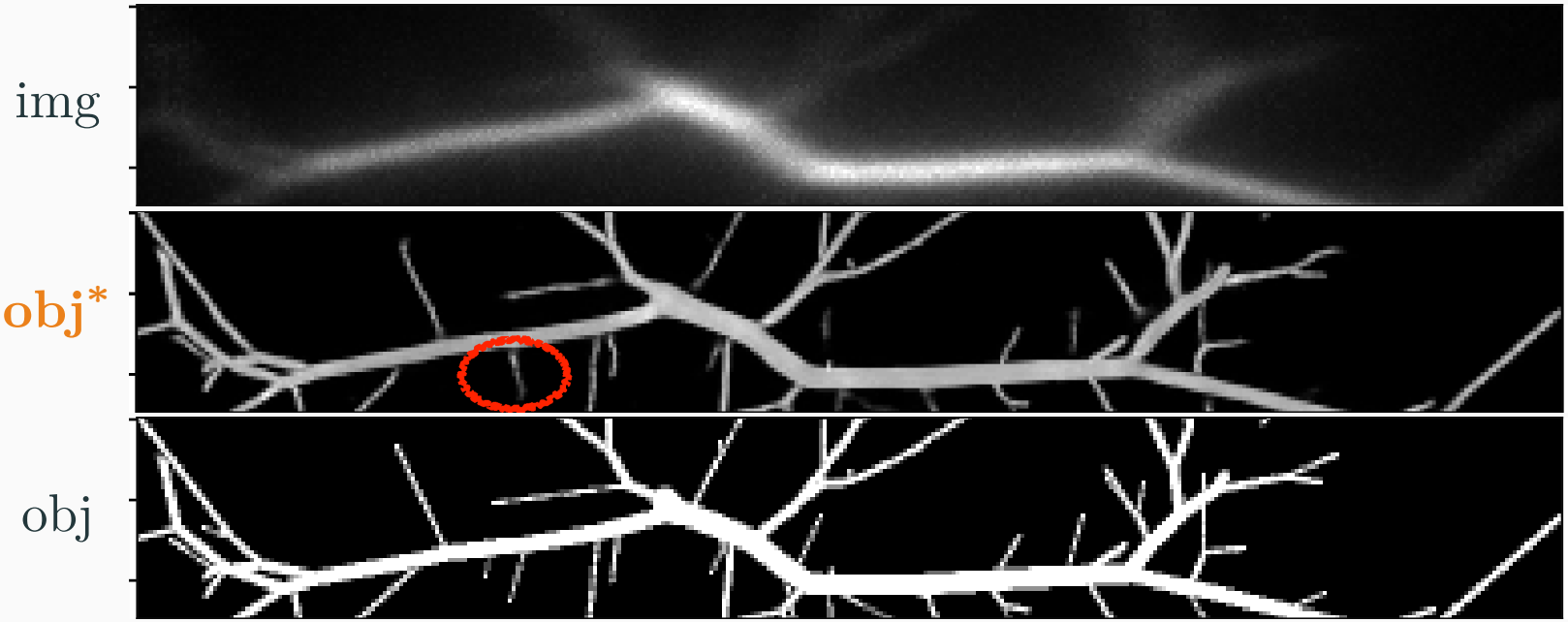
Using NDNet for Image Processing

Sönke Ziemer

20.12.2018

Master Thesis: 3D Deconvolution with a CNN

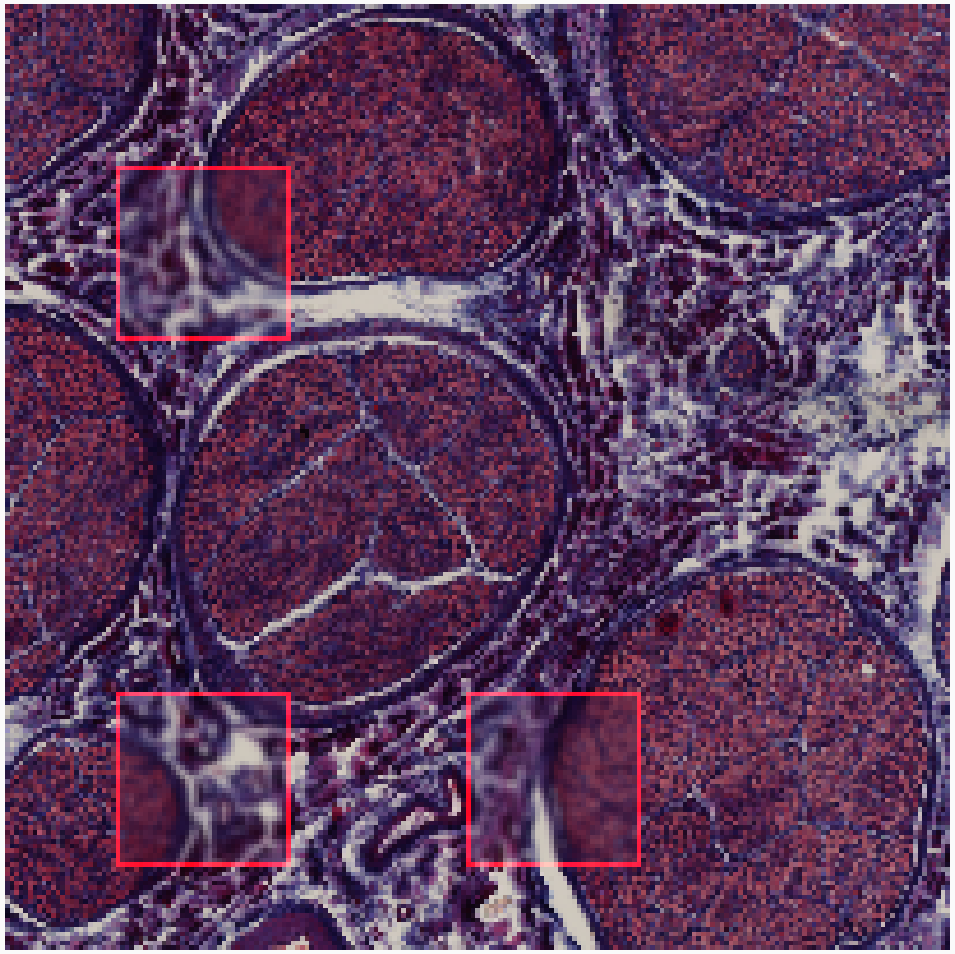
Pixel-wise Mean Squared Error Loss



