Non-Parametric Transformation Networks for Learning General Invariances from Data

Dipan K. Pal, Marios Savvides
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
{dipanp,marioss}@andrew.cmu.edu

Abstract

ConvNets, through their architecture, only enforce invariance to translation. In this paper, we introduce a new class of deep convolutional architectures called Non-Parametric Transformation Networks (NPTNs) which can learn *general* invariances and symmetries directly from data. NPTNs are a natural generalization of ConvNets and can be optimized directly using gradient descent. Unlike almost all previous works in deep architectures, they make no assumption regarding the structure of the invariances present in the data and in that aspect are flexible and powerful. We also model ConvNets and NPTNs under a unified framework called Transformation Networks (TN), which yields a better understanding of the connection between the two. We demonstrate the efficacy of NPTNs on data such as MNIST with extreme transformations and CIFAR10 where they outperform baselines, and further outperform several recent algorithms on ETH-80. They do so while having the same number of parameters. We also show that they are more effective than ConvNets in modelling symmetries and invariances from data, without the explicit knowledge of the added arbitrary nuisance transformations. Finally, we replace ConvNets with NPTNs within Capsule Networks and show that this enables Capsule Nets to perform even better.

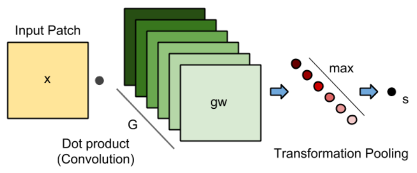
Introduction

The Fundamental Problem. One of the central problems of machine learning, has been supervised classification. A core challenge towards these problems is the encoding or learning of invariances and symmetries that exist in the training data. Indeed, methods which incorporate some known invariances or promote learning of more powerful invariances for a learning problem perform better in the target task given a certain amount of data. A number of ways exist to achieve this. One can present transformed versions of the training data as in (Niyogi, Girosi, and Poggio 1998), minimize auxiliary objectives promoting invariance during training as in (Hadsell, Chopra, and LeCun 2006) or pool over transformed versions of the representation itself as in (Liao, Leibo, and Poggio 2013; Pal, Juefei-Xu, and Savvides 2016; Pal et al. 2017).

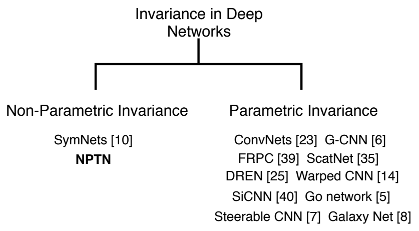
Convolutional Networks and Beyond. Towards this goal, ideas proposed in (LeCun et al. 1998) with the introduction of convolutional neural networks have proved to be very useful. Weight sharing implemented as convolutions followed by pooling resulted in the hard encoding of translation invariances (and symmetries) in the network. This made it one of the first applications of modelling invariance through a network's architecture itself. Such a mechanism resulted in greater regularization in the form of a structural or inductive bias in the network. With this motivation in mind, it is almost natural to ask whether networks which model more complicated invariances and symmetries perform better? Investigating architectures which invoke invariances not implicitly through the model's functional map but *explicitly* through an architectural property seems important.

New Dimensions in Network Architecture. Over the years, deep convolutional networks (ConvNets) have enjoyed a wide array of improvements in architecture. It was observed early on that a larger number of filters (width) in ConvNets led to improved performance, though with diminishing returns. (He et al. 2016; Huang et al. 2016) present another significant milestone with the development and maturity of residual connections and dense skip connections. Though there have been more advances in network architecture, many of the improvements have been derivatives of these two ideas (for instance (Zagoruyko and Komodakis 2016; Chen et al. 2017; Hu, Shen, and Sun 2017)). Recently however, (Sabour, Frosst, and Hinton 2017) introduced Capsule Nets which presented another potentially fundamental idea of encoding properties of an entity or an object in an activity vector rather than a scalar. With the goal of designing more powerful networks, ideas presented in this paper for modelling *general invariances in the same framework as ConvNets*, open up a new and potentially key dimension for architecture development.

Primary Contribution. In this work, we explore one such architecture class, called Transformation Networks (TN) which is a generalization of ConvNets. Additionally, we introduce a new type of TN using which a new class of networks can be built called Non-Parametric Transformation Networks (NPTNs). NPTNs networks have the ability to *learn* invariances to general transformations that are *observed* in the data which are non-parametric in nature (difficult to express mathematically). They can be easily im-



(a) A single NPTN node


Invariance in Deep Networks
Non-Parametric Invariance
SymNets [10]
NPTN
Parametric Invariance
ConvNets [23] G-CNN [6]
FRPC [39] ScatNet [35]
DREN [25] Warped CNN [14]
SiCNN [40] Go network [5]
Steerable CNN [7] Galaxy Net [8]

