Computer Vision Tasks:

Object Detection
&
Image Segmentation

Object Detection

- Until now we have seen only model for Image Classification
- That model were built to classify an image as whole

- ***Eg: They could not tell us where in the image a "cat" was***

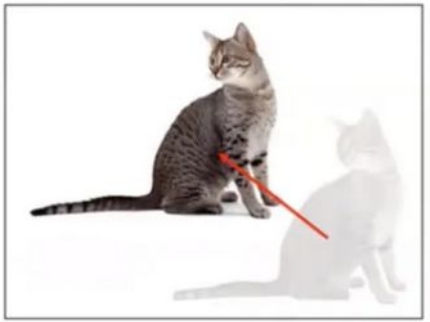
Do you remember the shift invariance in CNNs?

Shift invariance

Input $\mathbf{x}$

Output $f(\mathbf{x}) = 1$

Shifted input $S_v\mathbf{x}$



Output $f(S_v\mathbf{x}) = 1$

- **'Cat detector'** $f: \mathbb{R}^d \to \mathbb{R}$
- **Shift operator** $S_v: \mathbb{R}^d \to \mathbb{R}^d$ shifting the image by vector $\boldsymbol{v}$

Shift invariance: $f(\mathbf{x}) = f(S_v\mathbf{x})$

Object Detection

- Convolutional layers do identify and locate the things they detect.
- But in classification problems, the networks make no use of this information
- **They are trained on an objective where location does not matter**
- A picture of a cat is classified as such wherever the cat appears in the image

For object detection, we will add elements to the convolutional stack to extract and refine the location information and train the network to do so with maximum accuracy

YOLO

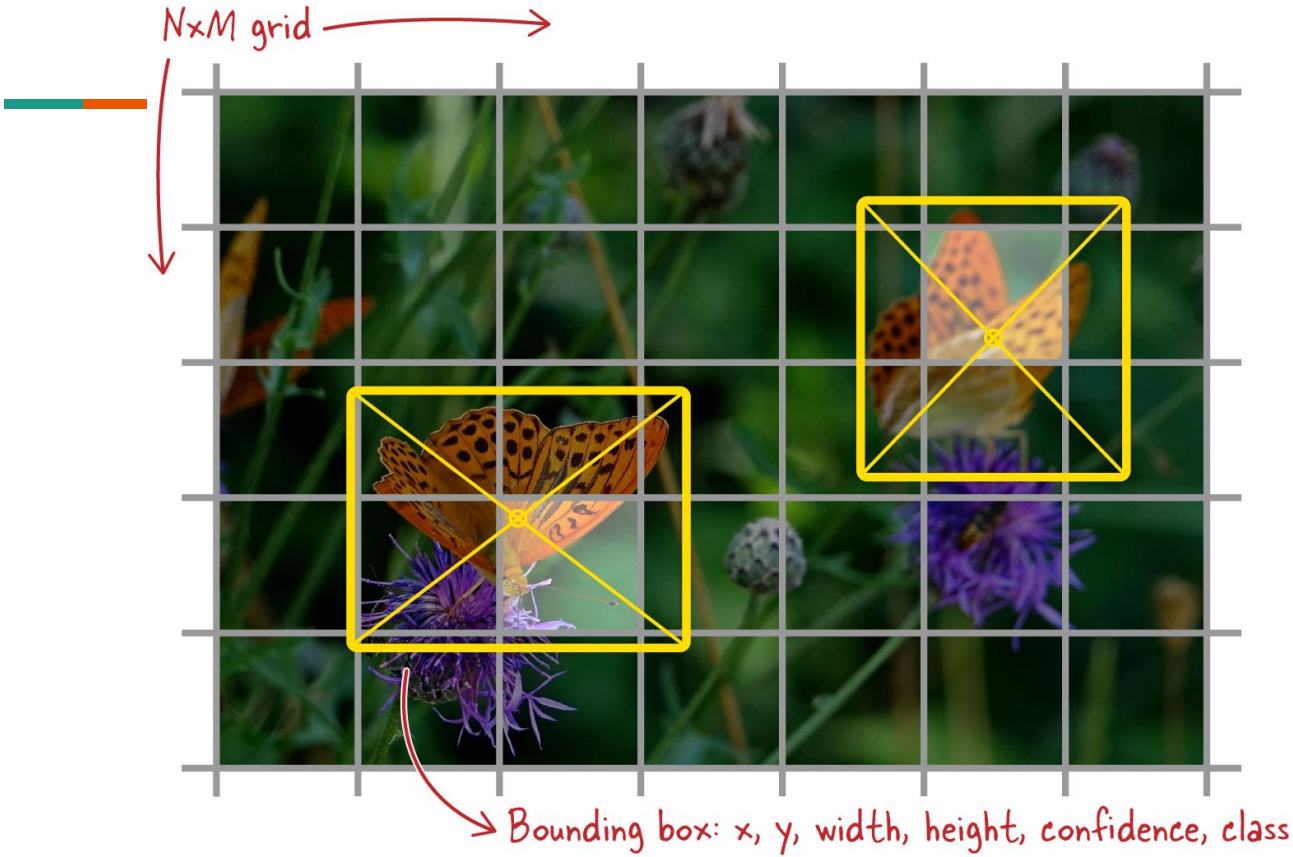
- YOLO (you only look once) is the simplest object detection architecture.
- It is not the most accurate, but it's one of the fastest
- Images are processed through the convolutional stack as in the image classification case
- The classification head is replaced with an object detection and classification head.

There exist different version of YOLO: *v1*, *v2*, …, *v8*, but we will cover only *v1* because is the simplest one to understand.

YOLO grid

- Divides a picture into a grid of *N*x*M* cells
- For each cell, it tries to **predict a bounding box for an object that would be centered in that cell**.
- The predicted bounding box can be larger than the cell from which it originates
- the only **constraint** is that the center of the box is somewhere inside the cell.

What does it mean to predict a bounding box?



[Arthropod Taxonomy Orders Object Detection dataset]

- The dataset contains **7 categories**

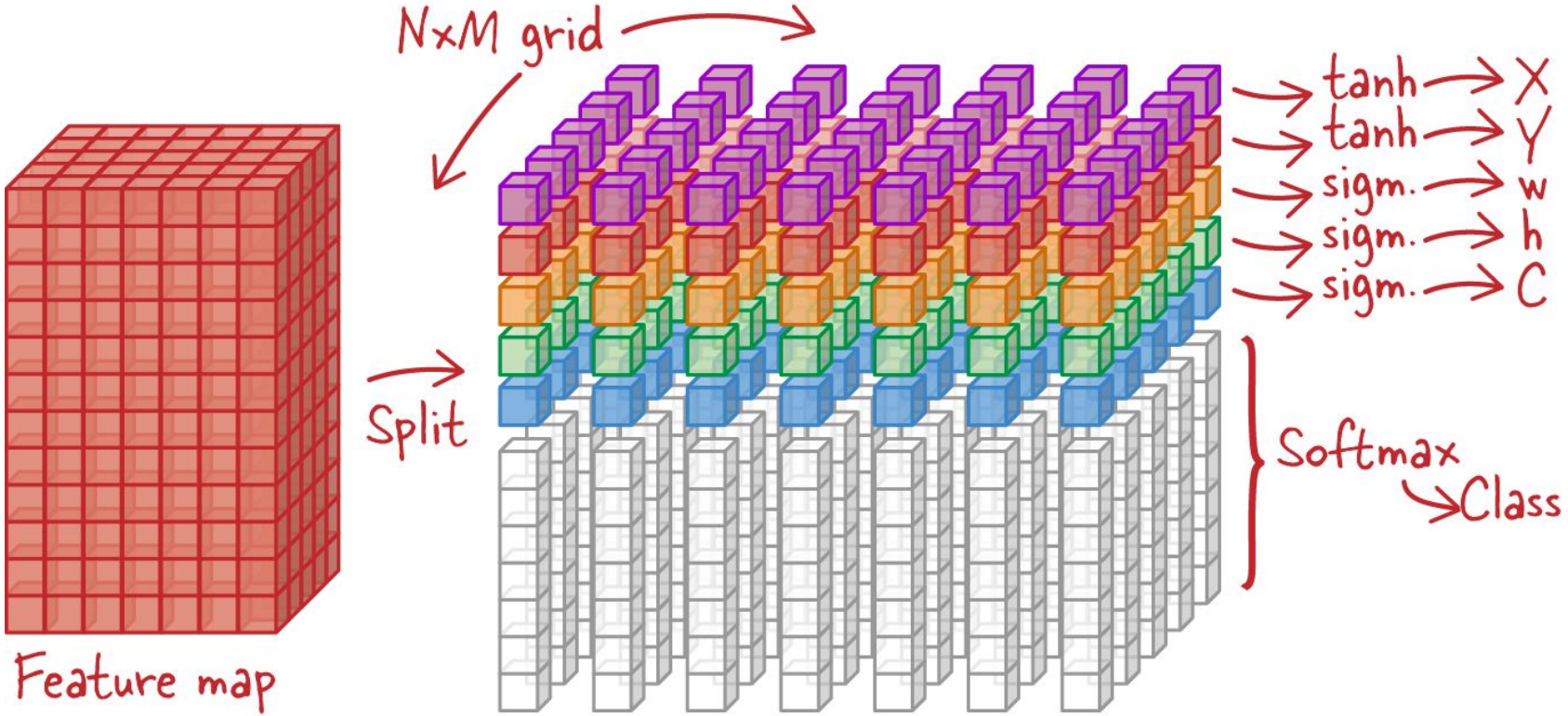
https://www.oreilly.com/library/view/practical-machine-learning/9781098102357/ch04.html

