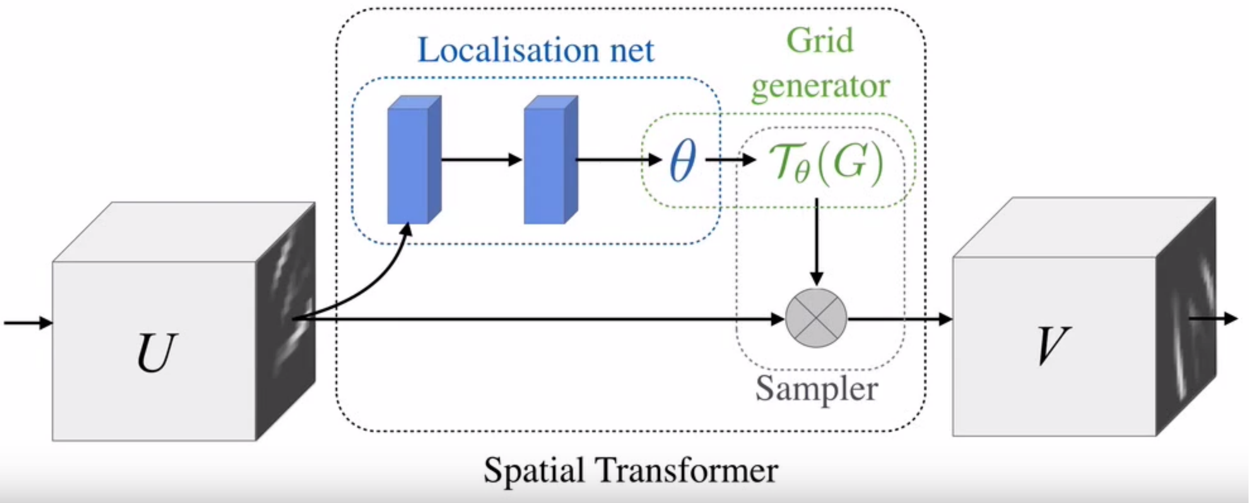
Some notes on
Spatial Transformer Networks

by
Dr. Maryam Khanian

Spatial Transformer Networks (STNs)

(A combination of a *localization network*, a *grid generator* and a *sampler*)

(A network can learn to be invariance to the input data transformations by using one or some spatial transformer networks in sequence or parallel without making any changes to the loss function)



A learnable (differentiable) module that explicitly allows the spatial manipulation of data within the network. It can be inserted into convolutional architecture giving them the ability to actively spatially transform feature maps results in models which learn invariance to translation, scale, rotation and more generic warpings.

Local max-pooling layers allow a network to be somewhat spatially invariant to the position of features. But this spatial invariance is only realise over a deep hierarchy of max-pooling and convolutions in such a way that intermediate feature maps in a CNN are not actually invariant to large transformations of the input data.

STNs vs. pooling layers

Unlike *pooling layers*, where *the receptive fields are fixed and local*, the ***spatial transformer module*** is a ***dynamic mechanism*** that can actively spatially transform an image (or a feature map) by producing an ***appropriate transformation*** for each input sample. The transformation is then performed on the entire feature map (non-locally) and can include scaling, cropping, rotations, as well as non-rigid deformations. This allows networks which include spatial transformers to not only select regions of an image that are ***most relevant (attention),*** but also to transform those regions to a canonical, expected pose to ***simplify recognition*** *in the following layers*.

In this work, authors aim to ***achieve invariant representation by manipulating the data*** rather than feature extractors that can be seen as a ***generalization of differentiable attention to any spatial transformation***.

