ML.init(0): Dive into PyTorch

A Beginner's Guide to AI & ML Exploration!

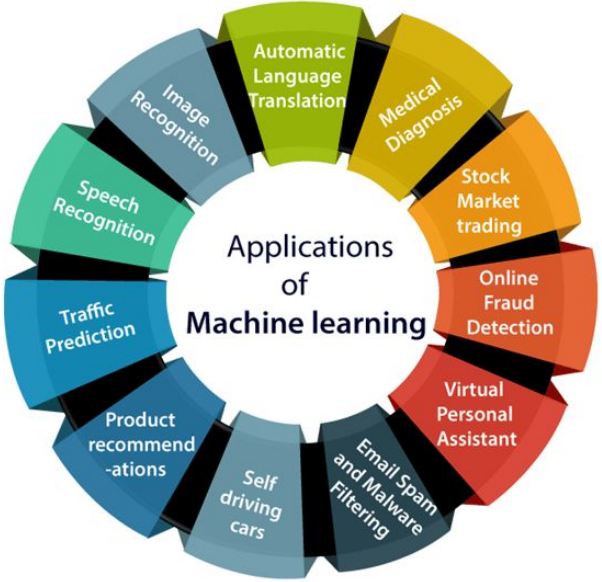
Mathilde Verlyck, Max Dang Vu, Edward Ferdian

Objectives

- Set up a machine learning project in PyTorch
- Train and test a machine learning (ML) model
- Run inferences using the trained model

What is PyTorch?

- Open-source deep learning library for Python
- Mainly used by data scientists for R&D
- Tensor computation, great for GPU use
- **Important modules:** autograd, optim, nn



How does it compare with TensorFlow?

PyTorch	TensorFlow
• **Pythonic implementation**; APIs easier to write, understand and debug • **Flexible** for experimentation, prototyping and data visualisation • **Rapidly expanding** community	• **Mature, extensive** documentation with a large, established community • Optimised for **production deployment** • Compatible with **existing** frameworks and industry standards

AI research project workflow

1. Pre-process & partition **data**

2. Create or use an existing ML **model**

3. Select **loss function** to compute error

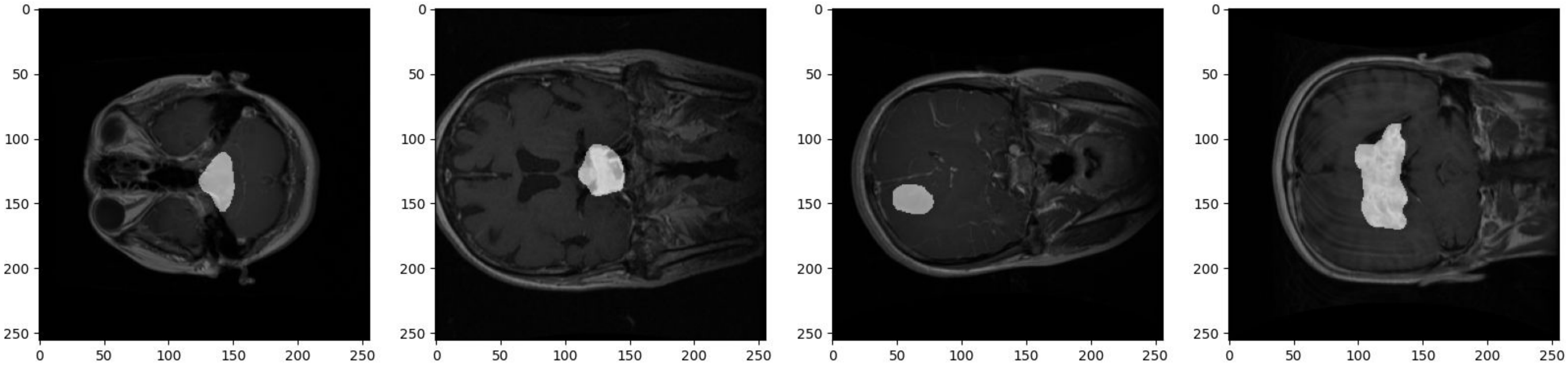
imizer to
el learning

5. **Train** ML model

6. Make model **inferences**

← *Scan here for code*

Example: Brain Tumor Segmentation



- 3064 T1 MRI slices from 233 patients
- Three brain tumor types:
  - Meningioma
  - Glioma
  - Pituitary tumor

