Lecture 13: Deep learning for image analysis

Nils Olovsson
nils.olofsson@igp.uu.se

Introduction to data science and Python programming for medical research
Uppsala University

2025

UPPSALA
UNIVERSITET

Content

Introduction

Introduction

Introduction: Problem

Classical image analysis methods struggle with many tasks.
E.g. image classification and image segmentation.

Want to use neural networks to get better performing methods.

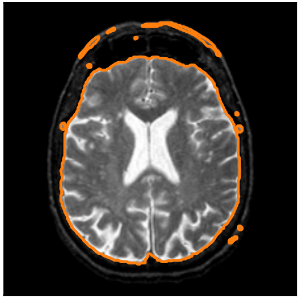


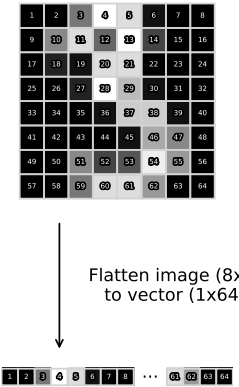
Image segmentation methods rarely just work and might require both pre and post processing.

Introduction: Flatten image

Could process images by flattening them.

We did this when we performed image analysis using PCA.

Lose spatial coherence of the pixels!

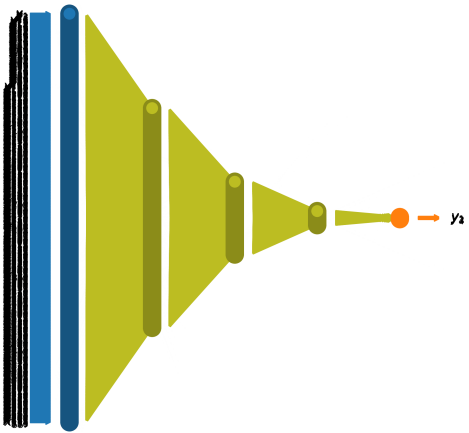


Introduction: Fully connected network

A flattened image could then be analyzed with fully connected, linear, layers.

Network can become very large even for small images.

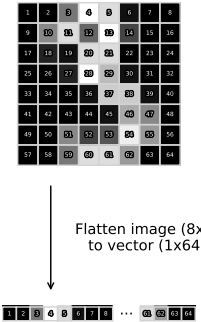
A 256×256 input layer connected to a layer with an equal number of outputs would have 256^4 parameters. This correspond to 17 GB! Just for a single layer of weights!

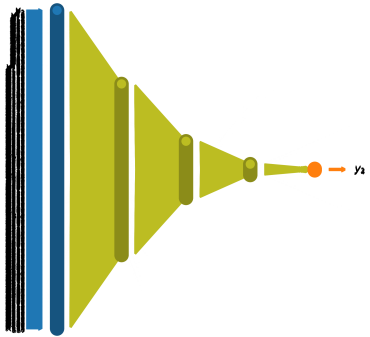


Introduction: Large and poor generalization

Lose information by flattening.
The network becomes very large and tends to generalize poorly.

Need alternative way of handling this.





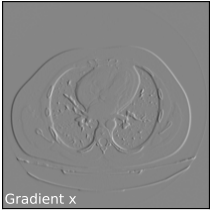
Convolutional neural networks

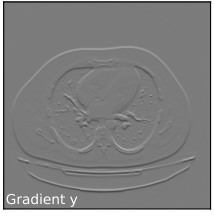
Convolutional neural networks: Convolution

∇_x (Gradient in x): $k = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

∇_y (Gradient in y): $k = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

∇^2 (Laplacian): $k = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

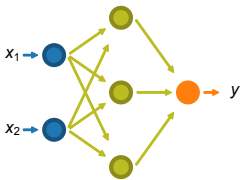


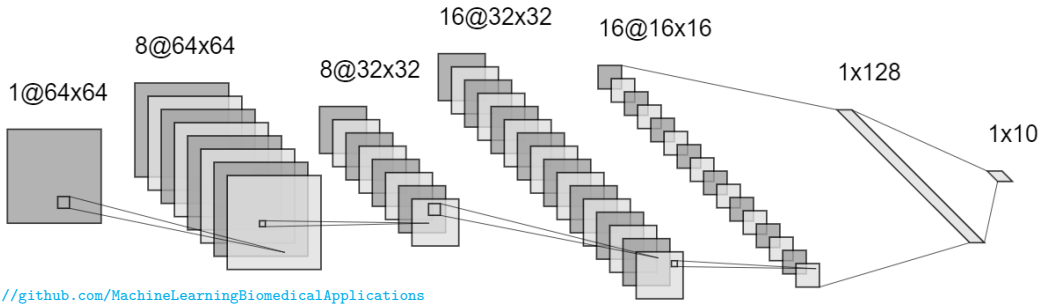


Convolutional neural networks: Deep learning

Instead of using linear layers, each layer represents a set of convolution kernels.

