Neural Architectures Towards Invariant Representation Learning

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Dipan K. Pal

B.E., Electronics and Communications Engineering Department

Birla Institute of Technology, Mesra, India

M.S., Electrical and Computer Engineering Department

Carnegie Mellon University, Pittsburgh, USA

Carnegie Mellon UniversityPittsburgh, PA

December 2020

To my mom and dad, Preeti and Sudip Pal

Acknowledgments

First, I would like to thank my advisor and the chair of my doctoral committee Prof. Marios Savvides for taking me into his lab and giving me the opportunity to do this. Over the years, he and the lab in general have helped me realize the importance of engineering and what kind of real impact such engineering can indeed have. Storytelling is a great and useful art that few realize the importance of. It was early in my career when he showed me how a good story helps have a more effective connection with the listener. His style has certainly influenced mine over these years. Perhaps, the biggest support that Prof. Marios provided over these years was almost complete academic freedom. He let me pursue whatever problem I wished to work on even though a few of these had no direct practical impact to the lab. During times when I was struggling to get my work published, he reassured his belief in me and my work. This patience and freedom let me hone one of the most important skills as a researcher, how to pick a problem to work on. I am deeply in debt to him for his guidance, support and the invaluable lessons I have learnt under his wing over the years.

I would also like to thank my doctoral committee members, Prof. Vijayakumar Bhagavatula, Prof. John M. Dolan and Dr. Saad Bedros for serving on my committee and for their comments, guidance, and suggestions. I am very fortunate to have such a distinguished thesis panel for which I am forever grateful.

I would like to also thank the wonderful lab mates I have had over the years all of whom have done great work and have taught me so many things over the years. Shreyas Venugopalan, Utsav Prabhu, Sekhar Bhagavatula, Felix Xu, Keshav Sheshadri, Raied Aljadany, Vishnu Boddeti, Khoa Luu, Thi Hoang Ngan (Nancy) Le, Chenchen Zhu, Yutong Zheng, Ran Tao, Thanh Hai Phan, Uzair Ahmed and Sreena Nallamothu have been an absolute pleasure to work with. I have deeply enjoyed the conversations with Raied, Yutong, Felix, Uzair, Sreena, Utsav, Shreyas and Sekhar which have spawned many ideas some of which were published.

I also thank the CyLab and CMU staff who have always helped us with any problems and the innumerable last minute requests. Thank you Chelsea Mendenhall, Tina Yankovich, Brittney Reyes and Nathan Snizaki for all your support. You all made our life here much easier with such amazing support over the years.

One of the hardest things to do early on in a PhD, is picking a research direction that you find most interesting. No other work has been more influential in this regard than the work of Prof. Tomaso Poggio at MIT. It was a cold email back in 2013 to which he graciously replied which started a long term commitment to the principles of research which his work has embodied. He was very kind in allowing me to attend the first CBMM Summer School back in 2014 which was an intense crash course in systems, computational and cognitive neuroscience. My project for that summer course on modelling random connections was the foundation upon which a part of this thesis, PRCNs are based.

My time in Pittsburgh would have no way near as fun and full-filled had it not been for the wonderful set of friends and support system that I was fortunate to have. I want to thank the members of the 'The Peace Band', 'The Club', 'Solway' and 'Shady'. In particular, I want to thank Annesha Ganguly, Purvasha Chakravarti, Abhishek Ravi, Aditya Menon, Vishal Pallikandi, Vaishnavi Kumar, Sanika Kokate, Arnab Debnath, Vaishanavh Nagarajan, Saurabh Kadekodi, Siddharth Singh, Ajay Pisat, Satwik Kottur, Deepanjana Gupta and Srujana Rao for the wonderful memories.

I also want to include my sister Abhishikta Pal in this list who, as fate would have it, graduated from CMU with a Masters in Urban Design. She has been just an immense source of love and support for over the years. Seldom are people blessed to have such talented and kind hearted siblings in life. One other person who has been an absolute pillar in this journey is Annesha Ganguly. Her love, support, banter, cheerfulness, vocal and culinary talents, along with devastating skill at one particular board game has given life the color it deserves. I am eternally fortunate to have them in my life.

Lastly, there are not many words to describe how privileged I am to have Sudip and Preeti Pal as my parents. They are my original source of inspiration in science. Every point in life was filled with their encouragement, love and support. Their tireless efforts to have me educated, be a good citizen and contribute to the world in anyway I can is the reason I am here today. This thesis is indeed dedicated to them.

This work has been partly funded by US Naval Air Systems Command (NAVAIR) grant no. N68335-16-C-0177, NAVMAR Applied Sciences Corporation grant no. NASC-0040-CMU and National Institute of Justice grant no. 2013-IJ-CX-K005.

Dipan K. Pal

Carnegie Mellon University

December 2020

Abstract

The world is complex, ever changing yet structured. Given this complexity, how can an organism living in such a world or even an artificial system distill this information to perceive what is important towards its intention and what is not? What theoretical principles underline such an ability? More interestingly, how do these principles operate in the chaotic randomness and complexities of neural circuits in mammalian brains? These questions form the very heart of the study of representation learning, and also point to perhaps the most interesting directions the field must explore.

Indeed, one of the fundamental pursuits of machine learning and artificial intelligence is learning to be invariant to nuisance transformations in the data. Most prior work has focused on addressing these challenges through different aspects of the deep learning pipeline such as loss functions, data augmentation and more recently self-supervision techniques. However, the core architecture or structure of these networks has yet to be adapted to these challenges. In this thesis, we explore how to learn and encode invariance towards nuisance transformations by redesigning the convolution architecture itself leading to more powerful and efficient neural networks. We present two fundamental improvements to neural architecture design through NPTNs (Non-Parametric Transformation Networks) and PRCNs (Permanent Random Connectome Networks). These are designed to be drop-in replacements for the ubiquitous vanilla convolution layer.

NPTNs are a natural generalization of ConvNets and unlike almost all previous works in deep architectures, they make no assumption regarding the structure of the invariances present in the data. PRCNs on the other hand, are initialized with random connectomes (not just weights) which are a small subset of the connections in a fully connected convolution layer. Importantly, these connections in PRCNs once initialized remain permanent throughout training and testing. Permanent random connectomes make these architectures loosely more biologically plausible than many other mainstream network architectures which require highly ordered structures. They also offer insights towards computational models of random connectomes in the visual cortex. Empirically, we find that these randomly initialized permanent connections have positive effects on generalization and parameter efficiency. These ideas open a new dimension in deep network design providing more versatile and effective learning. More importantly, they offer initial answers to some of the fundamental and motivating questions we highlighted above in representation learning.

Contents

List of Figures

List of Tables

Chapter 1

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 [2], minimize auxiliary objectives promoting invariance during training as in [3] or pool over transformed versions of the representation itself as in [4, 5, 6].

1.1 Convolutional Networks and Beyond.

Towards this goal, ideas proposed in [7] 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. [8, 9] 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 [10, 11, 12]). Recently however, [13] 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 thesis for modelling *general invariances in the same framework as ConvNets*, open up a new and potentially key dimension for architecture development. In this thesis, 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). Finally, we build upon NPTNs to introduce PRCNs or Permanent Random Connectome Networks.

1.2 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 [14, 15].

Incorporating known invariances using deep networks. Convolutional architectures have seen many efforts to produce rotation invariant representations. [16] and [17] rotate the input itself before feeding it into stacks of CNNs and generating rotation invariant representations through gradual pooling or parameter sharing. [18, 19, 20] 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 [21]. An interesting direction of research was explored by [22] where the rotation, scale and translation invariant filters were fixed and non-trainable. [23, 24] presented methods to incorporate parametric invariances using groups and warped convolutions. The transformations in [24, 25] 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. [26] 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 [4, 5, 6].

Learning unknown invariances from data through networks. Very few works have explored incorporating unknown invariances into deep networks. To the best of our knowledge, SymNets introduced in [1] 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. [27] provide sufficient conditions to enforce the learned representation to have symmetries learned from data. [28] modelled local invariances using pooling over sparse coefficients of a dictionary of basis functions.

[29] 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 [30].

Prior Art Learning Invariances from Data or using Random Connectomes. There have been many seminal works that have indeed explored the role of *temporary* random connections in deep networks such as DropOut [31], DropConnect [32] and Stochastic Pooling [33]. However, unlike the proposed approach, the connections in these networks randomly change at *every* forward pass, hence are temporary. More recently, random permanent connections were explored for large scale architectures [34]. It is important however to note that the basic unit of computation, the convolutional layer, remained unchanged. Our study explores permanent random connectomes *within* the convolutional layer itself, and explores how it can learn non-parametric invariances to multiple transformations simultaneously in a simple manner.

Chapter 2

The Transformation Network Paradigm

2.1 The Transformation Network

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 [35] which pools over all input channels). Fig. 2.1

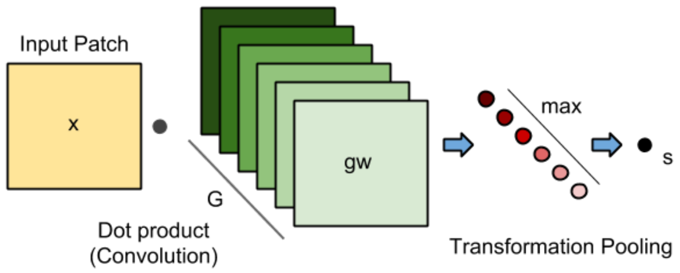


Figure 2.1: A single NPTN node: (a) Operation performed by a single Transformation Network (TN) node (single channel input and single channel output, Non-Parametric Transformation Networks are a kind of TN). TNs (and NPTNs) are a generalization of ConvNets towards learning general invariances and symmetries. 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.

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.

$$\Upsilon(x) = \max_{g \in G}(\langle x, gw \rangle) \tag{2.1}$$

Here, $\langle \quad \rangle$ 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[1]. 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.

2.2 Modeling Transformations as Unitary Groups.

The use of unitary groups to model transformations and invariances has emerged as a prominent theoretical tool [26, 6]. 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

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

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.

2.3 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 [26]). We demonstrate this in the form of the following simple result[2].

Lemma 2.3.1. *(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 $\Upsilon(x) = \Upsilon(g'x)$.*

Proof. Consider the distribution of elements of the set $S_{g'} = \{\langle g'x, gw \rangle\}$ over all $g \in G$ and for any particular $g' \in G$. This 1-D distribution characterizes the vector $g'x$ through the projections onto gw. Due to unitarity of G, and that $g' \in G$, we have $\langle g'x, gw \rangle = \langle x, g'^{-1}gw \rangle$. Now, since G is a group, we have for any $g' \in G$, $g'^{-1}g \in G$ due to the closure property. The set of elements in $S_{g'}$ contains all elements of G and hence must also contain $g'^{-1}g$. This implies that the action of g'^{-1} on the group G results in just a reordering of the group, leaving the distribution unchanged. Thus, the set $S_{g'}$ is unchanged. More specifically, $S_{g'} = \{\langle g'x, gw \rangle\} = \{\langle x, gw \rangle\} = S_e$, where e is the identity element of G. Thus, the two sets invoke the exact same

[2]Proof in the supplementary.

distribution, which results in their moments being the same. This includes the infinite moment, which implies $\Upsilon(g'x) = \max_{g \in G} S_{g'} = \max_{g \in G} S_e = \Upsilon(x)$. $\square$

Lemma 2.3.1 shows that for *any* input x (including test inputs), the node output is *invariant* to the transformation 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 [26]. Invariance is invoked for any arbitrary input x during test time, thereby demonstrating good generalization properties.

