Spatial Transformer Networks

Mason Sun (zs2321@columbia.edu)
Hung-Shi Lin (hl2997@columbia.edu)

ECBM 4040 (Fall 2016) – Final Project Report
Columbia University

Abstract

Despite the tremendous success of convolutional neural networks, these models are still limited by their inability to be geometrically invariant to the input data. In this paper, we replicate a popular module called the spatial transformer, which is a learnable, self-contained module that enables the explicit spatial transformation of features within the network. We demonstrate that inserting spatial transformers into neural networks allows models to learn invariance towards scaling, translation, rotation, and warping, all without introducing a significant computational overhead. In addition, incorporating spatial transformers successfully increased accuracy over their baseline models on several benchmark datasets, thus signaling their ability to achieve state-of-the-art performances.

1. Introduction

Convolutional neural networks (CNNs) have seen recent success in solving pertinent problems related to computer vision, achieving state-of-the-art performances on tasks such as image classification, segmentation, and localization.

A critical challenge in object recognition is developing representations that are invariant to transformations (e.g. translation, rotation, scaling, and generic warping). For instance, an image of a person should be recognizable regardless of the person's spatial orientation, location, or background texture. Although local max-pooling layers of CNNS allow networks to be quasi-geometrically invariant to feature positions, this invariance is only achieved over many layers of convolution and max-pooling [2]. This is mainly attributed to the small, predetermined spatial support of the pooling mechanism that deals with spatial variations of the data. Furthermore, feature maps in intermediate layers are not actually invariant to substantial transformations of the input data [3].

To address these issues, we aim to replicate the spatial transformer (ST), a dynamic, self-contained module that can be inserted into any standard neural network architecture to facilitate the capacity for spatial transformations of feature maps. STs can be trained with standard back-propagation without extra supervision, and can produce appropriate transformations for each input sample. Thus, this not only allows networks to autonomously select appropriate regions of interests, but also transforms such regions into canonical orientations to simplify recognition in subsequent layers [2].

To substantiate the ability of spatial transformers, we train it on two benchmark datasets: distorted MNIST and Street View House Numbers (SVHN). The former is the canonical MNIST dataset. However, images are spatially transformed (e.g. rotation, translation, scaling, projection, and warping), and the transformed ground truths are excluded. The latter is a more challenging real-world dataset, which contains house numbers within natural scenes. We subsequently compare the performances of fully-connected networks (FCNs) and CNNs that include spatial transformers to their respective baseline networks, demonstrating that inclusion of spatial transformers increase model performances across the board.

2. Summary of the Original Paper

In this section, we first examine the methodology used in the original paper, describing the spatial transformer and its implications in brevity. We then describe the key results and conclusions of the original paper, mostly noting the improved performances of spatial transformers on the two previously mentioned benchmark datasets. In addition, we will also list some noteworthy insights of the original paper that is beyond the scope of this report.

2.1 Methodology of the Original Paper

We first describe the components of a spatial transformer. A spatial transformer is a "differentiable module that applies a transformation to a feature map during a single forward pass"

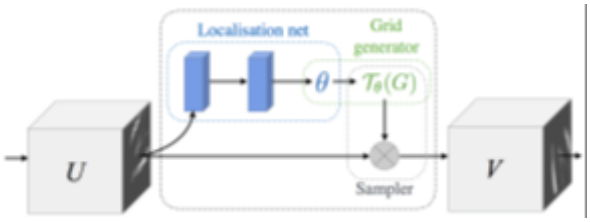

Figure 1: Spatial transformer, replicated from [2]. The input feature map U is passed to a localization network which regresses the transformation parameters θ. The regular spatial grid G over V is transformed to the sampling grid $T_\theta(G)$, which is applied to U, producing the warped output feature map V. The localization net, grid generator, and sampler constitute a spatial transformer.

