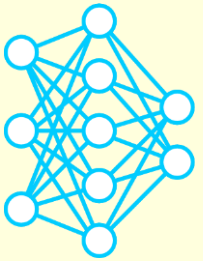
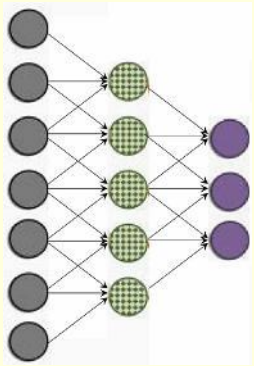


Deep Learning by Example on Biowulf

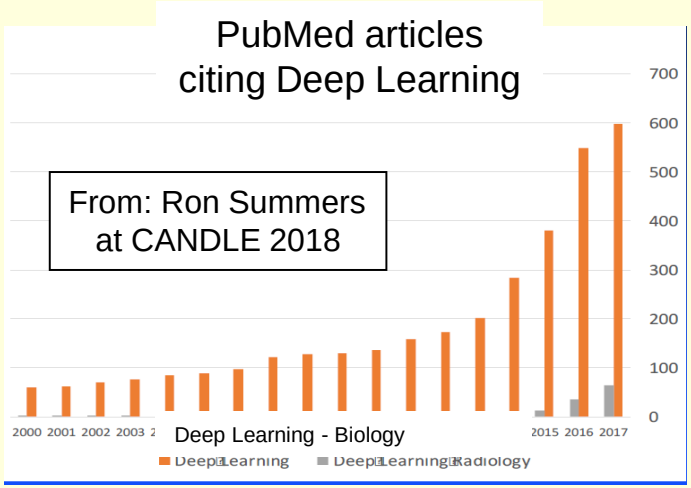
Class #1: Introduction to the deep learning with Keras. Convolutional Neural Networks and their application to semantic segmentation of biomages.

Gennady Denisov, PhD



Goals and target criteria

https://github.com/hussius/deeplearning-biology
https://hpc.nih.gov/docs/deep_learning.html
https://hpc.nih.gov/docs/deeplearning/multinode_DL.html



Standard DL benchmark examples:

- MNIST (hand written characters)

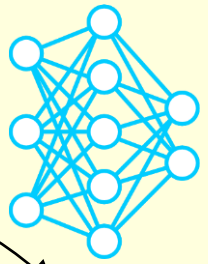
- CIFAR-10

TensorFlow, torch, Caffe, Keras, PYTORCH, Caffe2, theano, NVIDIA DIGITS, mxnet, Lasagne, cuDNN, DL4J

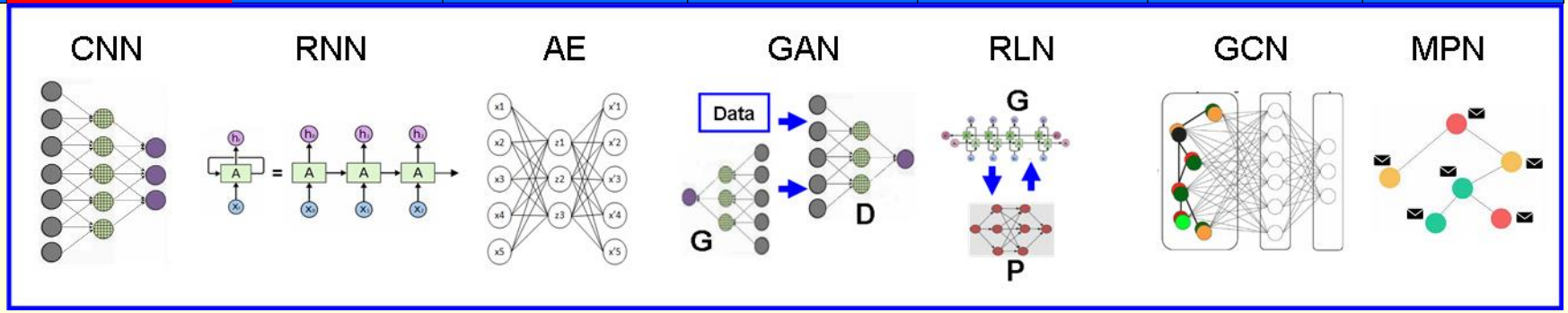
Target criteria for selecting biological examples:

- Cover a wide range of biological applications
- Represent all the major types of DL networks
- Be implemented in **Keras**

Examples summary

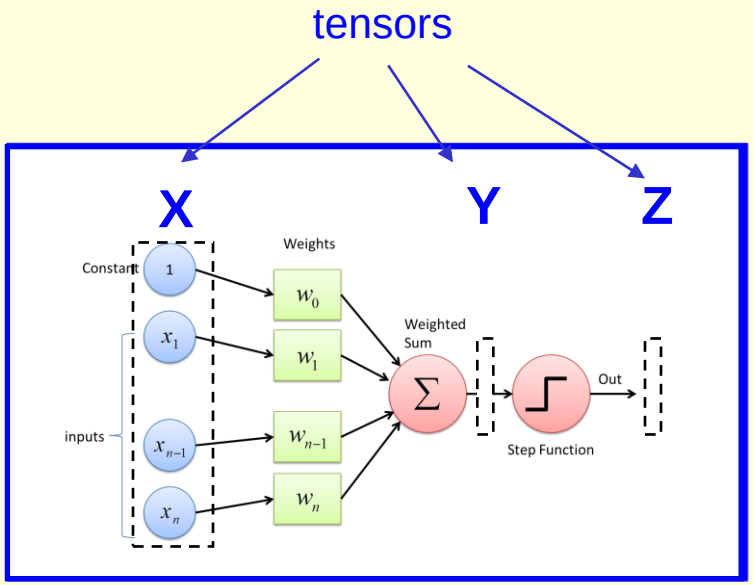


Class #	1	2	3	4	5	6	7
Bio app	Bioimage segmentation / fly brain connectome	Genomics / prediction of function of non-coding DNA	Genomics / reduction of dimensionality of cancer transcriptome	Bioimage synthesis / developmental biology	Drug molecule design	Genomics / classification of cancer types	Drug molecule property prediction
Neural network type	Convolutional	Recurrent or 1D-Convolutional	Autoencoder	Generative Adversarial	Reinforcement Learning	Graph Convolutional	Message Passing
ML type	Supervised	Supervised	Unsupervised	Unsupervised	Reinforcement	Supervised	Supervised



Perceptron: a model of an individual neuron

tensors, transformations, parameters and hyperparameters



Steps of data processing:

1) $Y = \sum w_i \cdot X_i + b; \quad b = X_0$

2) Z = Activation(Y)

Parameters
(adjustable automatically by Keras training procedure)

w_0 , …, w_n

Hyperparameters:
(non-adjustable automatically)

n+1, Activation

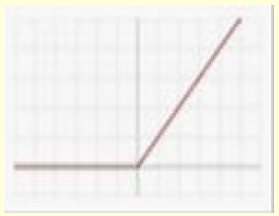
Examples of pre-defined activation functions:

Linear

$Z = \alpha \cdot Y$

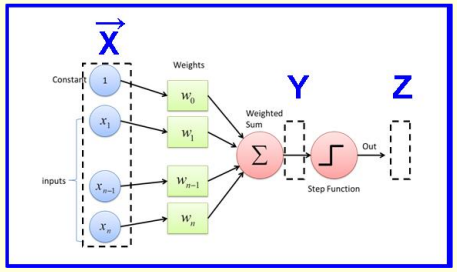
Sigmoid

$Z = 1/ (1 + \exp(-Y))$

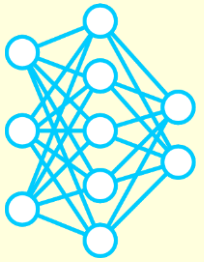


ReLU

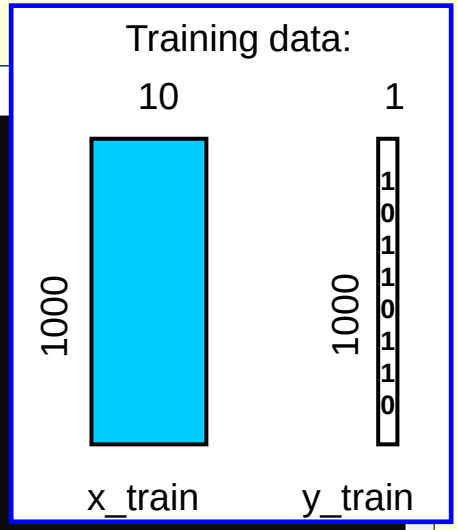
$$Z = \begin{cases} 0, & Y \le 0 \\ Y, & Y > 0 \end{cases}$$



Perceptron training code: the Functional API approach



backend, layer, loss, optimizer, checkpoint, epoch, callback, compile, fit



Header:
- general python imports
- Keras-related imports

Get data
- generate “synthetic” data
- training samples x_train and binary labels y_train

Define a model
- network (=graph)
- compiling
- function to be minimized
- minimization algorithm

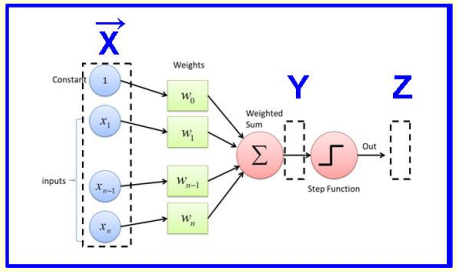
Run the model
- # epochs
- file to store the training results
- function(s) to call at each epoch

Select denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

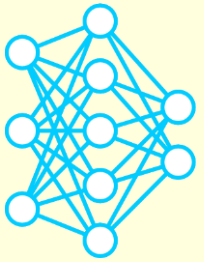
```
#!/usr/bin/env python

# Imports
import numpy as np
from keras.models import Input, Model
from keras.layers import Dense, Activation
from keras.callbacks import ModelCheckpoint

# Get data
num_samples = 1000
num_weights = 10
seed = 1
np.random.seed(seed)
x_train = np.random.uniform(-1, 1, (num_samples, num_weights))
y_train = np.where(np.sum(x_train, axis=1) > 0, 1, 0)

# Define a model
X = Input((num_weights,))
Y = Dense(1, input_dim = num_weights)(X)
Z = Activation('sigmoid')(Y)
model = Model(inputs = X, outputs = Z)
model.compile(loss='mean_squared_error', optimizer='sgd')

# Run the model on the data
checkpointer = ModelCheckpoint(filepath="perceptron.h5")
model.fit(x_train, y_train, epochs=100, callbacks=[checkpointer])
```

Keras <= v2.3.1 backends: Tensorflow (=default), Theano, or CNTK

to change a backend, edit the file: $HOME/.keras/keras.json



Perceptron training code (cont.): the Sequential Construct approach



Header:
- import Sequential
- do not import Activation

Get data
- generate "synthetic" data
- training samples x and labels y

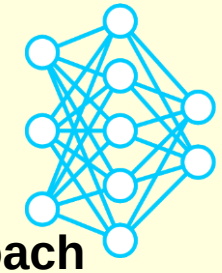
Define a model
- add layers to the Sequential container
- specify activation as a parameter to Dense
- compile

Run the model
- # epochs
- file to store the results
- function(s) to call at each epoch

denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

```
#!/usr/bin/env python

# Imports
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ModelCheckpoint


# Get data
num_samples = 1000
num_weights = 10
seed = 1
np.random.seed(seed)
x_train = np.random.uniform(-1, 1, (num_samples, num_weights))
y_train = np.where(np.sum(x_train, axis=1) > 0, 1, 0)


# Define a model
model = Sequential()
model.add(Dense(1, input_dim = num_weights, activation = 'sigmoid'))
model.compile(loss='mean_squared_error', optimizer='sgd')


# Run the model on the data
checkpointer = ModelCheckpoint(filepath="perceptron.h5")
model.fit(x_train, y_train, epochs=10, callbacks=[checkpointer])
                                                    28,0-1        All
```

The two approaches to building models in Keras: Functional API vs Sequential Construct

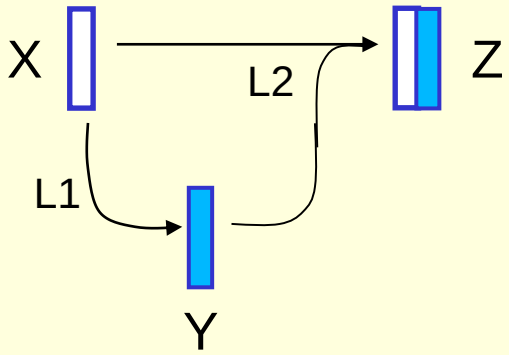


The Functional API approach

- explicitly mentions tensor names
- applicable to any type of networks, both branched and unbranched

```
from  keras.models import Input, Model
from  keras.layers import L1, L2
…
# Define a model
X = Input(…)
Y = L1(X)
Z = L2[ X, Y ]
model = Model( inputs = X, outputs = Z )
model.compile(…)
…
```

Example: “**mini-UNet**”

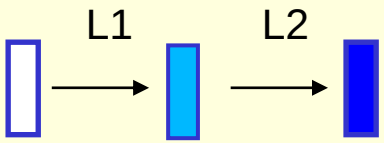


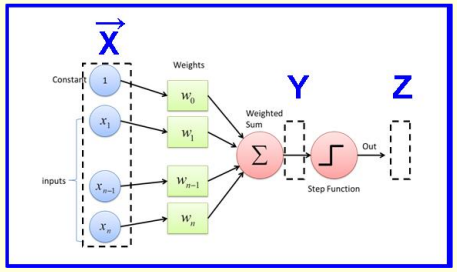
The Sequential Construct approach

- does not explicitly mention tensor names
- a slightly shorter code
- applicable only to unbranched /sequential networks

```
from  keras.models import Sequential
from  keras.layers import L1, L2
…
# Define a model
model = Sequential()
model.add(L1)
model.add(L2)
model.compile(…)
…
```

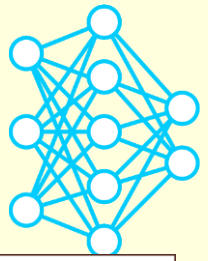
Example:





Perceptron prediction code

load_weights, predict



Header:
- general python imports
- Keras-related imports (no Activation)

Get data
- real data read from disk or “synthetic” data
- testing samples x and labels y

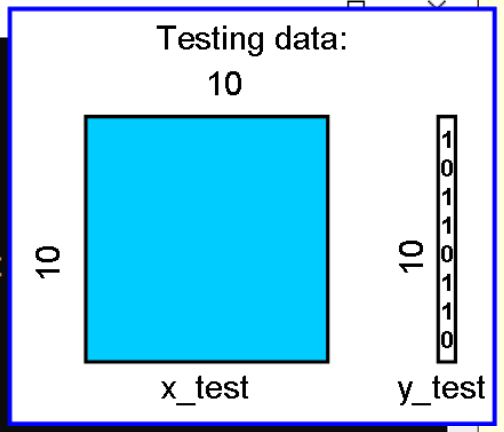
Define a model
- network
- compiling
- function to be minimized
- minimization algorithm

Run the model
- load weights from the the checkpoint file
- predict labels
- compare the predicted labels with ground truth