Object Detection Head

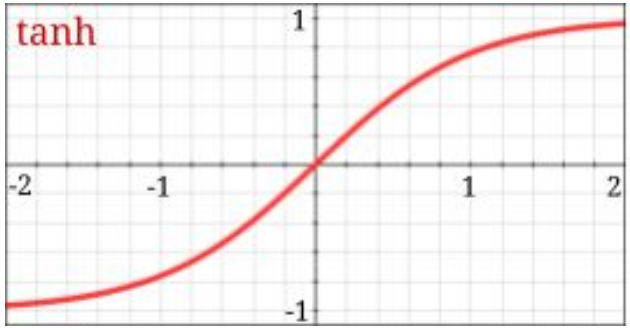
Predicting a bounding box amounts to predicting six numbers:

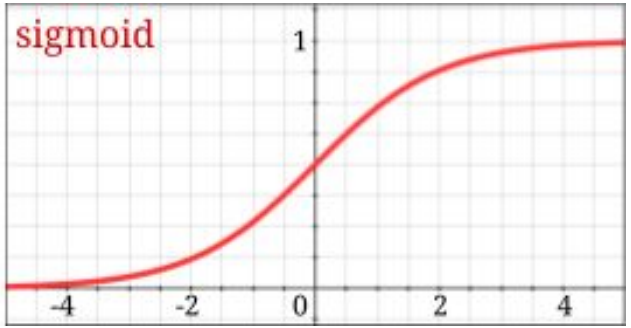
- the four coordinates of the bounding box
  - the ***x*** and ***y*** **coordinates of the center**, and the **width** and **height**
- a **confidence factor** which tells us if an object has been detected or not
- the **class** of the **object** (for example, "***butterfly***")

The YOLO architecture does this directly on the last feature map, as generated by the convolutional backbone it is using.

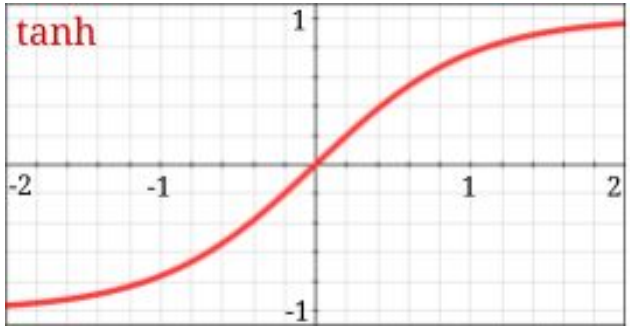


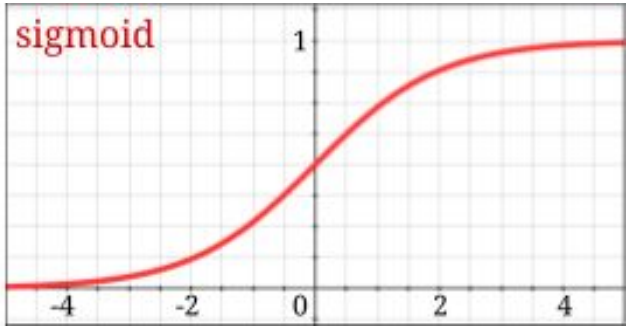
https://www.oreilly.com/library/view/practical-machine-learning/9781098102357/ch04.html





https://www.oreilly.com/library/view/practical-machine-learning/9781098102357/ch04.html

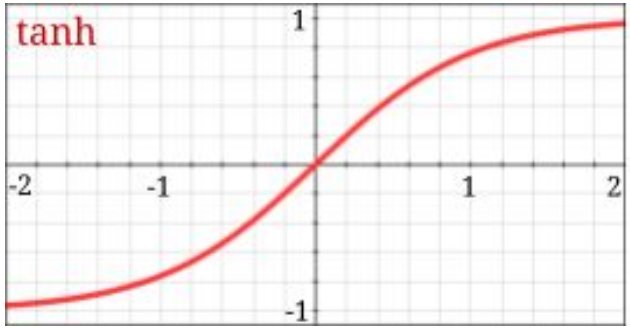


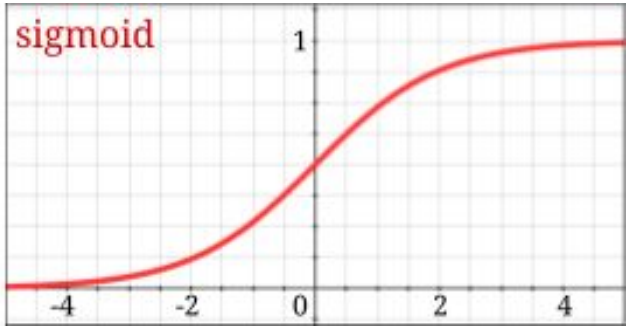


the ***x*** and ***y* coordinate** calculations use a **hyperbolic tangent** (**tanh**) activation so that the coordinates fall in the **[−1, 1] range**.

They will be the coordinates of the **center** of the detection box, relative to the center of the grid cell they belong to.

https://www.oreilly.com/library/view/practical-machine-learning/9781098102357/ch04.html