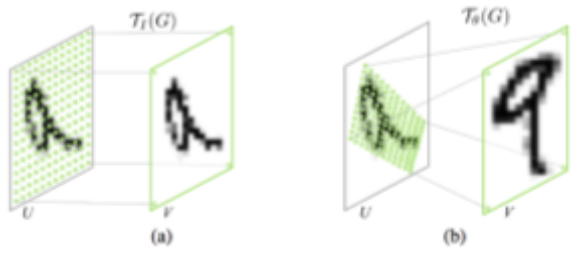

Figure 2: Examples of applying a sampling grid to an image to produce its transformed output. (a) Regular grid $T_I(G)$ using the identity transformation parameters. (b) Warping the regular grid with an affine transformation $T_\theta(G)$.

[2]. It is composed of three sequential parts, as illustrated in Fig. 1. First and foremost, the input feature map is passed into a *localization network*, which computes the parameters that should be applied to the feature map to produce a transformation dependent on the input data. That is, it takes the input feature map $U \in \mathbb{R}^{H \times W \times C}$ with width W, height H, and C channels and outputs the parameters θ of the transformation T_θ to be applied to U. This network can take any form (e.g. FCN, CNN), contingent on the fact that a final regression layer is included to compute θ.

The parameters are then used to create a sampling grid via the *grid generator*, which defines a set of points in the image space where the input map should be sampled to generate the corresponding transformed output. Fig. 2 provides a visual representation of such transformations. In the final step, the *sampler* takes both the sampling grid and feature map as inputs and generates the output map, which is sampled at the grid points of the input. For an in-depth discussion of the architecture and computations involved in each of the three components, we refer the readers to the original paper [2].

The spatial transformer, which comprises of the localization network, grid generator, and sampler, can be inserted into any standard neural network architecture, without drastically impeding training time even if used naively [2]. Including STs enables the network to learn how to actively transform feature maps to minimize the overall cost associated with training the model [2].

2.2 Key Results of the Original Paper

The overarching result of the original paper is as follows: the use of spatial transformer networks demonstrates state-of-the-art performances on several supervised learning tasks. The results of applying baseline networks along with those that include spatial transformers on the benchmark datasets (i.e. distorted MNIST, SVHN) are copied from the original paper, and summarized in Tables 1 and 2, respectively.

The authors demonstrate that STs improve the classification accuracy of the distorted MNIST handwriting dataset by

Model		MNIST Distortion R	RTS	P	E
FCN		2.1	5.2	3.1	3.2
CNN		1.2	0.8	1.5	1.4
ST-FCN	Aff	1.2	0.8	1.5	2.7
	Proj	1.3	0.9	1.4	2.6
	TPS	1.1	0.8	1.4	2.4
ST-CNN	Aff	0.7	0.5	0.8	1.2
	Proj	0.8	0.6	0.8	1.3
	TPS	0.7	0.5	0.8	1.1

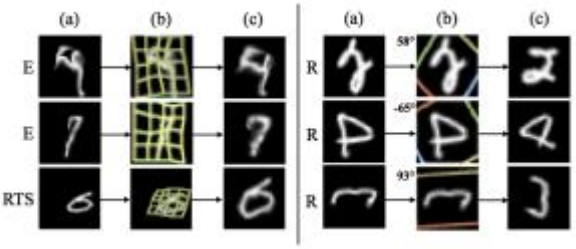

Table 1: *Left*: Percentage errors for different models on different distorted MNIST datasets. The different distorted MNIST datasets we test are TC: translated and cluttered, R: rotated, RTS: rotated, translated, and scaled, P: projective distortion, E: elastic distortion. All the models used for each experiment have the same number of parameters, and same base structure for all experiments. *Right*: Some example test images where a spatial transformer network correctly classifies the digit but a CNN fails. (a) The inputs to the networks. (b) The transformations predicted by the spatial transformers, visualized by the grid $T_\theta(G)$. (c) The outputs of the spatial transformers. E and RTS examples use thin plate spline spatial transformers (ST-CNN TPS), while R examples use affine spatial transformers (ST-CNN Aff) with the angles of the affine transformations given.

