Semantic Segmentation of RGB-D Images

- The DeResUNet
- BatchNorm2D & PixelShuffle
- Albumentations
- PyTorch Lightning

Semantic Segmentation

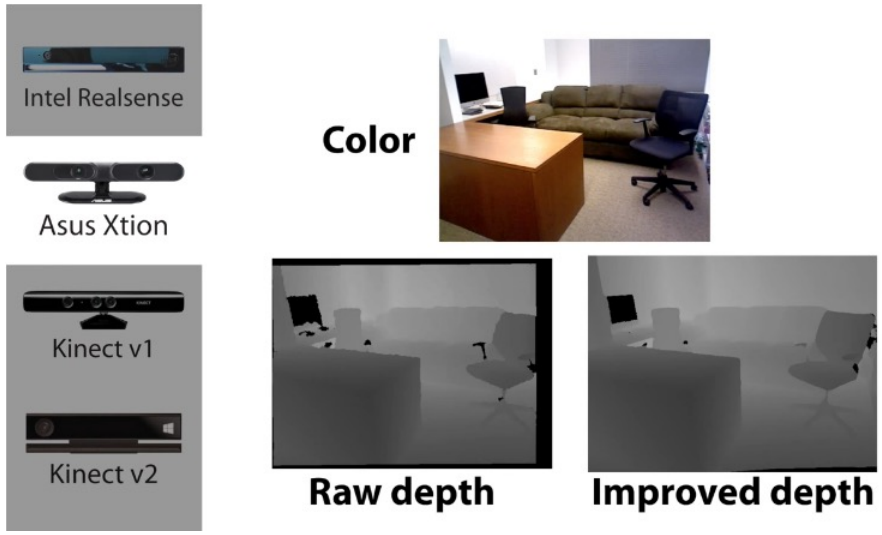
- "*Semantic*" refers to the meaning or context associated to objects present in an image
- "*Segmentation*" is the process of partitioning an image into segments, each grouping pixels sharing common features
- In our Case:

$$f : I_x(u, v) \rightarrow c \in C^N$$

$$||\phi(I_x(u1, v1)) - \phi(I_y(u2, v2))|| < \epsilon \Rightarrow f(I_x(u1, v1)) = f(I_y(u2, v2)) = c_i$$

where $f \widehat{=}$ segmentation function and $\phi(I_x) \widehat{=}$ repr. of I_x in feature space

The SunRGBD Dataset



10.335 RGB-D

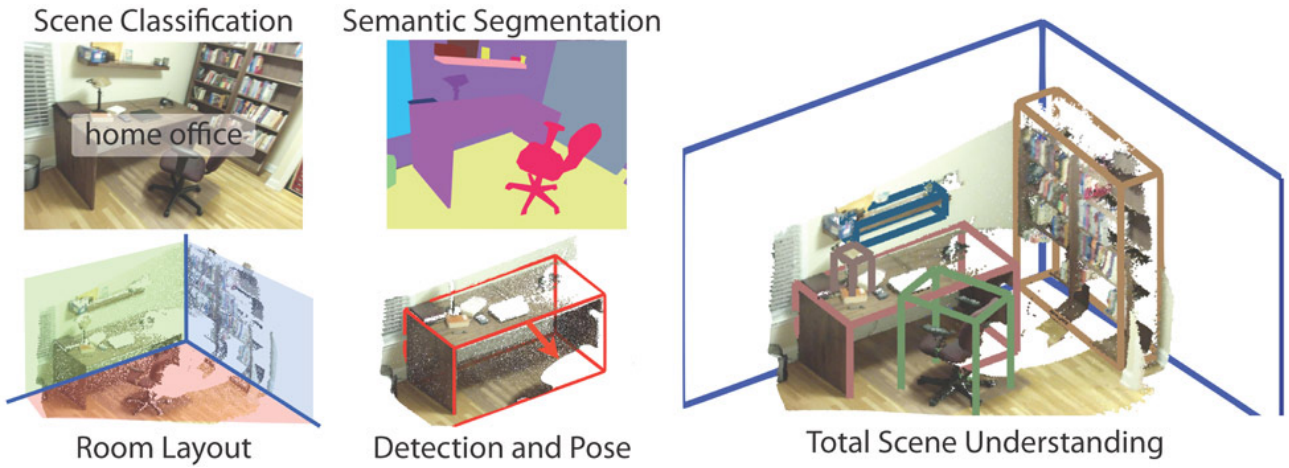
6.4 GB

Toolbox provided

Annotations for 2D/3D

146,617 2D polygons

58,657 3D bounding boxes



https://rgbd.cs.princeton.edu

Augmentation

Albumentations

vs torchvision.transforms

- ⊕ Performance (up to 23x faster)[A1]
- ⊕ More variety – Extreme Augmentations
- ⊕ Usage with any DL framework
- ⊕ Easy transforms for Segmentation and Detection Tasks

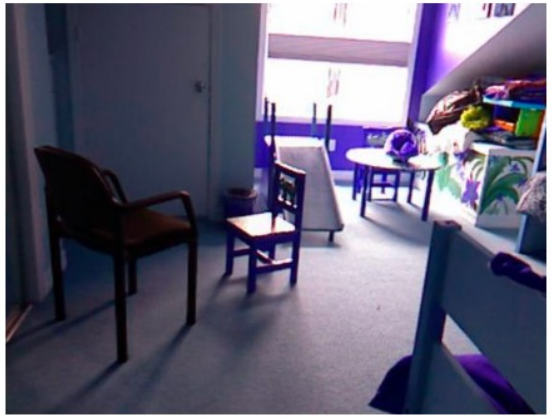
Optimized for NumPy and OpenCV

```
            A.RGBShift(
                r_shift_limit=20, g_shift_limit=20, b_shift_limit=20, p=0.95
            ),
            A.OneOf(
                [
                    A.Blur(blur_limit=3, p=0.5),
                    A.ColorJitter(p=0.8),
                ],
                p=0.8,
            ),
            A.CLAHE(p=0.5),
            A.RandomBrightnessContrast(p=0.6),
            A.GaussianBlur(p=0.4),
            A.RandomGamma(p=0.6),
        ]
    )
    self.tf_augment = A.Compose(
        [
            A.RandomCrop(
                width=int(self.width * 0.5), height=int(self.height * 0.5), p=0.45
            ),
            A.HorizontalFlip(p=0.5),
            A.VerticalFlip(p=0.2),
            A.RandomScale(scale_limit=0.2, p=0.8),
            A.ElasticTransform(alpha=32, sigma=50, alpha_affine=120 * 0.03, p=0.6),
            A.OpticalDistortion(p=0.3),
            A.GridDistortion(num_steps=2, distort_limit=0.2, p=0.4),
            A.ShiftScaleRotate(
                shift_limit=0.0625, scale_limit=0.1, rotate_limit=45, p=0.6
            ),
            A.RandomRotate90(p=0.25),
            A.CoarseDropout(max_holes=8, max_height=16, max_width=16, p=0.3),
        ],
    )
    self.tf_norm_and_resize = A.Compose(
        [
            A.Resize(
```