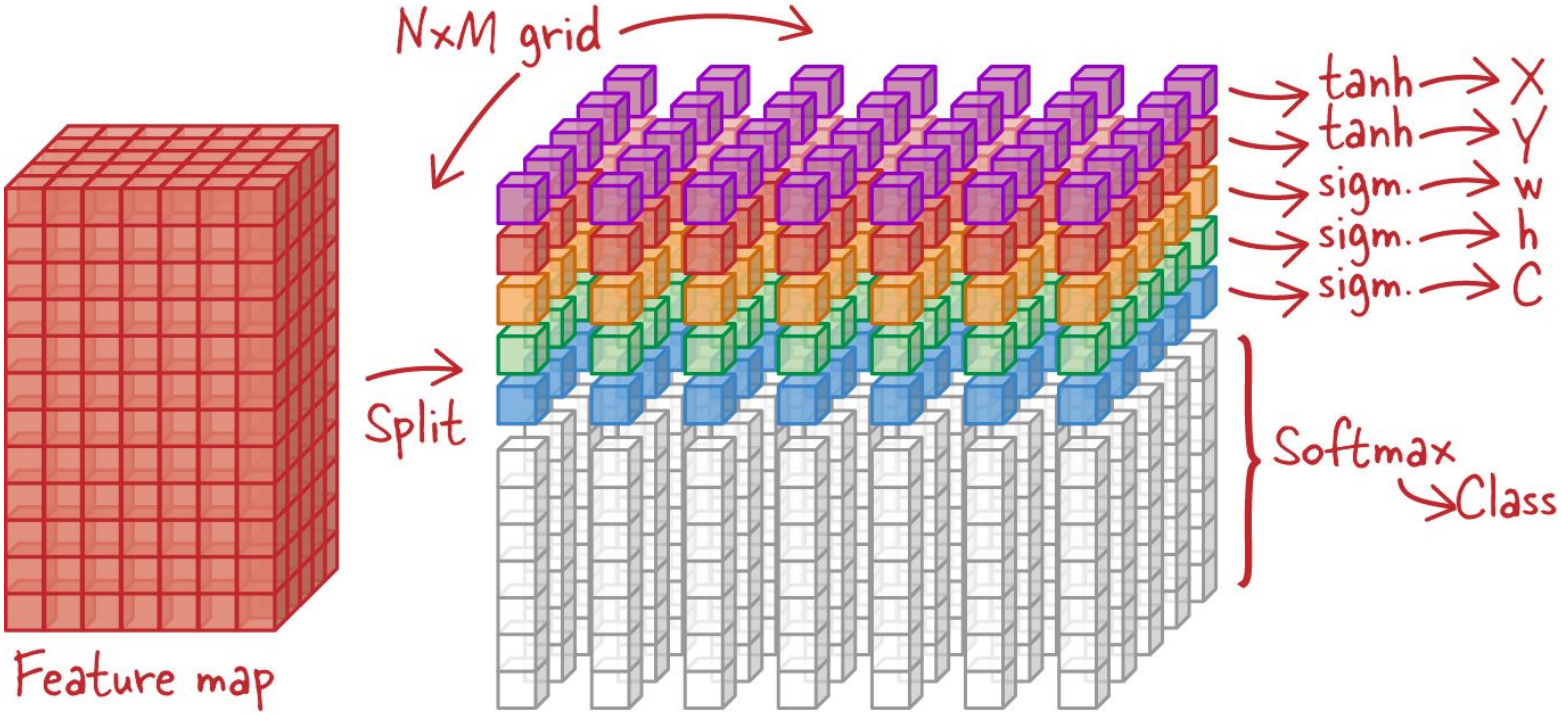


Width and **height** (***w***, ***h***) calculations use a **sigmoid** activation so as to fall in the **[0, 1]** range.

They will represent the **size** of the **detection box** relative to the entire image.

This allows detection boxes to be bigger than the grid cell they originate in.

https://www.oreilly.com/library/view/practical-machine-learning/9781098102357/ch04.html



- The **confidence factor**, ***C***, is also in the **[0, 1] range.**
- A **softmax** activation is used to predict the class of the detected object

https://www.oreilly.com/library/view/practical-machine-learning/9781098102357/ch04.html

Object Detection Head

- How to obtain a feature map of exactly the right dimensions?

Object Detection Head

- In the previous example, it must contain exactly ***7 * 5 * (5 + 7) values.***

Object Detection Head

- In the previous example, it must contain exactly 7 * 5 * (5 + 7) values.
- The 7 * 5 is because we chose a 7x5 YOLO grid.

Object Detection Head

- In the previous example, it must contain exactly 7 * 5 * (5 + 7) values.
- The 7 * 5 is because we chose a 7x5 YOLO grid.
- Then, for each grid cell, five values are needed to predict a box **(*x*, *y*, *w*, *h*, *C*)**

Object Detection Head

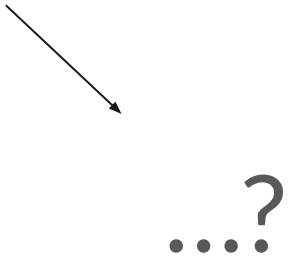
- In the previous example, it must contain exactly 7 * 5 * (5 + 7) values.
- The 7 * 5 is because we chose a 7x5 YOLO grid.
- Then, for each grid cell, five values are needed to predict a box (*x*, *y*, *w*, *h*, *C*)
- **7 additional values are needed because, in this example, we want to classify arthropods into seven categories** (*Coleoptera*, *Aranea*, *Hemiptera*, *Diptera*, *Lepidoptera*, *Hymenoptera*, *Odonata*).

Object Detection Head

- If you control the convolutional stack, you could try to tune it to get exactly 7 * 5 * 12 (420) outputs at the end.
- However, there is an easier way: **flatten** whatever feature map the convolutional backbone is returning and feed it through a **fully connected layer with exactly that number of outputs.**
- **You can then reshape the 420 values into a 7x5x12 grid, and apply the appropriate activations.**
- *The authors of the YOLO paper argue that the fully connected layer actually adds to the accuracy of the system.*

Loss Function

- How can we train an Object Detection model?
- In object detection, as in any supervised learning setting, the correct answers are provided in the training data



Loss Function

- How can we train an Object Detection model?
- In object detection, as in any supervised learning setting, the correct answers are provided in the training data

X and Y

Loss Function

- How can we train an Object Detection model?
- In object detection, as in any supervised learning setting, the correct answers are provided in the training data

ground truth boxes and their ***classes***

Loss Function

- In object detection, as in any supervised learning setting, the correct answers are provided in the training data: **ground truth boxes and their classes**.
- During training the network **predicts detection boxes**, and it has to **take into account errors** in the boxes' **locations** and **dimensions** and…

Loss Function

- In object detection, as in any supervised learning setting, the correct answers are provided in the training data: **ground truth boxes and their classes**.
- During training the network predicts detection boxes, and it has to take into account errors in the boxes' locations and dimensions and…
- **misclassification errors**
- also penalize detections of objects where there aren't any.

Loss Function

- The first step, though, is to correctly pair ground truth boxes with predicted boxes so that they can be **compared**.
- In the YOLO architecture, if each grid cell predicts a single box, this is straightforward. A ground truth box and a predicted box are **paired** if they are centered in the same grid cell

- But pairing these predictions with ground truth boxes requires more care.
- This is done by computing the *intersection over union* between all ground truth boxes and all predicted boxes within a grid cell, and selecting the pairings where the IOU is the highest.

Loss Function

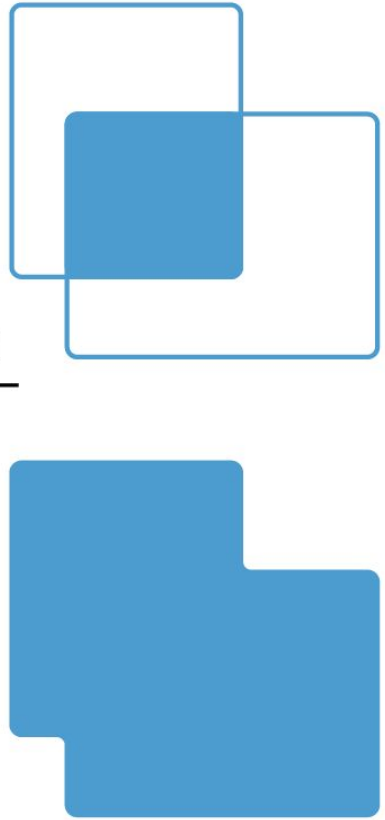
- How can we compare these boxes ?

Loss Function

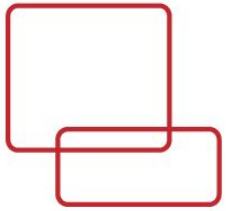
- This is done by computing the ***intersection over union*** **(IoU)** between all ground truth boxes and all predicted boxes within a grid cell, and selecting the pairings where the IOU is the highest.

Intersection Over Union (IoU)

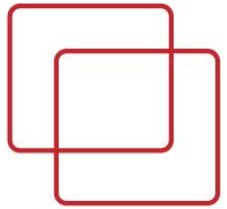
$$IOU = \frac{\text{Area of intersection}}{\text{Area of union}}$$



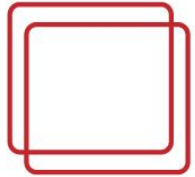
Examples



IOU = 0.1



IOU = 0.3



IOU = 0.6

Intersection Over Union (IoU)

- Ground truth boxes are assigned to grid cells by their centers and to the prediction boxes within these grid cells by IOU.

Loss Function

Object presence loss

- Each grid cell that has a ground truth box computes:

$$L_{obj}=(1-C)^2$$

Where C is the Confidence

Object absence loss

- Each grid cell that does not have a ground truth box computes:

$$L_{noobj}=(0-C)^2=C^2$$

Loss Function

Object classification loss

- Each grid cell that has a ground truth box computes:

$$L_{class} = \text{cross-entropy}(p, \hat{p})$$

where $\hat{p}$ is the vector of predicted class probabilities and p is the one-hot-encoded target class.

