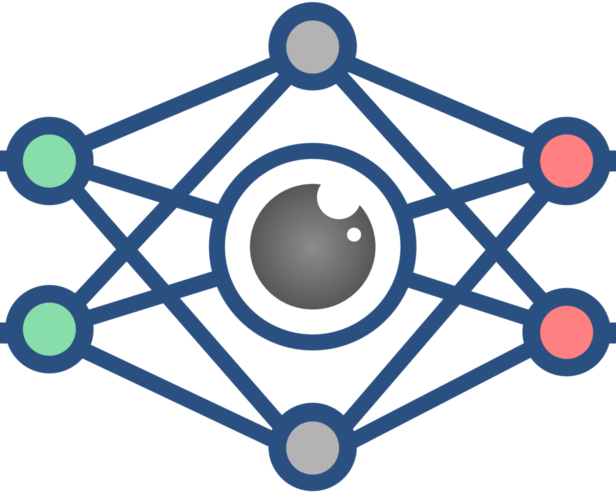
CS3485

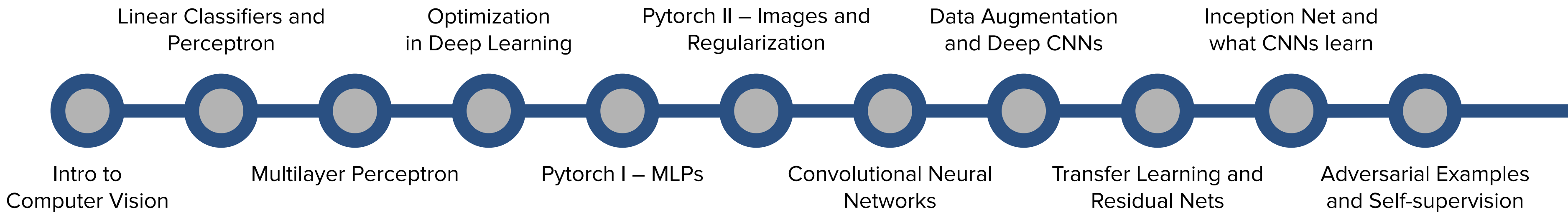
Deep Learning for Computer Vision



Lec 15: Intro to Image Segmentation

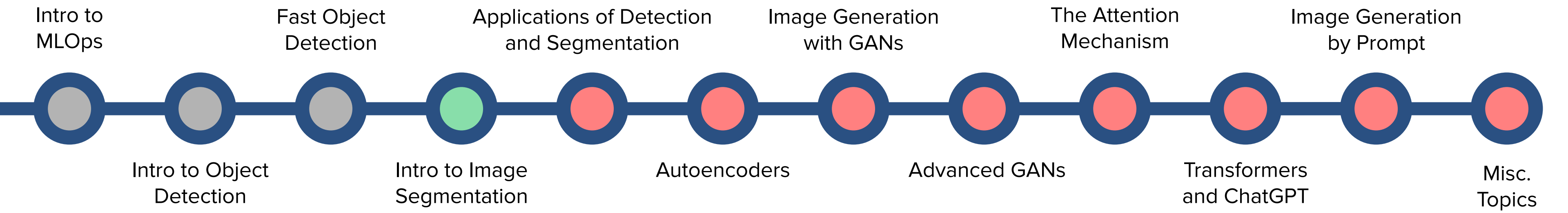
(Tentative) Lecture Roadmap

Basics of Deep Learning



- Intro to Computer Vision
- Linear Classifiers and Perceptron
- Multilayer Perceptron
- Optimization in Deep Learning
- Pytorch I – MLPs
- Pytorch II – Images and Regularization
- Convolutional Neural Networks
- Data Augmentation and Deep CNNs
- Transfer Learning and Residual Nets
- Inception Net and what CNNs learn
- Adversarial Examples and Self-supervision

Deep Learning and Computer Vision in Practice



- Intro to MLOps
- Intro to Object Detection
- Fast Object Detection
- Intro to Image Segmentation
- Applications of Detection and Segmentation
- Autoencoders
- Image Generation with GANs
- Advanced GANs
- The Attention Mechanism
- Transformers and ChatGPT
- Image Generation by Prompt
- Misc. Topics

What we’ve seen so far

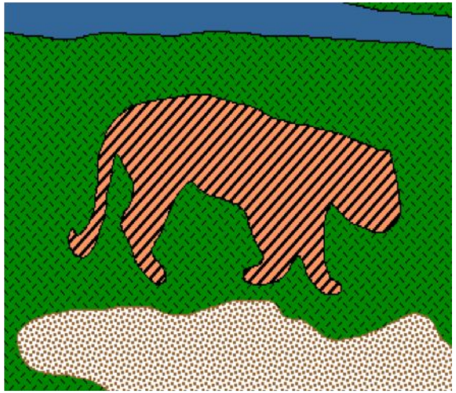
- In the previous lectures, we learned about detecting objects present in images, along with the classes that correspond to the detected objects.
- Today, we will further that by not only drawing a bounding box around the each object but also by identifying the exact pixels/image regions that contain an object.
- In Computer Vision, this task is called **Image Segmentation**:

  Image Segmentation is the task in which a image is partitioned into subgroups of pixels called segments or regions.

- This process also reduces the image’s complexity, making analysis **simpler**.

Original Image

Segmentation in 4 regions

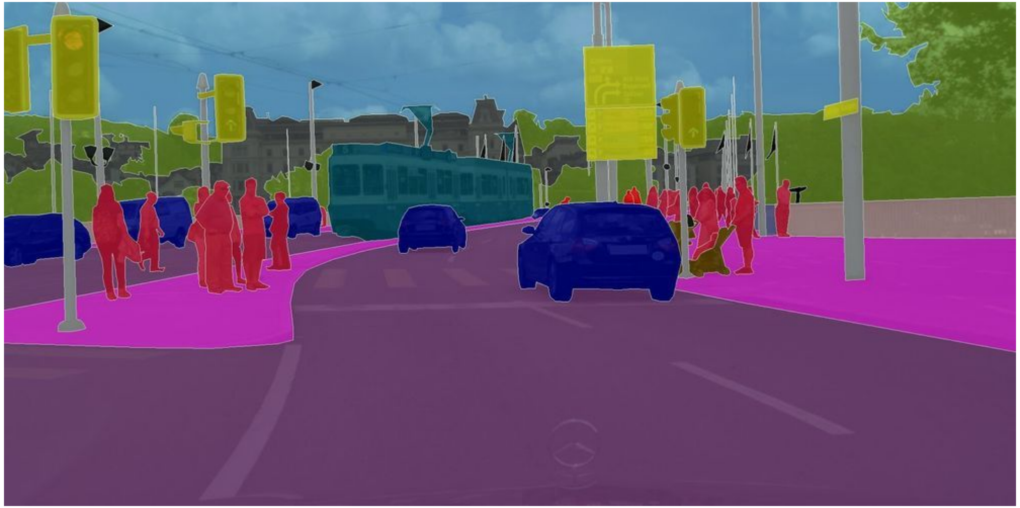


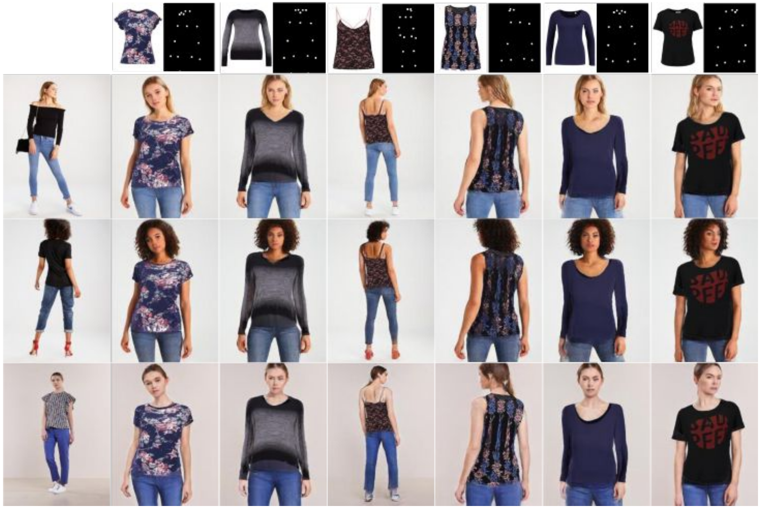
Applications of Image Segmentation

- As with Object Detection, Image Segmentation has some interesting applications:

Self-driving Cars: image segmentation is used to identify lanes and other necessary information

Virtual try-on: Segmenting the customer's body pixels from the clothes they are wearing.