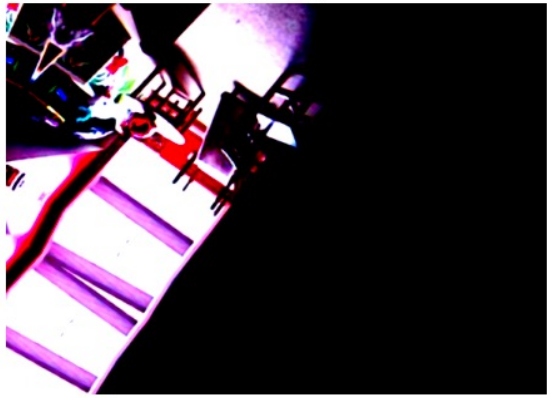
Augmentation

Sample 4637 - normalized: True - split_type: train

RGB image - Original

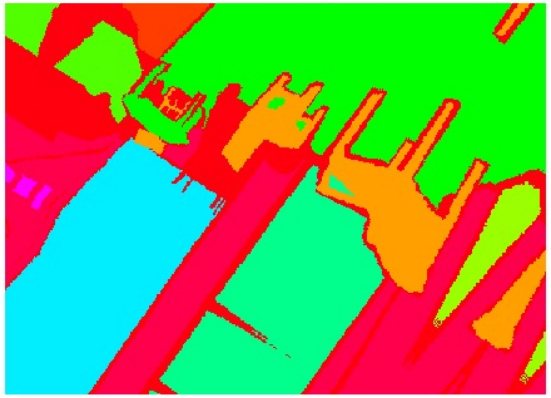


RGB image - Augmented



Depth map - Augmented

Ground truth - Augmented

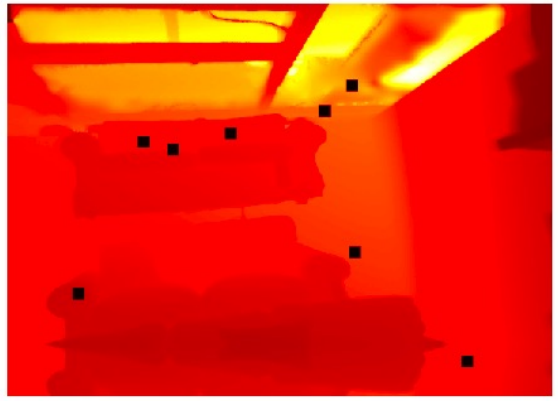


Sample 5443 - normalized: True - split_type: train

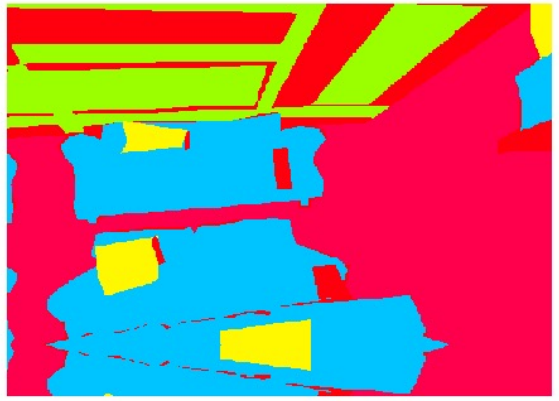
RGB image - Original

RGB image - Augmented

Depth map - Augmented



Ground truth - Augmented

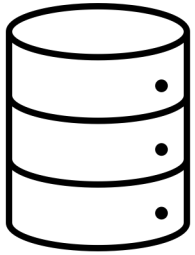


The Network

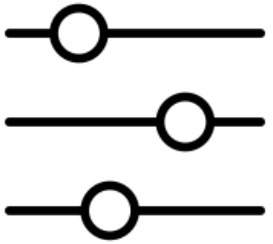
DeResUNet[1]

Layers: 399

(not)