Actual Result; obj* is the reconstruction; shown are Max-Projections

Project: Detecting Blurry Regions in 2D Images with a CNN

Semantic Segmentation Task; Pixel-wise Softmax Cross Entropy Loss



Desired Output, not a real result

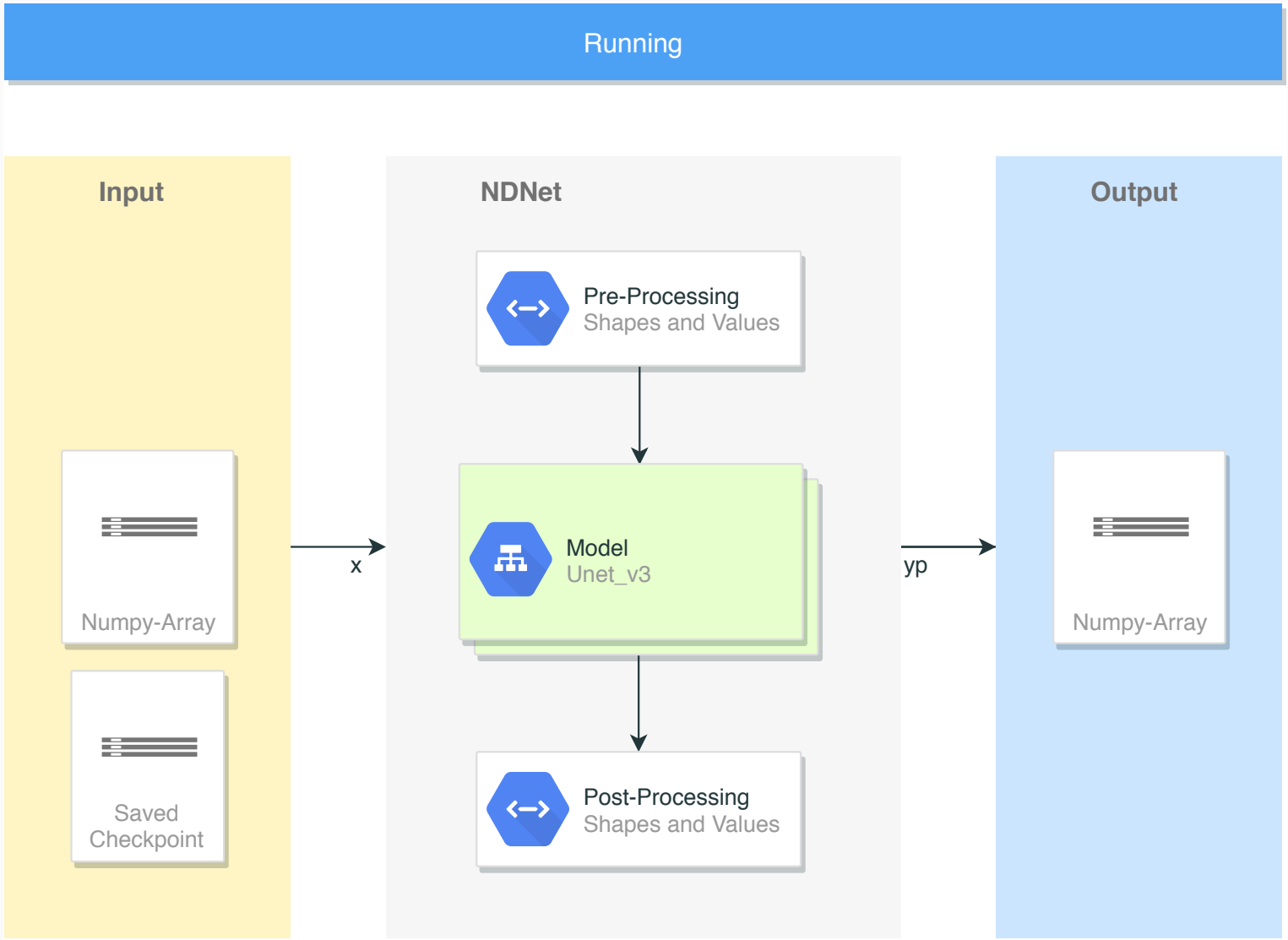
NDNet

Highlights:

- Simplifies Data Input by using a **standardized input pipeline**
- Can handle **2D and 3D** input with an **arbitrary number of input- and output channels** (”N”-D).
- Contains an Implementation of a **U-Net**-like CNN in 2D and 3D
- provides Logging via tensorboard
- Still provides flexibility for experimentation

Uses TensorFlow™

Running existing model with NDNet - Overview



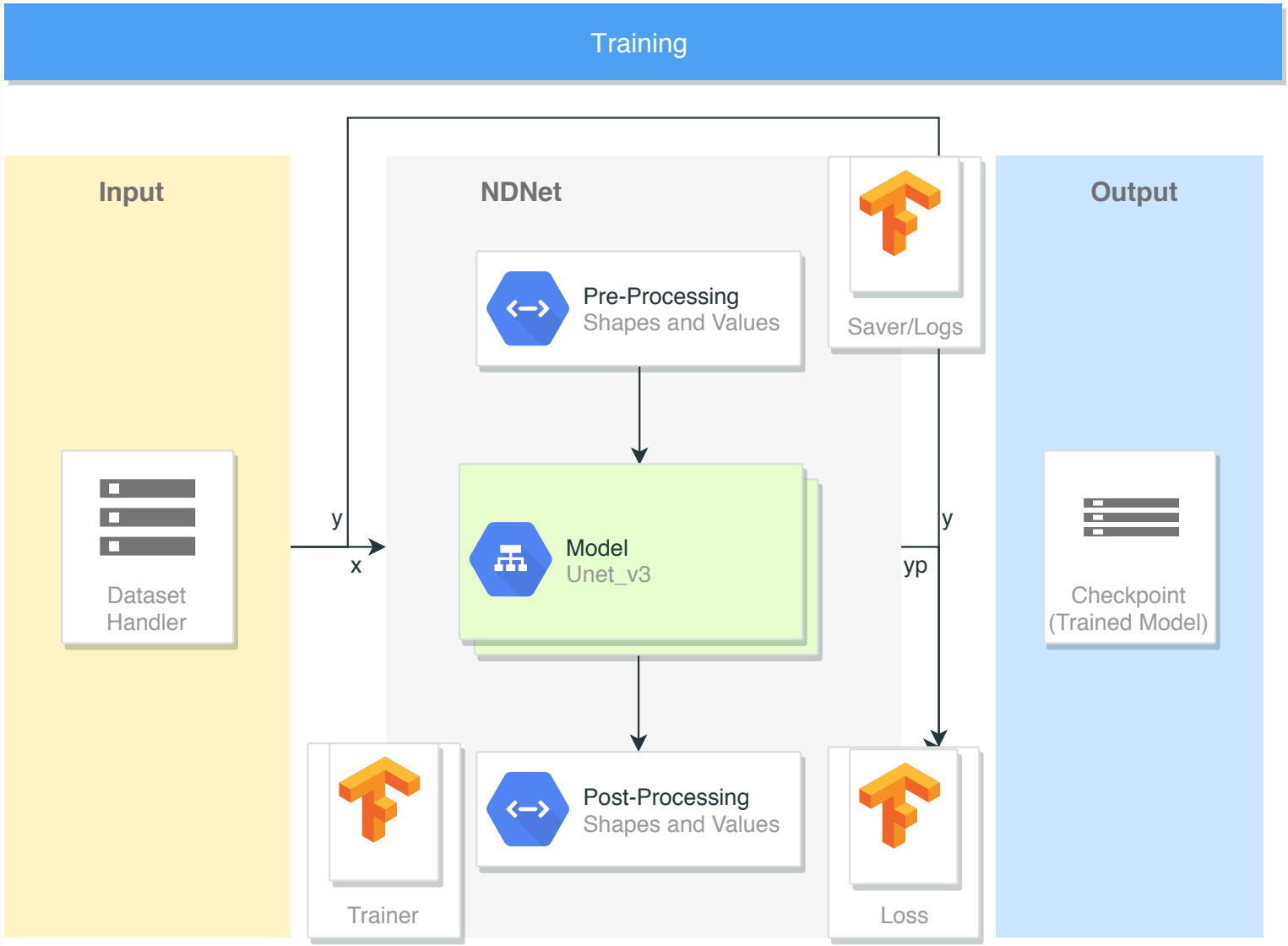
Running existing model with NDNet - Code

```
# path_to_image = os.path.join("testdata", "image0", "im.mat")
import dataset_handlers as dh
np_x = dh.dataset_utils.np_load(path_to_image)

# ids are defined during training
# model_id, dataset_id, run_id, run_number = ...
ckpt = os.path.join("models", model_id, dataset_id, run_id,
    run_number, "ckpts", run_number)
```

Running existing model with NDNet - Code

```
with tf.Session() as sess:
  # NDNet has many more keyword arguments
  deconv_net = NDNet(
    sess=sess,
    arch="unetv3",
    padding='valid')

  # run_on_image checks if deconv_net is compatible with ckpt
  # based on ids that form the path to the ckpt
  np_y = deconv_net.run_on_image(np_x, ckpt, "previous")
```

