upGrad

Session summary

Introduction to semantic segmentation

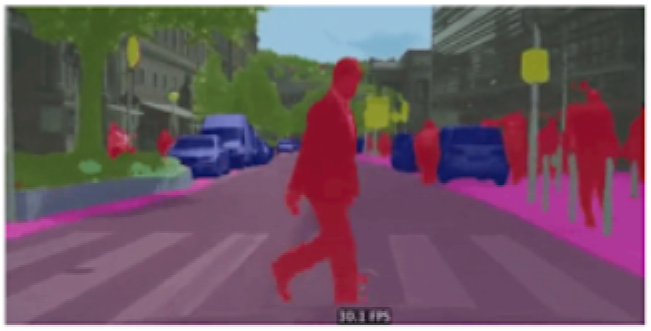
In this session we will learn about the second application of deep learning in computer vision, that is image Segmentation.

Image segmentation: is the process of partitioning a digital image into multiple segments, this Is an extension of object detection as instead of bounding boxes, we will try to mark the pixels where the object is present on the screen.

The following topics are covered in this session:

- Image segmentation
- Types of image segmentation
    - Semantic Segmentation
    - Instance segmentation
- Algorithms
    - Semantic segmentation: U-Net
        - Theory: Encoder-Decoder
        - Theory:Up-sampling, loss function
        - Implementation: Lung Segmentation
    - Instance Segmentation: Mask R-CNN
        - Architecture
        - Implementation: Oil Tank Segmentation

Example of Image segmentation:



🟥	Person
🟦	Cars
⬛	Buildings
🟩	Trees
🟪	Sidewalk

Source: Oles Andrienko (https://github.com/oandrienko) upGrad

Image Segmentation Vs Object Segmentation

Image segmentation is an extension of the concept of object detection in the field of computer vision.

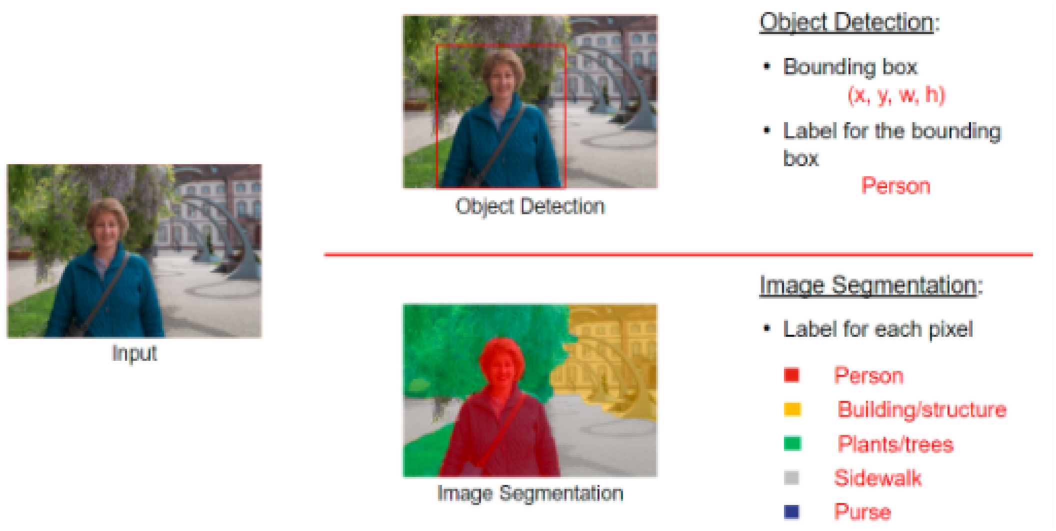
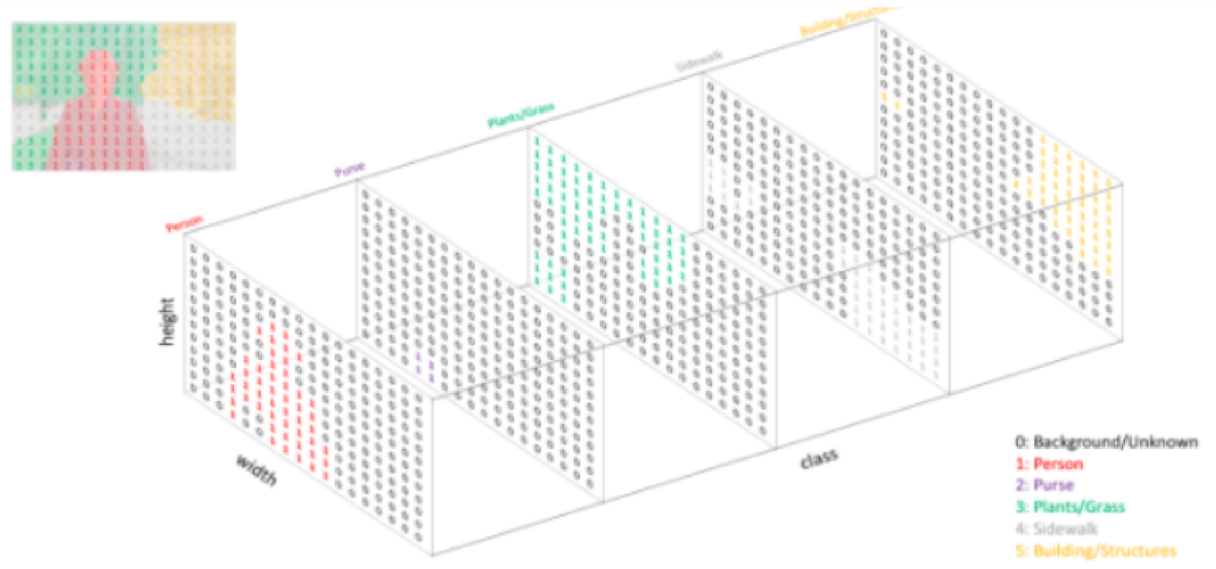


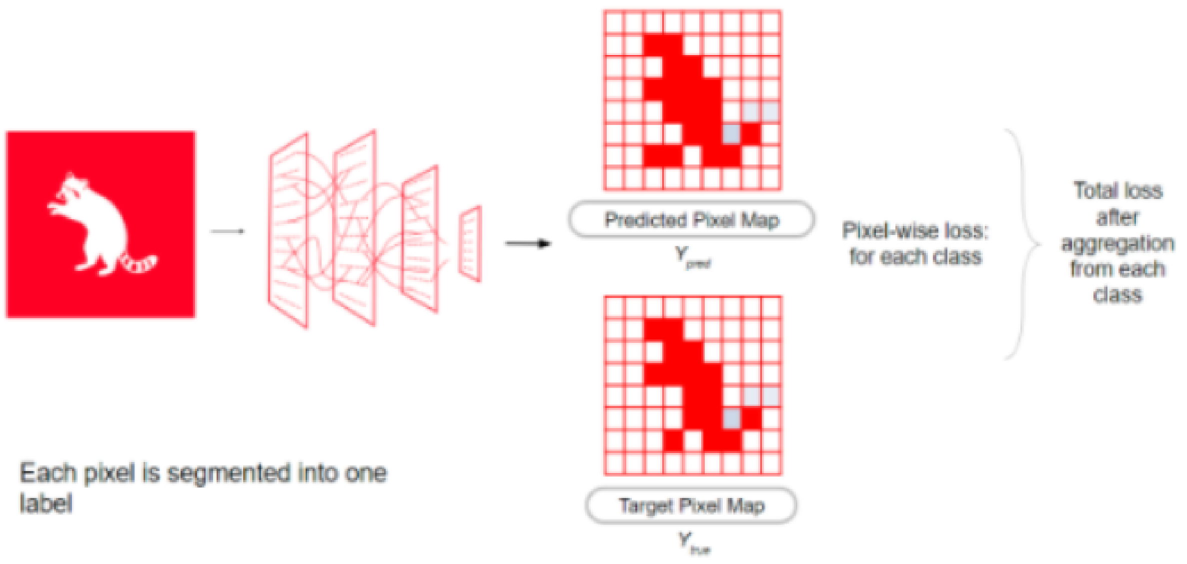
Image segmentation:
The image segmentation segregates the input image into different segments of particular quality. It works by labelling each pixel as a particular class of interest. As shown in the above image, different segments are person, Building, Plants, sidewalk and Purse.

