# Understanding when spatial transformer networks do not support invariance, and what to do about it

Lukas Finnveden*, Ylva Jansson* and Tony Lindeberg
Computational Brain Science Lab, Division of Computational Science and Technology
KTH Royal Institute of Technology, Stockholm, Sweden



***Abstract*—Spatial transformer networks (STNs) were designed to enable convolutional neural networks (CNNs) to learn invariance to image transformations. STNs were originally proposed to transform *CNN feature maps* as well as input images. This enables the use of more complex features when predicting transformation parameters. However, since STNs perform *a purely spatial transformation*, they do not, in the general case, have the ability to align the feature maps of a transformed image with those of its original. STNs are therefore unable to support invariance when transforming CNN feature maps. We present a simple proof for this and study the practical implications, showing that this inability is coupled with decreased classification accuracy. We therefore investigate alternative STN architectures that make use of complex features. We find that while deeper localization networks are difficult to train, localization networks that share parameters with the classification network remain stable as they grow deeper, which allows for higher classification accuracy on difficult datasets. Finally, we explore the interaction between localization network complexity and iterative image alignment.**

## I. Introduction

Spatial transformer networks (STNs) [1] constitute a widely used end-to-end trainable solution for CNNs to learn invariance to image transformations. This makes it part of a growing body of work concerned with developing CNNs that are invariant or robust to image transformations. The key idea behind STNs is to introduce a trainable module – the spatial transformer (ST) – that applies a data dependent spatial transformation of input images or CNN feature maps before further processing. If such a module successfully learns to align images to a canonical pose, it can enable invariant recognition. However, when transforming *CNN feature maps*, such alignment is, in general, not possible, and STNs can therefore not enable invariant recognition. The reasons for this are that: (i) STs perform a purely spatial transformation, whereas transforming an image typically also results in a shift in the channel dimension of the feature activations (Figure 1), (ii) The shapes of the receptive fields of the individual neurons are not invariant. (Figure 2). These problems have, to our knowledge, not been discussed in the STN literature. In the original paper and a number of subsequent works, STNs are presented as an option for achieving invariance also when applied to intermediate feature maps [1]–[5].

*The first and second authors contributed equally to this work.

Shortened version in International Conference on Pattern Recognition (ICPR 2020), pages 3427–3434. Jan 2021. The support from the Swedish Research Council (contract 2018-03586) is gratefully acknowledged.

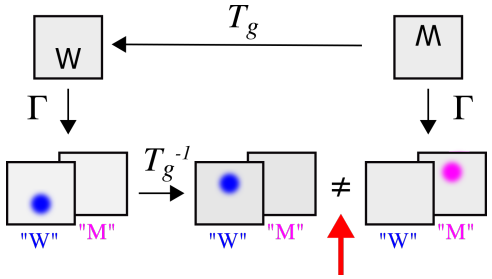


Fig. 1. *A spatial transformation of a* CNN feature map *cannot, in general, align the feature maps of a transformed image with those of its original*. Here, the network Γ has two feature channels "W" and "M", and $T_g$ corresponds to a 180° rotation. Since different *feature channels* respond to the rotated image as compared to the original image, it is not possible to align the respective feature maps by applying the inverse spatial rotation to the feature maps. This implies that *spatially transforming feature maps* cannot enable invariant recognition by the means of aligning a set of feature maps to a common pose.

Our first contribution is to present a simple proof that STNs do not enable invariant recognition when transforming CNN feature maps. We do not claim mathematical novelty of this fact, which is in some sense intuitive and can be inferred from more general results (see e.g. [6]), but we present a simple alternative proof directly applicable to STNs and accessible with knowledge about basic analysis and some group theory. We believe this is important since the idea that transforming feature maps can achieve invariant recognition is often either proposed explicitly or the question about the ability for invariance is ignored for a range of different methods transforming CNN feature maps or filters although often misunderstood or not taken into account. Our second contribution is to explore the practical implications of this result. Is there a point in transforming intermediate feature maps if this cannot enable invariant recognition? To investigate this, we compare different STN architectures on the MNIST [7], SVHN [8] and Plankton [9] datasets. We show that STNs that transform the feature maps are, indeed, worse at compensating for rotations and scaling transformations, while they – in accordance with theory – handle pure translations well. We also show that the inability of STNs to fully align CNN feature maps is coupled with decreased classification performance. Our third contribution is to explore alternative STN architectures that make use of more complex features when predicting image transformations. We show that: (i) Using more complex features can significantly improve performance, but most of

this advantage is lost if transforming CNN feature maps, (ii) sharing parameters between the localization network and the classification network makes training of deeper localization networks more stable and (iii) iterative image alignment can be complimentary to, but is no replacement for, using more complex features.

In summary, this work clarifies the role and functioning of STNs that transform input images vs. CNN feature maps and illustrates important tradeoffs between different STN architectures that make use of deeper features.

### A. Related work

Successful *image alignment* can considerably simplify a range of computer vision tasks by reducing variability related to differences in object pose. In classical computer vision, Lukas and Kanade [10] developed a methodology for image alignment, which estimates translations iteratively. This approach was later generalized to more general parameterized deformation models [11]. Affine shape adaptation of affine Gaussian kernels to local image structures – or equivalently, normalizing image structures to canonical affine invariant reference frames [12] – has been an integrated part in frameworks for invariant image-based matching and recognition [13]–[15]. Such classical approaches always align *the input images*.

Lately, the idea to combine structure and learning has given rise to the subfield of *invariant neural networks*, which add structural constraints to deep neural networks to enable e.g. scale or rotation invariant recognition [16]–[19]. *Spatial transformer networks* [1] are based on a similar idea of combining the knowledge about the structure of image transformations with learning. An STN, however, does not hard code invariance to any specific transformation group but learns input dependent image alignment from data. The original work [1] simultaneously claims *the ability to learn invariance from data* and that ST modules can be inserted at *"any depth"*. This seems to have left some confusion about whether STNs can enable invariance when transforming CNN feature maps. A number of subsequent works advocate to perform image alignment by transforming CNN feature maps [2]–[5] including e.g. pose alignment of pedestrians [5] and to use a spatial transformer to mimic the kind of patch normalization done in SIFT [2]. Additional works transform CNN feature maps without giving any specific motivation [20], [21]. As we will show, transforming CNN feature maps is *not equivalent* to extracting features from a transformed input image. Other approaches dealing with pose variability by transforming neural network *feature maps or filters* are e.g. spatial pyramid pooling [22] and dilated [23] or deformable convolutions [24]. Our results imply that these approaches have limited ability to enable e.g. scale invariance.

Weight sharing between the classification and the localization network has previously been considered in [25], primarily as a way of regularizing CNNs. Here, we are instead interested in it as a way to make use of deeper features when predicting image transformations. [26] combines STNs with iterative image alignment. In this paper, we investigate whether such iterative alignment is complimentary to using deeper features.

Previous theoretical work [6] characterizes all equivariant (covariant) maps between homogeneous spaces using the theory of fibers and fields. We, here, aim for a different perspective on the same theory. We present a simple proof for the special case of *purely spatial transformations* of CNN feature maps together with an *experimental evaluation* of different STN architectures. A practical study [27] indicated that approximate alignment of CNN feature maps can be possible if allowing for a full transformation, as opposed to the purely spatial transformations that we analyze in this paper.

## II. Theoretical analysis of invariance properties

Spatial transformer networks [1] were introduced as an option for CNNs to learn invariance to image transformations by transforming *input images or convolutional feature maps* before further processing. A spatial transformer (ST) module is composed of a localization network that predicts transformation parameters and a transformer that transforms an image or a feature map using these parameters. An STN is a CNN with one or several ST modules inserted at arbitrary depths.

