BVM 2019 Tutorial

Hands-on deep learning using PyTorch (Part 2)

Image Registration with PyTorch

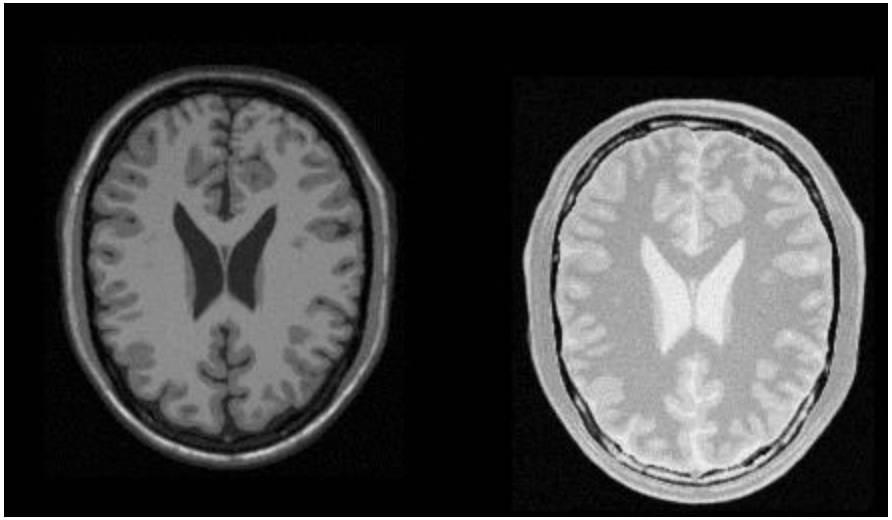
Max Blendowski, Christian Lucas & Mattias P. Heinrich

Institute of Medical Informatics

University of Lübeck

Motivation

- Speaking with images: You want to go from here …

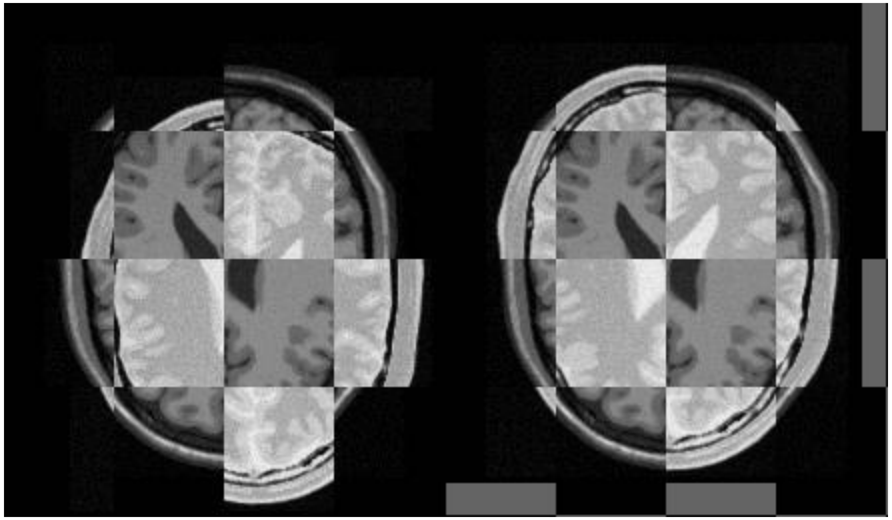


T1 (fixed) PD (moving)

[Image: itk Documentation : Perform Multi modality registration with Viola Wells MI]