Training NDNet



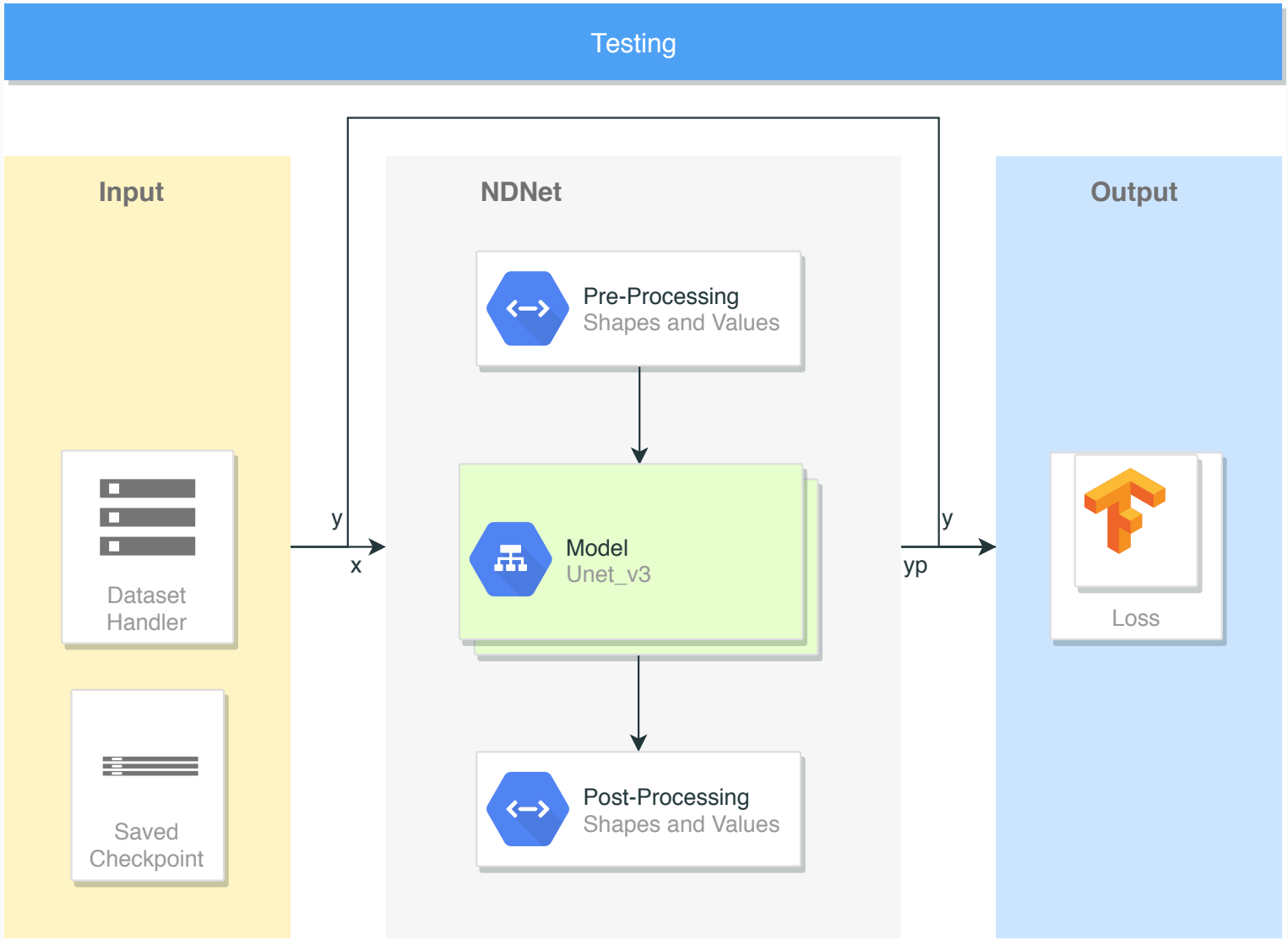
Training NDNet - Code

```
import dataset_handlers as dh
# x_filelist, y_filelist = ...
def load_fn(x_path, y_path):
  """ np-function.  Is converted to tf by dataset handler """
  np_x = dh.dataset_utils.np_load(x_path)
  np_y = dh.dataset_utils.np_load(y_path)
  return np_x, np_y

dataset_handler = dh.tfdata_dataset_handlers.
    BaseListDatasetHandler(x_filelist, y_filelist, load_fn)

with tf.Session() as sess:
  # NDNet has many more keyword arguments
  deconv_net = NDNet(
  sess=sess,
  arch="unetv3",
  padding='valid')

  # ...
```

Training NDNet - Code

```
1
2  # ...
3
4  # train has many more keyword arguments
5  deconv_net.train(
6    training_dataset_handler=dataset_handler,
7    n_epochs=2,
8    batch_size=1,
9    ckpt=None,  # start training from 0
10   learning_rate_fn=lambda global_step: 1e-4,
11   optimizer_fn=lambda lr: tf.train.AdamOptimizer(lr),
12   loss_fn=lf.regression.l2loss)
13
14 # This will store ckpt in os.path.join("models", model_id,
15 #   dataset_id, run_id, run_number, "ckpts", run_number)
```