### A. How STNs can enable invariance

We will here work with a continuous model of the image space. We model an image as a measurable function $f : \mathbb{R}^n \to \mathbb{R}$ and denote this space of images as $V$. Let $\{\mathcal{T}_h\}_{h\in H}$ be a family of image transformations corresponding to a group $H$. $\mathcal{T}_h$ transforms an image by acting on the underlying space

$$(\mathcal{T}_h f)(x) = f(T_h^{-1}x) \tag{1}$$

where $T_h : \mathbb{R}^n \to \mathbb{R}^n$ is a linear map. We here consider affine image transformations, but the general argument is also valid for non-linear invertible transformations such as e.g. diffeomorphisms. Let $\Gamma : V \to V^k$ be a (possibly non-linear) translation covariant feature extractor with $k$ feature channels. $\Gamma$ could e.g. correspond to a sequence of convolutions and pointwise non-linearities. $\Gamma$ is *invariant* to $\mathcal{T}_h$ if the feature response for a transformed image is equal to that of its original

$$(\Gamma\mathcal{T}_h f)_c(x) = (\Gamma f)_c(x), \tag{2}$$

where $c \in [1, 2, \cdots k]$ corresponds to the feature channel. An ST module can *support invariance* by learning to transform input images to a canonical pose, before feature extraction, by applying the inverse transformation

$$(\Gamma\,\mathrm{ST}(\mathcal{T}_h f))_c(x) = (\Gamma\mathcal{T}_h^{-1}\mathcal{T}_h f)_c(x) = (\Gamma f)_c(x). \tag{3}$$

We will in the following assume such *a perfect ST* that always manages to predict the correct pose of an object.[1] We now show that even a perfect ST cannot support invariance if instead applied to *CNN feature maps*.

[1]There is no hard-coding of invariance in an STN and no guarantee that the predicted object pose is correct or itself invariant. We here assume the ideal case where the localization network does learn to predict the transformations that would align a relevant subset of all images (e.g. all images of the same class) to a common pose.

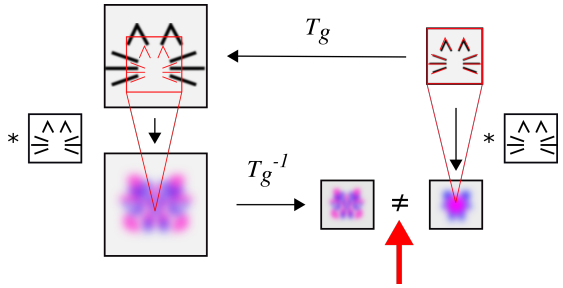


Fig. 2. *For any transformation that includes a scaling component, the field of view of a feature extractor with respect to an object will differ between an original and a rescaled image.* Consider a simple linear model that performs template matching with a single filter. When applied to the original image, the filter matches the size of the object that it has been trained to recognize and thus responds strongly. When applied to a rescaled image, the filter never covers the full object of interest. Thus, the response cannot be guaranteed to take even *the same set of values* for a rescaled image and its original.

### B. The problems of transforming CNN feature maps

An advantage of inserting an ST deeper into the network is that the ST can make use of more complex features shared with the classification network. When using STNs that transform feature maps, as opposed to input images, the key question is whether it is possible to undo a transformation of an image *after feature extraction*. Is there a $\mathcal{T}_g$ dependent on $\mathcal{T}_h$ such that (applying the same transformation in each feature channel)

$$(\mathcal{T}_g \Gamma \mathcal{T}_h f)_c(x) = (\Gamma \mathcal{T}_h f)_c(T_g^{-1} x) \stackrel{?}{=} (\Gamma f)_c(x) \qquad (4)$$

holds for all $f, \Gamma$ and $h$? If this is possible, we refer to it as *feature map alignment*. An ST that transforms CNN feature maps could then support invariance by the same mechanism as for input images. We here present the key intuitions and the outline of a proof that this is, in the general case, *not possible*. We refer to [28] for a mathematically rigorous proof. Note that for any translation covariant feature extractor, such as a continuous or discrete CNN, feature map alignment for *translated images* is, however, possible by means of a simple translation of the feature maps.

*1) Using $\mathcal{T}_g = \mathcal{T}_h^{-1}$ is a necessary condition to align CNN feature maps with an ST:* The natural way to align the feature maps of a transformed image to those of its original would be to apply *the inverse spatial transformation* to the feature maps of the transformed image i.e.

$$\mathcal{T}_h^{-1}(\Gamma \mathcal{T}_h f)_c(x) = (\Gamma \mathcal{T}_h f)_c(T_h x). \qquad (5)$$

For example, to align the feature maps of an original and a rescaled image, we would, after feature extraction, apply the inverse scaling to the feature maps. Using $\mathcal{T}_h^{-1}$ is, in fact, *a necessary condition* for (4) to hold [28]. To see this, note that the value for each spatial position $x$ must be computed from *the same region* in the original image for the right hand and left hand side. Clearly, features extracted from different image regions cannot be guaranteed to be equal. Assume that $(\Gamma f)_c(x)$ is computed from a region $\Omega$ centered at $x$ in the original image. Using the definition of $\mathcal{T}_h$ and the fact that $\Gamma$ is translation covariant, we get that $(\mathcal{T}_g \Gamma \mathcal{T}_h f)(x)$ is computed from a region $\Omega'$ centered at $T_g^{-1} T_h^{-1} x$. Now,

$$\Omega = \Omega' \implies T_g^{-1} T_h^{-1} x = x \implies \mathcal{T}_g = \mathcal{T}_h^{-1} \qquad (6)$$

and thus the only candidate transformation to align CNN feature maps with an ST is $\mathcal{T}_g = \mathcal{T}_h^{-1}$. We, next, show that using $\mathcal{T}_g = \mathcal{T}_h^{-1}$ is, however, not *a sufficient condition* to enable feature map alignment for two key reasons.

*2) Transforming an image typically implies a shift in the channel dimension of the feature map:* When transforming an input image, this typically causes not only a spatial shift in its feature representation but also a shift in *the channel dimension*. This problem is illustrated in Figure 1. Since an ST performs a *purely spatial transformation*, it cannot correct for this. A similar reasoning is applicable to a wide range of image transformations. An exception would be if the features extracted at a specific layer are themselves invariant to $H$. An example of this would be a network built from rotation invariant filters $\lambda$, where $\lambda(x) = \lambda(T_h x)$ for all $\lambda$. For such a network, or a network with more complex (learned or hardcoded) rotation invariant features at a certain layer, feature map alignment of rotated images would be possible.

It should be noted, however, that to have invariant features in intermediate layers is in many cases not desirable (especially not early in the network), since they discard too much information about the object pose. For example, rotation invariant edge detectors would lose information about the edge orientations which tend to be important for subsequent tasks.

*3) Receptive field shapes of neural networks are not invariant:* A second problem which in most cases prevents feature map alignment also by means of learning invariant features is that the receptive fields of neural networks are typically *not invariant* to the relevant transformation group. Consider e.g. a filter of finite support together with a scaling transformation. In that case, $\mathcal{T}_h^{-1}(\Gamma \mathcal{T}_h f)_c(x)$ will not only differ from $(\Gamma f)_c(x)$ because it might be computed from differently oriented image patches, but also because the scaling implies it will be computed from *not fully overlapping* image patches. This problem is illustrated in Figure 2. For a scaling transformation, a convolutional filter, adapted to detecting a certain feature at one scale, will for a larger scale never fully cover the relevant object. Since a non-trivial CNN feature extractor can not be guaranteed to take the same output for different inputs, this implies that the set of values in the two feature maps can not be guaranteed to be equal. Naturally if two feature maps do not contain the same values they can not be aligned.

It is not hard to show that invariant receptive fields of finite support only exist for transformations that correspond to reflections or rotations in some basis [28]. Intuitively this can be understood by considering how a scaling or shear transformation will always change the area covered by any finite sized template. Thus there are no non-trivial affine-,

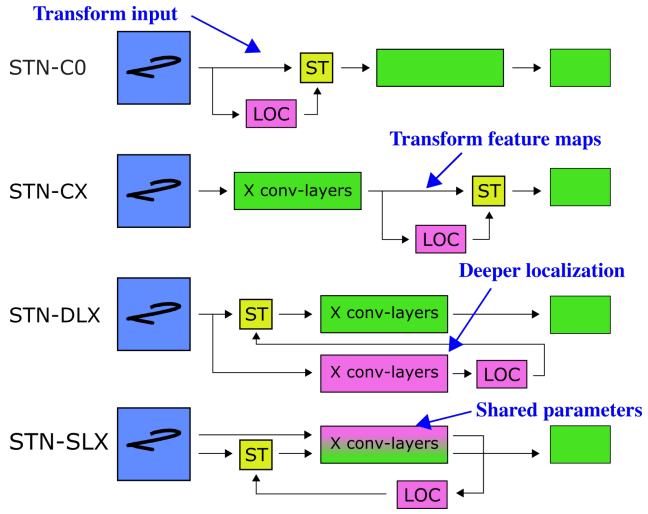


Fig. 3. *Depiction of four different ways to build STNs.* LOC denotes the localization network, which predicts the parameters of a transformation. ST denotes the spatial transformer, which takes these parameters and transforms an image or feature map according to them. In **STN-C0**, the ST transforms the input image. In **STN-CX**, the ST transforms a feature map, which prevents proper invariance. **STN-DLX** transforms the input image, but makes use of deeper features by including copies of the first X convolutional layers in the localization network. This is not fundamentally different from (1) but acts as a useful comparison point. **STN-SLX** is similar to STN-DLX, but shares parameters between the classification and localization networks.

scale- or shear-invariant filters with compact support[2].