Size: 777.36 MB



Params: 203,780,570

[1] *DualencoderResidualUNet*

The Network

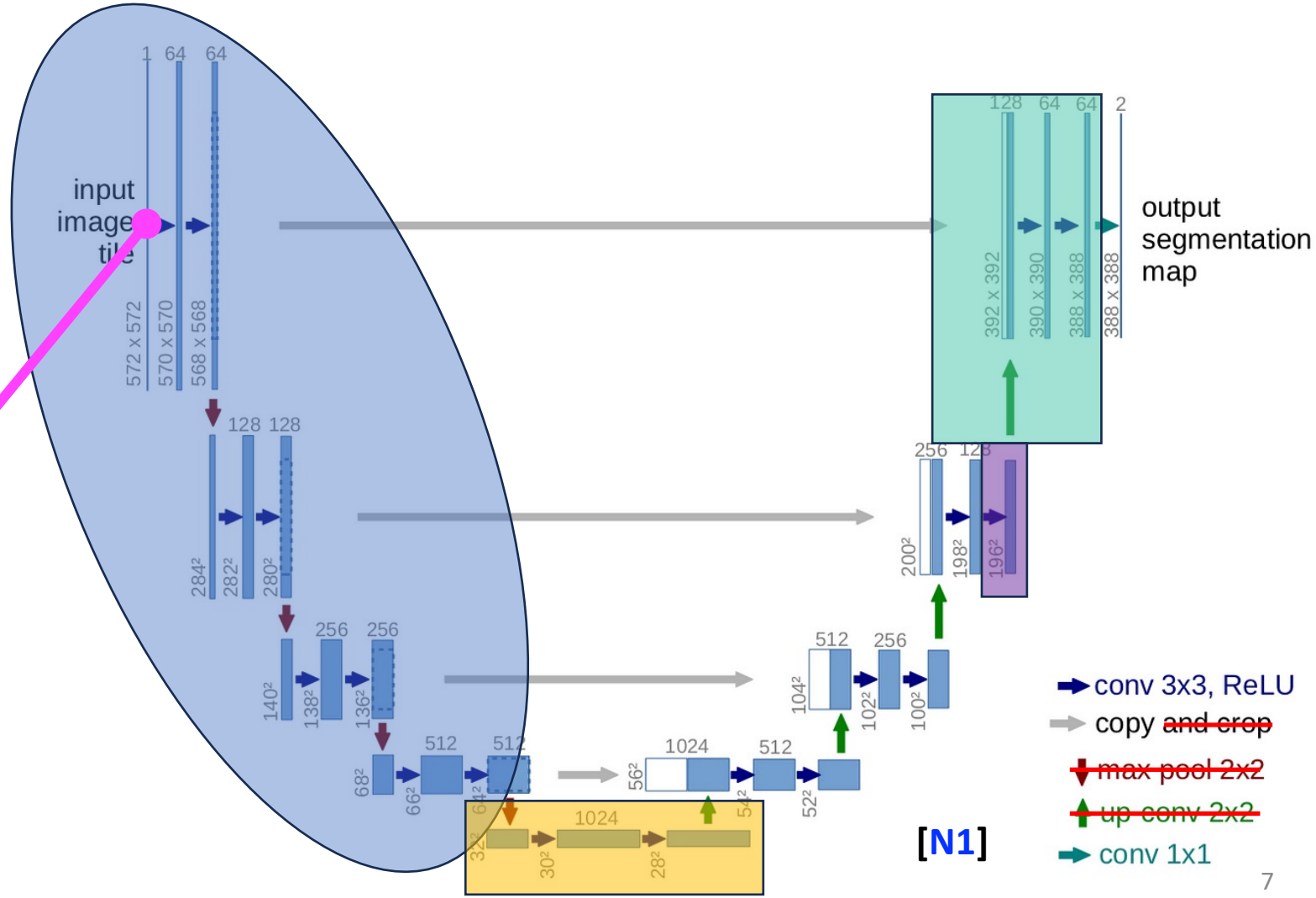
- 2 x parallel ResNet50 (`ResNet50_Weights.DEFAULT`)

$\Longleftrightarrow$ ResNet4Channel

- Bottleneck (latent space)

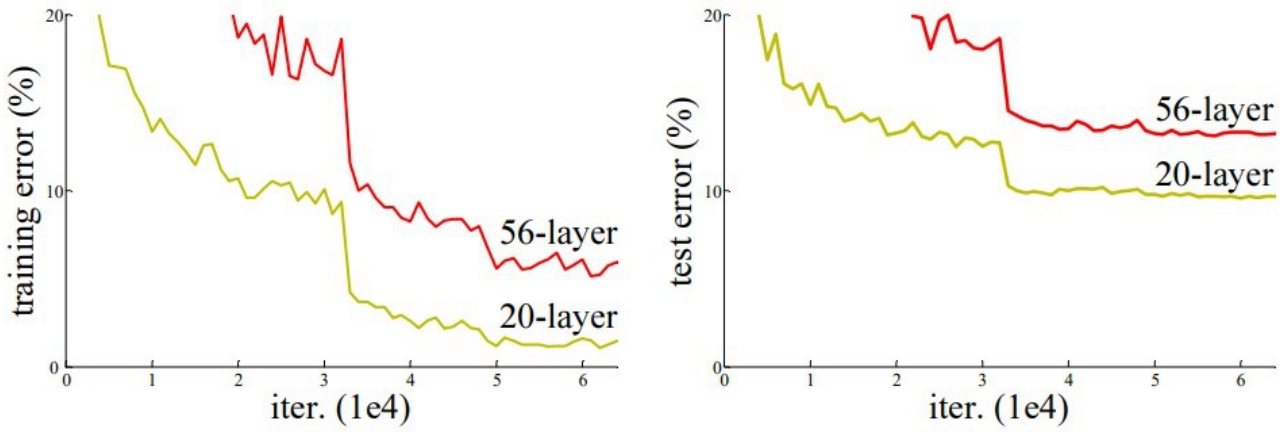
- UpConvPxShuffle[1]
  made out of:
- 2x ConvNormRelu

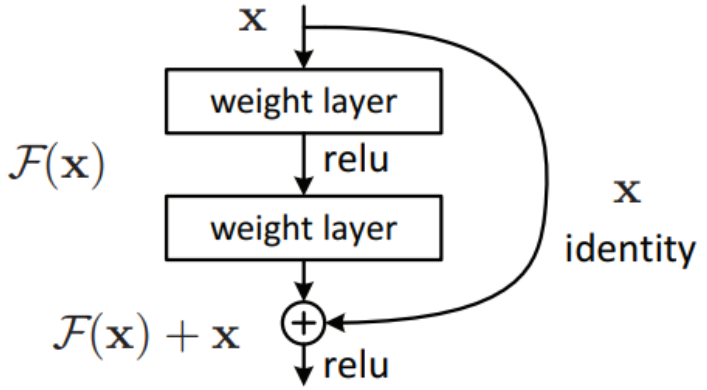
$$T_{in} \in \mathbb{R}^{B \times 4 \times H \times W}$$