Three (3) parts of the spatial transformer mechanism:

1. **Localization network:**

   That takes the *input feature map (U)* , and through *a number of hidden layers* outputs the *parameters of the spatial transformation* ***(Θ)*** (that should be applied to the feature map yielding a ***transformation conditional on the input***):

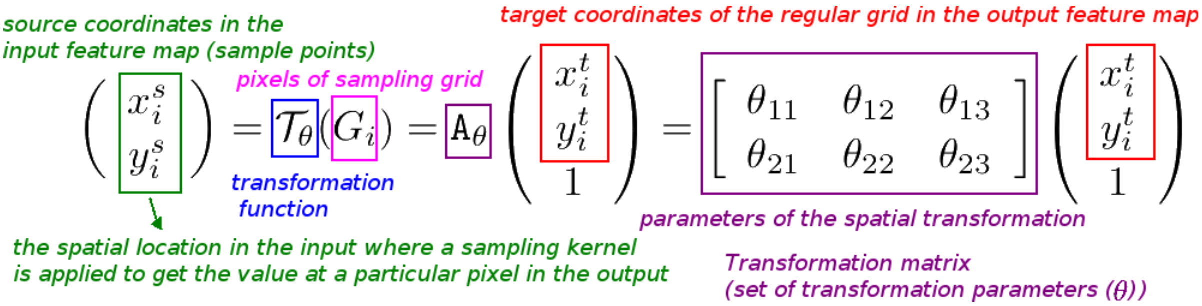
   $$\Theta = f_{loc}(U)$$

   - The size of θ can vary depending on transformation type.
   - The ***transformation*** itself is shown by $T(\theta)$.
   - The *localization network* should include a final ***regression layer*** to produce the transformation parameters *θ.*

2. **Grid generator:**

   Uses the *predicted transformation parameters* to create a *parameterized sampling grid.* (A set of point where the input map should be sampled to produce the transformed output “see below”)

In this step, each output pixel (in general, an element of a generic feature map) is computed by applying a *sampling kernel* centered at a particular location in the input feature map *U*. The output pixels are defined to lie on a regular grid *G* of pixels forming an output feature map *V* with the same number of channels with the input.
For example, for an *affine transformation* we have:


source coordinates in the input feature map (sample points)
target coordinates of the regular grid in the output feature map
pixels of sampling grid
transformation function
parameters of the spatial transformation
the spatial location in the input where a sampling kernel is applied to get the value at a particular pixel in the output
Transformation matrix (set of transformation parameters (θ))

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

That allows cropping, translation, rotation, scale, and skew to be applied to the input feature map, and requires only 6 parameters.

In an ***attention mechanism*** *parameters of the spatial transformation* are:

$$\mathtt{A}_\theta = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \end{bmatrix}$$

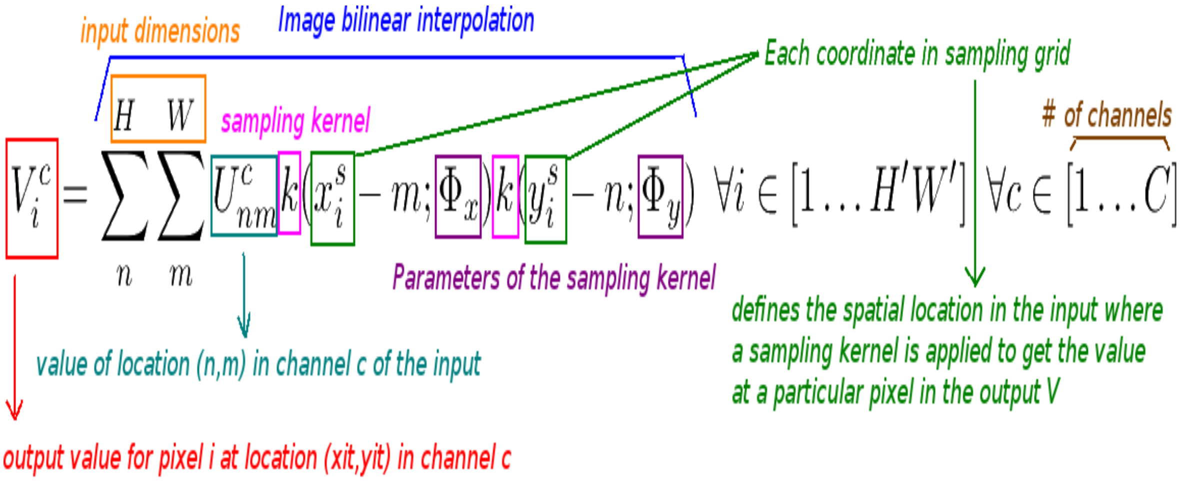
allowing cropping, translation, and isotropic scaling by varying s, t_x , and t_y .