2.4 Relaxing towards Non-group and Non-Unitary Structure in a TN node (Towards NPTNs).

Lemma 2.3.1 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 [26, 27, 5, 6, 4]. 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 [20, 19]. 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 chapter, 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 2.3.1 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 thesis. 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.

Chapter 3

Non-Parametric Transformation Networks

3.1 Generalizing Convolution Architectures.

In this chapter, we explore one 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 implemented 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 [18, 19, 20], 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 convolu-

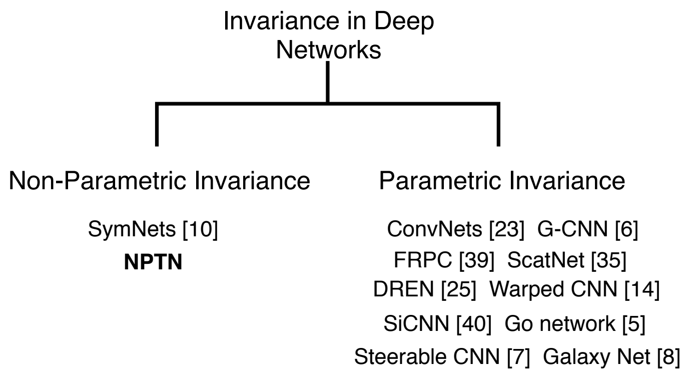

Invariance in Deep Networks
Non-Parametric Invariance
SymNets [10]
NPTN
Parametric Invariance
ConvNets [23] G-CNN [6]
FRPC [39] ScatNet [35]
DREN [25] Warped CNN [14]
SiCNN [40] Go network [5]
Steerable CNN [7] Galaxy Net [8]

Figure 3.1: Types of Invariances in Deep Networks (b) Most previous works in deep learning have focused on invariances to transformations that are parametric in nature and fixed. SymNets ([1]) and NPTNs to the best of our knowledge are the only architectures to *learn* invariances from data towards transformations that are not modelled by any expression in the network itself, *i.e.* the symmetries that are captured are non-parametric in nature.

tional 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. 2.1). 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. 3.1). 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.

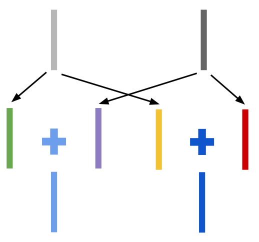
3.2 The NPTN

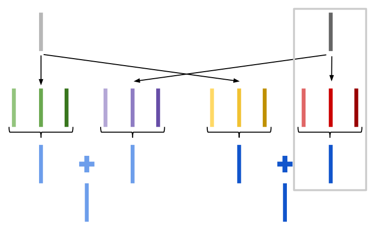
A Non-Parametric Transformation Network (NPTN) is a kind of TN that lacks any constraints on set of weights/filters w for any particular node. Here the set of filters G has two relaxations 1) need not have any group structure and 2) need not model any parametric and/or unitary transformations such as 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 [4, 5].

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 [18, 19]. 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. 3.2(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 [1], 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.

3.2.1 NPTN Layer Structure, Forward Pass and Training.

Fig. 3.2 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. 2.1. There are $|G|$ filters learned for each of the MN nodes, which each are convolved over the image similar


(a) Convolution layer


(b) NPTN layer

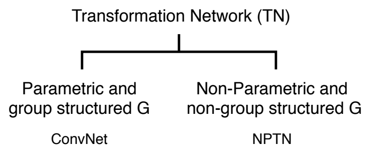

Transformation Network (TN)
Parametric and group structured G
Non-Parametric and non-group structured G
ConvNet
NPTN
(c) Relation between ConvNets and NPTNs

Figure 3.2: 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. 2.1) 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.

to a vanilla ConvNet. Consider Fig. 3.2(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 *output* feature map or channel. This is repeated for each of the N output channels[1]. 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 [16, 17, 18, 19, 20]. In fact, the computation complexity for NPTNs only increases with the order $|G|$ relative

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

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[2]. 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.

3.2.2 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. 3.1). 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

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

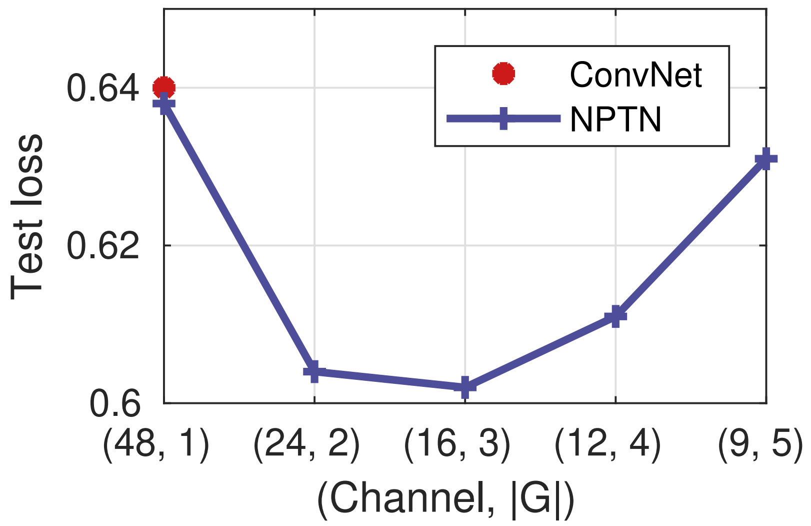


Figure 3.3: Test losses on CIFAR-10 for the two layered network. Each network listed has the same number of filters.

kinds of invariances. This is again in sharp contrast to ConvNets and other previous works such as [18, 19, 20] 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.

3.3 Empirical Evaluation of NPTNs

3.3.1 Benchmarking against ConvNets on CIFAR-10

In our first set of experiments, we benchmark and characterize the behavior of NPTNs against the standard ConvNets augmented with Batch Normalization [36]. The goal of this set of experiments is to observe whether learning non-parametric transformation invariance from complex visual data itself helps with object classification. For this experiment, we utilize the CIFAR-10 dataset[3]. 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. 3.3 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.

3.3.2 Benchmarking against other approaches: ETH-80

We now benchmark NPTNs against other approaches learning invariances on the ETH-80 dataset [37]. As our baseline, we follow the experimental setup and the specifications of the models described in [38]. 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

[3]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.

Method	Accuracy (%)
ConvNet	80.1
STN	45.1
DeepScat	87.3
HarmNet	94.0
TIGradNet	95.1
NPTN (Ours)	**96.2**

Table 3.1: Test accuracy on ETH-80. All models including NPTNs and the ConvNet had roughly the same number of parameters (about 1.4M). Results for models along with architecture details other than NPTN are cited as is from the same study as ConvNet.

to 8 classes. Each object has 41 images taken from a grid of different viewpoints on a hemisphere. Following [38], 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 [39], DeepScat [40], HarmNet [41] and TIGradNet [38]. The NPTN architecture was chosen to by replacing the convolution layers in the ConvNet architecture in [38] 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. We utilized the model architectures and results from ConvNet [38], STN [39], DeepScat [40], HarmNet [41], TIGradNet [38]. Table 3.1 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.

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 3.2: 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.

3.3.3 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 3.2 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)[4]. However, as the 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

[4]NPTNs perform slightly better than ConvNets for 0° rotations because for all rotation experiments, small translations up to 2 pixels were applied only in training.

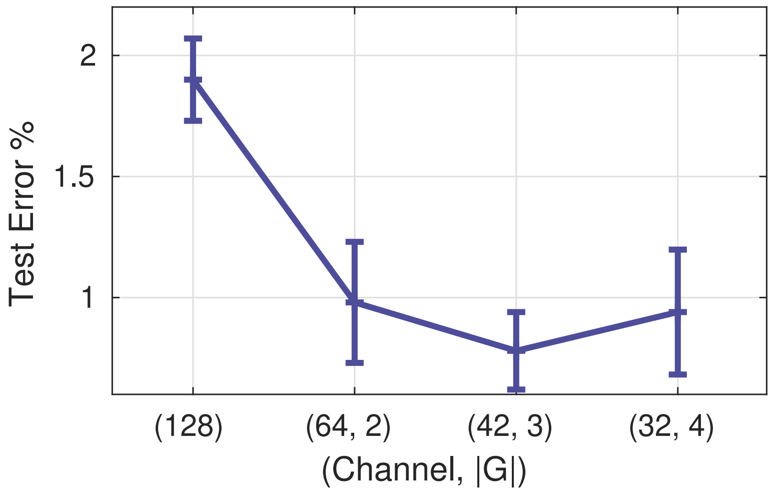


Figure 3.4: Test errors on MNIST for Capsule Nets augmented with NPTNs. (128) denotes a Capsule Network with a vanilla ConvNet. Other labels are NPTNs with (channels, $|G|$). The number of filters from left to right is $\{4224, 4160, 4074, 4128\}$. NPTNs significantly outperform ConvNets in Capsule Nets with fewer filters.

($|G| = 3$). 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.

3.3.4 NPTNs with Capsule Networks

Capsule Networks with dynamic routing were recently introduced as an extension of standard neural networks [13]. 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[5]. 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. 3.4. 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.

3.4 Discussion on NPTNs

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 CIFAR-10 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

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

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.

Chapter 4

Permanent Random Connectome Networks

4.1 Motivating Permanent Random Connectome Networks

4.1.1 The Problem of Invoking Invariances.

Learning invariances to nuisance transformations in data has emerged to be a core problem in machine learning [26, 3, 1, 39, 23]. Moving towards real-world data of different modalities, it is a daunting task to theoretically model all nuisance transformations. Towards this goal, methods which *learn* non-parametric invariances from the data itself without any change in architecture will be critical. However, before delving into methods which learn such invariances however, it is important to study methods which incorporate *known* invariances in data. An early method to incorporate the translation prior was the Convolutional Neural Network (ConvNet) [7]. Over the years, there have been efforts in investigating what other trans-

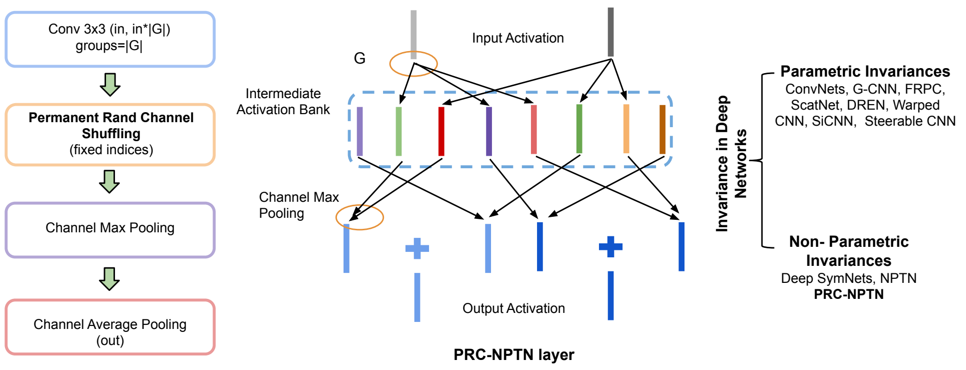


Figure 4.1: **Left:** Operations comprising the PRC-NPTN layer. The number of input and output channels in the Conv layer is (inch) and (inch$*|G|$) respectively. G is the number of filters (linear transformations learnt) for each input channel. The key operation proposed is the Permanent Random Channel shuffling operation with a fixed index mapping for every forward pass. This indexing or connectome is initialized randomly during network initialization. **Center:** Architecture of the PRC-NPTN layer. Each input channel is convolved with a number of filters (parameterized by G). Each of the resultant activation maps is connected to a one of the channel max pooling units randomly (initialized once, fixed during training and testing). Each channel pooling unit pools over a fixed random support of a size parameterized by CMP. **Right:** Explicit invariances enforced within deep networks in prior art are mostly parametric in nature. The important problem of learning *non-parametric* invariances from data has not received a lot of attention.

formations result in useful hand-crafted priors in data such as rotation and scale [17, 18, 20, 42, 23, 24, 25]. It is important to note however that these methods ultimately were limited to hand-crafted invariances assumed to be useful for the task at hand.