*4) Conclusion:* Our arguments show that a purely spatial transformation cannot align the feature maps of a transformed image with those of its original for general affine transformations. This implies that while an STN transforming feature maps will support invariance to translations, it will *not enable invariant recognition for more general affine transformations*. The exception is if the features in the specific network layer are themselves invariant to the relevant transformation group.

## III. STN ARCHITECTURES

We test four different ways to structure STNs, all depicted in Figure 3. By comparing these four architectures, we can separate out the effects of (i) whether it is good or bad to transform feature maps, (ii) whether it is useful for localization networks to use deep features when predicting transformation parameters, and (iii) whether it is better for a localization network to make use of representations from the classification network, or to train a large localization network from scratch.

*1) ST transforming the input:* The localization network and the ST are placed in the beginning of the network and transform the input image. This approach is denoted by STN-C0.

*2) ST transforming a feature map:* The localization network takes a CNN feature map from the classification network as input, and the ST transforms the feature map. This architecture does not support invariance. A network with the ST after X convolutional layers is denoted by STN-CX.

*3) Deeper localization:* The localization network is placed in the beginning, but it is made deeper by including copies of some layers from the classification network. In particular, an STN where the localization network includes copies of the first X convolutional layers is denoted by STN-DLX. STN-DLX is not fundamentally different from STN-C0, since both architectures place the ST before any other transformations, but it acts as a useful comparison point to STN-CX: Both networks can make use of equally deep representations, but STN-DLX does not suffer any problems with achieving invariance. In addition, the deep representations of STN-DLX are *independently trained* from the classification network. This is beneficial if different filters are useful to find transformation parameters than to classify the image, but requires more parameters. If the training signal becomes less clear when propagated through the ST, it could also make the network harder to train. These differences motivate our fourth architecture.

*4) Shared localization:* As with STN-DLX, we place a deeper localization network in the beginning. However, for each of the copied layers, we *share parameters* between the classification network and the localization network. An STN where the first X layers are shared will be denoted by STN-SLX. STN-SLX solves the theoretical problem, uses no more parameters than STN-CX, and like STN-CX, the localization network makes use of layers trained directly on the classification loss.

### A. Iterative methods

An iterative version of STNs known as IC-STN was proposed in [26]. It starts from an architecture that has multiple successive STs in the beginning of the network, and develops it in two crucial ways: (i) Instead of letting subsequent STs transform an already-transformed image, all predicted transformation parameters are remembered and composed, after which the composition is used to transform the original image. Note that each localization network still uses the transformed image when predicting transformation parameters: the composition is only done to preserve image quality and to remove artefacts at the edges of the image. (ii) All STs use localization networks that share parameters with each other.

Both of these improvements can be generalized to work with STN-SLX. The simplest extension is to use several STs at the same depth, where all STs share localization network parameters with each other in addition to sharing parameters with the classification network. Moreover, the first improvement can be used with STN-SLX even when there are multiple STs *at different depths*, since all of them will transform the input image, regardless. In this case, the final layers of their localization networks remain separate, but whenever they predict transformation parameters, the parameters are composed with the previously predicted transformations, and used to transform the input image. This is illustrated in Figure 4.

[2]A CNN might under certain conditions learn features approximately invariant over a limited transformation rate, e.g. by average pooling over a set of filters with *effectively covariant* receptive fields (e.g. learning zero weights outside an effective receptive field of varying size/shape)

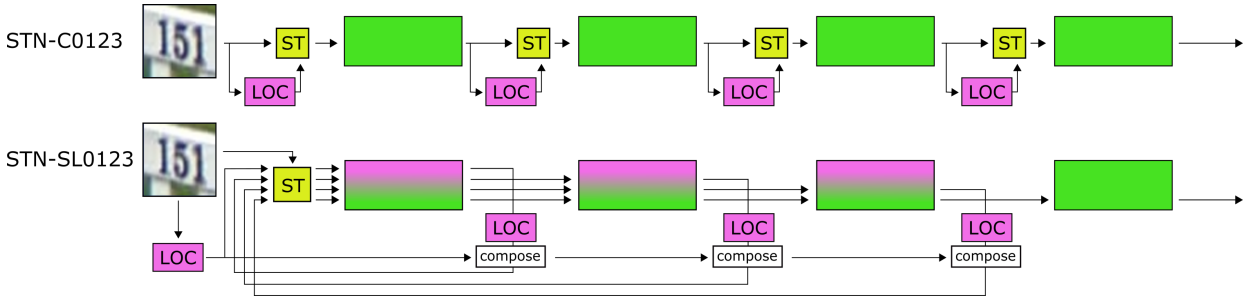


Fig. 4. *Depiction of how an STN transforming CNN feature maps at different depths can be transformed into an iterative STN with shared layers.* STN-C0123 transforms feature maps by placing STs at multiple depths [1]. STN-SL0123 instead iteratively transforms the input image and, in addition, shares parameters between the localisation networks and the classification network. The image is fed multiple times through the first layers of the network, each time producing an update to the transformation parameters. Thus, the transformation is, similarly to STN-C0123, iteratively finetuned based on more and more complex features but, at the same time, the ability to support invariant recognition is preserved.

## IV. Experiments

### A. MNIST

*1) Datasets:* MNIST is a simple dataset containing grayscale, handwritten digits of size 28x28 [7]. To see how different STN architectures compensate for different transformations, we compare them on 3 different variations of MNIST (the first two constructed as in [1]): In Rotation (R), the digits are rotated a random amount between $\pm 90°$. In Translation (T), each digit is placed at a random location on a 60x60-pixel background; to make the task more difficult, the background contains clutter generated from random fragments of other MNIST-images. In Scale (S), the digits are scaled a random amount between 0.5x and 4x, and placed in the middle of a 112x112-pixel background cluttered in a similar way to (T). Additional details about the experiments, networks and datasets are given in the Appendix.

*2) Networks:* We use network architectures similar to those in [1]. On all three datasets, the baseline classification network is a CNN that comprises two convolutional layers with max pooling. On (R), the localization network is a FCN with 3 layers. Since the images in (T) and (S) are much larger, their localization networks instead comprise two convolutional layers with max pooling and a single fully connected layer. Like in [1], the ST used with (T) produces an image with half the pixel width of its input. This is done because a perfect localization network should be able to locate the digit and scale it 2x, so all 60x60 pixels would not be needed.

MNIST is an easy dataset, so there is a risk that strong networks could learn the variations without using the ST. To make the classification accuracy dependent on the ST's performance, we intentionally use quite small networks. The networks of (R) and (T) have 50 000-70 000 learnable parameters in total, while those of (S) have around 135 000. All networks are trained using SGD, with cross-entropy loss. Networks trained on the same dataset have approximately the same number of parameters and use the same hyperparameters.

The tested architectures are STN-C0, STN-C1 (which places the ST after the first convolutional layer and max pooling), STN-DL1, and STN-SL1. A baseline CNN is also tested. Note that because of the transformations present in the training dataset, this equals a standard CNN trained with *data augmentation*. All networks are trained with 10 different random seeds, and each architecture is evaluated on 100 000 images generated by random transformations of the MNIST test set.

*3) Results:* As a first investigation of the different architectures, we study how well the STs learn to transform the digits to a canonical pose, when the networks are trained to predict the MNIST labels.

Figure 5 shows examples of how STN-C1 and STN-SL1 perform on (R), (T), and (S). As predicted by theory, STN-C1 can successfully localize a translated digit, but STN-SL1 is better at compensating for differences in rotation and scale. The difference between the networks' abilities to compensate for rotations is especially striking. Figure 6 shows that STN-SL1 compensates for rotations well, while STN-C1 barely rotates the images at all. The reason for this is that a rotation is not enough to align deeper layer feature maps.