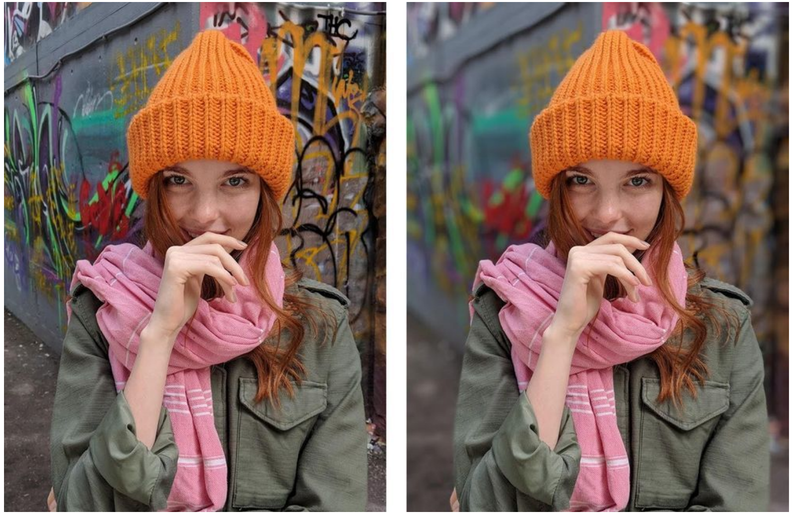




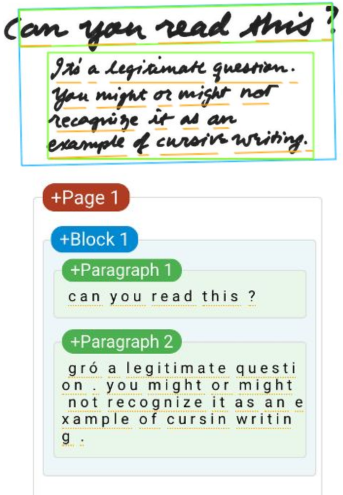
Applications of Image Segmentation

- As with Object Detection, Image Segmentation has some interesting applications:

Background Subtraction*: Useful in Video Conferences and Portrait mode



Handwritten text recognition



Can you read this?

It's a legitimate question. You might or might not recognize it as an example of cursive writing.

+Page 1

+Block 1

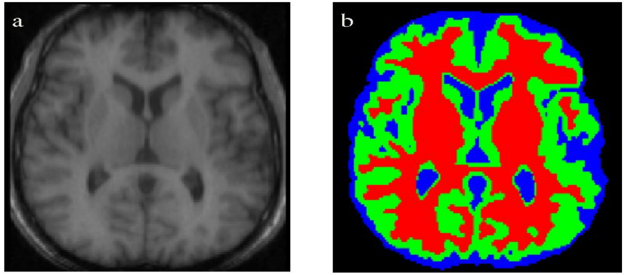
+Paragraph 1

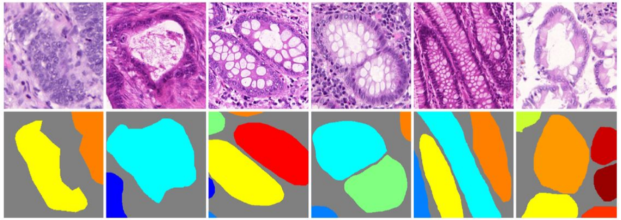
can you read this ?

+Paragraph 2

gró a legitimate question . you might or might not recognize it as an example of cursin writing .

Biomedical Image Understanding: MRI and, cancerogenous cell segmentation





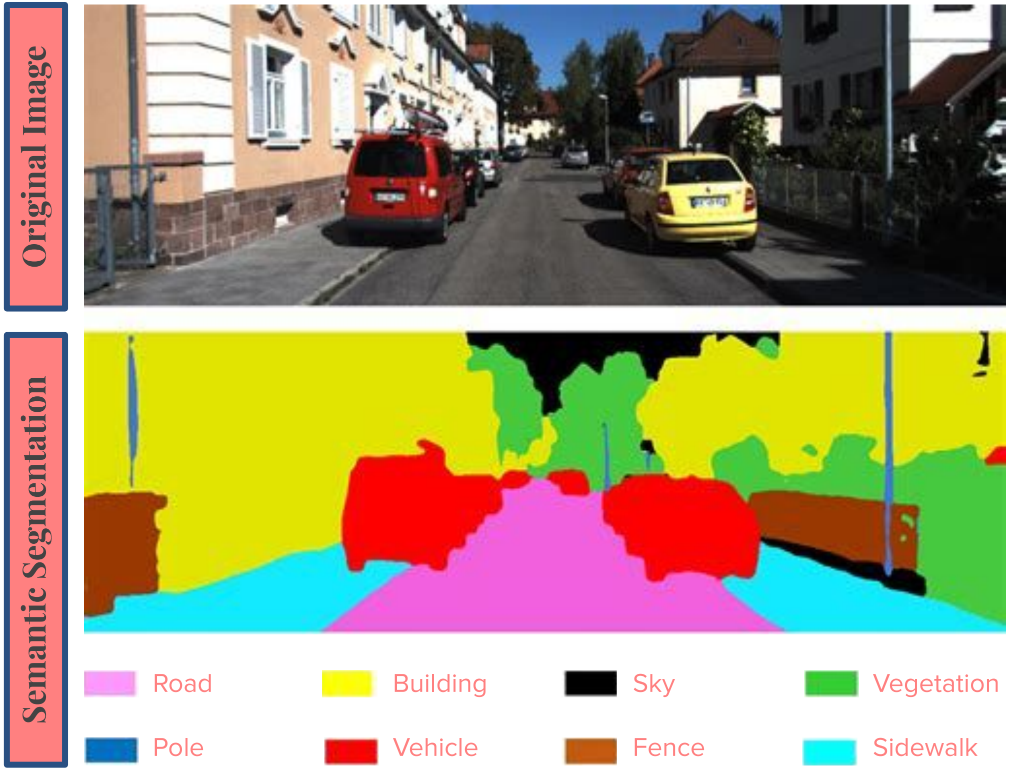
* Apple IOS even included it in one of its photo editing softwares! Check it out.

Semantic Segmentation

- In Deep Learning, one variant of image segmentation of particular interest is **Semantic Segmentation**:

  In Semantic Segmentation, each region corresponds to an object with semantic meaning, i.e., it has an specific class attached to it.

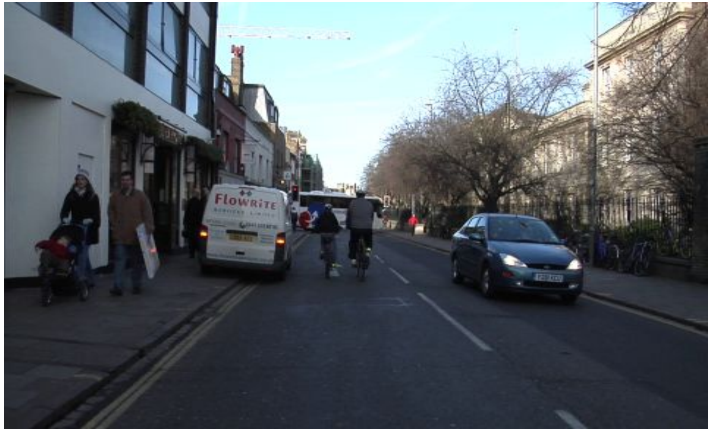
- Note that in non-semantic segmentation, image regions **don't need to** have an specific known semantic meaning in them.



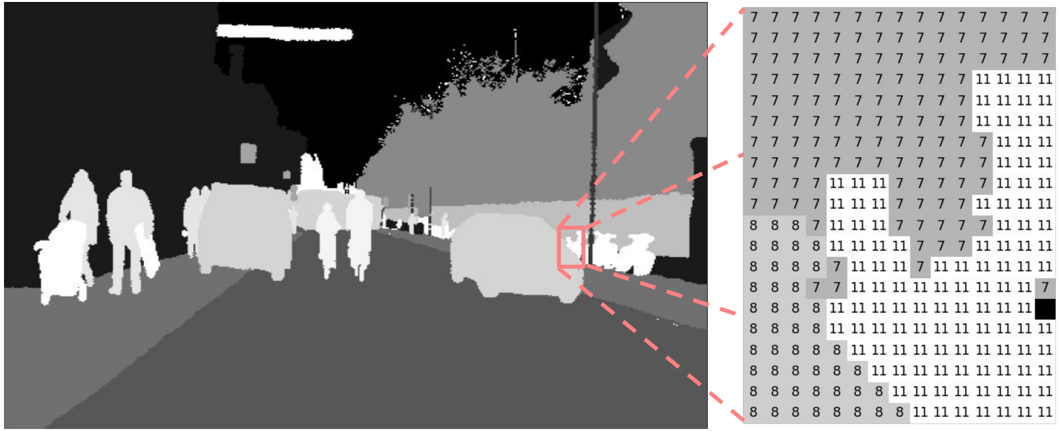
What type of data do we have here?