Object detection:
In Object detection, the algorithm is used to detect and identify the position of the object in the image. For example, You will be able to draw and label a bounding box around the person in the image above.

Image segmentation is helpful in cases where you want to understand the shape or the structure of the desired object in the image. This remains absent in the case of object detection as the bounding boxes are always rectangular in nature. The process of image segmentation results in a segmentation map. Where the height and width of the output and input is similar.



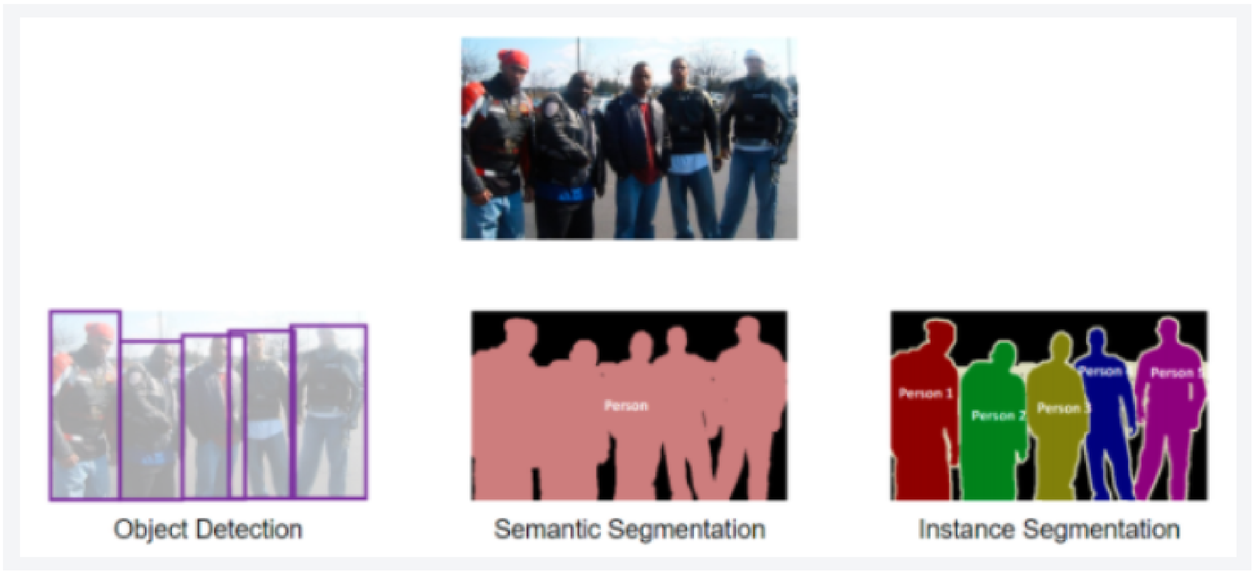
Loss function in image segmentation:
The most commonly used loss function here is a pixel-wise cross-entropy loss. This loss examines each pixel individually and compares the results with the class labels provided by the user as input. The result for each pixel is then aggregated in order to check the overall efficiency of the modal. We will learn more about loss function in future segments.


Predicted Pixel Map
Y_{pred}
Target Pixel Map
Y_{true}
Pixel-wise loss: for each class
Total loss after aggregation from each class
Each pixel is segmented into one label

Types Of Image Segmentation

Image segmentation helps to segment the image into different sections based on the class to which it belongs. There are two types of image segmentation.

1. Semantic segmentation
2. Instance segmentation



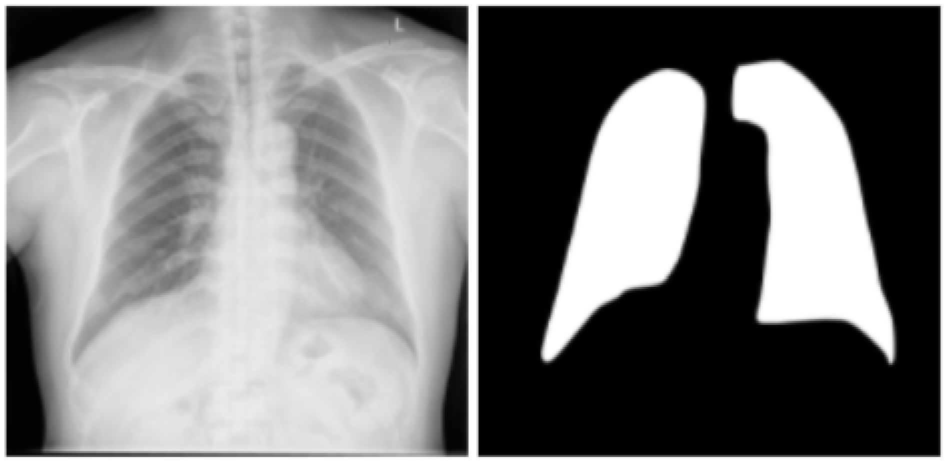
Semantic segmentation:
In this process each pixel of an image is labeled with a corresponding class of what is being represented. This is best used when different classes are present in an image. As seen in the above image, all the individuals are segmented as “person”.

Instance segmentation:
Instance segmentation is a conceptual extension of semantic segmentation where in along with pixel level classification, the model classifies each instance of a class separately. As you can see in the above image, each person is highlighted using different colors.

An example using sports analytics:
Consider that you want to analyse the movement of players in a football match. If you are only interested in the formations of the movement of the teams in the match, you can segment all the players under one category. However, if you want to dig deeper into the performance of each player, you will have to segment them individually to separate them from one another. Therefore, you can choose either of these applications based on the use case.

Similarly the task of image segmentation requires the user to provide additional details along with each image, one of the inputs are Masks.

Image segmentation: Masks
Masks are essentially the annotated examples of what we are trying to teach the algorithm to detect and segment. Masks can be binary or multi-label Masks depending on the problem at hand. In a mask black corresponds to the background class, that is irrelevant to the problem solving. And white is the area that needs to be problem solving.



Example of a binary mask

Industry Applications

Implementation of Image segmentation can be seen in multiple industries, these are examples of Semantic and instance image segmentation.

1. Autonomous driving:
Mostly used in self-driving cars for environment analysis. Semantic segmentation is used to label cars, pedestrians and sidewalks. Semantic segmentation is used as we are not really interested in labeling individual cars and pedestrians. If architecture detects the path, it will move ahead; else, the car would stop. In complex versions, you can identify different classes in the image and train the model to act accordingly for different cases.

2.Medical imaging:
Medical image segmentation is the task of segmenting objects of interest in a medical image. Image segmentation helps doctors organ mapping, detecting diseases from modalities like CT, X-ray, ultrasound, microscopy, MRI, ets. As we don't have multiple instances and as such, semantic segmentation would be ideal.

3.Surveillance:
Image segmentation is also used in tracking the movement of elements like cars, humans, etc. Instance segmentation finds a crucial role here as you can monitor a specific instance through the algorithms.