Motivation

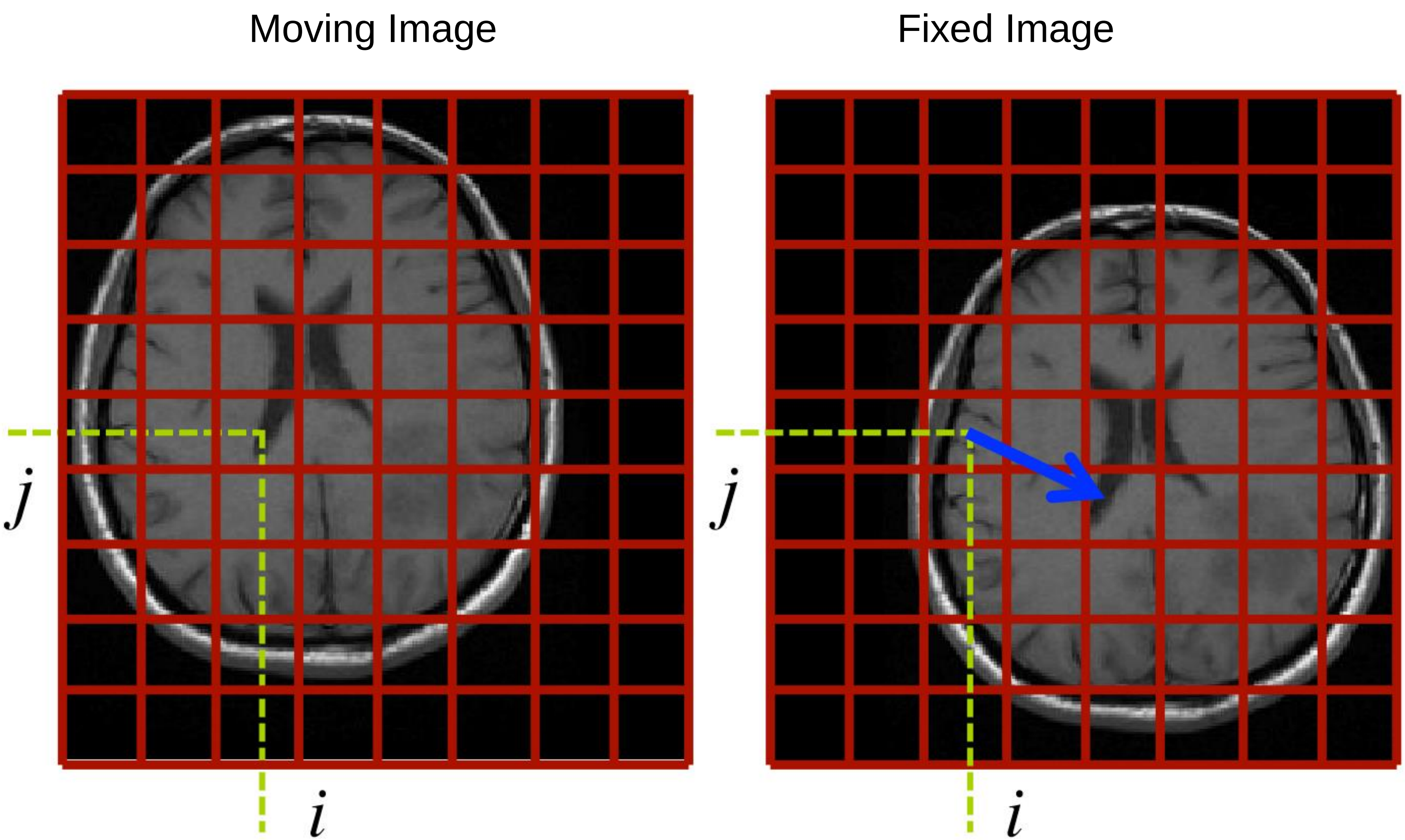
- ... to here!



init reg

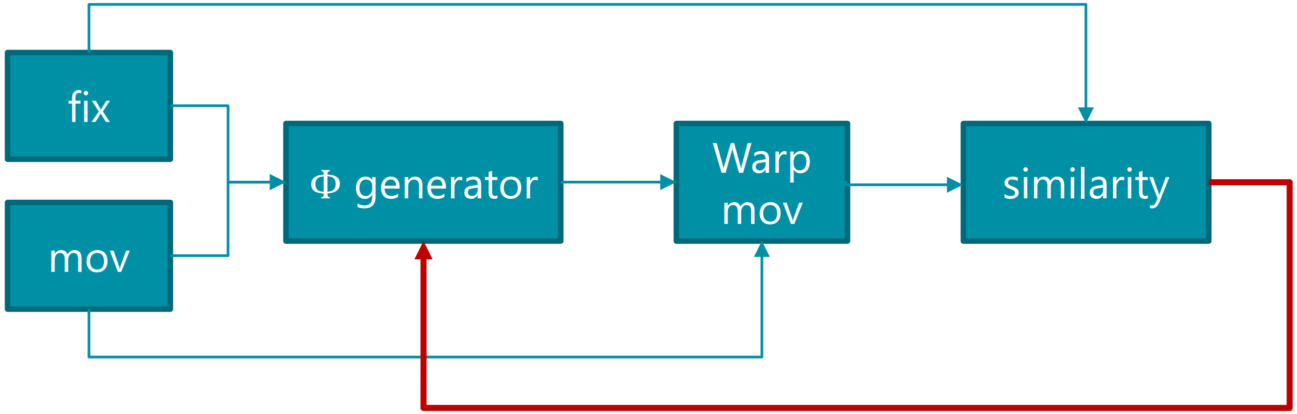
[Image: itk Documentation : Perform Multi modality registration with Viola Wells MI]

Motivation



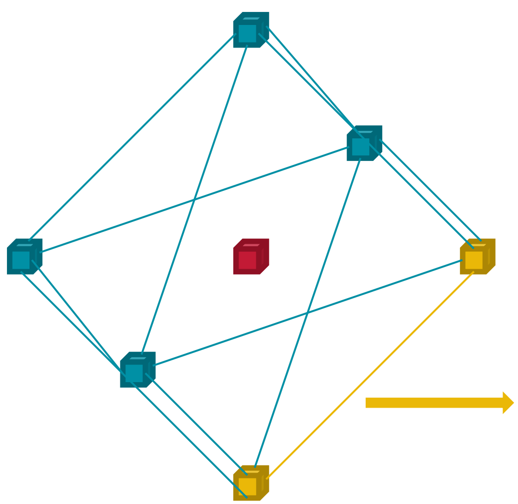
Basics

- How do we achieve this?
  - Find suitable **representations** for images
  - **Classical** approaches:
    - Mutual Information: focussing on the metric
    - MIND: Use structural image information
    - Many more handcrafted features
  - Find φ_{Trafo} that **transforms** *mov* towards *fix*