Motivation of this Study: In this study, we motivate and investigate one possible architecture that can learn invariances towards multiple transformations from data itself. At the heart of the architecture is the structure called the *permanent random* connectome. This simply refers to a channel shuffling layer that uses a fixed shuffling schedule throughout the life of the network (including training and testing) resulting in a *permanent* connectome. Importantly however, this shuffling indexing is chosen at random at the initialization of the network. Thereby leading to the layer being referred to as the permanent *random* connectome. We find that layers utilizing the permanent random operation allow architectures to learn multiple invariances efficiently from data itself. Our motivation also loosely stems from observations regarding connectomes in the cortex.

Encoding Invariances through Deep Architectures. Before delving into methods which learn such invariances, it is important to study methods which incorporate *known* invariances in data. Many times it is the case that a few most predominant nuisance transformations in data are well understood. Visual data is one such domain with translation being perhaps the most common nuisance transformation emerging. An early method to incorporate this prior into the algorithm was the Convolutional Neural Network (ConvNet) [7] with the pooling operation following the convolution. The success of ConvNets indicates that addressing predominant invariances in data warrants being a major objective. Over the years, there have been efforts in investigating what other transformations would result in similar breakthroughs. Rotation was investigated at length with studies rotating the inputs [17] and the convolution filters [18, 20]. Similarly combinations of rotation, scale and translation invariances were explored [42] along with more general parametric invariances [23, 24, 25]. These efforts provided valuable insights into the nature of visual data leading to more powerful networks, albeit for specific or specialized tasks. For more general tasks, methods which focused on better optimization, minimizing better objectives and developing more effective architectures proved to be more successful. Nonetheless, it is important to note that though these methods were motivated differently, they ultimately provided hand-crafted invariances assumed to be useful for the task at hand.

Incorporating known invariances using deep networks. Convolutional architectures have seen many efforts to produce rotation invariant representations. [16] and [17] rotate the input itself before feeding it into stacks of CNNs and generating rotation invariant representations through gradual pooling or parameter sharing. [18, 19, 20] 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 [21]. An interesting direction of research was explored by [22] where the rotation, scale and translation invariant filters were fixed and non-trainable. [23, 24] presented methods to incorporate parametric invariances using groups and warped convolutions. The transformations in [24, 25] 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.

4.1.2 Relaxed Biological Motivation for Randomly Initialized Connectomes.

Although not central to our motivation, the observation that the cortex lacks precise *local* pathways for back-propagation provided the initial inspiration for this study. It further garnered pull from the observation that random unstructured local connections are indeed common in many parts of the cortex [43, 44]. Moreover, it has been shown that orientation selectivity can arise in the visual cortex even through local random connections [45]. Though we do not explore these biological connections in more detail, it is still an interesting observation. There has also been some interesting work which explored the use of random weight matrices for back propagation [46]. Here, the forward weight matrices were updated so as to fruitfully use the random weight matrices during back propagation. The motivation of the [46] study was to address the biological implausibility of the transport of precise gradients through the cortex due to the lack of exact connections and pathways [47, 48, 49, 50]. The common presence of random connections in the cortex at a *local* level leads us to ask: Is it possible that such locally random connectomes improve generalization in deep networks? We provide evidence for answering this question in the positive.

Contributions. 1) We motivate permanent random connectomes from the perspective of learning invariance to multiple transformations directly from data. The fundamental problem of learning non -parametric invariances in perception has not received enough attention. We present an architectural prior capable of such a task with loose biological motivation. 2) We present a theoretical result on learning invariances to transformations which do not obey a group structure in contrast to prior work. 3) We provide results on learning invariances to individual and multiple transformations in data without any change in architecture whatsoever. Further, we demonstrate improvements in generalization while using PRC-NPTN as a drop in replacement to conv layers in DenseNets. 4) Finally, as an engineering effort, we develop fast and efficient CUDA kernels for random channel pooling which result in efficient implementations of PRC-NPTNs in terms of computational speed and memory requirements compared to traditional Pytorch code.

4.2 Permanent Random Connectome NPTNs

We begin by motivating permanent random connectomes from the perspective of selecting the support for pooling. We find that permanent random channel pooling invokes invariance to multiple transformations simultaneously. Investigating idea of pooling across transformations to invoke invariance, permanent random pooling emerges naturally. As part of our contribution, we present a theoretical result which confirms a long standing intuition that max pooling invokes invariance.

4.2.1 Invoking Invariance through Max Pooling.

In previous years a number of theories have emerged on the mechanics of generating invariance through pooling. [26, 27] develop a framework in which the transformations are modelled as a group comprised of unitary operators denoted by $\{g \in \mathcal{G}\}$. These operators transform a given filter w through the operation gw[1], following which the dot-product between these transformed filters and an novel input x is measured through $\langle x, gw \rangle$. It was shown by [26] that any moment such as the mean or max (infinite moment) of the distribution of these dot-products in the set $\{\langle x, gw \rangle | g \in \mathcal{G}\}$ is an invariant. These invariants will exhibit robustness to the transformation in $\mathcal{G}$ encoded by the transformed filters in practice, as confirmed by [4, 5]. Though this framework did not make any assumptions on the distribution of the dot-products, it imposed the restricting assumption of group symmetry on the transformations. We now show that invariance can be invoked even when *avoiding the assumption that the transformations in $\mathcal{G}$ need to form a group.* Nonetheless, we assume that the distribution of the dot-product $\langle x, gw \rangle$ is uniform and thus we have the following result[2].

Lemma 4.2.1. *(Invariance Property) Assume a novel test input x and a filter w both fixed vectors $\in \mathbb{R}^d$. Further, let g denote a random variable representing unitary*

[1]The action of the group element g on w is denoted by gw to promote clarity.

[2]We thank Purvasha Chakravarti for the proof. The assumption of the distribution being uniform is meant to provide insight into the general behavior of the max pooling operation, rather than a statement that deep learning features are uniformly distributed.

operators with some distribution. Finally, let $\Upsilon(x) = \langle x, gw\rangle$, with $\Upsilon(x) \sim U(a,b)$ i.e. *a Uniform distribution between a and b. Then, we have*

$$\mathrm{Var}(\max \Upsilon(x)) \leq \mathrm{Var}(\Upsilon(x)) = \mathrm{Var}(\langle g^{-1}x, w\rangle)$$

Proof. Let X be the random variable representing the randomness in $\langle x, gw\rangle$ for fixed x, w and random g. We assume that $X \sim U(0,1)$.

Considering a sample set $X_1, X_2 ... X_n \sim U(0,1)$, then $X_{(n)} = \max_{1\leq i\leq n X_n}$. Now,

$$\begin{aligned} P(X_{(n)} \leq x) &= P(X_i \leq x, \forall i) && (4.1)\\ &= P(X_i \leq x)^n && (4.2)\\ &= x^n && (4.3)\end{aligned}$$

Let the density of $X_{(n)}$ be denoted by $f_{X_{(n)}}(x)$, then

$$f_{X_{(n)}}(x) = \begin{cases} 0 & x \leq 0 \\ nx^{n-1} & 0 \leq x \leq 1 \\ 1 & x \geq 1 \end{cases}$$

Now,

$$\mathbb{E}[X_{(n)}] = \int_0^1 xnx^{n-1}dx = \frac{x^{n+1}}{n+1}n|_0^1 = \frac{n}{n+1}$$

$$\mathbb{E}[X_{(n)}^2] = \int_0^1 x^2nx^{n-1}dx = \frac{x^{n+2}}{n+2}n|_0^1 = \frac{n}{n+2}$$

Therefore,

$$\mathrm{Var}(X_{(n)}) = \frac{n}{n+1} - \left(\frac{n}{n+1}\right)^2 = \frac{n}{(n+2)(n+1)^2}$$

Since the variance of $U(0,1)$ is $\frac{1}{12}$ *i.e.* $\mathrm{Var(X_i)} = \frac{1}{12}$, and $\mathrm{Var}(X_{(n)})$ is a decreasing function in n, along with the fact that $\text{Var}(X_{(n)})$ for $n = 1$ is $\frac{1}{12}$, we have

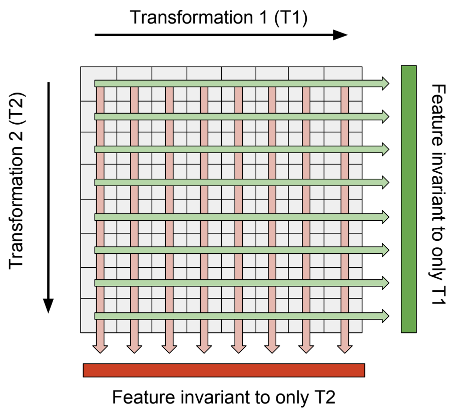
$$\text{Var}(X_{(n)}) \leq \text{Var}(X_i)$$

For general $U(a, b)$, it follows shortly after considering $Y_i = \frac{X_i - a}{b-a}$ and that $\text{Var}(Y_i) = \text{Var}(X_i)$. Finally, due to unitary g, $\langle x, gw\rangle = \langle g^{-1}x, w\rangle$. $\square$

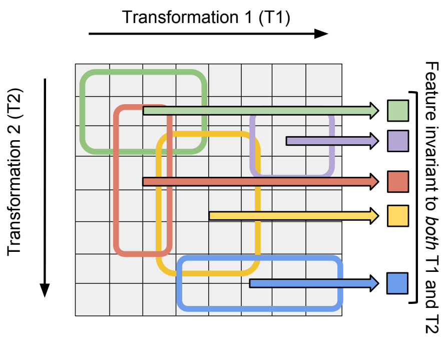
This result is interesting because it shows that the max operation of the dot-products has less variance due to g than the pre-pooled features. Though this is largely known empirical result, a concrete proof for invoking invariance was so far missing. More importantly, it bypasses the need for a group structure on the nuisance transformations $\mathcal{G}$. Practical studies such as [4, 5] had ignored the effects of non-group structure in theory while demonstrating effective empirical results. Also note that the variance of the max is less than the variance of the quantity $\langle g^{-1}x, w\rangle$, which implies that $\max \Upsilon(x)$ is more robust to g even in test, though it has never observed gx. This useful property is due to the unitarity of g.

4.2.2 Connection to Deep Networks.

PRC-NPTN as we will show, perform max pooling across *channels* not space, to invoke invariance. In the framework $\langle x, gw\rangle$, w would be one convolution filter with g being the transformed version of it. Note that this modelling is done only to satisfy a theoretical construction, we do not actually transform filters in practice. All transformed filters are learnt through backpropagation. This framework is already utilized in ConvNets. For instance, ConvNets [7] pool only across translations (convolution operation itself followed by *spatial* max pooling implies g to be translation).