To quantify the STs' abilities to find a canonical pose, we measure the standard deviation of the digits' final poses after they have been perturbed and the ST has transformed them. For the rotated, translated, and scaled dataset, the final pose is measured in units of degrees, pixels translated, and logarithm of the scaling factor, respectively.[3] Table I displays this value for each network.

TABLE I
AVERAGE STANDARD DEVIATION OF THE FINAL ANGLE (R), FINAL DISTANCE (T), AND FINAL SCALING (S).

| Network | R (degrees) | T (pixels) | S ($log_2$(det)) |
|---|---|---|---|
| STN-C0 | 23.2 | 1.16 | 0.319 |
| STN-C1 | 47.2 | 1.15 | 0.508 |
| STN-DL1 | 26.8 | **1.08** | **0.291** |
| STN-SL1 | **18.7** | 1.32 | 0.330 |

As can be seen, STN-SL1 is the best network on (R)

[3] As measure of the rotation of an affine transformation matrix, we compute $\arctan((a_{21} - a_{12})/(a_{11} + a_{22}))$ determined from a least-squares fit to a similarity transformation. $a_{13}$ and $a_{23}$ are used to measure the translation. As a measure of the scaling factor, we use the $\log_2$ of the determinant $a_{11}a_{22} - a_{12}a_{21}$. The standard deviation (or in the case of (T), the standard distance deviation) of the final pose is measured separately for each label, with Table I reporting the average across all labels and across 10 different random seeds.

and STN-DL1 is the best network on (T) and (S). Both these architectures use deeper localization networks, which gives them the potential to predict transformations better than STN-C0. As opposed to STN-C1, which also uses deeper representations for predicting transformation parameters, STN-SL1, STN-DL1, and STN-C0 all transform the input. This allows them to perform better on (R) and (S), as predicted by theory. However, STN-C1 performs adequately on (T).

STN-SL1 performing well on (R) can be explained by it sharing parameters between the localization network and classification network. The localization network processes images before the ST transforms them, and the classification network processes them after the transformation. Thus, we should expect parameter sharing to be helpful if the filters needed before and after the transformation are similar. This is true for (R), since edge- and corner-detectors of multiple different orientations are likely to be helpful for classifying MNIST-digits independently of digit orientation. In contrast, on both (T) and (S), the scale and resolution of the digits vary significantly before and after transformation. On (T), this is partly because the ST zooms in on the identified digit, and partly because the transformation produces an image with lower resolution, as described in Section IV-A2.[4] On (S), the scale is intentionally varied widely. Since the images contain scales and resolutions before the transformation that are not present after the transformation, the localization network requires filters that the classification network does not need. STN-DL1 allows for the networks to use different filters, and consequently, it does better than STN-SL1 on (T) and (S).

Do these differences in ST-performance affect the classification performance? Table II shows that they do. STN-SL1 remains the best network on (R), while STN-DL1 remains best on (T) and (S). The architectures that transform the input remain better than STN-C1 on (R) and (S). One difference is that STN-SL1 is better than STN-C1 on (T), which is the opposite from their relation in Table I. Since the differences in both Table I and Table II are small, this could be caused by STN-SL1 being better at compensating for rotation and scale differences inherent in the original MNIST dataset.

It is notable that all STs do improve performance. STN-C1 significantly improves on the CNN baseline even for (R), despite not compensating for rotations at all, as shown in Figure 6. So what is STN-C1 doing? One noticeably fact about its transformation matrices is that it typically scales down the image, as can be seen in Figure 5. On average, transformation matrices from STN-C1 have a determinant of 0.74 (which corresponds to scaling down the image), while the determinant of all other STNs are greater than 1. We do not have an explanation of why this should improve performance, but it is an example of how networks that spatially transform CNN feature maps may behave in unpredictable ways.

[4]Ideally, these effects would cancel out, since a zoomed-in image with fewer pixels could have the same resolution as the original. In practice, the STN learns to scale the image by less than 2x, which means that the digits are lower resolution after transformation. In addition, since the STN only learns to zoom in after a while, the network unavoidably processes digits at two different resolutions in the beginning.

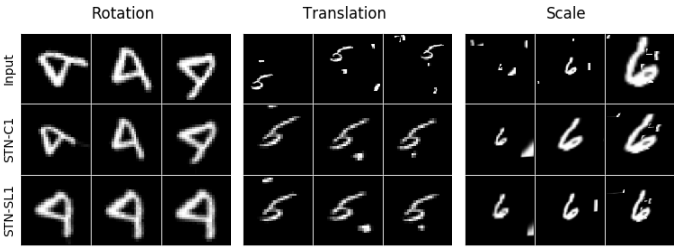


Fig. 5. *Illustration of how STN-C1 and STN-SL1 compensate for different perturbations.* The top row shows three digits rotated (first image), translated (second image), or scaled (third image) in three different ways. The middle row and bottom row show how STN-C1 and STN-SL1 transform the digits in the top row. STN-C1 does not compensate for rotations at all, but it successfully localizes and zooms in on translated digits. It only compensates somewhat for scaling. STN-SL1 finds a canonical pose for all perturbations. Note that STN-C1 does not transform the input image, so the middle row is just an illustration of the transformation parameters that are normally used to transform its CNN feature map.

TABLE II
AVERAGE CLASSIFICATION ERROR AND STANDARD DEVIATION ON THE MNIST TEST-SET, ACROSS 10 DIFFERENT RUNS.

| Network | R *(std)* | T *(std)* | S *(std)* |
|---|---|---|---|
| CNN | 1.71%(0.07) | 1.61%(0.06) | 1.38%(0.04) |
| STN-C0 | 1.08%(0.05) | 1.10%(0.11) | 0.85%(0.06) |
| STN-C1 | 1.32%(0.04) | 1.16%(0.03) | 0.96%(0.04) |
| STN-DL1 | 1.05%(0.02) | **1.08%**(0.07) | **0.77**%(0.04) |
| STN-SL1 | **0.98%**(0.06) | 1.13%(0.04) | 0.82%(0.06) |

*4) Concluding remarks:* As predicted by theory, it is significantly better to transform the input image when compensating for differences in rotation and scale, while transforming intermediate feature maps works reasonably well when compensating for differences in translation. STN-SL1 benefits from sharing parameters on the rotation task, but in the presence of large scale variations, it is better for the localization and the classification network to be independent of each other.

### *B. Street View House Numbers*

*1) Dataset:* In order to learn how the different STN architectures perform on a more challenging dataset, we evaluate them on the Street View House Numbers (SVHN) [8]. The photographed house numbers contain 1-5 digits each, and we preprocess them by taking 64x64 pixel crops around each sequence, as done in [1]. This allows us to compare our results with [1], who achieved good results on SVHN with an iterative ST that transforms feature maps. Since the dataset benefits from the use of a deeper network, it also allows us to test how the architectures' performance varies with their depth.

*2) Comparison with [1]:* We use the same baseline CNN as [1]. The classification network comprises 8 convolutional layers followed by 3 fully connected layers. Since the images can contain up to 5 digits, the output consists of 5 parallel softmax layers. Each predicts the label of a single digit.

As variations of the baseline CNN, [1] considers STNs with two different localization networks: One with a large

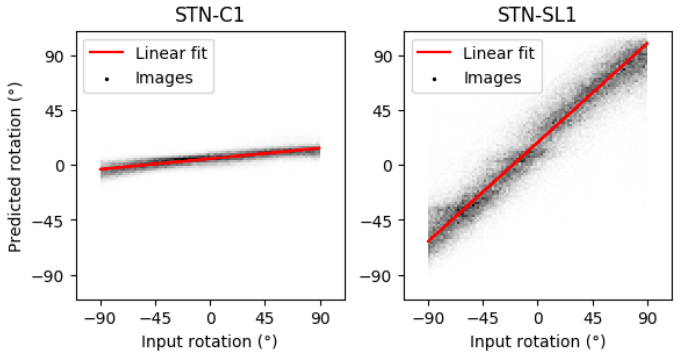


Fig. 6. *The rotation angle predicted by the ST module (y-axis) as a function of the rotation angle applied to the input image (x-axis).* The black points constitute a heatmap of 100 000 datapoints generated by random rotations of the MNIST test set, and reports the rotations done by a single ST. The red line corresponds to the best fit with orthogonal regression. STN-C1 cannot compensate for the rotations in a useful way, since it transforms CNN feature maps, while STN-SL1 directly counteracts the rotations by rotating the input.

localization network in the beginning, here denoted by STN-C0-large, and one with a small localization network before each of the first 4 convolutional layers, here denoted by STN-C0123. The large localization network uses two convolutional layers and two fully connected layers, while each of STN-C0123's localization networks uses two fully connected layers. As a further variation, we consider the network STN-SL0123, which is similar to STN-C0123 but always transforms the input, as described in Section III-A. STN-C0123 and STN-SL0123 are illustrated in Figure 4.

