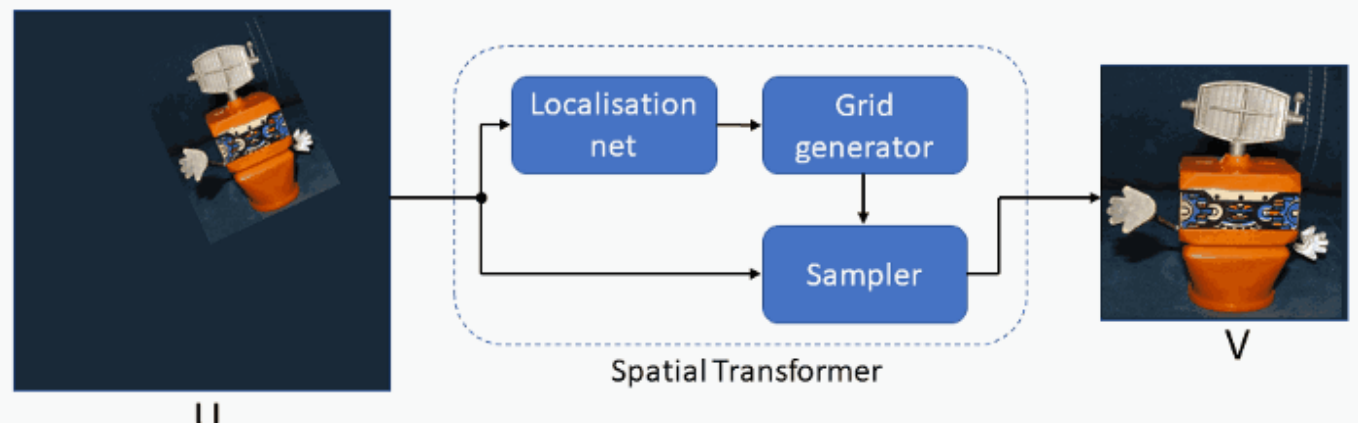
THOUGHTS AND THEORY

Spatial Transformer Networks

A Self-Contained Introduction

Thomas Kurbiel Sep 27 · 7 min read

Spatial Transformer modules, introduced by Max Jaderberg et al., are a popular way to increase spatial invariance of a model against spatial transformations such as translation, scaling, rotation, cropping, as well as non-rigid deformations. They can be inserted into existing convolutional architectures: either immediately following the input or in deeper layers. They achieve spatial invariance by adaptively transforming their input to a canonical, expected pose, thus leading to a better classification performance. The word adaptive indicates, that for each sample an appropriate transformation is produced, conditional on the input itself. Spatial transformers networks can be trained end-to-end using standard backpropagation.