The transformation can have *any parameterised form*, provided that it is ***differentiable with respect to the parameters***. This crucially allows gradients to be backpropagated through from the sample points $\mathcal{T}_\theta$ (G_i) to the localization network output θ.

If the transformation is parameterised in a *structured, low-dimensional way*, this reduces the *complexity of the task assigned to the localisation network*. For instance, if a generic class of structured and differentiable transformations is $\mathcal{T}_\theta = M_\theta B$, it will be possible to not only learn *how to predict θ* for a sample, but also *to learn B for the task at hand*.

3. **Sampler:**
   Having the *input feature map (U)* and the *sampling points* T_θ (G) as inputs, it produces the ***output map (V) sampled from the input at the grid points***.

   Here, the ***sampling kernel*** can be written as:



$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \ \forall i \in [1 \ldots H'W'] \ \forall c \in [1 \ldots C]$$

input dimensions
Image bilinear interpolation
Each coordinate in sampling grid
sampling kernel
# of channels
Parameters of the sampling kernel
defines the spatial location in the input where a sampling kernel is applied to get the value at a particular pixel in the output V
value of location (n,m) in channel c of the input
output value for pixel i at location (xit,yit) in channel c

For example for **Bilinear sampling kerrnel** we have: $k(d) = \max(0, 1 - |d|)$

Based on that the operation will be: $V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$

As we have in DenseCap: $V_{c,i,j} = \sum_{i'=1}^{W} \sum_{j'=1}^{H} U_{c,i',j'} k(i' - x_{i,j}) k(j' - y_{i,j})$

To allow backpropagation of the loss through this sampling mechanism we can define the gradients with respect to U and G:

For *Bilinear samplin*g the partial derivatives are:

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

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

And similarly for the *y* direction.

This gives us a sub-differentiable sampling mechanism, allowing loss gradients to flow back not only to the input feature map *U*, but also to the sampling grid coordinates *G.*

The sampling is done *identically for each channel of the input*, so every channel is transformed in an identical way (this preserves ***spatial consistency*** between channels).

Some important notes about **Spatial Transformer Networks**:

The combination of the localization network, grid generator, and sampler form a spatial transformer.

*This is a self-contained module which can be dropped into a CNN architecture at **any point**, and in **any number**.*

*This module can even **speedups attentive models** due to subsequent downsampling that can be applied to the output of the transformer.*

Placing spatial transformers within a CNN allows the network to learn how to actively transform the feature maps to help minimise the overall cost function of the network during training.

The knowledge of how to transform each training sample is compressed and cached in the weights of the localization network (and also the weights of the layers previous to a spatial transformer) during training.

For some tasks, it may also be useful to feed the output of the localisation network, θ, forward to the rest of the network, as it explicitly encodes the transformation, and hence the pose, of a region or object.

*It is also possible to use spatial transformers to **downsample** or **oversample** a feature map, as one can **define the output dimensions to be different to the input dimensions**.*

*However, with sampling kernels with a fixed, small spatial support (such as the bilinear kernel), downsampling with a spatial transformer can cause **aliasing** effects.*

*Finally, it is possible to have **multiple spatial transformers** in a CNN. Placing multiple spatial transformers **at increasing depths** of a network allow transformations of increasingly abstract representations, and also gives the localisation networks potentially more informative representations to base the predicted transformation parameters on.*

*One can also use multiple spatial transformers **in parallel**. This can be useful if there are **multiple objects or parts of interest** in a feature map that **should be focussed on individually**. (for example having different information in different channel)*

*A limitation of this architecture in a purely feed-forward network is that the **number of parallel spatial transformers limits the number of objects that the network can model**.*

*The ST-CNN (combination of the spatial transformer networks and convolutional neural networks) is able to discover and learn **part detectors** in a data-driven manner without any additional supervision.*