The average classification errors of these networks are shown in Table III. STN-SL0123 achieves the lowest error, which shows that it is better to transform the input than to transform CNN feature maps, also on the SVHN dataset.

TABLE III
CLASSIFICATION ERRORS ON THE SVHN DATASET, AVERAGED OVER 3 RUNS, COMPARING OUR IMPLEMENTATION WITH [1]

| Network | Error [1] | Error (ours) |
|---|---|---|
| CNN | 4.0% | 3.88% |
| STN-C0-large | 3.7% | 3.69% |
| STN-C0123 | 3.6% | 3.61% |
| STN-SL0123 | - | **3.49%** |

*3) Comparing STs at different depths:* If an ST is placed deeper into the network, it can use deeper features to predict transformation parameters, but the problem with spatial transformations of CNN feature maps may get worse. STN-DLX and STN-SLX would not suffer from the latter, but might benefit from the former. We test this by placing STs at depths 0, 3, 6, or 8 in our base classification network, using the small localization network from the previous section.

The results are shown in Table IV. STN-C3, STN-DL3, and STN-SL3 perform better than STN-C0, indicating that STN-C0's inability to find the correct transformation parameters causes more problems than STN-C3 causes by transforming the feature map at depth 3. However, at depths 6 and 8, STN-CX becomes worse than not using any ST at all (see Table III) and STN-DLX performs worse than at depth 3. This stands in sharp contrast to STN-SLX, where the classification error keeps decreasing, reaching 3.26% for STN-SL8.

TABLE IV
MEAN SVHN CLASSIFICATION ERROR OF STN-CX, STN-DLX AND STN-SLX AT 4 DIFFERENT DEPTHS, ACROSS 3 RUNS.

| Depth | STN-CX | STN-DLX | STN-SLX |
|---|---|---|---|
| $X = 0$ | 3.81% | - | - |
| $X = 3$ | 3.70% | 3.48% | 3.54% |
| $X = 6$ | 3.91% | 3.75% | 3.29% |
| $X = 8$ | 4.00%* | 3.76% | **3.26%** |

$*$ One run diverged to $> 99\%$ classification error. The error is the average of three runs where that did not happen.

This shows that localization networks benefit from using deep representations, if they use filters from the classification network, while still transforming the input. It is not achievable by spatially transforming CNN feature maps, since transforming deep feature maps causes too much distortion. Just using deeper localization networks does not always work, either, as these results show that they fail to find appropriate transformation parameters at greater depths.

Note that the larger localization networks of STN-DLX and STN-SLX take more time to train. STN-SL0123 takes 1.8 times as long to train as STN-C0123, while STN-SL8 takes 1.6 times as long as STN-C0123.

*4) Comparing iterative STNs:* Given that the use of deeper localization networks helps performance, and that [26] showed that iterative methods improve performance, a natural question is whether using iterative methods is a replacement for using deeper features or if iterations and deeper features are complimentary. To answer this, we train STN-C0, STN-DL3, STN-SL3, STN-SL6, and STN-SL8 with two iterations.

The second iteration does not change STN-C0's performance (mean error 3.80%). However, STN-SL3 improves significantly (mean error 3.38%), and STN-SL6 improves slightly (mean error 3.26%). STN-SL8 and STN-DL3 both perform worse with a second iteration. If a third iteration is added to STN-SL3 and STN-SL6 *during testing*, the errors further decrease to 3.33% and 3.18%, respectively.

This shows that multiple iterations do not improve performance when the localization network is too weak (as with STN-C0). Thus, multiple iterations is not a replacement for using deeper features, but can confer additional advantages. Here, parameter-shared networks are the only ones that benefit.

### C. Plankton

*1) Dataset:* Finally, we test the STN architectures on PlanktonSet [9]. This dataset is challenging, because it contains 121 classes and only about 30 000 training samples.

*2) Networks:* We use network architectures inspired by [29]. As base classification network, we use a CNN consisting of 10 convolutional layers with maxpooling after layers 2, 4, 7, and 10; followed by two fully connected layers before the final softmax output. As base localization network, two fully connected layers are used. All hyperparameters were chosen through experiments on the validation set.

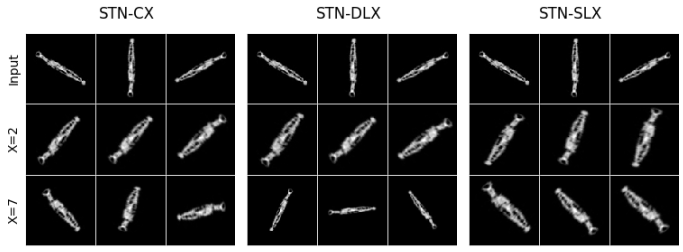


Fig. 7. *A rotated plankton transformed by various networks.* The top row contains the input images, the middle row contains the top images transformed by STN-C2, STN-DL2, and STN-SL2, and the bottom row contains the top images transformed by STN-C7, STN-DL7, and STN-SL7. In the middle row, all networks find a canonical angle. However, in the bottom row, only STN-SL7 finds a canonical angle, while STN-C7 suffer from not being able to transform the input directly, and STN-DL7's localization network is too deep.

*3) Results:* Table V displays the results, evaluated on the test set. The best architectures are STN-SL2 and STN-SL4. As on SVHN and MNIST, networks that transform the input are better than STN-CX. Also similar to the results on SVHN is that STN-DLX is significantly better for low values of X.

TABLE V
MEAN PLANKTONSET CLASSIFICATION ERROR OF STN-CX, STN-DLX, AND STN-SLX AT 5 DIFFERENT DEPTHS, ACROSS 4 RUNS.

| Depth | STN-CX | STN-DLX | STN-SLX |
|---|---|---|---|
| $X = 0$ | 22.1% | - | - |
| $X = 2$ | 22.3% | 21.6% | **21.5%** |
| $X = 4$ | 22.5% | 21.7% | **21.5%** |
| $X = 7$ | 22.9% | 22.7% | 21.6% |
| $X = 10$ | 22.7% | 22.7% | 21.9% |

Increasing the depth of STN-SLX does not decrease performance as much as it does for STN-DLX, but neither does it increase performance, as it does on SVHN. This is likely caused by differences between the datasets. Identifying the interesting objects in SVHN is not a trivial task, because the digits must be distinguished from the background. Since the classification network must be able to identify the digits, the localization network can be expected to benefit from sharing more layers. However, on PlanktonSet, the interesting objects are easily identifiable against the black background, which makes fewer layers sufficient. In addition, the easy separability of planktons and background allows for large-scale data augmentation on PlanktonSet, which makes the STs task both easier and less important for classification accuracy.

*4) Rotational invariance:* Figure 7 shows how the architectures transform a plankton rotated in different ways. In contrast to STN-C1 on MNIST (see Figure 5 and Figure 6), STN-C2 has learned to transform the example image to a canonical angle. Despite this, STN-C0, STN-DL2 and STN-SL2 perform substantially better in Table V. This shows that STN-CX's problem is not that it is difficult to find correct transformation parameters; it is the problem that even a "correct" transformation does not yield proper invariance. In this case, it is nonetheless beneficial for STN-C2 to rotate the plankton to a canonical angle, while transformations made by the deeper STN-C7 introduce enough distortions that it is better to do nothing.

*5) Comparing iterative STNs:* Training STN-C0 with a second iteration improves it somewhat, yielding mean error 21.8%, but training it with a third iteration decreases performance. STN-SL2, STN-SL7, and STN-DL2 are all slightly improved by a second iteration (mean error 21.4%, 21.5%, and 21.5%), while the performance of STN-SL4 and STN-DL4 decreases.

We, again, observe that multiple iterations is not a substitute for deeper features when predicting transformations, as the deeper networks remain better than STN-C0.

