A Tensor-based Convolutional Neural Network for Small Dataset Classification

Zhenhua Chen
Indiana University
Bloomington, IN 47408
chen478@iu.edu

David J. Crandall
Indiana University
Bloomington, IN 47408
djcran@iu.edu

Abstract

Inspired by ConvNets leveraging hierarchical representations, we introduce Tensor-based ConvNets (TConvNets) employing hierarchical neurons. TConvNets, a more generalized form of ConvNets, necessitate a generalized version of components and operations. Unlike ConvNets with scalar neurons, TConvNets use tensor-based neurons, relying on tensor production and combination as core operations instead of linear combinations. Key components, including tensor-based batch normalization and initialization, are developed for TConvNets. Additionally, methods for structuring/unstructuring input/output allow the utilization of ConvNets components like loss functions in TConvNets. Although TConvNets may offer many new attributes, this paper focuses solely on parameter-wise efficiency. Through constructing a TConvNet with high-rank neuron tensors, we conducted performance comparisons on CIFAR10, CIFAR100, and Tiny ImageNet datasets, revealing TConvNets' superior efficiency in parameter utilization.

1 Introduction

1.1 Overparameterization of Neural Networks

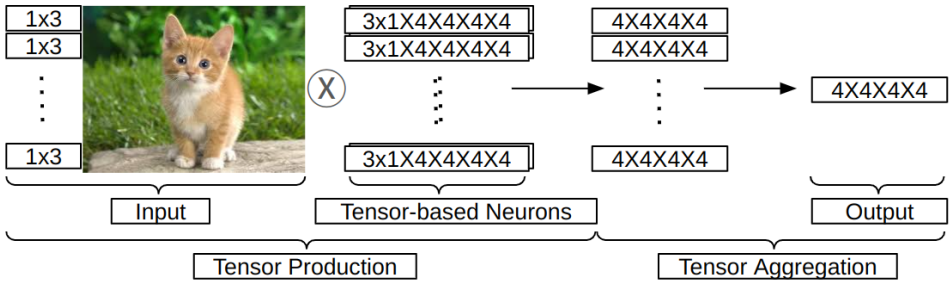
ConvNets are continually growing in size. While the initial motivation for the increasing size of models was primarily driven by their improved performance on large datasets, subsequent research has revealed additional advantages associated with these larger models. One of them is discriminability. With an increased number of parameters, larger models have the potential to capture intricate and complex features when provided with ample data and training epochs, leading to improved performance on challenging tasks. Recent advancements in generative models for languages, images, and videos further substantiate this claim [1].

Consequently, researchers have been reflecting the necessity of using so many parameters. This contemplation has revealed that ConvNets often suffer from the problem of overparameterization. In response, researchers have devised various approaches to compress neural networks.

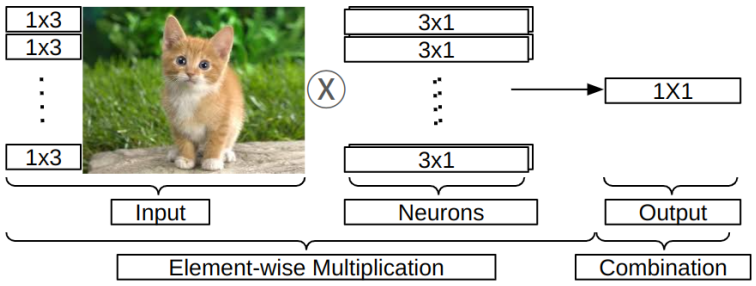
1.2 Pruning of Neural Networks

Pruning involves the removal of connections or neurons in a network that has limited contribution to the final output or limited impact on performance. This can be accomplished by directly removing connections with small weights or activations, or by applying heuristic algorithms to selectively remove neurons to achieve a balance between maximizing neuron removal and preventing significant degradation in the pruned model's performance [2, 3, 4]. Compression generally adopts matrix decomposition techniques like SVD or Tucker decomposition, etc. to replace the original weight matrix with smaller ones [5, 6, 7, 8]. Quantization replaces float weights with quantized numbers,

38th Conference on Neural Information Processing Systems (NeurIPS 2024).



(a)



(b)

Figure 1: Tensor-based convolution (a) versus normal convolution (b). The main idea of (a) is utilizing a tensor-based neuron with dimensions [3, 1, 4, 4, 4, 4] to transform input data of shape [1, 3] into a structured representation of shape [4, 4, 4, 4] through tensor production. Subsequently, all otput tensors, each with a shape of [4, 4, 4, 4], are aggregated to obtain a final output tensor with a shape of [4, 4, 4, 4]. In (b), a linear combination is applied between the input and the neurons (kernels).

which may reduce a model's size at the cost of performance [9, 10]. Distillation has the classic teacher-student framework, where a smaller neural network (the student) is trained using soft targets provided by the original network (the teacher) [11]. As a result, the knowledge is transferred from the teacher network to the student network, resulting in the student achieving comparable performance while being more compact. Each of these techniques may have varying degrees of impact on the performance compared to the original model.

All of the mentioned approaches follow a common principle: initially constructing large ConvNets and subsequently reducing their sizes. By employing pruning and compression techniques, it is possible to reduce the size of many large ConvNets, sometimes up to 90%. This raises the question: if a significant portion of parameters in ConvNets can be reduced, why not utilize fewer parameters from the beginning? Can we create a compact neural network in the first place? Some researchers have discovered that ConvNets with some structured neurons help alleviate overparameterizations [12, 13]. A logical question emerges: Can we further take advantage of the structured neuron attributes?

1.3 Tensor-based Neurons

In this context, scalar neurons mean that the neurons are not organized in a hierarchical manner, such as those in ConvNets. The capsules in CapsNets fall into the slot of structured neurons. In particular, CapsNets leverage feature maps originating from convolutional layers and subsequently re-organize these feature maps in a structured way, finally use capsules (matrix neurons) to map these representations.

Why not generate structured representations directly from neurons? TConvNets utilize tensor neurons to produce these representations, eradicating the need for a feature-extraction process. Moreover, we can adapt the matrix neurons in capsules to higher-rank tensor-based neurons to maximize advantages. This is the fundamental concept of TConvNets. Of course, we must establish appropriate basic

operations across layers, such as “tensor-based convolution", initialization, batch normalization, and other components for TConvNets.

So how are we supposed to build TConvNets? First, we employ high-rank tensor-based neurons and utilize the tensor product as the fundamental operations within the network. Next, we incorporate parameter sharing within each layer, similar to how convolutions operate, but with the use of tensor neurons. In summary, we adopt high-rank tensors as structured neurons, employ parameter sharing within each layer similar to ConvNets, package and unpackage the input/output to form and decompose tensors, modify batch normalization & initialization, maintain the same, activation & loss functions, and make the tensor product the default operation across layers. This new neural network, known as Tensor-based Convolutional Neural Networks (TConvNets), follows a similar organizational structure to ConvNets, except that the basic unit is a high-rank neuron tensor. Therefore, when each structured neuron comprises only a single scalar neuron, TConvNets essentially reduce to regular ConvNets. In other words, ConvNets can be viewed as a specialized type of TConvNets, while TConvNets represent a more general form of ConvNets. Figure 1 shows one example of tensor-based convolutions.