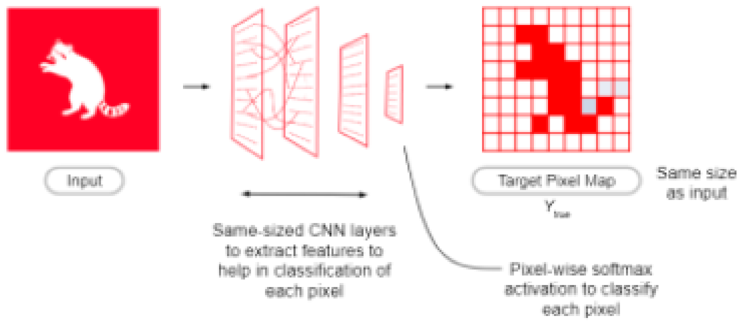
4.Satellite imagery:
As satellite imagery contains large scale landscape factors like deforestation, resource distribution, etc. Models can be trained to detect minerals and track shipments and trains, etc.

5.Portrait mode and other image effects:
Image segmentation can also be used in fields of photography and videography. To provide various features like background blur in phone cameras. Semantic segmentation is used to identify the regions of interest such as face, object, etc.

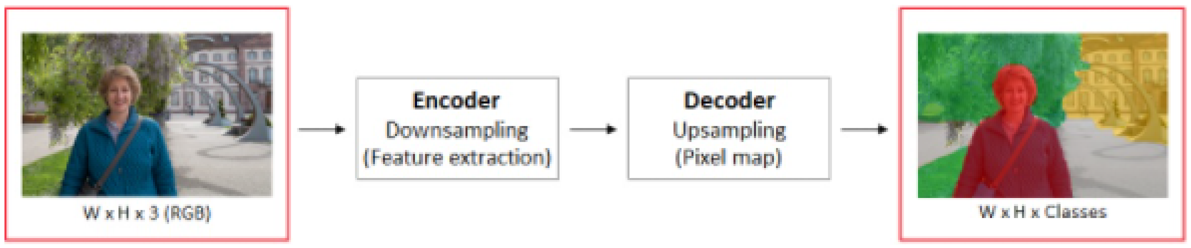
Semantic Segmentation

In this session, we will learn to solve problems faced in neural networks. In semantic segmentation, the prominent two different approaches are:

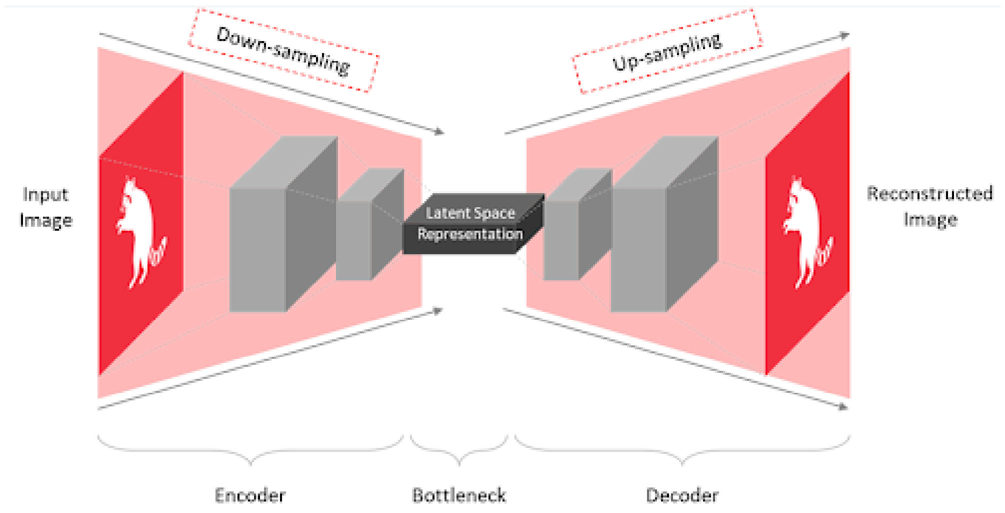
1.Simple CNN framework:
Semantic segmentation can be considered as pixel-level classification where each pixel can be segmented into a specific class.As the size of the input and the output is the same, you can either use kernels of size 1 x 1 or use padding to maintain the size across layers. With the increase in the number of layers, it takes more operations to obtain the final result. This makes the process computationally expensive as it involves the same sized layers at each level. Hence, this approach is not preferred.


Input
Target Pixel Map
Y_{true}
Same size as input
Same-sized CNN layers to extract features to help in classification of each pixel
Pixel-wise softmax activation to classify each pixel

2.Encoder-decoder framework:
Encoder-decoder framework is used to reduce the computational burden of processing. Encoder- decoder framework works by reducing the size of the input while learning the features to identify each class(encoder phase), and then reverse the process by upsampling the feature representation into a full resolution segmentation map (decoder phase) to classify each pixel of the input image.

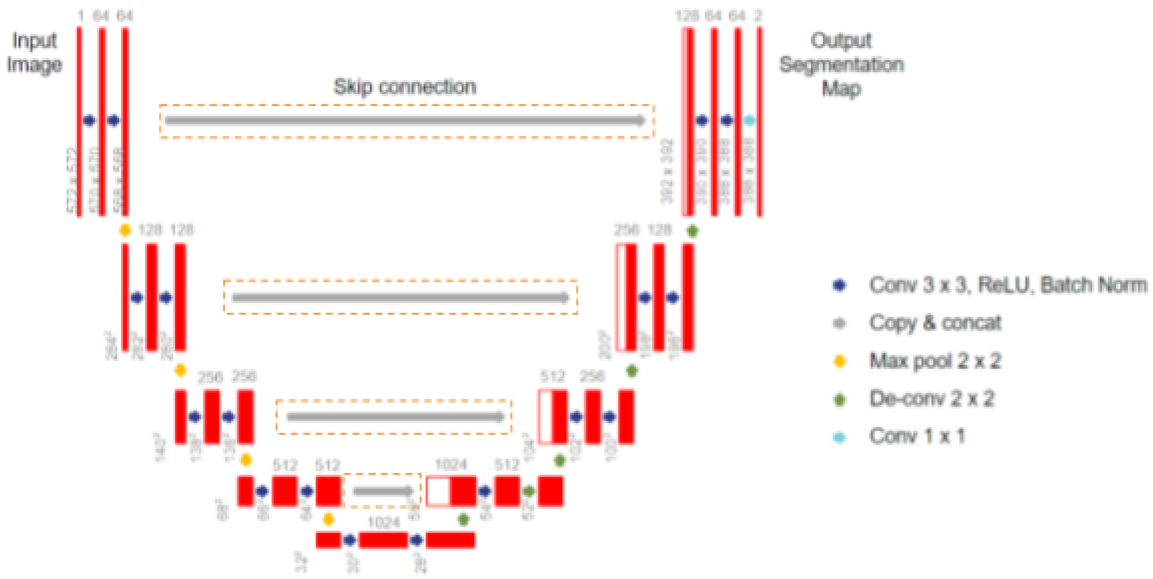


Encoder-decoder framework in more detail:
This architecture uses the convolutional neural network (CNN) to perform the required functions. This process starts by processing the input image using an encoder. This part acts as the feature extractor where all the features are learnt in a hierarchical order using CNN layers. In the process, the image undergoes different operations like convolution, pooling, etc. resulting in spatial reduction which is known as downsampling.



In a simple encoder-decoder framework, the encoder and the decoder are independent; different image segmentation architectures use different ways of combining and connecting these layers. One example of a simple encoder-decoder framework is SegNet. However, in the more recent architectures, the encoder and decoder sections are interconnected to provide better performance. One such example is U-Net. It was proposed in 2015 by Olaf Ronneberger, Philip Fischer and Thomas Brox in the paper - *U-Net convolutional networks for biomedical image segmentation*. The architecture derives its name from the shape as it resembles the alphabet 'U'.