The values of the kernels are the parameters (weights) optimized during training.





https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Convolutional neural networks: Convolution operation

$= 1\times55 + 0\times60 + -1\times65 + 2\times67 + 0\times65 - 2\times63 + 1\times73 + 0\times64 - 1\times62$

1	0	-1
2	0	-2
1	0	-1

$*$

55	60	65	68	60	60	73	87
67	65	63	67	70	73	103	126
73	64	62	65	80	97	145	169
69	65	68	83	133	165	216	227
71	79	89	108	165	195	234	242
68	89	105	131	198	220	237	240
67	110	146	177	233	245	238	238
80	129	172	206	241	246	234	233

$=$

9					

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Convolutional neural networks: Convolution operation

$= 1\times60 + 0\times65 + -1\times68 + 2\times65 + 0\times63 \ -2\times67 + 1\times64 + 0\times62 \ -1\times65$

1	0	-1
2	0	-2
1	0	-1

$*$

55	60	65	68	60	60	73	87
67	65	63	67	70	73	103	126
73	64	62	65	80	97	145	169
69	65	68	83	133	165	216	227
71	79	89	108	165	195	234	242
68	89	105	131	198	220	237	240
67	110	146	177	233	245	238	238
80	129	172	206	241	246	234	233

$=$

9	-13				

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Convolutional neural networks: Convolution operation

$= 1\times220 + 0\times237 + -1\times240 + 2\times245 + 0\times238 - 2\times238 + 1\times246 + 0\times234 - 1\times233$

1	0	-1
2	0	-2
1	0	-1

$*$

55	60	65	68	60	60	73	87
67	65	63	67	70	73	103	126
73	64	62	65	80	97	145	169
69	65	68	83	133	165	216	227
71	79	89	108	165	195	234	242
68	89	105	131	198	220	237	240
67	110	146	177	233	245	238	238
80	129	172	206	241	246	234	233

9	-13	-27	-36	-144	-205
27	-22	-108	-152	-246	-259
-5	-66	-224	-283	-300	243
-72	-118	-310	-345	-260	176
-171	-180	-349	-333	-152	-80
-287	-253	-336	-265	-42	7

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Convolutional neural networks: Convolution operation

1	0	-1
2	0	-2
1	0	-1

$*$

0	0	0	0	0	0	0	0	0	0
0	55	60	65	68	60	60	73	87	0
0	67	65	63	67	70	73	103	126	0
0	73	64	62	65	80	97	145	169	0
0	69	65	68	83	133	165	216	227	0
0	71	79	89	108	165	195	234	242	0
0	68	89	105	131	198	220	237	240	0
0	67	110	146	177	233	245	238	238	0
0	80	129	172	206	241	246	234	233	0
0	0	0	0	0	0	0	0	0	0

$=$

-185	-16	-18	3	10	-59	-107	249
-254	9	-13	-27	-36	-144	-205	424
-258	27	-22	-108	-152	-246	-259	609
-273	-5	-66	-224	-283	-300	243	811
-312	-72	-118	-310	-345	-260	176	921
-367	-171	-180	-349	-333	-152	-80	946
-438	-287	-253	-336	-265	-42	7	947
-368	-263	-221	-225	-148	9	33	706

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Convolutional neural networks: Convolutions in pytorch

Convolution Layers

`nn.Conv1d`	Applies a 1D convolution over an input signal composed of several input planes.
`nn.Conv2d`	Applies a 2D convolution over an input signal composed of several input planes.
`nn.Conv3d`	Applies a 3D convolution over an input signal composed of several input planes.
`nn.ConvTranspose1d`	Applies a 1D transposed convolution operator over an input image composed of several input planes.
`nn.ConvTranspose2d`	Applies a 2D transposed convolution operator over an input image composed of several input planes.
`nn.ConvTranspose3d`	Applies a 3D transposed convolution operator over an input image composed of several input planes.
`nn.LazyConv1d`	A `torch.nn.Conv1d` module with lazy initialization of the `in_channels` argument.
`nn.LazyConv2d`	A `torch.nn.Conv2d` module with lazy initialization of the `in_channels` argument.

https://pytorch.org/docs/stable/nn.html#convolution-layers

Convolutional neural networks: Pooling (Downsampling)