2 Related Work

2.1 Parameter Efficiency of Neural Networks

The effectiveness of neural networks can be assessed from various perspectives, such as the number of parameters, FLOPS, memory consumption, and training/inference time, etc. The number of parameters employed is a crucial aspect, as ConvNets are well known for their overparameterization, which restricts their deployment on resource-constrained environments like mobile devices. To mitigate this issue, two approaches can be taken: either compress the neural networks [5, 6, 7, 8], or design more efficient neural network architectures, such as SqueezeNets [14], MobileNets [15], ResNets [16], EfficientNets [17], and ShuffleNets [18]. Furthermore, reducing the model size by tailoring it to specific tasks or devices [19, 20, 21] is another viable option. In this paper, we consider these classic ConvNets as benchmarks and compare them with TConvNets.

2.2 Capsule Networks

Capsule networks [12] aims to better model hierarchical relationships between parts of objects in data. A typical CapsNet is composed of several convolutional layers, a final fully-connected capsule layer with a routing procedure, and a loss function, as Figure 2 shows. There are two key designs in CapsNets, one is the rouging mechanism, which helps in determining the optimal routing of information between capsules. In this way, the network can focus on relevant parts of the input data and ignore irrelevant information. For example, the CapsNet with dynamic routing [12] can separate overlapping digits more accurately, while the CapsNet with EM routing [13] achieves a lower error rate on smallNORB [22]. The other design is the capsule architecture. CapsNets organize data into a hierarchical structure, allowing for better representation and recognition of complex objects/relationships. In this way, capsules can effectively compress high-dimensional input data into lower-dimensional representations, making it easier for the network to learn meaningful patterns and relationships with fewer parameters. In summary, the above two key designs make CapsNets be able to effectively represent complex relationships within data.

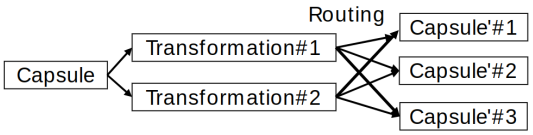


Figure 2: Capsule Networks.

The hierarchical attributes of CapsNets enables them to effectively represent information at different levels of abstraction. This is similar to the organization of cerebral cortex, which is also organized hierarchically to handel different tasks. Consequently, CapsNets can achieve similar accuracy while requiring significantly fewer parameters compared to ConvNets. In contrast, ConvNets are usually

overparameterized. As shown in [23, 24, 25, 26, 27], their compressed/pruned neural networks have much smaller sizes with hardly any accuracy drop.

Both TConvNets and CapsNets encode representations in a structured way. However, TConvNets differentiate from CapsNets in three key aspects:

- TConvNets employ tensor neurons to directly generate these structured hidden representations, whereas CapsNets produce these structured representations by rearranging the feature maps from ConvNets.
- tensor neurons serve as the fundamental units throughout the entire TConvNets, as opposed to being specialized components integrated into ConvNets.
- TConvNets do not rely on information routing mechanisms or specific encoding algorithms. The relationships between adjacent structured hidden representations in TConvNets are learnable, enabling the model to capture the necessary information without predefined procedures.

2.3 Other Tensor-based Neural Networks

Several other papers share similar titles to ours [28, 29, 30, 31]. For instance, [28] introduces a specialized operator based on Entangled Plaquette States (EPS), while [29] and [31] focus on compressing ConvNets. I believe [30] and our initialization are mathematically equivalent, though [30] is based on hypergraphs. These papers primarily aim to compress models built on ConvNets, whereas our work focuses on constructing tensor-based ConvNets from the ground up.

3 Tensor-based Convolutional Neural Networks

The primary operation at the core of ConvNets is convolution, which is a linear combination consisting of two sequential steps. Initially, it involves performing an element-wise multiplication between scalar neurons of the kernel size and feature maps of the same size. Following this, the resultant scalar values are aggregated to yield a single scalar value in the subsequent layer. In contrast, the foundational operation in TConvNets, known as tensor-based convolution, entails a linear combination of tensors. Unlike traditional convolutions, tensor-based convolutions commence by conducting tensor product operations between tensor neurons of the kernel size and structured hidden representations of the same size. Afterward, a combination operation is applied to these output tensors, which are structured hidden representations.

As a consequence, every convolution operation yields a single value, whereas each step of tensor-based convolution generates a multidimensional tensor. This leads to ConvNets and TConvNets having analogous yet distinct types of feature maps. Standard feature maps maintain a consistent homogeneity across layers, primarily differing in their dimensions (height, width, and number). Conversely, feature maps within TConvNets can display varying structures across layers. For example, a ConvNet feature map might be denoted as 32x128x128, signifying the presence of 32 channels, each with dimensions 128x128. In contrast, TConvNets' feature maps are expressed as 32x128x128xT, where T can vary between layers, ranging from T=1 to more complex tensors like T=3x4x5x6. When T equals 1, a tensor-based convolutional layer essentially becomes a conventional convolutional layer. Moreover, if T remains consistently equal to 1 throughout the network, TConvNets essentially transform into ConvNets.

Next, we provide a comprehensive explanation of tensor-based convolution and elaborate on certain adjustments we implement to ensure the feasibility of training TConvNets.

3.1 Batch Normalization for TConvNets

Batch normalization plays an important role in stabilizing the gradients in forward/backward passes. Similar to batch normalization in ConvNets, where batch normalization is typically applied in each channel, we generalized this idea, namely, not only applying batch normalization across channels but also across each structured feature dimension. Without this adapted batch normalization for TConvNets, the overall performance sees a significant decline. For more details on this implementation, refer to the Supplemental Material A.1.

3.2 Initialization for TConvNets

To give good initial values to the neuron tensors, we build upon the methods in [32] and [33], adapting them to suit TConvNets. In particular, we ensure that the variance and covariance among neuron tensors remain constant across layers, preventing gradient vanishing or exploding issues. For the forward pass, initialize using a zero-mean Gaussian distribution with a standard deviation of $\sqrt{\frac{2}{n_l d}}$, where $n_l = k^2 \times c$, where k is the kernel size, c is the input channel. Here, d represents the total dimensionality of the neuron tensor, calculated as $t_1 \times t_2 \dots t_n$, where each t denotes a dimension in the neuron tensor. The details are available in the Supplemental Material A.2.

3.3 Tensor-based Convolution

Tensor neurons serve as the fundamental units within the entire TConvNets architecture, responsible for converting input feature maps into output feature maps. For instance, they can transform an input tensor, denoted as $\mathbf{U} \in \mathbb{R}^{1\times2\times3\times4}$, into an output tensor, denoted as $\mathbf{V} \in \mathbb{R}^{1\times2\times7\times8}$, through the application of a tensor product operation with $\mathbf{W} \in \mathbb{R}^{4\times3\times7\times8}$, as Equation 1 shows. In theory, the tensor neurons within TConvNets have the capability to convert tensors of any shape into tensors of any other shape.

$$\mathbf{V}_i = \mathbf{U}_i \bigotimes \mathbf{W}_i \tag{1}$$

After the tensor production, TConvNets apply liner combinations, which can be defined as,

$$\mathbf{V} = \sum_{i}^{n} \mathbf{V}_i \tag{2}$$

Where $n = k \times k \times m$, k is the kernel size and m is the number of input channels. In comparison, the basic operation of ConvNets is linear combinations of scalars, $V = \sum_i^n V_i$.

3.4 Input&Output in TConvNets

We process each input image as in Figure 1 shows. To illustrate, consider a 128x128x3 color image, which can be regarded as a one-channel structured feature map with dimensions of $\mathbf{U} \in \mathbb{R}^{1\times128\times128\times(1\times3)}$. If we employ a tensor neuron with dimensions of ($\mathbf{W} \in \mathbb{R}^{(3\times3)\times1\times1\times3\times1\times4\times4\times4\times4)}$, the input can be transformed to a one-channel structured feature map with dimensions of $\mathbf{V} \in \mathbb{R}^{4\times63\times63\times(4\times4\times4\times4)}$, as Equation 5 shows. In this context, we are assuming a kernel size of 3, a stride of 2, and no padding.

