Spatial Transformer Networks

Overview

Spatial Transformer Networks are Convolutional Neural Networks, that contain one or several Spatial Transformer Modules. These modules attempt to make the network spatially invariant to its input data, in a computationally efficient manner, which leads to more accurate object classification results. Further they allow the localization of objects in an image and a sub-classification of an object, such as distinguishing between the body and the head of a bird in an unsupervised manner.

1 Overview

1.1 Introduction

2D Geometric Transformations

1.2 Translational Invariance through the Max-Pooling Layer

2 Spatial Transformer Module

2.1 Overview

2.2 Localization Network

2.3 Grid Generator

2.4 Sampler

2.5 Performance Compared to Standard CNNs

2.6 Unsupervised Sub-Object Classification

2.7 Problems and Limitations

3 Literature

4 Weblinks

Introduction

Current CNNs are only somewhat invariant in translation through the use of a max-pooling layer.\(^1\) A key requirement of a CNN is that it is able to correctly classify objects in real world image data. Objects in images are usually at random positions and taken from random viewpoints at different scales. Therefore, a CNN has to be implemented in a way, that the output of the network is invariant to the position, size and rotation of an object in the image. The Spatial Transformer Module manages to do that, with remarkable results.

2D Geometric Transformations

2D Geometric Transforms are a set of transforms to alter parameters such as scale, rotation and position of an image. The transformation is done by multiplying each coordinate vector of an image with one of the transformation matrices shown in table 1.

The effect of the respective transformations can be seen on the right.

Image 1: Behaviour of the spatial transformer, during 10 training steps. As visible, the spatial transformer is able to remove redundant background data from the image. Image source (2).
Table 1: Hierarchy of 2D geometrical transformations. Image source [2].

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Matrix</th>
<th>#DoF</th>
<th>Preserves</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>translation</td>
<td>$[l</td>
<td>t]_{2x3}$</td>
<td>2</td>
<td>orientation</td>
</tr>
<tr>
<td>rigid (Euclidean)</td>
<td>$[R</td>
<td>t]_{2x3}$</td>
<td>3</td>
<td>lengths</td>
</tr>
<tr>
<td>similarity</td>
<td>$[sR</td>
<td>t]_{2x3}$</td>
<td>4</td>
<td>angles</td>
</tr>
<tr>
<td>affine</td>
<td>$[A]_{2x3}$</td>
<td>6</td>
<td>parallelism</td>
<td></td>
</tr>
<tr>
<td>projective</td>
<td>$[H]_{3x3}$</td>
<td>8</td>
<td>straight lines</td>
<td></td>
</tr>
</tbody>
</table>

The following shows a generic matrix for each of the 2D geometric transformations. In order to perform the transformation in one single multiplication, homogeneous coordinates are used, which means, that a 1 is added as a third dimension, i.e. $\bar{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$.

### Transformation Matrix Entries

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Matrix Entries</th>
<th>$x'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; t_x \ 0 &amp; 1 &amp; t_y \end{bmatrix} \bar{x}$</td>
<td></td>
</tr>
<tr>
<td>Rigid (Rotation + Translation)</td>
<td>$\begin{bmatrix} \cos \theta &amp; -\sin \theta &amp; t_x \ \sin \theta &amp; \cos \theta &amp; t_y \end{bmatrix} \bar{x}$</td>
<td></td>
</tr>
<tr>
<td>similarity (scaled rotation)</td>
<td>$\begin{bmatrix} s \cos \theta &amp; -s \sin \theta &amp; t_x \ s \sin \theta &amp; s \cos \theta &amp; t_y \end{bmatrix} \bar{x}$</td>
<td></td>
</tr>
<tr>
<td>Affine Transformation</td>
<td>$\begin{bmatrix} a_{00} &amp; a_{01} &amp; a_{02} \ a_{10} &amp; a_{11} &amp; a_{12} \end{bmatrix} \bar{x}$</td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>$\begin{bmatrix} h_{00} &amp; h_{01} &amp; h_{02} \ h_{10} &amp; h_{11} &amp; h_{12} \end{bmatrix} \bar{x}$</td>
<td></td>
</tr>
</tbody>
</table>