Pooling layers

`nn.MaxPool1d`	Applies a 1D max pooling over an input signal composed of several input planes.
`nn.MaxPool2d`	Applies a 2D max pooling over an input signal composed of several input planes.
`nn.MaxPool3d`	Applies a 3D max pooling over an input signal composed of several input planes.
`nn.MaxUnpool1d`	Computes a partial inverse of `MaxPool1d`.
`nn.MaxUnpool2d`	Computes a partial inverse of `MaxPool2d`.
`nn.MaxUnpool3d`	Computes a partial inverse of `MaxPool3d`.
`nn.AvgPool1d`	Applies a 1D average pooling over an input signal composed of several input planes.
`nn.AvgPool2d`	Applies a 2D average pooling over an input signal composed of several input planes.
`nn.AvgPool3d`	Applies a 3D average pooling over an input signal composed of several input planes.

https://pytorch.org/docs/stable/nn.html#pooling-layers

Convolutional neural networks: Upsampling

Vision Layers

`nn.PixelShuffle`	Rearrange elements in a tensor according to an upscaling factor.
`nn.PixelUnshuffle`	Reverse the PixelShuffle operation.
`nn.Upsample`	Upsamples a given multi-channel 1D (temporal), 2D (spatial) or 3D (volumetric) data.
`nn.UpsamplingNearest2d`	Applies a 2D nearest neighbor upsampling to an input signal composed of several input channels.
`nn.UpsamplingBilinear2d`	Applies a 2D bilinear upsampling to an input signal composed of several input channels.

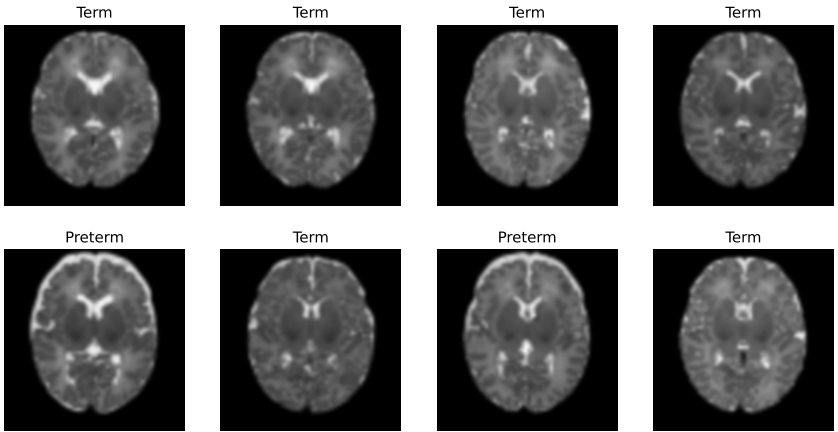
https://pytorch.org/docs/stable/nn.html#vision-layers

Classification

Classification

Classification: Binary image classification

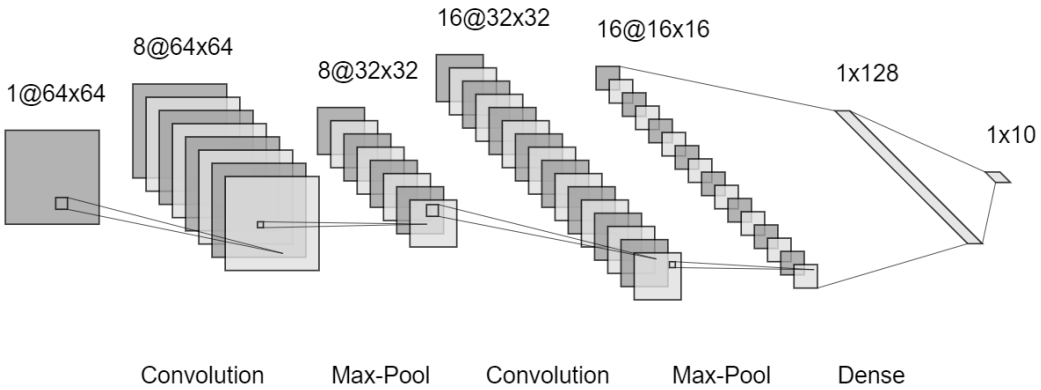
Classifying images according to being preterm or not.



https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Classification: CNN network architecture

Similar to this example but with only two outputs in the final layer.