- In Supervised Image Segmentation, we have two pieces of data for training:
  - An image I, which can be grayscale or RGB
  - A segmentation map (same size as I), where each entry is a label of the corresponding pixel in I.

Original Image



Segmentation Map* for a problem with 11 possible different labels

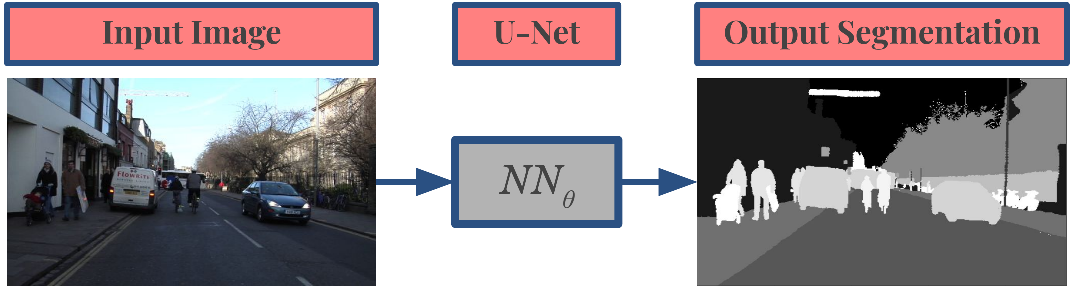


7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7	7	7	11	11	11	11
7	7	7	7	7	7	7	7	7	7	7	11	11	11	11
7	7	7	7	7	7	7	7	7	7	7	11	11	11	11
7	7	7	7	7	7	7	7	7	7	7	7	11	11	11
7	7	7	7	7	7	7	7	7	7	7	7	11	11	11
7	7	7	7	11	11	11	7	7	7	7	7	11	11	11
7	7	7	7	11	11	11	7	7	7	7	11	11	11	11
8	8	8	7	11	11	11	7	7	7	7	7	11	11	11
8	8	8	8	11	11	11	11	7	7	7	11	11	11	11
8	8	8	8	7	11	11	11	7	11	11	11	11	11	11
8	8	8	7	7	11	11	11	11	11	11	11	11	11	7
8	8	8	8	11	11	11	11	11	11	11	11	11	11	
8	8	8	8	11	11	11	11	11	11	11	11	11	11	11
8	8	8	8	8	11	11	11	11	11	11	11	11	11	11
8	8	8	8	8	8	11	11	11	11	11	11	11	11	11
8	8	8	8	8	8	8	11	11	11	11	11	11	11	11
8	8	8	8	8	8	8	8	11	11	11	11	11	11	11

* These matrices are not Grayscale images and their values should range from 1 to K (total number of possible labels in the problem).

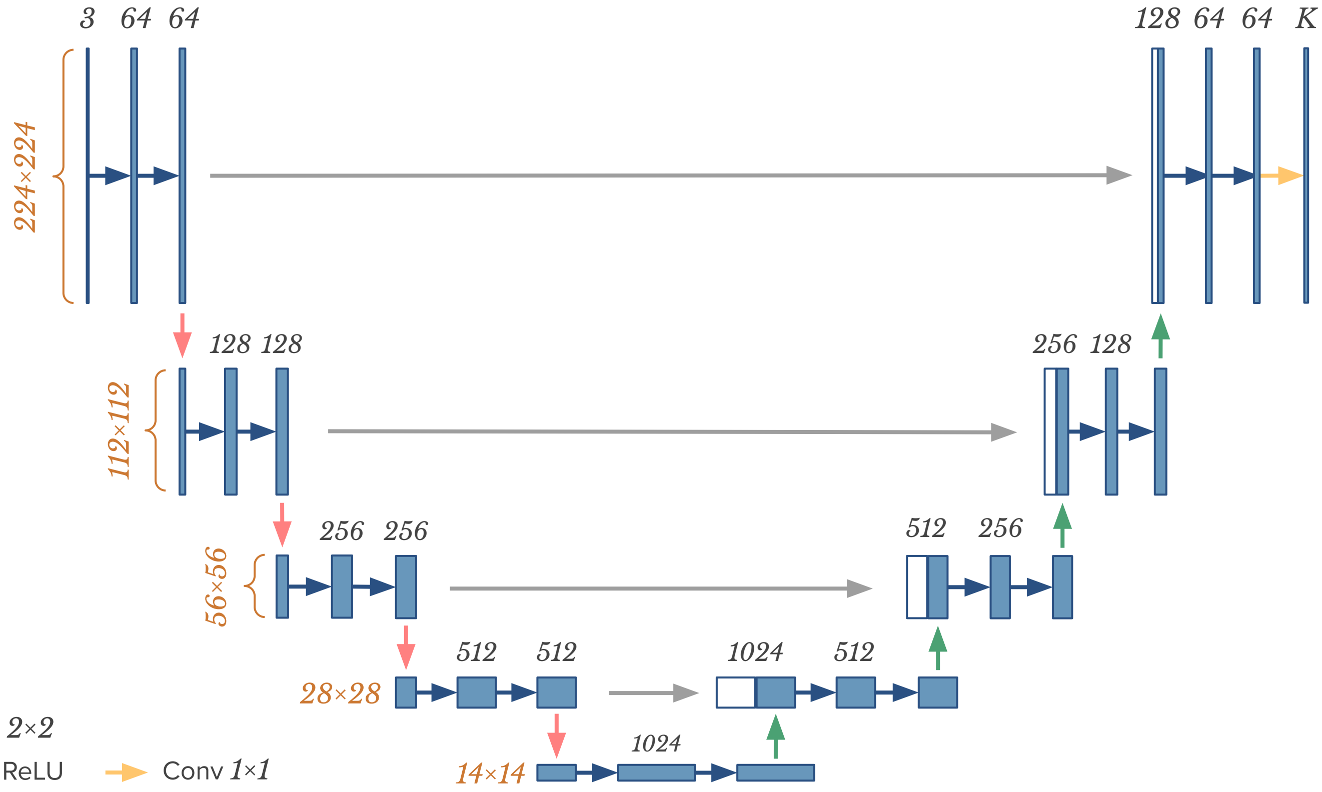
What type of Neural Network do we need here?

- Note that the input should have the same **height and width** as the input.
- This means that the neural network NN_θ we need here can be a **Fully Convolutional Network** (FCN), just like in YOLO, i.e.,
  - The input of the neural network is a tensor (a RGB image) of shape `(3,W,H)` and the output will be another tensor, now of shape `(K,W,H)`, with K channels (one for each segment class).
  - We don’t need dense layers, which are usually the source of most weights to be learned in the network.
- In this course, we’ll see one of the most successful FCNs in the literature called **U-Net**, published in 2015.
- Its name is due to its U shape.



U-Net

The U-Net architecture is is shown on the right. Each strip is a tensor (on a side-view) and the arrows represent tensor operations (like concatenation) or layers (like ConvLayers and Max Pooling).