Bounding box loss

- Each predicted box/ground truth box pairing contributes (*predicted coordinate marked with a hat, the other coordinate is the ground truth*):

$$L_{box} = (x - \hat{x})^2 + (y - \hat{y})^2 + (\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2$$

Notice here that the difference in box sizes is computed on the square roots of the dimensions. This is to mitigate the effect of large boxes, which tend to overwhelm the loss.

Loss Function

- All the loss contributions from the grid cells are added together, with **weighting factors**.
- A common problem in object detection losses is that small losses from numerous cells with no object in them end up overpowering the loss from a lone cell that predicts a useful box.
- Weighting different parts of the loss can alleviate this problem. The authors of the paper used the following empirical weights:

$$\lambda_{obj}=1 \qquad \lambda_{noobj}=0.5 \qquad \lambda_{class}=1 \qquad \lambda_{box}=5$$

YOLO Limitations

1. YOLO predicts a single class per grid cell and **will not work well if multiple objects of different kinds are present in the same cell**.
2. **The grid**: a fixed grid resolution imposes strong spatial constraints on what the model can do. YOLO models will typically not do well on collections of small objects, like a flock of birds, without careful tuning of the grid to the dataset.
3. **Low precision**. The main reason for that is that it works on the last feature map from the convolutional stack, which is typically the one with the lowest spatial resolution and contains only coarse location signals.

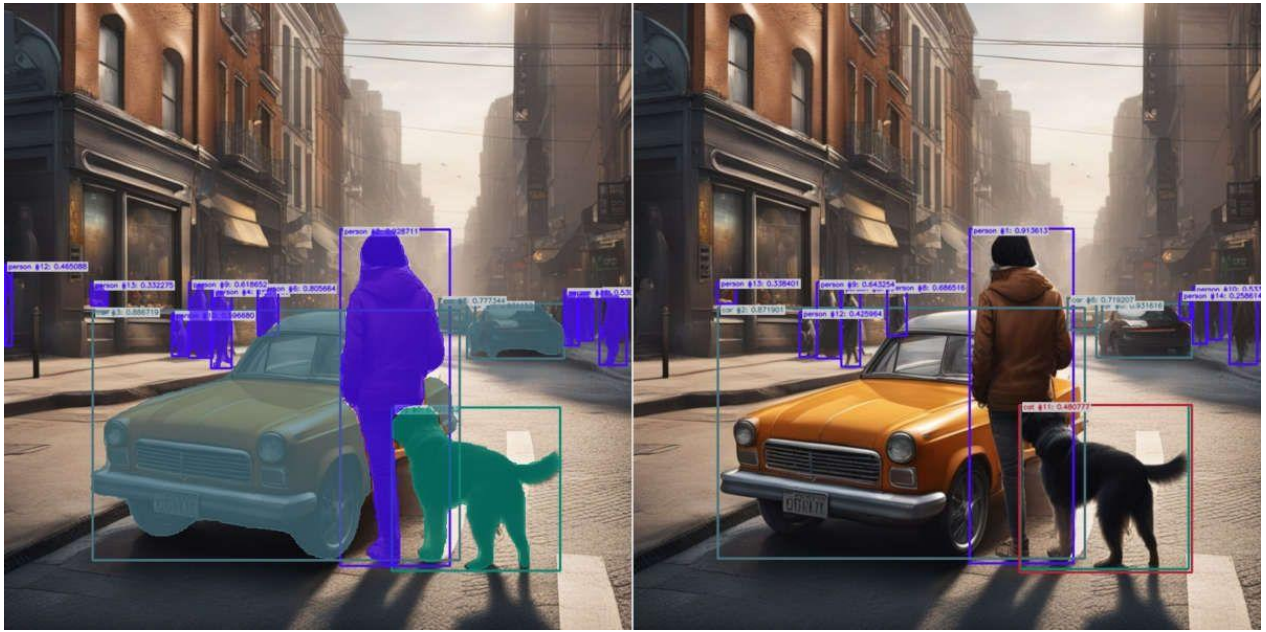
YOLO

- Despite these limitations, the YOLO architecture is **very simple to implement**, especially with a single detection box per grid cell, which makes it a good choice when you want to experiment with your own code.
- If a higher accuracy is needed, you can step up to the next level: **RetinaNet**.

Image Segmentation

- Object detection finds bounding boxes around objects and classifies them.
- ***Instance segmentation*** involves detecting objects and finding all the pixels that belong to each object.
- ***Semantic segmentation***, on the other hand, does not detect specific instances of objects but classifies every pixel of the image into a category like “road,” “sky,” or “people.”

Image Segmentation

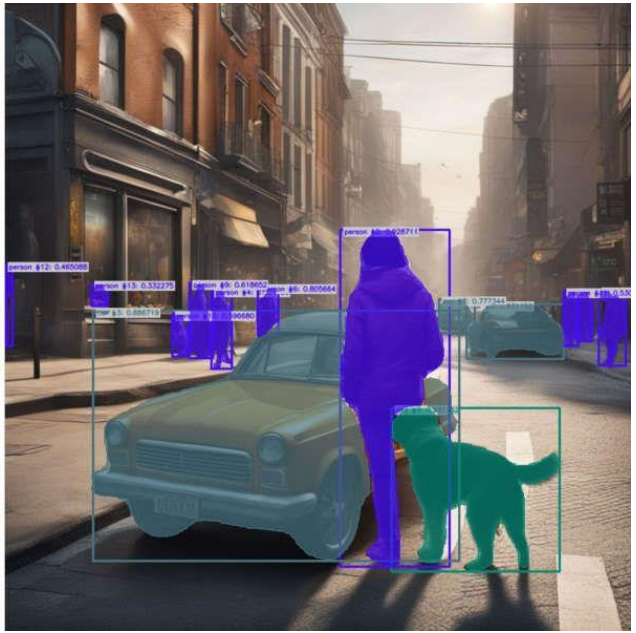


Instance segmentation

Object Detection

Image Segmentation

Semantic Segmentation



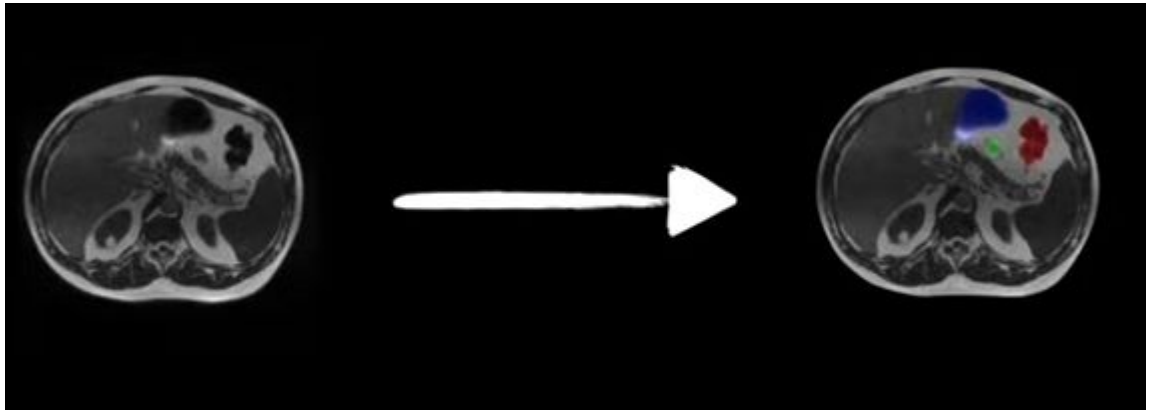
Instance segmentation

U-NET and Semantic Segmentation

- U-net was originally invented and first used for biomedical image segmentation.
- Autoencoder-like architecture:
  - **encoder** network followed by a **decoder** network.