We also observed that the *training error* decreases much more than the test error, when a second iteration is added. For example, STN-SL2, STN-DL4, and STN-SL7 all reduce their training error with 1.7% or more. This suggests that the second iterations make the STs substantially more effective, but that this mostly leads to overfitting.

## V. SUMMARY AND CONCLUSIONS

We have presented a theoretical argument why STNs cannot transform CNN feature maps in a way that enables invariant recognition for other image transformations than translations. Investigating the practical implications of this result, we have shown that this inability is clearly visible in experiments and negatively impacts classification performance. In particular, while STs transforming feature maps perform adequately when compensating for differences in translation, STs transforming the input perform better when compensating for differences in rotation and scale. Our results also have implications for other approaches that spatially transform CNN feature maps, pooling regions, or filters. We do not argue that such methods cannot be beneficial, but we do show that these approaches have limited ability to achieve invariance.

Furthermore, we have shown that STs benefit from using deep representations when predicting transformation parameters, but that localization networks become harder to train as they grow deeper. In order to use representations from the classification network, while still transforming the input, we have investigated the benefits of sharing parameters between the localization and the classification network. Our experiments show that parameter-shared localization networks remain stable better as they grow deeper, which has significant benefits on complex datasets. Finally, we find that iterative image alignment is not a substitute for using deeper features, but that it can serve a complimentary role.

## REFERENCES

[1] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2017–2025.
[2] C. B. Choy, J. Gwak, S. Savarese, and M. Chandraker, "Universal correspondence network," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 2414–2422.
[3] J. Li, Y. Chen, L. Cai, I. Davidson, and S. Ji, "Dense transformer networks," *arXiv preprint arXiv:1705.08881*, 2017.
[4] S. Kim, S. Lin, S. R. JEON, D. Min, and K. Sohn, "Recurrent transformer networks for semantic correspondence," in *Advances in Neural Information Processing Systems (NIPS)*, 2018, pp. 6126–6136.

[5] Z. Zheng, L. Zheng, and Y. Yang, "Pedestrian alignment network for large-scale person re-identification," *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.
[6] T. S. Cohen, M. Geiger, and M. Weiler, "A general theory of equivariant CNNs on homogeneous spaces," in *Advances in Neural Information Processing Systems (NIPS)*, 2019, pp. 9142–9153.
[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[8] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.
[9] R. K. Cowen, S. Sponaugle, K. Robinson, J. Luo, O. S. University, and H. M. S. Center, "PlanktonSet 1.0: Plankton imagery data collected from F.G. Walton Smith in Straits of Florida from 2014-06-03 to 2014-06-06 and used in the 2015 National Data Science Bowl (NCEI accession 0127422)." [Online]. Available: https://accession.nodc.noaa.gov/0127422
[10] B. D. Lukas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Image Understanding Workshop*, 1981.
[11] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani, "Hierarchical model-based motion estimation," in *European Conference on Computer Vision (ECCV)*. Springer, 1992, pp. 237–252.
[12] T. Lindeberg and J. Gårding, "Shape-adapted smoothing in estimation of 3-D depth cues from affine distortions of local 2-D structure," *Image and Vision Computing*, vol. 15, pp. 415–434, 1997.
[13] A. Baumberg, "Reliable feature matching across widely separated views," in *Proc. Computer Vision and Pattern Recognition (CVPR)*, Hilton Head, SC, 2000, pp. I:1774–1781.
[14] K. Mikolajczyk and C. Schmid, "Scale & affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, 2004.
[15] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. van Gool, "A comparison of affine region detectors," *International Journal of Computer Vision*, vol. 65, no. 1–2, pp. 43–72, 2005.
[16] L. Sifre and S. Mallat, "Rotation, scaling and deformation invariant scattering for texture discrimination," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 1233–1240.
[17] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International Conference on Machine Learning (ICML)*, 2016, pp. 2990–2999.
[18] R. Kondor and S. Trivedi, "On the generalization of equivariance and convolution in neural networks to the action of compact groups," in *International Conference on Machine Learning (ICML)*, 2018, pp. 2752–2760.
[19] T. Lindeberg, "Provably scale-covariant continuous hierarchical networks based on scale-normalized differential expressions coupled in cascade," *Journal of Mathematical Imaging and Vision*, vol. 62, no. 1, pp. 120–148, 2020.
[20] A. Arcos-García, J. A. Alvarez-Garcia, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Networks*, vol. 99, pp. 158–165, 2018.
[21] T. V. Souza and C. Zanchettin, "Improving deep image clustering with spatial transformer layers," in *International Conference on Artificial Neural Networks (ICANN)*. Springer, 2019, pp. 641–654.
[22] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *European Conference on Computer Vision (ECCV)*. Springer, 2014, pp. 346–361.
[23] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Int. Conf. on Learning Representations (ICLR)*, 2016.
[24] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proc. International Conference on Computer Vision (ICCV)*, 2017, pp. 764–773.
[25] B.-I. Cîrstea and L. Likforman-Sulem, "Tied spatial transformer networks for digit recognition," in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2016, pp. 524–529.
[26] C.-H. Lin and S. Lucey, "Inverse compositional spatial transformer networks," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2568–2576.
[27] K. Lenc and A. Vedaldi, "Understanding image representations by measuring their equivariance and equivalence," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 991–999.
[28] Y. Jansson, M. Maydanskiy, L. Finnveden, and T. Lindeberg, "Inability of spatial transformations of CNN feature maps to support invariant recognition," *arXiv preprint:2004.14716*, 2020.
[29] J. Burms, P. Buteneers, J. Degrave, S. Dieleman, I. Korshunova, A. van den Oord, and L. Pigou. Classifying plankton with deep neural networks. [Online]. Available: https://benanne.github.io/2015/03/17/plankton.html

# APPENDIX

## A.1 DETAILS OF MNIST EXPERIMENTS

### *Perturbed variants*

The rotated variant (R) is generated by random rotations of the MNIST training set, uniformly chosen between -90° and 90°. Bilinear interpolation is used for resampling.

The translated variant (T) is generated by placing a digit from the MNIST training set at a random location in a 60x60 image. In addition, noise is added by choosing 6 random images from the MNIST training set, choosing a random 6x6 square in each of them, and adding those squares to random locations in the 60x60 image.

The scaled variant (S) is generated by scaling images from the MNIST training set with a random scaling factor between 0.5 and 4, uniformly sampled on a logarithmic scale. The resulting image is placed at the center of a 112x112 image. In addition, noise is added by choosing 6 random images from the MNIST training set, choosing a random 6x6 square in each of them, scaling the square a random amount between 0.5 and 4 (again uniformly sampled on a logarithmic scale), and adding those squares to random locations in the 112x112 image. Bilinear interpolation is used for resampling.

All variants are normalized to mean 0 and standard deviation 1 after perturbation.

### *Architectures*

We use the same classification network for (R), (T), and (S), but vary the localization network. In addition, we vary the number of filters and neurons in the classification and localization networks, to keep the number of learnable parameters approximately constant. The network architectures and the number of learnable parameters are presented in Table VI. C($N$) denotes a convolutional layer with $N$ filters, and F($N$) denotes a fully connected layer with $N$ neurons. Note that the first layers in STN-SL1's localization and classification networks shares parameters, so STN-SL1 uses somewhat fewer parameters than STN-DL1.