https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Classification: Python

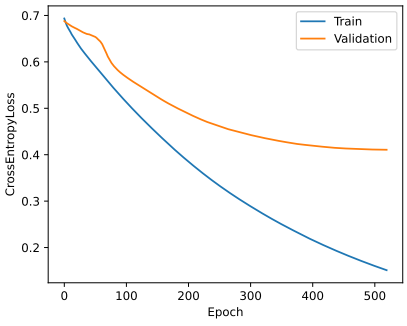
```python
class PredictPrematurityCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(1, 8, 3, padding=1),
            nn.BatchNorm2d(8),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.conv_block2 = nn.Sequential(
            nn.Conv2d(8, 16, 3, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.conv_block3 = nn.Sequential(
            nn.Conv2d(16, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc_block = nn.Sequential(
            nn.Linear(32 * 8 * 8, 32),
            nn.ReLU(),
            nn.Linear(32, 2),
            nn.ReLU()
        )
        return
```

Use `nn.Sequential` to create blocks of operations.

```python
    def forward(
        self,
        X: torch.Tensor,
    ) -> torch.Tensor:
        X = self.conv_block1(X)
        X = self.conv_block2(X)
        X = self.conv_block3(X)
        X = X.view(-1, 32 * 8 * 8)
        X = self.fc_block(X)
        return X
```

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Classification: Results




```
>> Device: cpu
>> Train accuracy: 0.95
>>               precision    recall  f1-score   support
>>
>>            0       0.94      0.97      0.96        66
>>            1       0.95      0.91      0.93        46
>>
>>     accuracy                           0.95       112
>>    macro avg       0.95      0.94      0.94       112
>> weighted avg       0.95      0.95      0.95       112
>>
>> Test accuracy: 1.0
>>               precision    recall  f1-score   support
>>
>>            0       1.00      1.00      1.00        19
>>            1       1.00      1.00      1.00        13
>>
>>     accuracy                           1.00        32
>>    macro avg       1.00      1.00      1.00        32
>> weighted avg       1.00      1.00      1.00        32
>>
```

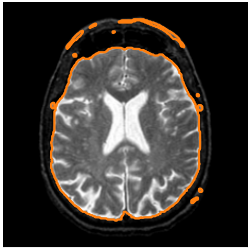
Segmentation

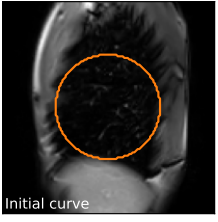
Segmentation

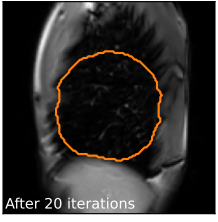
Segmentation: Classical methods are limited

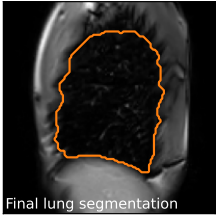
We saw that methods based on intensity threshold and active contours can have trouble producing satisfying image segmentations except for occasionally simple cases.

Results can be over or under segmented and methods can be sensitive to regions with poor contrast.







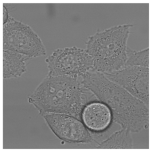


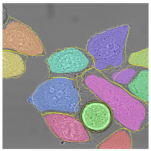
Segmentation: Unet

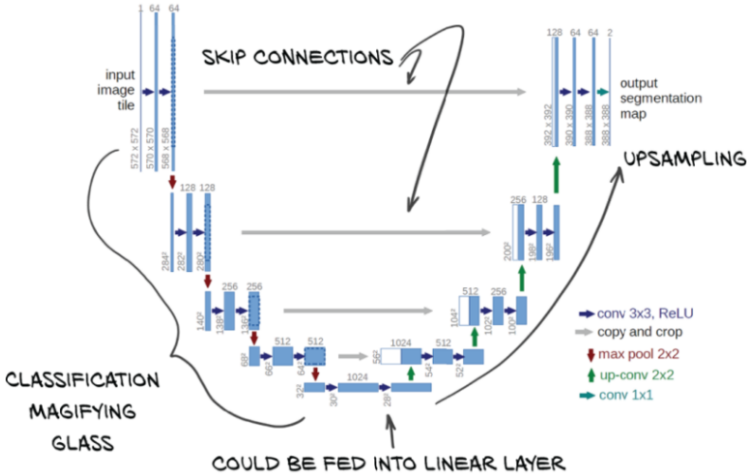
Unet architecture introduced for the segmentation of cells.

Has since proven very successful for other types of images as well.

Skip connections help parts of the network see both a bigger picture and fine details at the same time.