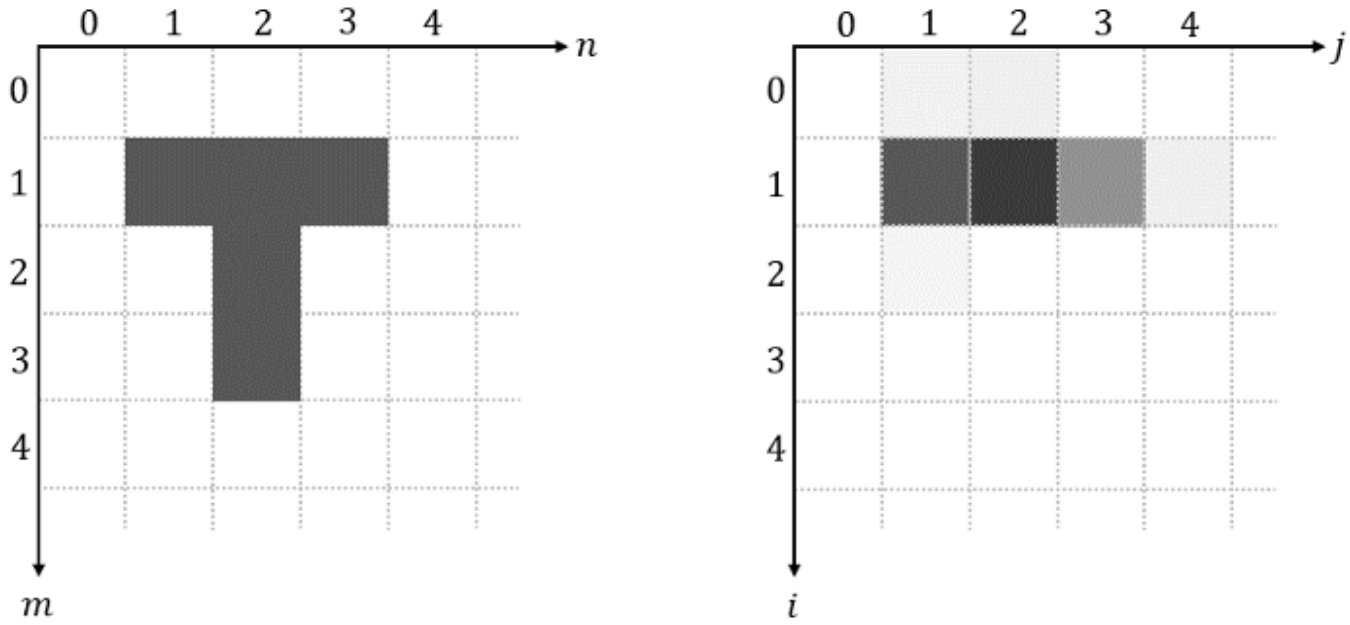


Spatial transformer module transforms inputs to a canonical pose,
thus simplifying recognition in the following layers (Image by author)

In this four-part tutorial, we cover all prerequisites needed for gaining a deep understanding of spatial transformers. In the last two posts, we have introduced the concepts of forward and reverse mapping and delved into the details of bilinear interpolation. In this post, we will introduce all building blocks a spatial transformer module is made of. Finally, in the next and last post, we will derive all necessary backpropagation equations from scratch.

Separation of Responsibilities

In order to understand the motivation behind some of the building blocks of the spatial transformer module, we have to quickly repeat the principles of reverse mapping introduced in the first post.



Reverse mapping (Image by author)

In reverse mapping we go through the output image, one pixel at a time, and for each position we perform two operations:

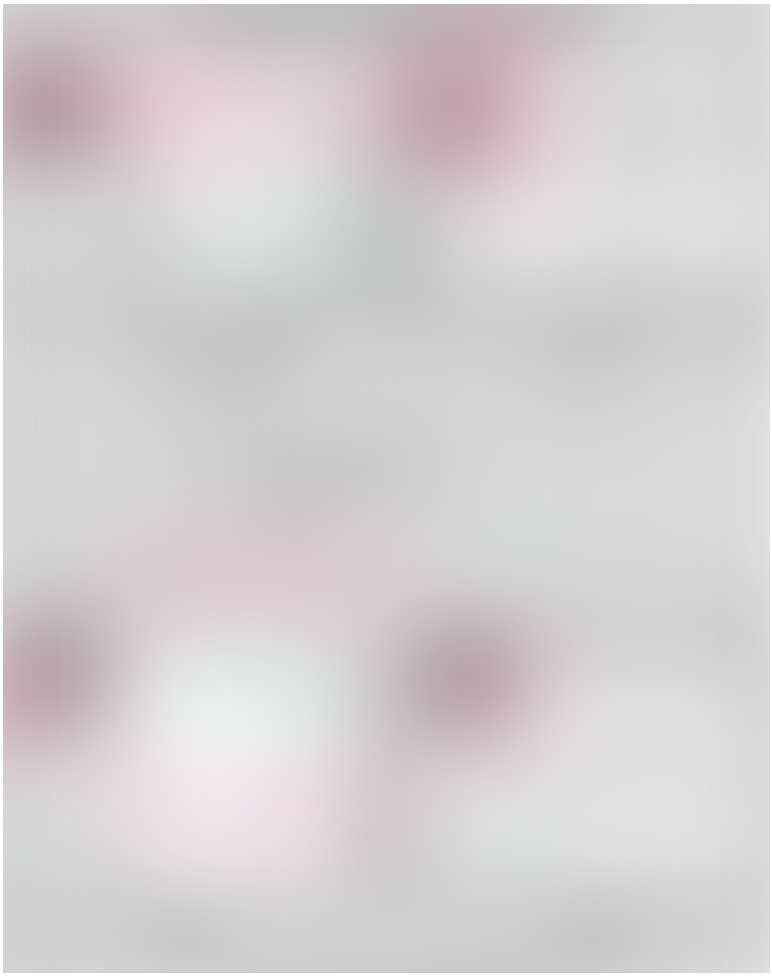
1. use the inverse transformation T-1{…} to calculate the corresponding position in the input image
2. sample the pixel value using bilinear interpolation

The reason we are performing both operations directly one after the other in the animation above is mainly to illustrate the concept of reverse mapping. When implementing reverse mapping, however, it is beneficial to first calculate the corresponding positions for all output pixels (and maybe store them), and only then apply bilinear interpolation. It should be obvious, that this has no effect on the final outcome.