U-NET and Semantic Segmentation

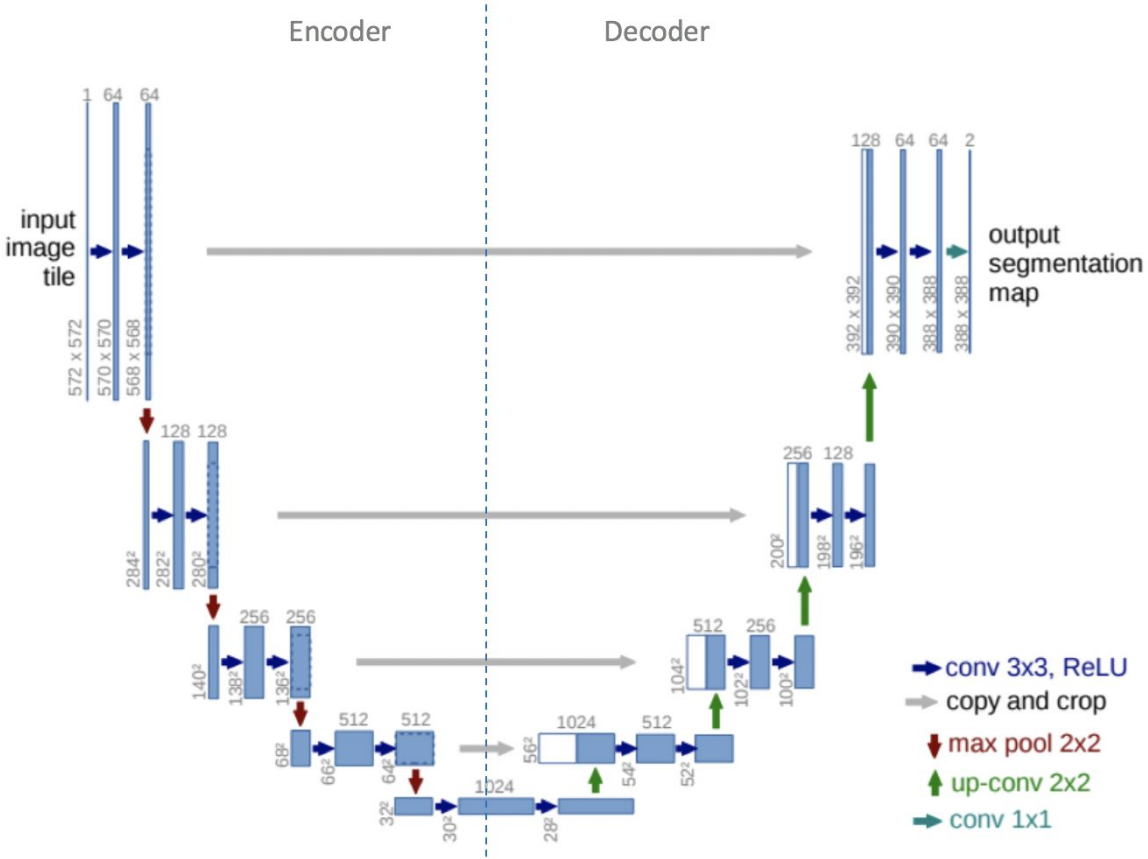
- U-net was originally invented and first used for biomedical image segmentation.
- Autoencoder-like architecture:
  - **encoder** network followed by a **decoder** network.



U-NET and Semantic Segmentation

- U-net was originally invented and first used for biomedical image segmentation.
- Autoencoder-like architecture:
  - **encoder** network followed by a **decoder** network.
- It can be used for different task in computer vision:
  - Image Segmentation
  - Super-resolution task
  - Diffusion models

U-NET


Encoder
Decoder
input image tile
572 x 572
570 x 570
568 x 568
output segmentation map
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

U-NET and Semantic Segmentation

- All of these task, take in input an image and produce a new image

U-NET and Semantic Segmentation

- All of these task, take in input an image and produce a new image
  - The networks always learn the mapping between input and output images
- For example in segmentation:
  - *We are trying to learn a mapping between the pixels of an image and the pixels of a segmentation mask*

Instance Segmentation

- Training them remains a **supervised training task** and the training data will have to contain **ground truth** segmentation masks for all objects.
- Masks are more **time-consuming** to generate by hand than bounding boxes
- Instance segmentation **datasets** are **harder** to **find** than simple object detection datasets.

Instance Segmentation

- Training them remains a **supervised training task** and the training data will have to contain **ground truth** segmentation masks for all objects.
- Masks are more **time-consuming** to generate by hand than bounding boxes
- Instance segmentation **datasets** are **harder** to **find** than simple object detection datasets.
- So, the loss function can be:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

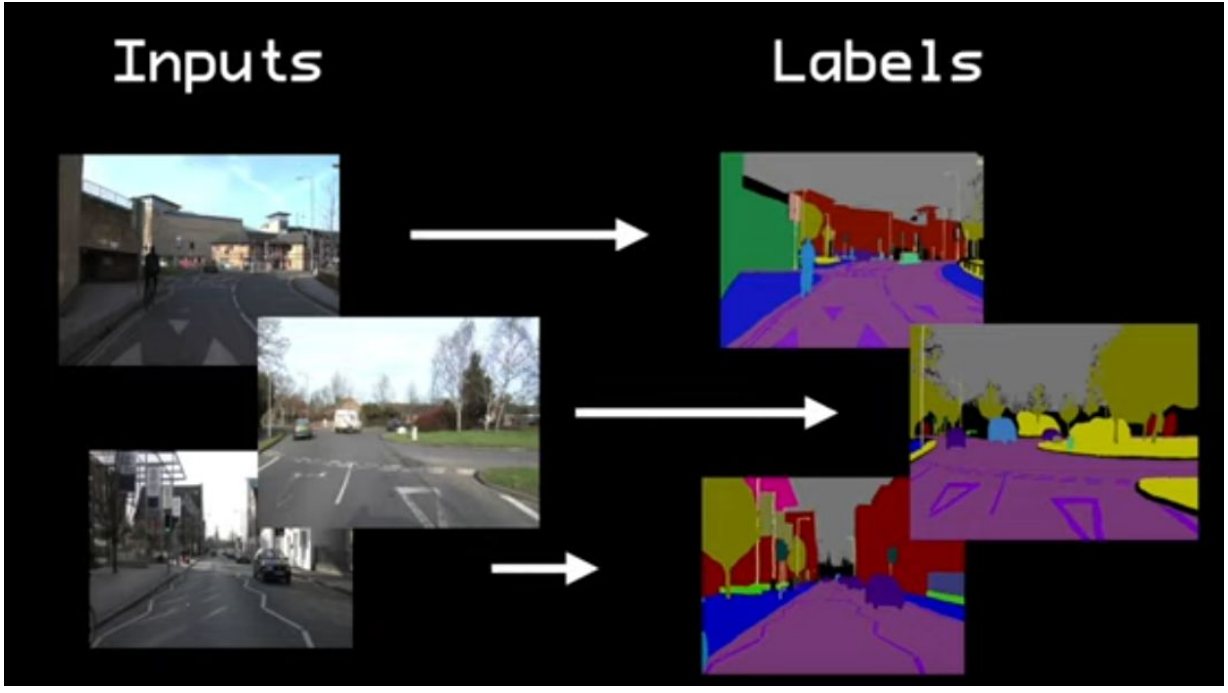
Instance Segmentation

- So, the loss function can be:

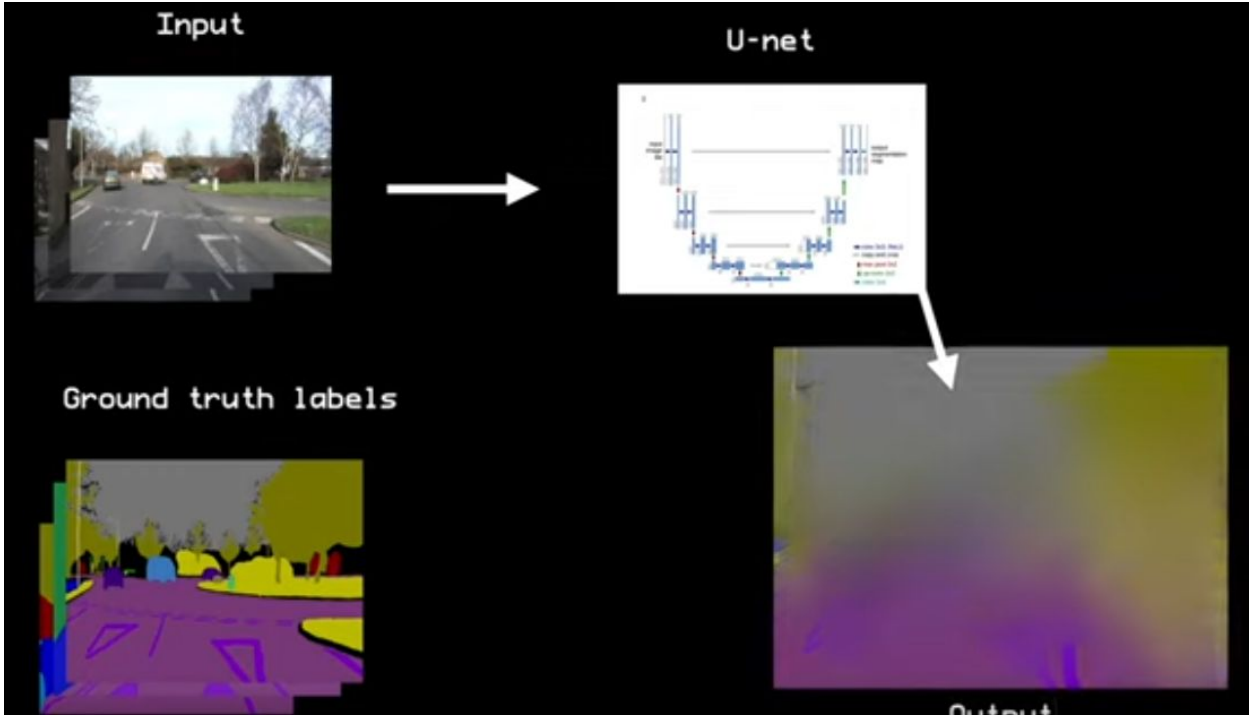
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