With both the input and the neurons structurized, the resulting output naturally inherits a structured format. Consequently, this necessitates the loss functions specifically designed to handle structured data, such as the margin loss in CapsNets. However, these loss functions often require additional steps. A more convenient approach is to destructurize the output and then use a regular loss function. Compared to the final output of ConvNets, TConvNets' final output has an extra dimension, namely the structured feature dimension $\mathbf{T}^{'} \in \mathbb{R}^{t_1^{'}\times t_2^{'}\dots t_n^{'}}$, and $d^1 = t_1^{'} \times t_2^{'} \dots t_n^{'}$. Our solution is compressing the last dimension by using $\mathbf{W}$ whose last dimension $\mathbf{T} \in \mathbb{R}^{t_1^{'}\times t_2^{'}\dots t_n^{'}}$, make it become normal feature maps. In other words, the final layer's outout of TConvNets becomes the same as ConvNets, as Equation 3 shows.

$$\mathbf{V}_i^f = \mathbf{U}_i \bigotimes \mathbf{W}_i \tag{3}$$

For example, assume we need to compress the tensors of the last layer $\mathbf{U} \in \mathbb{R}^{4\times4\times4\times4}$ to $\mathbf{V}^f \in \mathbb{R}^{1\times1\times1\times1}$ to fit a loss function. The neuron tensors is still $\mathbf{W}^{in} \in \mathbb{R}^{4\times4\times4\times4}$, with all the dimensions to contract.

4 Experiments

We conducted experiments with TConvNets on several small datasets, specifically CIFAR10 [34], CIFAR100 [34], and TinyImageNet [35]. Initially, we evaluated TConvNets against other ConvNets in a conventional manner, employing various data augmentation techniques such as resizing, cropping, flipping, and normalization. Subsequently, we conducted experiments without applying these techniques, maintaining the original input size and avoiding resizing, cropping, or flipping. Additionally, we omitted the normalization step based on prior knowledge. This approach was chosen to eliminate extraneous factors and provide a more accurate assessment and comparison of the performance across different neural network architectures. Throughout this paper, we maintain the same choice of a learning rate of 1e-3, a mini-batch size of 32, and the utilization of the Adam optimizer [36] across all datasets and settings. All experiments are carried out utilizing the NVIDIA RTX A6000 unless otherwise stated. All the benchmark models are from the PyTorch library [37].

We employ a singular TConvNets, referred to as t_conv_net, for all the datasets. The comprehensive configuration of this network can be observed in the Supplemental Material A.3.

4.1 CIFAR10 [34], CIFAR100 [34] and Tiny ImageNet [35]

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. For every sample in CIFAR10, our preprocessing pipeline involves resizing each image to dimensions of 42x42. Subsequently, we apply random cropping, resulting in a final size of 38x38. We also incorporate horizontal flipping for data augmentation purposes. Finally, each channel of an image is normalized using mean values of (0.485, 0.456, 0.406) and standard deviations of (0.229, 0.224, 0.225).

We then refrain from any preprocessing of the original CIFAR10 data. Specifically, we utilize the original input size of 32x32 and do not apply resizing, cropping, or normalization. As Table 1 shows, t_conv_net achieves the best performance with/without data augmentations than most models using far fewer parameters.

CIFAR100 has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. We first use exactly the same data augmentations as in section 4.1. Subsequently, we eliminate all augmentation techniques and re-evaluate our model. As evidenced by Table 1, t_conv_net exhibits the most superior performance among all models, both with and without data augmentations.

We observe similar findings in Tiny ImageNet [35]. For more detailed results, please refer to Supplemental Material A.4.

4.2 Ablation Study

We have seen some good results from pure TConvNets on classification tasks. It would be insightful to explore a neural architecture that combines both TConvNets and ConvNets. Semantic segmentation tasks, typically composed of backbones and heads, offer an ideal scenario to evaluate such a hybrid model.

Specifically, we evaluate a hybrid model featuring the same pre-trained ResNet101 backbone alongside different heads. These heads are sourced from FCN, DeepLab, or TConvNets. The ResNet101 backbone comes pre-trained (utilizing weights=DeepLabV3_ResNet101_Weights.DEFAULT) from the PyTorch library [37], while the heads are initialized randomly. The overarching learning rate is set at 1e-2, with a weight decay of 1e-1, and we adjust the heads to learn 200 times faster. The input size for both training and testing is 256. Similar to the setup for classification tasks, we refrain from using data augmentations apart from normalization. For optimization, we employ AdamW [40].

As depicted in Table 2, our model attains the highest IoU with fewer parameters on the CITYSCAPES dataset. On the Pascal VOC dataset, our model achieves the second-highest performance, surpassing the FCN model but falling short of the DeepLab model. We argue that our model employs only one head (DeepLab uses four heads) and abstains from utilizing dilated convolutions. In scenarios where a single head without dilated convolutions, akin to FCN, is employed, our TConvNets-based head demonstrates superior performance.

Table 1: TConvNets Performance Results on CIFAR10, CIFAR100, and Tiny ImageNet. The epoch# here is the epoch number when the lowest validation loss is recorded. For each accuracy item, the number on the left indicates the utilization of augmentations, while the number on the right indicates the absence of any augmentations.

Datasets	*Model*	*Acc.*	*#Params*	*#Epochs*
CIFAR10	shufflenet_v2_x0_5 [18]	76.5%/60.5%	**0.35M**	47/9
	t_conv_net	**88.4%/ 78.4%**	1.43M	71/11
	mobilenet_v3_small [15]	80.0%/63.2%	1.53M	17/13
	mobilenet_v2 [15]	84.4%/73.0%	2.24M	32/17
	regnet_y_400mf [38]	82.7%/68.0%	3.91M	30/14
	EfficientNet-B0 [17]	85.6%/70.0%	4.0M	79/8
	regnet_y_800mf [38]	82.9%/33.5%	5.66M	21/6
	EfficientNet-B1 [17]	87.2%/72.9%	6.5M	66/14
	EfficientNet-B2 [17]	86.1%/75.0%	7.72M	43/20
	resnet18 [16]	86.1%/73.9%	11.18M	13/6
	resnet34 [16]	86.5%/74.2%	21.29M	17/8
	convnext_tiny [39]	76.2%/60.4%	27.82M	19/5
	convnext_small [39]	84.6%/59.8%	49.45M	15/5
CIFAR100	shufflenet_v2_x0_5 [18]	39.7%/32.3%	**0.44M**	14/14
	t_conv_net	**56.8%/44.4%**	1.46M	81/23
	mobilenet_v3_small [15]	37.2%/33.8%	1.62M	20/28
	mobilenet_v2 [15]	54.5%/38.8%	2.35M	46/20
	regnet_y_400mf [38]	46.0%/31.9%	3.95M	19/8
	EfficientNet-B0 [17]	53.0%/35.0%	4.14M	38/21
	regnet_y_800mf [38]	49.2%/38.2%	5.73M	13/10
	EfficientNet-B1 [17]	53.2%/32.2%	6.64M	80/8
	EfficientNet-B2 [17]	54.2%/44.3%	7.84M	13/14
	resnet18 [16]	53.8%/42.9%	11.23M	12/9
	resnet34 [16]	53.4%/42.4%	21.34M	17/7
	convnext_tiny [39]	42.8%/30.4%	27.89M	15/5
	convnext_small [39]	44.1%32.6%	49.52M	16/8
Tiny ImageNet	shufflenet_v2_x0_5 [18]	35.7%/28.6%	**0.55M**	40/13
	t_conv_net	**48.5%**/34.8%	1.49M	75/25
	mobilenet_v3_small [15]	33.8%/28.7%	1.72M	83/18
	mobilenet_v2 [15]	39.1%/32.4%	2.35M	35/16
	regnet_y_400mf [38]	38.2%/29.3%	3.99M	21/12
	EfficientNet-B0 [17]	42.6%/32.4%	4.26M	90/13
	regnet_y_800mf [38]	40.5%/**35.2%**	5.8M	18/8
	EfficientNet-B1 [17]	44.4%/32.2%	6.77M	92/17
	EfficientNet-B2 [17]	40.4%/31.3%	7.98M	109/16
	resnet18 [16]	40.0%/33.0%	11.28M	12/5
	resnet34 [16]	39.9%/34.2%	21.39M	19/5
	convnext_tiny [39]	37.7%/26.9%	27.97M	14/9
	convnext_small [39]	38.2%/27.6%	49.59M	14/9

Table 2: TConvNets Performance Results on CITYSCAPES [41], and PASCAL VOC [42]. The epoch# here is the epoch number when the lowest validation loss is recorded.