(b) Types of Invariances in Deep Networks

Figure 1: (a) Operation performed by a single Transformation Network (TN) node (single channel input and single channel output, Non-Parametric Transformation Networks are a kind of TN). TNs (and NPTNs) are a generalization of ConvNets towards learning general invariances and symmetries. The node has two main components (i) Dot product implemented as Convolution due to weight sharing and (ii) Transformation Pooling. The dot-product between the input patch (x) and a set of $|G|$ number of filters gw (green) is computed (this results in convolution when implemented with spatially replicated nodes). Here $|G| = 6$ (different shades of green indicate transformed templates). g indicates the transformation applied to the template or filter w. The resultant six output scalars (red) are then max-pooled over to produce the final output s (black). The pooling operation here is not spatially (as in vanilla ConvNets) but rather across the $|G|$ channels which encode non-parametric transformations. The output s is now invariant to the transformation encoded by the set of filters G. Each plane indicates a single feature map/filter.

plemented using standard off-the-shelf deep learning frameworks and libraries. Further, they can be optimized using vanilla gradient descent methods such as SGD. Unlike other methods that enforce additional invariances in convolutional architectures (Teney and Hebert 2016; Wu, Hu, and Kong 2015; Li et al. 2017), NPTNs do not need to transform the input, activation maps or the filters at any stage of the learning/testing process. They enjoy benefits of a standard convolutional architecture such as speed and memory efficiency while being more powerful in modelling invariances and being elegant in their operation. When forced to ignore any learnable transformation invariances in data, they gracefully reduce to vanilla ConvNets in theory and practice. However, when allowed to do so, they outperform ConvNets by capturing more general invariances.

Some properties of NPTNs. The architecture itself of an NPTN allows it to be able to learn powerful invariances from data provided the transformations are observable in data (a single node is illustrated in Fig. 1(a)). NPTNs do not enforce any invariance that is not observed in the data (although translation invariance can still be enforced through spatial pooling). Learning invariances from data is different and more powerful than enforcing known and specific invariances as is more common in literature (see Fig. 1(b)). Networks which enforce predefined symmetries (including vanilla ConvNets) force the same invariances at all layers which is a strong prior. More complex invariances are left for the network to learn using the implicit functional map as opposed to the explicit architecture. The proposed NPTNs have the ability to learn *different* and independent invariances for different layers and in fact for different channel paths themselves. Vanilla ConvNets enforce translation invariance through the convolution operation followed by a aggregation operation (either pooling or a second convolution layer) and only need to learn the *filter instantiation*. However, an NPTN node needs to learn 1) the instantiation of the filter and 2) the transformation that the particular node is invariant towards encoded as a *set* of filters. Each node learns these entities independently of each other which allows for a more flexible invariance model as opposed to architectures which replicate invariances across the network.

Prior Art

Although past applications of incorporating invariances were more specific and relatively narrow, development of such methods offers a better understanding of the importance of the problem. Though in this work we focus on deep architectures, it is important to note a number of works on modifications of Markov Random Fields and Restricted Boltzman Machines to achieve rotational invariance (Kivinen and Williams 2011; Sohn and Lee 2012).

Incorporating known invariances using deep networks. Convolutional architectures have seen many efforts to produce rotation invariant representations. (Fasel and Gatica-Perez 2006) and (Dieleman, Willett, and Dambre 2015) rotate the input itself before feeding it into stacks of CNNs and generating rotation invariant representations through gradual pooling or parameter sharing. (Teney and Hebert 2016; Wu, Hu, and Kong 2015; Li et al. 2017) rotate the convolution filters (a cheaper albeit still expensive operation) instead of transforming the input followed by pooling. A similar approach was explored for scale by (Xu et al. 2014). An interesting direction of research was explored by (Sifre and Mallat 2013) where the rotation, scale and translation invariant filters were fixed and non-trainable. (Cohen and Welling 2016a; Henriques and Vedaldi 2017) presented methods to incorporate parametric invariances using groups and warped convolutions. The transformations in (Henriques and Vedaldi 2017; Cohen and Welling 2016b) are known apriori and the sample grids and steerable filters are generated offline. This limits the capability to learn arbitrary and adaptive transformations. NPTNs need no such apriori knowledge apart that encoded in its architecture, can learn arbitrary non-parametric transformations and finally are simpler and perhaps more elegant in implementation.

Learning unknown invariances from data. In most real world problems, nuisance transformations present in data are unknown or too complicated to be parameterized by some function. (Anselmi et al. 2013) proposed a theory of group invariances called I-theory and explored its connection to general classification problems and deep networks. Based off the core idea of measuring moments of a group invariant distribution, multiple works had demonstrated efficacy of the ideas in more challenging real-world problems such as face recognition, though not in a neural network setting. See (Liao, Leibo, and Poggio 2013; Pal, Juefei-Xu, and Savvides 2016; Pal et al. 2017).