U-Net typically uses a pixel-wise binary cross-entropy loss function, which measures the difference between the predicted segmentation mask and the ground truth mask for each pixel.

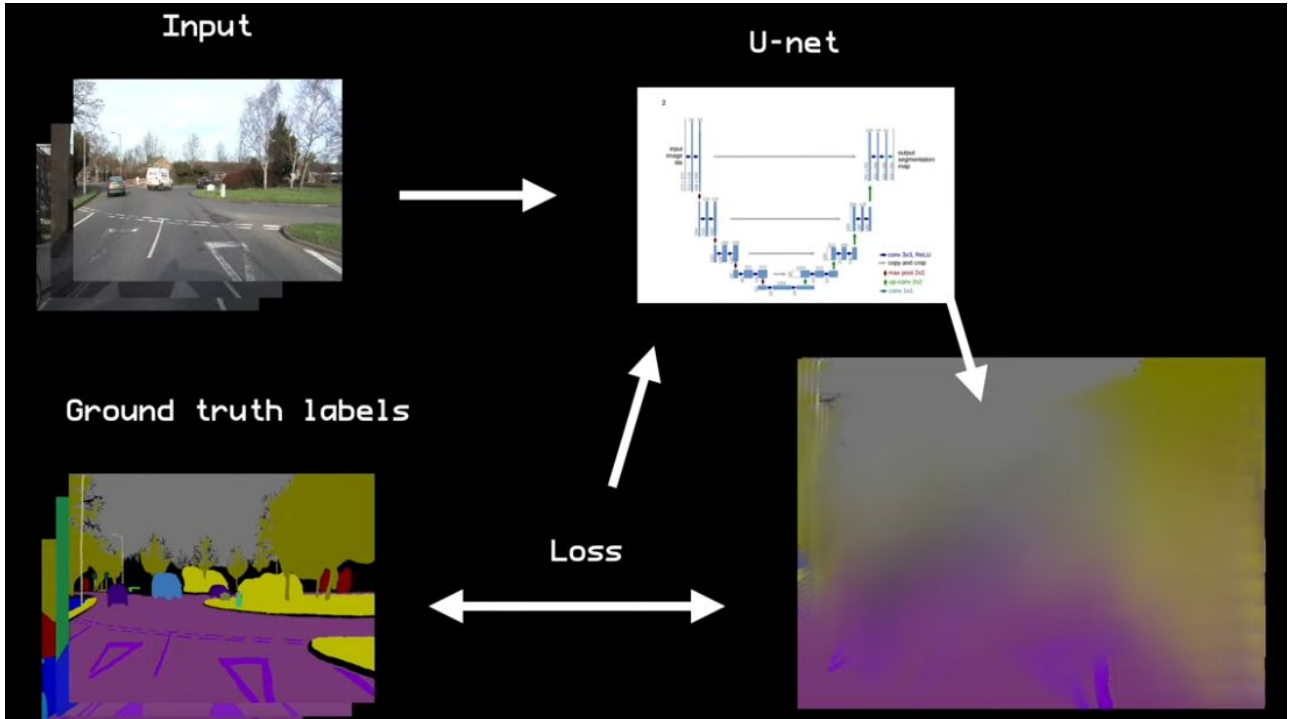
Instance Segmentation



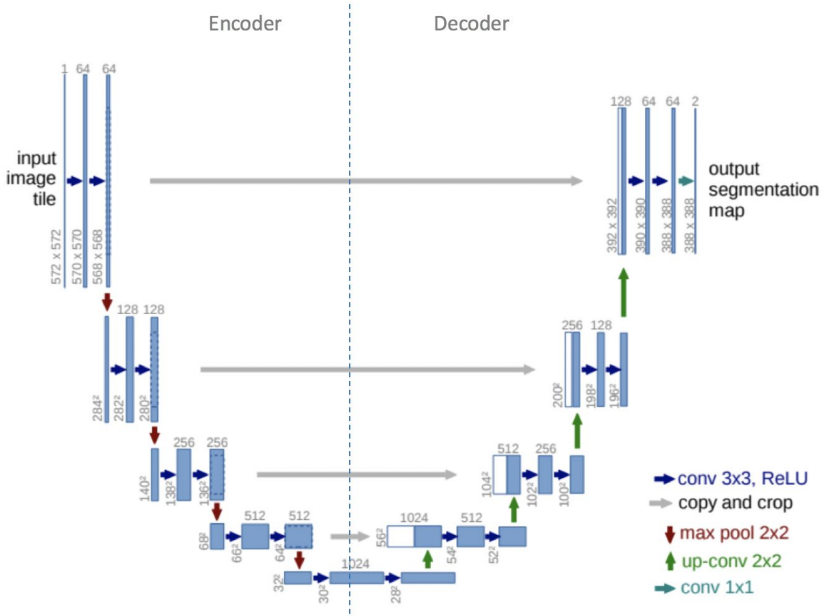
Instance Segmentation



Instance Segmentation



U-NET



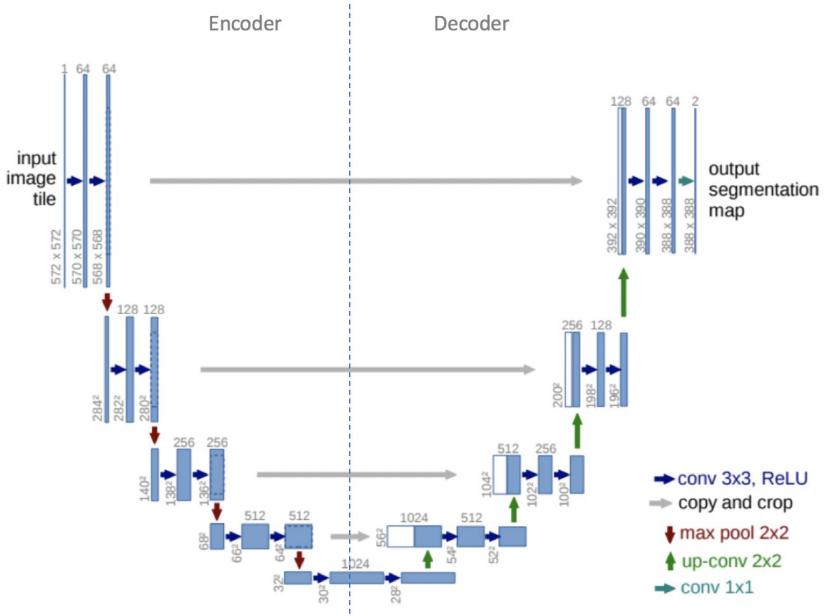
Encoder

- Is responsible for extracting features from the input image

Decoder

- Is responsible for up sampling intermediate features and producing the final output
- The output of the last decoder passes through a 1x1 convolution with **sigmoid activation**.
  - The sigmoid activation function gives the segmentation mask representing the pixel-wise classification.