Datasets	*Model*	*Avg IoU*	*#Params*	*#Epochs*
CITYSCAPES [41]	resnet101+t_conv_net1	65.3%	2.9M	30
	resnet101+fcn [37]	64.1%	9.4M	24
	resnet101+deeplab3 [37]	64.7%	16.1M	33
PASCAL VOC [42]	resnet101+t_conv_net1	64.7%	2.9M	30
	resnet101+fcn [37]	64.6%	9.4M	30
	resnet101+deeplab3 [37]	65.4%	16.1M	18

4.3 Some Other Experiments

Although this paper primarily emphasizes the parameter efficiency of TConvNets, their distinctive structure may offer other notable advantages. We explore the superior performance of TConvNets over CapsNets and their enhanced adversarial robustness compared to ConvNets. For further details, please refer to Supplemental Material A.5 and Supplemental Material A.6.

4.4 Discussion

4.4.1 Why TConvNets Perform Better than ConvNets?

It is not accurate to claim that TConvNets outperform ConvNets in terms of parameter-wise efficiency since TConvNets are essentially a generalized version of ConvNets. Unlike many state-of-the-art neural architectures [43, 44, 45], which are fundamentally based on convolutional layers, TConvNets utilize tensor-based convolutions. Consequently, the techniques employed in these works can also be applied to TConvNets.

TConvNets provide a way of "organizing neurons hierarchically, " which is akin to the capsules in CapsNets and resembles similar structures found in human brains. Therefore, the tensor neurons in TConvNets can represent information hierarchically, allowing for better generalization and recognition of complex patterns. Essentially, CapsNets can be viewed as a fusion of convolutional layers with one or two matrix-based TConvNet layers along with a routing mechanism.

The evidence supporting the reduced susceptibility to overfitting in TConvNets becomes apparent when observing their comparatively narrower generalization gaps. To illustrate, both ResNet34 [16] and EfficientNet-b3 [17] can attain nearly 100% accuracy on the Tiny ImageNet training set, yet their accuracy on the validation set remains consistently at around 30%.

4.4.2 Limitations

One limitation of our study is that we did not compare TConvNets with the latest architectures [43, 44, 45]. One reason is that they fall into different technique slots. ConvNets are based on convolutions while TConvNets are based on tensor-based convolutions. It makes more sense to compare TConvNets with plain ConvNets. Our primary objective is to introduce a generalized form of ConvNets, rather than achieving state-of-the-art performance. It would be interesting to apply the techniques in [43, 44, 45] to TConvNets, and we plan to consider this in our future work.

Another limitation of our work is that our TConvNets models are much slower than ConvNets during training/inference. To illustrate, when we compare the forward pass times of a tensor-based convolutional layer and a traditional convolutional layer, both with identical parameter counts (3x3x27=243), on a GTX TITAN, we find average elapsed times of 876ms and 0.376ms respectively. Consequently, this paper primarily focuses on classification tasks with smaller datasets due to these slower training and inference speeds. This sluggish performance can be attributed to the lack of acceleration APIs for tensor-based operations, such as those found in CUDA/CUDNN. Theoretically, TConvNets should match ConvNets in training and inference speed when they possess similar sizes of parameters, as their primary difference lies in the number of parallel matrix multiplications (where conventional convolutions can be seen as a few large matrix multiplications, while tensor-based convolutions involve numerous small matrix multiplications).

5 Conclusions

We introduce Tensor-based Convolutional Neural Networks, TConvNets. Unlike ConvNets that use scalar neurons, TConvNets utilize tensor neurons instead. This involves extending the convolution operation from a linear combination of scalars to encompass tensor product and aggregation, making ConvNets a specialized case of TConvNets. To support TConvNets, we propose adjustments to batch normalization, initialization, and methods for structurizing and unstructurizing input/output. Additionally, we retain certain components from ConvNets, such as ReLU and cross-entropy loss. Through experiments, we demonstrate that TConvNets offer improved parameter efficiency when applied to classification tasks across different datasets. We expect that TConvNets hold promise in various other artificial intelligence tasks and datasets.

References

[1] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. 2017.

[2] Tailin Liang et al. "Pruning and quantization for deep neural network acceleration: A survey". In: *Neurocomputing* 461 (2021), pp. 370–403. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.07.045. URL: https://www.sciencedirect.com/science/article/pii/S0925231221010894.

[3] Zhuang Liu et al. "Learning Efficient Convolutional Networks through Network Slimming". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2755–2763. DOI: 10.1109/ICCV.2017.298.

[4] Wenlin Chen et al. "Compressing Neural Networks with the Hashing Trick". In: *Compressing Neural Networks with the Hashing Trick* (Apr. 2015).

[5] Yihui He et al. "AMC: AutoML for Model Compression and Acceleration on Mobile Devices". In: *European Conference on Computer Vision (ECCV)*. 2018.

[6] Tien-Ju Yang et al. "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications". In: *The European Conference on Computer Vision (ECCV)*. Sept. 2018.

[7] Shangqian Gao, Cheng Deng, and Heng Huang. "Cross Domain Model Compression by Structurally Weight Sharing". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[8] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. "Efficient Neural Network Compression". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[9] Song Han, Huizi Mao, and William Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: Oct. 2016.

[10] Jiaxiang Wu et al. "Quantized Convolutional Neural Networks for Mobile Devices". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4820–4828. DOI: 10.1109/CVPR.2016.521.

[11] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression". In: *Knowledge Discovery and Data Mining*. 2006. URL: https://api.semanticscholar.org/CorpusID:11253972.

[12] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic Routing Between Capsules". In: *CoRR* abs/1710.09829 (2017). arXiv: 1710.09829. URL: http://arxiv.org/abs/1710.09829.

[13] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. "Matrix capsules with EM routing". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=HJWLfGWRb.

[14] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". In: *arXiv:1602.07360* (2016).

[15] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015).

[17] Naman Agarwal et al. "Efficient Full-Matrix Adaptive Regularization". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 102–110. URL: http://proceedings.mlr.press/v97/agarwal19b.html.

[18] Xiangyu Zhang et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[19] Mingxing Tan et al. "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[20] Han Cai, Ligeng Zhu, and Song Han. "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware". In: *International Conference on Learning Representations*. 2019. URL: https://arxiv.org/pdf/1812.00332.pdf.

[21] Zheng Zhan et al. “Achieving On-Mobile Real-Time Super-Resolution With Neural Architecture and Pruning Search”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 4821–4831.