Select denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

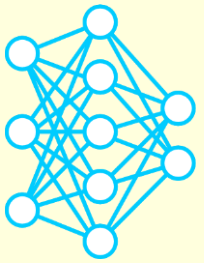
```
#!/usr/bin/env python

# Imports
import numpy as np
from keras.models import Input, Model
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint

# Get data
num_samples = 10
num_weights = 10
seed = 7
np.random.seed(seed)
x_test  = np.random.uniform(-1, 1, (num_samples, num_weights))
y_test  = np.where(np.sum(x_test, axis=1) > 0, 1, 0)

# Define a model
X = Input((num_weights,))
Z = Dense(1, input_dim=num_weights, activation='sigmoid')(X)
model = Model(inputs = X, outputs = Z)
model.compile(loss='mean_squared_error', optimizer='adam')

# Run the model on the data
model.load_weights("perceptron.h5")
y = model.predict(x_test)
for i in range(0,num_samples):
    print("y, y_test=", int(round(y[i][0])), y_test[i])
```

28,0-1 All



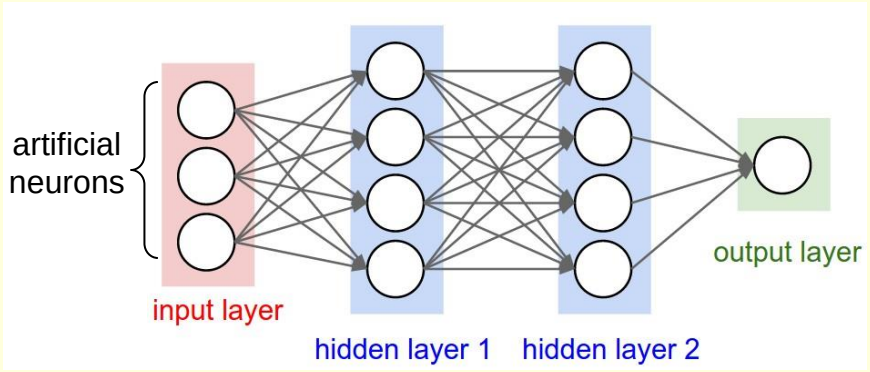
Multilayer Perceptron, a.k.a. Fully Connected Network

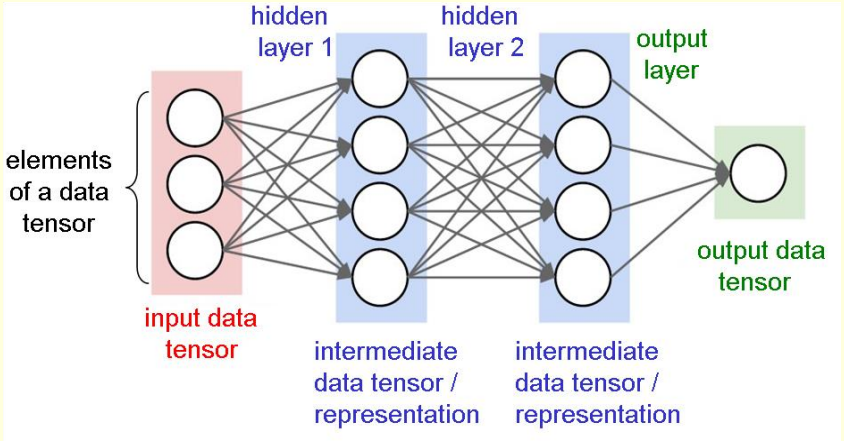
hidden layers, deep network



K.Hornik et al, Neural networks, 2(5):359-366, 1989.
M.Leshno et al, Neural networks, 6(6):861-867, 1993.

Two alternative, but **mathematically equivalent** interpretations of a neural network chart:





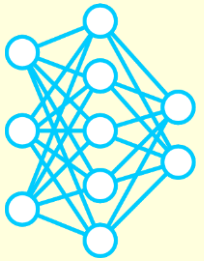
- the interpretation **<u>adopted by this course</u>**:

layer ≈ transformation between data tensors;

hidden layer produces an intermediate data tensor / representation

"Deep neural network":
- number of hidden layers with adjustable parameters ≥ 2
- a universal approximator, i.e. can approximate any function of its input

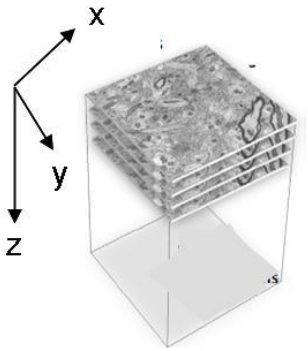
How to run the Perceptron application on Biowulf?



Select denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

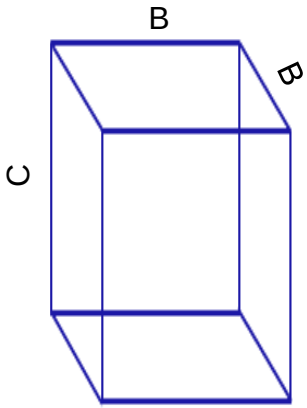
```
[user@biowulf] sinteractive  --mem=4g  --gres=gpu:p100:1,lscratch:10 -c4

[user@cn4464] module load DLByExample/class1
...
[+] Loading DLByExample class1  ....

[user@cn4464] ls $DLBYEXAMPLER_BIN
perceptron_predict.py  perceptron_train.py

[user@cn4464]$ perceptron_train.py
Using TensorFlow backend.
...
Epoch 1/100
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3069
Epoch 2/100
1000/1000 [==============================] - 0s 92us/step - loss: 0.3027
...
Epoch 100/100
1000/1000 [==============================] - 0s 84us/step - loss: 0.1152

[user@cn4464]$ perceptron_predict.py
Using TensorFlow backend.
...
y, y_test= 0 0
y, y_test= 1 1
y, y_test= 0 0
y, y_test= 1 1
...
                                                          2,72           All
```

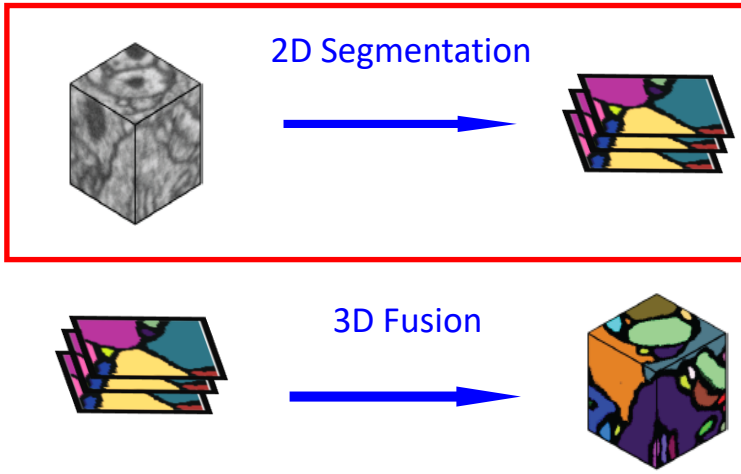
Biological example #1. Semantic segmentation with U-Net: a fly brain connectome project



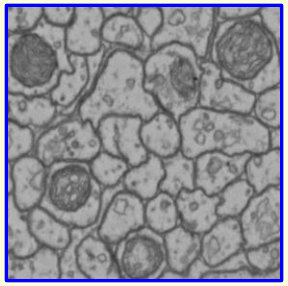
Elementary volume ("voxel")
of the anisotropic (TEM) data

