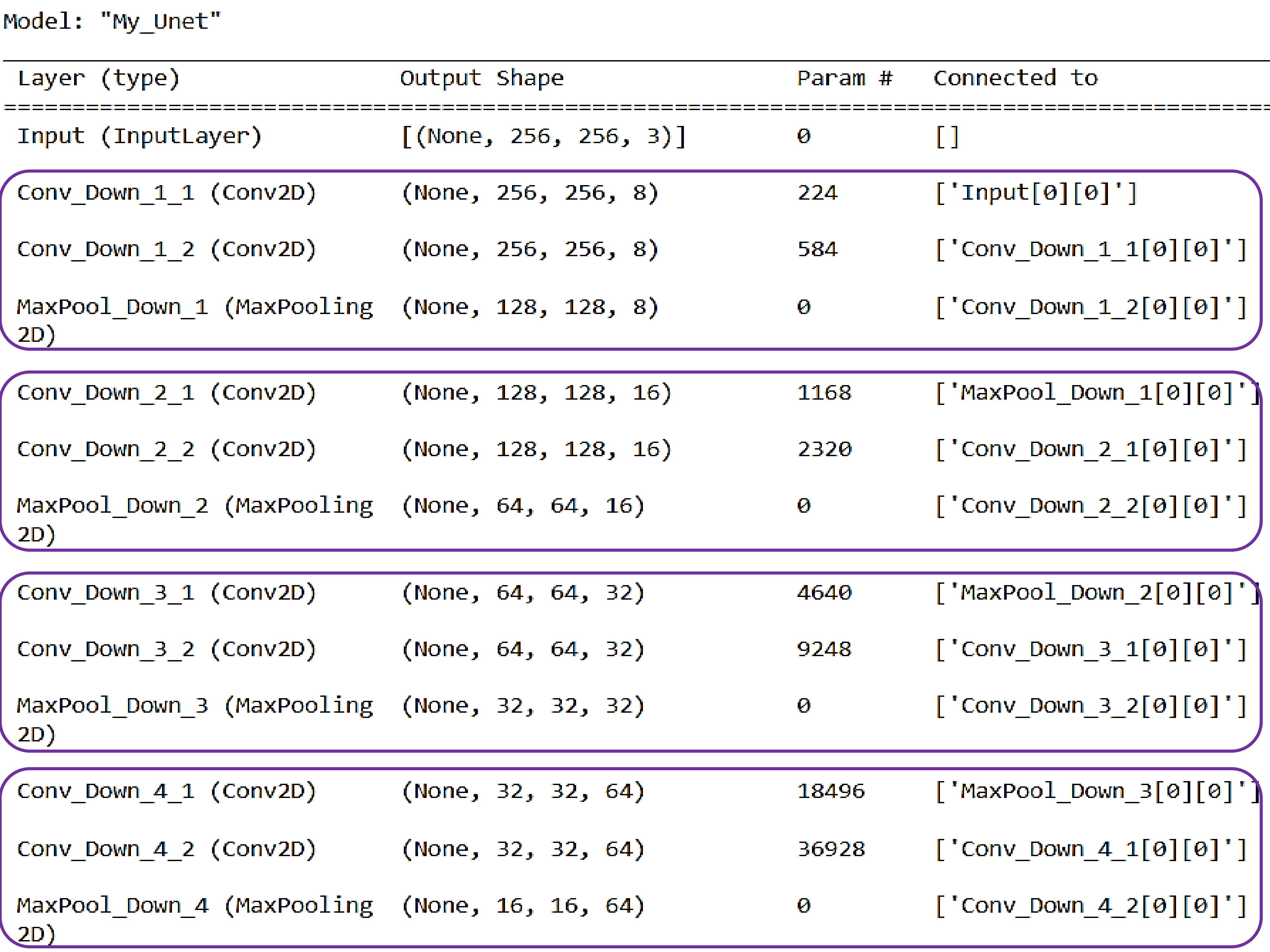
```
    #bottleneck
    j = len(layers) - 1
    x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Bottle_1') (x)
    x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Bottle_2') (x)
    x = Conv2DTranspose(ff2, 2, strides=(2, 2), padding='same', name='Conv_Transp_Bottle_') (x)
    x = Concatenate(axis=3, name='Concatenate_Bottle')([x, layers[j]])
    j = j -1

    #upsampling
    for i in range(0, 5):
        ff2 = ff2//2
        f = f // 2
        x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Up_'+str(i+1)+'_1') (x)
        x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Up_'+str(i+1)+'_2') (x)
        x = Conv2DTranspose(ff2, 2, strides=(2, 2), padding='same', name='Conv_Transp_Up_'+str(i+1)) (x)
        x = Concatenate(axis=3, name='Concatenate_Up_'+str(i+1))([x, layers[j]])
        j = j -1
```