Model		Size 64px	128px
Maxout CNN [13]		4.0	-
CNN (ours)		4.0	5.6
DRAM* [1]		3.9	4.5
ST-CNN	Single	3.7	**3.9**
	Multi	**3.6**	**3.9**

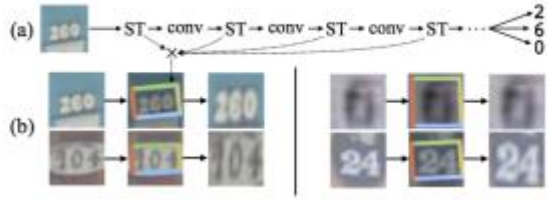

Table 2: *Left*: The sequence error for SVHN multi-digit recognition on crops of 64 × 64 pixels (64px), and inflated crops of 128 × 128 (128px) which include more background. *The best reported result from [1] (of the original paper) uses model averaging and Monte Carlo averaging, whereas the results from other models are from a single forward pass of a single model. *Right*: (a) The schematic of the ST-CNN Multi model. The transformations applied by each spatial transformer (ST) is applied to the convolutional feature map produced by the previous layer. (b) The result of multiplying out the affine transformations predicted by the four spatial transformers in ST-CNN Multi, visualized on the input image.

actively transforming the input feature maps, hence learning invariance to such transformations. Moreover, in using multiple spatial transformers embedded in CNNs, the authors obtain state-of-the-art results on the SVHN dataset for number recognition. Once again, for a more detailed analysis of the experimental results, we refer the readers to the original paper [2]. Thus, even though CNNs provide a robust baseline for multiple supervised learning tasks in computer vision, gains in accuracy may be further attained with the inclusion of spatial transformers in the network architecture.

In this section, we describe a few pertinent results mentioned in the original paper, which we do not empirically replicate as they are beyond the scope of our project, but would also incur a fee exceeding our budgets allocated for running GPU instances on Amazon Web Services (AWS). First, the authors achieved state-of-the-art performance for fine-grained classification on the CUB-200-2011 bird dataset via multiple parallel spatial transformers. These networks learned to discover object parts and attend to them. Second, the authors demonstrate the ability for STs to model multiple objects (A.1), co-localization (A.2), and extension to higher-dimensional transformers (A.3). These are all pragmatic yet intriguing applications we hope to pursue in the future.

3. Methodology

In this section, we describe the methodology and network architectures that we implemented to replicate the results presented in [2]. We also mention the discrepancies between our approach and that of the original paper, which may be attributed to our desire to reduce training time due to time and budget constraints. Finer details related to the differences in implementation and training procedure will be mentioned in Section 4. This section is organized as follows: we first provide our objectives and the challenges associated with them. Then, we formulate our problem more concretely, introducing the core engineering and mathematical concepts that underlie our design.

3.1. Objectives and Technical Challenges

Although the original paper covers much more material, we only aim to replicate their results pertaining to the distorted MNIST and SVHN datasets. In both cases, we experiment with FCNs and CNNs that include and exclude spatial transformers to determine their performances on the said benchmarks. The primary challenge is to train networks that are invariant to common spatial transformations, which is exactly what STs are made to address. We will describe our methodology in finer details in Section 4, after we describe the foundations of STs.

3.2. Problem Formulation and Design

As previously mentioned, spatial transformers are comprised of three sequential components: the localization network, grid generator, and sampler. Before mentioning the intricacies involved with spatial transformers, we would like to mention that most of this sections is based on the works presented in Spatial Transformer Networks [2]. We attribute all credit and merit to the original authors, as we are simply endeavoring to replicate their results.

As aforementioned, the localization network works as follows: it "takes the input feature map $U \in \mathbb{R}^{H \times W \times C}$ (width W, height H, and C channels) and outputs θ, the parameters of the transformation T_θ to be applied to the feature map, $\theta = f_{loc}(U)$" [2]. The shape of θ is dependent on the transformation type (e.g. 6-dimensional for affine transformations).