The Network

A Residual Block





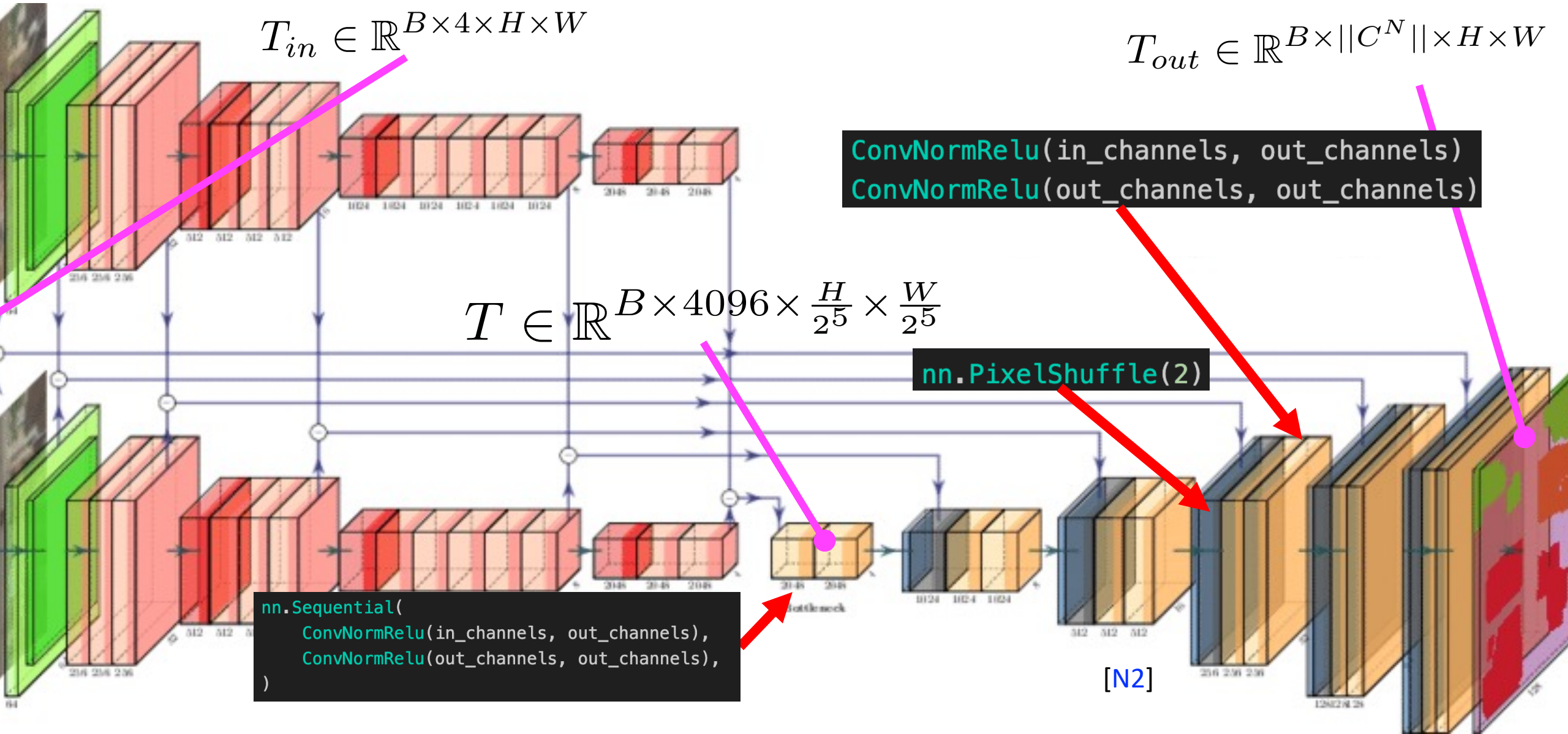
A basic residual block

- Deeper networks -> vanishing gradient problem
- *"Skip connections" that perform identity mapping*
- *Blocks can be stacked*

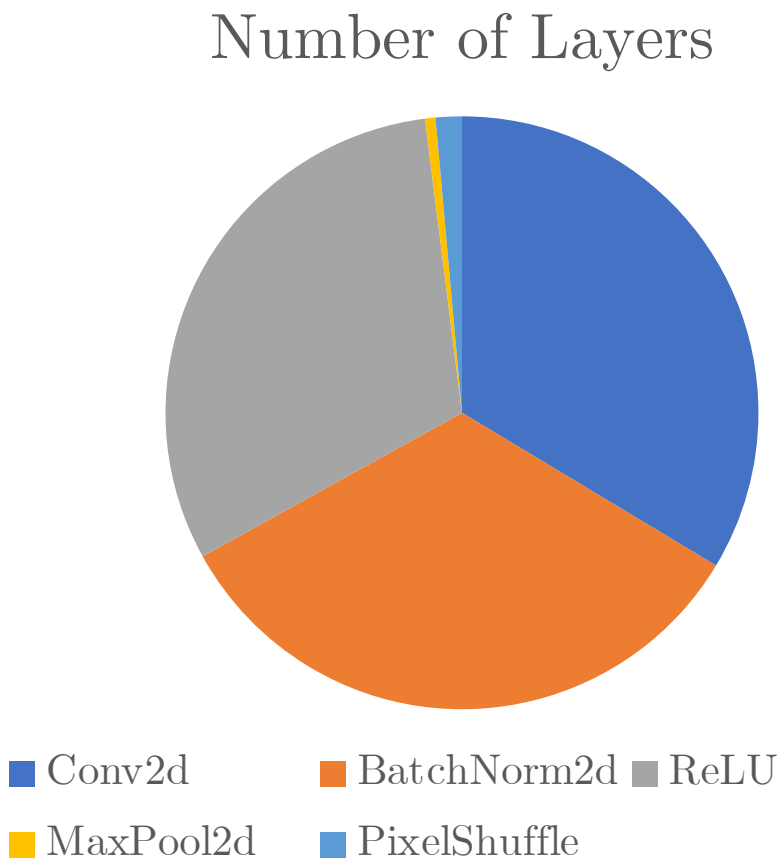
Variants of Residual Blocks:

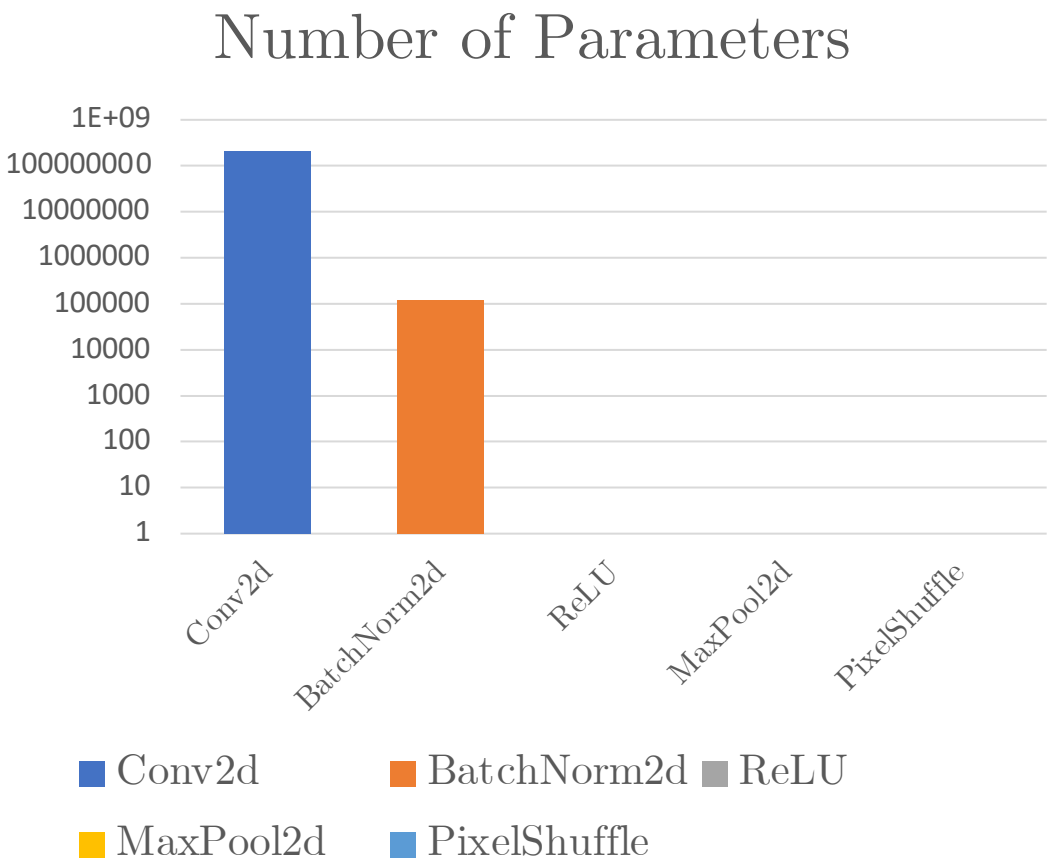
- *Basic Block*
- *Bottleneck Block*
- *Pre-activation Block*
- *Transformer Block*

The Network


$T_{in} \in \mathbb{R}^{B \times 4 \times H \times W}$
$T_{out} \in \mathbb{R}^{B \times ||C^N|| \times H \times W}$
$T \in \mathbb{R}^{B \times 4096 \times \frac{H}{2^5} \times \frac{W}{2^5}}$
ConvNormRelu(in_channels, out_channels)
ConvNormRelu(out_channels, out_channels)
nn.PixelShuffle(2)
nn.Sequential(
ConvNormRelu(in_channels, out_channels),
ConvNormRelu(out_channels, out_channels),
)
[N2]

