

# Spatial Transformer Networks

- [[📄 1506.02025v3.pdf]]

- **Basic Idea:**

- Can we achieve spatial invariance in CNNs in computationally and parameter efficient manner? In other words, can we transform the objects in the input image to a common pose which makes the job for CNN easier?
- Can we let the network focus on specific regions in the image which are most important for image classification (spatial attention)?
- In order to achieve the above, actively predict the parameters of a parametric transformation such as Euclidean, Affine, Projective etc. and apply this transformation to the input image/intermediate feature maps.

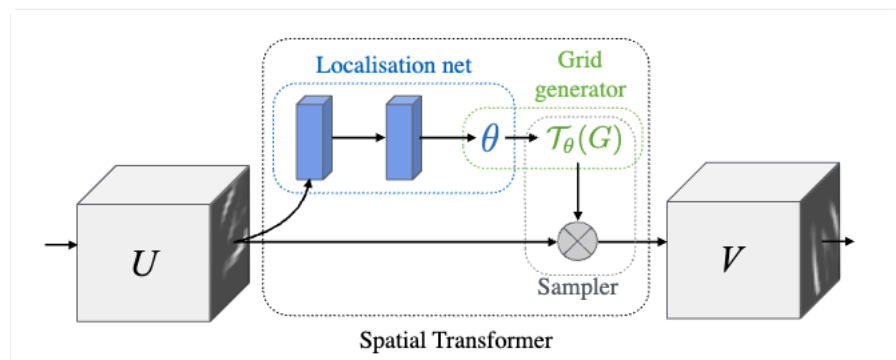
- **Comparison with other CNNs**

- 🟡 **P1** A desirable property of a system which is able to reason about images is to disentangle object pose and part deformation from texture and shape
- *Limited Spatial invariance achieved through Local Max-pooling:* 🟡 **P1** The introduction of local max-pooling layers in CNNs has helped to satisfy this property by allowing a network to be somewhat spatially invariant to the position of features. However, due to the typically small spatial support for max-pooling (e.g.  $2 \times 2$  pixels) this spatial invariance is only realised over a deep hierarchy of max-pooling and convolutions, and the intermediate feature maps (convolutional layer activations) in a CNN are not actually invariant to large transformations of the input data [6, 22].
- Spatial Transformers have the following advantages:
  - Transforms the input globally rather than locally
  - *Spatial Attention:* Allows the network to select regions which are most important to prediction which in turn can allow the downstream network to operate at a lower resolution because you can crop the important region thereby increasing computational efficiency
  - Multiple such parallel spatial transformers can identify multiple such regions in the image
  - Transforms the object in these regions to a common pose to simplify recognition in following layers. For example, 🟡 **P7** Interestingly, the transformation of inputs for all ST models leads to a “standard” upright posed digit – this is the mean pose found in the training data.
  - Transformations are actively learned i.e., are dependent on the input
  - *Co-localisation:* 🟡 **P2** given a set of images containing different instances of the same (but unknown) class, a spatial transformer can be used to localise them in each image
- Think of this kind of spatial attention similar to selective search in RCNN system.

- **Spatial Transformers**

- Components of spatial transformer:

🟡 **P3**



- *Localisation network* : 🟡 **P3** takes the input feature map, and through a number of hidden layers outputs the parameters of the spatial transformation that should be applied to the feature map– this gives a transformation conditional on the input.
- *Grid Generator* : 🟡 **P3** Then, the predicted transformation parameters are used to create a sampling grid, which is a set of points where the input map should be sampled to produce

the transformed output.

- **Sampler** : ● **P3** the feature map and the sampling grid are taken as inputs to the sampler, producing the output map sampled from the input at the grid points
- ● **P3** For multi-channel inputs, the same warping is applied to each channel ✎
- **Localisation Network**:
  - Simple CNN to regress the parameters of the transformation  $\mathcal{T}_\theta$ .

- **Parameterised Sampling Grid**:

- **Notations**:
  - $U \in \mathbb{R}^{H \times W \times C}$ : Input feature map
  - $V \in \mathbb{R}^{H' \times W' \times C}$ : Output feature map
  - $(x_i^t, y_i^t)$ : Denotes pixel coordinates in target (output) feature map
  - $(x_i^s, y_i^s)$ : Denotes pixel coordinates in source (input) feature map
  - $G = \{G_i\}$  where  $G_i = (x_i^t, y_i^t)$ : Regular grid of target coordinates
- Using the parameters predicted by the Localisation network, construct the backward warping map. For example, consider the case of 2D affine transformation defined by the matrix  $A_\theta$ :

● **P4**

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

The coordinates are height and width normalised i.e.,  $-1 \leq x_i^t, y_i^t \leq 1$  and  $-1 \leq x_i^s, y_i^s \leq 1$  when within spatial bounds of output and input respectively. For more information refer to the `torch.nn.functional.grid_sample`.