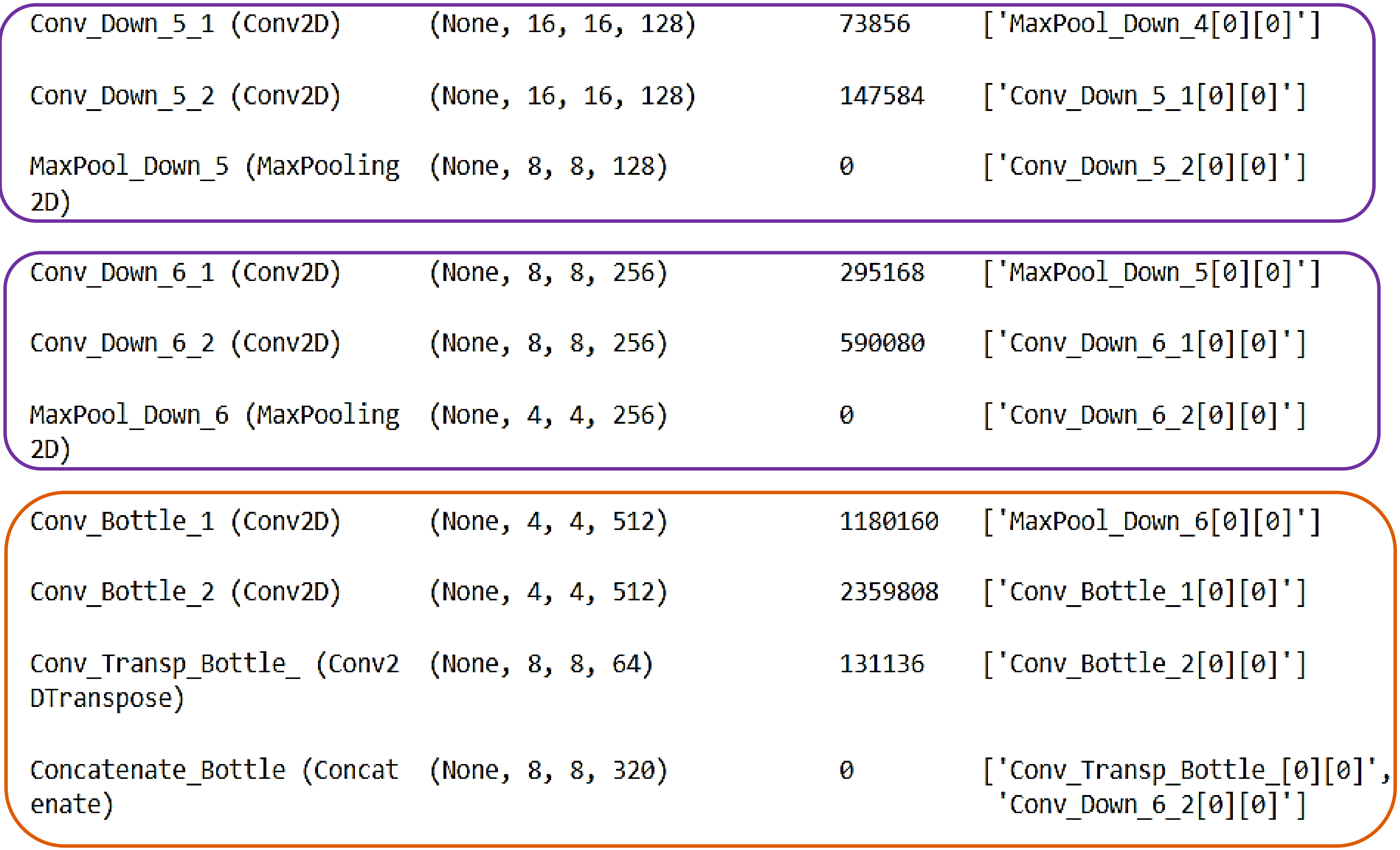
```
#classification
x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Classif_1') (x)
x = Conv2D(f, 3, activation='relu', padding='same', name='Conv_Classif_2') (x)
outputs = Conv2D(1, 1, activation='sigmoid', name='Conv_Out') (x)

#model creation
model = Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = [mean_iou])

return model
```



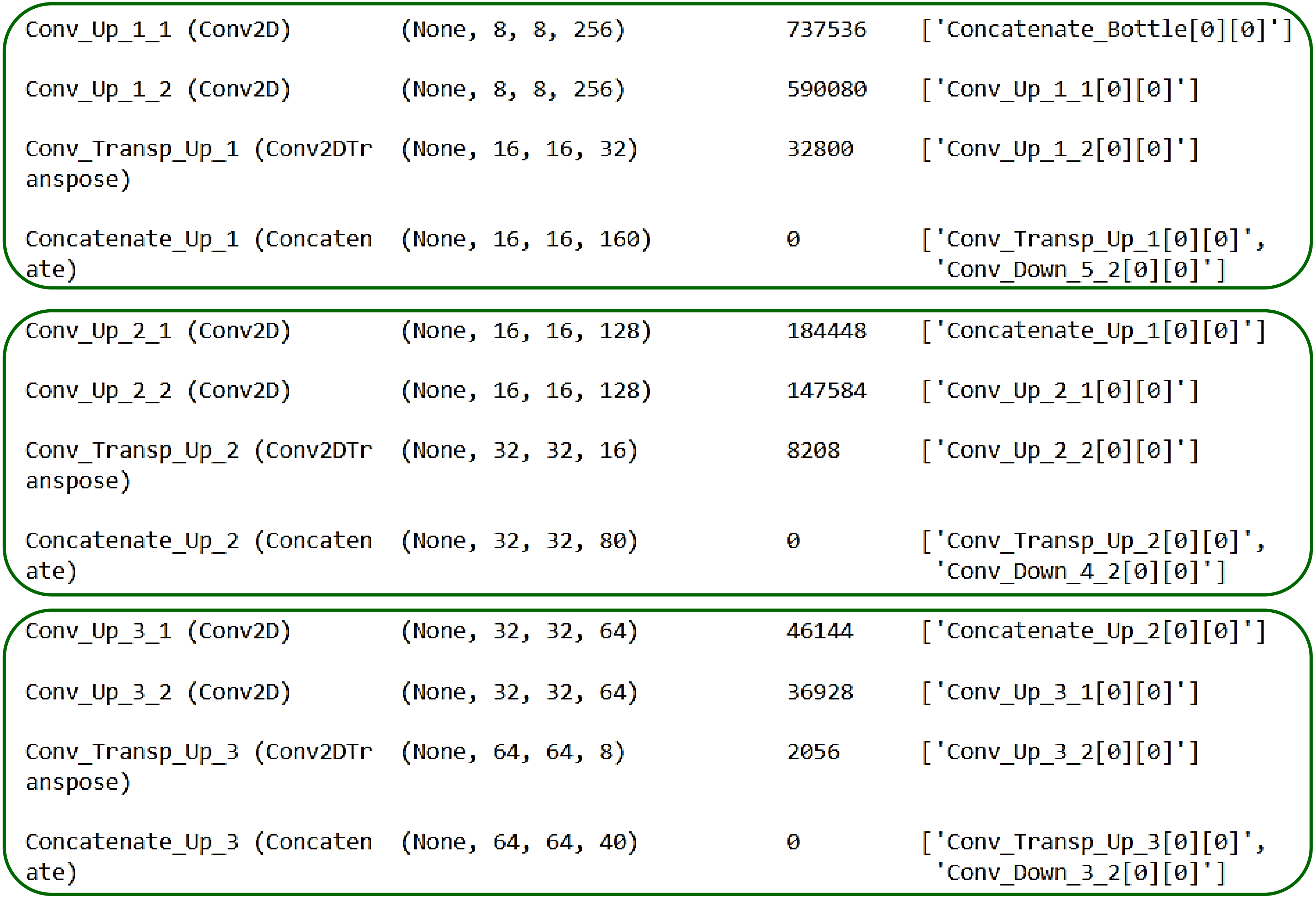
```
model = unet()
model._name = "My_Unet"
model.summary()
```