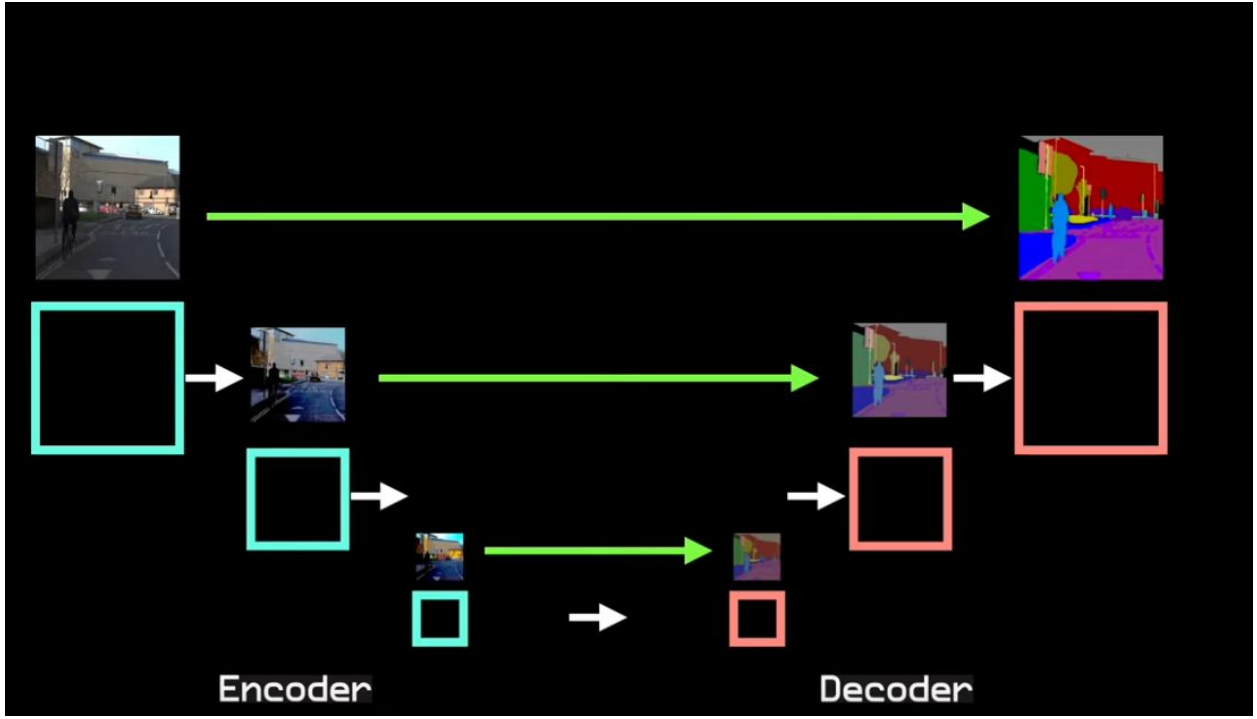
U-NET



The encoder and the Decoder are symmetrical

- **Encoder**
  - 3X3 conv layers
  - Followed by a RELU
  - followed by a max pooling (2x2)
- **Decoder**
  - 3X3 conv layers
  - Followed by a RELU
  - up sampling layer followed a 2x2 conv layer
- Connected by **paths and bottleneck**

U-NET… a closer look



U-NET

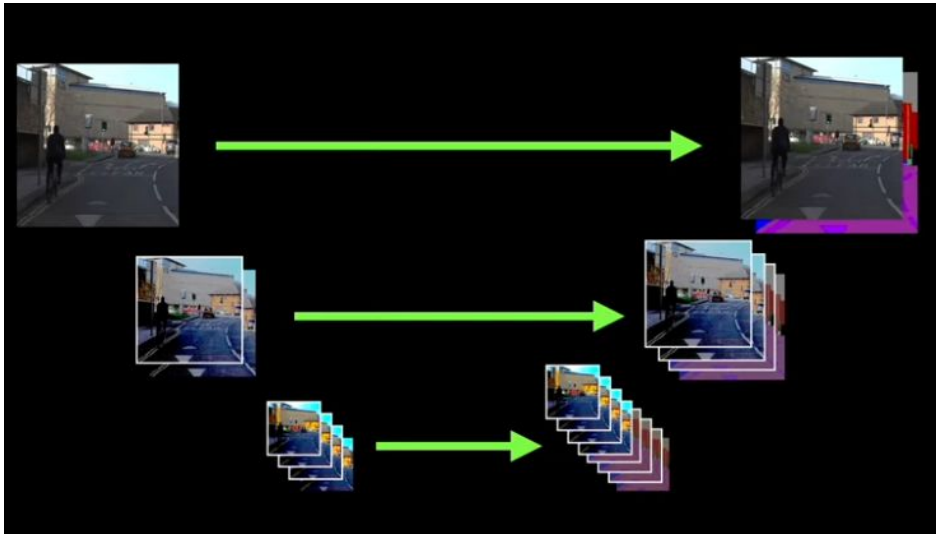
- Encoder
  - The channels are doubled after each down sampling operations
- Decoder
  - Reverse the encoder
  - Halve channels after upsampling
- There are 2 two types of connection between encoder and decoder layers:
  - Connecting paths
  - The bottleneck

U-NET: Connecting paths or Skip Connections

- They take a copy of the features from the symmetrical part of the encoder and concatenate them on the opposing stage in the decoder
- Because deep neural networks can “forget” certain features as it pass information through successive layers, skip connections can reintroduce them to make learning stronger.
- Skip connection was introduced in Residual Network (ResNet) and showed classification improvements as well as smoother learning gradients.

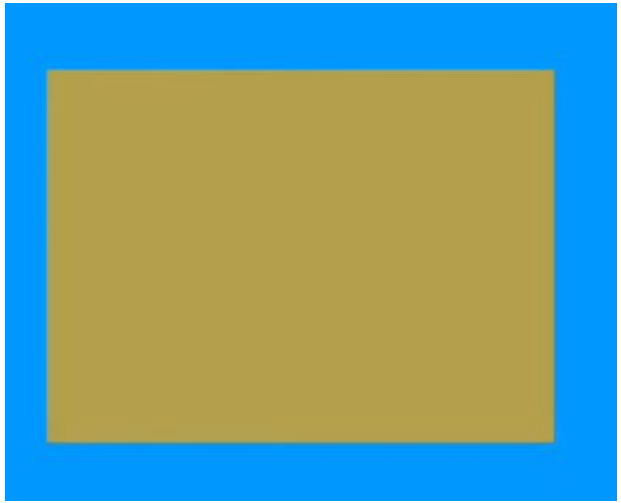
U-NET: Connecting paths

- They take a copy of the features from the symmetrical part of the encoder and concatenate them on the opposing stage in the decoder



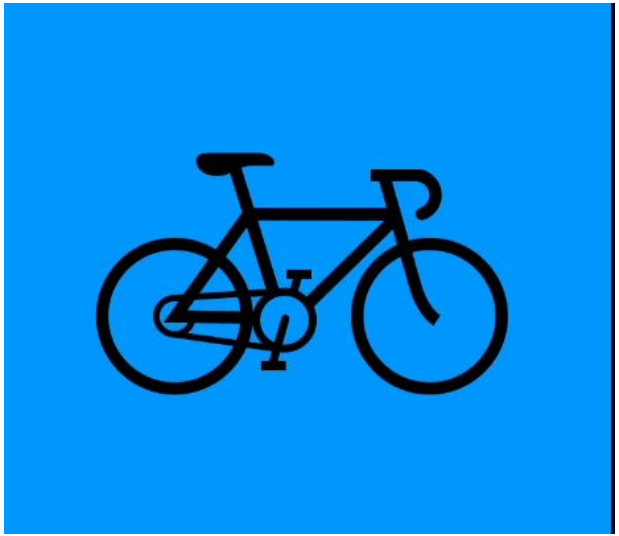
U-NET: Connecting paths

- The decoded features include more semantic information
  - *“this area is a bike”*