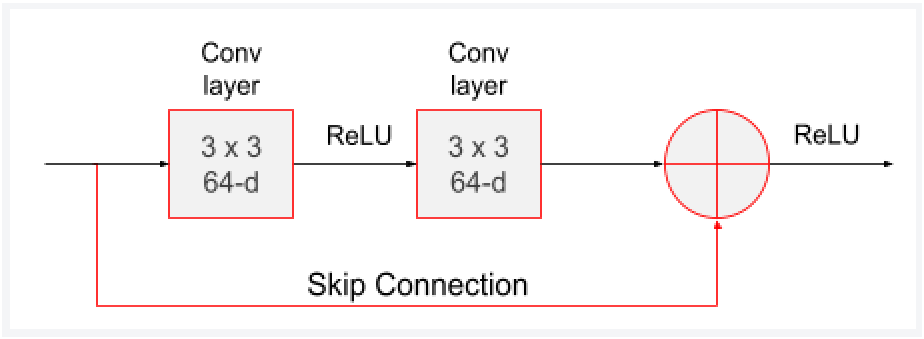
Encoder-Decoder Networks: UNET
In Encoder - Decoder Networks: UNET, the output mapping in units depends directly on the input space. The relationship of the input to the output is what we refer to as skip connections. The image below highlights these skip connections.



Skip Connections

From the previous session we learned that the U-Net framework differs from the basic structure as it uses skip connections.

More about Skip connections:
Skip connections were first introduced in the ResNet framework to overcome the problem of vanishing gradient. Vanishing gradient problem is prone to errors due to suboptimal error optimization. As the network becomes deeper, the initial layers tend to lose their importance in the process of backpropagation. Hence skip connections helped in overcoming this problem by offering an alternate path for the architecture to optimise the loss function.



U-Net: Architecture - 1

Encoder - decoder architecture in more detail:

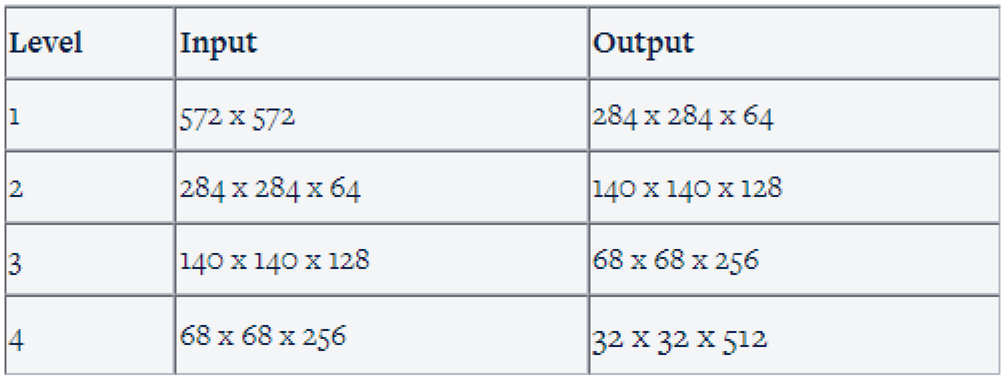
The encoder - decoder architecture consists of three sections:

- Encoder
- Bottleneck
- Decoder

1.Encoder/Contraction:

The first phase of the architecture is responsible for the downsampling process. As an example, we have taken a 572x572 image as the input.

- In the first level, the input image passes through two convolutional layers that have 64 filters each. These layers are responsible for extracting the lower level feature which will be combined along the encoder phase to obtain higher-level features present in the image.
- At the end of the convolution layers you have an output with 64 channels as depth which is passed through a max pooling layer with a 2x2 window and stride of 2. This is the key factor that results in downsampling as each pooling layer reduces the dimensions to half. so 568x568 is reduced to 284x284.
- The process is repeated with the increase in the number of feature maps and reduction of width and height dimensions at each level.



Level	Input	Output
1	572 x 572	284 x 284 x 64
2	284 x 284 x 64	140 x 140 x 128
3	140 x 140 x 128	68 x 68 x 256
4	68 x 68 x 256	32 x 32 x 512

The input image of size 572x572 has been reduced to 32x32, and there are 512 different features extracted from the image that will help in categorising pixels into a specific class.

U-Net: Architecture - 2

The output from the encoder is fed into the decoder via the bottleneck.

2.Bottleneck:

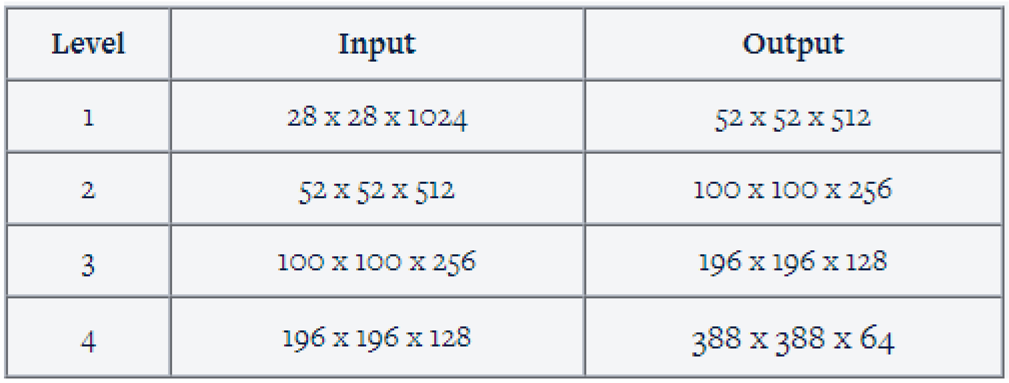
Bottleneck is a simple convolutional layer that generates more feature maps. However, there is no pooling layer to follow. As a result, you can see that the output now holds 1024 feature maps and the dimensions have reduced due to the convolutional layers.

3.Decoder/Expansion:

The last phase is responsible for generating the pixel map. In this process, the decoder also upsamples the output from the bottleneck to match the dimensions of the input image.

- As the first step, this block is upsampled to size 56x56 and depth decreases from 1024 to 512 feature maps.
- The upsampled output is combined with the output of the encoder phase at the same level through the skip connection which results in a feature map of size 56x56x1024. This process helps in reconstructing the image as concatenation will help in preserving the information at that level.
- The concatenated set of 1024 filters in then through 2 convolutional layers which reduce the depth to 512 feature maps.

The above process is repeated till you have an equivalent number of feature maps on both sides.



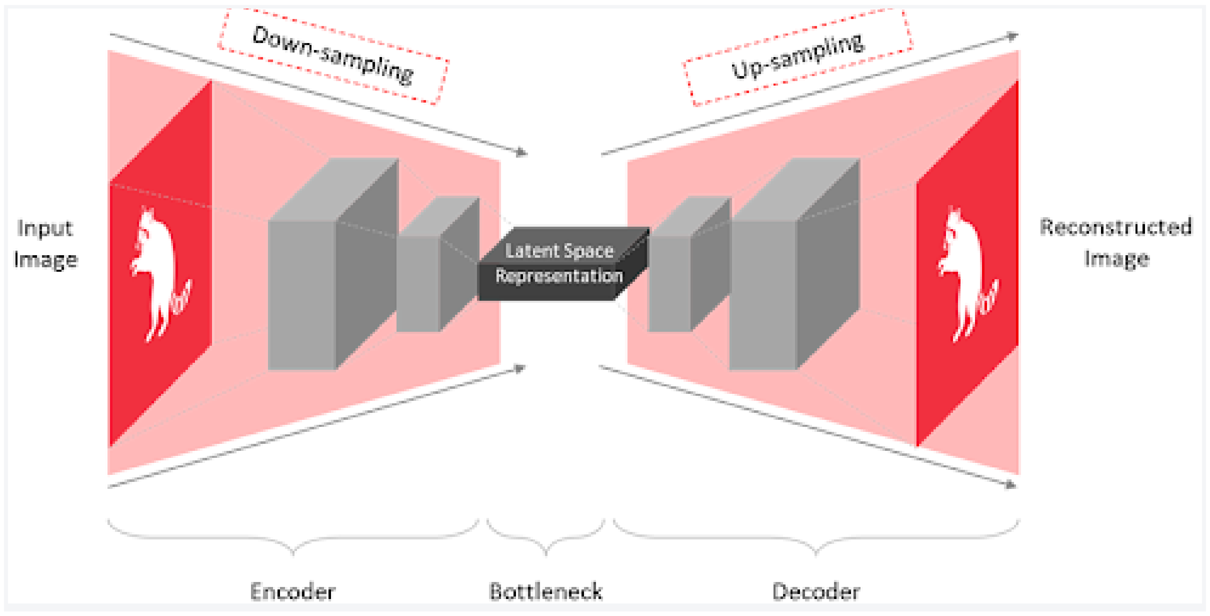
Level	Input	Output
1	28 x 28 x 1024	52 x 52 x 512
2	52 x 52 x 512	100 x 100 x 256
3	100 x 100 x 256	196 x 196 x 128
4	196 x 196 x 128	388 x 388 x 64

Upsampling

The architecture is based on two key processes of downsampling and upsampling.