B ~ 2 ÷ 5 nm
C ~ 30 ÷ 50 nm

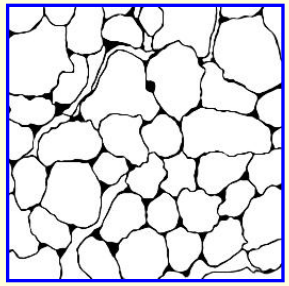




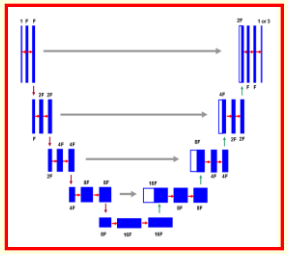
Grayscale image



Binary segmentation / mask



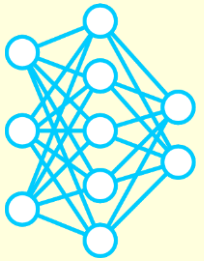
U-Net



Semantic segmentation: **every pixel** is assigned a label

Image classification (e.g. **MNIST and CIFAR10**): **entire image** is assigned a label

Overview of the training code
(only the main function is shown)



Imports statements, other function definitions

Header
- parse command line options

Getting data
- available data
- data augmentation

Defining a model
- UNet model
- Convolution2D
- MaxPooling2D
- UpSampling2D
- loss function
- optimizer

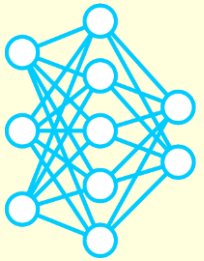
Running the model
- fit_generator
- batch_size

denisovga@biowulf:/data/denisovga/1_DL_Course/1_CNNs

```
if __name__ == "__main__":

    # Parse command line options
    opt, data_gen_args = parse_command_line_arguments("train")
    os.environ['CUDA_VISIBLE_DEVICES'] = "0,1,2,3"

    # Get training data
    input_data_dir = os.path.join(opt.data_folder, opt.object_class, "train")
    orig_data_size = get_orig_data_size(input_data_dir)
    training_data_generator = \
            data.trainGenerator(opt.batch_size*opt.num_gpus,
                                input_data_dir, 'image', 'mask',
                                data_gen_args, target_size=(opt.X, opt.Y),
                                image_color_mode = opt.image_color_mode,
                                mask_color_mode  = opt.mask_color_mode,
                                save_to_dir=opt.save_to_dir,
                                num_classes = opt.num_classes)

    # Define a model
    strategy = tf.distribute.MirroredStrategy()
    with strategy.scope():
        model = get_model(opt)
        model.compile(loss      = opt.loss, \
                      optimizer = Adam(lr = opt.learning_rate),
                      metrics = [opt.callback_metric])
        if opt.load_weights:
            model.load_weights(opt.checkpoint_file)
        model.summary()

    # Run the model
    callback = ModelCheckpoint(filepath=opt.checkpoint_file, \
                               verbose=opt.verbose, save_weights_only=True)
    model.fit_generator(training_data_generator, epochs=opt.num_epochs, \
                        steps_per_epoch=(orig_data_size*opt.augment_rate) // \
                                        (opt.batch_size*opt.num_gpus), \
                        callbacks=[callback], workers=opt.num_gpus,
                        class_weight=opt.class_weights)
                                                                55,61         Bot
```

Sample data for bioimage segmentation

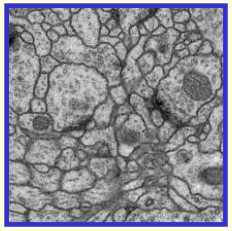
ground truth, overfitting, augmentation, fit_generator

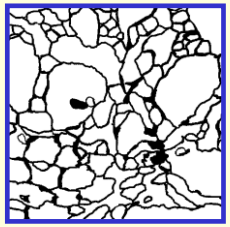


ISBI dataset (size: 30)

http://brainiac2.mit.edu/isbi_challenge/

```
Select denisovga@biow...
data
└── membrane
    ├── test
    └── train
        ├── image
        │   ├── 0.png
        │   ├── 1.png
        │   ├── 2.png
        │   ├── ...
        │   ├── 28.png
        │   └── 29.png
        └── label
            ├── 0.png
            ├── 1.png
            ├── 2.png
            ├── ...
            ├── 28.png
            └── 29.png
                1,1    All
```

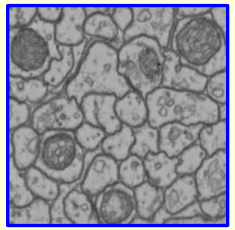


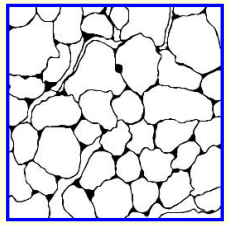


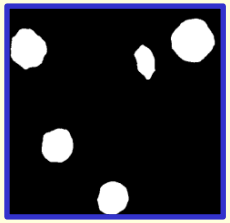
HHMI dataset (size: 24)

Zheng ea, Cell 2018, 174(3), 730-743

```
denisovga@biowulf:/data/den...
data_hhmi
├── membrane
│   ├── test
│   └── train
│       ├── image
│       │   ├── 0.png
│       │   ├── ...
│       │   └── 23.png
│       └── label
│           ├── 0.png
│           ├── ...
│           └── 23.png
└── mito
    ├── test
    └── train
        ├── image
        │   ...
        └── label
            ├── 0.png
            ├── ...
            └── 23.png
                22,0-1    All
```

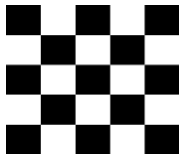




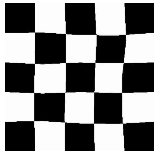


Overfitting: model fits the training data too well; fails to generalize

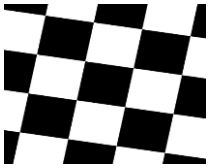
Augmentation: >= 20x; **fit → fit_generator**