The Network


Number of Layers
Conv2d
BatchNorm2d
ReLU
MaxPool2d
PixelShuffle

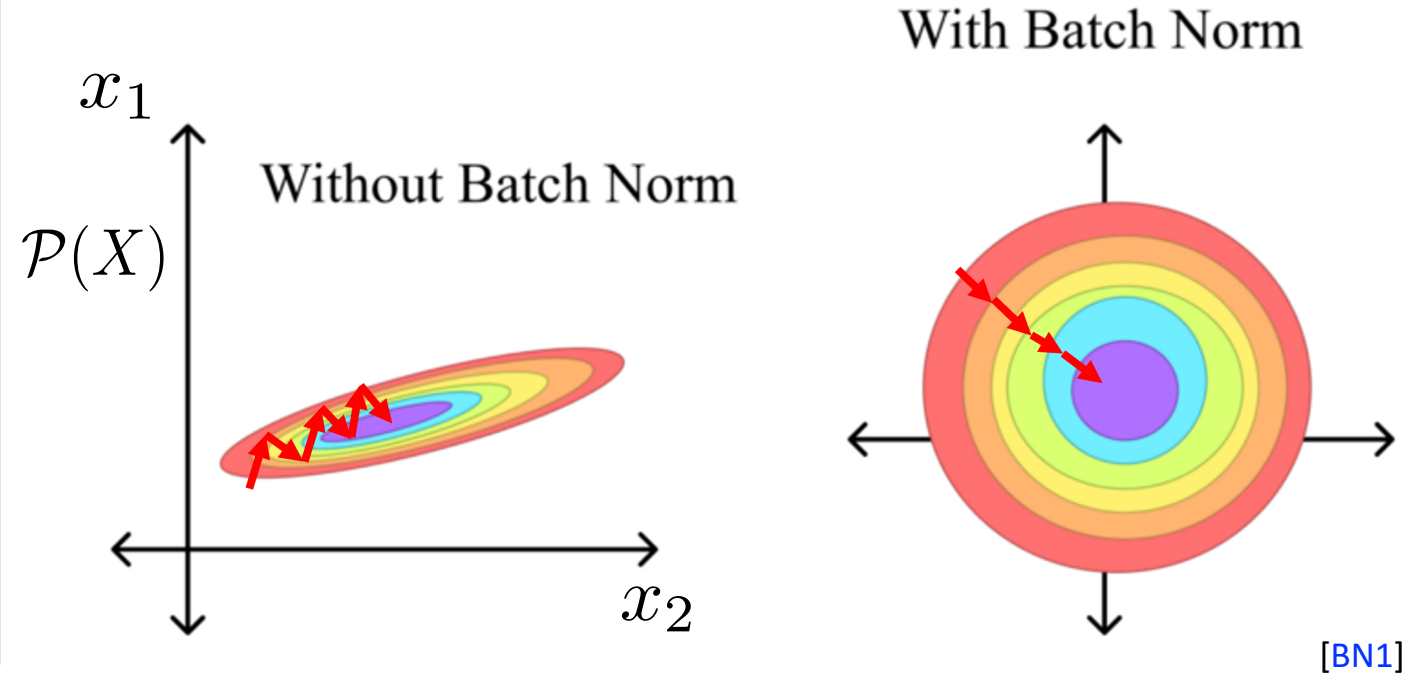

Number of Parameters
1E+09
100000000
10000000
1000000
100000
10000
1000
100
10
1
Conv2d
BatchNorm2d
ReLU
MaxPool2d
PixelShuffle
Conv2d
BatchNorm2d
ReLU
MaxPool2d
PixelShuffle

BatchNorm2D

- Reduces *internal covariate shift*
- Regularization:

$$\mu_x = \mathbb{E}[X_{MB}]$$
$$\sigma^2 = \mathbb{E}[(X_{MB} - \mu)^2]$$
$$, X_{MB} \in \mathcal{P}(X)$$

- Allows higher learning rates
  $\Rightarrow$ faster convergence!
- Decreases bias towards *initial weights*



[BN1]

$$\hat{X}_{MB} = \gamma \left(\frac{X_{MB} - \mu_{MB}}{\sqrt{\sigma_{MB}^2 + \epsilon}} \right) + \beta$$

Learnable params

BatchNorm2D

- Last 1x1 Conv2D layer probably can't deal with the introduced *stochasticity!*

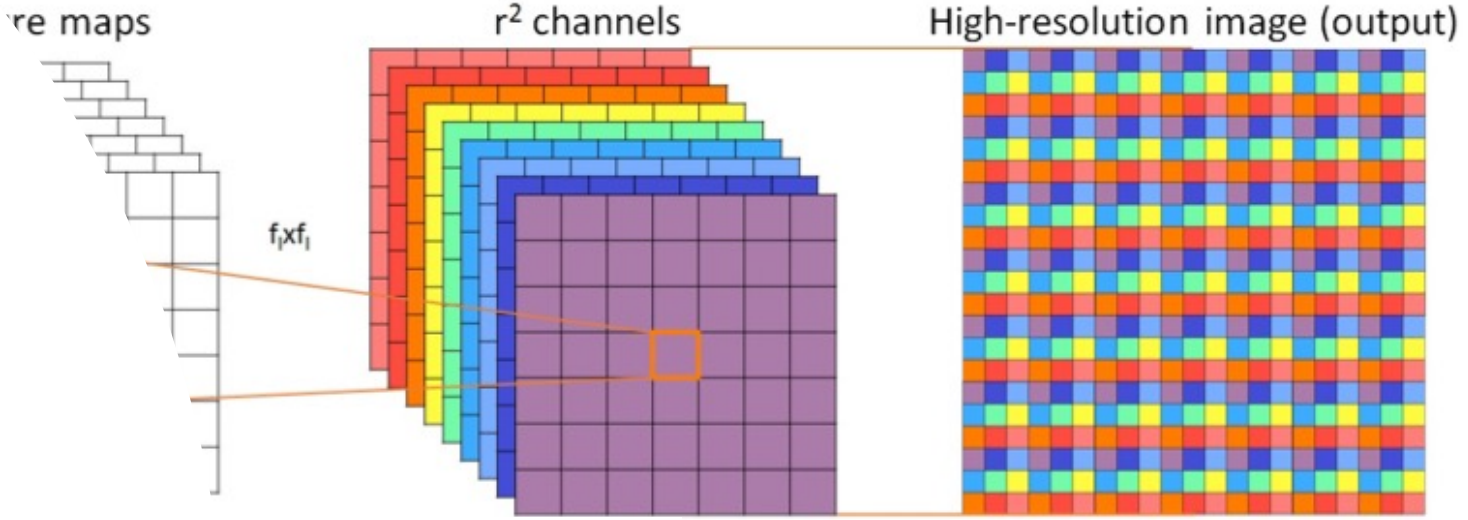
γ, β

```
self.last_upconv = nn.Sequential(
    nn.PixelShuffle(2),
    ConvNormRelu(16, 8),
)
self.last_conv = nn.Conv2d(8, self.num_classes, kernel_size=1, stride=1)
```

```
    PixelShuffle-394         [-1, 16, 480, 640]               0
         Conv2d-395          [-1, 8, 480, 640]           1,160
    BatchNorm2d-396          [-1, 8, 480, 640]              16
           ReLU-397          [-1, 8, 480, 640]               0
    ConvNormRelu-398         [-1, 8, 480, 640]               0
         Conv2d-399         [-1, 50, 480, 640]             450
```

PixelShuffle

$\mathcal{P}$ →periodic shuffling operator
LR →low resulution
SR →super resulution



$$\mathcal{PS} : T_{LR} \in \mathbb{R}^{C \cdot r^2 \times H \times W} \rightarrow T_{SR} \in \mathbb{R}^{C \times H \cdot r \times W \cdot r}$$

Sub-pixel convolution layer

[PS1]

- Performs a "sub-pixel convolution" (stride = $1/r$) by re-arranging its input pixels

⊕ efficient
⊕ no learnable params
⊕ less artefacts (than ConvTranspose)

```
oConvPxShuffle(nn.Module):
__init__(self, in_channels, out_channels, in_channels_up=None):
super().__init__()
if in_channels_up is not None:
    in_channels = in_channels + in_channels_up

self.upscale = nn.PixelShuffle(2)
self.conv1 = ConvNormRelu(in_channels, out_channels)
self.conv2 = ConvNormRelu(out_channels, out_channels)

def forward(self, x, x_cat):
    x = self.upscale(x)
    x = torch.cat([x, x_cat], dim=1)
    x = self.conv1(x)
    x = self.conv2(x)
```