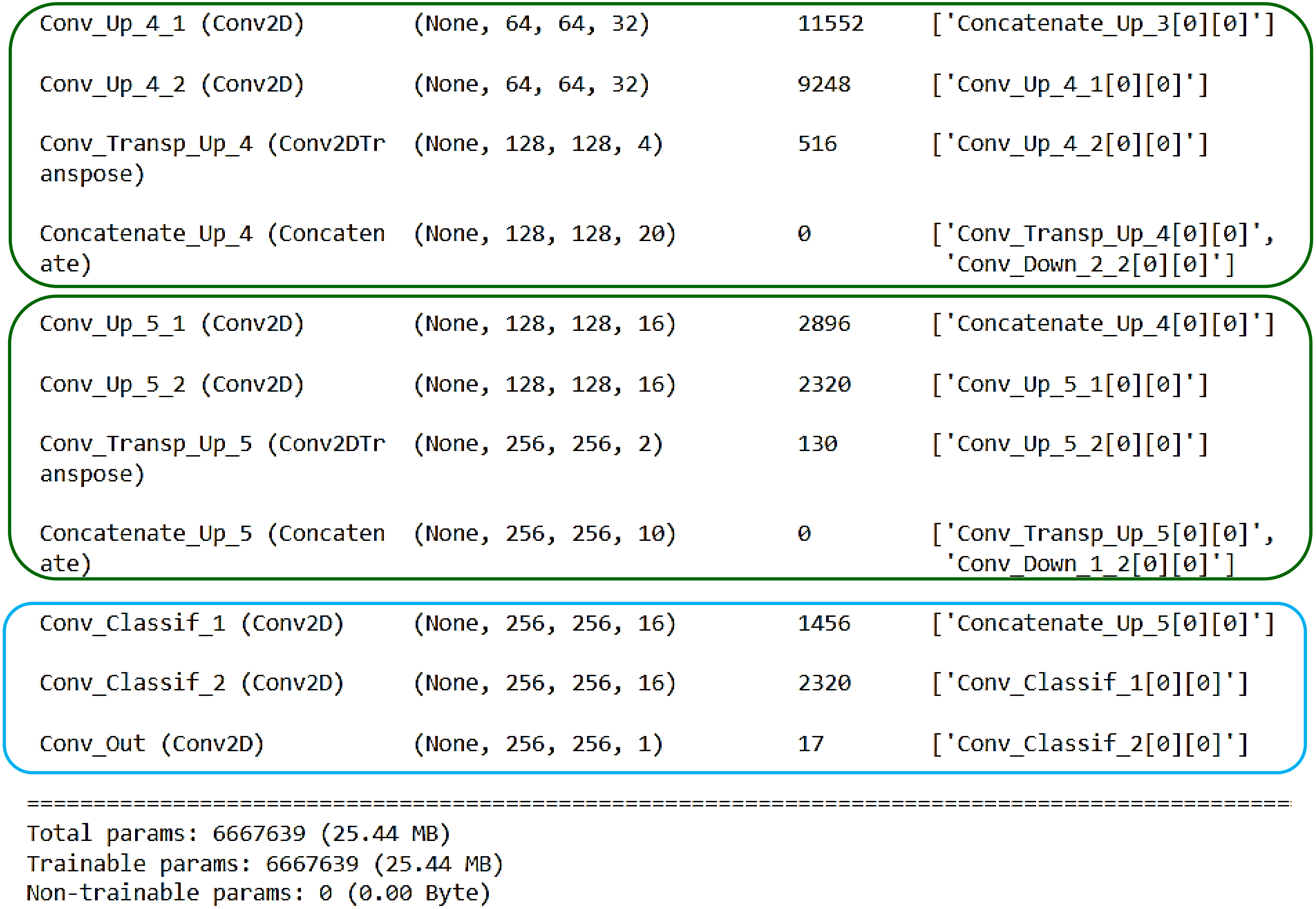
```
Model: "My_Unet"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 Input (InputLayer)             [(None, 256, 256, 3)]  0         []                               
                                                                                                  
 Conv_Down_1_1 (Conv2D)         (None, 256, 256, 8)  224         ['Input[0][0]']                  
                                                                                                  
 Conv_Down_1_2 (Conv2D)         (None, 256, 256, 8)  584         ['Conv_Down_1_1[0][0]']          
                                                                                                  
 MaxPool_Down_1 (MaxPooling     (None, 128, 128, 8)  0           ['Conv_Down_1_2[0][0]']          
 2D)                                                                                              
                                                                                                  
 Conv_Down_2_1 (Conv2D)         (None, 128, 128, 16  1168        ['MaxPool_Down_1[0][0]']         
                                )                                                                 
                                                                                                  
 Conv_Down_2_2 (Conv2D)         (None, 128, 128, 16  2320        ['Conv_Down_2_1[0][0]']          
                                )                                                                 
                                                                                                  
 MaxPool_Down_2 (MaxPooling     (None, 64, 64, 16)   0           ['Conv_Down_2_2[0][0]']          
 2D)                                                                                              
                                                                                                  
 Conv_Down_3_1 (Conv2D)         (None, 64, 64, 32)   4640        ['MaxPool_Down_2[0][0]']         
                                                                                                  
 Conv_Down_3_2 (Conv2D)         (None, 64, 64, 32)   9248        ['Conv_Down_3_1[0][0]']          
                                                                                                  
 MaxPool_Down_3 (MaxPooling     (None, 32, 32, 32)   0           ['Conv_Down_3_2[0][0]']          
 2D)                                                                                              
                                                                                                  
 Conv_Down_4_1 (Conv2D)         (None, 32, 32, 64)   18496       ['MaxPool_Down_3[0][0]']         
                                                                                                  
 Conv_Down_4_2 (Conv2D)         (None, 32, 32, 64)   36928       ['Conv_Down_4_1[0][0]']          
                                                                                                  
 MaxPool_Down_4 (MaxPooling     (None, 16, 16, 64)   0           ['Conv_Down_4_2[0][0]']          
 2D)                                                                                              
```



```
Conv_Down_5_1 (Conv2D)      (None, 16, 16, 128)        73856     ['MaxPool_Down_4[0][0]']

Conv_Down_5_2 (Conv2D)      (None, 16, 16, 128)        147584    ['Conv_Down_5_1[0][0]']

MaxPool_Down_5 (MaxPooling  (None, 8, 8, 128)          0         ['Conv_Down_5_2[0][0]']
2D)

Conv_Down_6_1 (Conv2D)      (None, 8, 8, 256)          295168    ['MaxPool_Down_5[0][0]']

Conv_Down_6_2 (Conv2D)      (None, 8, 8, 256)          590080    ['Conv_Down_6_1[0][0]']

MaxPool_Down_6 (MaxPooling  (None, 4, 4, 256)          0         ['Conv_Down_6_2[0][0]']
2D)

Conv_Bottle_1 (Conv2D)      (None, 4, 4, 512)          1180160   ['MaxPool_Down_6[0][0]']

Conv_Bottle_2 (Conv2D)      (None, 4, 4, 512)          2359808   ['Conv_Bottle_1[0][0]']

Conv_Transp_Bottle_ (Conv2  (None, 8, 8, 64)           131136    ['Conv_Bottle_2[0][0]']
DTranspose)

Concatenate_Bottle (Concat  (None, 8, 8, 320)          0         ['Conv_Transp_Bottle_[0][0]',
enate)                                                            'Conv_Down_6_2[0][0]']
```