The main benefit of this approach is that we now get two components with separate responsibilities: **grid generator** and **sampler**. The **grid generator** has the exclusive task of performing the inverse transformation and the **sampler** has the exclusive task of performing bilinear interpolation. Furthermore, as we will see in the next post, the separation strongly facilitates backpropagation.

Grid Generator

The **grid generator** iterates over the regular grid of the output/target image and uses the inverse transformation T-1{…} to calculate the corresponding (usually non-integer) sample positions in the input/source image:



Grid generator (Image by author)

The superscripts *t* and *s* are taken from the original paper and denote "*target image*" and "*source image*". The row and column indexes of the **sampling grid** are denoted as *i* and *j*, respectively. Please also note, that in the original paper the inverse transformation T-1{…} over the regular output grid is denoted as $\mathcal{T}\theta(G)$.

Whereas in the above illustration the coordinates are calculated in a sequential manner for the sake of clarity, real world implementations of the **grid generator** will try to transform as many points as possible in parallel for reasons of computational efficiency.

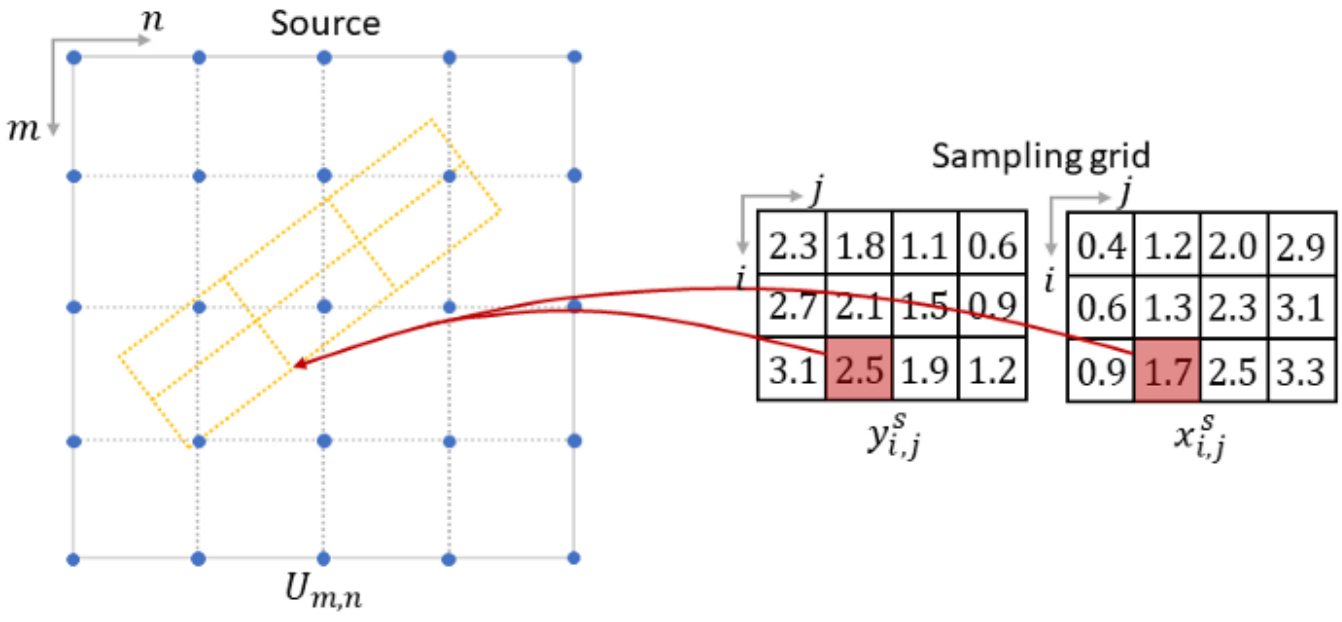
The output of the grid generator is the so called **sampling grid**, which is a set of points where the input map will be sampled to produce the spatially transformed output:

$$(y^s_{i,j}, x^s_{i,j})$$

where

$$0 \leq i < H' \quad \text{and} \quad 0 \leq j < W'$$

Please note, that the size of the **sampling grid**, determines the size of the target image.