NDNet


Testing
Input
NDNet
Output
Pre-Processing
Shapes and Values
Dataset
Handler
Saved
Checkpoint
y
x
Model
Unet_v3
Post-Processing
Shapes and Values
y
yp
Loss

Testing NDNet - Code

```
1
2  # dataset handler is defined identical to training
3  # But do use a different filelist :)
4  import dataset_handlers as dh
5  # x_filelist, y_filelist = ...
6  def load_fn(x_path, y_path):
7  """ np-function.  Is converted to tf by dataset handler """
8  np_x = dh.dataset_utils.np_load(x_path)
9  np_y = dh.dataset_utils.np_load(y_path)
10 return np_x, np_y
11
12 dataset_handler = dh.tfdata_dataset_handlers.
       BaseListDatasetHandler(x_filelist, y_filelist, load_fn)
13
14 # ids are defined during training
15 # model_id, dataset_id, run_id, run_number = ...
16 ckpt = os.path.join("models", model_id, dataset_id, run_id,
       run_number, "ckpts", run_number)
```

Testing NDNet - Code

```
with tf.Session() as sess:
  # NDNet has many more keyword arguments
  deconv_net = NDNet(
    sess=sess,
    arch="unetv3",
    padding='valid')

  mean_loss = deconv_net.test(
    testing_dataset_handler,
    ckpt,
    "previous",
    loss_fn=lf.regression.l2loss)
  # test checks if deconv_net is compatible with ckpt
  # based on ids that form the path to the ckpt
```

NDNet Info prints (Running)

```
2018-12-20 09:42:07.967555: I tensorflow/core/common_runtime/gpu
    /gpu_device.cc:1511] Adding visible gpu devices: 0
(... tensorflow Info prints)

input shape: (400, 100, 100, 1)
Cropping to nearest allowed input image size.
building unet_v3 for training
net input shape (1, 398, 94, 94, 1)
input_block1_32 (1, 396, 92, 92, 32)
down_block32_64 (1, 196, 44, 44, 64)
down_block64_128 (1, 96, 20, 20, 128)
bottom_block128_128 (1, 92, 16, 16, 128)
up_block128_64 (1, 180, 28, 28, 64)
up_block64_32 (1, 356, 52, 52, 32)
output_block32_1 (1, 354, 50, 50, 1)
net output shape (1, 354, 50, 50, 1)
output_shape: (354, 50, 50, 1)
```

NDNet Info prints (Training)

```
loss is  l2loss
determining number of trainable vars (except batch_norm) ...
done:  2407969
saving new ckpt and logs in models/
    unetv3_valid_fp0_pp0_bn00_channelslast/poisson_n1000_wl520/
    seed1_bs1_do0.0_loss=l2loss0_weightreg=0.001l2_loss_datareg
    =1e-08None_example/run0
...
starting training with start_step 0
epoch 1 / 2
----->saving 0
----->summarizing 0
----->loss 2101.303
iteration 1 / 1
...
End of sequence
It is not possible to run the final summary.
This behaviour is expected.
----->saving 2 (final state)
```

”Cropping to nearest allowed input image size” ?

In order to be able to concatenate activations from the down-path to the up-path in a meaningful way, input shapes must be considered.

Allowed shapes depend on padding of convolutions (”same” or ”valid”)

Rules:
- Input to every layer must be even
- Output from bottom layer must be able to expand in up-path.

”Cropping to nearest allowed input image size” ?

If shape does not match:

”**same**”: pad input with zeros during preprocessing and cut excess zeros during postprocessing.
$\rightarrow$ Output has same shape as input, but image is darker at the borders.

”**valid**”: crop pixels that do not fit or throw an error if input is smaller than the minimal allowed size. The minimal allowe size is where the border does not consume the entire image plus some extra pixels to allow image to expand in up path.
$\rightarrow$ A significant number of border pixels is cropped. The number grows exponentially with net depth.