Cheng (2017). Brain tumor dataset. figshare. Dataset.
https://doi.org/10.6084/m9.figshare.1512427.v5

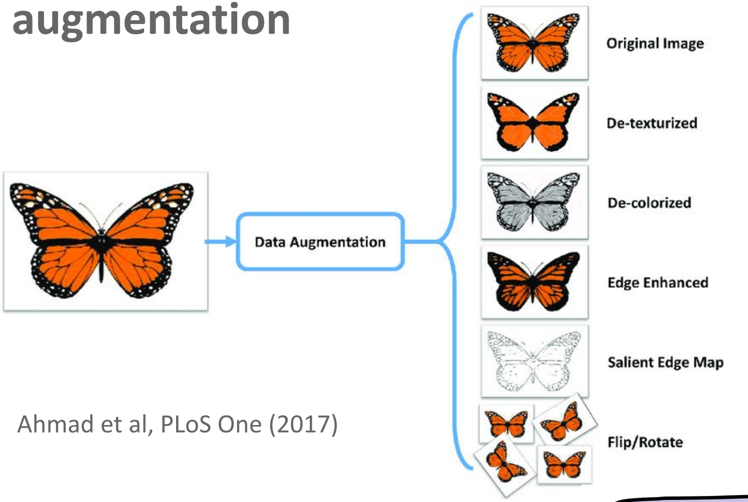
Important necessary libraries
Data -> Get Dataset
Data -> Data visualization

1. Data

Pre-process data using the **Transform** class

- Numpy arrays ⟶ Tensor (*torch.from_numpy*)
  - Tensor designed to work with GPU acceleration and backpropagation
- Resize image

Can also enlarge dataset by **data augmentation**



Ahmad et al, PLoS One (2017)

Data -> Dataset class
Data -> Transform class

1. Data

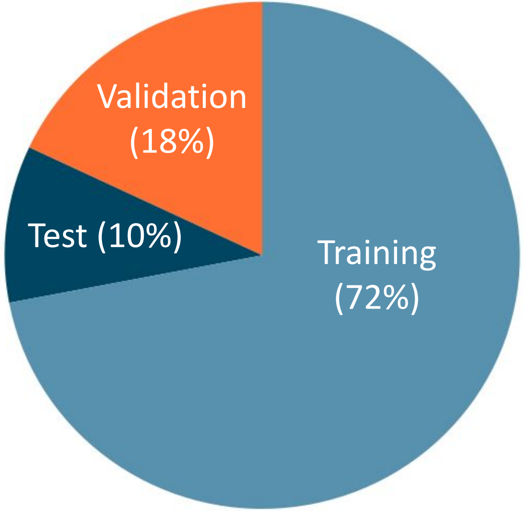
```python
class Resize(object):
    """Resize the image and mask.

    Args:
        output_size (tuple or int): Desired output size. If tuple, output is
            matched to output_size. If int, smaller of image edges is matched
            to output_size keeping aspect ratio the same.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        image = sample['image']
        mask = sample['mask']

        h, w = image.shape[:2]
        if isinstance(self.output_size, int):
            if h > w:
                new_h, new_w = self.output_size * h / w, self.output_size
            else:
                new_h, new_w = self.output_size, self.output_size * w / h
        else:
            new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        image = transform.resize(image, (new_h, new_w)) * 255.0
        image = np.stack((image,) * 3, axis=-1)
        mask = transform.resize(mask, (new_h, new_w)) * 255.0
        mask = np.expand_dims(mask,axis=-1)

        sample['image'] = image
        sample['mask'] = mask

        return sample
```

```python
aset):
"""

ot_dir, transform=None):

ing): Directory with all the images.
llable, optional): Optional transform to be applied
le.

oot_dir
listdir(self.root_dir)
transform

mages)

 idx):
r(idx):
ist()

h.join(self.root_dir,self.imag
filename,'r').get('cjdata')
'image')[()]
tumorMask')[()]

: image, 'mask': mask}

:
.transform(sample)
```

```python
class ToTensor(object):
    """Convert ndarrays to Tensors."""

    def __call__(self, sample):
        # swap color axis because
        # numpy image: H x W x C
        # torch image: C x H x W
        image = sample['image'].transpose((2, 0, 1))
        mask = sample['mask'].transpose((2, 0, 1))
        sample['image'] = torch.from_numpy(image)
        sample['mask'] = torch.from_numpy(mask)

        return sample
```