- If we restrict the  $H' < H$ ,  $W' < W$  and the determinant of the left  $2 \times 2$  sub-matrix to be less than unity, the transformation can be thought of as equivalent to cropping with pose correction. If we just want to achieve spatial attention (cropping), we can have a transformation defined as

● **P4**

$$A_\theta = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \end{bmatrix}$$

- ● **P4** The transformation  $T_\theta$  can also be more general, such as a plane projective transformation with 8 parameters, piecewise affine, or a thin plate spline. ✎

- **Differentiable Image Sampling**

- ● **P4** To perform a spatial transformation of the input feature map, a sampler must take the set of sampling points  $\mathcal{T}_\theta(G)$ , along with the input feature map  $U$  and produce the sampled output feature map  $V$ .
- The sampling can be thought of as a convolution operation. Therefore, ● **P4** Each  $(x_i^s, y_i^s)$  coordinate in  $\mathcal{T}_\theta(G)$  ● **P4** defines the spatial location in the input where a sampling kernel is applied to get the value at a particular pixel in the output  $V$ . This can be written as

● **P4**

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

where

- $\Phi_x$  and  $\Phi_y$ : Parameters of the sampling kernel  $k()$  which define the image interpolation
- $U_{nm}^c$ : Value of the input feature map at  $(n, m)$  in channel  $c$
- $V_i^c$ : Value of the output pixel  $i$  at location  $(x_i^t, y_i^t)$  in channel  $c$
- ● **P5** Note that 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).

- For example,

- *Nearest Neighbour Sampling* can be represented as follows:

● P5

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \delta(\lfloor x_i^s + 0.5 \rfloor - m) \delta(\lfloor y_i^s + 0.5 \rfloor - n)$$

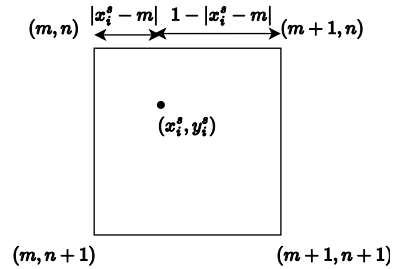
where  $\lfloor x + 0.5 \rfloor$  rounds  $x$  to the nearest integer and  $\delta()$  is the Kronecker delta function. This is equivalent to copying the value at nearest pixel to  $(x_i^s, y_i^s)$  and pasting it to the output location  $(x_i^t, y_i^t)$ . This is because the both the  $\delta()$  functions will be equal to 1 when  $\lfloor x + 0.5 \rfloor$  and  $\lfloor y + 0.5 \rfloor$  are equal to  $m$  and  $n$  respectively.

- *Bilinear Sampling* can be represented as follows:

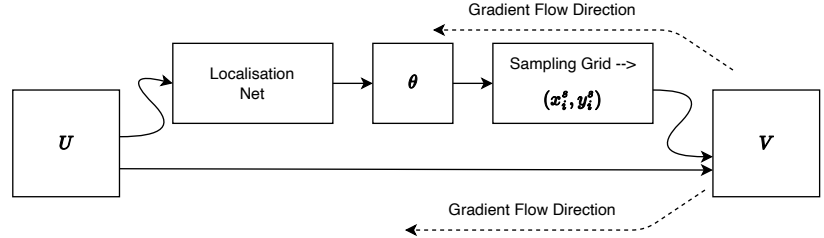
● P5

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

To understand why this is true, consider the following



- To understand the how to obtain derivatives for this sampling mechanism, consider the following computational graph



- Therefore, we require gradients of  $V$  wrt  $x_i^s, y_i^s$  (sampling grid coordinates) and  $U$  (input feature map). Once,  $\frac{\partial V_i^c}{\partial x_i^s}$  is computed then we can easily compute  $\frac{\partial x_i^s}{\partial \theta}$  because both are related through a simple matrix multiplication for example in affine/projective transformation.
- The gradients for bilinear sampling are defined as

● P5

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

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

- Notes:

- ● P5 Due to discontinuities in the sampling functions, sub-gradients must be used.
- ● P5 This sampling mechanism can be implemented very efficiently on GPU, by ignoring the sum over all input locations and instead just looking at the kernel support region for each output pixel.