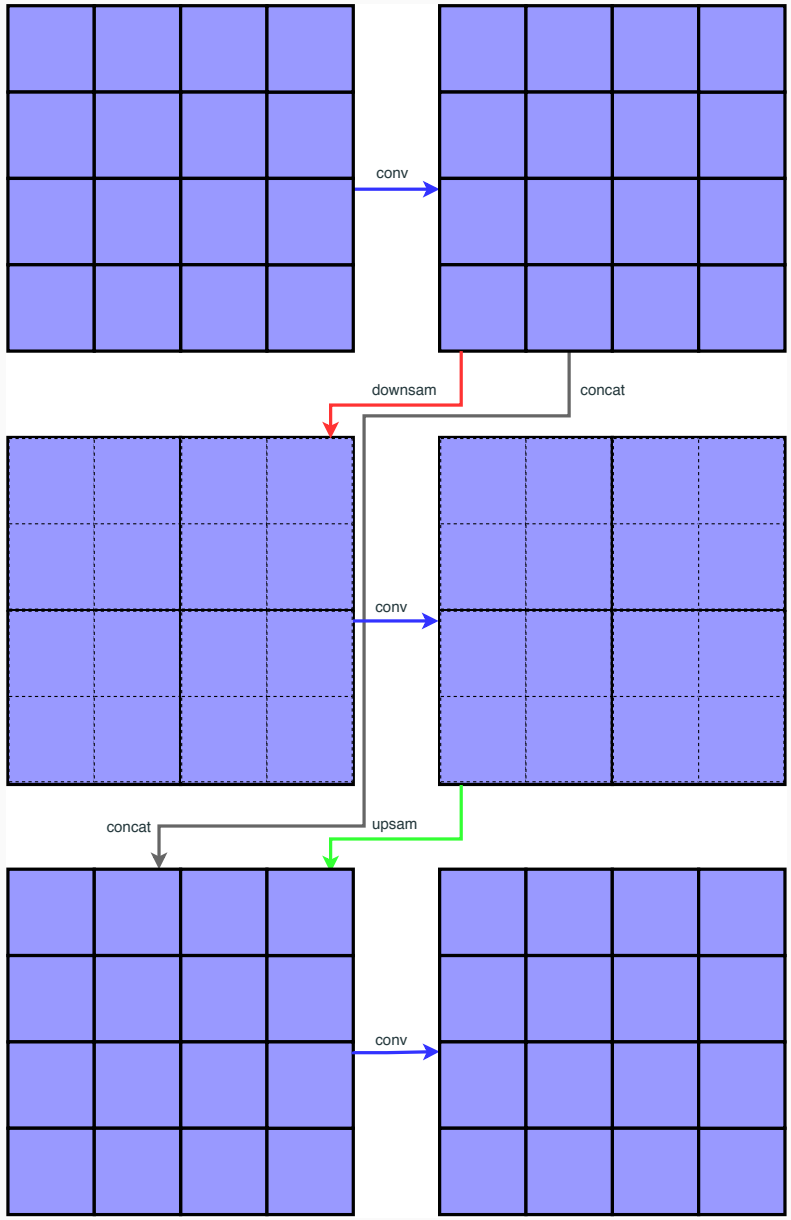
Visualizations

Explanation of the following slides:

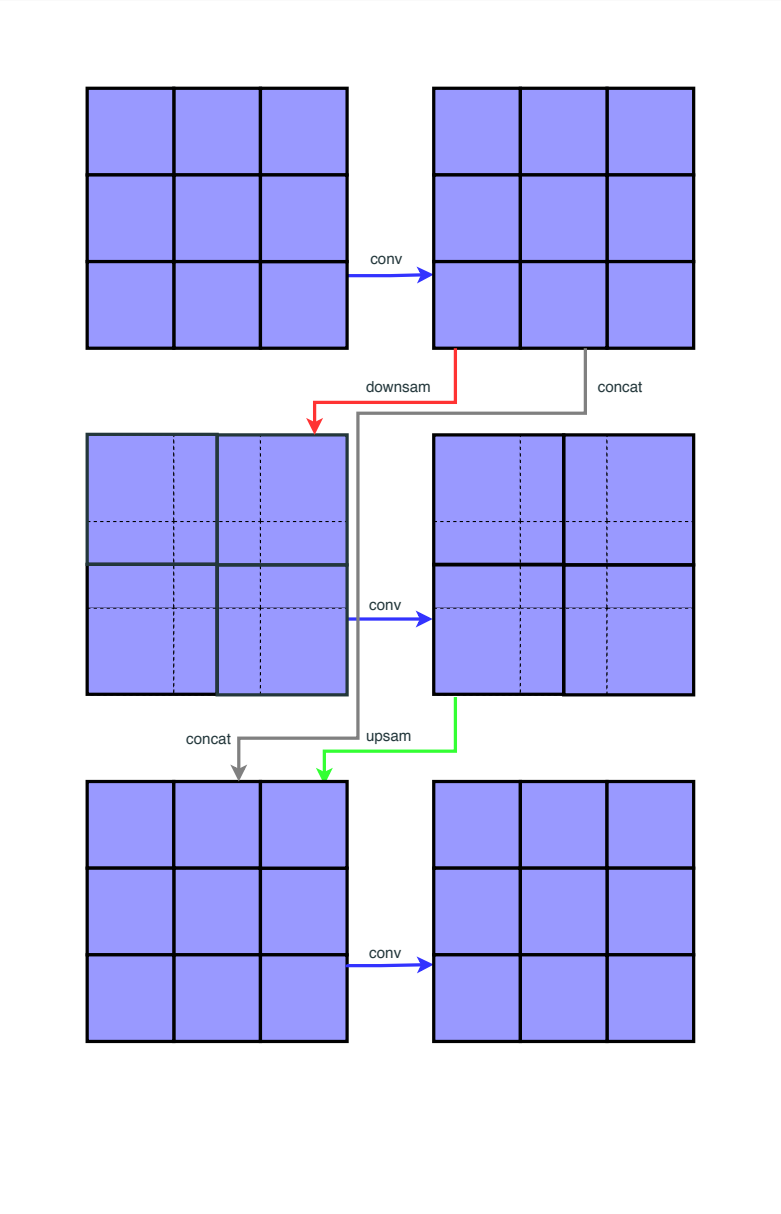
- Displayed are small images (eg. 4x4 pixels for the first one)
- Images are passed through a minimal U-Net
- Upper: Down Layer consisting of one conv
- Upper to Middle: Downsampling. Original scale still visible as dotted lines.
- Middle: Bottom layer consisting of one conv
- Middle to Lower: Upsampling + Concatenation with Top
- Lower: Up Layer consisting of one conv.