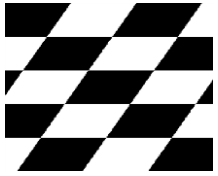
Original image



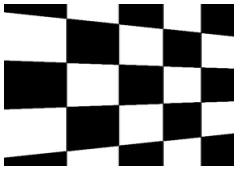
Elastic distortion



Rotate + crop



Shear + crop

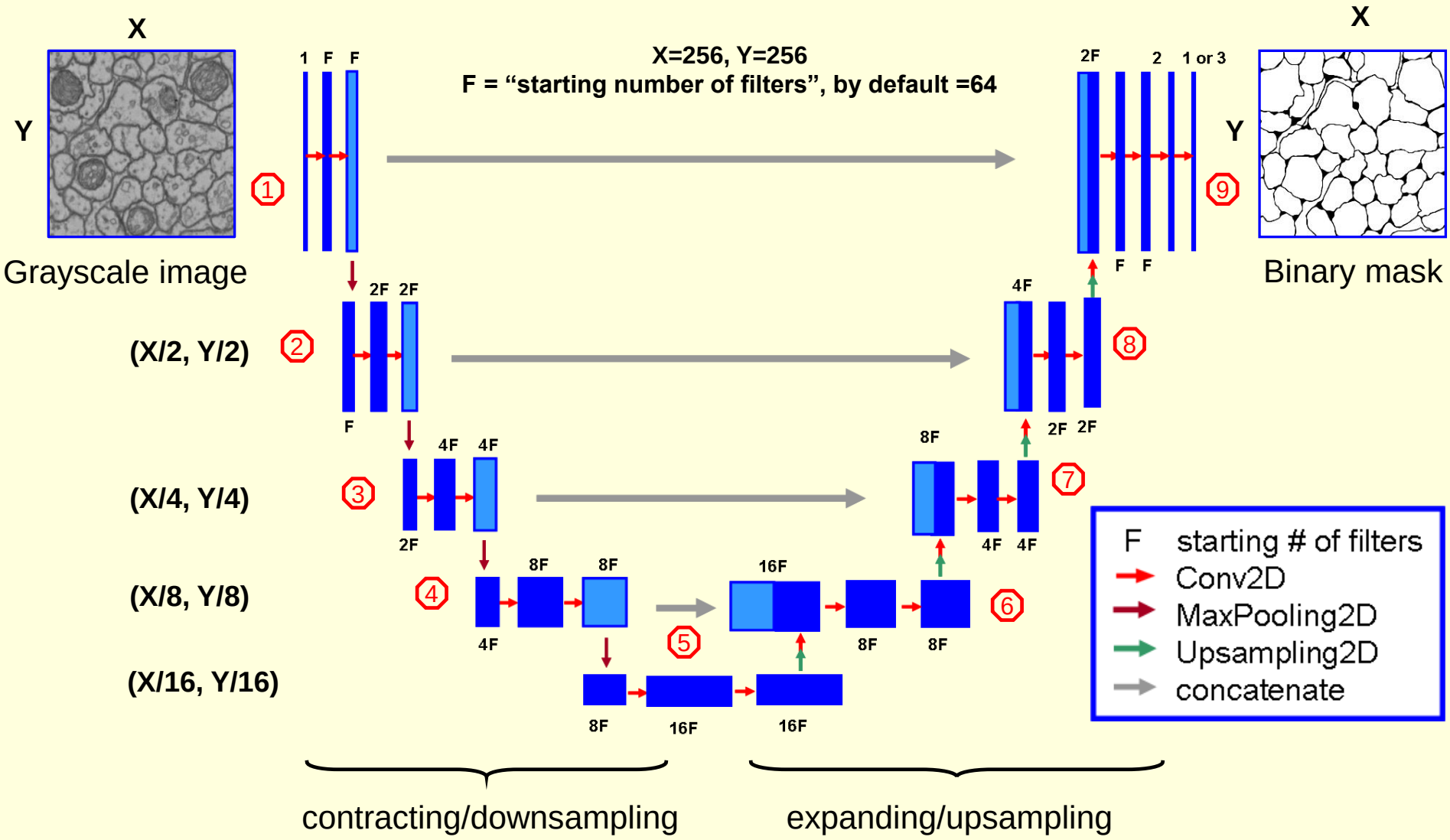


Skew + right tilt

Overview of the U-Net model

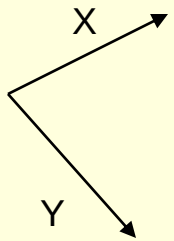
U-Net: O.Ronneberget et al., Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2015

https://github.com/zhixuhao/unet



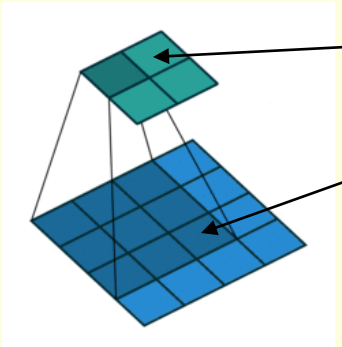
Convolution2D

kernel_size, padding, strides, dilation_rate, stack size



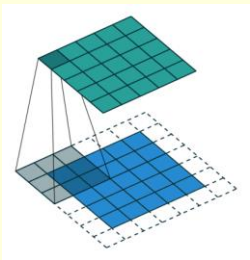
Output image

Input image

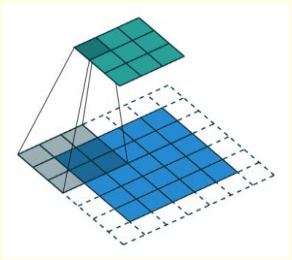


$$O = \sum w_i * I_i + b$$

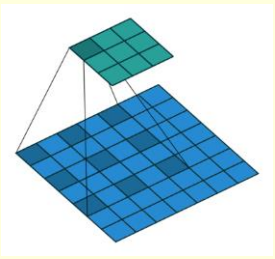
kernel_size = 3; padding = “valid”; stride = 1



padding = “same”, stride = 1



stride=2



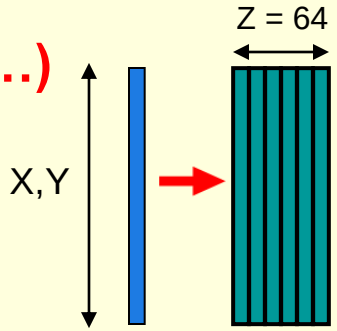
dilation_rate = 2

(dilated, or atrous filter)

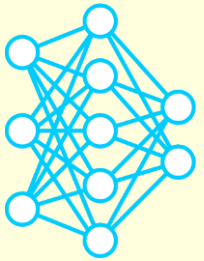
Conv2D(64, 3, padding = ‘same’, …)

number of filters / stack size

filter / kernel size



MaxPooling2D, Upsampling2D and concatenation

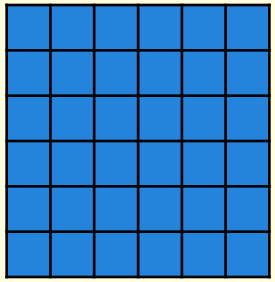


MaxPooling2D(pool_size = (2,2), strides=(2,2))

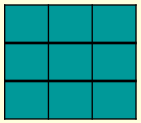
2.0	3.0	0.0	5.0	2.5	0.0
2.0	1.5	0.5	0.0	7.0	0.0
1.5	5.0	5.0	3.0	2.0	0.0
3.0	5.0	7.0	1.5	0.0	0.0
2.0	5.0	2.0	1.5	2.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0

3.0	5.0	7.0
5.0	7.0	2.0
5.0	2.0	2.0

6 x 6



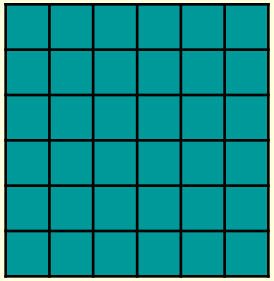
3 x 3



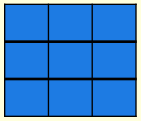
purpose: aggressively downsample data to prevent the model from overfitting

UpSampling2D(size = (2,2))

6 x 6



3 x 3

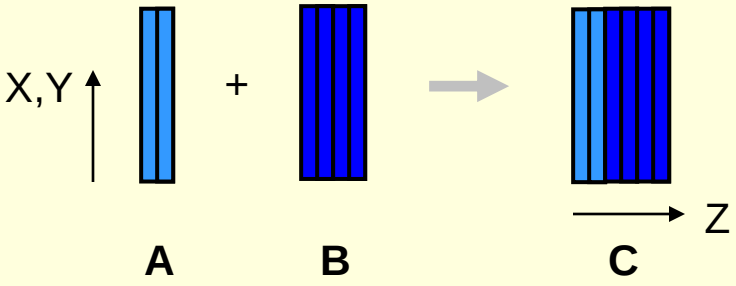


3.0	3.0	5.0	5.0	7.0	7.0
3.0	3.0	5.0	5.0	7.0	7.0
5.0	5.0	7.0	7.0	2.0	2.0
5.0	5.0	7.0	7.0	2.0	2.0
5.0	5.0	2.0	2.0	2.0	2.0
5.0	5.0	2.0	2.0	2.0	2.0