Learning unknown invariances from data using deep networks. Very few works have explored incorporating unknown invariances into deep networks. To the best of our knowledge, SymNets introduced in (Gens and Domingos 2014) was only one other previous study proposed deep networks which *learn* more general transformations. They were introduced as one of the first to model general invariances with back propagation. They utilize kernel based interpolation to tie weights enable them to model general symmetries. Nonetheless, the approach is complicated and difficult to scale. (Anselmi et al. 2017) provide sufficient conditions to enforce the learned representation to have symmetries learned from data. (Kavukcuoglu et al. 2009) modelled local invariances using pooling over sparse coefficients of a dictionary of basis functions. (Ngiam et al. 2010) achieved local invariance through complex weight sharing. Optimization was carried out through Topographic ICA and only carried out layer wise for deep networks. A separate approach towards modelling invariances was also developed where a normalizing transformation is applied to every input independently. This approach was applied to transforming auto encoders (Hinton, Krizhevsky, and Wang 2011) and Spatial Transformer Networks (Jaderberg et al. 2015).

The Transformation Network Paradigm

A Transformation Network (TN) is a feed forward network with its architecture designed to enforce invariance to some class of transformations through pooling. At the core of the framework is the TN node. A TN network consists of multiple such nodes stacked in layers. A single TN node is analogous to a single convolution layer with single channel input and single channel output.

Each TN node (single input channel and single output channel) internally consists of two operations 1) *(convolution)* the convolution operation with a bank of filters and 2) *(transformation pooling)* a max pooling operation *across* the set of the resultant convolution feature maps from the single input channel. Note the pooling is not spatial but rather across channels originating from the *same input channel* (this is different from MaxOut (Goodfellow et al. 2013) which pools over all input channels[1]). Fig. 1(a) illustrates the operation of single TN node with a single input/output channel for a single patch. The single channel illustrated in the figure takes in a single input feature map and convolves it with a bank of $|G|$ filters. Here $|G|$ is the cardinality (or size) of the set of transformations that the TN node is invariant towards, with G being the actual set itself. Next, the transformation max pooling operation max pools across the $|G|$ feature values to obtain a single TN activation value. When this node is replicated spatially, standard convolution layers can be utilized. Formally, a TN node denoted by Υ acting on a 2D image patch vectorized as $x \in \mathbb{R}^d$ can be defined as follows.

$$\Upsilon(x) = \max_{g \in G}(\langle x, gw \rangle) \tag{1}$$

Here, $\langle \ \rangle$ denotes a dot product and G is formally defined as a unitary group, *i.e.* a finite set obeying group axioms with each element being unitary. $w \in \mathbb{R}^d$ is the weight or filter, and gw is the group element g acting on w[2]. Therefore, the convolution kernel weights of a TN node are simply the transformed versions of w as transformed by the unitary group G. The TN node has to, only in theory, transform weight template w according to G to generate the rest of the filters to be pooled over during the transformation pooling stage. In practice however, these are simply stored as a set of templates or filters which only *implicitly* encode G through some constraints. For instance, vanilla ConvNets model the group G to be the translation group by enforcing it through the convolution operation. Thus, *a ConvNet can be exactly modelled by the TN framework when G is the translation group.* Gradient descent updates the filter w for a single node which immediately specifies the other filters in that node since they are the translated versions of w.

Significance of the Modeling of Transformations as Unitary Groups. The use of unitary groups to model transformations and invariances has emerged as a prominent theoretical tool (Anselmi et al. 2013; Pal et al. 2017). Group structure allows the computing of invariant objects such as group integrals. However, the significance of the unitary group lies in the fact that the vanilla ConvNet is invariant to translations, which is the simplest unitary group. Any framework that models invariance using the unitary group can be directly generalized to more complex groups such as rotations (rotation is an unitary transformation). This allows for seamless integration of the vanilla ConvNet into the theoretical framework and provides clear theoretical and practical connections to the same. Unitary groups in TNs allow them to exactly model ConvNets while generalizing to more complex networks invoking more complex invariances. The unitary group condition thus is only a useful theoretical tool, however should not be considered as a practical constraint.

Invariances in a TN node. Invariance in the TN node arises directly due to the symmetry of the unitary group structure of the filters. The max operation simply measures the infinite moment of an invariant distribution which invokes invariance (see (Anselmi et al. 2013)). We demonstrate this in the form of the following simple result[3].

Lemma 1. *(Invariance Property) Given vectors $x, w \in \mathbb{R}^d$, a unitary group $\mathcal{G}$ and $\Upsilon(x) = \max_{g \in G}(\langle x, gw \rangle)$, for any fixed $g' \in \mathcal{G}$, then $\Upsilon(x) = \Upsilon(g'x)$.*

Lemma 1 shows that for *any* input x (including test inputs), the node output is *invariant* to the transformation

[1]We discuss deviation from MaxOut in more detail in the supplementary.

[2]We use this shorter notation to reduce clutter.

[3]Proof in the supplementary.

group G. Note that invariance to test samples arises from two components. First, the group structure of G provides exact invariance and second, the unitary condition allows for the invariance properties to be extended to unseen test samples. This is interesting, since one does not need to observe any transformed version of say a test sample x during training which reduces sample complexity as explored by (Anselmi et al. 2013). Invariance is invoked for any arbitrary input x during test time, thereby demonstrating good generalization properties.