[22] Yann LeCun, Fu Jie Huang, and Léon Bottou. “Learning methods for generic object recognition with invariance to pose and lighting”. English (US). In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2004.

[23] Lucas Liebenwein et al. “Provable Filter Pruning for Efficient Neural Networks”. In: *International Conference on Learning Representations*. 2020.

[24] Huanrui Yang, Wei Wen, and Hai Li. “DeepHoyer: Learning Sparser Neural Network with Differentiable Scale-Invariant Sparsity Measures”. In: *International Conference on Learning Representations*. 2020.

[25] Shiyu Li et al. “PENNI: Pruned Kernel Sharing for Efficient CNN Inference”. In: *International Conference on Machine Learning*. 2020.

[26] Sidak Pal Singh and Dan Alistarh. “WoodFisher: Efficient Second-Order Approximation for Neural Network Compression”. In: *Conference on Neural Information Processing Systems*. 2020.

[27] Mart van Baalen et al. “Bayesian Bits: Unifying Quantization and Pruning”. In: *Conference on Neural Information Processing Systems*. 2020.

[28] Philip Blagoveschensky and Anh Huy Phan. “Deep convolutional tensor network”. In: *arXiv e-prints*, arXiv:2005.14506 (May 2020), arXiv:2005.14506. DOI: 10.48550/arXiv.2005.14506. arXiv: 2005.14506 [cs.LG].

[29] Rui Lin et al. “HOTCAKE: Higher Order Tucker Articulated Kernels for Deeper CNN Compression”. In: *2020 IEEE 15th International Conference on Solid-State Integrated Circuit Technology (ICSICT)*. 2020, pp. 1–4. DOI: 10.1109/ICSICT49897.2020.9278257.

[30] Yu Pan et al. “A Unified Weight Initialization Paradigm for Tensorial Convolutional Neural Networks.” In: *ICML*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 17238–17257. URL: http://dblp.uni-trier.de/db/conf/icml/icml2022.html#0005SLW0X22.

[31] Vadim Lebedev et al. “Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6553.

[32] Xavier Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Journal of Machine Learning Research - Proceedings Track* 9 (Jan. 2010), pp. 249–256.

[33] S. Ren K. He X. Zhang and J. Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *ICCV*. 2015.

[34] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-10 (Canadian Institute for Advanced Research)”. In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[35] Ya Le and Xuan S. Yang. “Tiny ImageNet Visual Recognition Challenge”. In: 2015.

[36] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[37] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[38] Ilija Radosavovic et al. “Designing Network Design Spaces”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[39] Zhuang Liu et al. “A ConvNet for the 2020s”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022).

[40] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[41] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[42] M. Everingham et al. “The Pascal Visual Object Classes Challenge: A Retrospective”. In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136.

[43] Yongming Rao et al. “HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 10353–10366. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/436d042b2dd81214d23ae43eb196b146-Paper-Conference.pdf.

[44] Siyuan Li et al. “MogaNet: Multi-order Gated Aggregation Network”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=XhYWgjqCrV.

[45] Weihao Yu et al. “MetaFormer Baselines for Vision”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.2 (2024), pp. 896–912. DOI: 10.1109/TPAMI.2023.3329173.

[46] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456. URL: https://proceedings.mlr.press/v37/ioffe15.html.

[47] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: (2015).

[48] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[49] Sai Samarth R. Phaye et al. “Dense and Diverse Capsule Networks: Making the Capsules Learn Better”. In: *CoRR* abs/1805.04001 (2018). arXiv: 1805.04001. URL: http://arxiv.org/abs/1805.04001.

[50] Jaewoong Choi et al. “Attention routing between capsules”. In: *CoRR* abs/1907.01750 (2019). arXiv: 1907.01750. URL: http://arxiv.org/abs/1907.01750.

[51] I. J. Goodfellow, J. Shlens, and C. Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *ArXiv e-prints* (Dec. 2014). arXiv: 1412.6572 [stat.ML].

A Supplemental Material

A.1 Batch Normalization for TConvNets

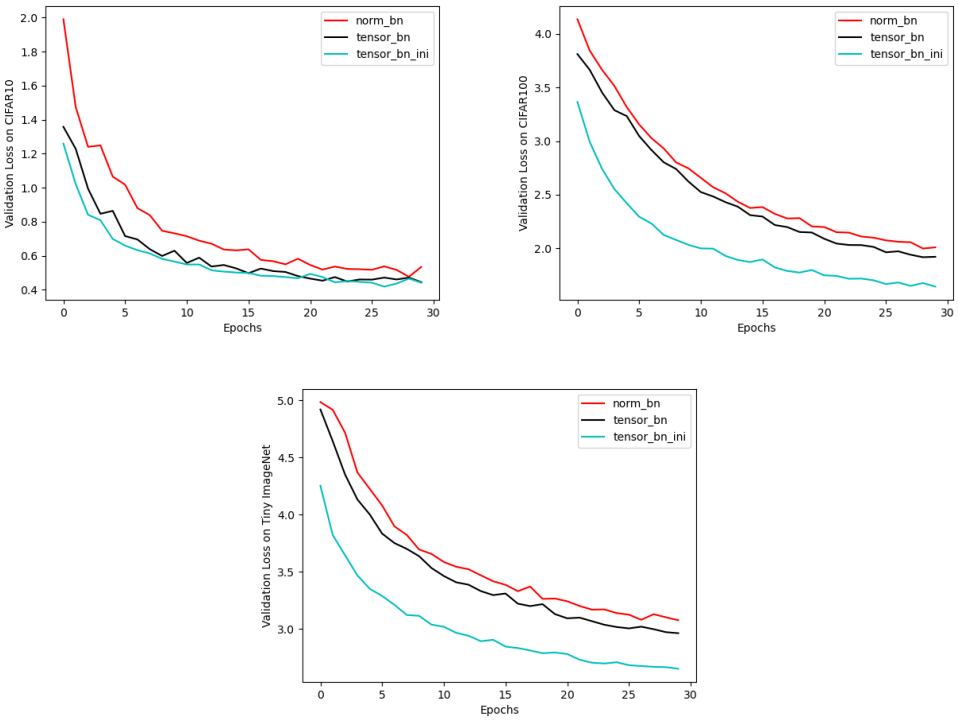


Figure 3: The convergence curve of t_conv_net (see Section 4) on CIFAR10, CIFAR100 and Tiny ImageNet. The red line represents the learning curve of t_conv_net with He Initialization [33] and normal batch normalization. The black line is the learning curve of t_conv_net with He Initialization [33] and the batch normalization in Section 3.1. The green line is the case when both the batch normalization in Section 3.1 and the initialization in Section A.2 are used.

Batch normalization is designed to address the “internal covariate shift" issue [46]. The primary idea involves normalizing each layer’s activations using a learned mean and standard deviation vector for each batch. Typically, batch normalization is applied in a channel-wise manner to preserve the independence across channels. In TConvNets, we apply batch normalization not only across channels but also across each structured feature dimension. In particular, for a layer with c input channels and d dimension structured features, $x = \left(x^1, x^2, \ldots, x^{c+d}\right)$

$$\hat{x}^i = \frac{x^i - \mathbb{E}\left[x^i\right]}{\sqrt{\mathrm{Var}\left[x^i\right]}} \tag{4}$$

We found that applying batch normalization across each structured feature dimension brings extra benefits, as Figure A.1 shows.