1. Data

Split dataset and **save** to respective folders



Load data with DataLoader

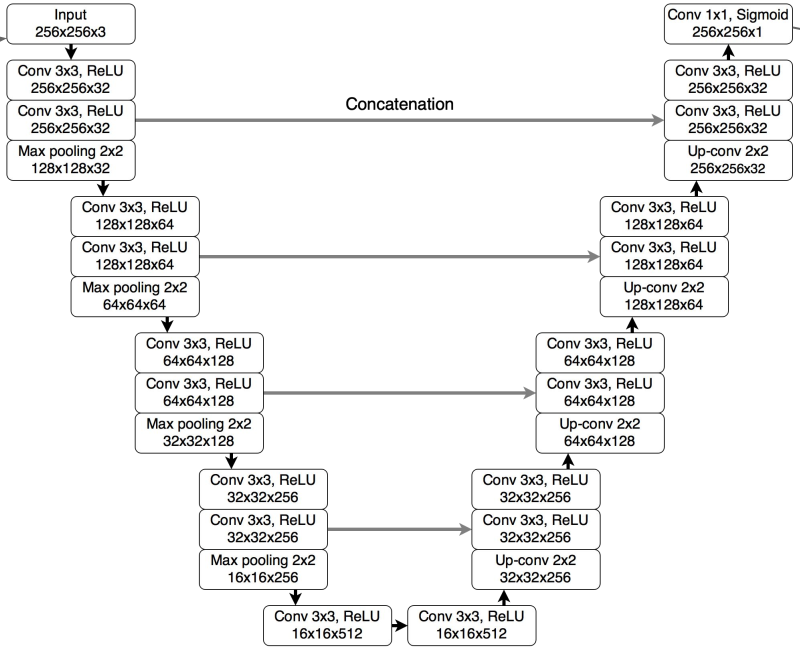
- *Batching:* Optimises memory usage
- *Shuffling:* Avoid learning patterns
- *Parallel loading:* Speeds up training

```
# Training data
train_dataloader = DataLoader(
    train_dataset,batch_size= 4, num_workers=2, shuffle=True)

# Test data
test_dataloader = DataLoader(
    test_dataset,batch_size= 4, num_workers=2, shuffle=False)

# Validation data
val_dataloader = DataLoader(
    val_dataset,batch_size= 4, num_workers=2, shuffle=False)
```