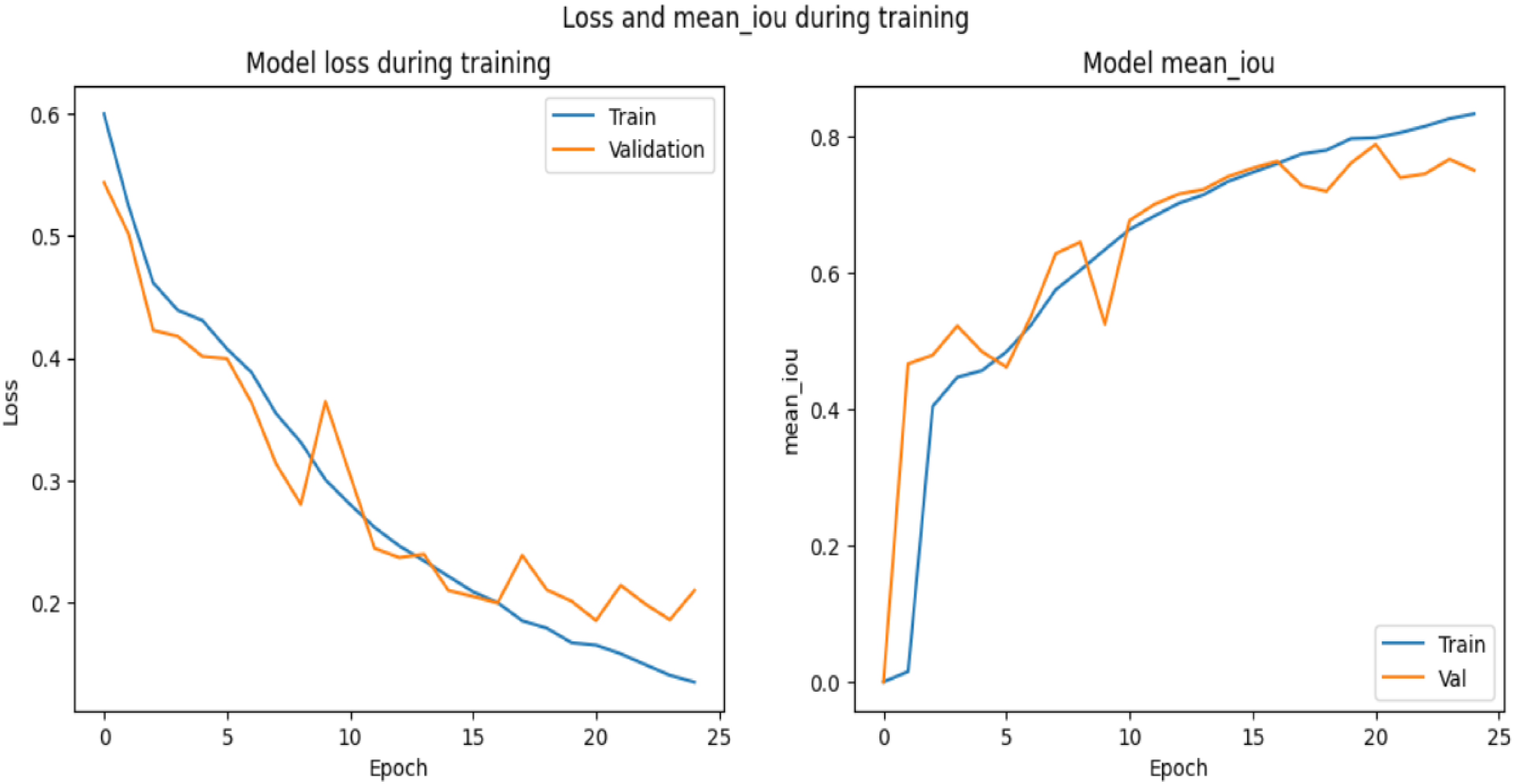
```
Conv_Up_1_1 (Conv2D)         (None, 8, 8, 256)           737536     ['Concatenate_Bottle[0][0]']

Conv_Up_1_2 (Conv2D)         (None, 8, 8, 256)           590080     ['Conv_Up_1_1[0][0]']

Conv_Transp_Up_1 (Conv2DTr  (None, 16, 16, 32)           32800      ['Conv_Up_1_2[0][0]']
anspose)

Concatenate_Up_1 (Concaten  (None, 16, 16, 160)          0          ['Conv_Transp_Up_1[0][0]',
ate)                                                                 'Conv_Down_5_2[0][0]']

Conv_Up_2_1 (Conv2D)         (None, 16, 16, 128)         184448     ['Concatenate_Up_1[0][0]']

Conv_Up_2_2 (Conv2D)         (None, 16, 16, 128)         147584     ['Conv_Up_2_1[0][0]']

Conv_Transp_Up_2 (Conv2DTr  (None, 32, 32, 16)           8208       ['Conv_Up_2_2[0][0]']
anspose)

Concatenate_Up_2 (Concaten  (None, 32, 32, 80)           0          ['Conv_Transp_Up_2[0][0]',
ate)                                                                 'Conv_Down_4_2[0][0]']

Conv_Up_3_1 (Conv2D)         (None, 32, 32, 64)          46144      ['Concatenate_Up_2[0][0]']

Conv_Up_3_2 (Conv2D)         (None, 32, 32, 64)          36928      ['Conv_Up_3_1[0][0]']

Conv_Transp_Up_3 (Conv2DTr  (None, 64, 64, 8)            2056       ['Conv_Up_3_2[0][0]']
anspose)

Concatenate_Up_3 (Concaten  (None, 64, 64, 40)           0          ['Conv_Transp_Up_3[0][0]',
ate)                                                                 'Conv_Down_3_2[0][0]']
```



```
Conv_Up_4_1 (Conv2D)        (None, 64, 64, 32)          11552     ['Concatenate_Up_3[0][0]']

Conv_Up_4_2 (Conv2D)        (None, 64, 64, 32)          9248      ['Conv_Up_4_1[0][0]']

Conv_Transp_Up_4 (Conv2DTr  (None, 128, 128, 4)         516       ['Conv_Up_4_2[0][0]']
anspose)

Concatenate_Up_4 (Concaten  (None, 128, 128, 20)        0         ['Conv_Transp_Up_4[0][0]',
ate)                                                                 'Conv_Down_2_2[0][0]']

Conv_Up_5_1 (Conv2D)        (None, 128, 128, 16)        2896      ['Concatenate_Up_4[0][0]']

Conv_Up_5_2 (Conv2D)        (None, 128, 128, 16)        2320      ['Conv_Up_5_1[0][0]']

Conv_Transp_Up_5 (Conv2DTr  (None, 256, 256, 2)         130       ['Conv_Up_5_2[0][0]']
anspose)

Concatenate_Up_5 (Concaten  (None, 256, 256, 10)        0         ['Conv_Transp_Up_5[0][0]',
ate)                                                                 'Conv_Down_1_2[0][0]']

Conv_Classif_1 (Conv2D)     (None, 256, 256, 16)        1456      ['Concatenate_Up_5[0][0]']

Conv_Classif_2 (Conv2D)     (None, 256, 256, 16)        2320      ['Conv_Classif_1[0][0]']

Conv_Out (Conv2D)           (None, 256, 256, 1)         17        ['Conv_Classif_2[0][0]']

==================================================================================================
Total params: 6667639 (25.44 MB)
Trainable params: 6667639 (25.44 MB)
Non-trainable params: 0 (0.00 Byte)
```