The grid generator performs a warping of the input feature map, in which output pixels are computed via a sampling kernel centered at a given location in the input feature map [2]. In general, "the output pixels are defined to line on a regular grid $G = \{G_i\}$ of pixels $G_i = (x_i^t, y_i^t)$, forming an output feature map $V \in \mathbb{R}^{H' \times W' \times C}$" [2].

Without loss of generality, we assume that T_θ is a 2D affine transformation A_θ. Then, the pixel-wise transformation is as follows:

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

Here, (x_i^t, y_i^t) are the target coordinates of the regular grid in the output feature map, and (x_i^s, y_i^s) are the source coordinates in the input feature map, which define the sample points [2]. We also normalize the height and width such that all values of the coordinates are between the interval [-1, 1].

Note that this transformation requires only 6 parameters, and allows for cropping, translation, rotation, scale, and skew [2]. To extend to projective transformations, A_θ simply takes on 8 parameters. In fact, it can have any dimension, contingent on the fact that it is differentiable with respect to its parameters. This ensures back-propagation is possible during training.

To spatially transform the input feature map, the sampler must take the following as inputs: the sampling points $T_\theta(G)$ and the input feature map U. Recall that each of the source coordinate in $T_\theta(G)$ defines the location in the input at which the sampling kernel is applied to produce the corresponding value in the output V. This can be formulated as follows:

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m\,;\, \Phi_x) k\big(y_i^s - n; \Phi_y\big)$$

$$\forall\, i \in [1, \dots, H'W'], c \in [1, \dots, C]$$

Here, Φ_i $(i = x, y)$ are the parameters that define a generic sampling kernel, which in turn defines the image (bilinear) interpolation. U and V define the value at location (n, m) of the input and outputs respectively. All other variables are as previously defined. In our case, the bilinear sampling kernel is defined as such:

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

To allow back-propagation of the loss through the spatial transformer, we use the proposed method mentioned in [2], in which the gradients are defined with respect to U and G. More specifically, in our case, the partial derivatives for bilinear sampling are as follows:

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

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

The partial derivative of V_i^c with respect to the y_i^s is can be defined in a similar fashion. As we can see, discontinuities exist in the sampling functions. To remediate this issue, we simply ignore the sum over all locations and just consider the support region of the kernel for each output value [2]. All of this is self-contained and can thus be included in any layer of a model. For a simple visualization of the above process, we refer the readers to Figures 1 and 2 as previously explained. They provide an intuitive illustration of an abstracted, top-level understanding of spatial transformers.

With the introduction of spatial transformers into any standard FCN or CNN architecture, we can successfully solve the spatial invariance problem many baseline models suffer from. It should be clear from above why the intrinsic design of spatial transformers can help models achieve invariance to common geometric transformations.

It should be mentioned for completeness that we purposefully did not include any system drawings, block diagrams, and/or circuit schematics as they would not only convolute the message being relayed but are also somewhat superfluous to the objectives on hand. We also believe the above descriptions of STs are succinct enough that extra flow charts and pseudo-code would not provide further insights to our software design or even the mechanism of STs. However, for a more fine-grained scrutiny of the software architecture, we kindly refer the readers to our source code repository [1]. In the next section, we describe in detail the specific network architectures implemented and the training process.

4. Implementation

In this section, we first discuss our experimentation with the distorted MNIST dataset. More specifically, we describe how we created this dataset for its canonical counterpart, the specific network architectures we implemented, and the corresponding training process. We then do the same for the SVHN dataset.

4.1. Deep Learning Network

For each benchmark dataset, we present the top-level architectural block diagrams, associated algorithms and network architecture, and data used. We also mention the more pertinent discrepancies between our implementation and that of the original paper.