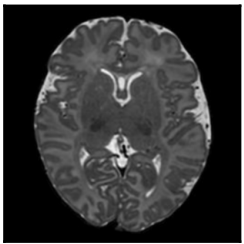




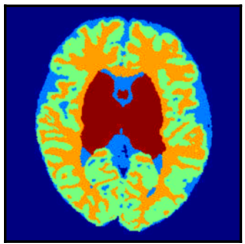
Ronneberger et al. 2015
http://arxiv.org/abs/1505.04597

Segmentation: Example problem

Magnetic resonance brain images of newborn/neonatal babies.
Paired with a labeled mask.
One label correspond to the cortex which will be used to train a segmentation deep learning model.



MR image



Labels

Cortex

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

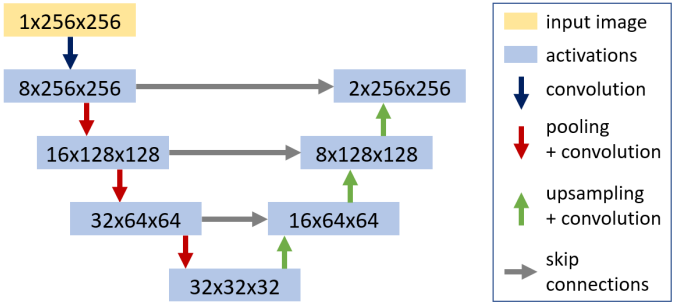
Segmentation: Unet design

Slightly simpler Unet design.

Input is a single channel 256×256 pixel image.

Output is a two channel 256×256 pixel image mask.

The two channels represent *probability* of pixel being either background or brain cortex.



https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

Segmentation: Unet implementation

```
class UNet(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1_down = nn.Conv2d(1, 8, 3, padding=1)
        self.conv2_down = nn.Conv2d(8, 16, 3, padding=1)
        self.conv3_down = nn.Conv2d(16, 32, 3, padding=1)
        self.conv4 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3_up = nn.Conv2d(32+32, 16, 3, padding=1)
        self.conv2_up = nn.Conv2d(16+16, 8, 3, padding=1)
        self.conv1_up = nn.Conv2d(8+8, 2, 3, padding=1)

        self.down = nn.AvgPool2d(kernel_size=2)
        self.up = nn.UpsamplingBilinear2d(scale_factor=2)

        self.relu = nn.ReLU()
        return
```

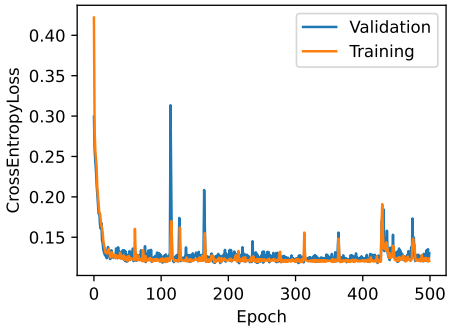
```
    def forward(
        self,
        X: torch.Tensor,
    ) -> torch.Tensor:
        # encoder
        enc1 = self.relu(self.conv1_down(X))
        enc2 = self.down(enc1)
        enc2 = self.relu(self.conv2_down(enc2))
        enc3 = self.down(enc2)
        enc3 = self.relu(self.conv3_down(enc3))
        enc4 = self.down(enc3)
        enc4 = self.relu(self.conv4(enc4))

        # decoder
        dec3 = self.up(enc4)
        dec3 = torch.cat([dec3, enc3], dim=1)
        dec3 = self.relu(self.conv3_up(dec3))
        dec2 = self.up(dec3)
        dec2 = torch.cat([dec2, enc2], dim=1)
        dec2 = self.relu(self.conv2_up(dec2))
        dec1 = self.up(dec2)
        dec1 = torch.cat([dec1, enc1], dim=1)
        dec1 = self.relu(self.conv1_up(dec1))

        return dec1
```

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

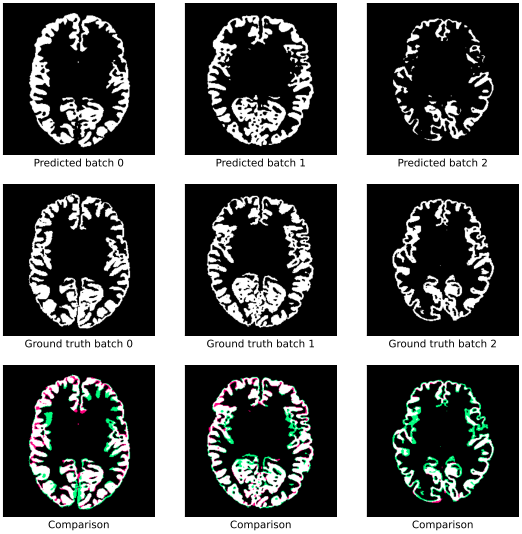
Segmentation: Unet training

```
net = UNet()
net.to(device)
loss_func = nn.CrossEntropyLoss().to(device)
optimizer = optim.Adam(net.parameters(),
                       lr=0.01, weight_decay=0.01)
epochs = 500
train_loss = []
val_loss = []
for epoch in range(epochs):
    break
    ## Training
    net.train()
    mean_ce = 0.0
    n = 0
    for i, (image, mask) in enumerate(
     data_train_loader):
        image = image.to(device)
        mask = mask.to(device)
        optimizer.zero_grad()
        pred = net(image)
        ce = loss_func(pred, mask)
        ce.backward()
        optimizer.step()
        mean_ce += ce.item()
        n += 1
    mean_ce /= n
    train_loss.append(mean_ce)
```