Data -> Split Data and create Data sets
Data -> Data Loders

2. Model

Three-channel MRI slice



Single-channel probability map of abnormal regions

U-Net model

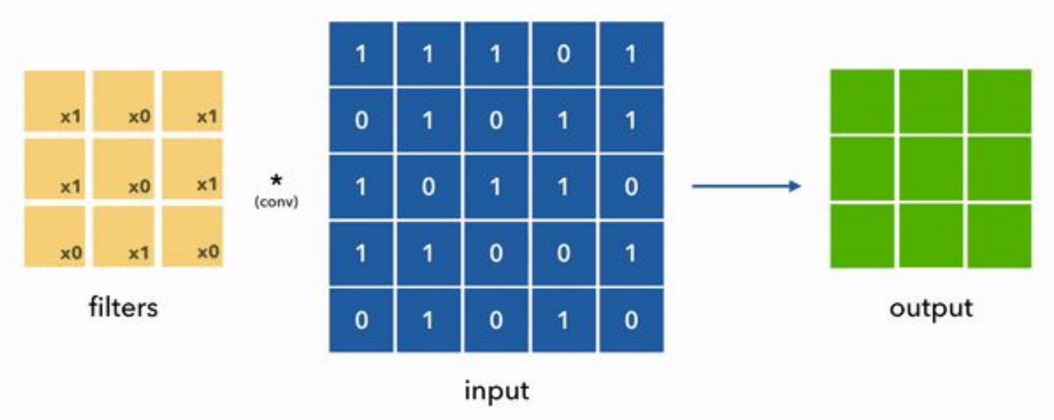
(developed specifically for segmenting abnormalities from brain MRI)

Buda et al, Computers in Biology and Medicine (2019)

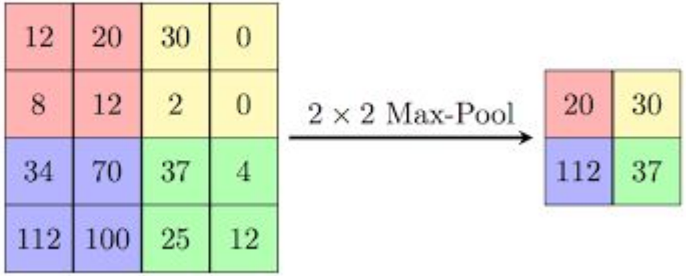
```python
model = torch.hub.load('mateuszbuda/brain-segmentation-pytorch', 'unet',
    in_channels=3, out_channels=1, init_features=32, pretrained=True)
```

Building blocks

Convolution



Max Pooling



3. Loss

- Quantify how well our model predictions **perform** v.s. ground truth
- Model **learns by reducing** the loss function
- Improve model's predictive ability by **minimising** the loss function
- **Choice** of loss function is important

3. Loss

DICE score
Accurate boundary localisation

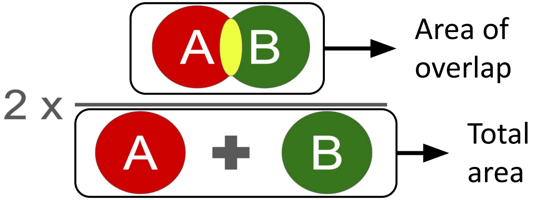
+

Binary Cross-Entropy (BCE)
Measures pixel-wise similarity between predicted & target mask

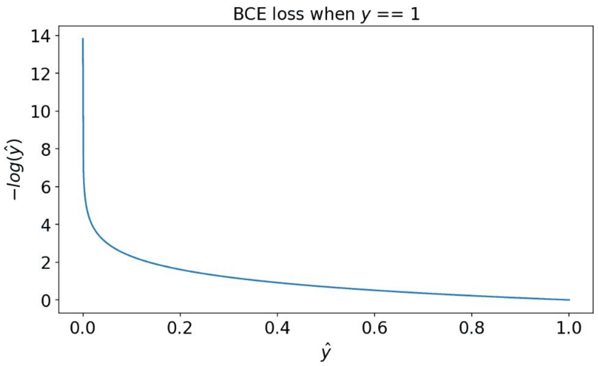
=

BCE Dice Loss
Accurate object localisation & overall segmentation accuracy

A = Prediction   B = Ground truth (target)