Basics : Baseline SSC Descriptor

- Self-Similarity Context Descriptor by Heinrich et al. [2]:
  - Similarity $S(x, y) = \exp\left(-\frac{SSD(x,y)}{\sigma^2}\right) \in [0,1]$
  - [0,1] quantized in 5 bins, 12 pairs: stored in 64float (xor,popcnt)
  - Fair 256 bit comparison: 4 SSC per position (center, right, front, bottom)



$$SSC = \{b_1, \dots, b_7, \dots, b_{12}, 0,0,0,0\}$$

$$b_7 = quant(0.227) = \{0,0,1,1,1\}$$

[2]: Heinrich et al.: Towards Real-time Multimodal Fusion for Image-Guided Interventions using Self-Similarites, MICCAI‘13

Basics

- We interprete images as 2D functions I(x,y) evaluated at **discrete** voxels (i,j)
- Seen from the **fixed image**, sample positions in **moving** to „drag“ it closer
- The type of dragging has to be defined beforehand
  - affine (scaling, rotation, translation → global transformations)
  - Deformable (will be used here!)

Basics

- At every image grid position, we want to know where to gather our image values from in the moving image

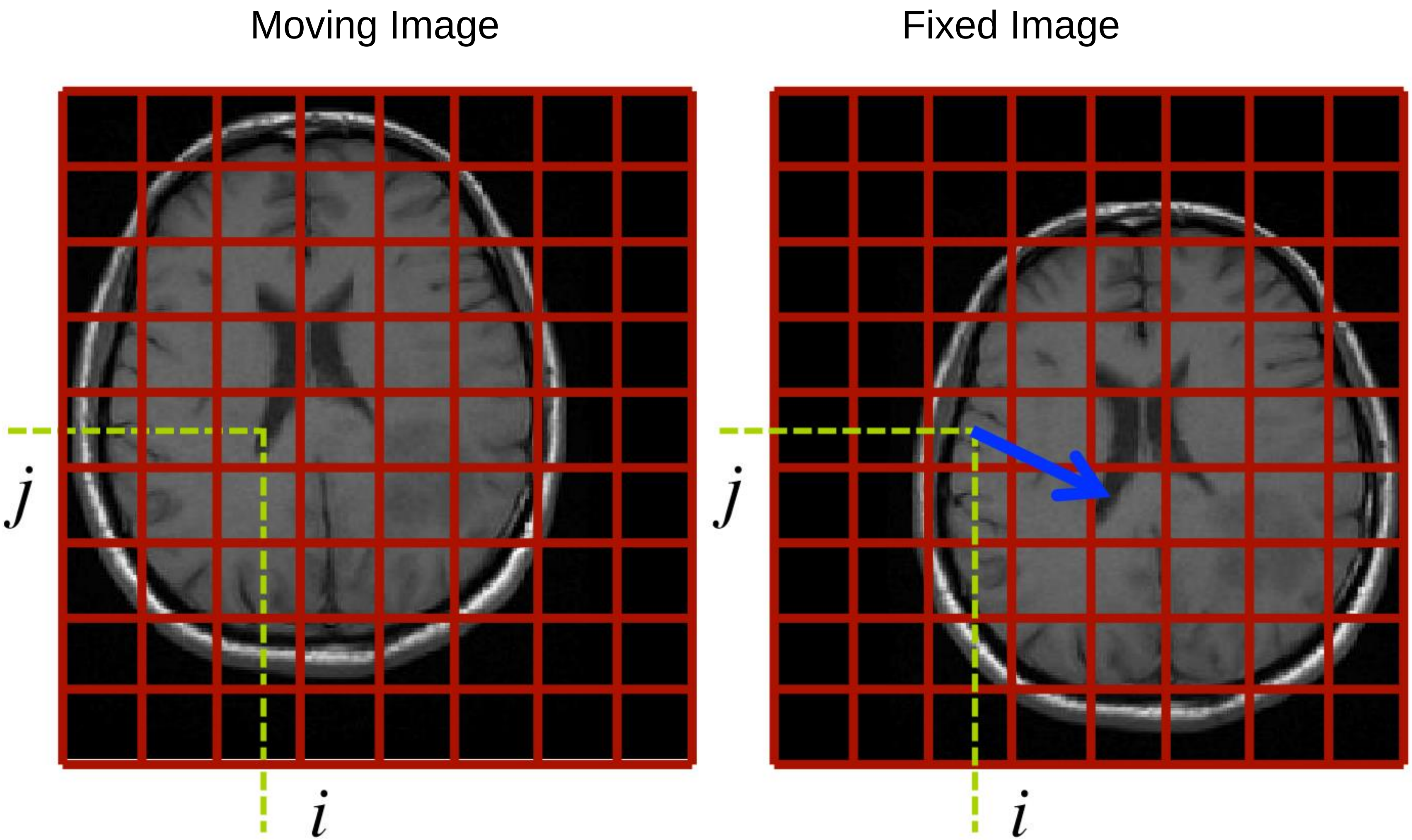
Basics

- To put it more mathematically, we are facing an **optimization problem**

$$argmin_{\varphi}\, \mathcal{D}(\mathbf{S}_{\mathcal{F}}, \varphi \circ \mathbf{S}_{\mathcal{M}}) + \alpha \mathcal{R}(\varphi)$$

