Gated Channel Transformation for Visual Recognition

Zongxin Yang[1,2], Linchao Zhu[2], Yu Wu[1,2], and Yi Yang[2*]
[1] Baidu Research [2] ReLER, University of Technology Sydney
{zongxin.yang,yu.wu-3}@student.uts.edu.au, {linchao.zhu,yi.yang}@uts.edu.au

Abstract

In this work, we propose a generally applicable transformation unit for visual recognition with deep convolutional neural networks. This transformation explicitly models channel relationships with explainable control variables. These variables determine the neuron behaviors of competition or cooperation, and they are jointly optimized with the convolutional weight towards more accurate recognition. In Squeeze-and-Excitation (SE) Networks, the channel relationships are implicitly learned by fully connected layers, and the SE block is integrated at the block-level. We instead introduce a channel normalization layer to reduce the number of parameters and computational complexity. This lightweight layer incorporates a simple ℓ_2 normalization, enabling our transformation unit applicable to operator-level without much increase of additional parameters. Extensive experiments demonstrate the effectiveness of our unit with clear margins on many vision tasks, i.e.*, image classification on ImageNet, object detection and instance segmentation on COCO, video classification on Kinetics.*

1. Introduction

Convolutional Neural Networks (CNNs) have proven to be critical and robust in visual recognition tasks, such as image classification [18], detection [32], and segmentation [32]. Notably, a single convolutional layer operates only on a neighboring local context of each spatial position of a feature map, which could possibly lead to local ambiguities [35]. To relieve this problem, VGGNets [31] were proposed to construct deep CNNs, using a series of convolutional layers with non-linear activation functions and downsampling operators to cover a large extent of context. Moreover, [13] introduced a residual connection to help CNNs benefit from deeper architectures further.

Apart from improving the depth of CNNs, another branch of methods focuses on augmenting the convolutional layer with modules that directly operate on context across large neighborhoods. Squeeze-and-Excitation Networks (SE-Nets) [17] leveraged globally embedding information to model channel relationships and modulate feature maps on the channel-wise level. Moreover, its following method, GE-Nets [16], used largely neighboring embedding instead. These modules can be conveniently assembled into modern networks, such as ResNets [13] and Inception [33] networks, to improve the representational ability of networks.

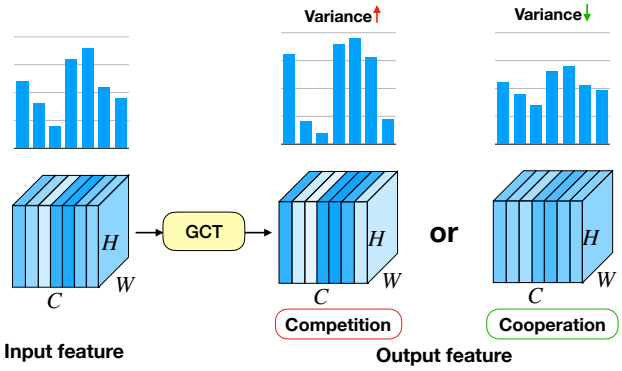


Figure 1: An illustration of the behavior of GCT. Combining normalization methods and gating mechanisms, GCT can create the channel relations of both competition (increasing the variance of channel activation) and cooperation (decreasing the variance of channel activation).

However, the SE module uses two fully connected (*FC*) layers to process channel-wise embeddings, which leads to two problems. First, the number of SE modules to be applied in CNNs is limited. In [17], SE module was applied at the block-level, *i.e.*, a single SE module is utilized per Res-block [13] or Inception-block [34]. The dimension of the *FC* layer is decreased to save the computational cost further. However, the designed *FC* layers still hinder the wide deployment of SE modules across all layers. Second, due to the complexity of the parameters in *FC* (or convolutional layer in GE), it is difficult to analyze the interactions among the channels at different layers. The channel relationships learned by convolution and *FC* operations are inherently implicit [17], resulting in agnostic behaviors of

*This work was done when Zongxin Yang and Yu Wu interned at Baidu Research. Yi Yang is the corresponding author.

the neuron outputs.