DICE score



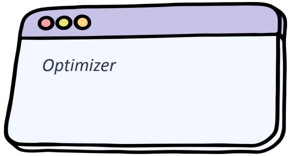

```python
def bce_dice_loss(inputs, target):
    dicescore = dice_loss(inputs, target)
    bcescore = nn.BCELoss()
    bceloss = bcescore(inputs, target)

    return bceloss + dicescore
```

Balanced approach for image segmentation

4. Optimizer

- Optimizer **updates the value of weights** using the gradient from loss.backward to **minimize loss**
  - Gradient = change in loss function w.r.t. weight of model
  - Gradient ⟶ 0: minimise loss, achieve **optimal model configuration**
- Lr scheduler to update learning rate depending on the training behaviour

```
optimizer = AdamW(model.parameters(), 0.1)
scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
```

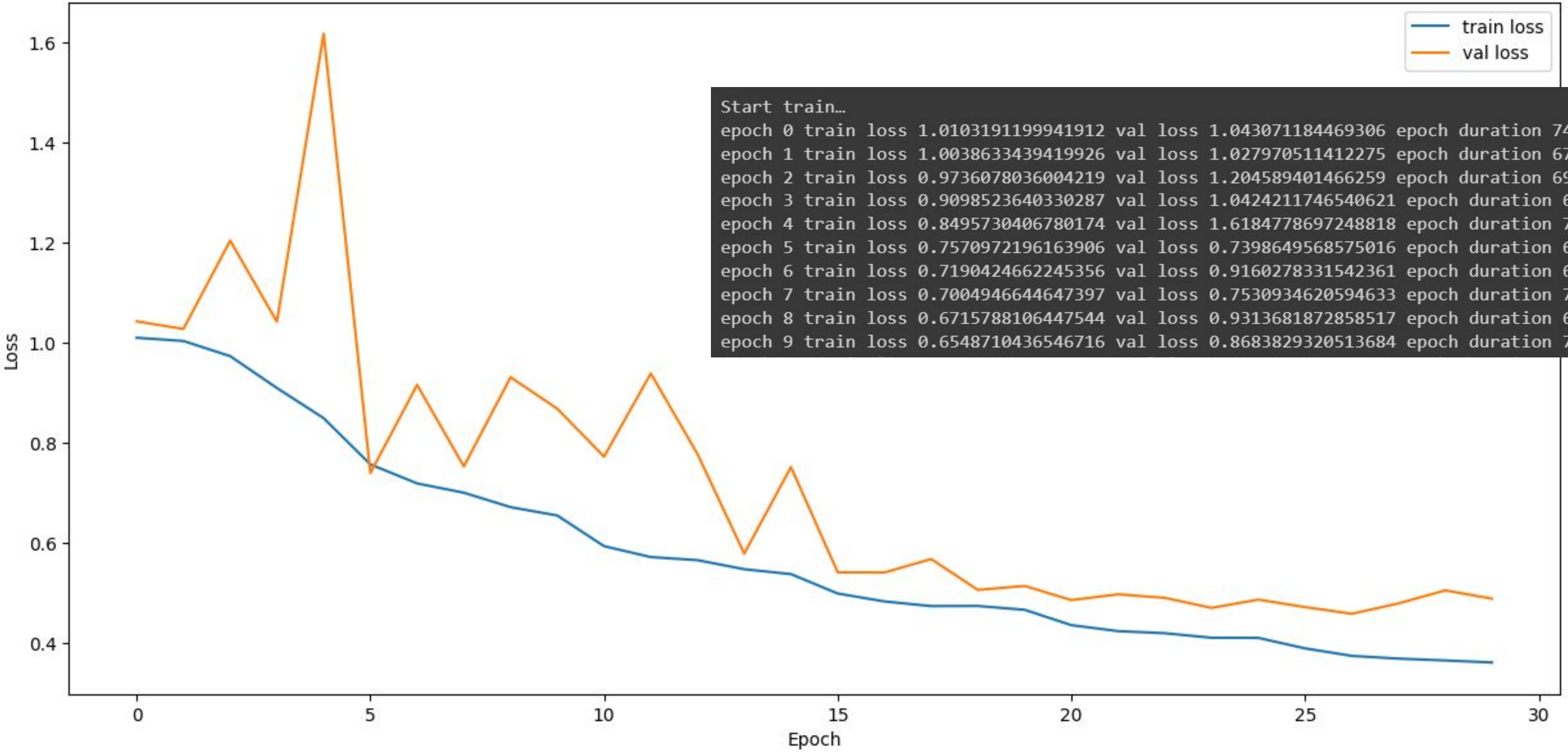


5. Training

Training loop:

- Train
  - Predict
  - Compute loss
  - Optimize
- Validate
  - Predict
  - Compute loss
- Save best model
- Adjust learning rate
- Repeat

5. Training




```
Start train…
epoch 0 train loss 1.0103191199941912 val loss 1.043071184469306 epoch duration 74.04199075698853
epoch 1 train loss 1.0038633439419926 val loss 1.027970511412275 epoch duration 67.39683508872986
epoch 2 train loss 0.9736078036004219 val loss 1.204589401466259 epoch duration 69.68581986427307
epoch 3 train loss 0.9098523640330287 val loss 1.0424211746540621 epoch duration 67.49174499511719
epoch 4 train loss 0.8495730406780174 val loss 1.6184778697248818 epoch duration 70.41903424263
epoch 5 train loss 0.7570972196163906 val loss 0.7398649568575016 epoch duration 67.40357279777527
epoch 6 train loss 0.7190424662245356 val loss 0.9160278331542361 epoch duration 68.81970643997192
epoch 7 train loss 0.7004946644647397 val loss 0.7530934620594633 epoch duration 70.83153796195984
epoch 8 train loss 0.6715788106447544 val loss 0.9313681872858517 epoch duration 69.40048336982727
epoch 9 train loss 0.6548710436546716 val loss 0.8683829320513684 epoch duration 73.2097098827362
```

6. Inference

- **Save & load** models
- Use **best model weights** to segment tumours (test set)

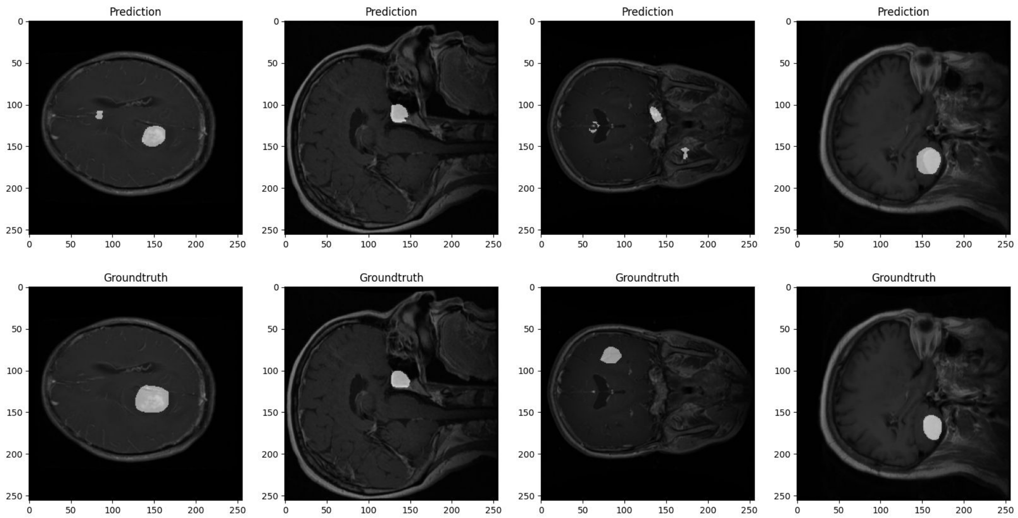
```
if curr_loss_val < best_loss_val:
    best_loss_val = curr_loss_val
    torch.save(model.state_dict(), r'/content/best_model.pth')
```

```
checkpoint = torch.load(
    '/content/best_model.pth', map_location=torch.device('cpu'))
model.load_state_dict(checkpoint)
```

