# Spatial Transformer Networks:

# Increasing Spatial Invariance Through

# Rotation and Scaling

William Kimball

University of Virginia

**Background:**

Spatial transformer networks (STN) are known to be a specific type of convoluted neural network (CNN). CNNs are often seen alongside classification and computer vision tasks. There are three primary elements to a CNN, the convolutional layer, the pooling layer, and the fully connected layer. We'll explore all three categories of the CNN, as they are essential to understanding the importance of the STN on the output of the CNN.

The convolutional layer is the cardinal and core building block of the network. Its contents consist of input data, a filter, and a feature map. The filter, commonly known as a feature detector, is a 2D array of weights which are applied to a specific part of the image (input data). The filter then repeatedly shifts across the image, scanning the kernel until the entire image has been processed. The intermediate calculation between every shift is a dot product between the input pixels and the filter. The cumulation of dot products results in a feature map, commonly known as a convolved feature. Weights in the feature detector remain fixed throughout the process, with some parameters like weight values adjusting through backpropagation and gradient descent. Backpropagation is a fundamental building block of neural networks, which effectively performs a backwards pass following each forward pass in an algorithm to appropriately adjust weights and biases. Although a very vital component of creating accurate deep learning models, it won't be necessary to go into depth on this subject when attempting to better understand the structure of the STN [8].

Furthermore, the feature detector does require a few specialized components for optimized performance. The first of these hyperparameters is the number of filters which primarily impacts the depth of the output. The stride is defined as the number of pixels the kernel moves over the input matrix with every shift of the filter. The last component consists of multiple different approaches to padding, the most popular of which being zero-padding. Zero-padding accounts for filters that do not fit the input image, ensuring all values outside of the filters boundaries are set to zero. The three other options include valid padding, same padding, and full padding [8].

An additional convolutional layer will often follow the initial layer, and this additional layer will convert the image into numerical values, allowing the neural network to interpret and extract relevant patterns. This additional layer perfectly embodies the naturally hierarchical identity of CNNs [8].

The succeeding layer is known as the pooling layer. This layer will reduce the number of parameters in the input data, sweeping a filter void of weights across the entire input. Two different approaches are commonly used for pooling, the most common of which being maximum pooling. Maximum pooling is the process by which the pixel with the maximum value is selected to be sent to the output array. Instead of using maximum pooling, some scientists will use average pooling which calculates the average value within the receptive field to send to the output array. Therefore, the pooling layer reduces complexity, improves efficiency while simultaneously limiting the risk of overfitting. Essentially, overfitting occurs when the network learns too much from the training data, and becomes too complex for the task at hand, thus poorly representing the data. To combat this dilemma, pooling will provide necessary simplification to the process [3].

Subsequently, the fully-connected (FC) layer will lead to the culmination of the network. Understanding the complexity of the FC layer is quintessential when integrating a STN because it includes the calculations which directly correlate with the final decisions of the model. Each node in the output layer connects directly to a node in the previous layer, which is a key part of classification tasks. The FC layer consists of a combination of linear transformations and a softmax function. Each neuron in a fully connected layer performs a linear transformation on its output.

$$y_{jk}(x) = f(\sum_{i=1}^{n} w_{jk}x_i + b_{j0})$$

X is observed as the input vector, W is the weight matrix, and B is the bias vector. Thus leaving Y to be the output vector.

The softmax function is then employed for calculating probabilities. The values are converted into probabilities using the following equation.

$$\text{softmax}(y_i) = \frac{e^{yi}}{\sum_j e^{yj}}$$

Yi is the output of the i-th neuron and the denominator is the sum of exponential values of all outputs in the layer, thus ensuring the probability lands somewhere between zero and one. The configuration of the FC layer is essential to understanding and interpreting its outputs. The output layer has as many neurons as there are classes. The output of each neuron represents the network’s confidence level, or the probability that the given input will be a member of the corresponding class being analyzed. The “argmax” operation is often used in programming because it determines the predicted class is the one corresponding to the neuron with the highest softmax output. A higher value in the argmax operation indicates higher confidence. Essentially, the class with the highest probability is classified as the model's final prediction [4].