A.2 Initialization for TConvNets

Initialization sets appropriate initial weights for a neural work. A bad initialization may result in slow convergence or poor local minimum. When linear activations are assumed, [32] propose the Xavier initialization. On the other hand, [33] take ReLU activations into consideration and propose Kaiming initialization, which works better in deep neural networks. The key idea of both methods is keeping the variance of each layer unchanged to prevent the network from suffering gradients vanishing

or exploding. We follow the same idea and design a modified version of He Initialization [33] for TConvNets. Specifically, in each layer,

$$\mathbf{Y}_l = \mathbf{U}_l \bigotimes \mathbf{W}_l \tag{5}$$

Here, $\mathbf{W}$ is a $k^2 \times c \times \mathbf{T}$ tensor that represents the $k \times k$ kernel, c input channel, and the neuron tensor $\mathbf{T}$ respectively. $\mathbf{T} \in \mathbb{R}^d$, and $d = t_1 \times t_2 \dots t_n$. Please note that no bias is employed in this context. $\mathbf{Y}$ is the corresponding response. l is the layer index. Let f be the activation function,

$$\mathbf{U}_l = f(\mathbf{Y}_{l-1}) \tag{6}$$

where the default f is ReLU.

We assume both the neurons in $\mathbf{W}$ and features in $\mathbf{U}$ are mutually independent and share the same distribution. $\mathbf{W}$ and $\mathbf{U}$ are independent with each other. To simplify the computation, we also assume the tensor features or tensor neurons are vectors though they may have varied dimensions. Here we set $n_l = k^2 \times c$, The we have,

$$\mathrm{Cov}\left(Y_l\right) = n_l \mathrm{Cov}\left(U_l \bigotimes W_l\right) \tag{7}$$

where Y_l, U_l, and W_l represent the random vectors in $\mathbf{Y}_l$, $\mathbf{U}_l$ and $\mathbf{W}_l$ respectively. We let W_l's mean be a all-zero vector. Given $\mathbf{U}_l$ and $\mathbf{W}_l$ are independent with each other, then we have,

$$\mathrm{Cov}\left(Y_l\right) = n_l \mathrm{Cov}\left(W_l\right) \bigotimes \mathbb{E}\left(U_l U_l^T\right) \tag{8}$$

We assume both W_{l-1} and Y_{l-1} have a symmetric distribution around zero, then we have $\mathbb{E}\left[U_l U_l^T\right] = \frac{1}{2}\mathrm{Cov}\left[Y_{l-1}\right]$. Putting this into Equation 8, we have,

$$\mathrm{Cov}\left[Y_l\right] = \frac{1}{2} n_l \mathrm{Cov}\left[W_l\right] \mathrm{Cov}\left[Y_{l-1}\right] \tag{9}$$

We want to avoid exploding or vanishing gradients so identity matrix Covariance is a reasonable choice. Thus we have,

$$\frac{1}{2} n_l \mathrm{Cov}\left[W_l\right] = I_d, \quad \forall l \tag{10}$$

We assume the d variables within each tensor neuron are independent with each other, we thus only need to consider the d variances at the diagonal. Thus for each neuron in the neuron tensor,

$$\frac{1}{2} n_l d \mathrm{Var}\left[w_l\right] = 1, \quad \forall l \tag{11}$$

In conclusion, for a forward pass we should set a zero-mean Gaussian distribution whose standard deviation is $\sqrt{\frac{2}{n_l d}}$. Here we assume not using bias, so we should set bias as 0 for initialization. The backward case is similar. Here we assume ReLU is used. If there is a negative slope a, the standard deviation becomes $\sqrt{\frac{2}{n_l d(1+a^2)}}$. In this paper we use PReLU [33], a is set as 0.25. As Figure A.1 shows, the initialization approach for TConvNets yields superior performance compared to that for ConvNets.

A.3 The TConvNets' Structure

Figure 4 and Table 3 show the TConvNet model used in this paper.

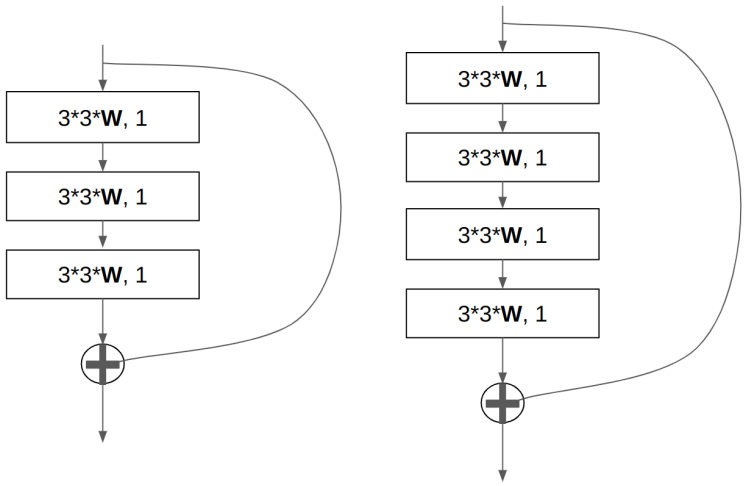


Figure 4: Two TConvNets residual structures. Left: the triple skips block, **Block#1** Right: the quadruple skips block, **Block#2**. $W \in \mathbb{R}^{k_1 \times k_2 \times k_3 \times k_4}$ is the neuron tensor. Each layer is followed by a PReLU [47] layer and a Batch Normalizaiton Layer [46].

Table 3: The structure of **g_conv_net**. The neuron tensor of *Layer#0* is $\mathbf{W}^{in} \in \mathbb{R}^{3\times1\times9\times9}$ that can transforms each input tensor $\mathbf{U}^{in} \in \mathbb{R}^{3\times1}$ to an output tensor $\mathbf{V} \in \mathbb{R}^{9\times9}$. *Layer#0* is followed by a BatchNorm Layer and a PReLU layer. So are the following layers. In the immediate layers, $\mathbf{W}^{in} \in \mathbb{R}^{9\times9\times9\times9}$ that can transforms each tensor $\mathbf{U}^{in} \in \mathbb{R}^{9\times9}$ to an output tensor $\mathbf{V} \in \mathbb{R}^{9\times9}$. In *Layer#4*, $\mathbf{W}^{f} \in \mathbb{R}^{4\times9\times9\times9\times9}$. In *Layer#6*, the final output number depends on the number of classes.

#Layer	*Neural Tensors*	*#Channel*
0	**Block#1**	1
1	**Block#2**	1
2	**Block#1**	1
3	**Block#2**	1
4	$3 \times 3 \times \mathbf{W}^{f}$	4
5	Pooling	4
6	$324 \times 10/100/200$	10/100/200

A.4 Tiny ImageNet [35]

Tiny ImageNet [35] is a subset of the ImageNet dataset [48], which contains 100,000 images of 200 classes (500 for each class) downsized to 64×64. The preprocessing pipeline encompasses the resizing of each image to dimensions of 80x80. Following this, random cropping is applied, resulting in a final size of 64x64. Additionally, horizontal flipping is incorporated for the purpose of data augmentation. Lastly, each channel of an image is normalized using mean values of (0.485, 0.456, 0.406) and standard deviations of (0.229, 0.224, 0.225).

Upon removing all preprocessing steps and retesting our model, we obtained results of 48.5% and 34.8% for the two respective settings. The utilization of data augmentation techniques leads to the highest performance for t_conv_net. Without augmentations, t_conv_net still achieves the second-best result with significantly fewer parameters compared to the best one.

A.5 Plain TConvNets versus CapsNets on MNIST

Given the resemblance between TConvNets and CapsNets, it's intriguing to compare TConvNets with various CapsNets variants. To conduct this comparison in a relatively impartial manner, we constructed a simple four-layer TConvNet without residual connections. We use a small neural tensor who has a shape of $\mathbf{T}^{in} \in \mathbb{R}^{2\times2\times2\times2}$. As Table 4 shows, both versions of TConvNets exhibit superior efficiency in terms of parameters compared to the CapsNets variations.