Training

- Batch-size: 6
- Initial Learning Rate: 1e-4
  - With `ReduceLROnPlateau`
- Epochs: 50
  - With `EarlyStopping`
- Input-size: (4, 240, 320)
- Cost function: `CrossEntropyLoss`

Examples

Segmentation of a random sample

Predicted segmentation

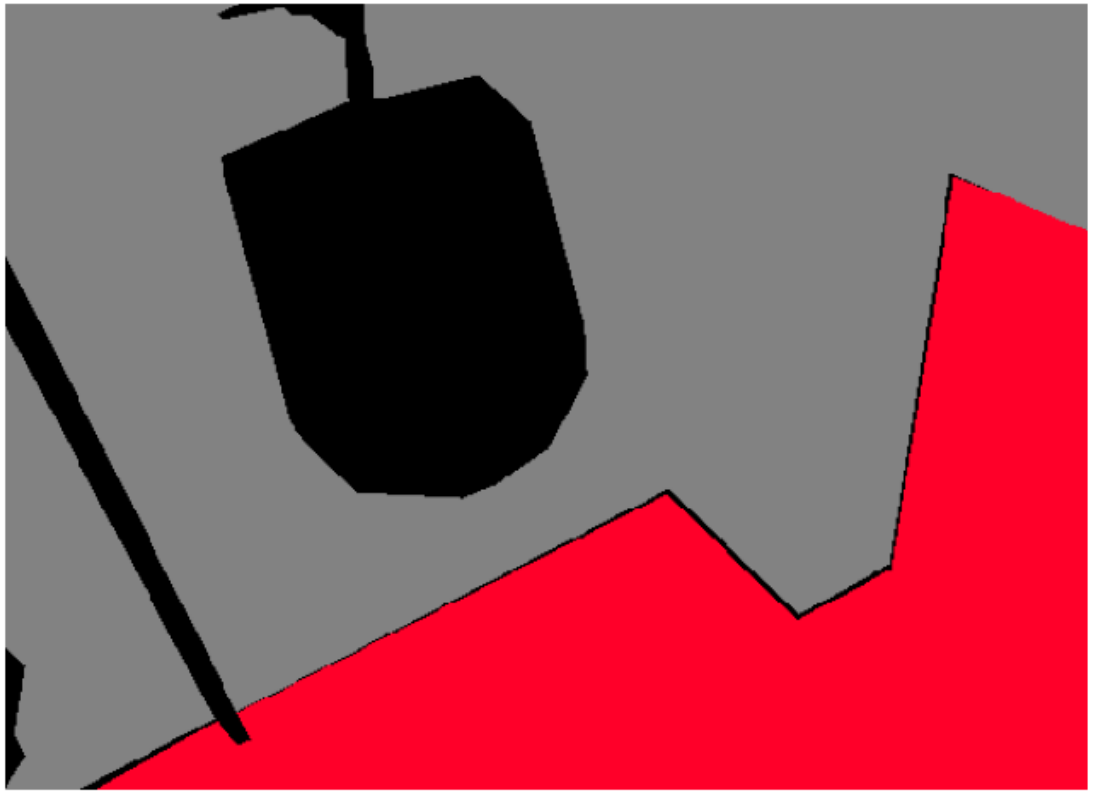
Ground truth

Examples

Segmentation of a random sample

Predicted segmentation

Ground truth



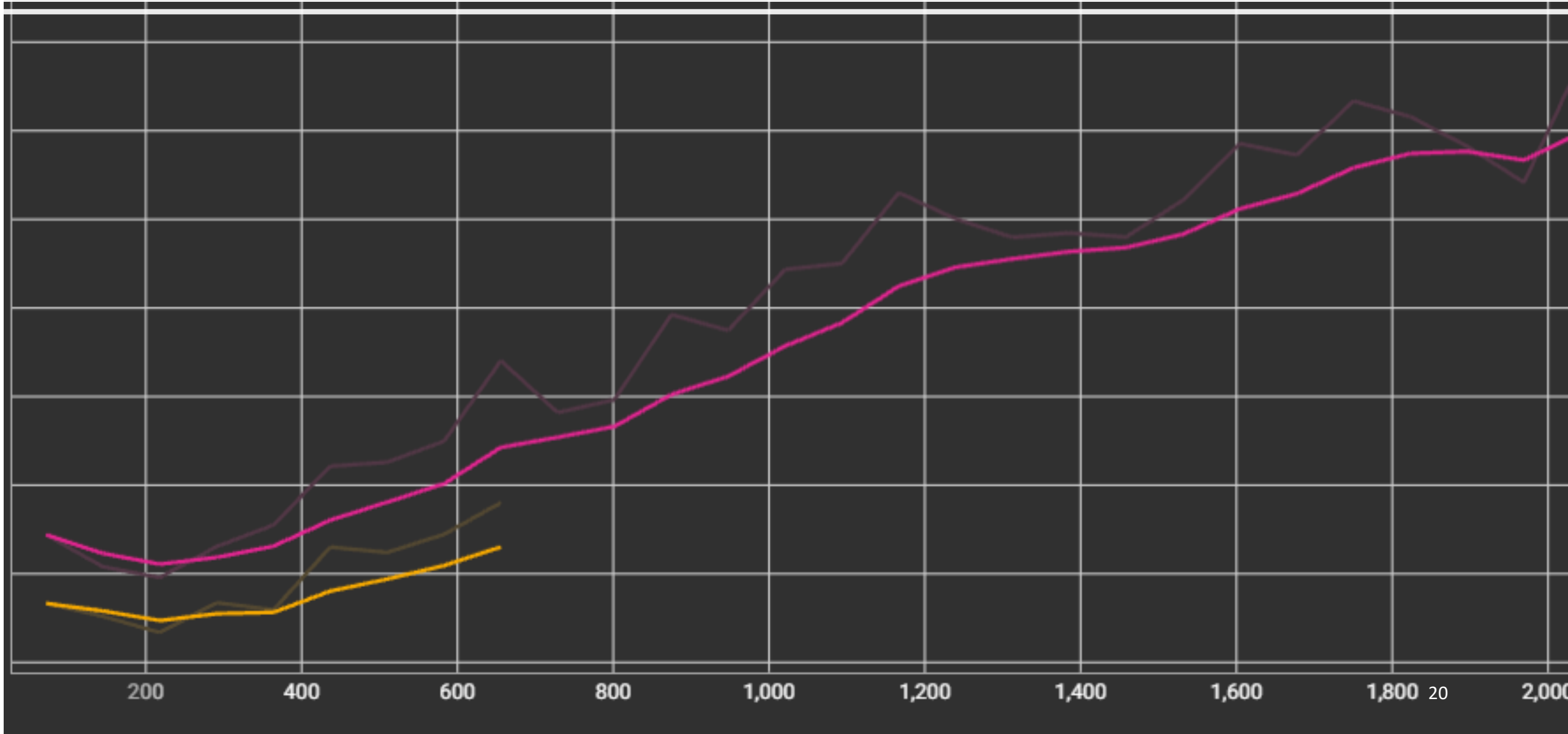
Performance – Loss values

Performance - IoU

- Metric for segmentation & object detection tasks
- Can be used as regression loss
- Plateau for non-overlapping areas
- Addressed by GIoU

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Performance – IoU values



PyTorch Lightning

#earlyStopping

#checkpoints

#logging

#easyScaling

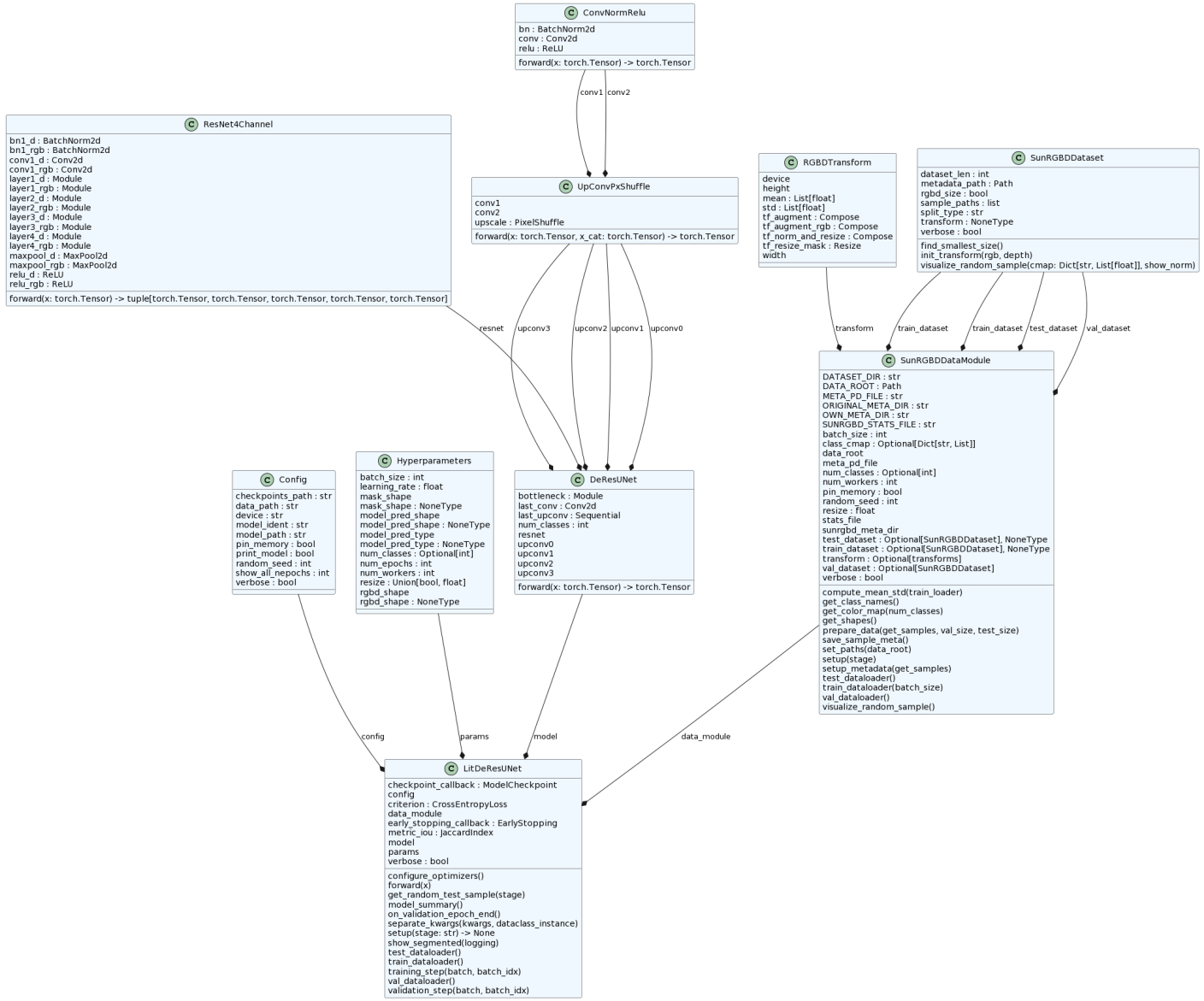
#noBoilerplate

#visualization

#metrics

- PyTorch Lightning
- PyTorch Lightning Tutorials by Aladdin Persson

- [Full Project UML]



[S1] https://pbs.twimg.com/media/Fv1EjNuakAE44z_.jpg:large
[N1] https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/
[N2] https://www.researchgate.net/publication/346089961_Assessing_out-of-domain_generalization_for_robust_building_damage_detection
[PS1] https://arxiv.org/pdf/1609.05158v2.pdf
[BN1] https://www.linkedin.com/pulse/ways-improve-your-deep-learning-model-batch-adam-albuquerque-lima/
[A1] https://towardsdatascience.com/getting-started-with-albumentation-winning-deep-learning-image-augmentation-technique-in-pytorch-47aaba0ee3f8

\imagine The intersection of science and machine learning visualized as a sprawling metropolis --s 100 --style raw --version 5.2