Relaxing towards Non-group and Non-Unitary Structure in a TN node (Towards NPTNs). Lemma 1 guarantees exact invariance perfectly for vanilla ConvNets and TNs which model G as having a group-structure and the unitary condition. For methods that do not enforce these conditions (unitary group conditions) in theory, no test generalization claim can be made. However, a number of studies have observed approximate albeit sufficient invariances in practice under this setting (Anselmi et al. 2013; 2017; Pal, Juefei-Xu, and Savvides 2016; Pal et al. 2017; Liao, Leibo, and Poggio 2013). The main motive for modelling transformations as unitary groups was to provide a theoretical connection to ConvNets and other methods that enforce other kinds of unitary invariance such as rotation invariance (Li et al. 2017; Wu, Hu, and Kong 2015). However, real-world data experiences a large array of transformations acting, which certainly lie outside the span of unitary transformations. Keeping this in mind, constraining the network to model only unitary transformations limits their ability to learn these more general invariances which are difficult to characterize.

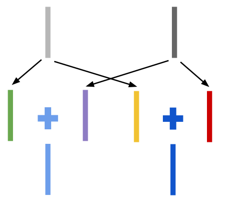
In the following section, we introduce a new kind of TN called the NPTN which is free from the constraints and limitations of unitary modelling, thereby being more expressive. Indeed, in our experiments, we observe that the NPTN architectures are able to perform better by learning invariance (signified by better test generalization) towards both 1) group structured, unitary and parametric transformations such as translations and rotations, and also towards 2) general non-group structured and non-parametric transformations (as in general object classification) which are difficult to characterize. Note that Lemma 1 only serve as a result for ConvNets and TNs, they do not characterize the invariance properties on NPTNs and general non-group non-unitary transformation. Investigation of such properties of NPTNs under the general setting is arduous and is outside the scope of this paper. Further, note that developing TNs and relating the unitary condition is not necessary for the development or motivation of NPTNs. TNs however provide a more elegant story and more importantly clarify the connection to vanilla ConvNets and helps to put our contribution in perspective.

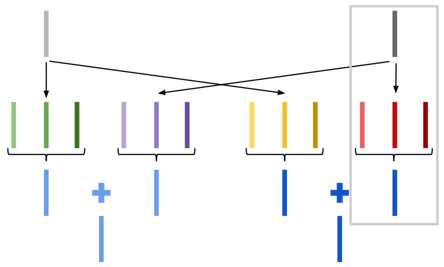
Non-Parametric Transformation Networks

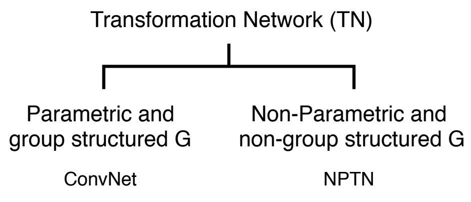
A Non-Parametric Transformation Network (NPTN) is a kind of TN that lacks any constraints on set of weights/filters w for any particular node. Here the set of filters G has two relaxations 1) need not have any group structure and 2) need not model any parametric and/or unitary transformations such as the translations or rotations. The term G in an NPTN represents simply a *set* of arbitrary filters modelling arbitrary transformations which are (potentially) non-parametric. One might think of the analogy from statistics where the Gaussian distribution is parametric, however for many real-world distributions a non-parametric tool such as a histogram is more appropriate. Note that however, there is no constraint that prevents a NPTN from learning translation and rotation invariance. In fact, in one of our experiments this is exactly the requirement. *Under the two relaxations, the invariance invoked to these arbitrary transformations in an NPTN would only be approximate.* Nonetheless and consistent with previous work, we find in our experiments that despite the approximation, there is much to be gained overall and the invariance invoked suffices in practice as also found by (Liao, Leibo, and Poggio 2013; Pal, Juefei-Xu, and Savvides 2016).

In an NPTN, both the entities (w, G) are learned, *i.e.* a NPTN node is tasked with learning both the filter instantiation w, *and* the set of transformations G to which the node is to be invariant towards. Nonetheless and rather importantly, no generation of transformed filters is necessary during any forward pass of a NPTN layer since the set G of transformed filters is always maintained and updated by gradient descent. This significantly reduces computational complexity compared to some previous works (Teney and Hebert 2016; Wu, Hu, and Kong 2015). Learning G from data is in sharp contrast with the vanilla convolutional node in which only the filter instantiation w is learned and where G is *hard coded* to be the translation group which is a parametric transformation (and also arguably the most elementary). Thus, ConvNets are a kind of Parametric Transformation Networks (PTNs) (see Fig. 2(c)). It is also important to note that however, setting $|G| = 1$ *and* incorporating spatial pooling, a NPTN is reduced to a vanilla ConvNet in practice. Compared to other approaches to learn and model general invariances such as SymNets (Gens and Domingos 2014), the NPTN architecture is elegantly simple and also a close generalization of ConvNets. Further, they can replace any convolution layer in any architecture making them versatile. We now describe the NPTN layer in more detail and discuss its characteristics.