Although the calculations in the FC are very intricate and provide ample insight toward prediction, there are a few concerns to be addressed. Overfitting is a primary concern when creating deep learning networks as they can often be overconfident in their predicted values.

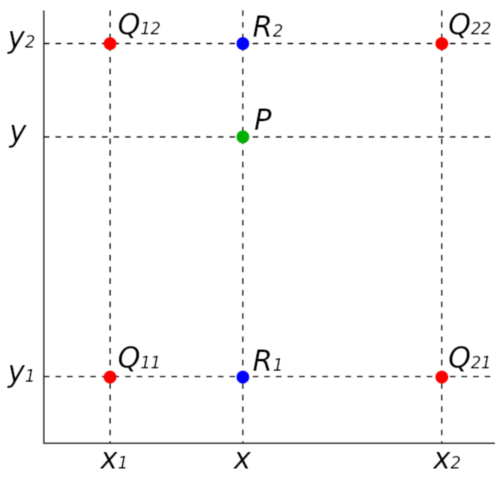
Calibration misconceptions based upon the concept that high softmax probabilities correlate directly to high model accuracy can also be a major issue in the interpretation stage. The performance of the neural network is also heavily dependent on the quality of the input data, thus making the margin for human error in selection higher. To successfully create an accurate predictive model, it's essential to avoid these errors [3].

Moreover, there are many methods that diminish the frequency of encountering these concerns. However, spatial transformer networks are, when it comes to convolutional neural networks, one of the best approaches.

**Spatial Transformer Networks:**

Now that we understand the processes behind a CNN, we can dive deeper into the functionality and purpose of a STN. STNs can be inserted into existing CNNs either immediately following the input or in deeper layers. They achieve spatial invariance through continually transforming input data into a specified, expected pose, thus leading to better classification performance. However, before we begin this analysis, we must look into a key aspect of the STN: bilinear interpolation.

Bilinear interpolation is a resampling method that uses the distance-weighted average of the four nearest pixel values to estimate a new pixel value.



Examining the vector example to the left [10] will provide a better understanding of the process. The point P is our desired point, the points Q are our known points, and the points R are the points on the line with the known points and desired point. Initially, point R1 is defined as $R_1(x,y) = Q_{11} \cdot (x_2 - x) / (x_2 - x_1) + Q_{21} \cdot (x -$

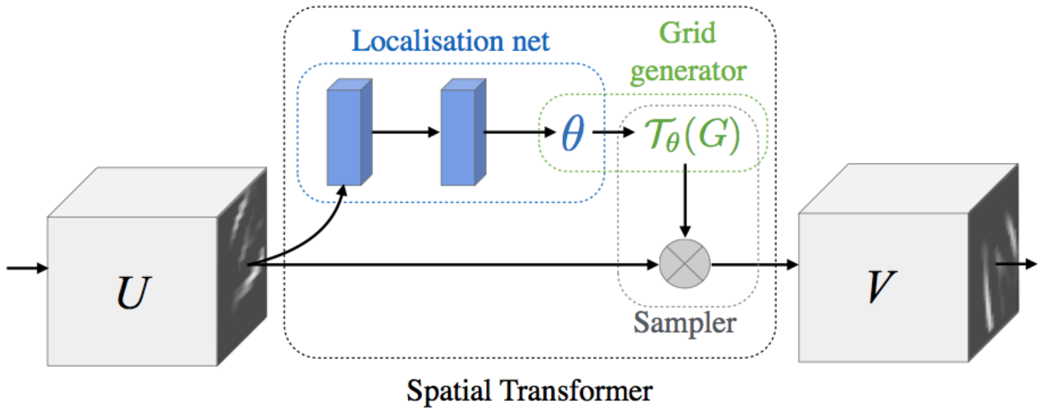
$x_1$) / ($x_2 - x_1$). The point R2 is defined as $R_2(x, y) = Q_{12} \cdot (x_2 - x) / (x_2 - x_1) + Q_{22} \cdot (x - x_1) / (x_2 - x_1)$. Using both of these points' values, we can then calculate the desired point as $P(x, y) = R_1 \cdot (y_2 - y) / (y_2 - y_1) + R_2 \cdot (y - y_1) / (y_2 - y_1)$. By continually calculating this desired point in the input data, the network will construct a transformed version of the input image. The details of bilinear interpolation's integration in the STN will be further discovered later on in the paper [9].