Output is on the same scale as input

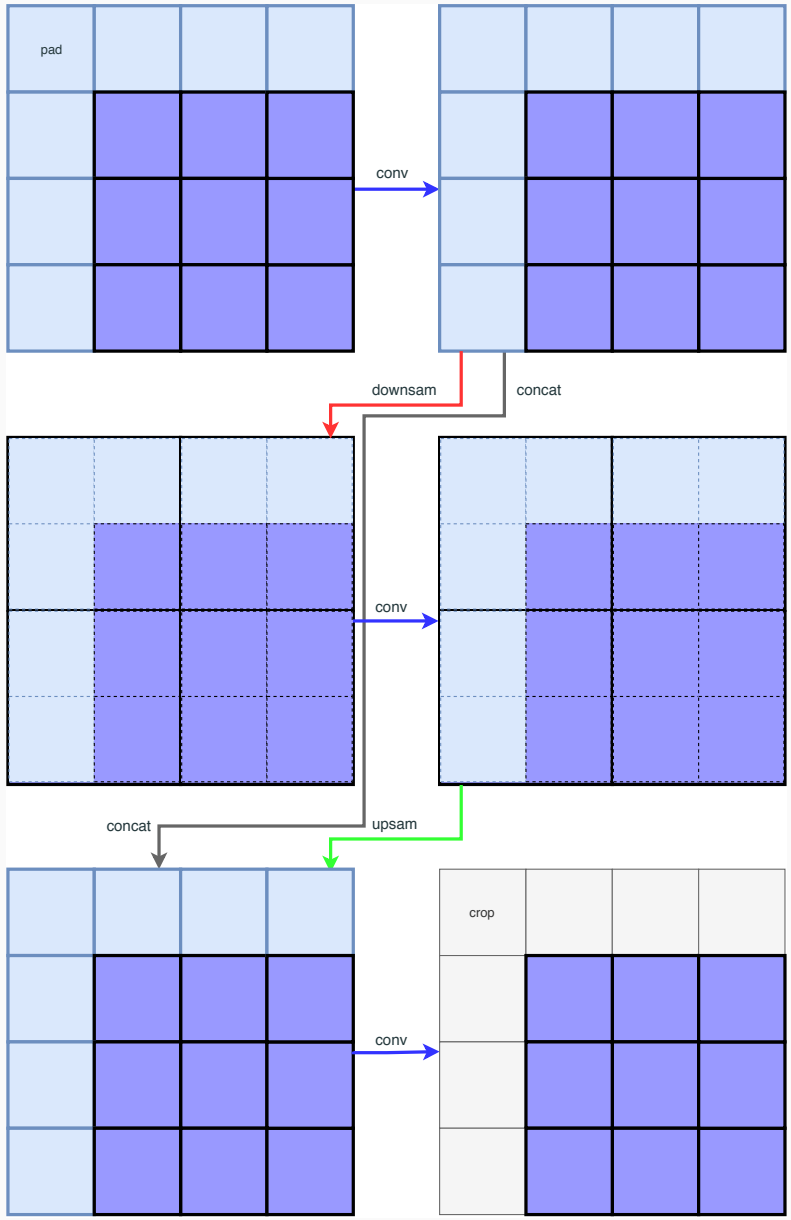
Visualization: ”same” padding ✓ (allowed input size)



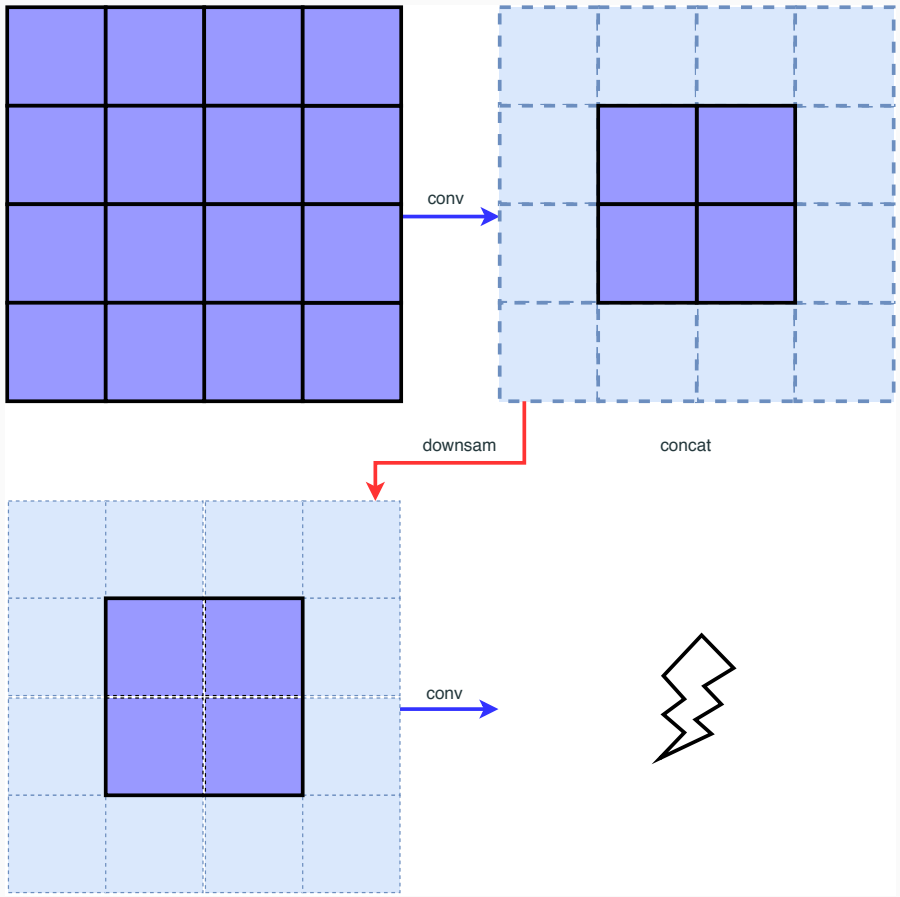
”same” padding $\times$ (not allowed) (not implemented like this)