Inference -> Load best model weights
Inference -> Visualization
Inference -> Accuracy

6. Inference

- Save & load models
- Use best model weights to segment tumours (test set)
- **Visualize** predicted and ground truth images
- Compute **model accuracy** using test cases




```
[27] # Test set
     acc_test = compute_acc(test_dataloader, model)
     print(f'Acccuracy on the train set is {acc_test}')

     Acccuracy on the train set is 0.6898969995913493
```

Inference -> Load best model weights
Inference -> Visualization
Inference -> Accuracy

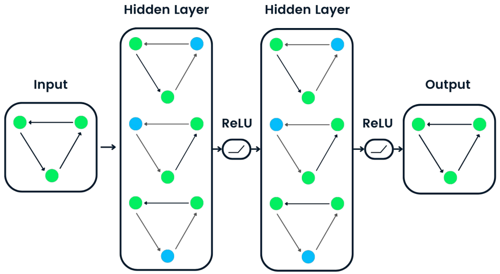
Summary

- **Learnt** how to **set up** a machine learning project in PyTorch
- **Trained** and **tested** a machine learning model to perform brain tumour **segmentation**
- Performed **inferences** with the trained model

Now, let's head over to the code!

What’s next?

Graph Convolution Network



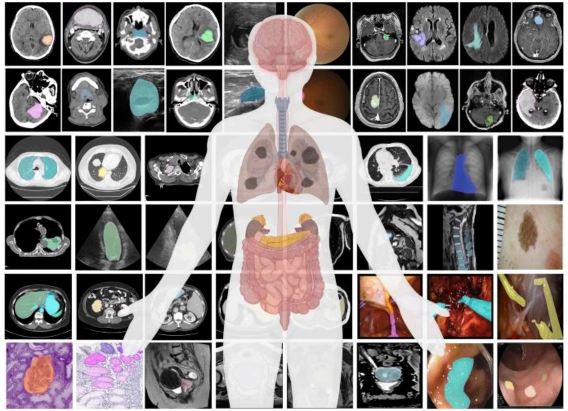
Diffusion Models

Article | Open access | Published: 22 January 2024

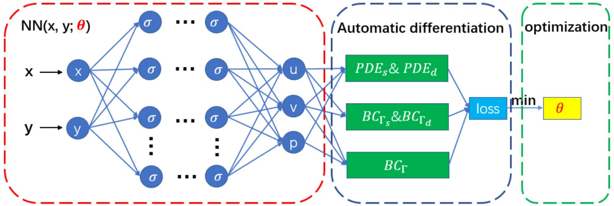
Segment anything in medical images

Jun Ma, Yuting He, Feifei Li, Lin Han, Chenyu You & Bo Wang

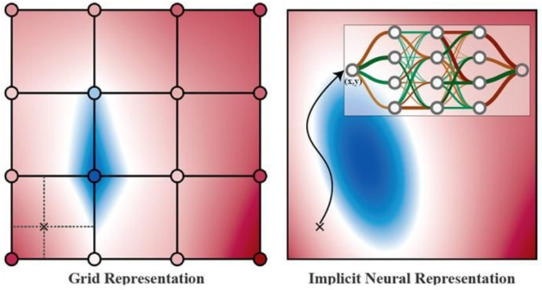
Nature Communications 15, Article number: 654 (2024) | Cite this article



Physics-Informed Neural Networks (PINNs)



Neural Implicit Representation (NIR)



ML.init() team

Gonzalo Maso Talou

Edward Ferdian

Alireza Farrokhi Nia

Jiantao Shen

Mathilde Verlyck

Matthew French

Max Dang Vu

Mostafa Papen

If you’re interested to use ML in your work, or are just curious:

feel free to reach out for a chat!