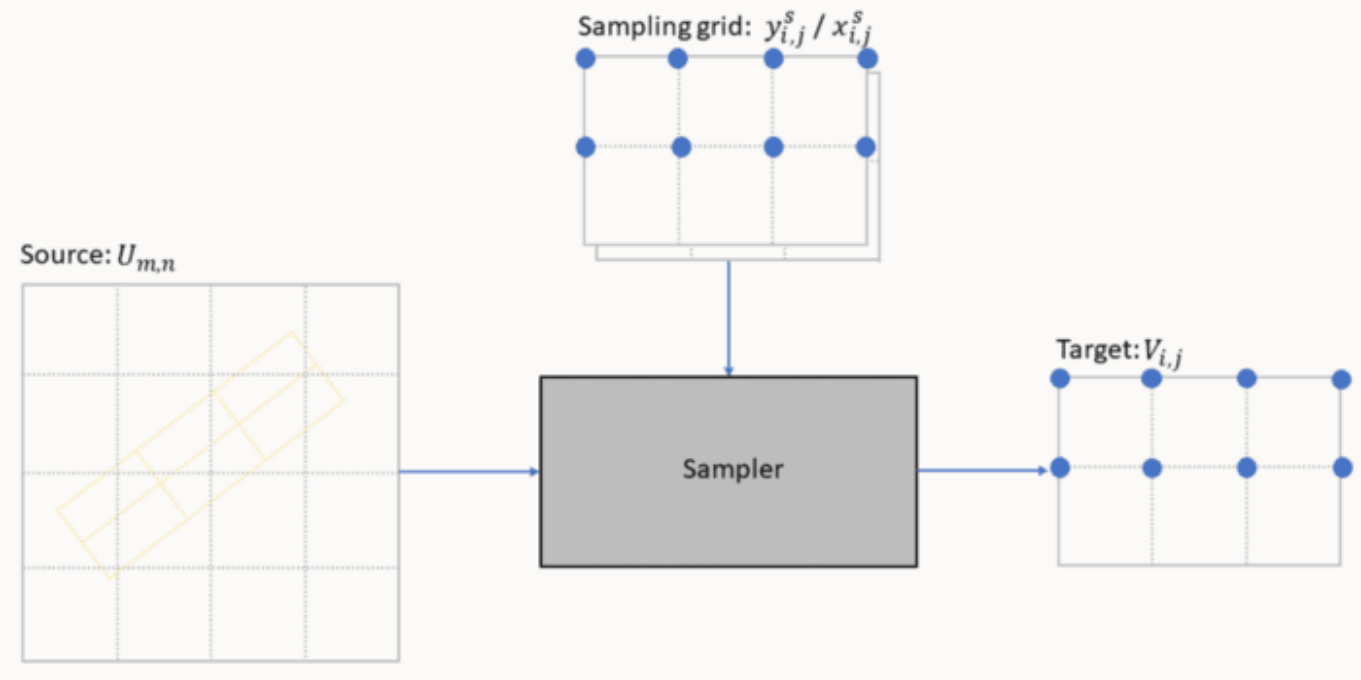


Sampling grid contains points where input/source map will be sampled (Image by author)

One last important thing to mention about the **sampling grid** is, that its height and width do not necessarily need to be the same as the height and width of the input image.

Sampler

The **sampler** iterates over the entries of the **sampling grid** and extracts the corresponding pixel values from the input map using bilinear interpolation:



Sampler (Image by author)

The extraction of a pixel value consists of three operations:

1. find the four neighboring points (upper left, upper right, lower left and lower right)
2. for each neighboring point calculate its corresponding weight
3. take the weighted average to produce the output

All operations are summarized in the following equation, which has been derived in the last post:

$$V_{i,j} = U_{y_{i,j},x_{i,j}} \cdot d_{y_{i,j}} \cdot d_{x_{i,j}} +$$

$$U_{\underline{y_{i,j}},\overline{x_{i,j}}} \cdot d_{y_{i,j}} \cdot (1 - d_{x_{i,j}}) +$$

$$U_{\overline{y_{i,j}},\underline{x_{i,j}}} \cdot (1 - d_{y_{i,j}}) \cdot d_{x_{i,j}} +$$