4.1.1. Generating the distorted MNIST dataset

The MNIST dataset contains 50K, 10K, and 10K images for training, validation, and testing. To generate the distorted images, we closely follow the methods of the original paper. For the rotated (R) dataset, we rotate each MNIST training digit with a uniformly-sampled random rotation between the interval [-90°, 90°]. For the rotated, translated, and scaled (RTS) dataset, we rotate each MNIST digit by ±45°, scale by a value from the interval [0.7, 1.2], and translate the digit to a random location in a 42 × 42 image. All operations were performed randomly with uniform distributions. The projected (P) dataset was produced by randomly scaling a digit between 0.75 and 1.0, and stretching the corners of the digit by an amount sampled from a normal distribution zero mean and five-pixel standard deviation. We chose to ignore the elastically (E) warped dataset, which perturbs 16 control points of a thin plate spline arranged in a regular grid on the image by an amount sampled from a normal distribution, since training time was ostensibly interminable.

For each modified dataset, the ground truths used were that of the original MNIST dataset. That is, we use the labels of the undistorted images for their distorted counterparts. Thus, during training, the network is coerced to be invariant to geometric transformations to perform well.

4.1.2. Network architecture for distorted MNIST

Figure 3 illustrates the two baseline CNN and FCN we used for classification of the distorted MNIST digits. Figure 4 shows the spatial transformers that were added to the baseline networks to help realize spatial invariance to input feature maps. In both cases, the ST modules were placed at the beginning of the respective networks. Moreover, in all cases, rectified linear nonlinearities and softmax classifiers were used.

The number of units in both the FCN- and CNN-based classification models varies as to "ensure that all networks for a particular experiment contain the same number of

learnable parameters" [2]. Hence, networks that include STs generally have less parameters in the latter layers since we need to account for those used by the localization network in the ST modules.

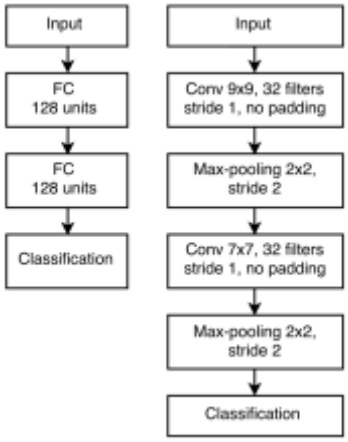

Figure 3: *Left:* baseline fully-connected network. Two hidden fully connected layers with 128 hidden units each. *Right*: baseline convolutional neural network. Conv represents convolutional layers. Both networks used rectified linear nonlinearities and softmax classifiers.

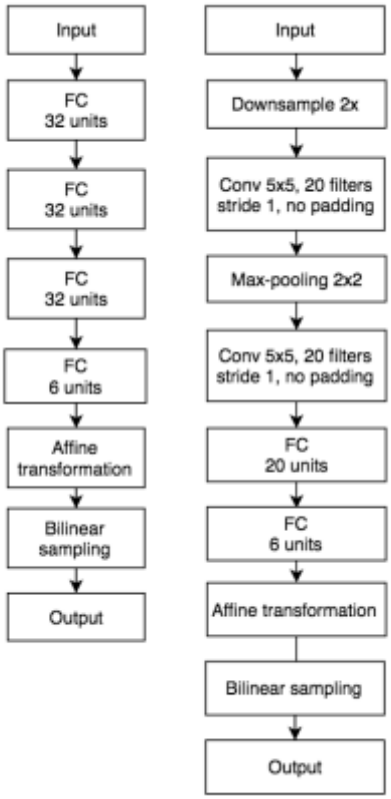

Figure 4: *Left*: spatial transformer for a fully-connected (FC) network. The three FC layers constitute the localization network. *Right*: spatial transformer for the convolutional neural network (CNN). All ST-enabled networks place the above two modules at the beginning of their respective networks.

It should be noted that we only implemented an affine transformer, whereas the original paper also used projective and thin plate spline spatial transformers. We ignored the latter two due to time constraints and the associated cost we would incur for training such transformers.