- Note that usually it is assumed that φ already contains the identity and only the displacements from the pixel center have to be optimized

Basics



Basics

- How do we optimize our transformation?
- The standard approach: use **gradient based optimization**
- Depending on the transformation model, the regularizer, the similarity metric, (...) this can be a **very challenging** task ...
- ….. Until now! This is where **autograd engines** come in handy!

Basics

- A screenshot from the old times

Calculating derivatives

$$\frac{\partial e_{ij}}{\partial \Theta_k} = \underbrace{\frac{\partial}{\partial \Theta_k}\Big(I\big(x(i,j;\Theta), y(i,j;\Theta)\big)}_{\begin{pmatrix} \frac{\partial x(i,j)}{\partial \Theta_k} & \frac{\partial y(i,j)}{\partial \Theta_k} \end{pmatrix} \begin{pmatrix} \frac{\partial I\left(x(i,j;\Theta), y(i,j;\Theta)\right)}{\partial x} \\ \frac{\partial I\left(x(i,j;\Theta), y(i,j;\Theta)\right)}{\partial y} \end{pmatrix}} - \underbrace{I_R\big(i,j\big)}_{\text{independent of } \Theta}\Big)$$

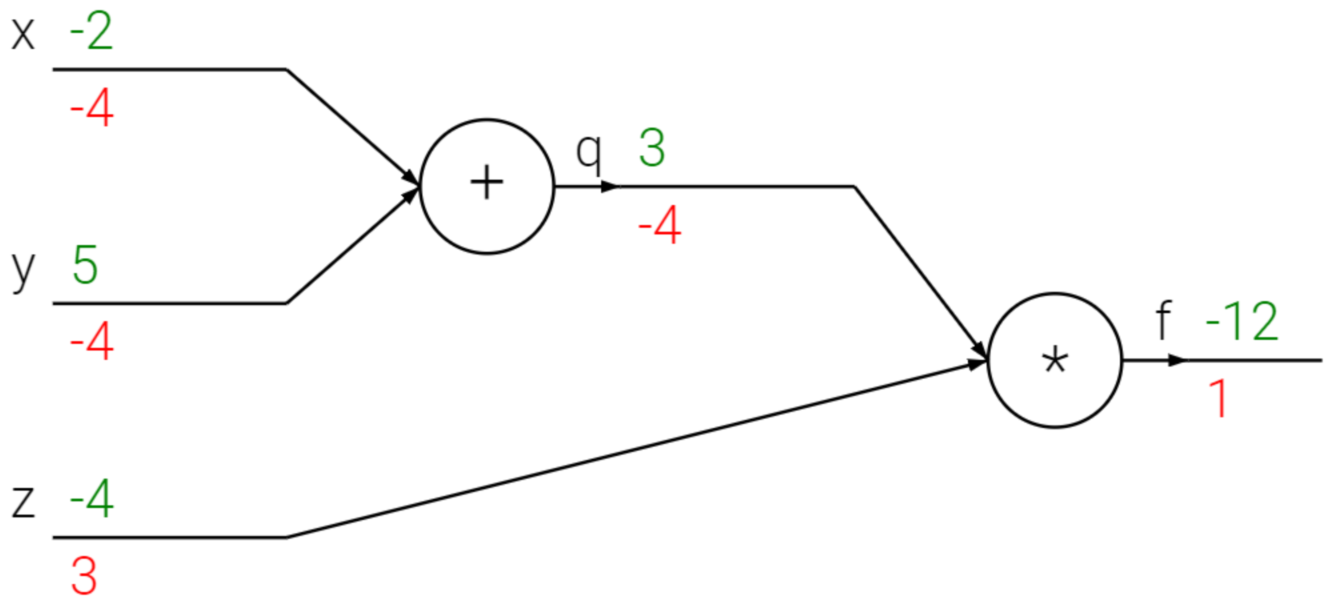
(chain rule in higher dimensions)

Autograd

- Frameworks as **PyTorch** are keeping track of operations and allow to backpropagate gradients automatically
- Only prerequisite: for every operation, the backward step has to be defined locally (given the input and the gradient at the output → chain rule)