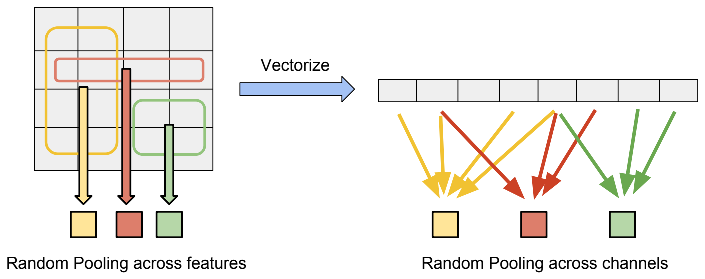


(a) Homogeneous Structured Pooling



(b) Permanent Random Support Pooling

Figure 4.2: **(a) Homogeneous Structured Pooling** pools across the entire range of transformations of the *same kind* leading to feature vectors invariant only to that particular transformations. Here, two *distinct* feature vectors are invariant to transformation T_1 and T_2 independently. **(b) Heterogeneous Random Support Pooling** pools across randomly selected ranges of multiple transformations *simultaneously*. The pooling supports are defined during initialization and remain fixed during training and testing. This results in a *single* feature vector that is invariant to multiple transformations simultaneously. Here, each colored box defines the support of the pooling and pools across features only inside the boxed region leading to one single feature.


Vectorize
Random Pooling across features
Random Pooling across channels

(a) PRC-NPTN Random Channel Pooling

Figure 4.3: **Vectorized Random Support Pooling** extends this idea to convolutional networks, where one realizes that the random support pooling on the feature grid (on the left) is equivalent to random support pooling of the vectorized grid. Each element of the vector (on the right) now represents a single *channel* in a convolutional network and hence random support pooling in PRC-NPTNs occur across channels.

4.2.3 Invoking Invariance through Channel Pooling in Deep Networks.

Consider a grid of features that have been obtained through a dot product $\langle x, gw \rangle$ (for instance from a convolution activation map, where the grid is simply populated with each $k \times k \times 1$ filter, not $k \times k \times c$) (see Fig. 4.2(a)). Assume that along the two axes of the grid, two different kinds of transformation are acted. T_1 along the horizontal axis and T_2 along the vertical. $T_1 = g_1(\cdot; \theta_1)$ where g_1 is a transformation parameterized by θ_1 that acts on w and similarly $T_2 = g_2(\cdot; \theta_2)$. Now, pooling homogeneously across one axis invokes invariance *only* to the corresponding g (for a more in depth analysis see [26]). Similarly, pooling along T_2 only will result in a feature vector (Feature 2) invariant *only* to T_2. These representations (Feature 1 and 2) have complimentary invariances and can be used for complimentary tasks *e.g.* face recognition (invariant to pose) versus pose estimation (invariant to subject). This approach has one major limitation that tihs scales linearly with the number of transformations which is impractical. One therefore would need features that are invariant to multiple transformations simultaneously. A simple yet effective approach is to pool along all axes thereby being invariant to all transformations simultaneously. However, doing so will result in a degenerative feature (that is invariant to everything and discriminative to nothing). Therefore, the key is to limit the *range* of pooling performed for each transformation.

4.2.4 Choosing the Support for Pooling at Random: Permanent Random Connectomes.

A solution to trivial feature problem described above, is to limit the *range* or support of pooling as illustrated in Fig.4.2(b). One simple way of selecting such a support for pooling is at random. This selection would happen only once during initialization of the network (or any other model), and will remain fixed through training and testing. In order to increase the selectivity of such features, multiple such pooling units are needed with such a randomly initialized support [26, 6]. These multiple pooling units together form the feature that is invariant to multiple transformations simultaneously, which improves generalization as we find in our experiments. This is called heterogeneous pooling and Fig. 4.2(b) illustrates this more concretely. We therefore find that permanent random pooling is motivated naturally through the need to attain invariance to multiple transformations simultaneously.

4.2.5 The PRC-NPTN layer.

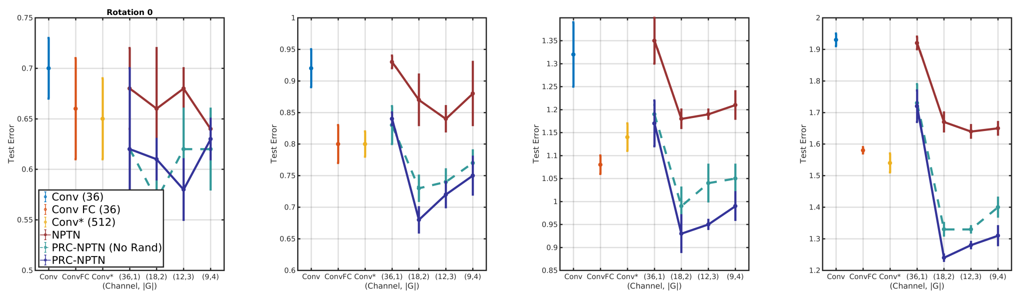
Fig. 4.1 shows the the architecture of a single PRC-NPTN layer . The PRC-NPTN layer consists of a set of $N_{in} \times G$ filters of size $k \times k$ where N_{in} is the number of input channels and G is the number of filters connected to each input channel. More specifically, each of the N_{in} input channels connects to $|G|$ filters. Then, a number of channel max pooling units randomly select a fixed number of activation maps to pool over. This is parameterized by Channel Max Pool (CMP). Note that this random support selection for pooling is the reason a PRC-NPTN layer contains a permanent random connectome. These pooling supports once initialized do not change through training or testing. Once max pooling over CMP activation maps completes, the resultant tensor is average pooled across channels with a average pool size such that the desired number of outputs is obtained. After the CMP units, the output is finally fed through a two layered network with the same number of channels with 1×1 kernels, which we call a pooling network. This small pooling network helps in selecting non-linear combinations of the invariant nodes generated through the CMP operation, thereby enriching feature combinations downstream. For experimental rigor, we also benchmark against the baseline ConvNet supplemented with this 1x1 pooling network.

Invariances in a PRC-NPTN layer. Recent work introducing NPTNs [51] had highlighted the Transformation Network (TN) framework in which invariance is generated during the forward pass by pooling over dot-products with transformed filter outputs. A vanilla convolution layer with a single input and output channel (therefore a single convolution filter) followed by a $k \times k$ *spatial* pooling layer can be seen as a single TN node enforcing translation invariance with the number of filter outputs being pooled over to be $k \times k$. It has been shown that $k \times k$ spatial pooling over the convolution output of a single filter is an approximation to channel pooling across the outputs of $k \times k$ translated filters [51]. The output $\Upsilon(x)$ of such an operation with an input patch x can be expressed as

$$\Upsilon(x) = \max_{g \in \mathcal{G}} \langle x, gw \rangle \tag{4.4}$$

where $\mathcal{G}$ is the set of filters whose outputs are being pooled over. Thus, $\mathcal{G}$ defines the set of transformations and thus the invariance that the TN node enforces. In a vanilla convolution layer, this is the translation group (enforced by the convolution operation followed by *spatial* pooling). An NPTN removes any constraints on $\mathcal{G}$ allowing it to approximately model arbitrarily complex transformations. A vanilla convolution layer would have *one* filter whose convolution is pooled over spatially (for translation invariance). In contrast, an NPTN node has $|\mathcal{G}|$ *independent* filters whose convolution outputs are pooled across *channel* wise leading to general invariance.

A PRC-NPTN layer inherits the property from NPTNs to learn arbitrary transformations and thereby arbitrary invariances using $\mathcal{G}$. As Fig. 4.1(b) shows, individual channel max pooling (CMP) nodes act as NPTN nodes sharing a *common* filter bank as opposed to independent and disjoint filter banks for vanilla NPTNs. This allows for greater activation sharing, where transformations learned from data through one subset of filters can be used for invoking similar invariances in a parallel computation path. This sharing and reuse of activation maps allows for higher parameter and sample efficiency. As we find in our experiments, randomization plays a critical role here, allowing for a simple and quick approximation to obtaining high performing invariances. A high activation map can activate multiple CMP nodes, winning over multiple sub-sets of low activations. Gradients flow back to these winning activations updating the filters to further model the features observed during that particular batch. Note that, CMP nodes in the same layer can pool over disjoint subsets to invoke a variety of invariances, leading to a more versatile network and also better modelling of a particular kind of invariance as we find in our experiments. Further, the primary source of invoking invariances in NPTN was understood to be the symmetry of the unitary group action space [51]. General invariances were assumed to be only approximately forming a group. Lemma 4.2.1 shows that group symmetry is not necessary to reduce variance of the quantity $\max \Upsilon(x)$ due to the action of the set elements g on some test input patch x. Though, the result makes a strong assumption regarding the distribution of $\Upsilon(x)$, it to the best of our knowledge the first result of its kind to show increased invariance without a group symmetric action.



(a) Rotation 0° (b) Rotation 30° (c) Rotation 60° (d) Rotation 90°

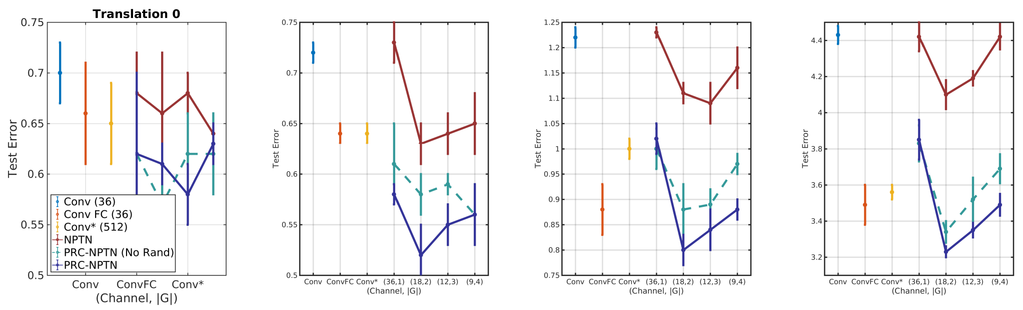
Figure 4.4: **Only Rotation Transformation Results:** Test error statistics with mean and standard deviation on MNIST with progressively extreme **random rotations**. ConvNet FC denotes the addition of a 2-layered pooling 1×1 pooling network after every layer. Note that for this experiment, CMP=$|G|$. Permanent Random Connectomes help with achieving better generalization despite increased nuisance transformations.

4.3 Empirical Evaluation and Discussion

Goal. The goal of our evaluation study is to demonstrate PRC-NPTNs as capable of learning transformations from data and to showcase improvements in generalization in supervised classification over relevant baselines. The goal is not in fact, to compete with the state of the art approaches for any dataset.

General Experimental Settings. For all experiments, we run all models for 300 epochs trained using SGD. The initial learning rate was kept at 0.1 and decreased by 10 at 50% and 75% epoch completion. Momentum was kept at 0.9 with a weight decay of 10^{-5}. Batch size was kept at 64 for both MNIST and ETH-80 [3]. For the MNIST experiments, gradients were clipped to norm 1. Each block for all baselines for ConvNet and PRC-NPTN had either a convolution layer or PRC-NPTN layer followed by batch normalization, PReLU and spatial max pooling. The convolutional kernel size for all models was kept at 5×5 for all MNIST experiments

[3]We provide experiments on CIFAR-10 in the supplementary.