Downsampling:
The encoder is responsible for the process of downsampling which is the same as the convolutional neural network. The pooling layers used in the architecture reduce the size of the input at each level. Resulting in a spatially reduced output

Upsampling in detail:
Upsampling path is where the output segmentation mask is being reconstructed with the aid of the skip connections.



The semantic segmentation algorithm focuses on two parts:

1. What: Classes that the architecture is expected to identify
2. Where: Position (pixels) of the object in the image.

The encoder answers the ‘what’ component as it is able to extract the features to identify the classes, While it progressively loses information as to WHERE it is to be found. This is mainly due to the iterative planar reduction of the feature maps via pooling. The objective of semantic segmentation is to improve the answer to both the questions - ‘What’ and ‘Where’. Therefore, the process of upsampling is required to map the identified features back to their position in the image. The following upsampling techniques can be used:

- Bi-linear interpolation
- Cubic interpolation
- Nearest neighbor interpolation
- Unpooling
- Transposed convolution

Transposed convolution:
The upsampling technique is the reverse of the convolution process. The convolution process results in the reduction of the size of the input, whereas, the transposed convolution method is used to upscale the image. The filter matrix used is transformed and then multiplied with the downsampled values to obtain the upsampled output.

Dice Coefficient

Every machine learning model is always accompanied by different evaluation metrics to evaluate the performance of the model. Previously we learned that pixel-wise cross-entropy loss can be used to evaluate the image segmentation models. The pixel-wise cross-entropy loss has different challenges associated with it. Since each pixel is examined individually, the function doesn't seem to perform well in the case of class imbalance due to unequal representation of the classes in the image. All these challenges gave rise to the Dice coefficient.

Dice coefficient is similar to the Intersection over Union (IoU) metric which tries to measure the overlap between the actual mask and the mask predicted by the architecture. As highlighted in the video, the value of the metric ranges 0 to 1.

- 1 denotes perfect and complete overlap between the ground truth and the model prediction. (Best case)
- 0 denotes that there is no overlap between the two (Worst case)

The dice coefficient loss function is defined as 1-DC Therefore, if the dice coefficient loss is used, the final model will try to minimize the given metric.

Lung Segmentation using U-Net - 1

This session is on practical understanding of the U-Net algorithm. We will learn how to use the framework to perform semantic segmentation over a collection of check X-ray images. The U-Net architecture was developed by Olaf Ronneberger, Philipp Fischer, Thomas Brox for the purpose of biomedical image segmentation.

Download the Google colab for the session.

The final objective of this demonstration is to build a semantic segmentation model that can segment the lungs in a chest X-ray image.

Start by loading the data into the colab environment.

The steps:

1. Install the Kaggle API to download the data set.
2. Defining the directories to store the data.

Kaggle API allows you to download and load the data set directly into the Google Colab notebook. Once you have established the connection with the Kaggle API, you can use the following code to download the data set:

```
!kaggle datasets download '<user_name>/<data_set_name>'
```

Note: The user_name above corresponds to the account which holds the dataset.

The following items to perform the segmentation task:

- Raw images (training and testing set).
- Annotation masks for each image.

3. Define the directories to store each element separately.
4. Temporary directory is also created to ease the process ahead.

As the zip file contains the raw images and the information of the masks associated with each of them. Therefore, the files are moved into the required directories for further processing. At this stage, the data set obtained from kaggle is raw and must be prepared according to the requirements of the algorithm.

Lung Segmentation using U-Net - 2

Before proceeding in the future, we must first prepare the data set to suit the requirements of the algorithm.
The U-Net architecture expects the following conditions to be true before you train a model:

1. Every source image present in the data set must have a corresponding mask image.

2. The source image and the corresponding mask image must have the same name to map them adequately.
3. All the pixels present in the masked images must be mapped to a particular value in order to distinguish one class from another in the image. For example, in the case of lung segmentation, you can encode the pixel values in the binary format as 0 for the background and 1 for lungs.

The first step is to prepare the data as it dealt with the file names of the mask images. The user-defined function is iterated over all the images to remove the suffix'_mask' from the name of all the images.

We must make sure every image has an accompanying mask image. In order to verify, print out essentially the number of files in the raw images directory and in the mask directory. If the number of images and masks didn't match. It is not necessary that you always obtain masks for every image in the data set. In such cases, you have two options. You can either use an annotation tool to create the mask image or remove those images which do not have the mask associated with them.

The next step is to convert the mask images into the required format for the U-Net library. As we are dealing with only two classes (lungs and background), each pixel is mapped to either 0 (for background) or 1 (for lungs) in the above video based on the class to which it belongs. This is essential for the machine to distinguish between the classes during the training and the evaluation steps.

After compilation of above steps, we are ready to define the model to perform image segmentation on the images.

Lung Segmentation using U-Net - 3

As all the images in the data set have a corresponding mask image which also follows the required naming convention. We will proceed with steps associated with model building, training and evaluation.

Steps to build the model from Github repository:

1. Cloning the github repository for U-Net architecture.
2. Resizing the input images for the model.

3. Loading the U-Net architecture from the cloned repository.
4. Importing and instantiating the VGGNet as the base for the U-Net model.

As we are working with a predefined architecture built by another user. It saves a lot of computation time as you are not expected to define the layers of the architecture and the weights associated with each layer are also loaded along with the architecture. The model uses VGG-16 as the backbone for building the architecture.

After this step, we now have the model architecture ready for training. During the training process, the network will learn the weights associated with different layers to minimize the loss function. Before training the model, we need to create a temporary directory into the root path in order to store the model checkpoints.

The following parameters need to be defined to train the model:
1. Provide a path to train X-ray images.
2. Provide a path to mask images.
3. Model checkpoints directory path.
4. Provide the number of epochs.

The model is now trained to perform the task of lung segmentation on the test images.

Session summary

Introduction to Instance segmentation

This session we will cover Instance segmentation and its architecture "Mask R-CNN". As part of the object detection module. The following components of the architecture are explained in this session.

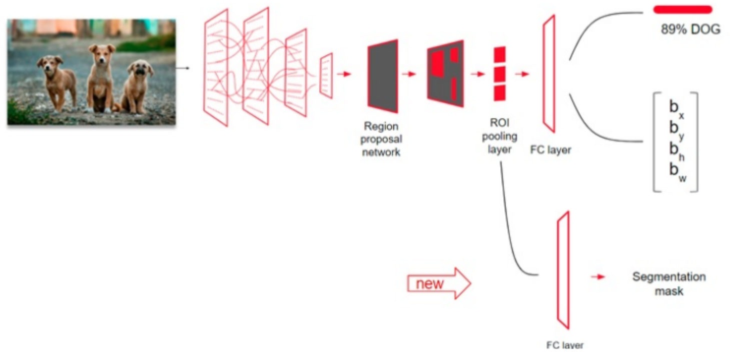
- Backbone model
- Region Proposal Network
- Regions of Interest (RoI Align)
- Segmentation mask

Session includes a case study based on oil tanker segmentation using the Mask R-CNN framework. As part of the exercise, you will learn how to obtain masks from the bounding boxes.