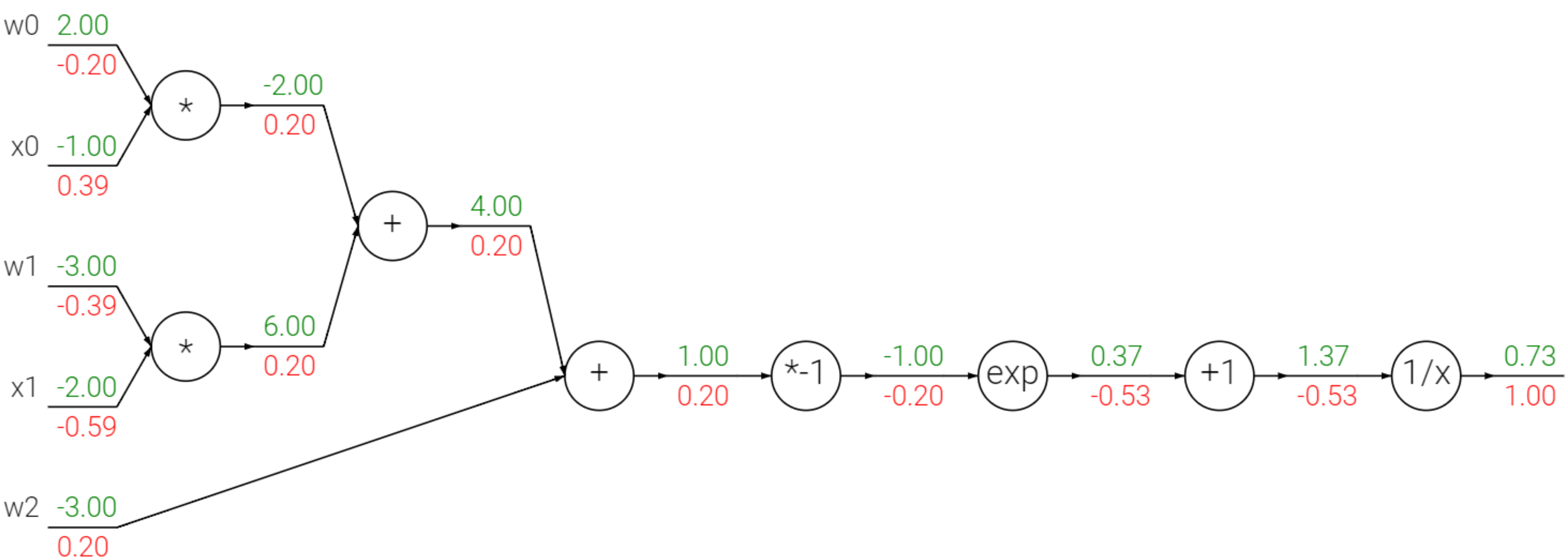
Autograd

- Again more graphically: (CS231n Stanford!)
- See whiteboard



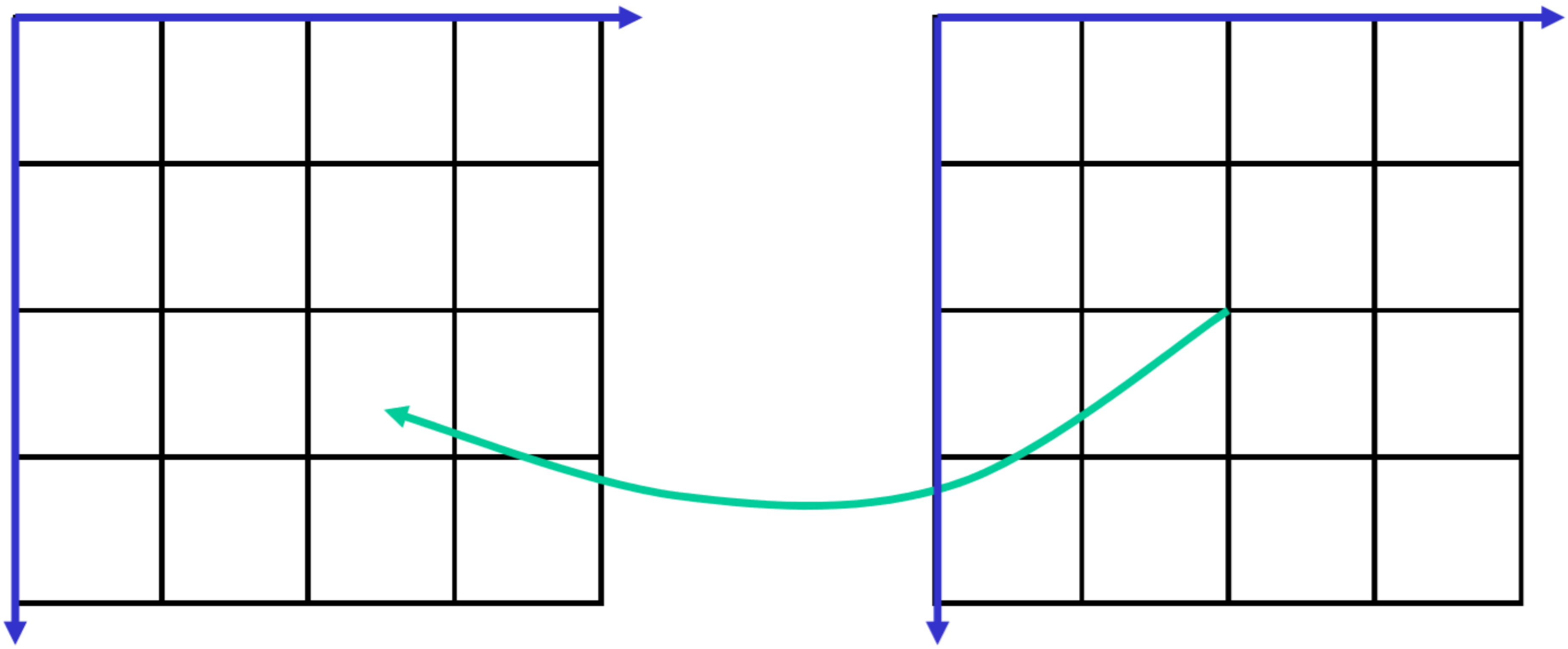
Autograd

- Works also for more complex examples (and deep networks)
- Which function is depicted by this graph?