4.1.3 Network training for distorted MNIST

In all cases, the classification models were trained with vanilla gradient descent for 40,000 epochs. This differs from the original paper, which uses 150,000 iterations. All other hyperparameters remained the same. That is, we also used a batch size of 256 and learning rate of 0.01. No weight decay or dropout was included. We initialize all network weights randomly, except for the final regression layer of the localization networks. This layer is initialized to regress the identity transform [2]. In addition, the authors perform each training three times and report the average. We only perform training once. Thus, our results will obviously be subpar in comparison.

4.1.4 Street View House Numbers dataset

The Street View House Numbers (SVHN) dataset contains 200K real world images of house numbers, in which the task is to recognize the sequence of numbers in each image. Since the SVHN dataset already contains intrinsic transformations of digits within the images (e.g. digits within natural scenes), we do not perform any preprocessing and leave it as is.

4.1.5 Network architecture for SVHN

Our network architecture for the SVHN dataset differs significantly from that implemented by the original authors. In the original paper, the authors trained an 11-layer baseline CNN followed by 5 independent softmax classifiers, each one predicting the digit at a certain position in the sequence [2]. To incorporate the ST module, the author constructed two types of network architecture. First, they placed a single ST, in which the localization network is a four layer CNN, at the beginning of the network (ST-CNN Single). This is akin to the architecture used for the distorted MNIST dataset. Second, the authors insert four STs before the first four convolutional layers of the baseline CNN (ST-CNN Multi). Here the localization networks are all two-layer FC networks with 32 units per layer. For a detailed description of the authors' implementation, we refer the readers to Appendix A.5 of the original paper [2].

On the contrary, due to time and budget constraints for our project, we simply re-implemented our CNN-based network architecture for the SVHN dataset (i.e. right diagram in Figure 3 and 4). With fewer layers and parameters, we surmise our model performances would not reflect the values obtained in the original paper. However, like the original paper, we trained all networks with gradient descent, used

randomly initialized weights except for the regression layer of STs, and implemented affine transformations and bilinear sampling kernels for all ST modules.

Although using a more parsimonious model seems like a contrived endeavor to tackle the SVHN dataset, we see it as an intriguing experiment that delineates how the same model applied to datasets of varying difficulty could influence the outcome of certain supervised learning tasks.

4.1.6 Network training for SVHN

Our training procedure and the chosen hyperparameters are also somewhat different from those used in the original paper. The authors used the method from [4], which was to select hyperparameters from a validation set of 5k images from the training set. In addition, the authors trained for 400k iterations with SGD using a batch size of 128 and a learning rate of 0.01, which was decreased by an order of magnitude every 80k iterations. They also used a weight decay of 0.0005 and dropout of 0.5. On the contrary, we simply reused the hyperparameters implemented for the distorted MNIST dataset (Section 4.1.3). Once again, the authors report the average of two runs whereas we only perform one run. It should therefore be obvious that our cruder approach to the SVHN dataset would ultimately yield poor results relative to that of the original paper. However, we think this might provide some noteworthy insights as to how a network that performs well on an easier dataset fairs on a more challenging and somewhat less contrived dataset.

4.2. Software Design

We provide a concise flow chart of the computational flow of a spatial transformer, which should be a sufficient figure for our purposes. Any lower-level flow chart pertaining to our implementations would not only provide diminishing marginal gains in the information conveyed, but would also convolute our analysis.

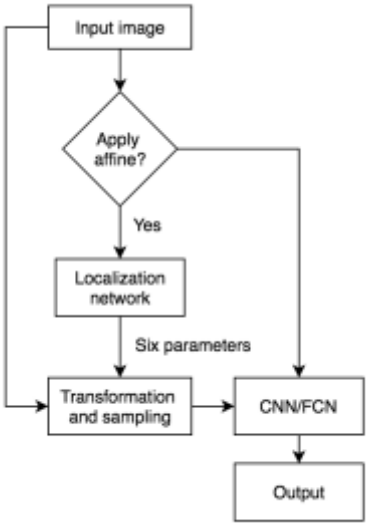

Figure 5: Flow chart of the top-level architecture of a spatial transformer module.