NPTN Layer Structure, Forward Pass and Training. Fig. 2 illustrates a NPTN layer and compares it to a vanilla ConvNet layer. The NPTN layer shown has 2 input channels, 2 output channels and $|G| = 3$. For a NPTN layer with M input channels and N output channels, there would be MN NPTN nodes each identical to the one shown in Fig. 1(a). There are $|G|$ filters learned for each of the MN nodes, which each are convolved over the image similar to a vanilla ConvNet. Consider Fig. 2(b), once the input is convolved with the $M \times |G|$ filters, the M sets each with $|G|$ feature maps each are max pooled across the $|G|$ feature maps. More specifically, each of $|G|$ feature maps from a single input channel results in one intermediate feature map after max pooling (across the $|G|$ channels). This is the primary step that invokes invariances to transformations. After this operation there are MN intermediate feature maps which are transformation invariant. Now, the sum (alternatively the mean) of these M feature maps results in one *out-*


(a) Convolution layer


(b) NPTN layer



Transformation Network (TN)
Parametric and group structured G
Non-Parametric and non-group structured G
ConvNet
NPTN

(c) Relation between ConvNets and NPTNs

Figure 2: Comparison between (a) a standard Convolution layer and (b) a NPTN layer with $|G| = 3$. Each layer depicted has 2 input (shades of grey) and 2 output channels (shades of blue). The light grey rectangle encloses a single TN node (see Fig. 1(a)) The convolution layer has therefore, $2 \times 2 = 4$ filters, whereas the NPTN layer has $2 \times 2 \times 3 = 12$ filters. The different shades of filters in the NPTN layer denote transformed versions of the same filter (same color) which are max pooled over (support denoted by inverted curly bracket). The + operation denotes channel addition. In our experiments, we adjust the input/output channels of the NPTN layer to have the same number of parameters as the ConvNet baselines. (c) Shows how ConvNets and NPTNs are categorized under the TN framework.

put feature map or channel. This is repeated for each of the N output channels[4]. Note that there is no operation in this forward pass where the input or the filters need to be transformed on-the-fly, which makes it NPTNs computationally efficient compared to some previous models (Fasel and Gatica-Perez 2006; Dieleman, Willett, and Dambre 2015; Teney and Hebert 2016; Wu, Hu, and Kong 2015; Li et al. 2017). In fact, the computation complexity for NPTNs only increases with the order $|G|$ relative to a vanilla convolution layer. This is countered in our experiments by decreasing M and N, primarily to preserve the number of parameters. The NPTN layer can be trained using standard back-propagation. Back-propagation updates each of the $|G|$ filters of the NPTN independently depending on which of the $|G|$ filters is the 'winner' during the channel max pooling operation. Note again that this operation is very different from MaxOut which pools over inputs from *all* channels, whereas here each max operation pools over $|G|$ channels only from the *same input* channel[5]. Since the filters are not constrained to form any group, we do not expect to see any regular transformations being observed in the filters (for instance, rotated filters for rotation invariance). This might seem as a slight hindrance to interpretibility, nonetheless in our experiments, we find NPTNs perform well in specific applications where learning invariance from the data is necessary.

Invariance Modelling in NPTNs is Data Driven and Highly Flexible. It is important to note that though the architecture of NPTNs allows it to *learn* invariances, it does *not* in fact enforce any particular invariance by itself. NPTNs can only learn invariances to transformations that are observed in data, and thereby are even more benefited from data augmentation and natural variation. This is a critical difference between NPTNs and other works which do enforce specific invariances through design (see under Parametric Invariance in Fig. 1(b)). Another important and powerful property that emerges from having independent filter sets for each of the NPTN nodes in an entire network, is that each individual node can model invariance to a completely different transformation. Concretely, a single NPTN layer with M input channels and N output channels potentially can model MN different kinds of invariances. This is again in sharp contrast to ConvNets and other previous works such as (Teney and Hebert 2016; Wu, Hu, and Kong 2015; Li et al. 2017) where each layer and in fact each of the channel paths model exact same invariance, either translation, rotation or scale. NPTNs thus offers immense flexibility in invariance modelling.

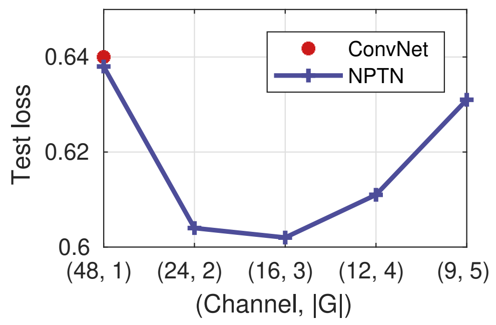


Figure 3: Test losses on CIFAR10 for the two layered network. Each network listed has the same number of filters.

Empirical Evaluation of NPTNs

Benchmarking against ConvNets on CIFAR10

In our first set of experiments, we benchmark and characterize the behavior of NPTNs against the standard ConvNets augmented with Batch Normalization (Ioffe and Szegedy 2015). The goal of this set of experiments is to observe

[4]We provide implementation details of NPTNs using standard libraries in the supplementary.

[5]We discuss deviation from MaxOut in more detail in the supplementary.