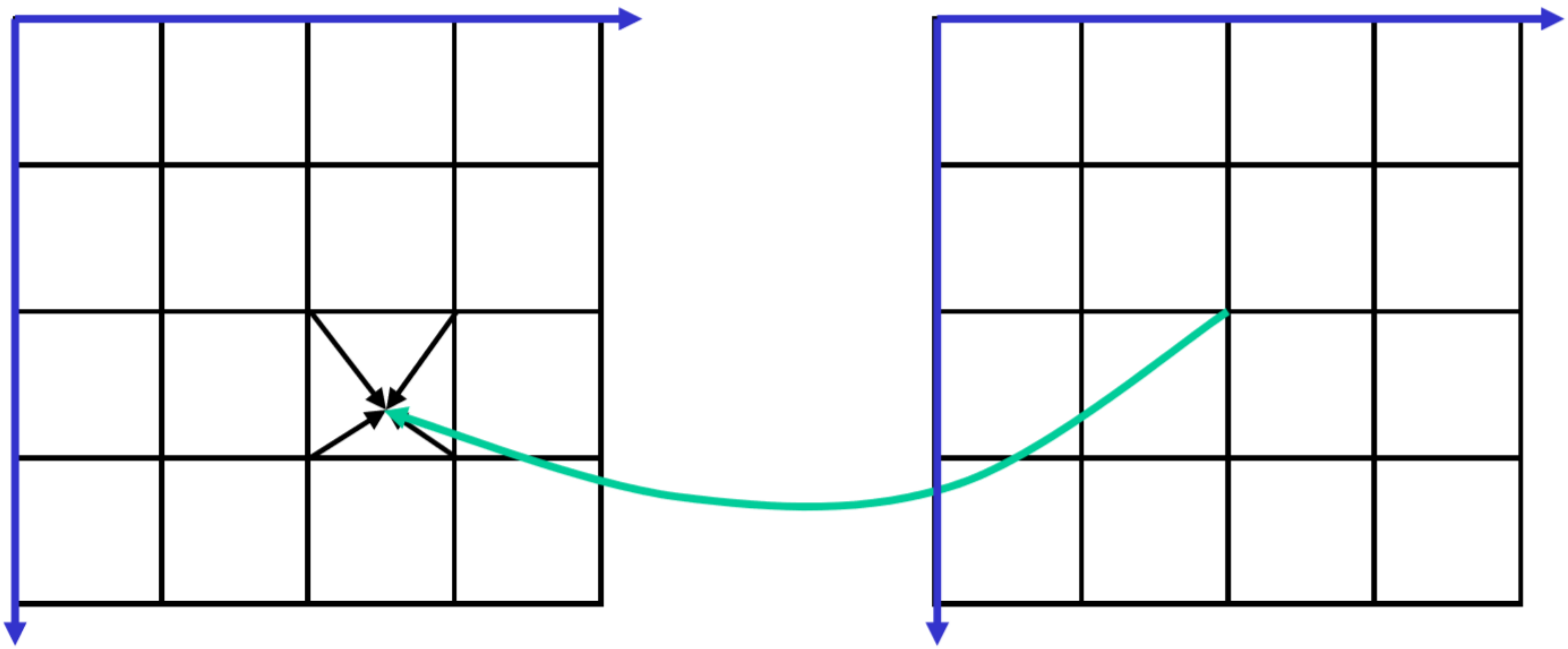
Autograd

- Return to image registration



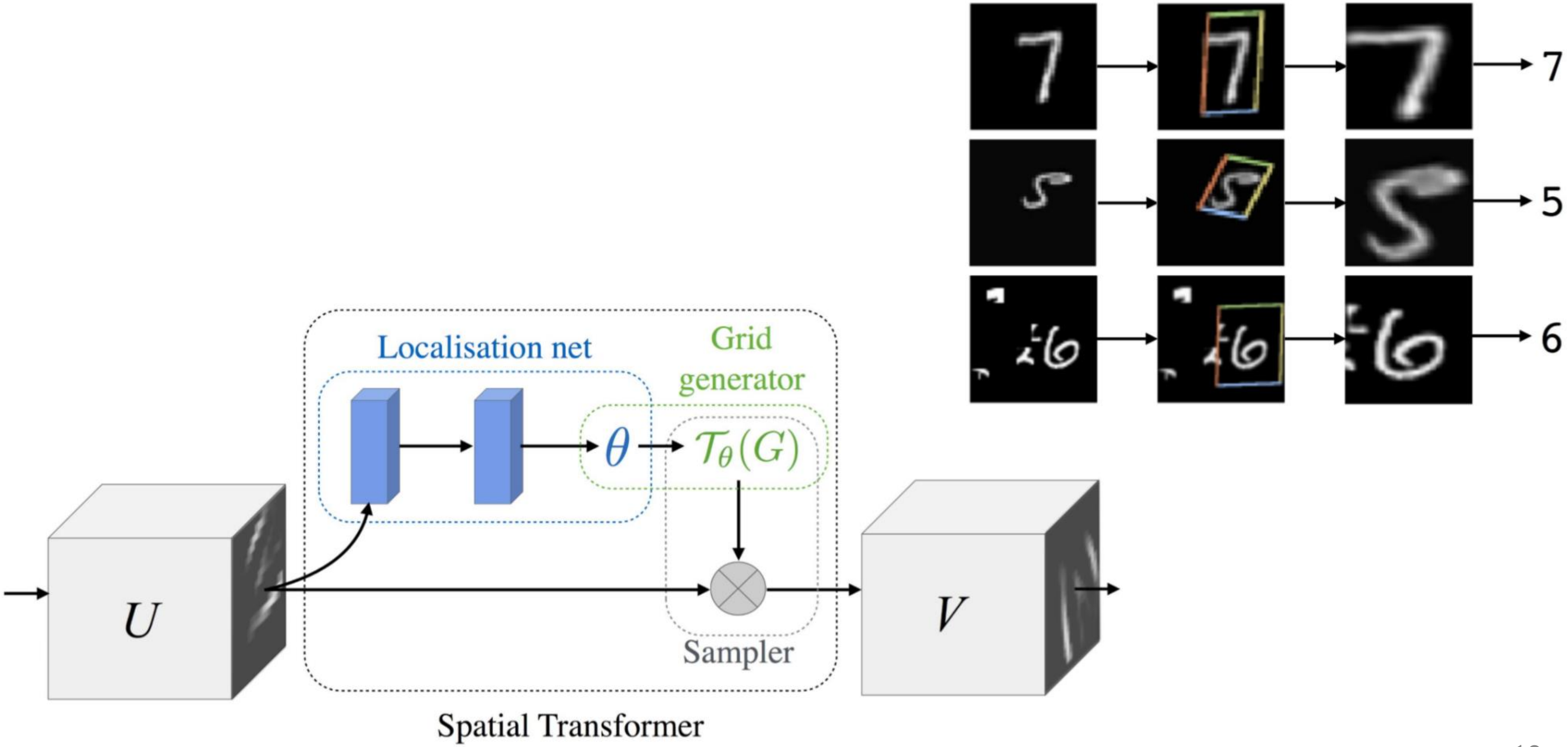
Autograd

- Often we need to interpolate. How do the gradients look like?



Autograd

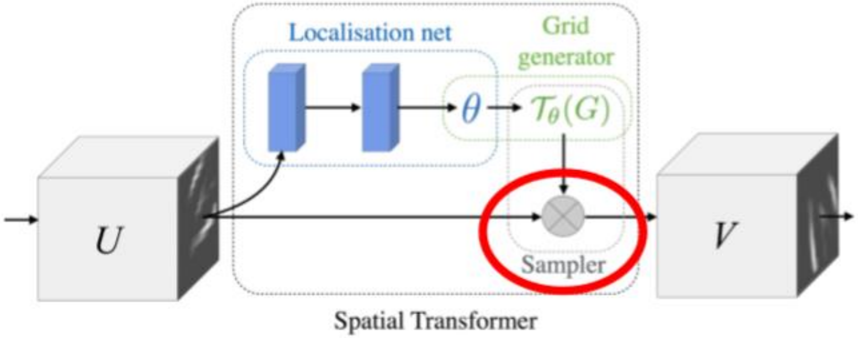