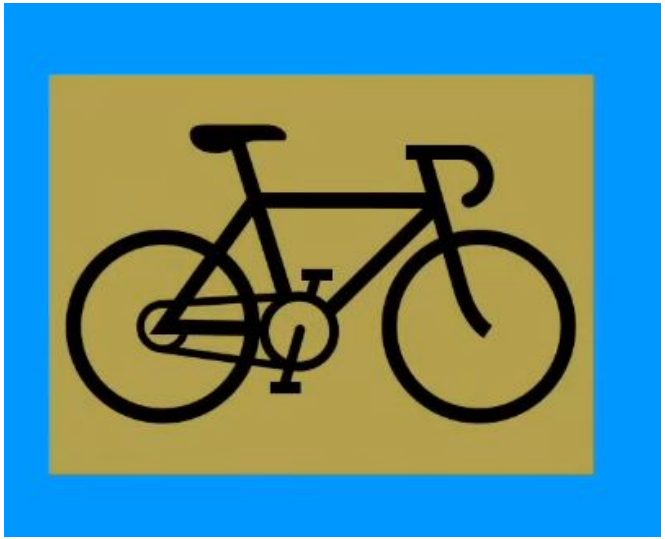
U-NET: Connecting paths

- The encoded features contains more spatial information
  - *“ these are the pixels where the object is”*



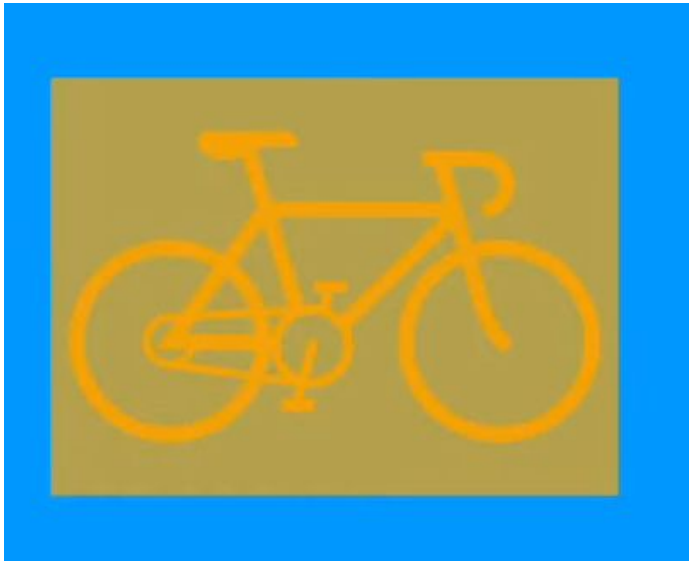
U-NET: Connecting paths

- When you combine these features together you can get pixel segmentation



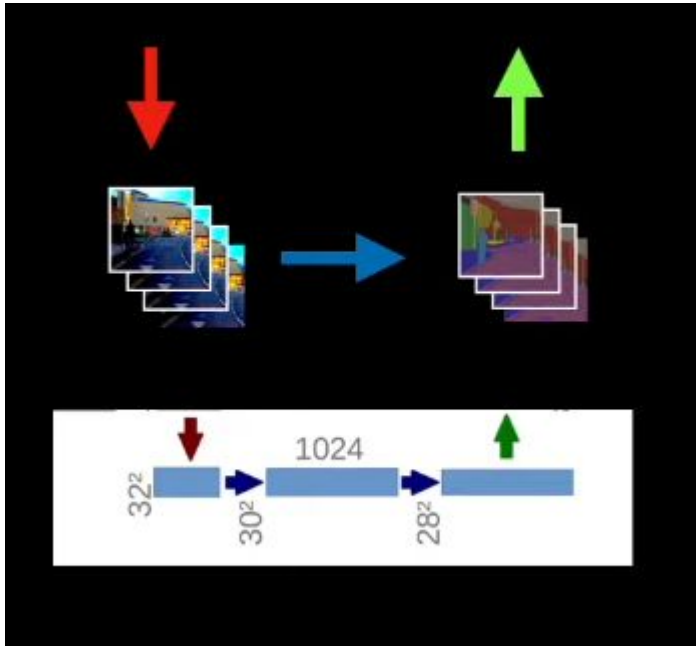
U-NET: Connecting paths

- When you combine these features together you can get pixel segmentation

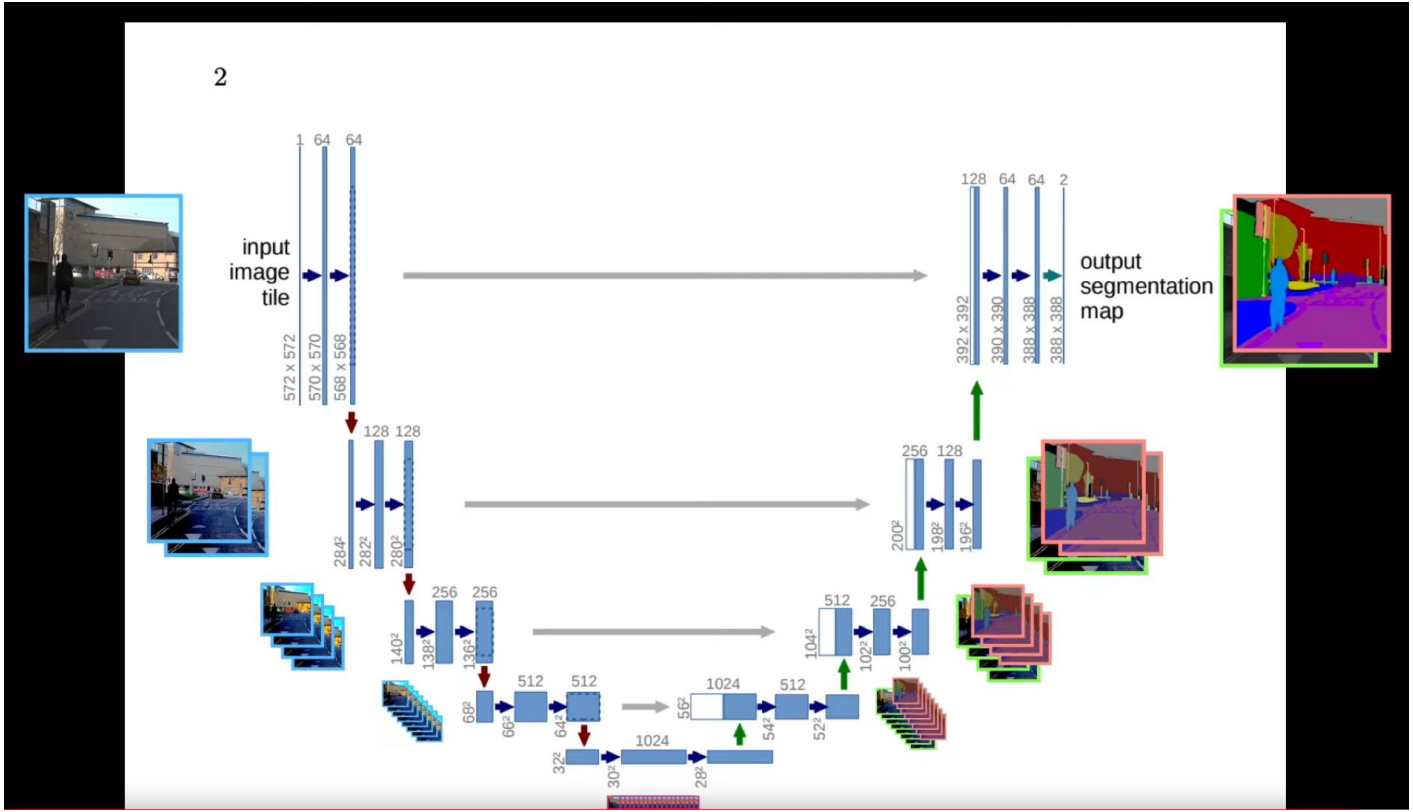


U-NET: Bottleneck

- Downsample with 2x2 max pooling
- Repeated 3x3 convolutional layers+ReLU
- Double channels with conv after max pool
- Upsample followed by 2x2 conv layer



U-NET: Bottleneck



U-NET and Image Segmentation

Key Takeaways

- Image segmentation is essential in computer vision tasks, allowing the division of images into meaningful regions or objects.
- It is a fully convolutional neural network (FCN) combining an encoding path to capture high-level features and a decoding method to generate the segmentation mask.
- Skip connections in U-NET preserve spatial information, enhance feature propagation, and improve segmentation accuracy.
- Found successful applications in medical imaging, satellite imagery analysis, and industrial quality control, achieving notable benchmarks and recognition in competitions.