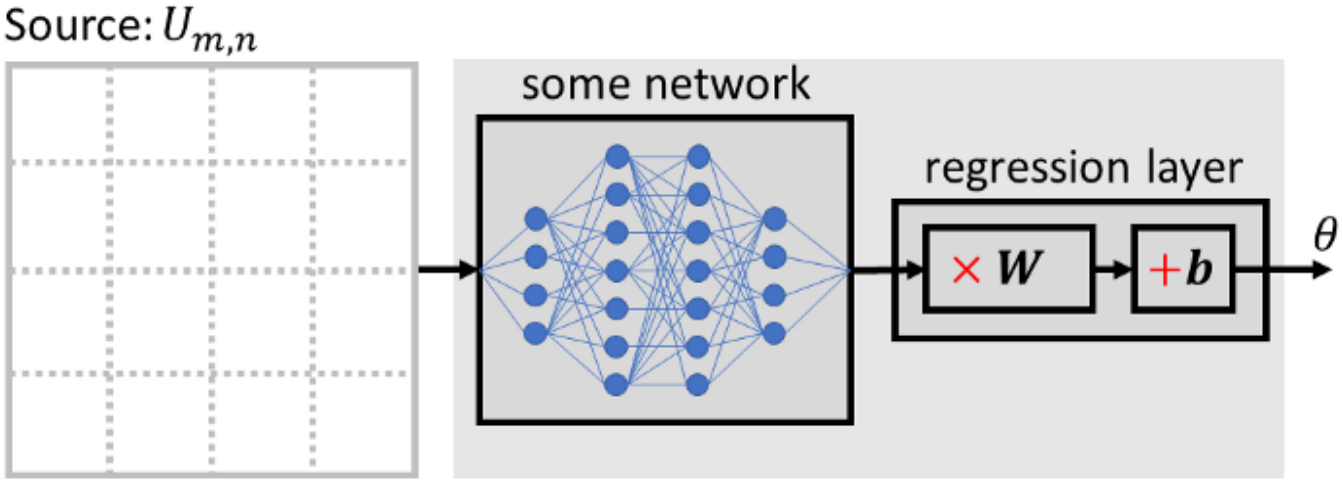
$$U_{\overline{y_{i,j}},\overline{x_{i,j}}} \cdot (1 - d_{y_{i,j}}) \cdot (1 - d_{x_{i,j}})$$

Remember dx denotes the horizontal distance from the sample point to the right cell border and dy the vertical distance to the top cell border.

As in the **grid generator** the calculation of each output pixel is totally independent of any other output pixel. Hence again, real world implementations of the **sampler** will speed up the process, by extracting in parallel as many points as possible.

Localisation Net

The task of the **localisation network** is to find parameters θ of the inverse transformation T-1{…}, which puts the input feature map to a canonical pose, thus simplify recognition in the following layers. The **localisation network** can take any form, such as a fully-connected network or a convolutional network, but should include a final regression layer to produce the transformation parameters θ:



Localisation net (Image by author)

The size of θ can vary depending on the transformation that is parameterized, e.g. for an affine transformation θ is 6-dimensional:

$$\begin{bmatrix} x_{i,j}^s \\ y_{i,j}^s \end{bmatrix} = T^{-1}\left\{\begin{bmatrix} x_{i,j}^t \\ y_{i,j}^t \end{bmatrix}\right\} = \begin{bmatrix} \theta_1 & \theta_2 \\ \theta_3 & \theta_4 \end{bmatrix} \cdot \begin{bmatrix} x_{i,j}^t \\ y_{i,j}^t \end{bmatrix} + \begin{bmatrix} \theta_5 \\ \theta_6 \end{bmatrix}$$