Method	Accuracy (%)
ConvNet (Khasanova and Frossard 2017)	80.1
STN (Jaderberg et al. 2015)	45.1
DeepScat (Oyallon and Mallat 2015)	87.3
HarmNet (Worrall et al. 2017)	94.0
TIGradNet (Khasanova and Frossard 2017)	95.1
NPTN (Ours)	**96.2**

Table 1: Test accuracy on ETH-80. All models including NPTNs and the ConvNet had roughly the same number of parameters (about 1.4M). Models followed the architecture described in (Khasanova and Frossard 2017). Results for models other than NPTN are cited as is from (Khasanova and Frossard 2017).

Rotations	0°	30°	60°	90°
ConvNet (36)	0.75	1.16	2.05	3.32
NPTN (36, 1)	0.68	1.27	2.01	3.36
NPTN (18, 2)	0.66	1.09)	1.72	2.88
NPTN (12, 3)	**0.63**	**1.08**	**1.71**	**2.76**
NPTN (9, 4)	0.66	1.17	1.83	2.94
Translations	0 pix	4 pix	8 pix	12 pix
ConvNet (36)	0.62	0.95	1.97	7.00
NPTN (36, 1)	**0.62**	0.88	1.84	7.22
NPTN (18, 2)	0.74	0.75	1.70	6.26
NPTN (12, 3)	0.66	**0.70**	**1.58**	**6.20**
NPTN (9, 4)	0.64	0.76	1.59	6.37

Table 2: Test error on progressively transformed MNIST with (a) random rotations and (b) random pixel shifts. NPTNs can learn invariances to arbitrary transformations from the data itself without any a priori knowledge. All models have same number of parameters.

whether learning non-parametric transformation invariance from complex visual data itself helps with object classification. For this experiment, we utilize the CIFAR10 dataset[6]. The networks we experiment with are not designed to compete with state-of-the-arts on this data but rather throw light into the behavior of NPTNs. We therefore utilize a small network, specifically a two layered network, for these experiments. Each layer block of the baseline ConvNets consist of the convolution layer, followed by batch normalization and the non-linearity (PReLU) and finally by a 2 by 2 spatial max pooling layer. Each corresponding NPTN network replaces only the convolution layer with the NPTN layer. Thus, NPTN is allowed to model non-parametric invariance in addition to the typically enforced translation invariance due to spatial max pooling. The two layered network baseline ConvNet has channels [3, 48, 16] with a total of $3 \times 48 + 48 \times 16 = 912$ filters. The NPTN variants in this experiment keep the total number of filters constant with 48 channels with $|G| = 1$ denoted by (48 1), 24 channels with $|G| = 2$ denoted by (24 2), and so on up until 9 channels with $|G| = 5$ (9 5). Fig. 3 shows the testing losses. Each network experimented with has the same number of parameters. We find all NPTN variants which learn a non-trivial set of transformations ($|G| > 1$) outperform the ConvNet baseline significantly, with NPTN $|G| = 3$ performing the best.

Benchmarking against other approaches: ETH-80

We now benchmark NPTNs against other approaches learning invariances on the ETH-80 dataset (Leibe and Schiele 2003). As our baseline, we follow the experimental setup and the specifications of the models described in (Khasanova and Frossard 2017). Note that for this experiment, our goal is not to attain state-of-the-art results, but rather benchmark against other related methods under a comparable setting. The dataset has 80 objects belonging to 8 classes. Each object has 41 images taken from a grid of different viewpoints on a hemisphere. Following (Khasanova and Frossard 2017), we resize the images to 50×50 and train on 2,300 images and test on the rest. The isometric transformations in the dataset present a good challenge for approaches to invoke invariance in a real-world setting. For this experiment, we compare against standard ConvNets, Spatial Transformer Networks (Jaderberg et al. 2015), DeepScat (Oyallon and Mallat 2015), HarmNet (Worrall et al. 2017) and TIGradNet (Khasanova and Frossard 2017). The NPTN architecture was chosen to by replacing the convolution layers in the ConvNet architecture in (Khasanova and Frossard 2017) with NPTN layers while setting $|G| = 3$ and reducing the number of channels to preserve the number of parameters. All models in this experiment (including NPTNs) have about 1.4M parameters. Table 1 presents the test accuracy on ETH-80. We find that NPTN outperforms these other high-performing algorithms on this task with an accuracy of 96.2 %. Thus, NPTNs despite having much simpler architecture and the same number of parameters, is able to perform well in a task where the primary nuisance transformation is due to varying 3D pose of the objects.

Learning Unknown Transformation Invariances from Data

We now demonstrate the ability of NPTN networks to learn invariances directly from data without any apriori knowledge. For this experiment, we augment MNIST with extreme a) random rotations b) random translations, *both* in training and testing data thereby increasing the complexity of the learning problem itself. For each sample, a random instantiation of the transformation was applied. For rotation, the angular range was increased, whereas for translations it was the pixel shift range. Table 2 presents these results. All networks in the table are two layered and have the exact same number of parameters. As expected, NPTNs match the performance of vanilla ConvNets when there were no additional transformations added (0° and 0 pixels)[7]. However, as the

[6]With standard data augmentation of random cropping after a 4 pixel pad, and random horizontal flipping. Training was for 300 epochs with the learning rate being 0.1 and decreased at epoch 150, and 225 by a factor of 10.