Legend:

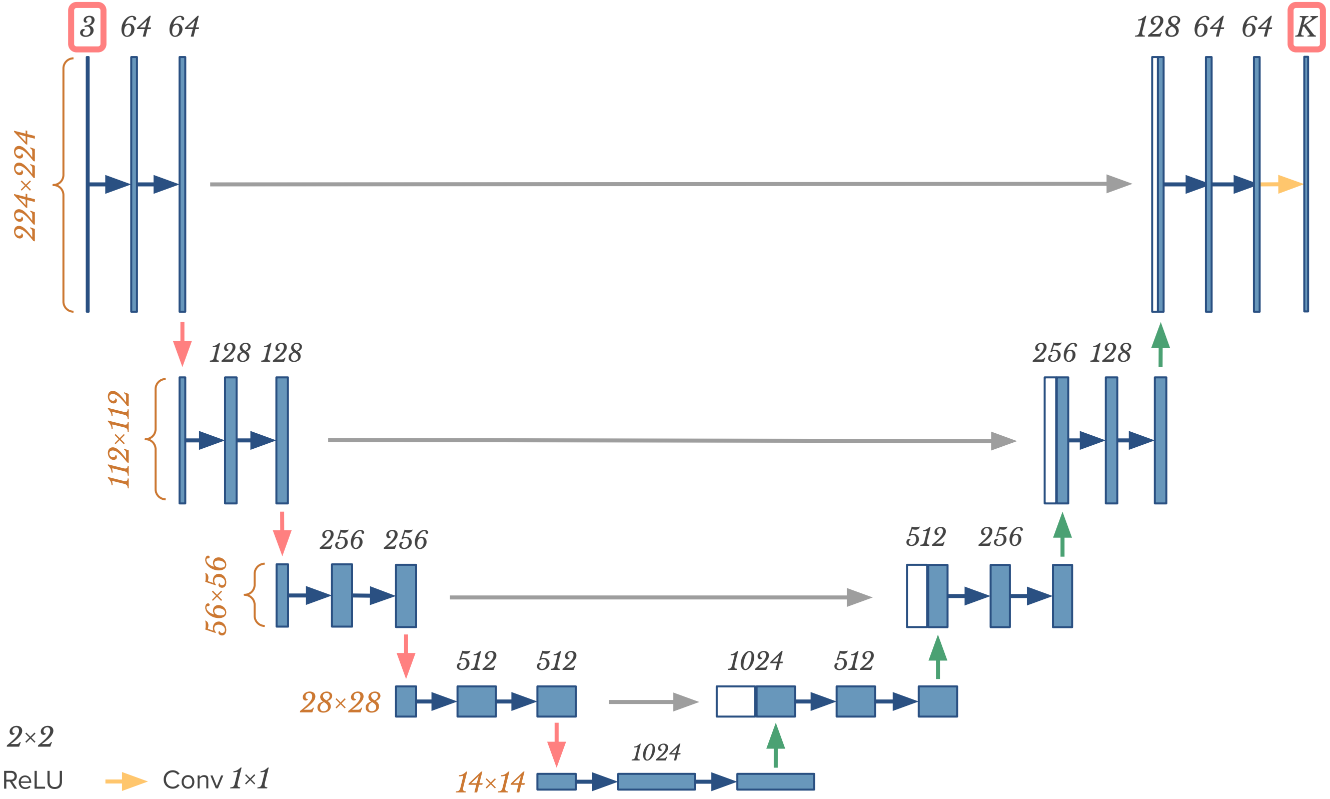
→ Conv 3×3 + ReLU → Max-pool 2×2

→ Concat. → Transp. Conv 2×2 + ReLU → Conv 1×1

U-Net

The numbers on top of each strip is the number of channels of that tensor and the numbers on the side are its heights and widths.

Note that we start we an RGB image (3 channels) and end up with a tensor of K channels (one for each segment class).



Legend:

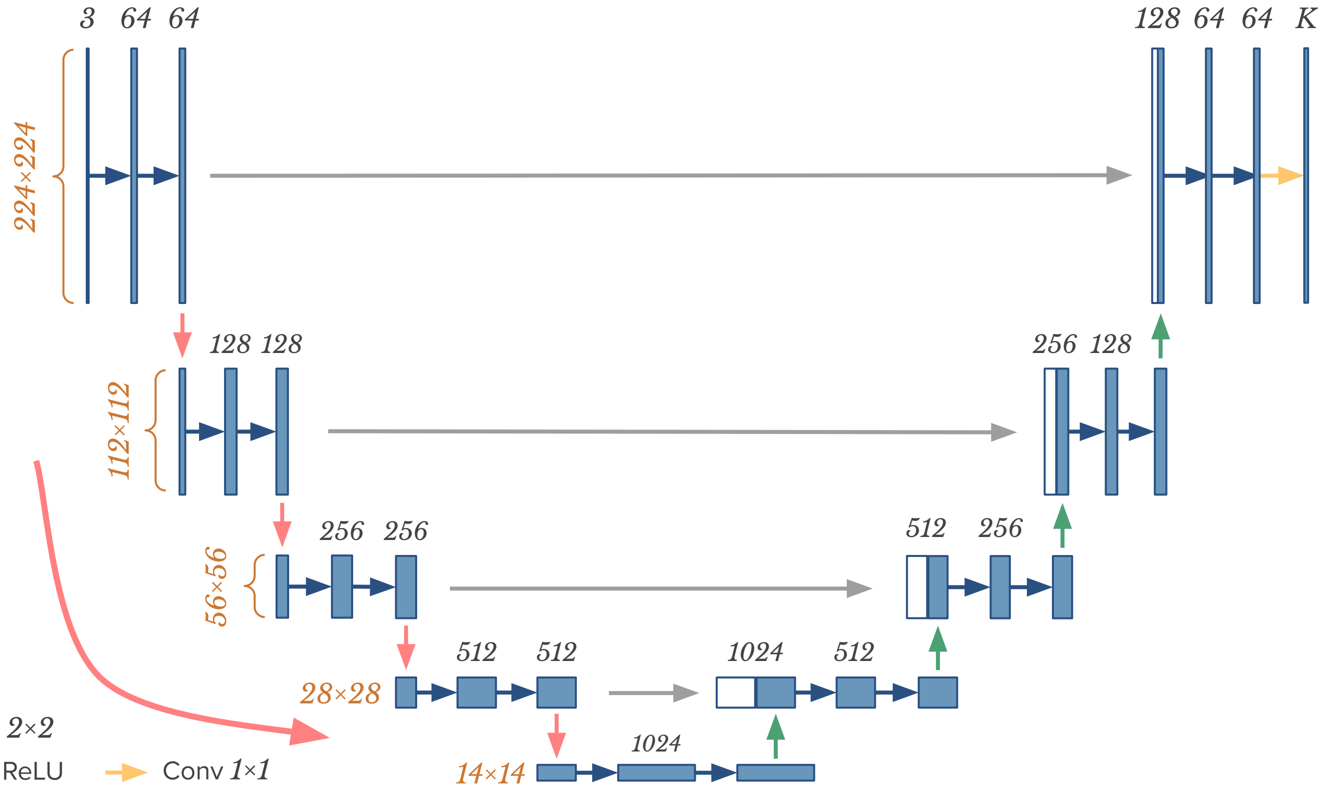
→ Conv 3×3 + ReLU → Max-pool 2×2

→ Concat. → Transp. Conv 2×2 + ReLU → Conv 1×1

U-Net

The main idea is that the original image gets “smaller” in terms of height and width, but “thicker” in regard the number of channels. This is called **Downsampling**.

It “compresses” / **encodes** the visual information its most important features.



Legend:

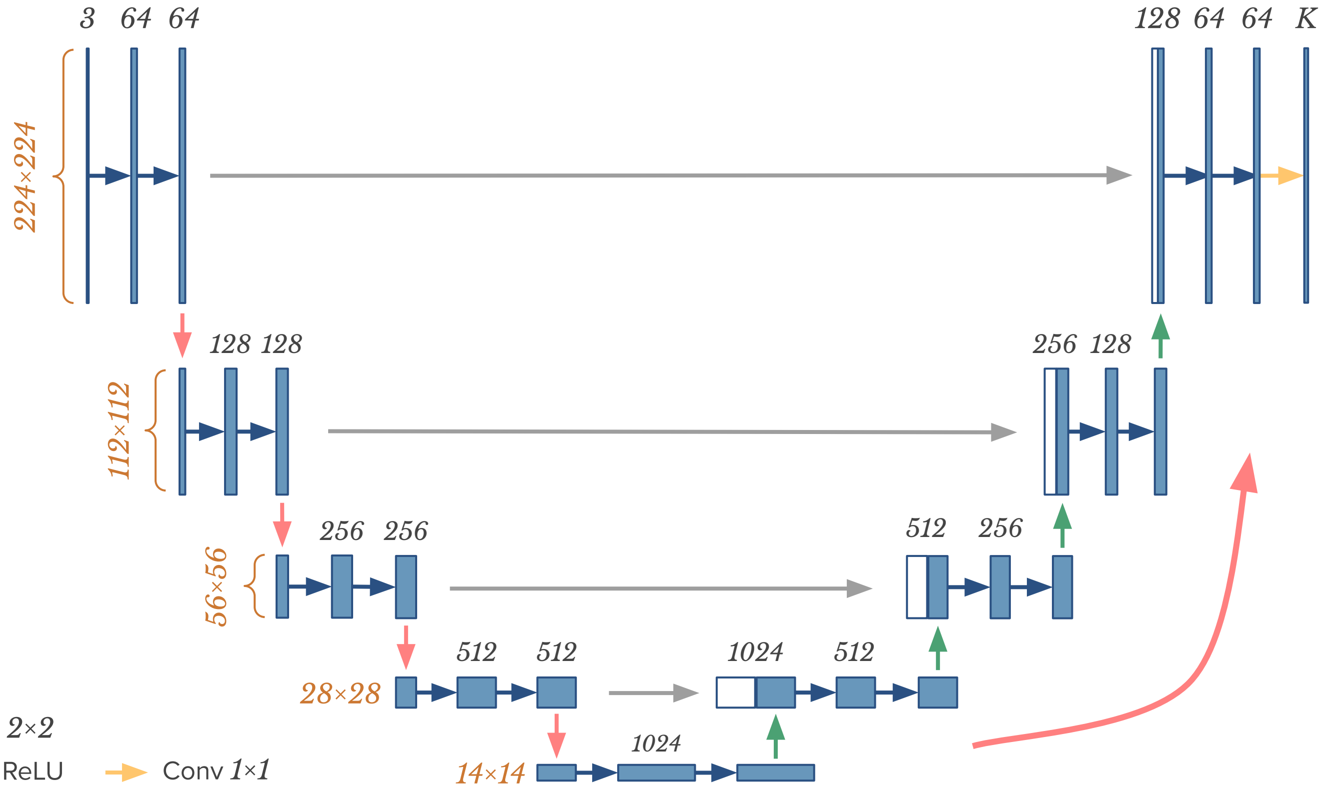
→ Conv 3×3 + ReLU → Max-pool 2×2

→ Concat. → Transp. Conv 2×2 + ReLU → Conv 1×1

U-Net

Then it gets thinner and larger, going back to its original shape. This process is called **Upsampling**.

It ensures that only useful information of the image is being “decompressed”/ **decoded** (*More on it later*). This is done via **Transpose Convolutions**.



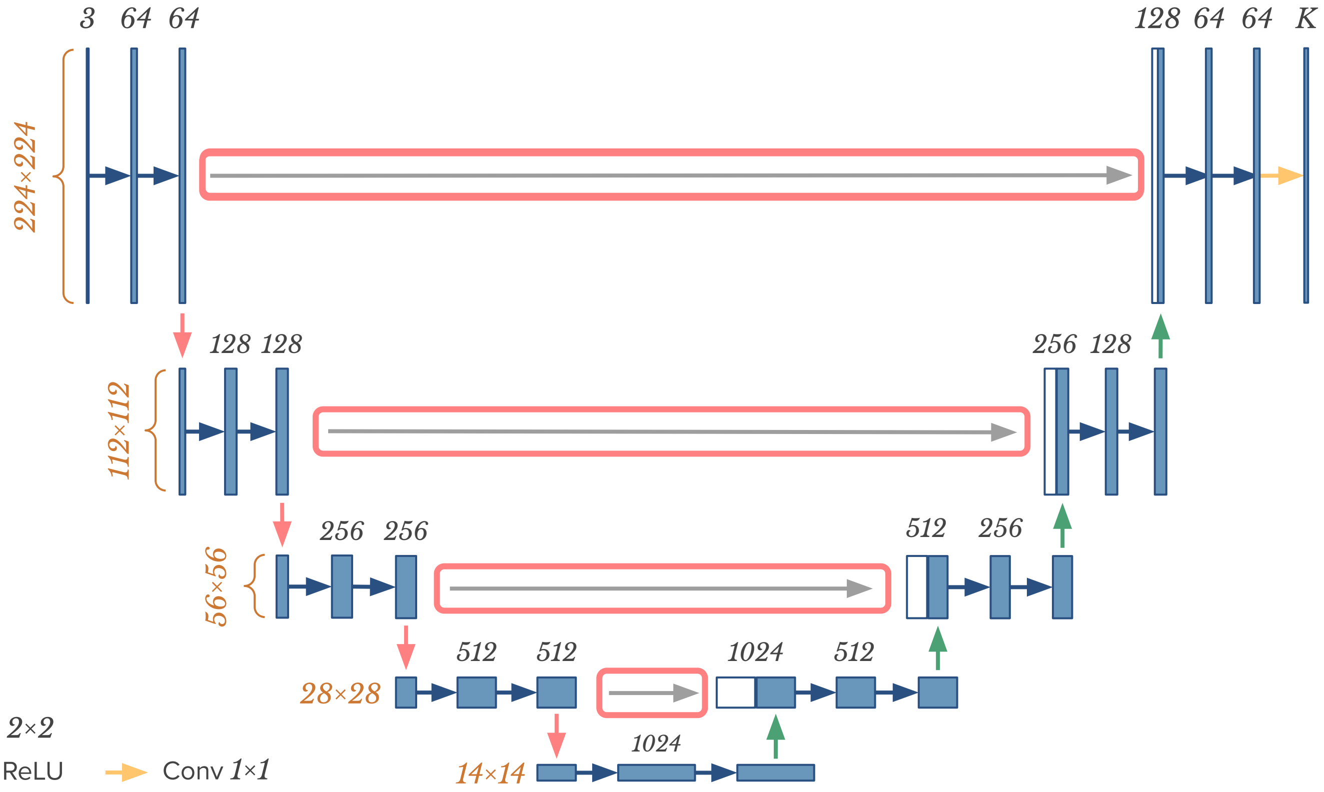
Legend:

→ Conv 3×3 + ReLU → Max-pool 2×2

→ Concat. → Transp. Conv 2×2 + ReLU → Conv 1×1

U-Net

U-Net also presents these **skip connections** that: (1) as with ResNets, mitigate the issue of vanishing gradients and (2) add extra information to the decoder that might be lost because of the downsampling on the encoder side of the network.



Legend:

→ Conv $3{\times}3$ + ReLU → Max-pool $2{\times}2$

→ Concat. → Transp. Conv $2{\times}2$ + ReLU → Conv $1{\times}1$

The Transpose Convolution operation

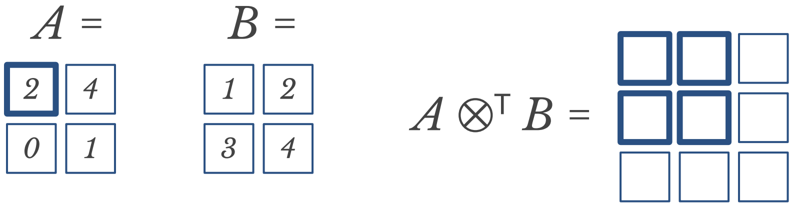
- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here’s an example of this process when the stride is 1 for when A and B are as below:

$$A = \begin{array}{|c|c|} \hline 2 & 4 \\ \hline 0 & 1 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}$$

$$A \otimes^{\mathsf{T}} B =$$

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:



$$A = \begin{bmatrix} \boxed{2} & 4 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad A \otimes^{\mathsf{T}} B = \begin{bmatrix} \; & \; & \; \\ \; & \; & \; \\ \; & \; & \; \end{bmatrix}$$

We start by filling the output matrix from its top left corner.

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:

$$A = \begin{bmatrix} \boxed{2} & 4 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} \boxed{1} & 2 \\ 3 & 4 \end{bmatrix}$$

$$A \otimes^{\mathsf{T}} B = \begin{bmatrix} \boxed{2} & \boxed{} & \\ \boxed{} & \boxed{} & \\ & & \end{bmatrix}$$

$$2 \times 1 = 2$$

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:

$$A = \begin{bmatrix} \boxed{2} & 4 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & \boxed{2} \\ 3 & 4 \end{bmatrix}$$

$$2 \times 2 = 4$$

$$A \otimes^{\mathsf{T}} B = \begin{bmatrix} \boxed{2} & \boxed{4} & \\ \boxed{} & \boxed{} & \\ & & \end{bmatrix}$$

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
    - Multiplying each element of A by the **whole** of matrix B (the kernel here).
    - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:

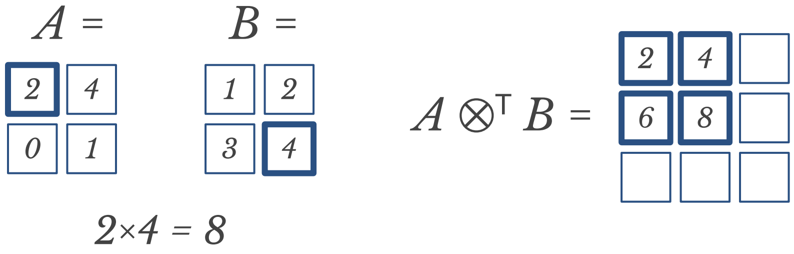
$$A = \begin{bmatrix} \boxed{2} & 4 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ \boxed{3} & 4 \end{bmatrix}$$

$$A \otimes^{\mathsf{T}} B = \begin{bmatrix} \boxed{2} & \boxed{4} & \\ \boxed{6} & \boxed{} & \\ & & \end{bmatrix}$$

$$2 \times 3 = 6$$

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:



$$A = \begin{bmatrix} \boxed{2} & 4 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & \boxed{4} \end{bmatrix} \qquad A \otimes^{\mathsf{T}} B = \begin{bmatrix} \boxed{2} & \boxed{4} & \\ \boxed{6} & \boxed{8} & \\ & & \end{bmatrix}$$

$2 \times 4 = 8$

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:

$$A = \begin{array}{|c|c|}\hline 2 & \boxed{4} \\ \hline 0 & 1 \\ \hline\end{array} \qquad B = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline\end{array}$$

$$A \otimes^{\mathsf{T}} B = \begin{array}{|c|c|c|}\hline 2 & 4 & \\ \hline 6 & 8 & \\ \hline & & \\ \hline\end{array} + \begin{array}{|c|c|c|}\hline & \boxed{4} & \boxed{8} \\ \hline & \boxed{12} & \boxed{16} \\ \hline & & \\ \hline\end{array}$$

We go to the next entry in A and, since the stride is 1, we move one entry to the right on the output. Then fill it like before.