Challenging to train!



Segmentation: Results



Network gives results that are similar to the expected results.
(But far from perfect. Network performance very inconsistent between different training sessions.)

```
>> Device: cpu
>> ## Test set scores:
>> precision score: 0.92
>> recall score:    0.72
>> f1 score:        0.8
```

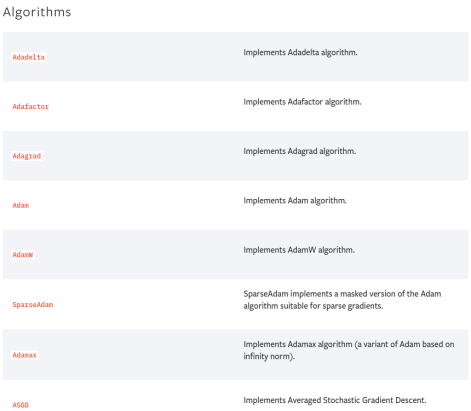
Optimization

Optimization

Optimization: Many different methods

Popular ones:

- Stochastic gradient descent (SGD)
- ADAM



Algorithms

Adadelta	Implements Adadelta algorithm.
Adafactor	Implements Adafactor algorithm.
Adagrad	Implements Adagrad algorithm.
Adam	Implements Adam algorithm.
AdamW	Implements AdamW algorithm.
SparseAdam	SparseAdam implements a masked version of the Adam algorithm suitable for sparse gradients.
Adamax	Implements Adamax algorithm (a variant of Adam based on infinity norm).
ASGD	Implements Averaged Stochastic Gradient Descent.

https://pytorch.org/docs/stable/optim.html
https://pytorch.org/docs/stable/optim.html#algorithms

Optimization: Learning rate

Learning rate determines how large the steps taken in the direction of the gradient descent are each iteration.

There are ways to make this adaptive or changing during the course of training.
E.g. start by taking larger steps and in the end take smaller steps.

How to adjust learning rate

`torch.optim.lr_scheduler.LRScheduler` provides several methods to adjust the learning rate based on the number of epochs.
`torch.optim.lr_scheduler.ReduceLROnPlateau` allows dynamic learning rate reducing based on some validation measurements.

Learning rate scheduling should be applied after optimizer's update; e.g., you should write your code this way:

Example:

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
scheduler = ExponentialLR(optimizer, gamma=0.9)