Table 4: Comparison between TConvNets and several CapsNets in terms of error rate on MNIST. The number in each routing type means the number of routing times.

Models	Routing	Error rate(%)	Param #
DCNet++ ([49])	Dynamic (-)	0.29	13.4M
DCNet ([49])	Dynamic (-)	0.25	11.8M
CapsNets ([12])	Dynamic (1)	$0.34_{\pm 0.03}$	6.8M
CapsNets ([12])	Dynamic (3)	$0.35_{\pm 0.04}$	6.8M
Atten-Caps ([50]	Attention (-)	0.64	$\approx$ 5.3M
CapsNets ([13])	EM (3)	0.44	320K
TConvNet	-	$0.32_{\pm 0.03}$	171K
TConvNet	-	$0.41_{\pm 0.05}$	22.2K

A.6 Comparison of TConvNets and ConvNets in Terms of Adversarial Robustness on the MNIST

TConvNets have demonstrated superior parameter efficiency, making it intriguing to explore how their structured neurons affect adversarial robustness. Here, we present a comparison of several TConvNet-based models and ConvNet-based models in terms of attack success rates. When comparing the robustness of models, we always take their sizes into account, as larger models generally tend to be more robust.

We use the FGSM [51] algorithm on the MNIST dataset, aiming to manipulate correctly predicted samples by applying varying levels of perturbations (with epsilon values ranging from 0.05 to 0.3) to the original images. No clipping algorithm is applied to the input after the perturbations are introduced.

The baseline ConvNets are all sourced from the PyTorch library [37]. In particular, they include shufflenet_v2_x0_5(351K) [18], MobileNet(2.2M) [15], regnet_y_400mf(3.9M) [38], efficientnet_b0_b0(4M) [17], and resnet18(11M) [16]. Since these models are designed for three-channel inputs, which is incompatible with MNIST, we modify the input channel to one and adjust the final layer to match MNIST's number of classes.

We designed three TConvNets: the first is a simple TConvNet with 4 layers and 179 thousand parameters. The other two are deeper models with 18 layers—one containing 558 thousand parameters and the other with 1.24 million parameters. Both of the deeper models have residual structures.

We make all three TConvNets attack the previously mentioned ConvNet models, and vice versa. The results are presented in Table 5, Table 6, and Table 7.

In all three tables, when epsilon equals 0, the diagonal elements represent the accuracy of each model without any perturbations. For example, in Table 5, the red number 99.08% (the first row, eps=0) indicates that the prediction accuracy of the TConvNet with 179 thousand parameters is 99.08%.

How do we determine which model is more robust? Take Table 5 as an example: when EfficientNet attacks the TConvNet at epsilon 0.15, the accuracy drops from 99.08% to 98.81% (third column, highlighted in red). In contrast, when the TConvNet attacks EfficientNet at the same epsilon, the accuracy decreases from 97.67% to 95.6% (seventh column, highlighted in green). We observe that as a defender, the TConvNet's accuracy is less impacted by the perturbations generated by EfficientNet. Meanwhile, as an attacker, the TConvNet can reduce EfficientNet's prediction accuracy by a larger margin. We can draw the same conclusion when we when testing the TConvNet models with 558K and 1.24M parameters, as shown in Table 6 and Table 7. Moreover, we can observe the same pattern for other models across varying epsilon values. Specifically, TConvNets demonstrate greater robustness than the ConvNets listed when defending and are more effective as attackers.

Table 5: Comparison of TConvNets (4 layers, 179K parameters) and ConvNets in terms of adversarial robustness on the MNIST. The epsilon ranges from 0.05 to 0.3.

	Models	*TconvNets*	*ShuffleNet*	*MobileNet*	*RegNet*	*EfficientNet*	*ResNet*
eps:0	TConvNets	99.08%	N/A	N/A	N/A	N/A	N/A
	ShuffleNet	N/A	97.02%	N/A	N/A	N/A	N/A
	MobileNet	N/A	N/A	96.74%	N/A	N/A	N/A
	RegNet	N/A	N/A	N/A	96.86%	N/A	N/A
	EfficientNet	N/A	N/A	N/A	N/A	97.67%	N/A
	ResNet	N/A	N/A	N/A	N/A	N/A	98.46%
eps:0.05	TConvNets	97.8%	96.61%	96.27%	96.56%	97.26%	98.15%
	ShuffleNet	99.05%	92.38%	N/A	N/A	%	N/A
	MobileNet	98.93%	N/A	89.71%	N/A	N/A	N/A
	RegNet	99%	N/A	N/A	93.18%	N/A	N/A
	EfficientNet	98.81%	N/A	N/A	N/A	92.82%	N/A
	ResNet	98.89%	N/A	N/A	N/A	N/A	96.84%
eps:0.1	TConvNets	94.88%	96.07%	95.95%	96.28%	96.61%	97.89%
	ShuffleNet	98.98%	84.54%	N/A	N/A	N/A	N/A
	MobileNet	98.76%	N/A	79.17%	N/A	N/A	N/A
	RegNet	98.84%	N/A	N/A	86.94%	N/A	N/A
	EfficientNet	98.58%	N/A	N/A	N/A	82.64%	N/A
	ResNet	98.64%	N/A	N/A	N/A	N/A	94.6%
eps:0.15	TConvNets	89.65%	95.53%	95.56%	95.83%	95.6%	97.47%
	ShuffleNet	98.87%	74.48%	N/A	N/A	N/A	N/A
	MobileNet	98.6%	N/A	65.9%	N/A	N/A	N/A
	RegNet	98.68%	N/A	N/A	77.68%	N/A	N/A
	EfficientNet	98.11%	N/A	N/A	N/A	65.17%	N/ A
	ResNet	98.28%	N/A	N/A	N/A	N/A	90.36%
eps:0.2	TConvNets	78.89%	94.78%	94.93%	95.52%	94.37%	97.08%
	ShuffleNet	98.76%	61.74%	N/A	N/A	N/A	N/A
	MobileNet	98.14%	N/A	51.89%	N/A	N/A	N/A
	RegNet	98.39%	N/A	N/A	65.43%	N/A	N/A
	EfficientNet	97.49%	N/A	N/A	N/A	42.97%	N/A
	ResNet	97.7%	N/A	N/A	N/A	N/A	83.4%
eps:0.25	TConvNets	63.59%	93.68%	94.04%	95.14%	92.85%	96.49%
	ShuffleNet	98.61%	48.82%	N/A	N/A	N/A	N/A
	MobileNet	97.46%	N/A	37.66%	N/A	N/A	N/A
	RegNet	97.92%	N/A	N/A	53.06%	N/A	N/A
	EfficientNet	96.47%	N/A	N/A	N/A	25.96%	N/A
	ResNet	97.03%	N/A	N/A	N/A	N/A	73.26%
eps:0.3	TConvNets	47.61%	91.93%	93.32%	94.63%	90.72%	95.86%
	ShuffleNet	98.38%	37.43%	N/A	N/A	N/A	N/A
	MobileNet	96.51%	N/A	25.79%	N/A	N/A	N/A
	RegNet	97.34%	N/A	N/A	42.36%	N/A	N/A
	EfficientNet	94.78%	N/A	N/A	N/A	14.14%	N/A
	ResNet	95.8%	N/A	N/A	N/A	N/A	59.98%

Table 6: Comparison of TConvNets (18 layers, 558K parameters) and ConvNets in terms of adversarial robustness on the MNIST. The epsilon ranges from 0.05 to 0.3.