[7]NPTNs perform slightly better than ConvNets for 0° rotations because for all rotation experiments, small translations up to 2 pix-

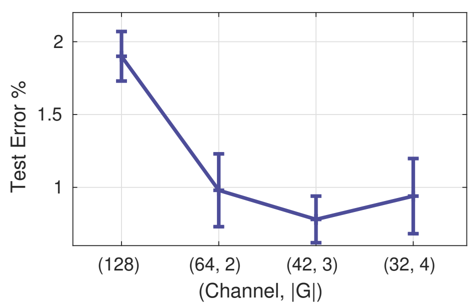


Figure 4: Test errors on MNIST for Capsule Nets augmented with NPTNs. (128) denotes a Capsule Network with a vanilla ConvNet. Other labels are NPTNs with (channels, $|G|$). The number of filters from left to right is $\{4224, 4160, 4074, 4128\}$. NPTNs significantly outperform ConvNets in Capsule Nets with fewer filters.

transformation intensity (range) is increased, NPTNs perform significantly better than ConvNets. Trends consistent with previous experiments were observed with the highest performance observed with NPTN ($|G| = 3$). This highlights the main feature of NPTNs, *i.e.* their ability to model arbitrary transformations observed in data without any apriori information and without changes in architecture whatsoever. They exhibit better performance in settings where both rotation invariance and *stronger* translation invariance is required (even though ConvNets are designed specifically to handle translations). This ability is something that previous deep architectures did not possess nor demonstrate.

NPTNs with Capsule Networks

Capsule Networks with dynamic routing were recently introduced as an extension of standard neural networks (Sabour, Frosst, and Hinton 2017). Since the original architecture is implemented using vanilla convolution layers, invariance properties of the networks are limited. Our goal for this experiment is to replace Convolution Capsule Nets with NPTN Capsules. We replace the convolution layers in the Primary Capsule layer of the published architecture with NPTN layers while maintaining the same number of parameters (by reducing number of channels and increasing $|G|$). Our baseline is the proposed CapsuleNet with 3 layers using a third party implementation in PyTorch[8]. The baseline convolution capsule layer had 128 output channels. The NPTN variants progressively decreased the number of channels as $|G|$ was increased. All other hyperparameters were preserved. The networks were trained on the 2-pixel shifted MNIST for 50 epochs with a learning rate of 10^{-3}. The performance statistics of 5 runs are reported in Fig. 4. We find that for roughly the same number of kernel filters (and parameters), Capsule Nets have much to gain from the use of NPTN layers (a significant test error decrease from 1.90 to 0.78 for $\frac{1}{3}$ of the baseline number of channels and $|G| = 3$). The learning of invariances within each capsule significantly increases efficacy and performance of the overall architecture.

els were applied only in training.

[8]https://github.com/dragen1860/CapsNet-Pytorch.git

Current Limitations

We observed that NPTNs and the idea of learning general non-parametric invariances form the data itself has potential and merits further investigation as a component in more broader studies. Nonetheless, our current implementation of NPTN in PyTorch suffers from high memory usage and high computation time partly due to channel shuffling. Given our limited computation resources as this time, the size and depth of the networks that we can train is subsequently limited. We therefore focus more on a systematic and thorough investigation and benchmarking of a smaller network on multiple tasks and experiments rather than larger scale experiments. Such studies will be within scope once more efficient function routines for specific tasks are available in deep learning packages.

Discussion

It is clear that the success of ConvNets is not the whole story towards solving perception. Studies into different aspects of network design will prove to be paramount in addressing the complex problem of not just visual but general perception.

The development of NPTNs offer one such design aspect, *i.e.* learning non-parametric invariances and symmetries directly from data. Through our experiments, we found that NPTNs can indeed effectively learn general invariances without any apriori information. Further, they are effective and improve upon vanilla ConvNets even when applied to general vision data as presented in CIFAR10 and ETH-80 with complex unknown symmetries. This seems to be a critical requirement for any system that is aimed at taking a step towards general perception. Assuming detailed knowledge of symmetries in real-world data (not just visual) is impractical and successful models would need to adapt accordingly.

In all of our experiments, NPTNs were compared to vanilla ConvNet baselines with the same number of filters (and thereby more channels). Interestingly, the superior performance of NPTNs despite having fewer channels indicates that better modelling of invariances is a useful goal to pursue during design. Explicit and efficient modelling of invariances has the potential to improve many existing architectures. Indeed, we outperform several state-of-the-art algorithms on ETH-80. In our experiments, we also find that Capsule Networks which utilized NPTNs instead of vanilla ConvNets performed much better. This motivates and justifies more attention towards architectures and other solutions that efficiently model general invariances in deep networks. Such an endeavour might not only produce networks performing better in practice, it also promises to deepen our understanding of deep networks and perception in general.