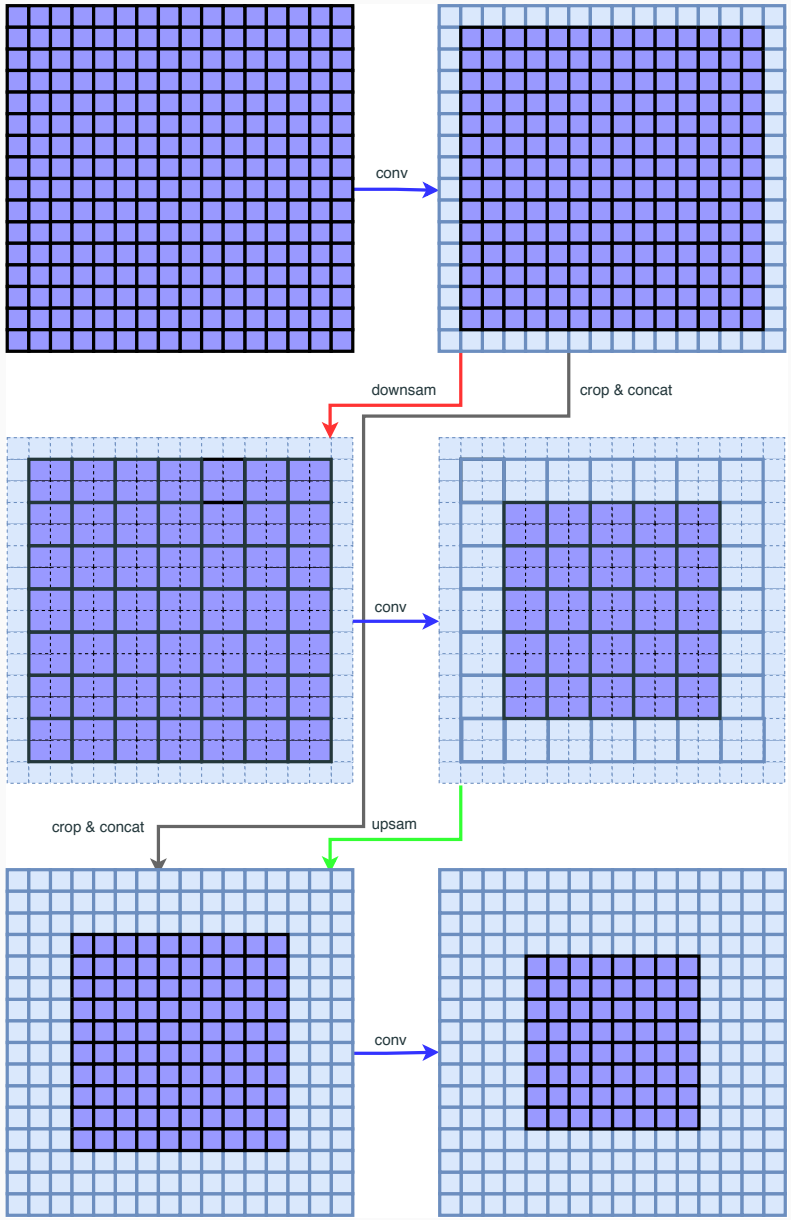
”same” padding: Solution -> zero-padding input



Visualization: ”valid” padding: $\times$ (not allowed) (input too small)



”valid” padding: ✓ (allowed input size)



Dataset Handler Class

Thin layer around tf.data.Dataset
$\rightarrow$ loading, repeating, shuffling, preprocessing, ...
not well profiled, but GPU becomes quite hot

preprocessing done here, especially handling of input shape

Model Class

Handles everything between pre-processing and post-processing

Can be chosen by model string, e.g. (”unetv3”)

```
network_depth=3
initial_channel_growth=32, channel_growth=2
conv_size=(3,3,3), pool_size=(2,2,2)
input_output_skip=False
nonlinearity=tf.nn.relu
```

Other hyperparameters:

```
net_channel_growth, padding, batch_norm, dropout, ...
```

Currently only supports different forms of unet.

Installing NDNet

Requirements: Python 3, Tensorflow, some other packages (included in Anaconda).

NDNet was developed for Tensorflow 1.12.

Get Anaconda

```
https://www.anaconda.com/download/
```

Get Tensorflow

```
https://www.tensorflow.org/install/gpu
conda install tensorflow-gpu
```

Get NDNet

```
git clone ...
```

Thank you for your attention

Appendix