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:

$$A = \begin{bmatrix} 2 & 4 \\ \boxed{0} & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A \otimes^{\mathsf{T}} B = \begin{bmatrix} 2 & 4 & \\ 6 & 8 & \\ & & \end{bmatrix} + \begin{bmatrix} & 4 & 8 \\ & 12 & 16 \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ \boxed{0} & \boxed{0} & \\ \boxed{0} & \boxed{0} & \end{bmatrix}$$

Repeat this process for the next entry in A and the next spots in the output according to the stride.

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:

$$A = \begin{bmatrix} 2 & 4 \\ 0 & \boxed{1} \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A \otimes^{\mathsf{T}} B = \begin{bmatrix} 2 & 4 & \\ 6 & 8 & \\ & & \end{bmatrix} + \begin{bmatrix} & 4 & 8 \\ & 12 & 16 \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ 0 & 0 & \\ 0 & 0 & \end{bmatrix} + \begin{bmatrix} & & \\ & \boxed{1} & \boxed{2} \\ & \boxed{3} & \boxed{4} \end{bmatrix}$$

Repeat this process for the next entry in A and the next spots in the output according to the stride.

The Transpose Convolution operation

- The typical way in Deep Learning to increase the size (length and width) of a tensor via Transpose Convolution operations.
- If we have two matrices, A and B, their transpose convolution convolution, denoted here as $A \otimes^{\mathsf{T}} B$, is computed via:
  - Multiplying each element of A by the **whole** of matrix B (the kernel here).
  - Placing the resulting matrix in a new matrix according with a predefined stride, summing entries whenever there is an overlap.
- Here's an example of this process when the stride is 1 for when A and B are as below:

$$A = \begin{bmatrix} 2 & 4 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A \otimes^{\mathsf{T}} B = \begin{bmatrix} 2 & 4 & \\ 6 & 8 & \\ & & \end{bmatrix} + \begin{bmatrix} & 4 & 8 \\ & 12 & 16 \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ 0 & 0 & \\ 0 & 0 & \end{bmatrix} + \begin{bmatrix} & & \\ & 1 & 2 \\ & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 8 & 8 \\ 6 & 21 & 18 \\ 0 & 3 & 4 \end{bmatrix}$$

Finally, sum all the matrices you've computed so far and that's your output!

The Transpose Convolution operation

- Note that the **value of the stride is crucial** when defining the output size of the transpose convolution operation.
- One particular setting of particular interest in Deep Learning is when we have a transpose convolution whose kernel is 2×2 and the stride is 2. For example:

$$A = \begin{bmatrix} 2 & 4 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A \otimes^{\mathsf{T}} B \text{ (Stride 2)} =$$

$$\begin{bmatrix} 2 & 4 & & \\ 6 & 8 & & \\ & & & \\ & & & \end{bmatrix} + \begin{bmatrix} & & 4 & 8 \\ & & 12 & 16 \\ & & & \\ & & & \end{bmatrix} + \begin{bmatrix} & & & \\ & & & \\ 0 & 0 & & \\ 0 & 0 & & \end{bmatrix} + \begin{bmatrix} & & & \\ & & & \\ & & 1 & 2 \\ & & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 4 & 8 \\ 6 & 8 & 12 & 16 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 3 & 4 \end{bmatrix}$$

- Note that the tensor size just doubled! This effect is more or less the opposite of 2×2 max-pooling's and is very useful in the U-Net's upsampling phase.

Transpose Convolution and U-Net in Pytorch

- Just as Pytorch defines `nn.Conv2d()` for convolutional layers, it has the module `nn.ConvTranspose2d()` for transpose convolutions:

```
nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, bias=True)
```

- Here, the parameters above* (`in_channels`, `out_channels`, `kernel_size`, `stride`, and `bias`) are of definitions similar to what is found in `nn.Conv2d()`, but with the operations that follow the procedure shown in the previous slides.
- Also, as in `nn.Conv2d()`, the weights to be learned in this layer are **the ones in the kernel** and the amount of them does not change with the input size, only with the input's number of channel.
- While the U-Net model is not pre-implemented in the Torch library, you can find it (along with **a bunch of other segmentation algorithms**) in the Segmentation Models library.

*Check the documentation here for more details on the layer and on other possible parameters.

Exercises (*In pairs*)

- Do the transpose convolution between A and B with stride 2:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \\ 3 & 3 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

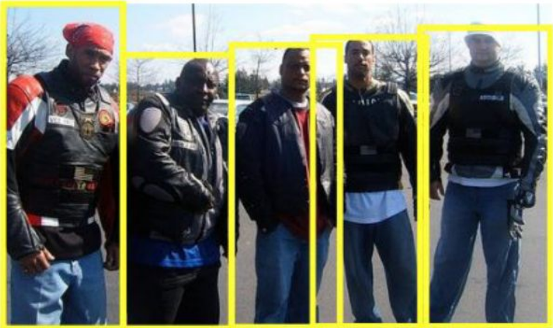
Instance Segmentation

- Another special type of segmentation is Instance Segmentation:

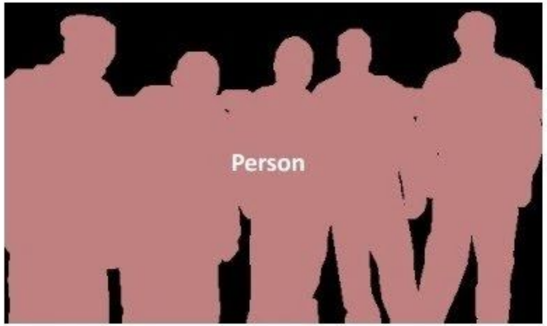
Instance segmentation is the task of detecting and delineating each distinct object of interest appearing in an image.

- It means that, **for each instance** of an object in the image, we want its segmentation.
- In other words, instance segmentation = object detection + semantic segmentation.