- Ways to integrate/use spatial transformer blocks (Full details of the architecture can be found in the paper):

- Can be used in the beginning of existing CNNs (between the input and first convolutional layers to downsample as well as extract important region. For example, **P15** each transformer predicts the location (x,y) of the attention window, while the scale is fixed to 50% of the image size.
- **P5** For some tasks, it may also be useful to feed the output of the localisation network,  $\theta$ , forward to the rest of the network, as it explicitly encodes the transformation, and hence the pose, of a region or object
- **P5** It is also possible to use spatial transformers to downsample or oversample a feature map, as one can define the output dimensions  $H'$  and  $W'$  to be different to the input dimensions  $H$  and  $W$ . However, with sampling kernels with a fixed, small spatial support (such as the bilinear kernel), downsampling with a spatial transformer can cause aliasing effects.
- **P6** 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.
- **P6** 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. 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. For example, **P8** one spatial transformer (red) has learnt to become a head detector, while the other (green) fixates on the central part of the body of a bird. The resulting output from the spatial transformers for the classification network is a somewhat pose-normalised representation of a bird
- **3D Transformer**: For performing transformations on 3D inputs. For example, the spatial transformers can also learn to perform 3D affine transformations.
- **P13** Another interesting way to use the 3D transformer is to flatten the 3D output across one dimension, creating a 2D projection of the 3D space, e.g.  $W_{nm}^c = \sum_l V_{nml}^c$  such that  $W \in \mathbb{R}^{H' \times W' \times C}$ . **P13** This allows the original 3D data to be intelligently projected to 2D, greatly reducing the dimensionality and complexity of the subsequent processing.
- **P4** For instance, a generic class of structured and differentiable transformations, which is a superset of attention, affine, projective, and thin plate spline transformations, is  $T\theta = M\theta B$ , where  $B$  is a target grid representation (e.g. in (10),  $B$  is the regular grid  $G$  in homogeneous coordinates), and  $M\theta$  is a matrix parameterised by  $\theta$ . In this case it is possible to not only learn how to predict  $\theta$  for a sample, but also to learn  $B$  for the task at hand.

- **Tricks of the trade**

- **Initialisation**: Initialise the weights of the regression layer of the localisation network to predict identity transform (weights are set to zero, bias term is set to identity for identity transform).
- **Lower learning rate for localisation net**: **P14** The learning rate for localisation networks of spatial transformer networks was set to a tenth of the base learning rate. **P15** For stability, the localisation network's learning rate is the base learning rate multiplied by 10–4.


- **Co-localisation Experiment**

- **P11** The co-localisation task is as follows: given a set of images that are assumed to contain instances of a common but unknown object class, localise (with a bounding box) the common object. Neither the object class labels, nor the object location ground truth is used for optimisation, only the set of images.
- **Idea**: **P11** distance between the image crop corresponding to two correctly localised objects is smaller than to a randomly sampled image crop, in some embedding space.
- This can be achieved using a triplet loss:

● P12

$$\sum_n^N \sum_{m \neq n}^M \max(0, \|e(I_n^T) - e(I_m^T)\|_2^2 - \|e(I_n^T) - e(I_n^{\text{rand}})\|_2^2 + \alpha)$$

where,

- $\mathcal{I} = \{I_n\}$  is the dataset of  $N$  images
- $I_n^T$  is the image crop of  $I_n$  corresponding to the localised object
- $I_n^{\text{rand}}$  is a randomly sampled patch from  $I_n$
- $e()$  is the encoding function
- $\alpha$  is the margin
- The spatial transformer can be used as a localiser, such that  $I_n^T = \mathcal{T}_\theta(I_n)$  where  $\theta = f_{\text{loc}}(I_n)$ , interpreting the parameters of the transformation  $\theta$  as the bounding box of the object.
- ● P12 We can minimise this with stochastic gradient descent, randomly sampling image pairs  $(n, m)$ . 
- This can be further used in object tracking problems.

• **Food for thought:**

- Most of the components in a typical CNN are applied on the pixel intensities (except max-pooling). Can we apply transformations on the spatial domain such that pixels can move around spatially?
- Since we are transforming the input images spatially (i.e., moving pixels around), can these kind of spatial transformer networks struggle against dense prediction tasks such as semantic segmentation, depth estimation etc?
- One of the downsides of using a spatial transformer with high resolution images is that the localisation network has to be deep enough to predict the parameters of the transformation  $\mathcal{T}$ . This increases the overall capacity of the entire network, leaving the network exposed to the chances of severe overfitting.
  - A sub-thought which arises is can we design networks which can quickly reduce downsample the image (shallow networks) so as to predict the global properties of the image?
  - Another way to overcome this challenge could to be share weights between the localisation network and the downstream classification network.
- In a typical CNN, max-pooling is generally used to downsample the image. Can we use simplify spatial transformers and use them as the basic operation in CNNs for downsampling? This is because max-pooling is a straight forward operation whereas spatial transformation is much more informed and actively (depends on the input) determined.