Batch size =32, Epochs = 25

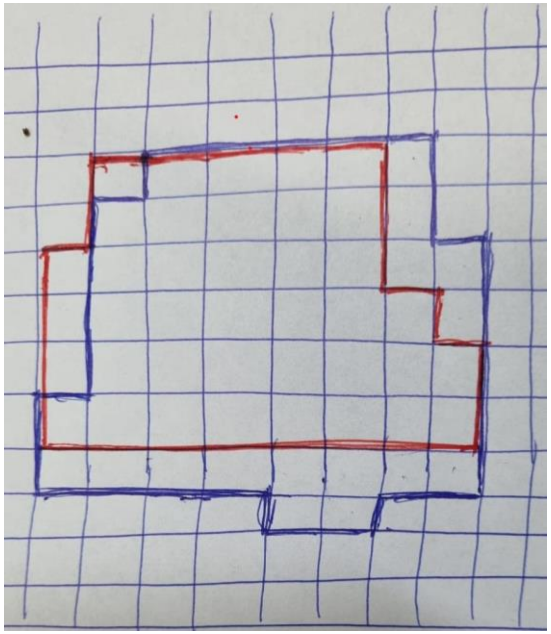
IOU at pixel level



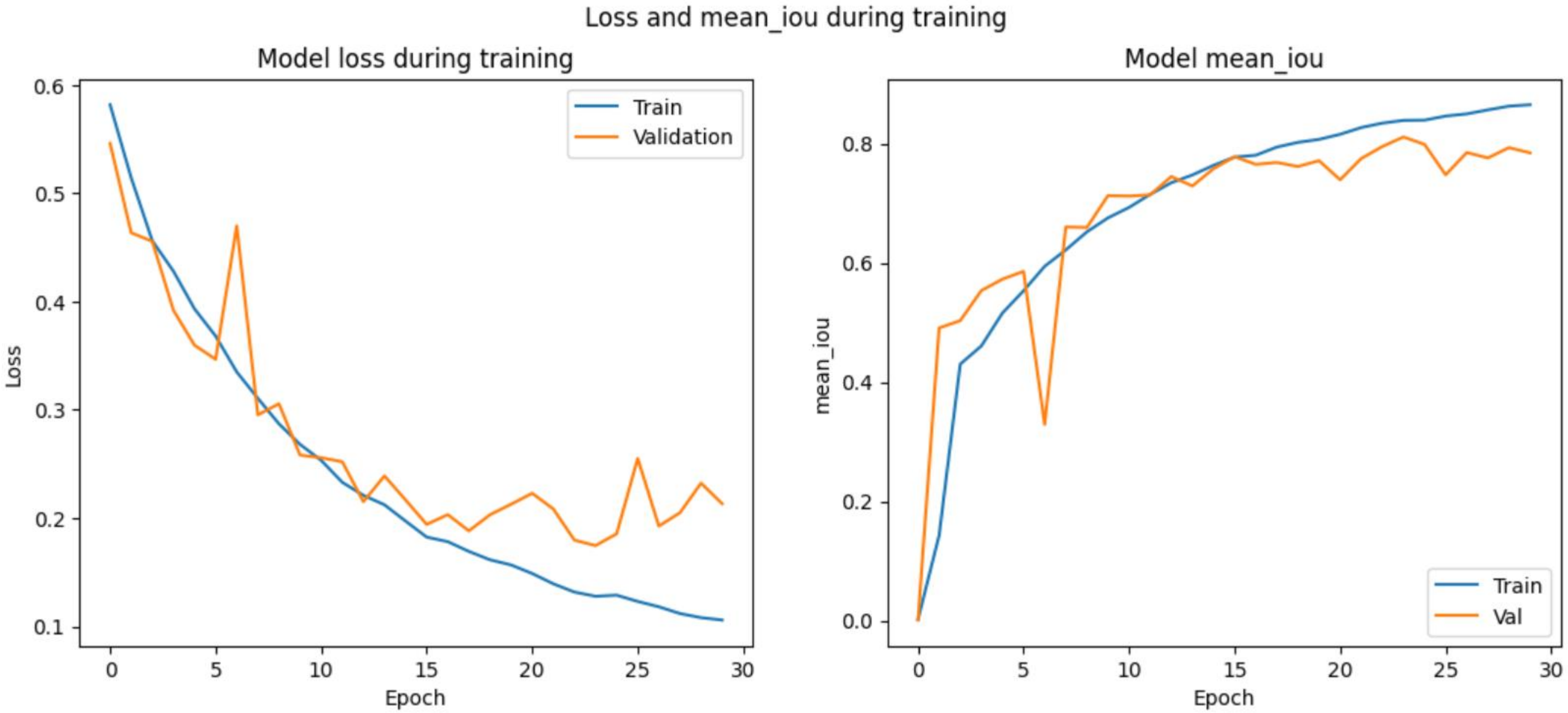
Exercise

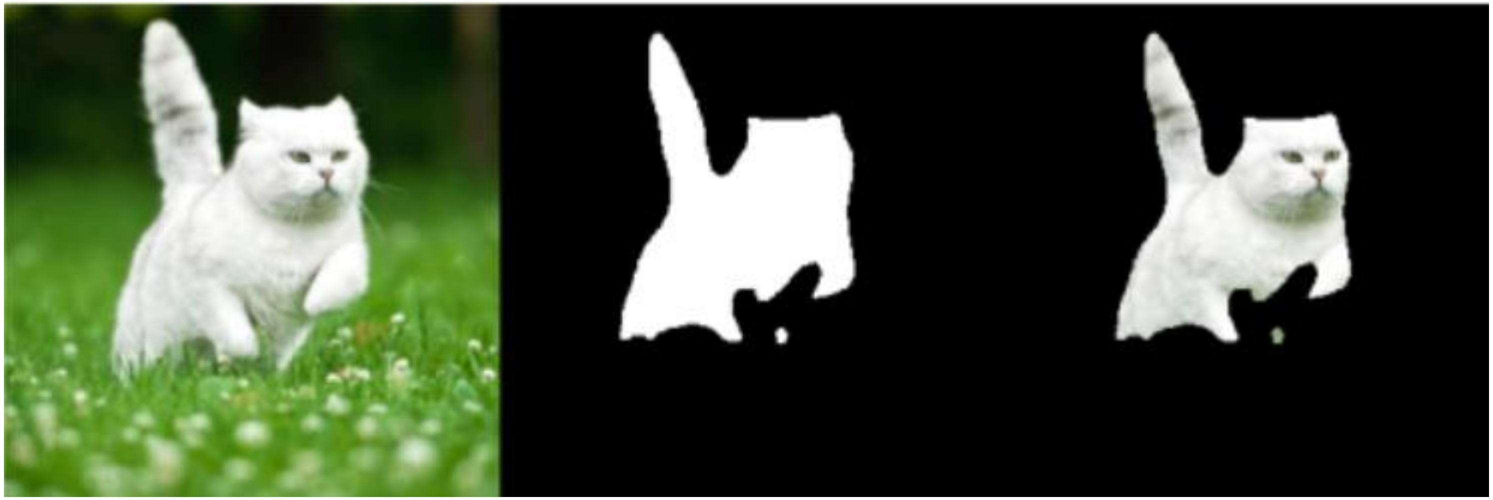