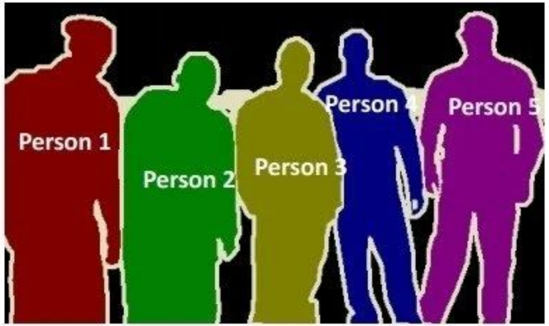
Object Detection



Semantic Segmentation



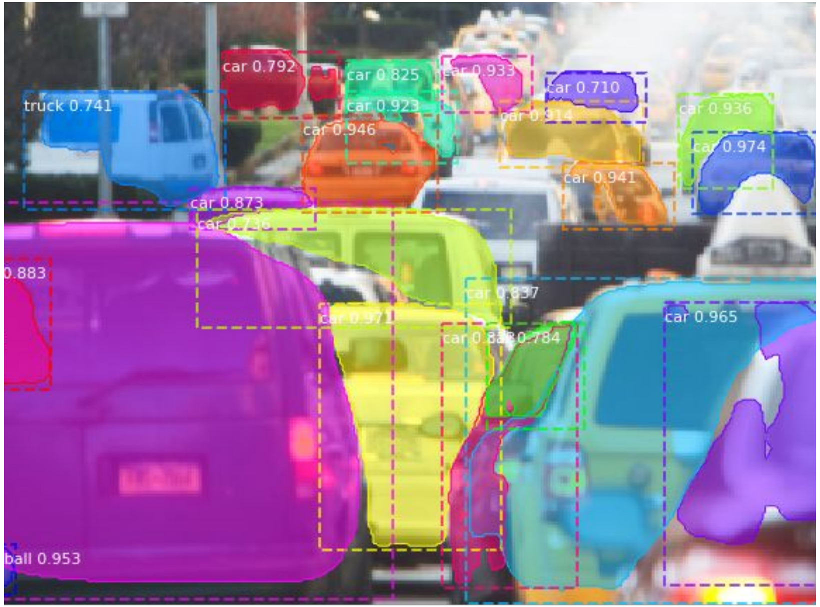
Instance Segmentation



Applications of Instance Segmentation

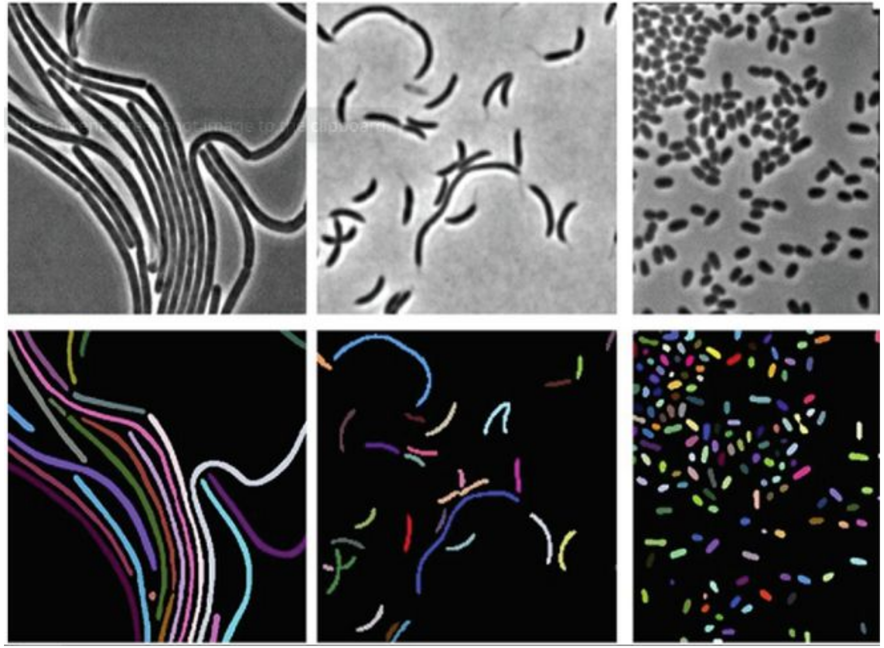
- Instance Segmentation is useful when distinct objects of a similar type are present and **need to be monitored separately**.
- It then has applications in, for example:
  - **Self-driving cars**: Keeping track of individual pedestrians and cars in videos.
  - **Medical scans**: In histopathologic images, we can segment different nuclei which can be further processed for the detection of dangerous diseases like cancer.
  - **Satellite imagery**: detection and counting of cars, ships detection for maritime security, and sea pollution monitoring

Instance segmentation on self driving cars

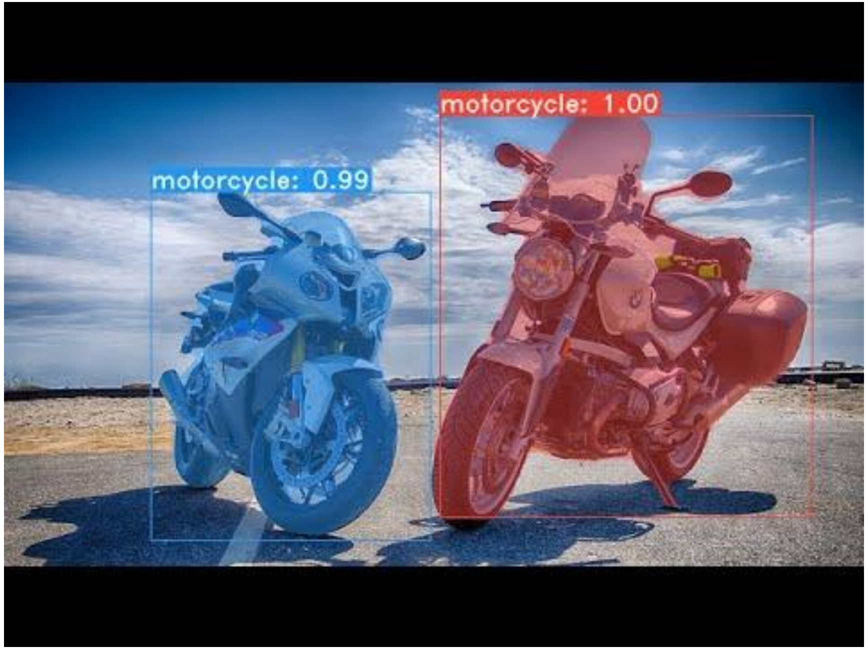


Applications of Instance Segmentation

Instance segmentation of different bacterial strains



Instance segmentation of in Videos

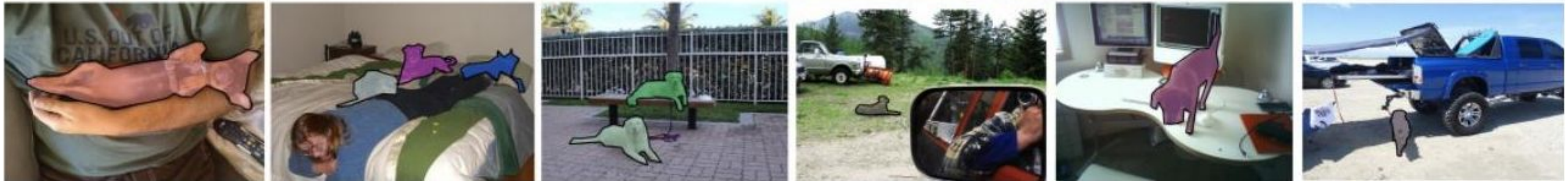


Data in Instance Segmentation

- In Instance segmentation, we need the following data to train a model:
  - The images
  - Object bounding boxes,
  - Instance-level segmentation ground-truth.
- A popular dataset that offers this kind of data (and many more*) is called **MS COCO (Microsoft Common Objects in Context)**. It contains a total of 328K images with annotated objects from 80 classes. Here are some images with annotated dogs:

COCO

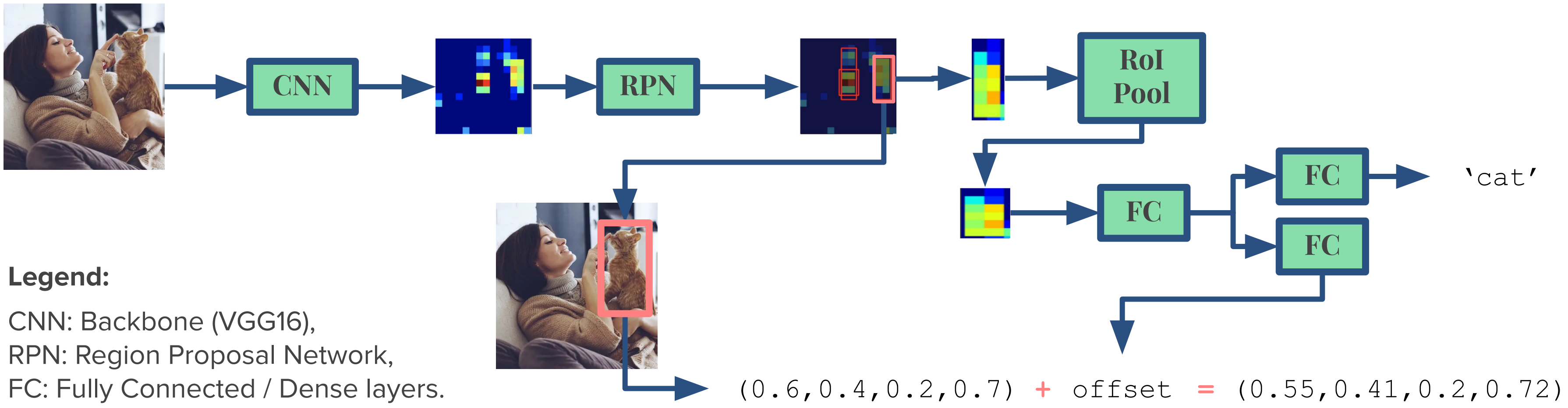
Common Objects in Context



* It also proved data for keypoint detection and image captioning (*more on these later in the course*).