The final layer in the classification-architecture (not included in the table) is a 10-neuron softmax output layer, while the final layer in the localization network is a 6-neuron output describing an affine transformation. All layers use ReLU as activation function. The classification network's first convolutional layer has filter-size 9x9, and the second has filter-size 7x7. Convolutional layers in localization networks use filter-size 5x5. Each convolutional layer is followed by a 2x2 max-pooling with stride 2, except for the last convolutional layer in

TABLE VI
THE TYPES OF LAYERS AND NUMBER OF PARAMETERS IN EACH OF THE NETWORKS USED ON MNIST. C(N) DENOTES A CONVOLUTIONAL LAYER WITH N FILTERS, AND F(N) DENOTES A FULLY CONNECTED LAYER WITH N NEURONS.

| Dataset | Rotation | | | Translation | | | Scale | | |
|---|---|---|---|---|---|---|---|---|---|
| Network | Classification | Localization | Params | Classification | Localization | Params | Classification | Localization | Params |
| CNN | C(32),C(32) | - | 54122 | C(21),C(32) | - | 66692 | C(23),C(21) | - | 136674 |
| STN-C0 | C(16),C(32) | F(32),F(16)x2 | 53744 | C(16),C(32) | C(20)x2,F(10) | 67138 | C(16),C(16) | C(16)x2,F(16) | 136448 |
| STN-C1 | C(16),C(32) | F(16)x3 | 53984 | C(16),C(32) | C(20)x2,F(20) | 67088 | C(16),C(16) | C(16)x2,F(16) | 137072 |
| STN-DL1 | C(16),C(32) | C(16),F(16)x3 | 55296 | C(16),C(32) | C(16),C(20)x2,F(20) | 66800 | C(16),C(16) | C(16)x3,F(16) | 138384 |
| STN-SL1 | C(16),C(32) | C(16),F(16)x3 | 53984 | C(16),C(32) | C(16),C(20)x2,F(20) | 65488 | C(16),C(16) | C(16)x3,F(16) | 137072 |

the localization network of STN-C1, STN-DL1, and STN-SL1 on (T).

For STN-C1, the ST is placed after the first convolution's max-pooling layer, while STN-DL1 and STN-SL1 includes a copy of the max-pooling layer in their localization networks. In order to keep the number of parameters similar across all networks, the 112x112 image is downsampled by 2x before it is processed by STN-C0's localization network, on (S).

The ST on (T) produces an image with half the pixel width of its input, as is done in [1]. This leads to STN-C1 having slightly more parameters than STN-DL1 and STN-SL1, as the image becomes downsampled further into the classification network.

On all variants, the ST uses bilinear interpolation. When the ST samples values outside the image, the value of the closest pixel is used.

*Training process*

The bias of the localization network's final layer is initialized to predict the identity transformation, while the weights are initialized to 0. All other learnable parameters are initialized uniformly, with the default bounds in Pytorch 1.3.0.

During training, we use a batch size of 256. All networks are trained for 50 000 iterations with the initial learning rate, before lowering it 10x and training for another 20 000 iterations. We use initial learning rate 0.02 on (R) and (S), and initial learning rate 0.01 on (T). During training, we continuously generate new, random transformations of the 60 000 MNIST training images. During testing, we randomly generate 10 different transformations of each of the 10 000 MNIST test images, and report the percentage error across all of them.

### A.2 QUANTIFYING ST PERFORMANCE

To measure the ST's ability to align perturbed images to a common pose, we need a measure of how consistent the pose of a set of perturbed digits are *after alignment*. We measure this using the standard deviation of the digits' *final pose* (i.e. final orientation, scale or translation) for each label, using some measure of the initial perturbation, $\theta$ and the ST's *compensation*, $\theta'$. The compensation is extracted from the affine transformation matrix

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

output by the localization network. The standard deviation is measured for each label separately, since each class might have a unique canonical pose. Since the translation data is 2-dimensional, we use the *standard distance deviation* on (T). All reported results are the average standard deviation, across all 10 class labels and across 10 networks trained with different random seeds.

*Estimating rotations from affine transformation matrix*

First, we describe a general method of estimating rotations from affine transformation matrices. Given a predicted affine transformation matrix (excluding translations)

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \tag{8}$$

we want to find the combined rotation and scaling transformation

$$R = \begin{pmatrix} S\cos\varphi & -S\sin\varphi \\ S\sin\varphi & S\cos\varphi \end{pmatrix} \tag{9}$$

that minimises the Frobenius norm of the difference between the model and the data

$$\min_{S,\varphi} \|A - R\|_F. \tag{10}$$

Introducing the variables $u = S\cos\varphi$ and $v = S\sin\varphi$, this condition can be written

$$\min_{u,v} (a_{11}-u)^2 + (a_{12}+v)^2 + (a_{21}-v)^2 + (a_{22}-u)^2. \tag{11}$$

Differentiating with respect to $u$ and $v$ and solving the resulting equations gives

$$\begin{aligned} S &= \sqrt{u^2+v^2} \\ &= \frac{1}{2}\sqrt{a_{11}^2 + a_{12}^2 + a_{21}^2 + a_{22}^2 + 2(a_{11}a_{22} - a_{12}a_{21})}, \end{aligned} \tag{12}$$

$$\tan\varphi = \frac{v}{u} = \frac{a_{21} - a_{12}}{a_{11} + a_{22}}. \tag{13}$$

Thus, we can estimate the amount of rotation from an affine transformation matrix in a least-squares sense from

$$\varphi = \arctan\left(\frac{a_{21} - a_{12}}{a_{11} + a_{22}}\right) + n\pi. \tag{14}$$

*Measuring rotations on (R)*

On (R), the perturbation $\theta$ is the angle with which the digit is initially rotated. To estimate the rotation $\theta'$ made by the ST, we apply the method described in the previous section. Choosing the sign of $\theta$ and $\theta'$ so that a positive rotation is in the same direction, the final pose (rotation) is defined to be $\theta + \theta'$.

Note that, in Figure 6, the x-axis is $\theta$ and the y-axis $-\theta'$. These graphs were generated from the median models (specifically, the 5th best) among the 10 trained models, as measured by the standard deviation of the final pose.

*Measuring translation on (T)*

On (T), the perturbation $\theta = (x, y)$ is the distance in pixels between the middle of the 28x28 MNIST image and the middle of the 60x60 image that it is inserted in, horizontally and vertically. The ST's horizontal and vertical translation is extracted as $\theta' = (x', y') = m(a_{13}, a_{23})$, where the sign of $m$ is chosen so that $\theta$ and $\theta'$ define a positive translation in the same direction. In order to have $\theta'$ measure distance in pixels (in the original image), we choose $|m| = 29.5$ for STN-C0, STN-DL1, and STN-SL1, and $|m| = 25$ for STN-C1 (see next paragraph for a more detailed explanation). Since $\theta$ and $\theta'$ are 2-dimensional, we measure the standard distance deviation of the final pose $\theta + \theta'$, as follows: For a dataset of $n$ images of a particular label, with perturbations $(\theta_i)_{1 \leq i \leq n}$ and ST transformations $(\theta'_i)_{1 \leq i \leq n}$, the *mean translation* is $\bar{\theta} = (\bar{x}, \bar{y}) = \Sigma_{i=1}^{n}(\theta_i + \theta'_i)$. Then, the standard distance deviation of the final pose is

$$\sqrt{\Sigma_{i=1}^{n}((x_i + x'_i - \bar{x})^2 + (y_i + y'_i - \bar{y})^2)}$$

As mentioned, the matrix's translational elements are multiplied by 29.5 for networks that translate the input image, and 25 for STN-C1. This is because we want to measure the distance in units of pixels. Pytorch's spatial transformer module interprets a translation parameter of 1 as a translation of half the image-width. Since the image is 60x60 pixels wide, and pixels on the edge are assumed to be at the very end of the image,[5] the image-width is $60 - 1 = 59$. Thus, a translation parameter of 1 corresponds to 29.5 pixels, for networks that transform the input. However, STN-C1 transforms an image that has been processed by a 9x9 convolutional layer and a max pooling layer. Since the 9x9 convolution is done without padding, it shrinks the image to be 52x52, after which the max pooling shrinks it to be 26x26. Thus, the image-width is $26 - 1 = 25$, and a translation parameter of 1 corresponds to a translation of 12.5 feature-map neurons. However, two adjacent neurons after max pooling are on average computed from regions that are twice as distant from each other as two adjacent pixels in the original image. Thus, a translation parameter of 1 applied after the max pooling corresponds to a translation of $2 \cdot 12.5 = 25$ pixels in the input image.

[5]This description applies to the behavior in Pytorch 1.3.0 and earlier. From version 1.4.0, the default behavior is changed such that edge-pixels are considered to be half a pixel away from the end of the image.

This adjustment needs to be applied whenever STN-C1's transformations are compared with transformations of the input image. In particular, in Figure 5, the translational elements of STN-C1's predicted matrix was multiplied by $\frac{25}{29.5}$ before transforming the input image.

*Measuring scaling on (S)*

On (S), the perturbation $\theta$ is measured as the $\log_2$ of the scaling factor squared. The ST's transformation is measured as $\theta' = \log_2(|A|) = \log_2(a_{11}a_{22} - a_{12}a_{21})$. Choosing the sign of $\theta$ and $\theta'$ so that a positive value scales up the image, for both, the final pose (scale) is measured as $\theta + \theta'$.