- Jaderberg et al. introduced **Spatial Transformer Networks**



Autograd

- Following Tyagi & Gupta

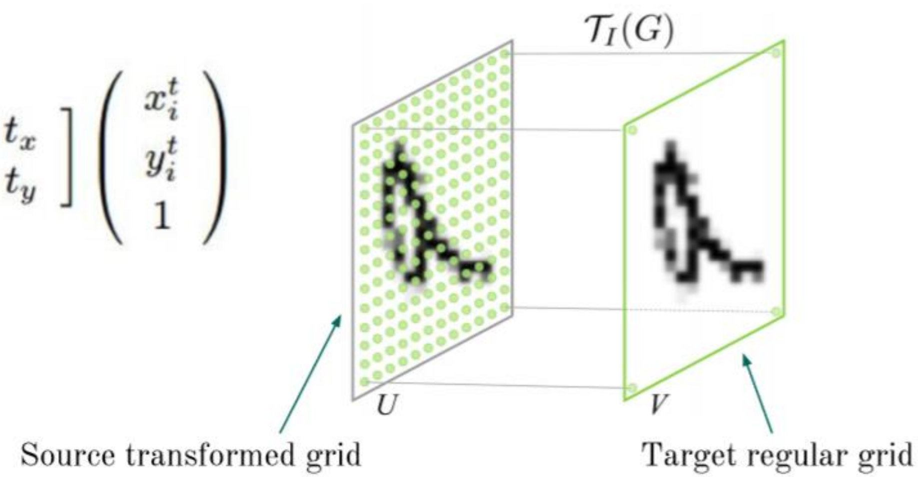
Samples the input feature map by using the sampling grid and produces the output map.