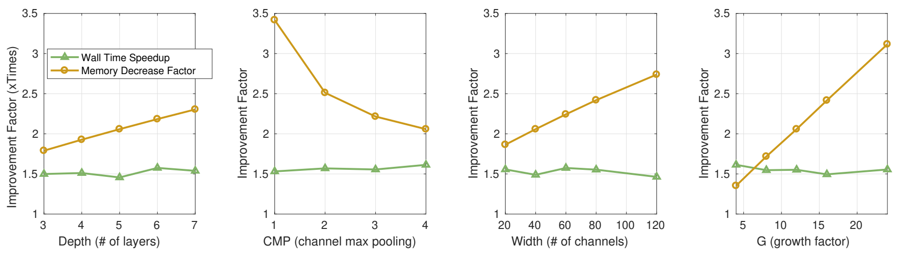


(a) Translation 0 pixels (b) Translation 4 pixels (c) Translation 8 pixels (d) Translation 12 pixels

Figure 4.5: **Only Translation Transformation Results:** Test error statistics with mean and standard deviation on MNIST with progressively extreme **random pixel translations**. ConvNet FC denotes the addition of a 2-layered pooling 1×1 pooling network after every layer. Note that for this experiment, CMP=$|G|$. Permanent Random Connectomes help with achieving better generalization despite increased nuisance transformations.

and 3×3 for all other models. Spatial max pooling of size 3×3 was performed after every layer, BN and PReLU for MNIST models.

Limitations in Typical Implementations and Developing Faster Kernels. Our implementation with traditional PyTorch still suffered from heavy GPU memory use and slower run times despite optimizing code at the PyTorch abstraction level. The key bottleneck in computational and memory efficiency was found to be the randomized channel pooling operation. The issue was addressed by developing CUDA kernels that performed pooling on non-contiguous blocks of memory without creating copies of the same. This allowed for faster non-contiguous pooling over feature and activation maps with a significant reduction in memory usage. The operation was built as a CUDA-kernel that is interfaced with PyTorch through CuPy. This engineering effort is part of our contribution and we demonstrate improvements in memory and computational efficiency in our experiments (see Fig. 4.6). We observe a consistent speed up of atleast 1.5x and a significant reduction in memory



(a) Depth (b) CMP (c) Width (d) Growth Factor

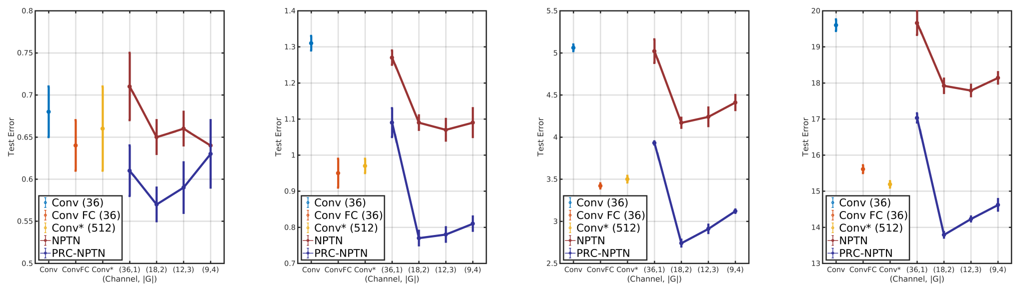
Figure 4.6: Computational efficiency improvements of our CUDA implementations.

usage.

4.3.1 Efficacy in Learning Arbitrary and Unknown Transformations Invariances from Data.

We evaluate on one of the most important tasks of any perception system, *i.e.* being invariant to nuisance transformations *learned* from the data itself. Most other architectures based on vanilla ConvNets learn these invariances through the implicit neural network functional map rather than explicitly through the architecture as PRC-NPTNs. Moreover, most previous approaches needed hand crafted architectures to handle different transformations. We benchmark our networks based on tasks where nuisance transformations such as large amounts of in-plane rotation and translation are steadily increased, with no change in architecture whatsoever. For this purpose, we utilize MNIST where it is straightforward to add such transformations without any artifacts.

Protocol: We benchmark on such a task as described in [51] and for fair comparisons, we follow the exact same protocol. We train *and* test on MNIST augmented with progressively increasing transformations *i.e.* **1)** extreme random translations (up to 12 pixels in a 28 by 28 image), **2)** extreme random rotations (up



(a) (0°, 0 pix) (b) (30°, 4 pix) (c) (60°, 8 pix) (d) (90°, 12 pix)

Figure 4.7: **Simultaneous Transformation Results:** Test error statistics with mean and standard deviation on MNIST with progressively extreme transformations with **random rotations** and **random pixel shifts simultaneously**. For PRC-NPTN and NPTN the brackets indicate the number of channels in the layer 1 and G. Note that for this experiment, CMP=$|G|$.

to 90° rotations) and finally **3)** both transformations simultaneously. *Both* train and test data were augmented randomly for every sample leading to an increase in overall complexity of the problem. No architecture was altered in anyway between the two transformations *i.e.* they were not designed to specifically handle either. The same architecture for all networks is expected to learn invariances directly from data unlike prior art where such invariances are hand crafted in [18, 20, 22, 21, 23, 24].

For this experiment, we utilize a two layered network with the intermediate layer 1 having up to 36 channels and layer 2 having exactly 16 channels for all networks (similar to the architectures in [51]) except a wider ConvNet baseline with 512 channels. All ConvNet, NPTN and PRC-NPTN models have the similar number of parameters (except the ConvNet with 512 channels). For PRC-NPTN, the number of channels in layer 1 was decreased from 36, through to 9 while $|G|$ was increased in order to maintain similar number of parameters. All PRC-NPTN networks have a two layered 1×1 pooling network with same number of channels as that layer. For a fair benchmark, Convnet FC has 2 two-layered pooling networks with 36 channels each. Average test errors are reported over 5 runs for all networks.

Discussion. We present all test errors for this experiment in Fig. 4.4, Fig. 4.5 and Fig. 4.7[4]. From both figures, it is clear that as more nuisance transformations act on the data, PRC-NPTN networks outperform other baselines with the same number of parameters. In fact, even with significantly more parameters, ConvNet-512 performs worse than PRCN-NPTN on this task for all settings. Since the testing data has nuisance transformations similar to the training data, the only way for a model to perform well is to learn invariance to these transformations. It is also interesting to observe that permanent *random* connectomes do indeed help with generalization. Indeed, without randomization the performance of PRCN-NPTNs drop substantially. The performance improvement of PRC-NPTN also increases with nuisance transformations, showcasing the benefits arising from modelling such invariances. This is particularly apparent from Fig. 4.7, where the two simultaneous nuisance transformations pose a significant challenge. Yet, as the transformations increase, the performance improvements increase as well.

4.3.2 Evaluation on the ETH-80 dataset

The ETH-80 dataset was introduced in [37] as a benchmark to test models against 3D pose variation of various objects. The dataset contains 80 objects belonging to 8 different classes. Each object has images from different viewpoints on a hemisphere for a total of 41 images per object. The images were resized to 50×50 following [38]. This dataset is perfectly poised to test how efficiently a model can learn invariance to 3D viewpoint variation.

Protocol: For this experiment, we follow the evaluation protocol as de-

[4]We display only the (12, 3) configuration for NPTN as it performed the best. The translation results and more benchmarks with NPTNs are provided in the supplementary. We obtain similar peforamnce improvements with extreme translation as well.

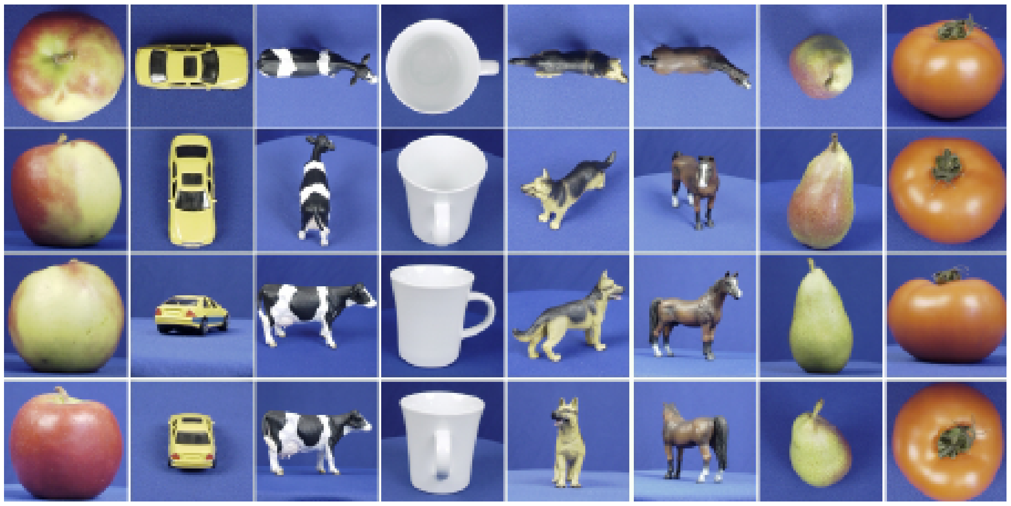


Figure 4.8: Sample images from the ETH-80 database. The dataset contains 80 objects belonging to 8 different classes. Each object has images from different viewpoints on a hemisphere resulting in 3D pose and viewpoint variation for each object.

scribed in [38] [5]. We randomly select 2,300 images to train and test on the rest. For a fair comparison we retrain the ConvNet described in [38]. We design two ConvNet architectures which reflect more modern architecture choices such as a smaller FC layer or having only a global average pooling after a number of conv layers. Table. 4.1 presents the architectures that we train for this experiment. Every conv layer (except the first conv layer *within* a PRC-NPTN layer) is followed by BatchNorm and ReLU. We train corresponding PRC-NPTN models that have fewer parameters. For these experiments with PRC-NPTN, we replace the average pooling across channels (not max pooling or CMP) with a 1×1 convolution layer. We do this to explore the effect of weighted pooling instead of vanilla channel average pooling. To maintain a fair comparison, we compare against equivalent ConvNet baselines with an extra 1×1 added. We also perform an ablation study with the randomization removed. All models were trained with Adam with a learning rate

[5]We also present results on a harder protocol we devised in the supplementary

Architecture	
ConvNet B	C(12) - C(24) - C(48) - C(48) - GAP - FC(300) - FC(200) - FC(8)
NPTN-large B	C(12) - NPTN(24) - NPTN(48) - NPTN(48) - GAP - FC(300) - FC(200) - FC(8)
NPTN-small B	C(12) - NPTN(8) - NPTN(24) - NPTN(48) - GAP - FC(300) - FC(200) - FC(8)
PRC-NPTN B	C(12) - PRC(24) - PRC(48) - PRC(48) - GAP - FC(300) - FC(200) - FC(8)
ConvNet C	C(12) - C(24) - C(48) - C(64) - C(128) - GAP - FC(8)
PRC-NPTN C	C(12) - PRC(24) - PRC(48) - PRC(64) - PRC(128) - GAP - FC(8)

Table 4.1: **Architectures tested on ETH-80.** C - convolution layer, FC - fully connected layer, PRC - PRC-NPTN layer, NPTN - NPTN layer, GAP - global average pooling layer. Every Conv, NPTN and PRC-NPTN layer was followed by a spatial pooling layer of kernel size 2 except the last layer before the GAP. The 1×1 versions of these architectures have a $1\times$ conv layer after every 3×3 layer except the first C(12) layer. ConvNet B was designed to be similar to the architecture explored in Khasanova*et.al.*, 2017 however with more layers. ConvNet C was designed to be more aligned with modern architecture choices such as global average pooling followed by just one FC layer.

of 0.01 for 100 epochs and a batch size of 64. Each architecture was trained 10 separate times with the mean of the runs being reported.