	Models	*TconvNets*	*ShuffleNet*	*MobileNet*	*RegNet*	*EfficientNet*	*ResNet*
eps:0	TConvNets	98.53%	N/A	N/A	N/A	N/A	N/A
	ShuffleNet	N/A	97.02%	N/A	N/A	N/A	N/A
	MobileNet	N/A	N/A	96.74%	N/A	N/A	N/A
	RegNet	N/A	N/A	N/A	96.86%	N/A	N/A
	EfficientNet	N/A	N/A	N/A	N/A	97.67%	N/A
	ResNet	N/A	N/A	N/A	N/A	N/A	98.46%
eps:0.05	TConvNets	96.58%	97.41%	96.35%	96.61%	97.41%	98.11%
	ShuffleNet	98.36%	92.38%	N/A	N/A	N/A	N/A
	MobileNet	98.33%	N/A	89.71%	N/A	N/A	N/A
	RegNet	98.41%	N/A	N/A	93.18%	N/A	N/A
	EfficientNet	98.32%	N/A	N/A	N/A	92.82%	N/A
	ResNet	98.19%	N/A	N/A	N/A	N/A	96.84%
eps:0.1	TConvNets	93.17%	96.87%	95.74%	96.32%	96.87%	97.8%
	ShuffleNet	98.19%	84.54%	N/A	N/A	N/A	N/A
	MobileNet	98.13%	N/A	79.17%	N/A	N/A	N/A
	RegNet	98.22%	N/A	N/A	86.94%	N/A	N/A
	EfficientNet	98.01%	N/A	N/A	N/A	82.64%	N/A
	ResNet	97.8%	N/A	N/A	N/A	N/A	94.6%
eps:0.15	TConvNets	88.41%	96.04%	95.19%	96%	96.04%	97.44%
	ShuffleNet	98.01%	74.48%	N/A	N/A	N/A	N/A
	MobileNet	97.88%	N/A	65.9%	N/A	N/A	N/A
	RegNet	98%	N/A	N/A	77.68%	N/A	N/A
	EfficientNet	97.56%	N/A	N/A	N/A	65.17%	N/A
	ResNet	97.35%	N/A	N/A	N/A	N/A	90.36%
eps:0.2	TConvNets	80.42%	95.06%	94.47%	95.63%	95.06%	96.84%
	ShuffleNet	97.82%	61.74%	N/A	N/A	N/A	N/A
	MobileNet	97.46%	N/A	51.89%	N/A	N/A	N/A
	RegNet	97.71%	N/A	N/A	65.43%	N/A	N/A
	EfficientNet	96.95%	N/A	N/A	N/A	42.97%	N/A
	ResNet	96.47%	N/A	N/A	N/A	N/A	83.4%
eps:0.25	TConvNets	67.13%	93.68%	93.59%	95.31%	93.68%	96.28%
	ShuffleNet	97.57%	48.82%	N/A	N/A	N/A	N/A
	MobileNet	96.46%	N/A	37.66%	N/A	N/A	N/A
	RegNet	97.12%	N/A	N/A	53.06%	N/A	N/A
	EfficientNet	95.28%	N/A	N/A	N/A	25.96%	N/A
	ResNet	96.02%	N/A	N/A	N/A	N/A	73.26%
eps:0.3	TConvNets	49.5%	91.96%	92.62%	94.75%	91.96%	95.56%
	ShuffleNet	97.06%	37.43%	N/A	N/A	N/A	N/A
	MobileNet	94.52%	N/A	25.79%	N/A	N/A	N/A
	RegNet	95.87%	N/A	N/A	42.36%	N/A	N/A
	EfficientNet	87.78%	N/A	N/A	N/A	14.14%	N/A
	ResNet	92.21%	N/A	N/A	N/A	N/A	59.98%

Table 7: Comparison of TConvNets (18 layers, 1.24M parameters) and ConvNets in terms of adversarial robustness on the MNIST. The epsilon ranges from 0.05 to 0.3.

	Models	*TconvNets*	*ShuffleNet*	*MobileNet*	*RegNet*	*EfficientNet*	*ResNet*
eps:0	TConvNets	98.88%	N/A	N/A	N/A	N/A	N/A
	ShuffleNet	N/A	97.02%	N/A	N/A	N/A	N/A
	MobileNet	N/A	N/A	96.74%	N/A	N/A	N/A
	RegNet	N/A	N/A	N/A	96.86%	N/A	N/A
	EfficientNet	N/A	N/A	N/A	N/A	97.67%	N/A
	ResNet	N/A	N/A	N/A	N/A	N/A	98.46%
eps:0.05	TConvNets	97.63%	96.54%	96.27%	96.61%	97.39%	98.1%
	ShuffleNet	98.74%	92.38%	N/A	N/A	N/A	N/A
	MobileNet	98.67%	N/A	89.71%	N/A	N/A	N/A
	RegNet	98.74%	N/A	N/A	93.18%	N/A	N/A
	EfficientNet	98.63%	N/A	N/A	N/A	92.82%	N/A
	ResNet	98.61%	N/A	N/A	N/A	N/A	96.84%
eps:0.1	TConvNets	95.36%	96.1%	95.69%	96.19%	96.86%	97.66%
	ShuffleNet	98.63%	84.54%	N/A	N/A	N/A	N/A
	MobileNet	98.47%	N/A	79.17%	N/A	N/A	N/A
	RegNet	98.56%	N/A	N/A	86.94%	N/A	N/A
	EfficientNet	98.4%	N/A	N/A	N/A	82.64%	N/A
	ResNet	98.27%	N/A	N/A	N/A	N/A	94.6%
eps:0.15	TConvNets	90.61%	95.3%	94.96%	95.87%	96.38%	97.15%
	ShuffleNet	98.51%	74.48%	N/A	N/A	N/A	N/A
	MobileNet	98.24%	N/A	65.9%	N/A	N/A	N/A
	RegNet	98.41%	N/A	N/A	77.68%	N/A	N/A
	EfficientNet	98.07%	N/A	N/A	N/A	65.17%	N/ A
	ResNet	97.8%	N/A	N/A	N/A	N/A	90.36%
eps:0.2	TConvNets	82.91%	94.4%	94.2%	95.48%	95.59%	96.59%
	ShuffleNet	98.21%	61.74%	N/A	N/A	N/A	N/A
	MobileNet	97.89%	N/A	51.89%	N/A	N/A	N/A
	RegNet	98.15%	N/A	N/A	65.43%	N/A	N/A
	EfficientNet	97.57%	N/A	N/A	N/A	42.97%	N/A
	ResNet	97.2%	N/A	N/A	N/A	N/A	83.4%
eps:0.25	TConvNets	72.3%	93.1%	93.21%	95.01%	94.57%	95.87%
	ShuffleNet	98.04%	48.82%	N/A	N/A	N/A	N/A
	MobileNet	97.23%	N/A	37.66%	N/A	N/A	N/A
	RegNet	97.79%	N/A	N/A	53.06%	N/A	N/A
	EfficientNet	96.71%	N/A	N/A	N/A	25.96%	N/A
	ResNet	95.9%	N/A	N/A	N/A	N/A	73.26%
eps:0.3	TConvNets	58.97%	91.49%	92.01%	94.33%	93.08%	95.2%
	ShuffleNet	97.6%	37.43%	N/A	N/A	N/A	N/A
	MobileNet	95.92%	N/A	25.79%	N/A	N/A	N/A
	RegNet	97.09%	N/A	N/A	42.36%	N/A	N/A
	EfficientNet	94.29%	N/A	N/A	N/A	14.14%	N/A
	ResNet	93.45%	N/A	N/A	N/A	N/A	59.98%