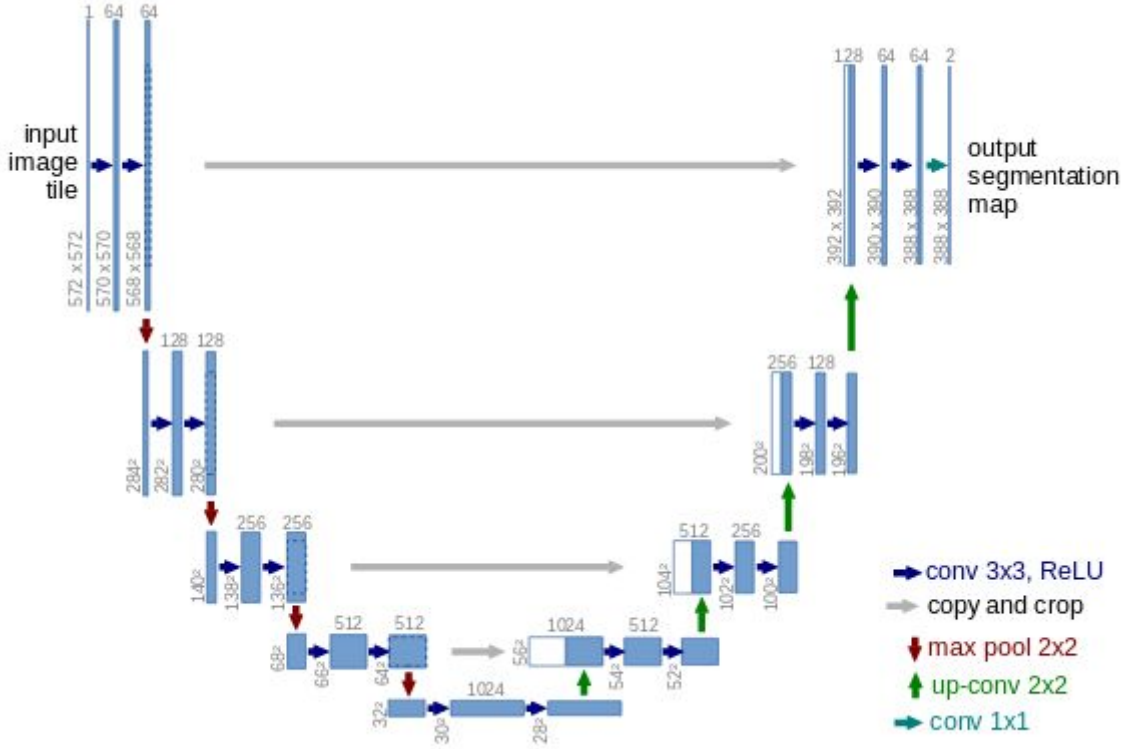
The U-net

- A classifier in Deep learning is an algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