Mask R-CNN

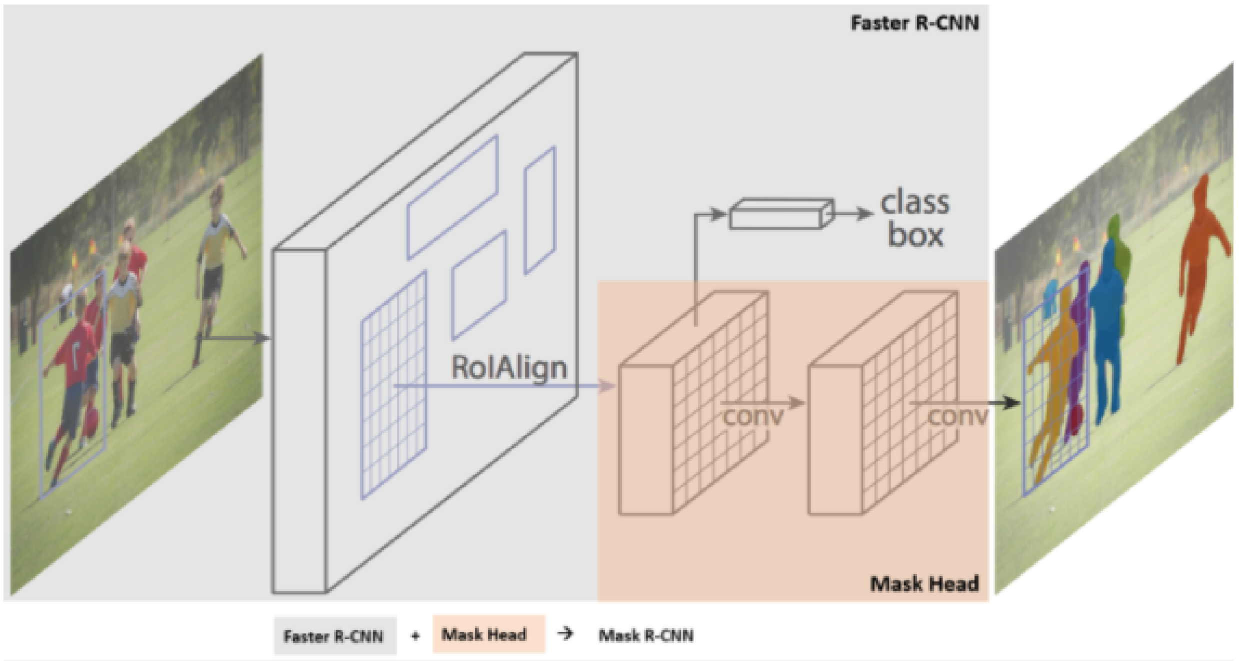
Mask R-CNN is a region-based convolutional neural network (CNN) that specialises in instance segmentation. This framework was developed by Kaiming He and a team of researchers at Facebook AI Research (FAIR) and is an extension of the Faster R-CNN architecture, which is used for object detection. The Mask R-CNN framework is on top of faster R-CNN.

Faster R-CNN provides two outputs for all the objects detected in the image: a class label and a bounding-box offset. Input images go through ConvNet, Region Proposal Network (RPN), RoI pooling layer, Fully connected layer. To generate class and box outputs.



Mask R-CNN adds another branch to the framework that is responsible for generating the object mask. Like the Faster R-CNN, Mask-CNN also have two stages:

- The first stage of the architecture is responsible for the generation of the proposals about the regions where the object might be present in the input image.
- In the second stage, the architecture uses the region proposals to produce the output by predicting the class of the object, refining the bounding box, and generating a mask at the pixel level of the object.



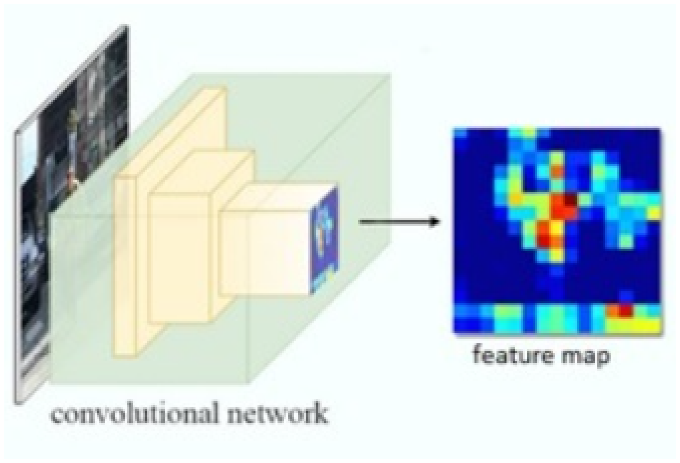
Source: Mask R-CNN (FAIR)

The next segment will cover the components of the mask R-CNN architecture.

Mask R-CNN: Architecture

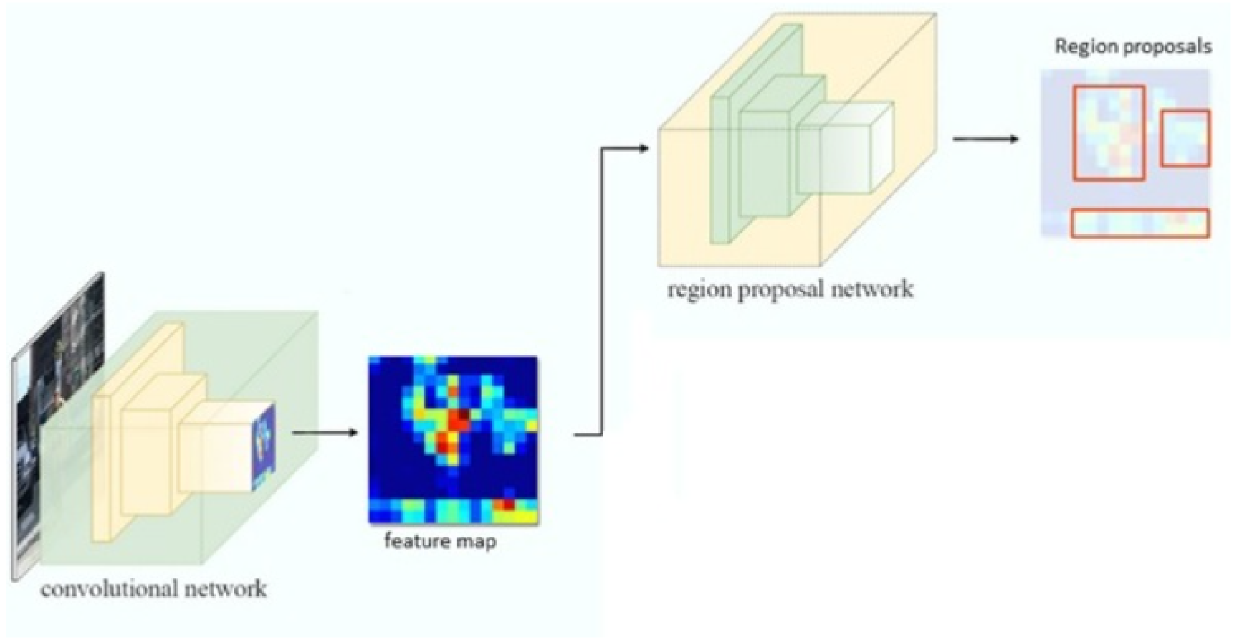
Mask R-CNN architecture comprises two stages, The different networks present in these two stages are:

- Backbone
- Region Proposal Network (RPN)
- Region of interest
- Segmentation Masks



Region Proposal Network (RPN)
RPN scans the image in a sliding window fashion (like filters in CNNs) and proposes the regions of the image which may contain objects. The network uses anchors to scan different parts of the image. They are a set of boxes with predefined sizes and scales relative to the images. The RPN generates two outputs for each anchor:

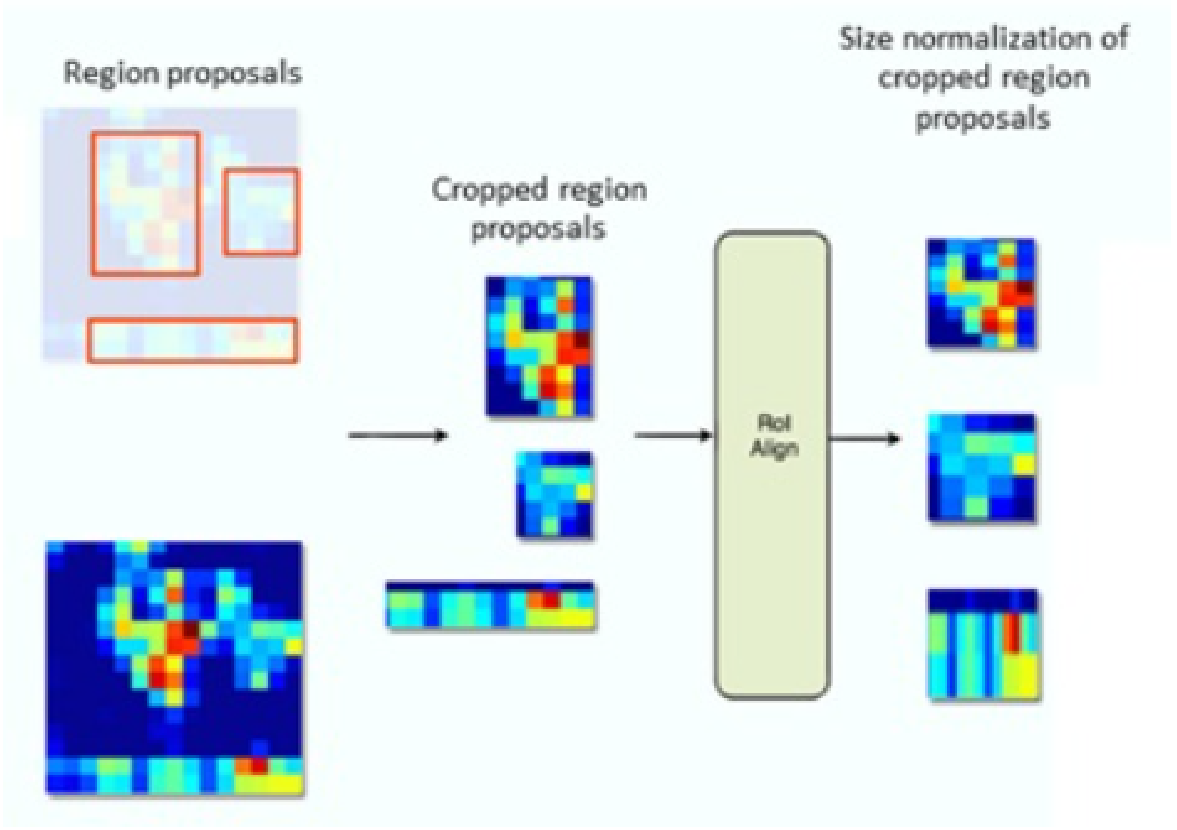
1. Anchor class: Foreground or background. The Foreground class implies that it is likely that an object is present in that anchor box.
2. Bounding Box Refinement: Spatial displacement and resizing. This is an estimation of percentage change in the dimensions and the position of the bounding box (x,y, width, height) to refine that anchor box to fit the object better.



Region of interest(ROI) classifier and bounding box regressor:
This network is part of the second stage of the architecture. There are two sources of inputs for this network. Backbone network provides the feature maps to classify the objects and the RPN is responsible for delivering the Regions of Interest (ROIs) to locate them in the image. Architecture is responsible for generating two outputs for each ROI:

- Class: This is the class of the object in the ROI
- Bounding box refinement: This is similar to the RPN as it also tries to refine the location and size of the bounding box to encapsulate the object.

ROI Pooling:
The ROI pooling layer helps in standardising the size of the ROIs obtained from the RPN by cropping a part of the feature maps and resizing them to a fixed size. The image below summarises the function of the layer.



Segmentation Mask:
The mask network is the addition to the Faster R-CNN that the Mask R-CNN paper introduced. This convolution network takes the output from the ROIAlign layer (or ROI Pooling) and is responsible for generating the segmentation masks for the regions identified by the ROI Regressor. The masks generated by this network are smaller in size (28 x 28 pixels) which helps in keeping the computation time in check. However, these masks are upsampled during model inference to fit the size of the ROI obtained from the previous layers. The image below summarises the entire architecture.

Oil Tanker Segmentation - 1

This segment is to follow with a practical demonstration of Mask R-CNN. Assert mapping is one of the key applications of image segmentation and this case study will help you implement it for mapping oil tankers present in the input image.

Google collab file can be available on the platform.

Instructions:

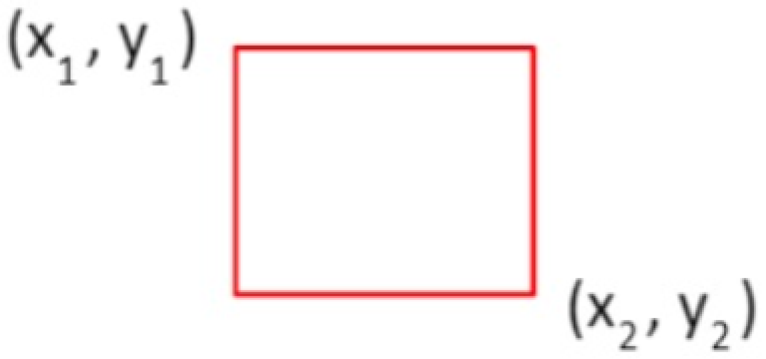
- Note that the runtime must be changed to 'GPU' for the purpose of this demonstration.
- You must run the code along with Georgios for a complete understanding of the concepts. However, you can skip the training process as it will take hours to complete.

Steps to setup google colab:

- Following libraries available in github are used for case study.
    - Mask R_CNN by Matterport,Inc.
    - Mask R-CNN extended for Tensor Flow2.x by Ahmed Gad.
    - Image to COCO JSON converter by chrise96
- Use Kaggle API to download the data set directly into the colab environment. The data set we are using is by Airbus Defense and Space Intelligence, and it is titled Airbus Oil Storage detection Dataset.
- Prepare the dataset for preferred architecture that is Mask R-CNN framework. Following are general steps for preparation.
    - The images must be resized for the architecture to process them faster. However, this must be done without sacrificing spatial resolution.
    - The data set must be split into training and validation sets for model training and evaluation purposes.
    - The annotation file associated with the training and the validation set must be in the designated format. As part of this case study, the MS COCO format must be followed as the pre-trained architecture is based on the same.

Oil Tanker Segmentation - 2

Prepare provided along with the data set to generate the masks for each image. Annotation file along with data sets coordinates for the top-left corner and the bottom-right corner of the image.