for epoch in range(20):
    for input, target in dataset:
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()
    scheduler.step()
```

https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate

Regularization

Regularization

Regularization: Improve training and robustness

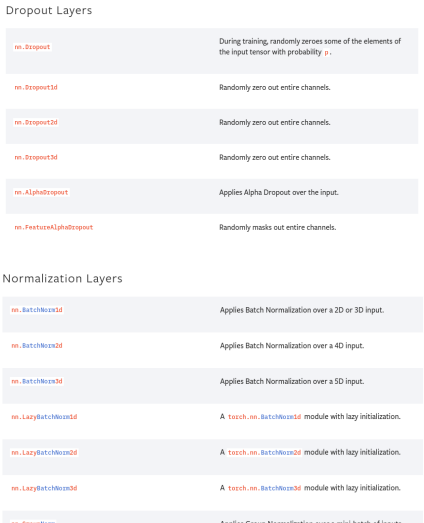
Problem with models being not robust.

Problem with overfitting.

Want to regularize the model!

- L1 and L2 regularization (Ridge and lasso)
- Batch normalization
- Dropout
- Weight decay

https://pytorch.org/docs/stable/nn.html#dropout-layers



Dropout Layers

nn.Dropout	During training, randomly zeroes some of the elements of the input tensor with probability p.
nn.Dropout1d	Randomly zero out entire channels.
nn.Dropout2d	Randomly zero out entire channels.
nn.Dropout3d	Randomly zero out entire channels.
nn.AlphaDropout	Applies Alpha Dropout over the input.
nn.FeatureAlphaDropout	Randomly masks out entire channels.

Normalization Layers

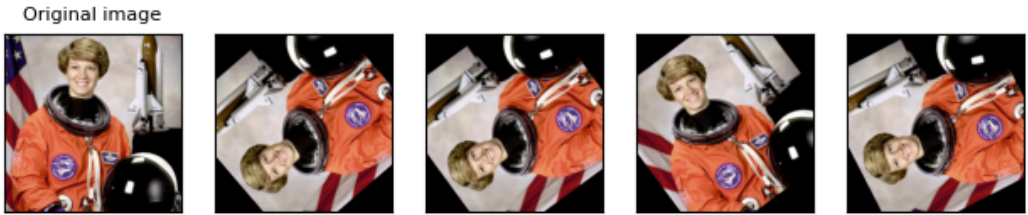
nn.BatchNorm1d	Applies Batch Normalization over a 2D or 3D input.
nn.BatchNorm2d	Applies Batch Normalization over a 4D input.
nn.BatchNorm3d	Applies Batch Normalization over a 5D input.
nn.LazyBatchNorm1d	A torch.nn.BatchNorm1d module with lazy initialization.
nn.LazyBatchNorm2d	A torch.nn.BatchNorm2d module with lazy initialization.
nn.LazyBatchNorm3d	A torch.nn.BatchNorm3d module with lazy initialization.

Data augmentation

Data augmentation

Data augmentation: Geometrical

Geometrical transforms like cropping, translations, rotations and deformations can be applied to the images used to train the model.

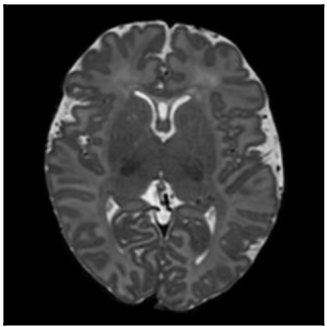


https://pytorch.org/vision/0.11/transforms.html#transforms
https://pytorch.org/vision/0.11/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

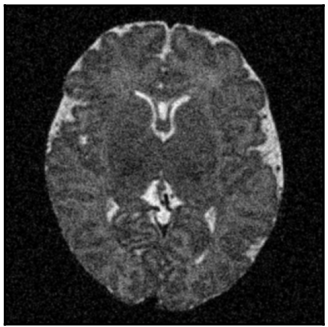
Data augmentation: Signal noise

Noise can be added to images.

The type of noise should match the type of noise you would expect to see in the images.



Image



Noisy

Data loading

Data loading

Data loading: Custom class

```python
class NeonatalDataset(Dataset):

    def __init__(
        self,
        set_type: str = 'train',
        directory: str = os.path.join('data', 'mridata'),
    ):
        self.directory = directory
        n_train = 320
        n_val = 32
        n_test = 44
        if set_type == 'train':
            self.start_idx = 0
            self.len = n_train
        elif set_type == 'val':
            self.start_idx = n_train
            self.len = n_val
        elif set_type == 'test':
            self.start_idx = n_train + n_val
            self.len = n_test
        return

    def __len__(self) -> int:
        return self.len
```

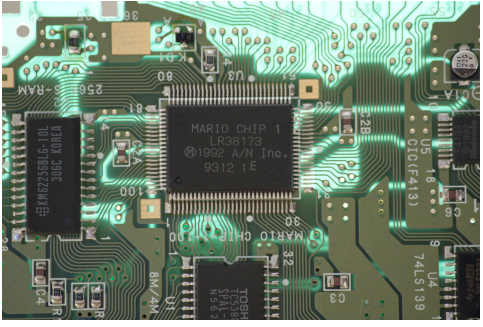
```python
    def __getitem__(
        self,
        idx: int,
    ):
    #) -> List[torch.Tensor, torch.Tensor]:
        ## Load the MR image
        image_filename = f'{idx}_t2w.jpg'
        image_filepath = os.path.join(self.directory, image_filename)
        image = io.imread(image_filepath, as_gray=True)
        image = image / 255.0

        ## Load the mask labeled image
        labels_filename = f'{idx}_lab.jpg'
        labels_filepath = os.path.join(self.directory, labels_filename)
        labels = io.imread(labels_filepath, as_gray=True)
        cortex_mask = np.logical_and(labels>=100, labels<=150)