Discussion: We showcase the results in Table. 4.2. We find that of the two different types of architectures that we explore, PRC-NPTNs outperform both corresponding ConvNet architectures. Further, they do so not only with fewer number of parameters, but also fewer number of 3×3 filters. In fact, PRC-NPTN C for $|G| = 8$ and CMP=4 outperforms the corresponding ConvNet C architectures with a $2.97\times$ reduction in the number of parameters and a $10.44\times$ reduction in the number of 3×3 convolution filters. Similarly, PRC-NPTNs outperform two architectures of NPTNs [51] both with significantly fewer parameters and 3×3 filters. These results illustrate that PRC-NPTN can utilize filters and parameters more efficiently on a classification problem which requires 3D pose invariance. This efficiency we conjecture, is due to the fact that permanent random pooling results in an inductive bias that explicitly helps the learning of multiple invariances within the same layer thereby vastly increasing model capacity. Note that the almost 3X reduction in the

Method (Protocol 1)	Accuracy (%)	#params	Factor	#filters	Reduction
ConvNet*	93.69	1.4M		230	-
NPTN*	96.2	1.4M		230	-
ConvNet B	95.61	110K	1×	3780	1×
ConvNet B (1 × 1)	94.54	115K	0.95×	3780	1×
NPTN-large B (3)	94.63	189K	0.58×	11268	0.33×
NPTN-small B (3)	95.09	120K	0.91×	4356	0.86×
PRC-NPTN B (8, 2)	**96.72**	**97K**	**1.13**×	**708**	**5.33**×
ConvNet C	93.90	116K	1×	12740	1×
ConvNet C (1 × 1)	95.98	138K	0.84×	12740	1×
PRC-NPTN C (8, 2)	95.93	64K	1.81×	**1220**	**10.44**×
PRC-NPTN C (8, 4)	**96.40**	**39K**	**2.97**×	**1220**	**10.44**×

Table 4.2: **Test accuracy on ETH-80 Protocol 1.** $*$ indicates the result was obtained from the corresponding paper (Khasanova *et.al.*, 2017 and Pal and Savvides, 2019), and is not on the split used for our experiments. For NPTN is number in the bracket denotes $|G|$, for PRC-NPTN the numbers denote $|G|$ and CMP respectively.

number of parameters and 10X reduction in the number of filters is achieved without the use of any network pruning or post processing methods. Note that competing methods presented in [38] all perform comparably however with 1.4 million parameters each with the highest result being TIGradNet [38] at 95.1, HarmNet at 94.0 [41]. PRC-NPTN outperforms these methods which were designed to invoke invariances through inductive biases while using a fraction of the number of parameters.

4.3.3 Efficacy on CIFAR-10 Image Classification.

MNIST was a good candidate for the previous experiment where the addition of nuisance transformations such as translation and rotation did not introduce any artifacts. However, in order to validate permanent random connectomes on more realistic data, we utilize the CIFAR-10 dataset and AutoAugmentation [52] as the nuisance transformation. Note that, from the perspective of previous works in network invariance, it is unclear how to hand craft architectures to handle invariances due to the variety of transformations that AutoAugment invokes. Here is where the general invariance learning capability of PRC-NPTNs would help, without the need of expertise in such hand-crafting.

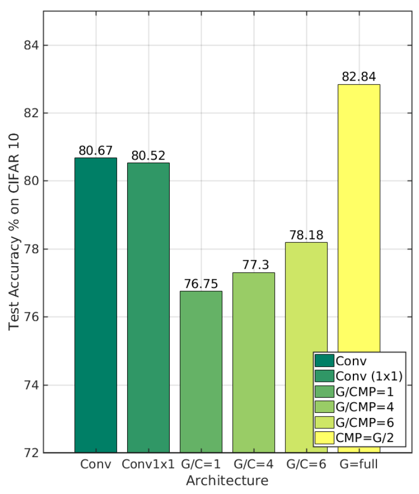
Protocol: We replace vanilla convolution layers with kernel size 3 in DenseNets with PRC-NPTNs without the 2-layered pooling networks. There was another modification for this experiment. For each input channel of a layer, a total of $|G| = 12$ filters were learnt. However only a few of them were pooled over (channel max pool or CMP). We pool with CMP $= 1, 2, 3$ or 4 channels randomly keeping $|G| = 12$ fixed always. Note that in contrast with the MNIST experiment, pooling was always done over $|G|$ number of channels (CMP=$|G|$). This provides a different setting under which PRC-NPTN can be utilized. All models in this experiment were trained with AutoAugment and were tested on both a) the original testing images and also on b) the test set transformed by AutoAugment. Similarly to the previous experiment, a model would have learn invariance towards these auto-augment transformations in order to perform well. All DenseNet models have 12 layers with the PRC-NPTN variant having the same number of parameters to enable us to perform multiple runs in a reasonable amount of time. The lower accuracy compared to other studies can be accounted by this. We train 5 models for each setting and report the mean and standard deviation of the errors. Training 5 runs for each of the hyperparameter combination to account for the randomization is yet another reason which tended to result in unreasonably large experiment times. Importantly, the goal of this experiment is not to push the state-of-the-art, but rather to investigate the behavior of DensePRC-NPTNs within the limits of computational resources available for this study while executing 5 runs for each network.

Discussion. Table. 4.3 presents the results of this experiment. We find

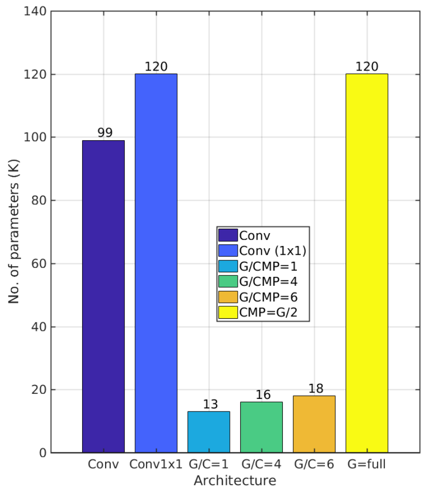
Method	CIFAR-10	(no Rand)	CIFAR 10++	(no Rand)	Speed	Memory
DenseNet	$11.47_{\pm 0.19}$	-	$21.37_{\pm 0.29}$	-		
PRCN (1)	$11.82_{\pm 0.20}$	$13.33_{\pm 0.23}$	$22.03_{\pm 0.08}$	$23.88_{\pm 0.38}$	1.29x	2.92x
PRCN (2)	$10.78_{\pm 0.31}$	$11.67_{\pm 0.36}$	$20.71_{\pm 0.23}$	$21.90_{\pm 0.33}$	1.34x	1.96x
PRCN (3)	$10.95_{\pm 0.12}$	$11.59_{\pm 0.23}$	$20.95_{\pm 0.20}$	$21.80_{\pm 0.42}$	1.36x	1.64x
PRCN (4)	$\mathbf{10.61}_{\pm \mathbf{0.11}}$	$11.41_{\pm 0.12}$	$\mathbf{20.80}_{\pm \mathbf{0.12}}$	$21.47_{\pm 0.16}$	1.36x	1.48x

Table 4.3: **Efficacy on CIFAR-10:** Test error statistics on CIFAR-10 with mean and standard deviation. ++ indicates AutoAugment testing. Each DenseNet and its corresponding PRC-NPTN variant has the same number of parameters (number in bracket determines CMP). $|G| = 12$ for PRC-NPTN and growth rate was kept at 12 for DenseNet-Conv. (w/o Random) indicates no randomization in the connectomes constructed (as an ablation study). The speed and memory improvements are multiplicative improvement factors of our CUDA kernel implementation compared to baseline optimized PyTorch code.

PRC-NPTN provides clear benefits even with architectures employing heavy use of skip connections such as DenseNets with the same number of parameters. Performance seems to increase as channel max pooling increased. Further, randomization seems to be important to the overall architecture even when given the complex nature of real image transformations. PRC-NPTN helps DenseNets account for nuisance transformations better even for those as extreme as auto-augment with its 16 transformation types ShearX/Y, TranslateX/Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout, Sample Pairing to various degrees. With these evidence, it is interesting to find that random connectomes can be motivated from the perspective of learning heterogeneous invariances from data without any change in architectures. We find that they provide a promising alternate dimension in future network design in contrast to the ubiquitous use of highly structured and ordered connectomes.



(a) Test Performance



(b) No. of Parameters (in thousands)

Figure 4.9: **PRC-NPTNs allow for significantly smaller networks with similar performance.** PRC-NPTNs provide a factor between **6.6× to 9.2× worth of reduction in parameters** while suffering a test accuracy degradation only between 3.08% to 4.82%. Whereas, using the same number of parameters, PRC-NPTN achieves 82.84% test accuracy versus the baseline 80.67%.

4.3.4 Exploring parameter reduction due to PRC-NPTNs

Until now, we have focused on the benefits of permanent random connectomes on better modelling nuisance transformations by learning these from the data itself. Though these provide performance and accuracy benefits per say, it is equally interesting to ask the corollary question: Given a certain level of performance in terms of accuracy, how low can the parameter requirement be pushed to?

We have seen a hint of such benefits in terms of parameters and the size of the network in Table. 4.2 showcasing the results on the ETH-80 dataset. We observed a healthy decrease factor in the number of parameters of up to 2 to 3 times. Additionally, an almost 5 to 10 times reduction in the number of convolutional filters being employed while obtaining similar or slightly greater accuracy. Though, these

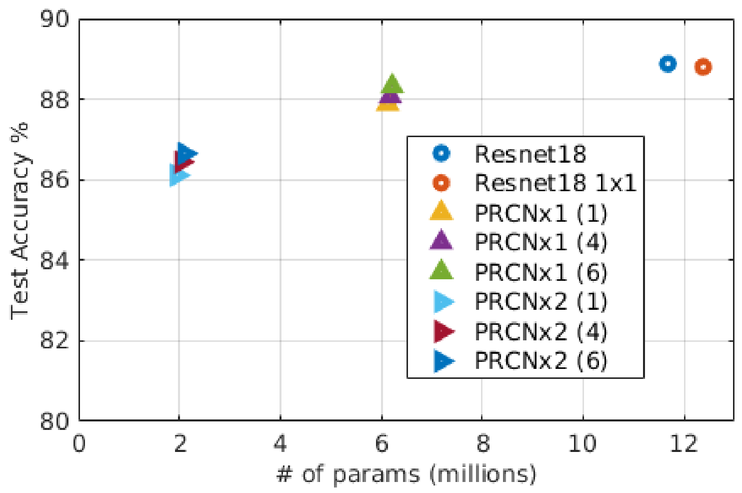
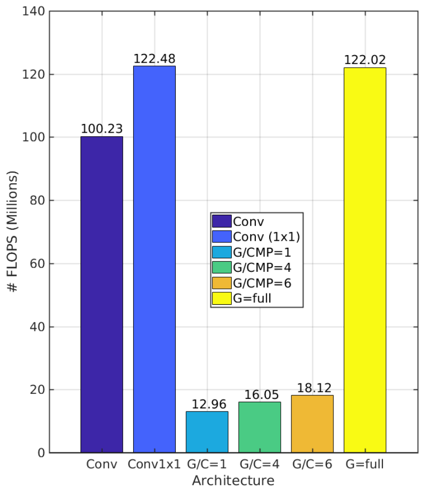


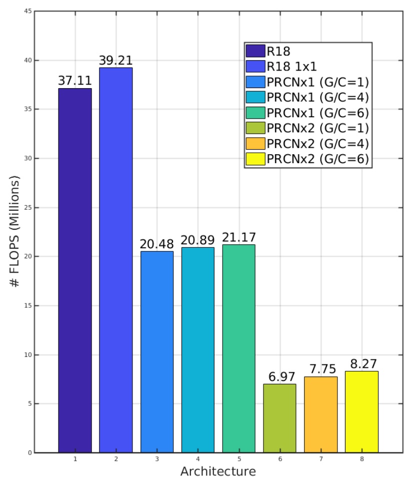
Figure 4.10: **PRC-NPTN version of ResNets allow for significantly smaller networks with similar performance.** PRC-NPTNs provide a factor between **1.93× to 6.1× worth of reduction in parameters** while suffering a test accuracy degradation only between 1% to 2.2%.

results seem interesting, it would be useful if PRC-NPTNs displayed similar levels of computational savings and competitive performance in more real-world datasets such as CIFAR 10 and larger more mainstream architectures.