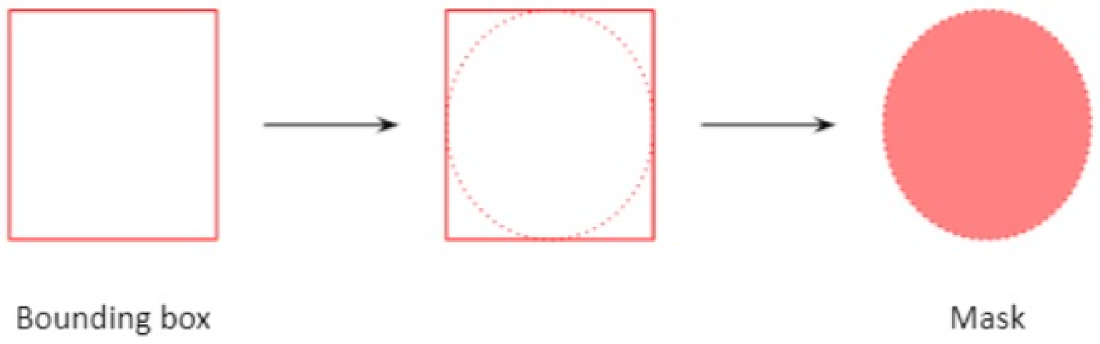
The annotation fil will transform into the following format for processing:

image_id	x	y	width	height	class_id

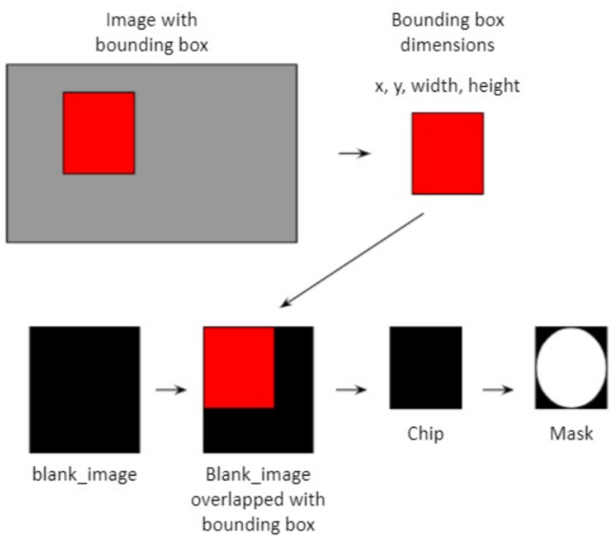
Where,

- ***image_id*** is the image name and the unique identifier of the image,
- **X** and ***y*** are the coordinates for the top-left corner of the bounding box,
- ***width*** and ***height*** capture the dimensions of the bounding box,
- ***class_id*** stores the class associated with the bounding box

Bounding boxes generate the masks for the images as shown in the graphics below.



The process of mask generation is summarised in the graphics below:



After following these steps, a data set has been created which contains the images and the corresponding masks. Remove the original data set to free up space.

Oil Tanker Segmentation - 3

As we generated the masks for each image, now we need to split the data into training and validation sets for further processing.

Generate the training and validation sets using the training_test_split function in a 9:1 ratio. Store each set in a separate folder as the path must be provided separately during the model training process.

In Order to perform the task of instance segmentation, Mask R-CNN expects the raw images along with the vector annotations in MS COCO compatible format.

The following are five blocks of information present in the annotation file:

- Info
- License
- Image: stores information about the images like file name, height, width, ID, etc.
- Categories: Contains information about all the class labels and the associated ID.
- Annotation: Information about the masks associated with an image.

The required file can be downloaded from the GitHub repository shared in the platform. Update the file attributes according to the problem at hand. Provide the directory path of the masks, image names, image IDs and the tuple the store the color associated with the background class. To train and validate sets.

Oil Tanker Segmentation - 4

Follow the steps to build Mask R-CNN architecture, which should be modified for the task of oil tanker segmentation.

- The first step clones the code for Mask R-CNN into the Colab environment. This template will now be altered to suit the requirements of the case study.
- Next, the second step helps in downloading the weights of the model trained on the COCO data set for the process of transfer learning. These parameters will be the initial weights during the training process.

As we are working with a pre-written code to train the Mask R-CNN model. Following adjustments need to be made before the training process.

- The data set format must be consistent with the template code used to read and process the images.
- The model parameters (like input size, classes, number of classes, etc.) must be tuned to suit the requirements of the final objective.

After following the necessary steps, the data set has been transformed into the required format. Both the training and the validation sets are now ready for model training. However, as an additional step, you can also implement data augments to prevent overfitting. Date augmentation creates separate copies of the image after introducing variations in scale, position, rotation, etc.

Now you have the raw images and the annotation file ready for the purpose of model training. As part of the case study, you will be working with an existing repository built for Mask R-CNN architecture, which will be modified for the task of oil tanker segmentation.

Oil Tanker Segmentation - 5

In order to prepare the model to perform the task of oil tanker segmentation. Some of the key model parameters need to be modify, parameters are:

- Backbone (ResNet50, ResNet101,etc.)
- Number of classes
- Computational parameters like the size of the GPU, batch size, etc.

After following necessary steps, the final model will be ready for training. The config class is used to set the model parameters. Moreover, the model is also initialised with the weights from the COCO dataset. The model summary highlights that the architecture runs very deep and there are approximately 44.66 million parameters to be learnt during the training process. The training process will be implemented multiple times after altering a few parameters like learning rate, trainable layers, etc. to optimal model.

The following three experiments need to be conducted as part of the case study:

- Training the initial layers with a higher learning rate.
- Training all the layers with a higher number of epochs.
- Training all the layers with a higher number of epochs and decreasing learning rate.

Oil Tanker Segmentation - 6

After following necessary steps from previous segments, our model will now be read to train and we will run three experiments in different versions in this case study. The following are the results of the experiment.

Experiment 1

The model was trained using the following parameters:

- Leaning rate: 0.0002
- Epochs: 6
- Layers: Head (FC layers were not trained)

The loss of the training set reduces with the increase in the number of epochs, however, in the case of the validation set, the loss tends to rise after a particular point. Therefore, the model needs modifications to improve its performance.

Experiment 2

The model was trained using the following parameters:

- Learning rate: 0.0001
- Epochs: 12
- Laters: All

This experiment shows similar results as experiment 1. The loss function has reduced but increases after a particular point. In the next experiment, the leaning rate is reduced for further epochs on top of the previous model.

Experiment 3:

The model was trained using the following parameters:

- Leaning rate: 0.00002
- Epochs:24
- Layers: All

The model is not able to achieve the targeted loss of 0.2. However, the model has shown improvement from its predecessors. The model can further be improved by altering different parameters like learning rate, dropout layers, normalization layers, etc. which can be explored at your end.

By understanding the performance of the model through the plots. We can see the loss curve for training and validation sets follow a similar pattern. However, the model seems to be stagnant at a particular point and would need further adjustments in the model parameters in order to improve further.

Oil Tanker Segmentation - 7

In order to perform the next step of model inference. We need to select one model from the previous three experiments to perform image segmentation, but only one model will be used for inference. It is clear that the third model performs best but you still need to find the best iteration of epoch within the model. The simple approach is to identify the iteration with the minimum validation loss. The difference in the output of these iterations is the weights learnt during the process of model training.

In order to obtain the weights to implement the model on another image, the code scans through the logs generated during the training process to identify the best weights. Completion of each iteration generates a checkout which is stored in the log directory and can be used later to load the weights for the model implementation.

The model performs the following three outputs:

- Bounding box.
- Label with confidence score that the object is detected correctly.
- Segmentation mask.

According to the case study, the results of the model are both positive and negative. The segmentation in some images is clear, however, There is a spillover in multiple instances. One of the reasons for the spillover can be that the generation of segmentation masks depending on the bounding box is not provided as the start of the training purpose, but generated using code. Coming to the final output, you can alter different parameters such as threshold value to improve the results. You can decrease the value to include more anchor boxes in the final result depending on the use case. With this, you have come to the end of the case study.