In this paper, we propose a Gated Channel Transformation (GCT) for efficient and accurate contextual information modeling. First, we use a normalization component, instead of *FC*, to model channel relations. Normalization methods, *e.g.*, Local Response Normalization (LRN) [23], can create competitions among different neurons (or channels) in neural networks. Batch normalization [19] and its variants can smooth gradient and have been widely used in accelerating CNNs training process. We leverage a simple ℓ_2 normalization for modeling channel relationship, which is more stable and computationally efficient comparing to *FC* layers. Second, we carefully design the trainable architecture of GCT based on the normalization component and introduce a few channel-wise parameters to control the behavior of the gated adaptation of feature channels. Compared to the large number of parameters in *FC*, our designed parameters are much more lightweight. Moreover, the channel-wise gating weight parameter is convenient for channel relationship analysis and helps understand the effect of GCT modules across a whole CNN network.

Following SE, our GCT employs gating mechanisms to adapt the channel relationships. However, the $Sigmoid$ activation of SE is easy to cause vanishing gradient in training, when the $Sigmoid$ value is close to either 0 or 1. To relieve this problem, we introduce the residual connection [13] into the gating adaptation by using a $1+tanh(x)$ gate activation, which gives GCT an ability to model identity mapping and makes the training process more stable. Combining normalization methods and gating mechanisms, GCT can create the channel relations of both competition and cooperation, as shown in Fig. 1. According to our visualization analysis (Sec. 4.4), GCT prefers to encourage cooperation in shallower layers, but competition is enhanced in deeper layers. Generally, the shallow layers learn low-level attributes to capture general characteristics like textures. In deeper layers, the high-level features are more discriminative and task-related.

Our experiments show that GCT is a simple and effective architecture for modeling relationships among channels. It significantly improves the generalization capability of deep convolutional networks across visual recognition tasks and datasets.

2. Related Work

Gating and attention mechanisms. Gating mechanisms have been successfully deployed in some recurrent neural network architectures. Long Short-Term Memory (LSTM) [15] introduced an input gate, output gate and forget gate, which are used to regulate the flow of information into and out of the module. Based on gating mechanisms, some attention methods focus on forcing computational resources towards the most informative components of features [24, 27]. The attention mechanism has achieved promising improvements across many tasks including sequence learning [4], lip reading [7], image captioning [42], localization and understanding in images [20, 5].

Recent works introduce the attention mechanism into convolutional networks (*e.g.*, [10, 9]). A non-recurrent approach to combine gating mechanisms with convolutional networks achieves promising performance in the language task, which was always studied based on recurrent networks before [9]. Following these studies, SE-Nets [17] and its following work GE-Nets [16] introduced a lightweight gating mechanism that focuses on enhancing the representational power of the convolutional network by modeling channel-wise relationship. Compared to the SE module, our GCT also pays attention to the cross-channel relationship but can achieve better performance gains with less computation and parameters.

Normalization layers. In recent years, normalization layers have been widely used in deep networks to create competition between neurons [23] and produce smoother optimization surfaces [19]. Local Response Normalization (LRN) [23] computes the statistics in a small neighborhood among channels for each pixel. Batch Normalization (BN) [19] utilizes global spatial information along the batch dimension and suggests to be deployed for all layers. Layer Normalization (LN) [3] computes along the channel dimension instead of the batch dimension. Group Normalization (GN) [40] differently divides the channels into groups and computes within each group the mean and variance for normalization. Similar to LRN, GN and LN, our GCT also utilizes channel-related information with normalization structure.

Deep architectures. VGGNets [31] and Inception networks [33] demonstrated that it was significant to improve the quality of representation by increasing the depth of a network. ResNets [13] utilized shortcut connections to identity-based skip connections, and proved that it was highly effective to build considerably deeper and stronger networks with them. Some other researchers focused on improving the representation ability of the computational elements contained within a network [34]. The more diverse composition of operators within a computational element can be constructed with multi-branch convolutions or pooling layers. Other than this, grouped convolutions have proven to be a practical method to increase the cardinality of learned transformations [41].

Based on the success of CNNs for image tasks, 3D convolutions [21, 36] (C3D) are introduced for video classification task. In addition to C3D, Non-local neural networks [39] (NL-Nets) design a non-local operation for capturing long-range, non-local dependency, which significantly improves the accuracy of video classification.

We build our GCT on some of these deep architectures.