Protocol: For this experiment, we explore two separate architectures on CIFAR 10. a) The first is a four layered vanilla convolution network with batch normalization, b) the second being the standard ResNet 18 architecture (BasicBlock), c) the popular VGG architecture and finally d) a 6 layered DenseNet. The set of these structures and their variations have been used extensively across many different applications. We follow a similar experimentation protocol as we did for CIFAR 10 in section 4.3.3. For both the vanilla ConvNet and ResNet 18 architecture, we also benchmark performance against the 1x1 version of these networks where are every convolution layer (except the first one), we add in a 1x1 layer to have the baselines be closer to the complexity of the PRC-NPTN layer with the 1x1 convolution layers (the pooling network). For ResNet 18, the default 1x1 layer in the bottleneck



(a) PRC-NPTN FLOPS



(b) ResNet PRC-NPTN FLOPS

Figure 4.11: **PRC-NPTNs allow for significantly more efficiently networks with similar performance.** Both for the four layered network and ResNet, the PRC-NPTN versions provide significant improvements in terms of FLOP reduction.

layer of the residual block doubled as the pooling network for the PRC-NPTN.

For PRC-NPTN version of the vanilla ConvNet we kept G=CMP while ranging the value between 1, 4 and 6 and also keeping G high enough so that number of filters in the network is the same as the baseline. This version is labelled as G=full in the result figures. For ResNet 18, we varied G similarly, however we also bench marked a second version of PRC-NPTN where both the convolution layers in the ResNet block were replaced with the PRC-NPTN variants for varying G. For VGG 19 and DenseNet, we kept G=CMP and varied as similarly between 1, 4 and 6.

Discussion: Fig. 4.9, Fig 4.10 and Table. 4.4 present the results of this experiment. We find that in Fig. 4.9, PRC-NPTNs provide a factor between 6.6× to 9.2× worth of reduction in parameters while only suffering from a 3.08% to 4.82% drop in test accuracy. Furthermore, while using the same number of parameters,

Method	Test Acc.	% change	Parameters	% decrease
VGG 19	**93.97%**	-	20.04 M	-
VGG 19 - PRCN	93.61%	0.38% decr.	**14.63 M**	**26.99%**
DenseNet	88.08%	-	44.41 K	-
DenseNet - PRCN	**89.1%**	**1.15% incr.**	**32.31 K**	**27.24%**

Table 4.4: **PRC-NPTNs allow for significantly smaller networks with similar performance.** PRC-NPTN versions of VGG 19 and DenseNet allow for about a 27% decrease in the number of parameters while suffering only a marginal decrease in accuracy in the case of VGG 19.

PRC-NPTNs achieve 82.84% accuracy versus 80.67% test accuracy consistent with previously observed trends. Even with residual networks as shown in Fig 4.10, PRC-NPTNs provide a factor between 1.93× to 6.1× worth of reduction in parameters while suffering a test accuracy degradation only between 1% to 2.2%. Table. 4.4 provides similar observations in the case of VGG 19 and a smaller DenseNet. Further, from Fig. 4.11 we find up to 5× to 9× more efficiency in terms of FLOPs between the a) four layered PRC-NPTN network and the b) PRC-NPTN version of ResNet 18 compared to their corresponding baselines. This showcases that architectural choices within the convolution layer itself do matter in terms of both accuracy and computational performance. Such benefits can expand to a large array of network architectures such as VGG, ResNets and DenseNets.

4.3.5 Pruning PRC-NPTNs for extreme parameter reduction

Until now, we have explored the benefits that the ideas of permanent random connectomes bring to the table in terms of better prediction performance, transformation modelling and computational complexity savings. We saw that networks employing PRC-NPTNs can allow significantly smaller sized networks compared to baselines to offer competing performance. This begs the question, how do PRC-NPTNs behave, when parameter efficiency is pushed to the limit through the practice of pruning deep networks? Pruning is an area that has gained popularity in recent years as a way to distill a large trained network down to a smaller size, while keeping most of its prediction performance intact. It is clear that the goal of such a process aligns with one of the practical motivations of PRC-NPTNs, *i.e.* to provide maximum prediction performance for a given amount of limited computational resources/parameters. In these experiments, we demonstrate that applying standard off-the-shelf pruning techniques to PRC-NPTNs provide a distilled (smaller/lower parameter networks) which offer better performance than vanilla ConvNet baselines.

Protocol: We utilized L1 pruning in these experiments. Here, a filter is chosen to prune out if the L1 norm of its response is in the bottom segment as defined by a hyperparameter. This effectively only keeps filters that on average provide high enough activation responses. The pruning factor can be set between 1 (no pruning) down to 0 (complete pruning). It can be understood as the factor or percent of parameters to keep while pruning/permanently deactivating the rest. We followed the standard experiment protocols for training on CIFAR 10 as in previous experiments. Once the network is trained, it is pruned in the trained state using a chosen pruning factor. Once the network is pruned, it is finetuned again using the same 300 epoch protocol with a learning rate starting from 0.01 decreased by a factor of 10 at 150 and 250 epochs. In this manner, a curve is formed by training a single model for a given fixed set of parameters, and then pruning the network for different amounts and finetuning. Thereby this is pruning at a single shot for a single amount and not iteratively as the degree of pruning increases. To study the effects of varying CMP, we bench-marked PRC-NPTNs for G fixed at 6 and ranged CMP from 2 through 12. On the other hand, to study the study of varying G, we

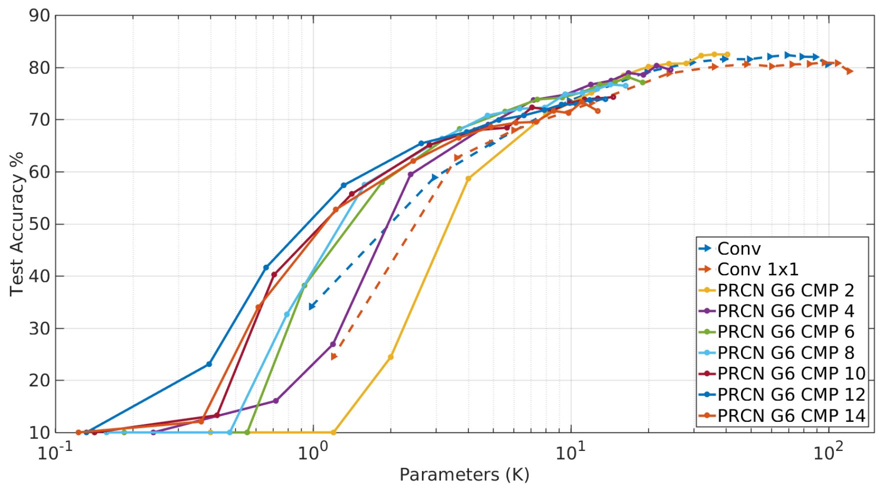


Figure 4.12: **Varying CMP: PRC-NPTN pruning for different amounts. Different models have varying CMP** PRC-NPTN along with pruning allows control over focus on performance at lower or higher parameter regimes. Each curve is a separate model trained for a different value of CMP. The yellow PRC-NPTN curve shows better performance at high parameter regime for low CMP. On the other hand, the dark blue PRC-NPTN curve shows better performance at low parameter regime using high CMP.

bench-marked PRC-NPTNs for CMP fixed at 8 and ranged G from 2 through 12.

Discussion: Fig. 4.12 and Fig. 4.13 showcase the results of this experiment. We find that PRC-NPTNs do indeed provide significantly higher test performance for a given parameter budget at the lower parameter settings. In fact, at around 1100 parameters in Fig. 4.12, PRC-NPTNs with CMP higher than 2 provide accuracies in the range around 60% whereas baseline ConvNet achieved 50% and in fact the deeper ConvNet 1×1 baseline achieved only 30% accuracy. Furthermore and interestingly, we observe that for lower CMP=2, PRC-NPTN outperforms both baselines with an accuracy of 82.44% using 40.4K parameters versus the highest achieved by ConvNet baseline of 82.33% using 68.9K parameters. Hence, PRC-NPTNs provide this additional tuning parameter called CMP or channel max pooling, using with

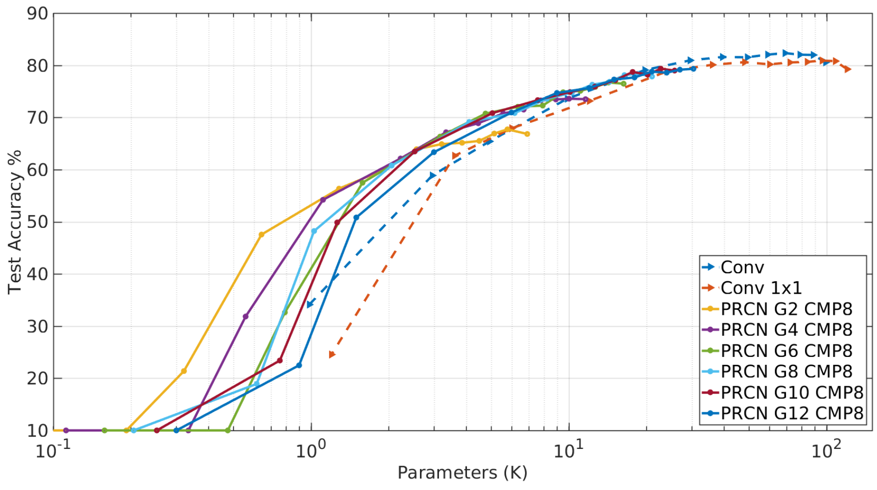


Figure 4.13: **Varying G: PRC-NPTN pruning for different amounts. Different models have varying G** PRC-NPTN along with pruning allows control over focus on performance at lower or higher parameter regimes. Each curve is a separate model trained for a different value of G. The yellow PRC-NPTN curve shows better performance at low parameter regime for low G. On the other hand, the dark blue PRC-NPTN curve shows better performance at high parameter regime using high G.

higher performance can be desired in lower and higher parameter settings. We find similar trends in Fig. 4.13 with much higher performance at even lower ranges. However, performance peaks out in the higher parameter regime with larger G. Overall, this provides a new and effective dimension along with one can decide how best to utilize the resources utilized by the network. Such a dimension simply does not exist with baseline architectures which do not have the PRC-NPTN structure.

Choosing G and CMP: Looking at these results, we find that a potential method of choosing hyperparameters G and CMP is to have it depend on the compute resources of the application at hand. If the application is able to support larger models and provide more resources to the model to perform, then a higher G and lower CMP PRC-NPTN is expected to be beneficial. However, for compute constrained environments such as edge devices etc., a lower G and higher CMP might prove to be more useful. These are simply heuristics upon which this decision need be made. Some amount of hyperparameter optimization will nonetheless be needed to pick optimum values for any application and it's data. That said, PRC-NPTNs do provide a new dimension along which networks can be further optimized to provide benefits that can adapt to each individual application. This was previously not possible with vanilla ConvNets and it's derivative structures.