The target mask for an image segmentation is represented in blue
The predicted mask is represented in red.

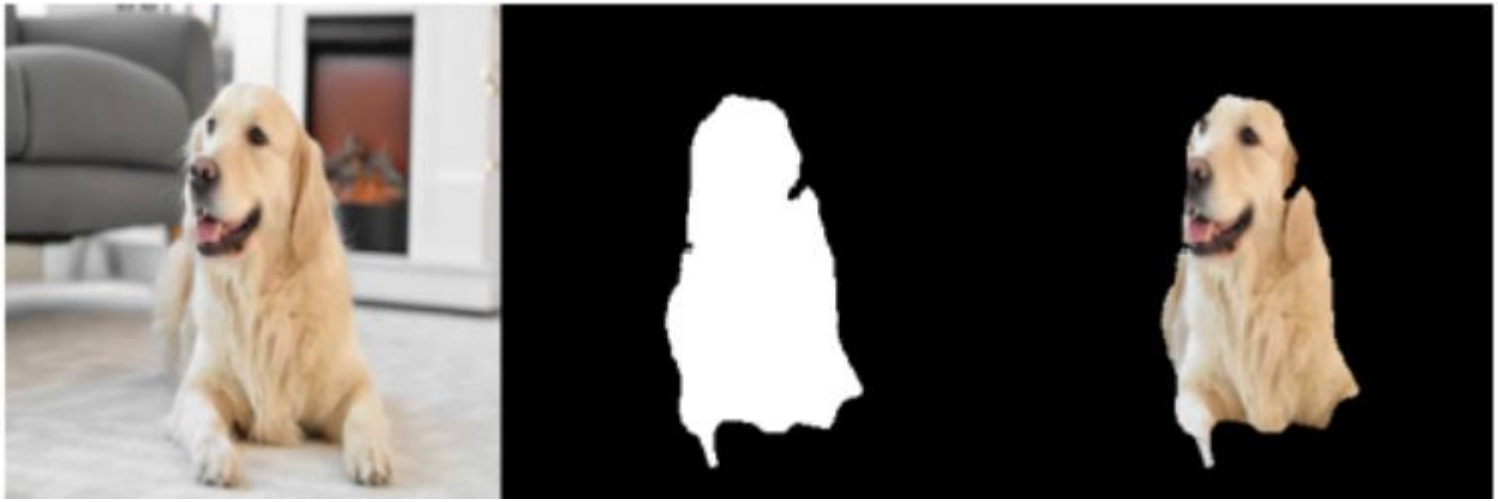
What is the IoU value?