For a step-by-step description of our algorithm, we refer the reader to Section 4, which does an apt job conveying the corresponding information. To facilitate the readers to conceptualize our actual implementation, we provide here the pseudocode of the spatial transformer for CNNs and FCNs. This is for the distorted MNIST dataset, but is similar enough to our implementation for the SVHN dataset.

```
/* FCN */
spatial_transformer = ST-FCN(
    input_dim=(batch_size, 1, 28, 28)
)

classifier = MultiLayerPerceptron(
    input = spatial_transformer.output,
    input_dimension = 1 x 28 x 28,
    number_of_hidden_units = 128,
    number_of_output_units = 10,
    number_of_hidden_layers = 2,
    activation_function = ReLU
)

/* CNN */
spatial_transformer = ST-CNN(
    input_dimension = (batch_size, 1, 28, 28),
    number_of_filters = 20,
    scale = 2
)

// first convolutional pooling layer
cp1 = ConvolutionalPooling(
    input = spatial_transformer.output,
    image_shape = (batch_size, 1, 28, 28),
    filter_shape = (32, 1, 9, 9),
    pooling=(2,2)
)

// second convolutional pooling layer
cp2 = ConvolutionalPooling(
    input = cp1.output,
    image_shape = (batch_size, 1, 10, 10),
    filter_shape = (32, 32, 7, 7),
    pooling = (2,2)
)

rg = Regression(
    input = cp2.output,
    input_dimension = 32 x 2 x 2,
    output_dimension = 10
)
```

We do not provide any further details regarding subroutines and auxiliary functions, since describing the network architecture and training procedure (Section 4) has implicitly delineated our algorithm. However, if finer details are needed, the reader may find our source code repository helpful [1].

5. Results

5.1. Project Results

To test the ability for neural networks to be invariant towards common spatial transformations, we trained our models on the MNIST dataset, in which digits were subjected to rotation, translation, scaling, and projection. Figure 6 shows some of the digits after such transformations.

When we trained our FCN and CNN-based classification models on the distorted MNIST dataset, we observe that CNNs perform much better than FCNs. Models that involved spatial transformer modules were also much more robust towards geometric variance. This is intuitive since CNNs were explicitly designed to handle images by exploiting spatially-local correlations in the image space. Furthermore, STs, which are designed to internalize spatial invariance, would obviously perform better on datasets that purposefully contain geometric variance within the input images. Table 3 lists the corresponding validation error, test error, and training duration for the four network architectures on three types of digit distortion. In line with the trends as previously mentioned, the model that performed best was the ST-CNN. Contrarily, the model that performed the worst was FCN. This holds for all three types of distortion (i.e. R, RTS, and P).

We tested our models using different random seeds, and similar results were procured in all cases. This substantiates the fact that the inclusion of spatial transformers does indeed help models achieve spatial invariance, and hence better overall performances.

5.2. Comparison of Results

Due to the blatant discrepancy between our hardware infrastructure, allotted time, and budget and that of the original paper (i.e. Google DeepMind), our results did not fare as well. We obtained higher error rates for any given model and distortion. Table 4 describes the differences in error rates between our implementation and that of the original paper. Nonetheless, even though we obtained subpar results, the trend amongst models between the two papers are consistent, with CNNs outperforming FCNs, and ST-enabled networks exceeding baseline networks in performance.

Figure 6: Examples of MNIST digits subjected to *left*: rotation, *middle*: rotation, translation, and scaling, *right*: projection. We used such images for training our models.

	R	RTS	P
FCN	7.492	19.65	10.79
	7.850	20.58	10.94
	3.76	4.00	3.80
ST-FCN	5.16	6.27	8.28
	5.38	6.90	8.27
	26.43	25.25	25.81
CNN	4.02	8.08	6.59
	4.41	8.11	6.99
	40.55	40.85	39.12
ST-CNN	3.18	5.67	6.30
	3.37	6.33	6.44
	78.38	85.12	78.59

Table 3: Performance of baseline and spatial transformer enabled networks on the distorted MNIST dataset. The rows denote the network type: FCN, fully connected network; CNN, convolutional neural network; ST, spatial transformer. The columns denote the type of distortion applied to the MNIST dataset: R, rotation; RTS, rotation, translation, scaling; P, projection. For each cell, the first, second, and third entries describe the validation error (%), test error (%), and training time (minutes).