References

Anselmi, F.; Leibo, J. Z.; Rosasco, L.; Mutch, J.; Tacchetti, A.; and Poggio, T. 2013. Unsupervised learning of invariant representations in hierarchical architectures. *arXiv preprint arXiv:1311.4158*.

Anselmi, F.; Evangelopoulos, G.; Rosasco, L.; and Poggio, T. 2017. Symmetry regularization. Technical report, Center for Brains, Minds and Machines (CBMM).

Chen, Y.; Li, J.; Xiao, H.; Jin, X.; Yan, S.; and Feng, J. 2017. Dual path networks. In *Advances in Neural Information Processing Systems*, 4470–4478.

Cohen, T., and Welling, M. 2016a. Group equivariant convolutional networks. In *International Conference on Machine Learning*, 2990–2999.

Cohen, T. S., and Welling, M. 2016b. Steerable cnns. *arXiv preprint arXiv:1612.08498*.

Dieleman, S.; Willett, K. W.; and Dambre, J. 2015. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society* 450(2):1441–1459.

Fasel, B., and Gatica-Perez, D. 2006. Rotation-invariant neoperceptron. In *Pattern Recognition, ICPR. 18th International Conference on*, volume 3, 336–339. IEEE.

Gens, R., and Domingos, P. M. 2014. Deep symmetry networks. In *Advances in neural information processing systems*, 2537–2545.

Goodfellow, I.; Warde-Farley, D.; Mirza, M.; Courville, A.; and Bengio, Y. 2013. Maxout networks. In *International Conference on Machine Learning*, 1319–1327.

Hadsell, R.; Chopra, S.; and LeCun, Y. 2006. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, 1735–1742. IEEE.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Henriques, J. F., and Vedaldi, A. 2017. Warped convolutions: Efficient invariance to spatial transformations. In *International Conference on Machine Learning*.

Hinton, G. E.; Krizhevsky, A.; and Wang, S. D. 2011. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, 44–51. Springer.

Hu, J.; Shen, L.; and Sun, G. 2017. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*.

Huang, G.; Liu, Z.; Weinberger, K. Q.; and van der Maaten, L. 2016. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456.

Jaderberg, M.; Simonyan, K.; Zisserman, A.; et al. 2015. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, 2017–2025.

Kavukcuoglu, K.; Fergus, R.; LeCun, Y.; et al. 2009. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 1605–1612. IEEE.

Khasanova, R., and Frossard, P. 2017. Graph-based isometry invariant representation learning. In *International Conference on Machine Learning*, 1847–1856.

Kivinen, J. J., and Williams, C. K. 2011. Transformation equivariant boltzmann machines. In *International Conference on Artificial Neural Networks*, 1–9. Springer.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Leibe, B., and Schiele, B. 2003. Analyzing appearance and contour based methods for object categorization. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, II–409. IEEE.

Li, J.; Yang, Z.; Liu, H.; and Cai, D. 2017. Deep rotation equivariant network. *arXiv preprint arXiv:1705.08623*.

Liao, Q.; Leibo, J. Z.; and Poggio, T. 2013. Learning invariant representations and applications to face verification. In *Advances in Neural Information Processing Systems*, 3057–3065.

Ngiam, J.; Chen, Z.; Chia, D.; Koh, P. W.; Le, Q. V.; and Ng, A. Y. 2010. Tiled convolutional neural networks. In *Advances in neural information processing systems*, 1279–1287.

Niyogi, P.; Girosi, F.; and Poggio, T. 1998. Incorporating prior information in machine learning by creating virtual examples. *Proceedings of the IEEE* 86(11):2196–2209.

Oyallon, E., and Mallat, S. 2015. Deep roto-translation scattering for object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2867–2873.

Pal, D.; Kannan, A.; Arakalgud, G.; and Savvides, M. 2017. Max-margin invariant features from transformed unlabelled data. In *Advances in Neural Information Processing Systems 30*, 1438–1446.

Pal, D. K.; Juefei-Xu, F.; and Savvides, M. 2016. Discriminative invariant kernel features: a bells-and-whistles-free approach to unsupervised face recognition and pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5590–5599.

Sabour, S.; Frosst, N.; and Hinton, G. E. 2017. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, 3859–3869.

Sifre, L., and Mallat, S. 2013. Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1233–1240.

Sohn, K., and Lee, H. 2012. Learning invariant representations with local transformations. *arXiv preprint arXiv:1206.6418*.

Teney, D., and Hebert, M. 2016. Learning to extract motion from videos in convolutional neural networks. In *Asian Conference on Computer Vision*, 412–428. Springer.

Worrall, D. E.; Garbin, S. J.; Turmukhambetov, D.; and Brostow, G. J. 2017. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5028–5037.

Wu, F.; Hu, P.; and Kong, D. 2015. Flip-rotate-pooling convolution and split dropout on convolution neural networks for image classification. *arXiv preprint arXiv:1507.08754*.

Xu, Y.; Xiao, T.; Zhang, J.; Yang, K.; and Zhang, Z. 2014. Scale-invariant convolutional neural networks. *arXiv preprint arXiv:1411.6369*.

Zagoruyko, S., and Komodakis, N. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146*.