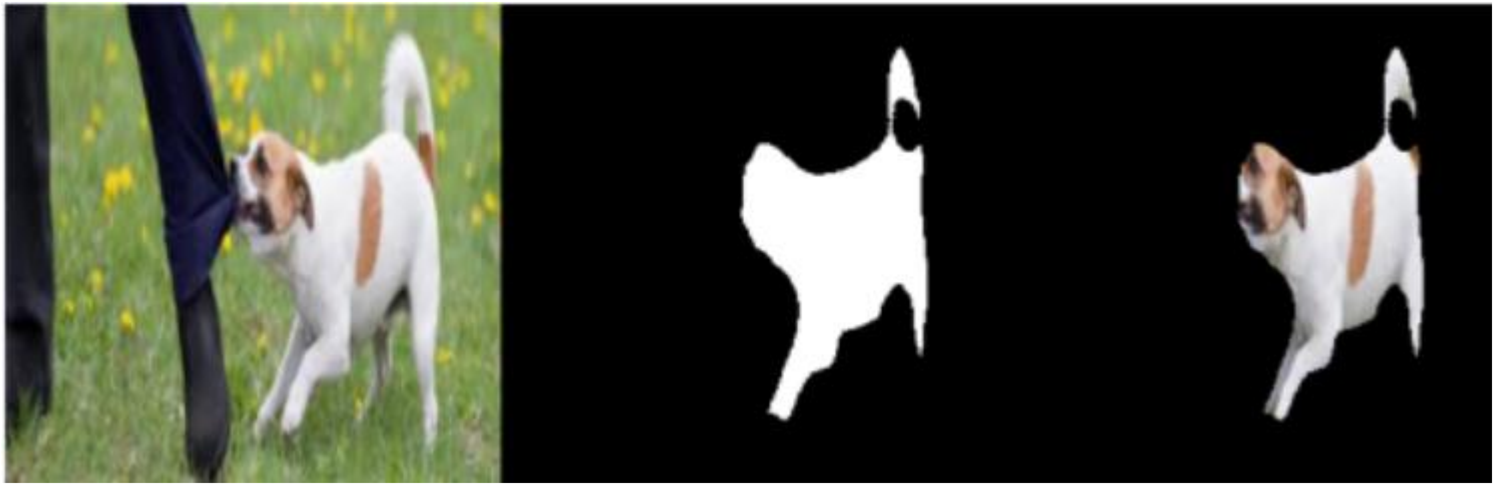


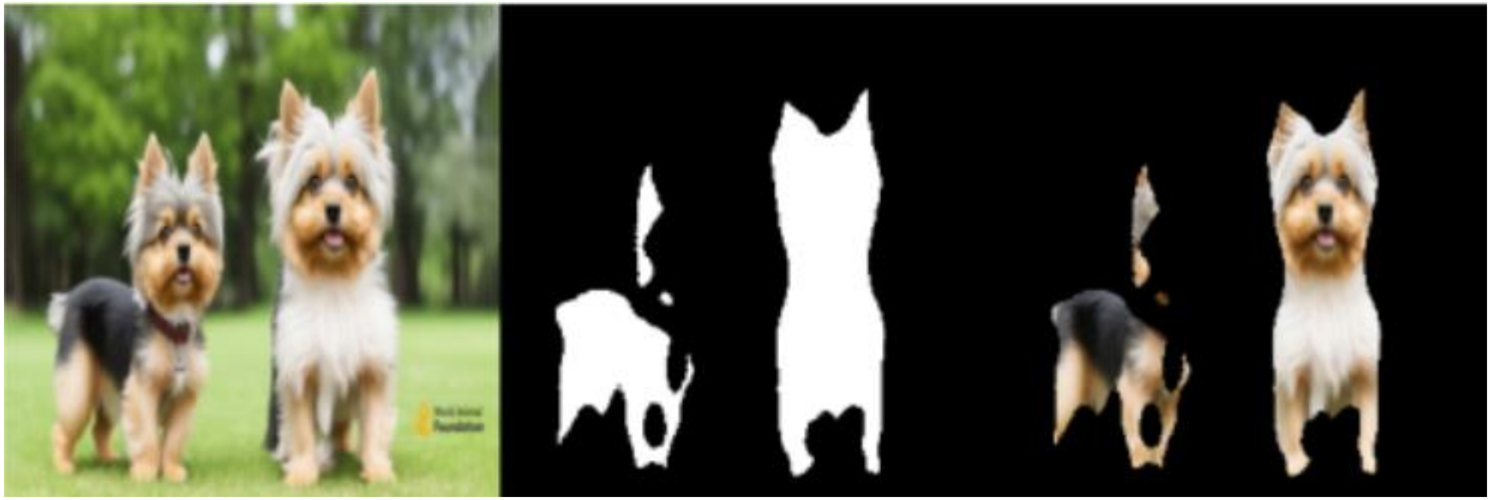
Batch size =32, Epochs = 30





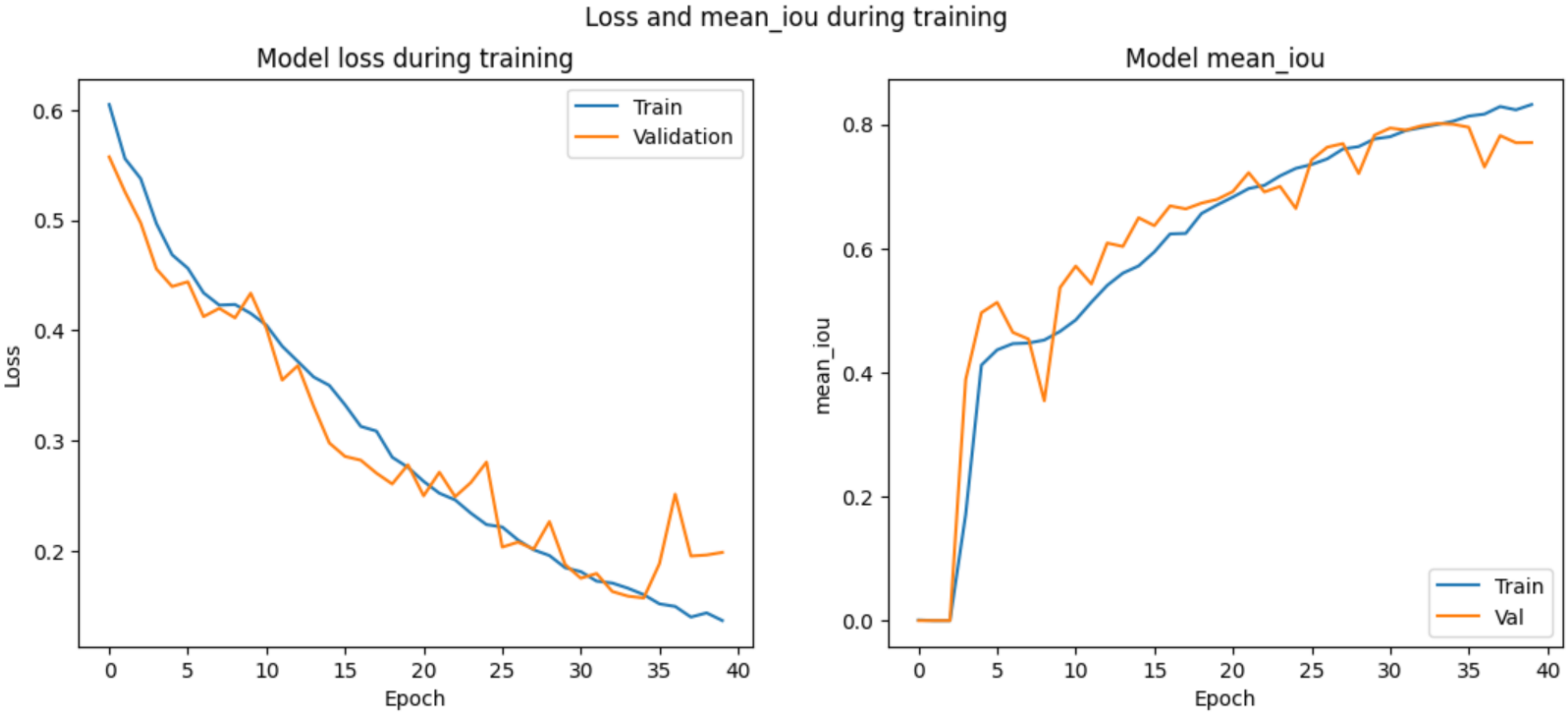


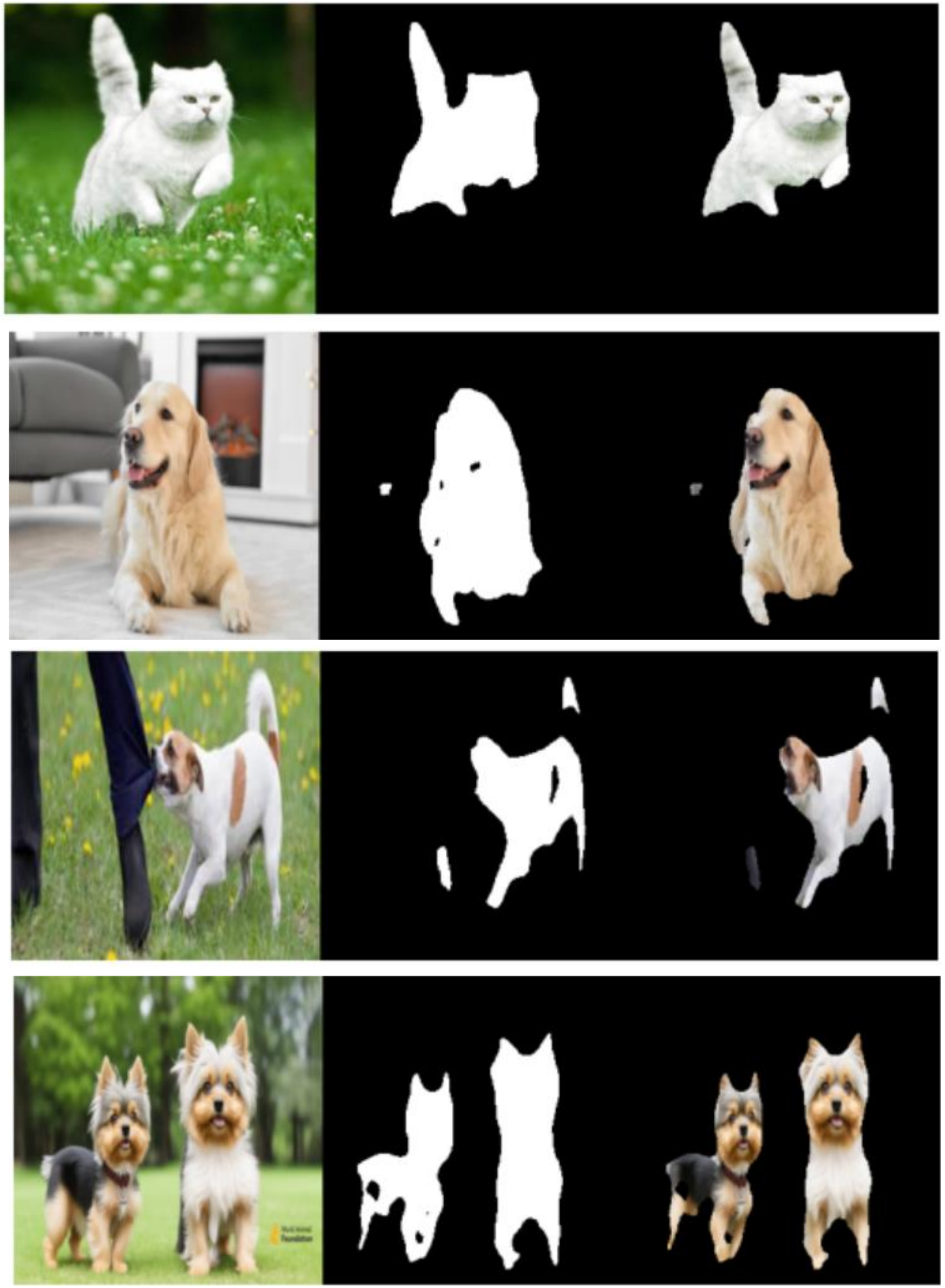




Batch size =64, Epochs = 40,

approx.. 54 minutes, GPU T4, Google Colab





Data Augmentation and Regularization:

UNET should employ data augmentation and regularisation techniques to improve its resilience and generalization ability during training.

To increase the diversity of the training data, data augmentation entails adding numerous transformations to the training images, such as rotations, flips, scaling, and deformations.

Regularisation techniques such as dropout and batch normalization prevent overfitting and improve model performance on unknown data.

Applications and use cases of Semantic Segmentation

Semantic segmentation is used for a wide variety of applications.

Autonomous vehicles, for example, require perception, planning and execution in constantly changing environments.
They also require high accuracy, as **safety must be foolproof on the road**. Thanks to semantic segmentation, it is possible for unmanned cars to detect open spaces on lanes, road markings and traffic signs.

This AI technique is also used for **medical diagnostics.** The machines can support the analyses carried out by radiologists in order to reduce the time needed to make diagnoses.

Another use case is **satellite mapping**, which is very important for monitoring deforestation areas or for urbanisation. Semantic segmentation allows distinguishing different types of land in an automated way. The detection of buildings and roads is also very useful for traffic management or urban planning.

Finally, **precision agriculture robots** can use semantic segmentation to distinguish plantations from weeds. This allows them to automate weed control using less herbicide.

[https://datascientest.com/en/u-net-computer-visions-neural-network]