*Note on the predicted transformation matrix*

When using spatial transformer networks, the transformation predicted by the localization network is used to transform *the points from which the image is resampled*. This corresponds to the inverse transformation of *the image itself*.

For the methods used to extract the rotation and scaling, this changes the sign of $\theta'$, which must be accounted for before summing $\theta$ and $\theta'$. For the translation, it does not only change the sign, but it also allows us to directly extract the translation as $m(a_{13}, a_{23})$. For the inverse transformation matrix, that transforms the image directly,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ 0 & 0 & 1 \end{pmatrix} \tag{15}$$

the correct translation parameters would be

$$\theta' = -m \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}^{-1} \begin{pmatrix} b_{13} \\ b_{23} \end{pmatrix}. \tag{16}$$

This is because $(b_{13}, b_{23})$ corresponds to a translation *after* the purely linear transformation has been applied, which means that it may act on a different scale and in a different direction than the original perturbation $\theta$. This is accounted for by first applying the inverse transformation in (16). In addition, the same factor $m$ must be applied to convert to units of pixels, but with the reverse sign.

### A.3 Details of SVHN experiments

The SVHN data set contains 235 754 training images and 13 068 test images, where each image contains a digit sequence obtained from house numbers in natural scene images. We follow [1] in how we generate the dataset, using 64x64 crops around each digit sequence. Each color channel is normalized to mean 0 and standard deviation 1.

*Architectures*

The CNN, STN-C0-large, and STN-C0123 architectures exactly correspond to the CNN, STN-Single, and STN-Multi architectures, respectively, in [1]. Using *C(N)* to denote a convolutional layer with *N* filters of size 5x5, *MP* to denote 2x2 max pooling layer with stride 2, and *F(N)* to denote a fully connected layer with *N* neurons, the classification network is *C(48)-MP- C(64)-C(128)-MP- C(160)-C(192)-MP- C(192)-C(192)-MP- C(192)-F(3072)-F(3072)-F(3072)*, followed by 5

parallel *F(11)* softmax output layers. STN-C0-large uses a localization network with architecture *C(32)-MP-C(32)-F(32)-F(32)*. STN-C0123 and all architectures in Section IV-B3 use localization networks with architecture *F(32)-F(32)*. Each layer in the classification network except for the first uses dropout with probability 0.5. Localization networks do not use dropout, except for the layers that STN-DLX and STN-SLX copy from the classification network. All layers use ReLU as activation function.

The ST uses bilinear interpolation. When the ST samples values from outside the image, $(0, 0, 0)$ is used. When inserting STs at depth X, they are placed after the first X convolutional layers and after any max pooling or dropout layers that follow the last of those convolutional layers (i.e., we always place the ST right before the next convolutional or fully connected layer).

*Initialization*

The last layer of the localization network is initialized to predict the identity transformation. We do this by setting both weights and biases to zero, but when transforming images, we calculate the affine transformation matrix from the 6 output parameters $o_1, ..., o_6$ as

$$\begin{pmatrix} o_1 + 1 & o_2 & o_3 \\ o_4 & o_5 + 1 & o_6 \\ 0 & 0 & 1 \end{pmatrix}. \tag{17}$$

By letting $o_1, ..., o_6 = 0, ..., 0$ represent the identity transform, L2-regularization pushes the localization network towards the identity transformation, which improves classification performance.

All other learnable parameters are initialized uniformly, with the default bounds in Pytorch 1.3.0.

*Training process*

Our training process differs somewhat from [1]. We train all networks for 120 000 iterations with learning rate 0.03, 120 000 iterations with learning rate 0.003, and finally 60 000 iterations with learning rate 0.0003. We use batch size 128. The localization learning rate is always 0.01 times the base learning rate, except for STN-DLX, which significantly benefit from a higher learning rate multiplier. STN-DL3 uses multiplier 0.3, while STN-DL6 and STN-DL8 uses 0.1. In SL-STN, the shared layers use the classification network's learning rate. We use L2-regularization 0.0002 to regularize all layers.

When training the networks with *two iterations*, we reduce the localization networks' learning rate by factors of approximately 3 until further reductions do not increase performance. On STN-C0, the localization network's learning rate is 0.001 of the classification network's; on STN-SL3 it is 0.01; on STN-SL6 it is 0.003. On STN-SL8 and STN-DL3, the best results (mean error $3.39\%$ and $3.57\%$) are achieved when the localization network's learning rate multipliers are 0.0003 and 0.03, respectively, but this is worse than what the networks achieve with a single iteration.

All reported results are the mean classification error across 3 networks trained with different random seeds.

### A.4 Details of PlanktonSet experiments

PlanktonSet was originally used in a competition on the website Kaggle.[6] In the competition, 30% of the test set was used for public leaderboards, and 70% was used for final evaluation. With the same split, we use the first 30% of the test set as a validation set, while all reported results are evaluated on the final 70%. All such results are mean classification error across 4 models trained with different random seeds.

*Architectures*

The classification network architecture and the training process are inspired by those used in [29], but they are not identical. For example, we do not use ensemble learning, since this would not give any additional insight into the questions we investigate in this study.

Using *C(N)* to denote a convolutional layer with *N* filters of size 3x3, *MP* to denote a 3x3 max pooling layer with stride 2, and *F(N)* to denote a fully connected layer with *N* neurons, the classification network is *C(32)-C(32)-MP-C(64)-C(64)-MP-C(128)-C(128)-C(128)-MP-C(256)-C(256)-C(256)-MP-F(512)-F(512)* followed by a *F(121)* softmax output layer. Dropout layers with dropout probability 0.5 are placed before each of the final 3 fully connected layers. The base localization network is *F(64)-F(64)*. All layers use leaky ReLUs with negative slope $\frac{1}{3}$ as activation functions, except for the base localization network, which uses non-leaky ReLUs.

The ST uses bilinear interpolation. When the ST samples values from outside the image, the value of the closest pixel is used. When inserting STs at depth X, they are placed after any max-pooling layer at depth X.

*Training process*

The networks are initialized as on SVHN (Section A.3). All networks are trained for 215 000 iterations using a batch size of 64. The initial learning rate of 0.003 is divided by 10 after iteration 180 000, and divided by 10 again after iteration 205 000. All networks use L2-regularization 0.0002, and Nesterov momentum 0.8.

During training, each image is rescaled so that its longest side is 192 pixels, and placed in a 192x192 image without changing its aspect ratio. We apply data augmentation with rotations $\pm 180°$, translation $\pm 20$ pixels, shear $\pm 20°$, and scale factor $[\frac{1}{1.3}, 1.3]$. Each amount is sampled uniformly, except for scaling, which is sampled uniformly from a logarithmic scale. Finally, the image has a 50% chance of being horizontally flipped, before it is rescaled to 95x95 pixels. The images were not normalized.

For all STN architectures, we searched for localization learning rate multipliers between 0.01 and 1, with factors of approximately 3 between tested multipliers. For each architecture, we chose the learning rate multiplier with the smallest validation error, and evaluated the models trained with that multiplier on the test set (i.e., we did not retrain

[6]The competition website is at https://kaggle.com/c/datasciencebowl

the models). STN-C0, STN-C2, and STN-C4 were all trained with multiplier 0.3, while STN-C7 was trained with 0.03 and STN-C10 used 0.1. STN-SL2 and STN-SL4 were trained with 0.1, while STN-SL7 and STN-SL10 were trained with 0.03. STN-DL2 was trained with 0.3, while STN-DL4, STN-DL7, and STN-DL10 used 1, i.e. used the same learning rate in the localization network and classification network.

We used a similar process when choosing multipliers for the iterative STN architectures. For each architecture, we reduced the localization learning rate multiplier by factors of approximately 3 until further reductions did not decrease validation error. The models that were trained with the best multiplier were then evaluated on the test-set. This was 0.3 for STN-DL2, 0.01 for STN-SL2, and 0.003 for STN-SL7. STN-DL4 and STN-SL4 performed best when trained with 0.1 and 0.03, respectively, but the mean test errors (21.8% and 21.6%) were higher than when trained with a single iteration. When training STN-C0 with two iterations, the best multiplier was 0.1; when training it with three iterations, the best multiplier was 0.03, although the mean test error (22.1%) was higher than for 2 iterations.