Translational Invariance Through the Max-Pooling Layer

Max-Pooling is a form of non-linear downsampling, which is an essential part of standard CNNs. The input to the CNN is separated into non-overlapping squares of the same size. Each square is then reduced to its maximum pixel value, while the other values are dropped. For further information on Max-Pooling, please refer to the basics pages. While the Max-Pooling Layer reduces computational complexity in the higher layers of a network, it also provides a form of translation invariance, which will be covered in the following.
The figure above shows the 8 possible translations of an image by one pixel. The 4x4 matrix represents 16 pixel values of an example image. When applying 2x2 max-pooling, the maximum of each colored rectangle is the new entry of the respective bold rectangle. As one can see, the output stays identical for 3 out of 8 cases, making the output to the next layer somewhat translation invariant. When performing 3x3 max-pooling, 5 out of 8 translation directions give an identical result, which means that the translation invariance rises with the size of the max-pooling.

As described before, translation is only the most simple scenario of a geometric transformation. Other transformations listed in the table above can only be handled by the spatial transformer module.

**Spatial Transformer Module**

**Overview**

![Spatial Transformer Module Diagram](image)

**Image 4:** Architecture of a spatial transformer module. U and V are the input and output feature map respectively. The goal of the spatial transformer is to determine the parameters for $\theta$, i.e. the parameters for the geometric transform. Image source [1].

**Image 5:** Illustration of a transformation showing the sampling. While $T_I(G)$ depicts a unitary transformation (i.e. no change at all), $T_\theta(G)$ shows an affine transform. One can see, that the sampling grid produced by $T_\theta(G)$ does not correspond to the pixel coordinates of U. Therefore some form of interpolation has to be performed. Image 5 also shows that the spatial transformer module is also able to crop the image and remove (possibly) redundant information from the image and focusing on key elements of the image. Image source [1].
Localization Network

The localization network transforms the input feature map $\mathbf{U}$, which is shown above into the output parameters $\theta$. The dimension of $\theta$ depends on the type of transformation which the network should be able to perform. As for example a projective transform has 8 degrees of freedom, the dimension of $\theta$ would have to be 8. While the localization network can be either a CNN or a classic neural network, it should contain a final regression layer, in order to output the parameters $\theta$. In general any parameterizable transformation, that is differentiable towards its parameters can be used in a STN.

Grid Generator

The grid generator $T_\theta(G)$ uses the transformation parameters to create a sampling grid, which is a set of points $(x_i', y_i')$. One of these points defines, where a sampling kernel has to be applied on $\mathbf{U}$ in order to obtain a certain output pixel in $\mathbf{V}$.

Sampler

The sampler combines the input feature map and the sampling grid, resulting in the output feature map $\mathbf{V}$, by performing some form of interpolation. The interpolation step is necessary as the coordinates of the sampling points $(x_i', y_i')$ in general won’t be existing coordinates of the input $\mathbf{U}$.

The sampling step can be performed with any kernel, while the general formula is

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

Here $k(\cdot)$ can be any kind of sampling method, with $\phi_x$ and $\phi_y$ being the sampling parameters. The only constraint for the sampling kernel is, that it is differentiable with respect to the sampling points $(x_i', y_i')$. This is necessary, for the backpropagation algorithm, that is used for training the spatial transformer network. In their paper, Jaderberg et al are using a bilinear sampling kernel. This changes the above equation to

$$V_{ic} = \sum_n \sum_m U_{nm}^c \max(0, 1 - |x_i' - m|) \max(0, 1 - |y_i' - n|).$$