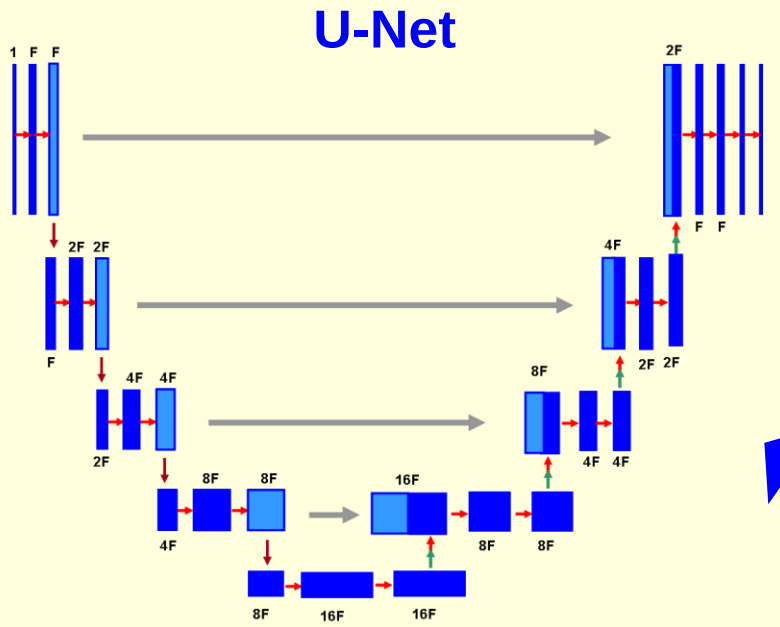
3.0	5.0	7.0
5.0	7.0	2.0
5.0	2.0	2.0

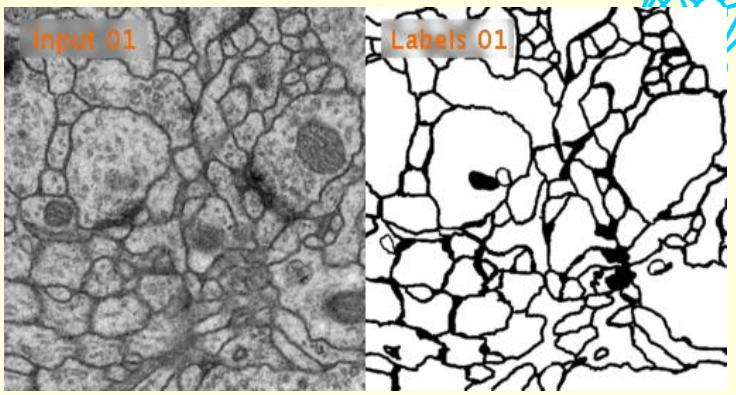
purpose: resize data for convenience of subsequent transformations

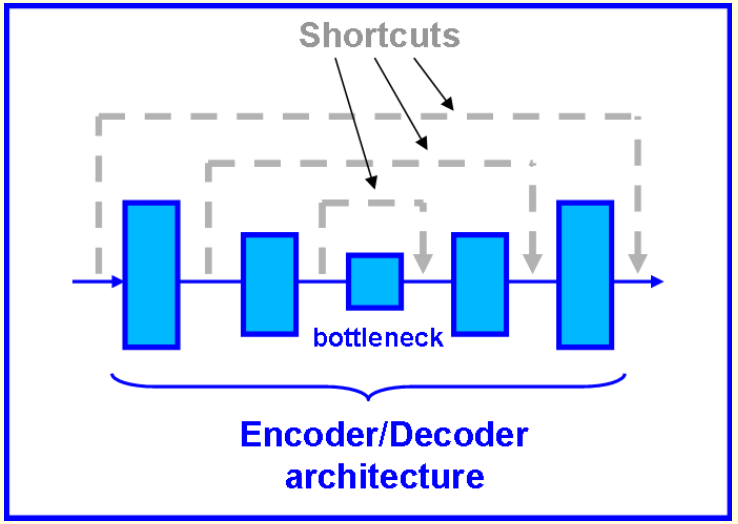
C = concatenate ([A, B], axis = 3)



What makes the U-Net architecture special?







- other popular semantic segmantation models: Segnet, FCN

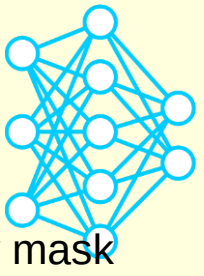
- development of U-Net was inspired by analysis of biomedical images

- the U-path extracts features at multiple scales using Encoder/Decoder-like architecture

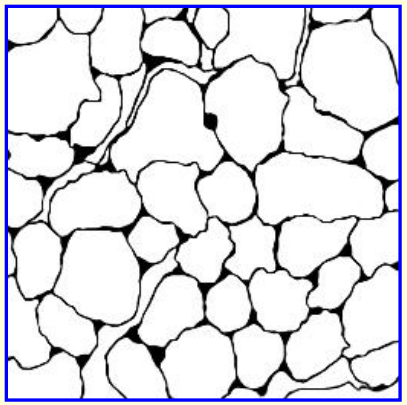
- the shortcuts / concatenation transformations communicate the features that were not transmitted through the bottleneck

Binary Cross-Entropy (BCE): the loss function for binary segmentation

binary crossentropy loss



Binary mask



$$L_{BCE}(p(w)) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p_i(w)) + (1-y_i)\cdot \log(1-p_i(w))$$

N = number of pixels in the binary mask

y_i = the ground truth labels (=0 or 1)

$p_i(w)$ = "predicted labels", given w $(0 \le p_i(w) \le 1)$

$$L_{BCE} = \sum_i L^{(i)}_{BCE} \to \min=0 \Leftrightarrow p_i(w) == y_i \text{ for all } i$$

$$\Leftarrow \frac{\partial L_{BCE}}{\partial p_i} \begin{cases} < 0, \text{ if } p_i < y_i \\ > 0, \text{ if } p_i > y_i \end{cases}$$

$\Downarrow$

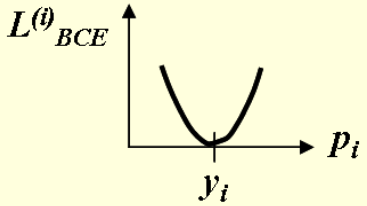
$\Uparrow$

Limiting cases:

1) $p_i = y_i$: $L^{(i)}_{BCE}(w) \to 0\cdot \log(0) + 1\cdot \log(1) = 0$ $\Rightarrow$

2) $y_i = 1, p_i = 0$: $L^{(i)}_{BCE}(w) \sim -y \cdot \log(p) \to +\text{Inf}$

3) $y_i = 0, p_i = 1$: $L^{(i)}_{BCE}(w) \sim -(1-y)\cdot \log(1-p) \to +\text{Inf}$



Conclusion: L_{BCE} as a function of $p = (p_1, \ldots, p_N)$ has a **single global minimum**, and **no local minima**

How to run the U-Net code on Biowulf?

https://hpc.nih.gov/apps/UNet.html

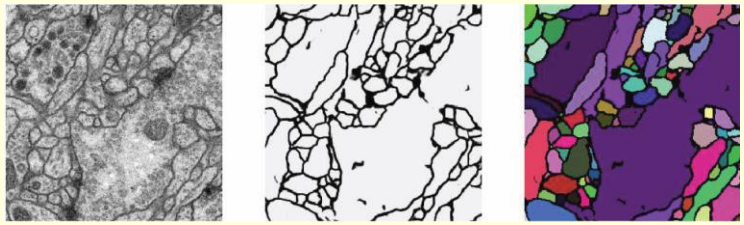
Using a single GPU:

```
denisovga@biowulf:/data/denisovga/1_DL_Course/1_CNNs
ssh -Y biowulf.nih.gov
sinteractive --mem=4g --gres=gpu:v100:1,lscratch:10 -c4
module load unet
cp -r $UNET_DATA/*  .
train.py     -d <data_folder> [ other options ]
predict.py   -d <data_folder> [ other options ]
visualize.py -d <data_folder> -t <target_id> [ other options ]
                                              14,1          All
```

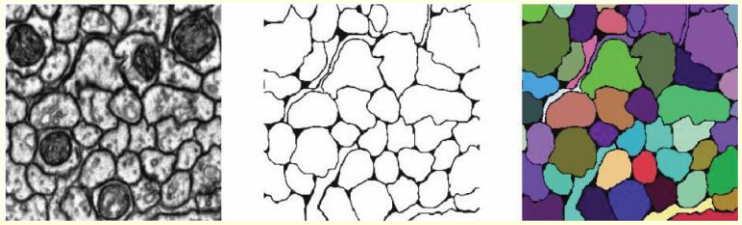
Using 4 GPUs:

```
denisovga@biowulf:/data/denisovga/1_DL_Course/1_CNNs
sinteractive --mem=8g --gres=gpu:v100:4,lscratch:40 -c14
module load unet
cp -r $UNET_DATA/*  .
train.py -d <data_folder> -g 4 [ other options ]
                                          9,1           Top
```

data_isbi/membr



data_hhmi/membr

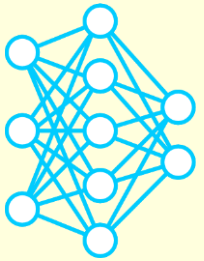


data_hhmi/mito

Available data folders:

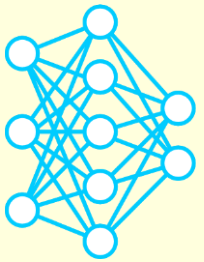
- data_isbi
- data_hhmi

Summary



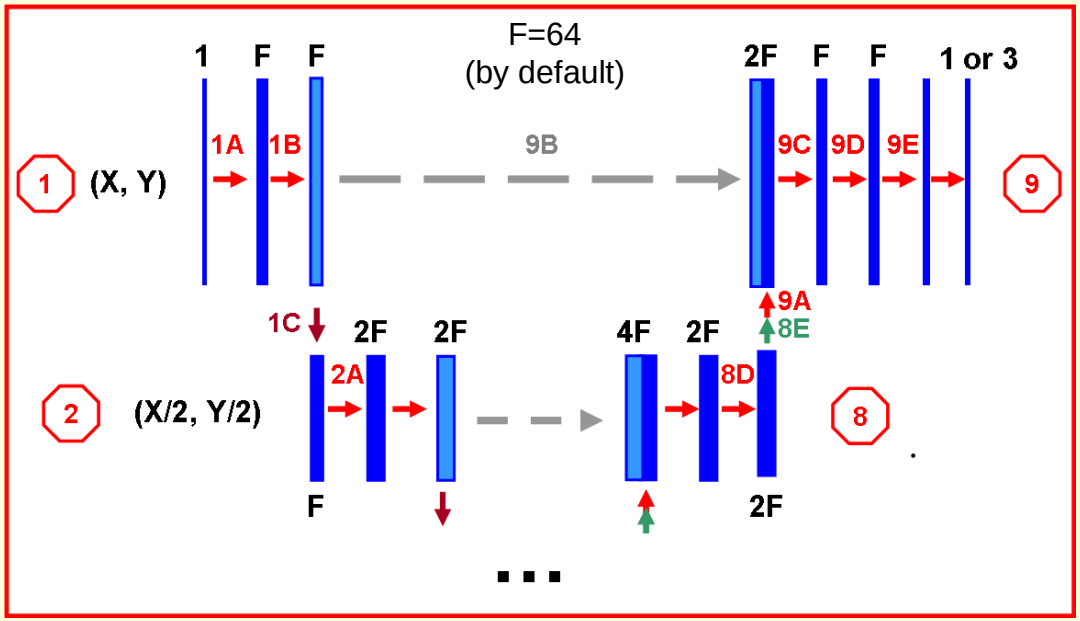
1) Introduction to the DL with Keras using Perceptron as an example.
   Key points:
   - the notion of tensors and layers
   - distinction between parameters and hyperparameters
   - Keras layers: Dense and Activation
   - Functional API and Sequential construct approaches to building models in Keras; branched vs sequential networks
   - the notion of hidden layers and deep network.

2) Semantic segmentation with Convolutional Neural Networks (CNNs).
   Key points:
   - the challenge of biomedical image segmentation: ground truth labels needed for training
   - to avoid over-fitting, one can perform data augmentation
   - in CNNs, adjustable parameters come (primarily) from convolutional layers
   - Keras layers: Conv2D, MaxPooling2D, Upsampling2D and concatenation
   - the loss function for binary segmentation



BACKUP SLIDES

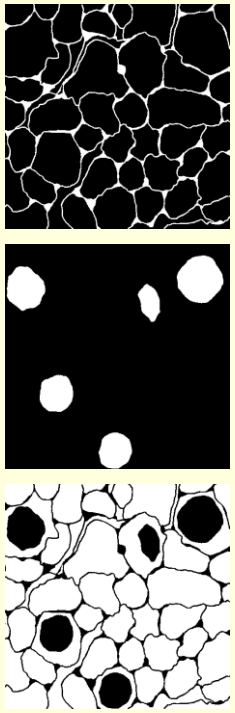
Coding the U-Net model




```
denisovga@biowulf:/data/denisovga/1_DL_Course/1_CNNs
...
inputs   = Input(input_size)
conv_1A = Conv2D(F  ,3, padding='same',...)(inputs)
conv_1B = Conv2D(F  ,3, padding='same',...)(conv_1A)
pool_1C = MaxPooling2D(pool_size=(2, 2))(conv_1B)
conv_2A = Conv2D(F*2,3,padding='same',...)(pool_1C)
..
up_8E    = UpSampling2D(size = (2,2))(conv_8D)
conv_9A = Conv2D(F,2,padding='same',...)(up_8E)
merge_9B= concatenate([conv_1B, conv_9A], axis = 3)
conv_9C = Conv2D(F,3,padding='same',...)(merge_9B)
conv_9D = Conv2D(F,3,padding='same',..)(conv_9C)
conv_9E = Conv2D(2*num_classes,3,padding ='same',...)(conv_9D)
outputs = Conv2D(1,          1, activation='sigmoid')(conv_9E)\
          if num_classes < 3 else \
          Conv2D(num_classes,1, activation='softmax')(conv_9E)
...
                                                17,4          All
```

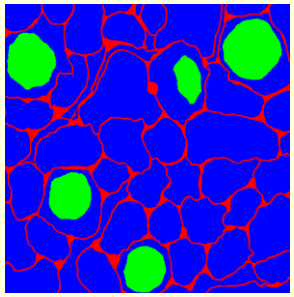
Multi-class semantic segmentation

(weighted) categorical cross-entropy loss, class imbalance



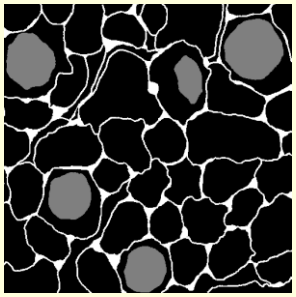
1-channel,
one hot encoded
masks

3-channel,
one hot encoded mask



Augmentation not currently supported by Keras

1-channel,
value-encoded mask



Can be augmented together with grayscale images

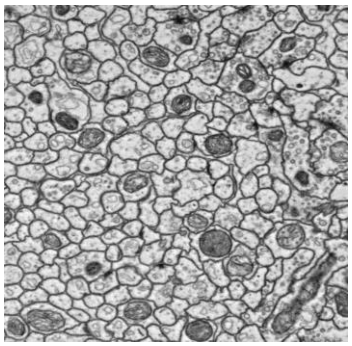
$$L_{WCCE}(p(w)) = -\frac{1}{N}\frac{1}{C}\sum_{i=1}^{N}\sum_{c=1}^{C} w_c \cdot y_{i,c} \cdot log(p_{i,c}(w))$$

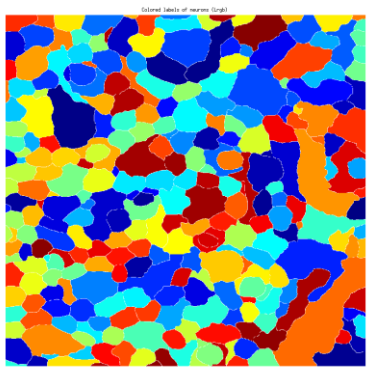
$C = 3$

$$w_c = (1/A_c) \; / \; \sum_{k=1}^{C} (1/A_k)$$

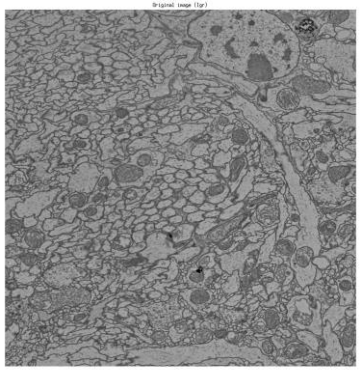
A_c = area (# pixels) occupied by the object(s) of class c