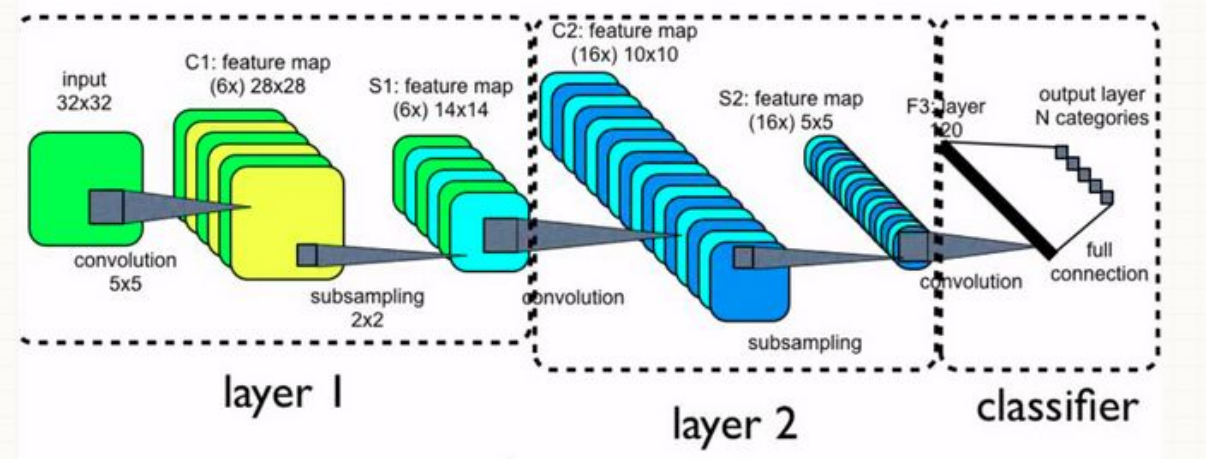
- **the U-net architecture:**



→ The equivalent Pytorch implementation into the nn.unet_origin.py module.

- Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.
- Max pooling is a **sample-based discretization process.** The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. It's about applying a

Maximal filter to non-overlapping subregions of the initial representation..



Tutorial for Pytorch

- **Pytorch definition :**

→ It's a Python based scientific computing package targeted at two sets of audiences:

- A replacement for NumPy to use the power of GPUs
- a deep learning research platform that provides maximum flexibility and speed

- **Tensors:**

Tensors are similar to NumPy's ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

- Converting a Torch Tensor to a NumPy Array:

```
a = torch.ones(5)
print(a)

Out:
tensor([ 1.,  1.,  1.,  1.,  1.])

b = a.numpy()
print(b)

Out:
[1. 1. 1. 1. 1.]
```

- Converting NumPy Array to Torch Tensor:

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a) % addition of  1 the numpy array a
print(a)
print(b)
```

- CUDA Tensors

→ Tensors can be moved onto any device using the .to method.

```
# let us run this cell only if CUDA is available
# We will use ``torch.device`` objects to move tensors in and out of GPU
if torch.cuda.is_available():
    device = torch.device("cuda")          # a CUDA device object
    y = torch.ones_like(x, device=device)  # directly create a tensor on GPU
    x = x.to(device)                       # or just use strings ``.to("cuda")``
    z = x + y
    print(z)
    print(z.to("cpu", torch.double))       # ``.to`` can also change dtype together!
```

Out:

```
tensor([ 1.9422], device='cuda:0')
tensor([ 1.9422], dtype=torch.float64)
```

● Parameters:

class torch.nn.Parameter

→ A kind of Tensor that is to be considered a module parameter.

● Autograd: automatic differentiation:

→ Central to all neural networks in PyTorch is the autograd package. The autograd package provides automatic differentiation for all operations on Tensors.

Tensor:

- torch.Tensor is the central class of the package. If you set its attribute .requires_grad as True, it starts to track all operations on it. When you finish your computation you can call .backward() and have all the gradients

computed automatically. The gradient for this tensor will be accumulated into .grad attribute.

- To stop a tensor from tracking history, you can call .detach().
- Function: There's one more class which is very important for autograd implementation - a Function.
- Tensor and Function are interconnected and build up an acyclic graph, that encodes a complete history of computation. Each variable has a .grad_fn attribute that references a Function that has created the Tensor (except for Tensors created by the user - their grad_fn is None).
- If you want to compute the derivatives, you can call .backward() you need to specify a gradient argument to the backward() function.

Gradients:

Example:

```
gradients = torch.tensor([0.1, 1.0, 0.0001], dtype=torch.float)
y.backward(gradients)

print(x.grad)
```

Out:
tensor([51.2000, 512.0000,   0.0512])

● Neural Networks

→ Neural networks can be constructed using the torch.nn package.
class torch.nn.Module

class torch.nn.Module
→ Base class for all neural network models. All the neural network should subclass this class.