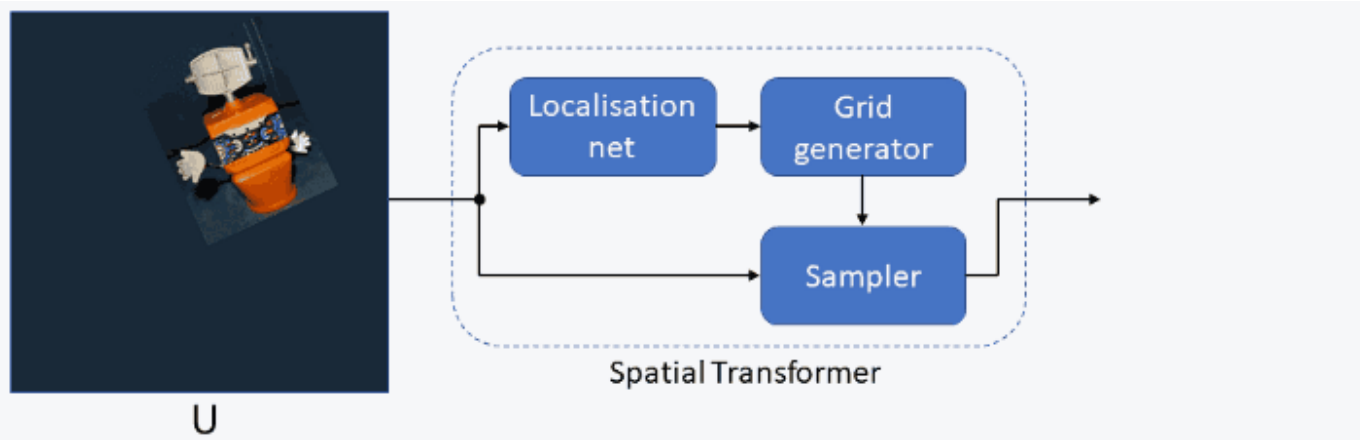
The affine transform is quite powerful and contains translation, scaling, rotation and shearing as special cases. For many tasks however a simpler transformation may be sufficient, e.g. a pure translation is implemented using only 2 parameters:

$$\begin{bmatrix} x_{i,j}^s \\ y_{i,j}^s \end{bmatrix} = T^{-1}\left\{\begin{bmatrix} x_{i,j}^t \\ y_{i,j}^t \end{bmatrix}\right\} = \begin{bmatrix} x_{i,j}^t \\ y_{i,j}^t \end{bmatrix} + \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

Both the **grid generator** and the **sampler** are parameter less operations, i.e. they don't have any trainable parameters. In this regard they are comparable to a max-pooling layer. The brainpower of a spatial transformer module hence comes from the **localisation net**, which must learn to detect the pose of the input feature map (such as its orientation, scale etc.) in order to produce an appropriate transformation.

Complete Module

Finally, let us take a look at how the single building blocks of the spatial transformer module interact with each other. The input feature map U is first passed to the **localisation network**, which regresses the appropriate transformation parameters θ. The **grid generator** then uses the transformation parameters θ to produce the **sampling grid**, which is a set of points where the input feature map shall be sampled. Finally, the **sampler** takes both the input feature map and the **sampling grid** and using e.g. bilinear interpolation outputs the transformed feature map.



Spatial transformer module (Image by author)

At this point we would like to call again attention to the fact, that the **localisation net** predicts the transformation parameters individually for each input. In this manner, the spatial transformer module becomes an adaptive component, whose behavior is conditional on the input.

Multiple Channels

So far, we have demonstrated the principles of the spatial transformer module on inputs with a single channel $C=1$, as encountered in e.g. grayscale images. However, oftentimes spatial transformer modules are used in deeper layers and operate on feature maps, which usually have more than one channel $C > 1$. Even when used immediately following the input, spatial transformer modules may face inputs with more than one channel, such as RGB images which have 3 channels.

The extension is simple: for multi-channel inputs, the mapping is done identically for each channel of the input, so every channel is transformed in an identical way. This way we preserve spatial consistency between channels. Note, that spatial transformer modules do not change the number of channels C, which remains the same in input and output feature maps.

We come to the end of the third post. By now you are familiar with the two cornerstones of the spatial transformer module: reverse mapping and bilinear interpolation. You know all the building blocks of the spatial transformer module and how they interact with each other. The numerous animations should have helped you to develop a strong mental concept. You are now all ready to use spatial transformer modules in automatic differentiation frameworks such as TensorFlow or PyTorch.

The next and last post is primarily meant for advanced readers, who wonder how bilinear interpolation could possibly be a differentiable operation. We will have a detailed look at how gradients flow through the **sampler** back not only to the input feature map, but also to the sampling grid coordinates.