Transitioning into the fundamental structure of a STN, we must first look at the concept of reverse mapping. Reverse mapping is a method of separating responsibilities within the network, that ultimately facilitates a quicker and smoother backpropagation process. The output image is traversed, pixel by pixel, and for each position two different operations are performed. The first of which is using the inverse transformation to calculate the corresponding position in the input image. Sequentially, each pixel value will be sampled using bilinear interpolation. The inverse transformation will be applied to each pixel first, followed by the sampling. This creates two distinct and separate components: a grid generator and sampler. The grid generator has the responsibility of calculating inverse transformations while the sampler enacts bilinear interpolation [7].

However, before the concept of reverse mapping is utilized, the input image will be passed into the localization network. This network will take an input feature map $U \in R^{H \times W \times C}$ where H is the height of the grid, W the width and C is the number of channels. The outputs of the localization network will be θ, and the equation used to determine this parameter will be $\theta = floc(U)$. The size of θ will vary depending on the type of transformation, but in all localization networks there must be a final regression layer that produces the transformation parameters θ [7].

The grid generator follows the localization network, effectively producing a grid of coordinates that maps the pixel from the input image to the transformed output. A transformation matrix will often be inputted into a grid generator, and is learned from the localization network, which has already determined the spatial transformation that will be employed. A normalized coordinate grid of the same dimensions as the output image will be produced next. Each point in this grid will represent a coordinate in the output space. The coordinate grid is then transformed using the prior transformation matrix, which generates a new grid that specifies each output pixel's location in the input image that it corresponds to. This transformation matrix often encodes certain spatial transformations including translation, rotation, scaling and shearing. When the matrix multiplication is applied, the points in the grid will shift according to the specific spatial transformation [7].

Following the generation of the sampling grid, the sampler begins to perform bilinear interpolation. The interpolated values create the transformed output feature map, which accurately reflects the spatial transformations learned by the network. The sampler, as well as the localization network and grid generator, are all differentiable operations. This is critical to the success of the CNN as a whole, as it ensures the gradient can flow through every module in the network allowing for smoother transformations [7].



**Application:**

The code application of the STN, as well as all the comments on its functionality, are provided within my repository on GitHub [5].

## Works Cited

[1] Chatgpt - openai. (n.d.). https://openai.com/chatgpt

[2] DeepAI. (2019, May 17). *Spatial Transformer Network*. https://deepai.org/machine-learning-glossary-and-terms/spatial-transformer-network#:~:text=In

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2023). *Deep learning*. Alanna Maldonado.

[4] Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2016, February 4). *Spatial Transformer Networks*. arXiv.org. https://arxiv.org/abs/1506.02025

[5] Kevinzakka (2017), spatial-transformer-network. GitHub. https://github.com/kevinzakka/spatial-transformer-network/blob/master/stn/transformer.py

[6] Kostadinov, S. (2019, August 12). *Understanding backpropagation algorithm*. Medium. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

[7] Kurbiel, T. (2021, October 21). *Spatial Transformer Networks*. Medium. https://towardsdatascience.com/spatial-transformer-networks-b743c0d112be

[8] *What are convolutional neural networks?*. IBM. (n.d.). https://www.ibm.com/topics/convolutional-neural-networks

[9] Wijenayaka, S. (2023, July 6). *[bitesize ML] bilinear interpolation*. Medium. https://ai.plainenglish.io/bitesize-ml-bilinear-interpolation-d732b799fc4e?gi=81b0011cfc74

[10] X-Engineer.org. (n.d.). *Home*. x. https://x-engineer.org/bilinear-interpolation/