Revisiting Faster R-CNN

- With such data we could Faster R-CNN (*shown below*) for object detection, for example.
- Faster R-CNN is also be the basis for Mask R-CNN, published by the same authors in 2017, which adapts Fast R-CNN to Instance Segmentation.

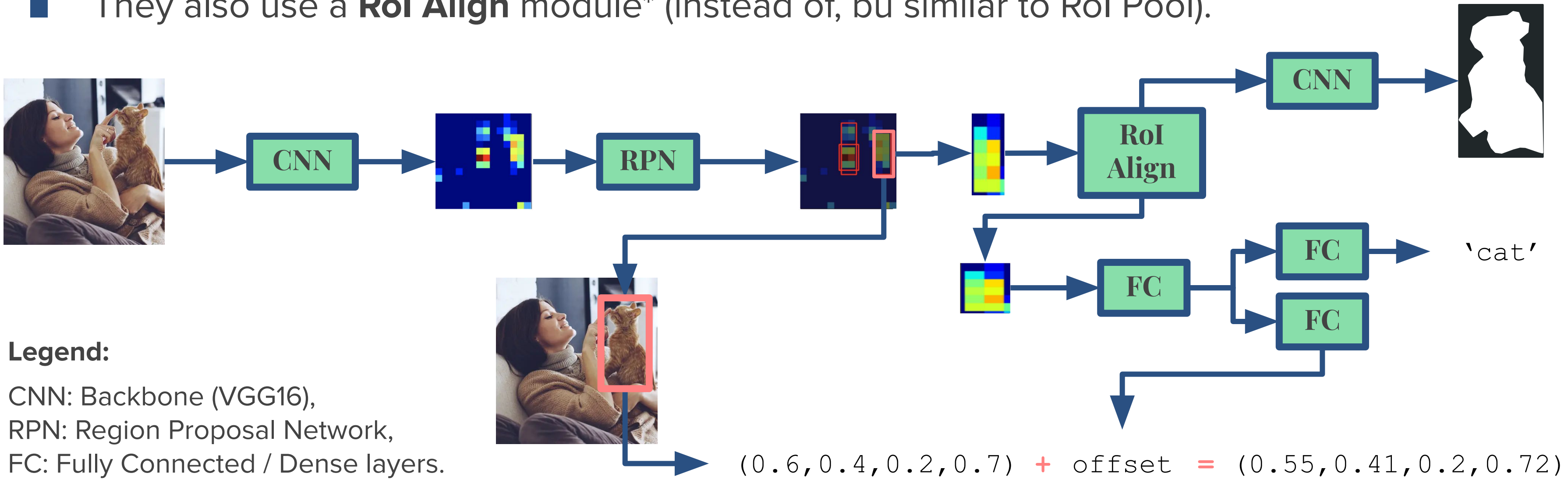


Legend:

CNN: Backbone (VGG16),
RPN: Region Proposal Network,
FC: Fully Connected / Dense layers.

Mask R-CNN

- The main difference between both methods is that Mask R-CNN adds a CNN module for segmentation after the features/region proposals RoI pooling phase.
- They also use a **RoI Align** module* (instead of, bu similar to RoI Pool).



Legend:

CNN: Backbone (VGG16),
RPN: Region Proposal Network,
FC: Fully Connected / Dense layers.

* RoI pool introduces slight object misalignments that don't affect object classification, but hinder pixel level segmentation. An object alignment step is necessary to improve segmentation (ref.).

Advantages of Mask R-CNN

- Some advantages of Mask R-CNN for the task of Instance Segmentation:
  - **Simplicity**: Mask R-CNN is simple to train.
  - **Efficiency**: The method is very efficient and adds only a small overhead to Faster R-CNN.
  - **Flexibility**: Mask R-CNN is easy to generalize to other tasks. For example, it is possible to use it for **pose estimation** in the same framework (*more on it later*).
- Mask R-CNN's pretrained model is also available in PyTorch! It is similar to Faster R-CNN:

```
from torchvision.models.detection import maskrcnn_resnet50_fpn
from torchvision.models.detection import MaskRCNN_ResNet50_FPN_Weights

model_mask_rcnn = maskrcnn_resnet50_fpn(weights=MaskRCNN_ResNet50_FPN_Weights.COCO_V1)
```

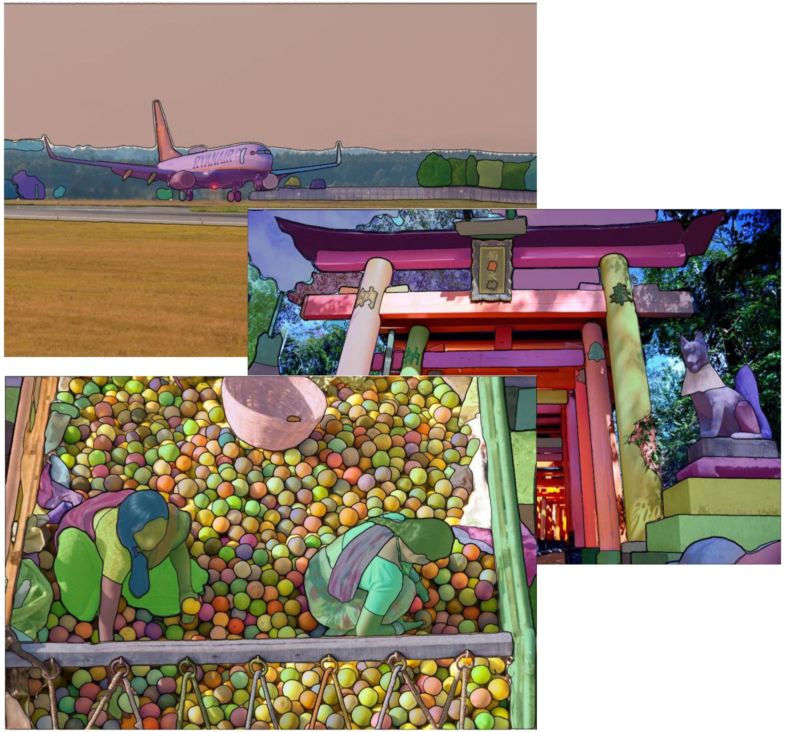
- For a properly processed image called `img`, can do inference* via:

```
results = model_mask_rcnn(img)
```

* For more details details on the necessary preprocessing and the output format of Mask R-CNN, check out this link.

Segment Anything

- In a 2023 paper, Meta (aka Facebook) released one of the most powerful segmentation models available, called **Segment Anything Model (SAM)**.
- It was trained on what is now called SAM dataset, which contains an *staggering* high quality 1.1 **billion segmentation masks** and 100 **million images.**
- The architecture, on the other hand, was a simple encoder-decoder network (*more on it later in the course*).
- You can play with it in their official website.



Panoptic Segmentation

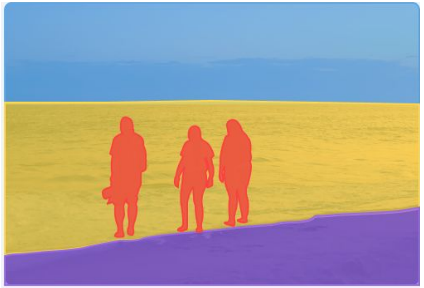
- A final type of segmentation is called **Panoptic Segmentation:**

  Panoptic Segmentation is the task to detect and segment all objects in the picture, including the background, and distinguish different instances.

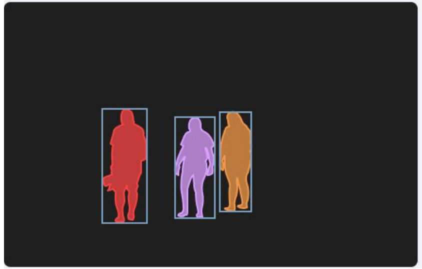
- Note that: *pan* (from greek meaning “all”) + *optic* (meaning visual data).
- In PyTorch, it can be achieved using the many detection tools in the Detectron2 Library* (*more next time*).

Original Image

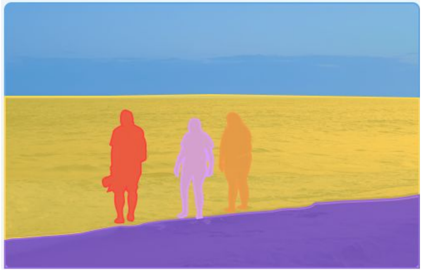
Semantic Segmentation



Instance Segmentation



Panoptic Segmentation



* Here’s a nice video of its results for Panoptic Segmentation

Video: Segmentation and Seft-driving cars