For performing backpropagation, one has to find the gradients for $\mathbf{U}$ and $G$, which brings us to the equations

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

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

$$\frac{\partial V_i^c}{\partial y_i'} = \sum_n \sum_m U_{nm}^c \max(0, |x_i' - m|) \begin{cases} 0 & \text{if } |y_i' - m| \geq 1 \\ 1 & \text{if } m \geq y_i' \\ -1 & \text{if } m < y_i' \end{cases}.$$
These equations, allow the loss gradients to backpropagate to the input feature map $U$ of the transformer module on one hand, and to the transformation parameters $\theta$ of the ST-layer by using the further derivatives $\frac{dy}{d\theta}$ and $\frac{dy}{d\theta}$.

Performance Compared to Standard CNNs

While providing state-of-the-art results, the computation time of the Spatial-Transformer-CNN introduced by Jaderberg et. al is only 6% slower than the corresponding standard CNN.\(^1\)

The table in image 6\(^1\) shows the comparison of the results from different neural networks on the MNIST data-set. The table distinguishes between fully convolutional networks (FCN) and convolutional neural networks (CNN). It further includes a spatial transformer module to each of the network types (ST-FCN and ST-CNN) with $\theta$ containing either 6 (Aff) or 8 (Proj) transformation parameters or a thin plate spline (TPS). The MNIST data-set was distorted by rotation (R), rotation, translation and scaling (RTS), a projective transform (P) or an elastic distortion (E). The results show, that each spatial transformer network outperforms its standard counterpart, as the classification error is smaller in all cases.

Unsupervised Sub-Object Classification

Image 7: The table shows the classification results of different network architectures for the CUB-200-2011 bird data set. The spatial transformer networks are able to outperform the other networks. The images on the left show the behaviour of the spatial transformers in the network, when 2 (upper row) or 4 (lower row) are in parallel inside the network. Interestingly, each of the spatial transformers learned to focus on either the head or the body of the bird, in a completely unsupervised manner. Image source [1].
Another important discovery from Jaderberg et al is, that they achieved a form of unsupervised and fine-grained classification. The presented experiments were done on the CUB-200-2011 bird data set which contains images of 200 different bird species. These bird images not only consist of different species, but are taken from different angles and points of view, with different scaling and individual background scenery. This gives an idea, how challenging this dataset is.

Previous to the introduction of the spatial transformer module Simon and Roder also performed unsupervised fine-grained classification on this bird data set. By analyzing the constellation of part detectors, that fire at approximately the same relative location from each other they achieved a state-of-the-art classification accuracy of 81.0%. The table in Image 7 shows, that Jaderberg et al were able to improve this result by 1.3 percentage points using a CNN model with Inception architecture.[1] Given the latter network as a basis, they inserted either 2 or 4 spatial transformer modules into the network architecture, achieving even higher classification results. The images next to the table on the right show the region of interest on which each of the transformer modules focused. This shows that when spatial transformer modules are put in parallel, each can learn a different part of an object. In this case, one of the transformer modules focused on the head and the other on the body of a bird. Other work on this data set with sub-object classification was done by Branson et al. While the latter explicitly defined parts of the bird, and trained separate detectors on these parts, Jaderberg et al achieved this separation in a completely unsupervised manner.

Problems and Limitations

When using a spatial transformer, it is possible to downsample, or oversample a feature map.[1] Using a sampling kernel such as the bilinear kernel which is of fixed width, can cause aliasing effects in the output feature map.

As described in the last chapter, spatial transformer modules can be used for fine-grained classification, which means, that also sub-parts of a class can be detected by a neural network. While this is a very promising result, the number of objects a STN can model is limited to the number of parallel spatial transformers in the network.[1]

Literature


Weblinks

(1) http://deeplearning.net/tutorial/lenet.html
(2) http://torch.ch/blog/2015/09/07/spatial_transformers.html

(3) http://yann.lecun.com/exdb/mnist/