Chapter 5

The Next Chapter

The study of representation learning seems to be right at the heart of perception and reasoning. Indeed, developing invariance and striving towards an invariant representation lies at the center of better generalization, robustness, sample and computational efficiency. Given these benefits, one does wonder why hasn't the core vanilla convolution layer adapted to better learn and enforce invariance. Addressing this need and exploring some useful principles along the way was the primary objective of this endeavour.

We presented the Transformation Network paradigm, which was the first attempt at efficiently learning and enforcing invariance within a convolution layer. Building on top, we developed Non-Parametric Transformation Networks (NPTNs) which are capable of learning transformation invariance from data itself. NPTN being a direct generalization of the convolution layer provided deeper insights into what makes a network generalize better. The channel max pooling operation not only served as an approximate invariant function, but also dynamically and cheaply directed flow of information through a network.

Following these insights, we sought to apply them to the question, why do biological circuits having almost entirely random connectivity at local level still provide consistent visual perception? Interestingly, we observe that these unchanging random connectomes in fact help to drastically reduce the complexity of the network to generate invariance. This is the first glimpse towards the paradigm that randomness in biological circuits is not something to overcome and work around, but rather is an asset towards better efficiency and generalization. Using these insights, we developed Permanent Random Connectome Networks or PRCNs. These networks provided further performance benefits compared to NPTNs and also observed significant parameter efficiency. Indeed, many of the networks that offered similar performance to the baseline networks were many times smaller in size. This throws light onto what core mechanisms are critical in information processing within networks and which are not.

These observations force us to explore more efficient neural architectures and to be inspired by biological mechanisms not for its own sake, but rather solely from the perspective of functionality. The approach of understanding theoretical principles driving perception in artificial networks, and then looking for these principles in the structure and functioning of biological circuits, promises to be extremely valuable. Naturally, one must be vary of ones own biases and also let biology and theory guide us through mutual reinforcement. An idea that is theoretically useful must be present in biology in some form and vice versa for the idea to have merit towards the how intelligence in implemented in nature. This is the process that this thesis provides evidence towards, and is the one that I hope to adhere to in the future. The ambition shall always remain:

To reason about perceiving while better perceiving reasoning.

Bibliography

[1] Robert Gens and Pedro M Domingos, "Deep symmetry networks," in *Advances in neural information processing systems*, 2014, pp. 2537–2545. (document), 1.2, 3.1, 3.2, 4.1.1

[2] Partha Niyogi, Federico Girosi, and Tomaso Poggio, "Incorporating prior information in machine learning by creating virtual examples," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2196–2209, 1998. 1

[3] Raia Hadsell, Sumit Chopra, and Yann LeCun, "Dimensionality reduction by learning an invariant mapping," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. IEEE, 2006, vol. 2, pp. 1735–1742. 1, 4.1.1

[4] Qianli Liao, Joel Z Leibo, and Tomaso Poggio, "Learning invariant representations and applications to face verification," in *Advances in Neural Information Processing Systems*, 2013, pp. 3057–3065. 1, 1.2, 2.4, 3.2, 4.2.1, 4.2.1

[5] Dipan K Pal, Felix Juefei-Xu, and Marios Savvides, "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, 2016, pp. 5590–5599. 1, 1.2, 2.4, 3.2, 4.2.1, 4.2.1

[6] Dipan K. Pal, Ashwin Kannan, Gautam Arakalgud, and Marios Savvides, “Max-margin invariant features from transformed unlabelled data,” in *Advances in Neural Information Processing Systems 30*, 2017, pp. 1438–1446. 1, 1.2, 2.2, 2.4, 4.2.4

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 1.1, 4.1.1, 4.1.1, 4.2.2

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 1.1

[9] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten, “Densely connected convolutional networks,” *arXiv preprint arXiv:1608.06993*, 2016. 1.1

[10] Sergey Zagoruyko and Nikos Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016. 1.1

[11] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng, “Dual path networks,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4470–4478. 1.1

[12] Jie Hu, Li Shen, and Gang Sun, “Squeeze-and-excitation networks,” *arXiv preprint arXiv:1709.01507*, 2017. 1.1

[13] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton, “Dynamic routing between capsules,” in *Advances in Neural Information Processing Systems*, 2017, pp. 3859–3869. 1.1, 3.3.4

[14] Jyri J Kivinen and Christopher KI Williams, “Transformation equivariant boltzmann machines,” in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 1–9. 1.2

[15] Kihyuk Sohn and Honglak Lee, “Learning invariant representations with local transformations,” *arXiv preprint arXiv:1206.6418*, 2012. 1.2

[16] Beat Fasel and Daniel Gatica-Perez, “Rotation-invariant neoperceptron,” in *Pattern Recognition, ICPR. 18th International Conference on*. IEEE, 2006, vol. 3, pp. 336–339. 1.2, 3.2.1, 4.1.1

[17] Sander Dieleman, Kyle W Willett, and Joni Dambre, “Rotation-invariant convolutional neural networks for galaxy morphology prediction,” *Monthly notices of the royal astronomical society*, vol. 450, no. 2, pp. 1441–1459, 2015. 1.2, 3.2.1, 4.1.1, 4.1.1

[18] Damien Teney and Martial Hebert, “Learning to extract motion from videos in convolutional neural networks,” in *Asian Conference on Computer Vision*. Springer, 2016, pp. 412–428. 1.2, 3.1, 3.2, 3.2.1, 3.2.2, 4.1.1, 4.1.1, 4.3.1

[19] Fa Wu, Peijun Hu, and Dexing Kong, “Flip-rotate-pooling convolution and split dropout on convolution neural networks for image classification,” *arXiv preprint arXiv:1507.08754*, 2015. 1.2, 2.4, 3.1, 3.2, 3.2.1, 3.2.2, 4.1.1

[20] Junying Li, Zichen Yang, Haifeng Liu, and Deng Cai, “Deep rotation equivari-

ant network," *arXiv preprint arXiv:1705.08623*, 2017. 1.2, 2.4, 3.1, 3.2.1, 3.2.2, 4.1.1, 4.1.1, 4.3.1

[21] Yichong Xu, Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, and Zheng Zhang, "Scale-invariant convolutional neural networks," *arXiv preprint arXiv:1411.6369*, 2014. 1.2, 4.1.1, 4.3.1

[22] Laurent Sifre and Stéphane Mallat, "Rotation, scaling and deformation invariant scattering for texture discrimination," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 1233–1240. 1.2, 4.1.1, 4.3.1

[23] Taco Cohen and Max Welling, "Group equivariant convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2990–2999. 1.2, 4.1.1, 4.1.1, 4.3.1

[24] João F Henriques and Andrea Vedaldi, "Warped convolutions: Efficient invariance to spatial transformations," in *International Conference on Machine Learning*, 2017. 1.2, 4.1.1, 4.1.1, 4.3.1

[25] Taco S Cohen and Max Welling, "Steerable cnns," *arXiv preprint arXiv:1612.08498*, 2016. 1.2, 4.1.1, 4.1.1

[26] Fabio Anselmi, Joel Z Leibo, Lorenzo Rosasco, Jim Mutch, Andrea Tacchetti, and Tomaso Poggio, "Unsupervised learning of invariant representations in hierarchical architectures," *arXiv preprint arXiv:1311.4158*, 2013. 1.2, 2.2, 2.3, 2.3, 2.4, 4.1.1, 4.2.1, 4.2.3, 4.2.4

[27] Fabio Anselmi, Georgios Evangelopoulos, Lorenzo Rosasco, and Tomaso Pog-

gio, "Symmetry regularization," Tech. Rep., Center for Brains, Minds and Machines (CBMM), 2017. 1.2, 2.4, 4.2.1

[28] Koray Kavukcuoglu, Rob Fergus, Yann LeCun, et al., "Learning invariant features through topographic filter maps," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1605–1612. 1.2

[29] Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang W Koh, Quoc V Le, and Andrew Y Ng, "Tiled convolutional neural networks," in *Advances in neural information processing systems*, 2010, pp. 1279–1287. 1.2

[30] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 44–51. 1.2

[31] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. 1.2

[32] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066. 1.2

[33] Matthew D Zeiler and Rob Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013. 1.2

[34] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He, "Exploring randomly wired neural networks for image recognition," *arXiv preprint arXiv:1904.01569*, 2019. 1.2

[35] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio, "Maxout networks," in *International Conference on Machine Learning*, 2013, pp. 1319–1327. 2.1

[36] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015, pp. 448–456. 3.3.1

[37] Bastian Leibe and Bernt Schiele, "Analyzing appearance and contour based methods for object categorization," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*. IEEE, 2003, vol. 2, pp. II–409. 3.3.2, 4.3.2

[38] Renata Khasanova and Pascal Frossard, "Graph-based isometry invariant representation learning," in *International Conference on Machine Learning*, 2017, pp. 1847–1856. 3.3.2, 4.3.2, 4.3.2, 4.3.2

[39] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al., "Spatial transformer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2017–2025. 3.3.2, 4.1.1

[40] Edouard Oyallon and Stéphane Mallat, "Deep roto-translation scattering for object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2867–2873. 3.3.2

[41] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J

Brostow, “Harmonic networks: Deep translation and rotation equivariance,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5028–5037. 3.3.2, 4.3.2

[42] Stéphane Mallat and Irene Waldspurger, “Deep learning by scattering,” *arXiv preprint arXiv:1306.5532*, 2013. 4.1.1, 4.1.1

[43] Joe Corey and Benjamin Scholl, “Cortical selectivity through random connectivity,” *Journal of Neuroscience*, vol. 32, no. 30, pp. 10103–10104, 2012. 4.1.2

[44] Manuel Schottdorf, Wolfgang Keil, David Coppola, Leonard E White, and Fred Wolf, “Random wiring, ganglion cell mosaics, and the functional architecture of the visual cortex,” *PLoS computational biology*, vol. 11, no. 11, pp. e1004602, 2015. 4.1.2

[45] David Hansel and Carl van Vreeswijk, “The mechanism of orientation selectivity in primary visual cortex without a functional map,” *Journal of Neuroscience*, vol. 32, no. 12, pp. 4049–4064, 2012. 4.1.2

[46] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman, “Random synaptic feedback weights support error backpropagation for deep learning,” *Nature communications*, vol. 7, pp. 13276, 2016. 4.1.2

[47] Stephen Grossberg, “Competitive learning: From interactive activation to adaptive resonance,” *Cognitive science*, vol. 11, no. 1, pp. 23–63, 1987. 4.1.2

[48] David G Stork, “Is backpropagation biologically plausible,” in *International Joint Conference on Neural Networks*, 1989, vol. 2, pp. 241–246. 4.1.2

[49] Pietro Mazzoni, Richard A Andersen, and Michael I Jordan, "A more biologically plausible learning rule for neural networks.," *Proceedings of the National Academy of Sciences*, vol. 88, no. 10, pp. 4433–4437, 1991. 4.1.2

[50] Xiaohui Xie and H Sebastian Seung, "Equivalence of backpropagation and contrastive hebbian learning in a layered network," *Neural computation*, vol. 15, no. 2, pp. 441–454, 2003. 4.1.2

[51] Dipan K Pal and Marios Savvides, "Non-parametric transformation networks for learning general invariances from data," *AAAI*, 2019. 4.2.5, 4.2.5, 4.3.1, 4.3.1, 4.3.2

[52] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le, "Autoaugment: Learning augmentation policies from data," *arXiv preprint arXiv:1805.09501*, 2018. 4.3.3