	R	RTS	P
FCN	5.75	15.38	7.84
ST-FCN	4.18	6.10	6.77
CNN	3.21	7.31	5.49
ST-CNN	2.67	5.83	5.64

Table 4: Differences in performance (on the distorted MNIST dataset) between that of the original paper and our implementation. The rows denote the network type: FCN, fully connected network; CNN, convolutional neural network; ST, spatial transformer. The columns denote the type of distortion applied to the MNIST dataset: R, rotation; RTS, rotation, translation, scaling; P, projection. Each entry denotes the percentage difference in error rates (i.e. $\text{error}_{us} - \text{error}_{original}$), in which larger positive values imply that our model performed poorer relative to the original paper.

Since the original paper did not explicitly mention training time, we are unable to fully compare this aspect. However, since the original paper implemented more than 150K iterations, in contrast to our 40K iterations, it is obvious that their training time significantly exceeded ours. We surmise this is an important factor contributing to their superior performance on the distorted MNIST dataset.

It should be noted that we purposefully excluded the results we obtained from the SVHN dataset, since they were extremely poor (more than 35% in validation and testing error) and would not yield any meaningful discussion. The only fruitful discussion we can make regarding this result is that more complicated, deeper networks are required for more challenging datasets.

5.3. Discussion of Insights Gained

Based on our observations from experimenting with different variants of the FCN and CNN-based classification models, we can see that incorporating spatial transformer modules not only greatly improves a network's ability to be spatially invariant to the input feature map, but also yields state-of-the-art performances on certain tasks related to computer vision, particularly for image classification.

Using a CNN baseline model while including ST modules in the network greatly improves the capacity of the model to deal with certain geometric transformations (e.g. rotation, translation, scaling, and warping), which are common motifs underlying natural images. With the number of parameters being held constant, inserting a ST module in a network saw more than 2% decreases in testing error across the board, regardless of the distortion under consideration. Hence, the utilization of spatial transformers could greatly facilitate the capability for any given network architecture to remediate performance issues caused by geometric variance.

6. Conclusion

In this work, we replicate the spatial transformer, which is a self-contained module that facilitates the ability of neural networks to learn geometric invariance towards their input data. We successfully demonstrate that incorporating spatial transformers helps improve the performance of any FCN or CNN-based classification model, since such models achieved better accuracy than their baseline counterparts on the distorted MNIST dataset. Nonetheless, we were unable to achieve similar success on the SVHN dataset. We attribute this shortcoming to our crude network design. We believe that implementing a deeper model with more parameters, such as the one presented in the original paper, we would achieve much better results. This is an area for further improvement. We also believe that spatial transformers could be used in recurrent models, providing a catalyst to solve tasks requiring the ability to learn object reference frames. All these facets not only accentuate the ability of spatial transformers to achieve state-of-the-art performance, but also their potential to solve other relevant computer vision tasks.

6. Acknowledgement

We thank the teaching assistants of ECBM 4040 for their guidance and advise throughout the process of completing our final project.

7. References

[1] https://bitbucket.org/e_4040_ta/e4040_project_rcnn
[2] M. Jaderberg, K. Simonyan, A. Zisserman, K. Kavukcuoglu, "Spatial Transformer Networks", arXiv:1506.02025, 2015.
[3] T.S. Cohen, M. Welling. Transformation properties of learned visual representations, ICLR, 2015.
[4] I.J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks. arXiv:1312.6082, 2013.

8. Appendix

8.1 Individual student contributions

	zs2321	hl2997
Last name	Sun	Lin
Fraction of contribution	½	½
Details	Both authors contributed equally to the report and source code.	