Autograd

- Following Tyagi & Gupta

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

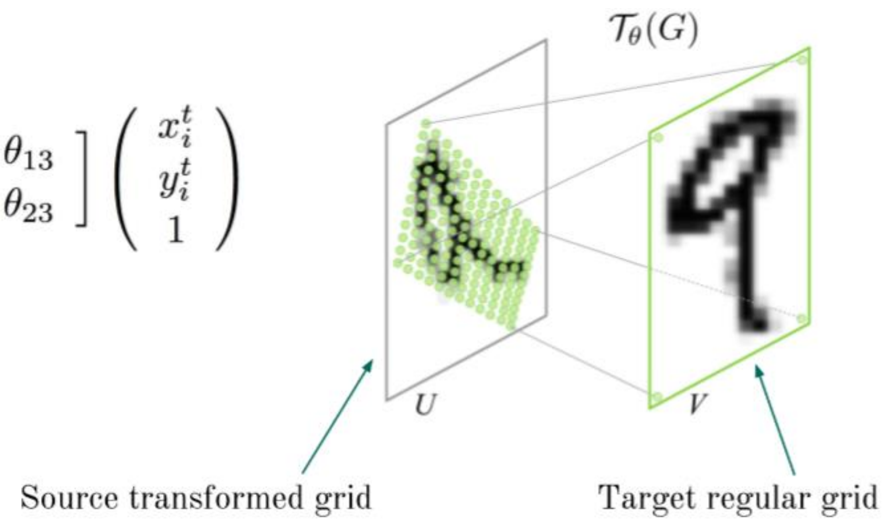


Identity Transform
(s=1, t_x=0, t_y = 0)

Autograd

- Following Tyagi & Gupta

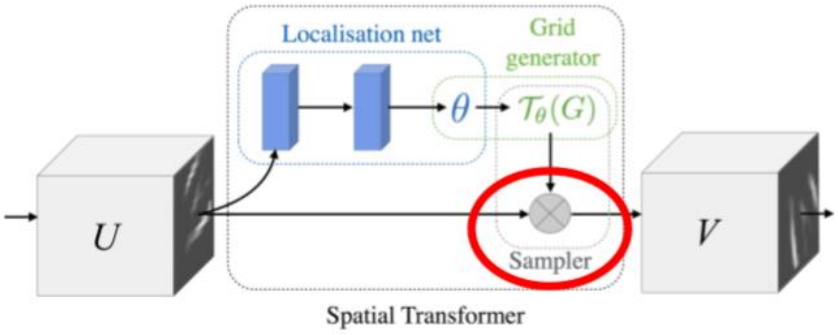
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$



Autograd

- Following Tyagi & Gupta

Samples the input feature map by using the sampling grid and produces the output map.



Autograd

- Following Tyagi & Gupta

Bilinear sampling kernel

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

Gradient with bilinear sampling kernel

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_n^H \sum_m^W \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } m < x_i^s \end{cases}$$

Autograd

- This is all implemented efficiently in PyTorch
- torch.nn.functional.grid_sample() → read the docs and check part two of this tutorial with its image_warp and label_warp functions!

Autograd

```
import torch
import numpy as np
import matplotlib.pyplot as plt
```

```
from PIL import Image

homer = Image.open('homer.png').convert('RGBA')
homer = np.array(homer).astype('float')
homer = (torch.from_numpy(homer).permute(2,0,1).unsqueeze(0).contiguous()/255.0).float()
plt.imshow(homer.permute(0,2,3,1)[0,...])
plt_max_val = torch.max(homer.view(-1))
```

Autograd

```
# generate sampling grid
smp_0 = torch.linspace(-0.3,0.4,600) # location on axis 0 & sampling frequency
smp_1 = torch.linspace(-0.5,0.5,700) # location on axis 1 & sampling frequency
g0,g1 = torch.meshgrid(smp_0,smp_1) # build a mesh from it
smp_mesh = torch.stack((g1,g0),2).unsqueeze(0) # grid has to be of dimensions [B,1,H,W]

sampled_img = torch.nn.functional.grid_sample(homer,smp_mesh) # sample from the input

plt.imshow(sampled_img.permute(0,2,3,1)[0,...],vmin=0,vmax=max_plt_val) #show
```

```
<matplotlib.image.AxesImage at 0x114dc060ac8>
```