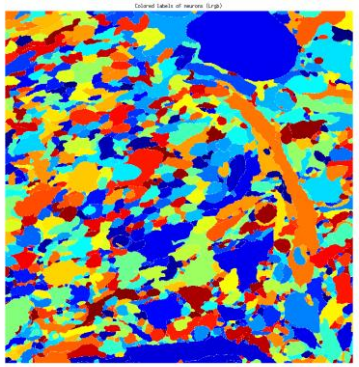
Examples of more complex fly brain image data



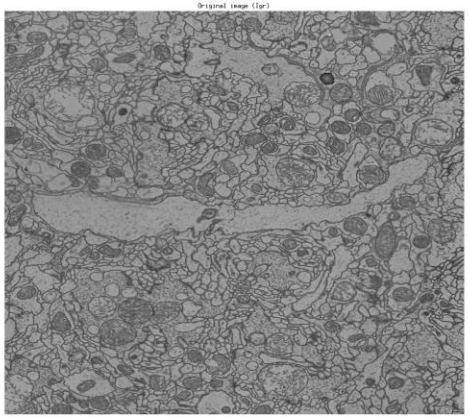


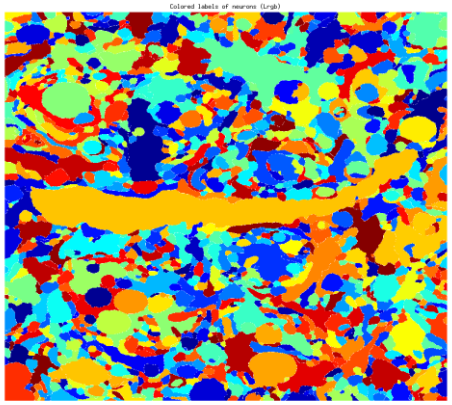
Sample_A (“simple”)





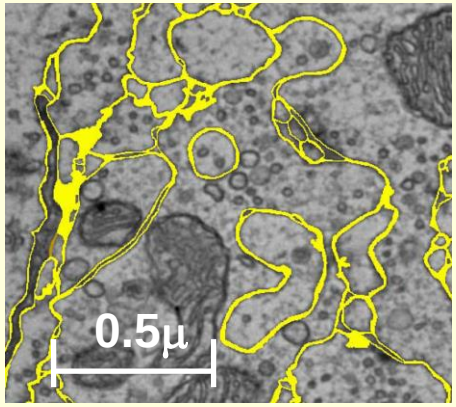
Sample_B (“medium”)



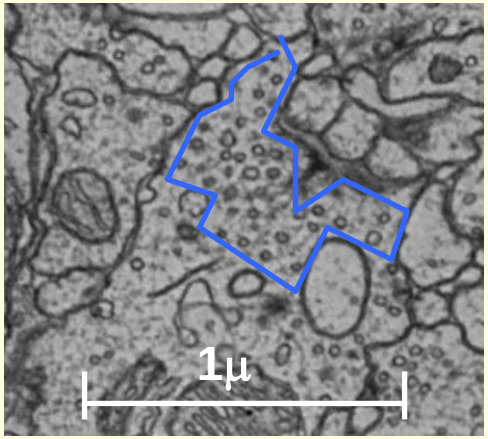


Sample_C (“hard”)

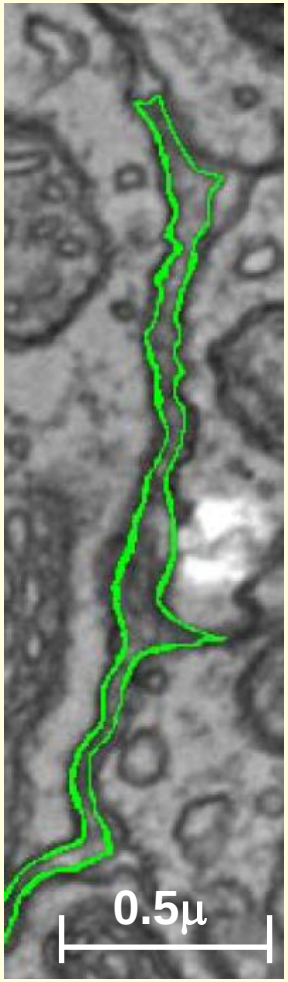
Biological structures of practical interest for automatic detection in Drosophila brain



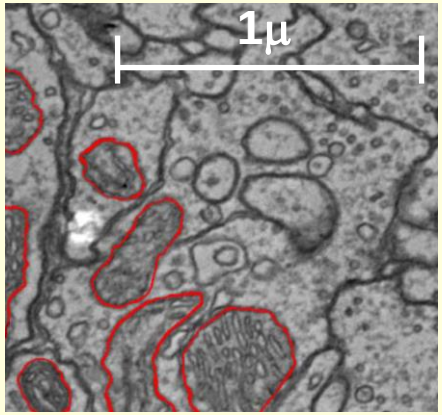
Neural cells



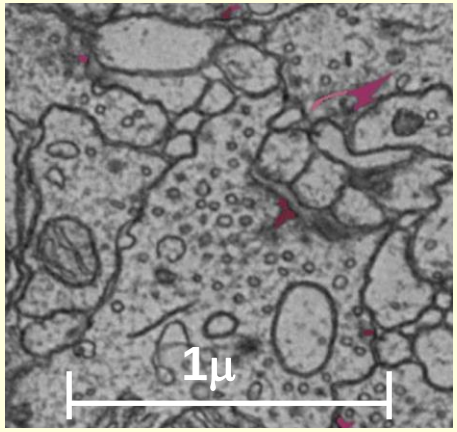
Small vesicles + microtubules



Glial cell



Mitochondria

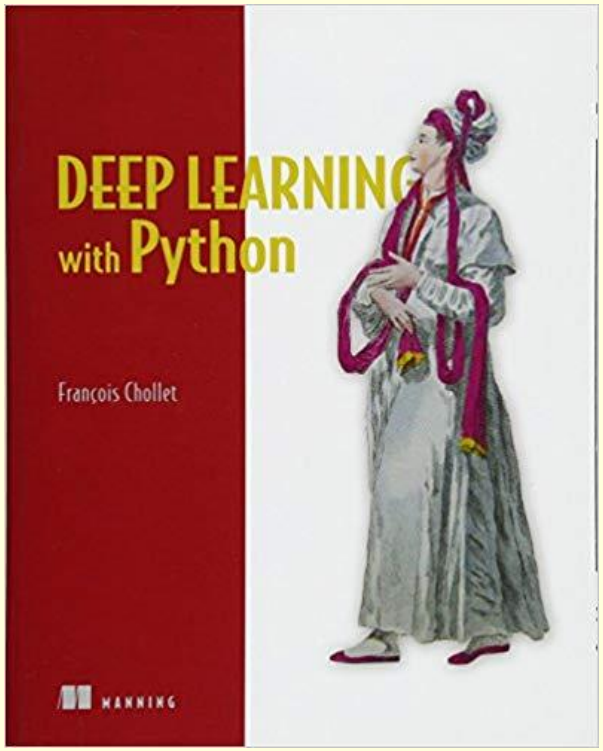


Pre-synaptic structures (T-bars)

Further reading

1. https://github.com/hussius/deeplearning-biology
2. http://keras.io
3. https://keras.io/getting-started/faq

4.



5.