        ## Convert to Tensors
        image = torch.tensor(image).float().unsqueeze(0)
        cortex_mask = torch.tensor(cortex_mask).long()

        data = [image, cortex_mask]
        return data
```

https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

GPU

GPU

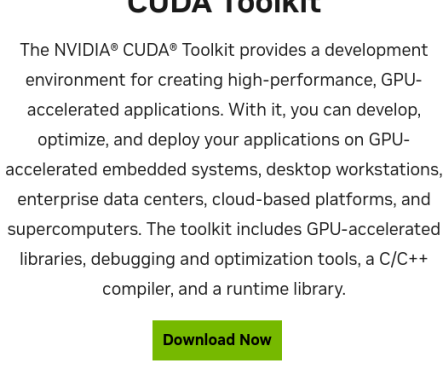
GPU: Graphics processing unit



https://en.wikipedia.org/wiki/Super_FX

GPU: CUDA

CUDA (Compute Unified Device Architecture) is a proprietary software for general purpose computing on GPU.

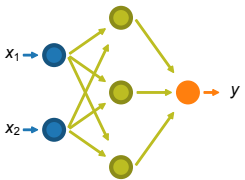


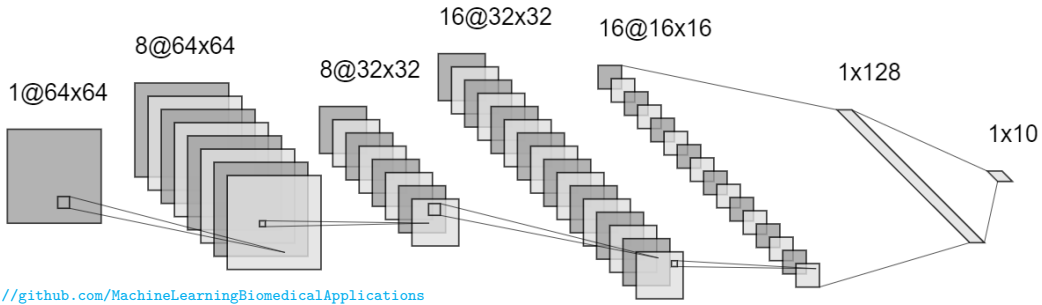
https://developer.nvidia.com/cuda-toolkit
https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units

GPU: Deep learning

Many computations performed in neural networks are independent.

This means they are trivial to execute in parallel.





https://github.com/MachineLearningBiomedicalApplications
https://github.com/MachineLearningBiomedicalApplications/notebooks/tree/main/Chapter%2011%20-%20Convolutional%20networks

GPU: Pytorch device

```python
import torch
gpuid = 0
device = torch.device(f'cuda:{gpuid}')

msg = ''
msg += f"Execution device: {device}\n"
msg += f"Pytorch version:  {torch.__version__}\n"
msg += f"Cuda available:   {torch.cuda.is_available()}\n"
msg += f"Cuda version:     {torch.version.cuda}\n"
msg += f"Cuda device:      {torch.cuda.get_device_name(gpuid)
    }\n"

with open('torch-cuda-device.txt', 'w') as f:
    f.write(msg)
```

```
Execution device: cuda:0
Pytorch version:  2.6.0+cu126
Cuda available:   True
Cuda version:     12.6
Cuda device:      Quadro RTX 6000
```

Have to check to see if there is a GPU device available.

GPU: Python example

```
191 device = torch.device('cpu')
192 if torch.cuda.is_available():
193     device = torch.device('cuda')
194 print(f"Device: {device}")
195
196 net = UNet()
197 net.to(device)
198 loss_func = nn.CrossEntropyLoss().to(device)
199 optimizer = optim.Adam(net.parameters(),
200                        lr=0.01, weight_decay=0.01)
201 epochs = 500
202 train_loss = []
203 val_loss = []
204 for epoch in range(epochs):
205     break
206     ## Training
207     net.train()
208     mean_ce = 0.0
209     n = 0
210     for i, (image, mask) in enumerate(data_train_loader):
211         image = image.to(device)
212         mask = mask.to(device)
213         optimizer.zero_grad()
214         pred = net(image)
215         ce = loss_func(pred, mask)
216         ce.backward()
217         optimizer.step()
218         mean_ce += ce.item()
219         n += 1
220     mean_ce /= n
221     train_loss.append(mean_ce)
```

Need to send model, loss function and tensors to the GPU device.

In later stages need to send tensors back to cpu.
E.g. in order to convert to numpy arrays.

Conclusions

Conclusions

Conclusions: Image analysis

With deep learning and convolutional neural networks we can solve a lot of image analysis tasks.