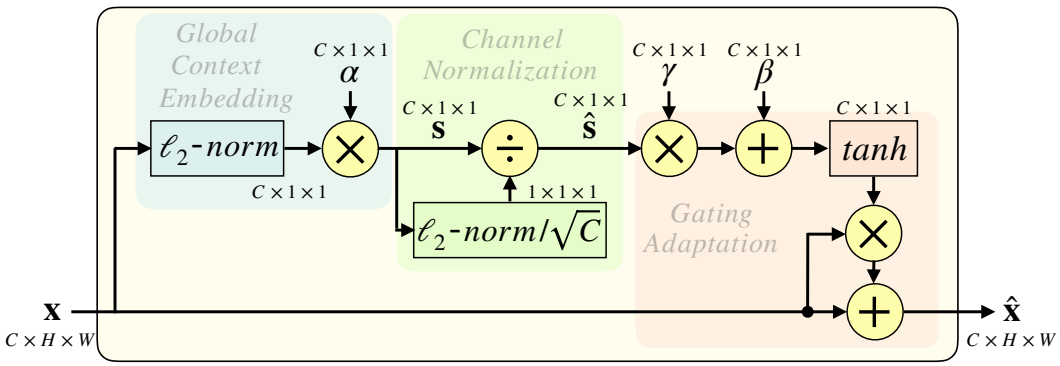


Figure 2: An **overview** of the structure of Gated Channel Transformation (GCT). The embedding weight, $\boldsymbol{\alpha}$, is responsible for controlling the weight of each channel before the channel normalization. And the gating weight and bias, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, are responsible for adjusting the scale of the input feature $\boldsymbol{x}$ channel-wisely.

All the networks with GCT achieve promising performance improvements, but the growth of computational complexity is negligible.

3. Gated Channel Transformation

We propose a Gated Channel Transformation for highly efficient, channel-wise, contextual information modeling. GCT employs a normalization method to create competition or cooperation relationships among channels. Notably, the normalization operation is parameter-free. To make GCT learnable, we design a global context embedding operator, which embeds the global context and controls the weight fo each channel before the normalization and a gating adaptation operator, which adjusts the input feature channel-wisely based on the output of the normalization. The channel-wise trainable parameters are light-weight yet effective and make GCT convenient to be extensively deployed while occupying a small number of parameters. Besides, the parameters of the gating adaptation operator are easy and intuitive to be visualized for explaining the behavior of GCT. In summary, we carefully design the highly light-weight, explainable, but effective architecture of GCT based on the normalization operation for modeling channel relationships.

Let $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$ be an activation feature in a convolutional network, where H and W are the spatial height and width, and C is the number of channels. In general, GCT performs the following transformation:

$$\hat{\mathbf{x}} = F(\mathbf{x}|\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\beta}), \boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^C. \tag{1}$$

Here $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are trainable parameters. Embedding weight $\boldsymbol{\alpha}$ is responsible for adapting the embedding outputs. The gating weight $\boldsymbol{\gamma}$ and bias $\boldsymbol{\beta}$ control the activation of the gate. They determine the behavior of GCT in each channel. Notably, the parameter complexity of GCT is $O(C)$, which is smaller than the SE module ($O(C^2)$) [17]. In SENet, two *FC* layers are leveraged, which have the parameter complexity of $O(C^2)$.

An illustration of the structure of GCT is shown in Fig. 2. Let $\mathbf{x} = [x_1, x_2, ..., x_C], x_c = [x_c^{i,j}]_{H \times W} \in \mathbb{R}^{H \times W}, c \in \{1, 2, ..., C\}$, where x_c is corresponding to each channel of $\mathbf{x}$. The detailed transformation consists of following parts.

3.1. Global Context Embedding

The information with a large receptive field is useful to avoid local ambiguities [35, 16] caused by the information with a small receptive field (*e.g.*, a convolutional layer). Hence, we firstly design a global context embedding module to aggregate global context information in each channel. The module can exploit global contextual information outside the small receptive fields of convolutional layers. Given the embedding weight $\boldsymbol{\alpha} = [\alpha_1, ..., \alpha_C]$, the module is defined as:

$$s_c = \alpha_c ||x_c||_2 = \alpha_c \{[\sum_{i=1}^{H} \sum_{j=1}^{W} (x_c^{i,j})^2] + \epsilon\}^{\frac{1}{2}}, \tag{2}$$

where ϵ is a small constant to avoid the problem of derivation at the zero point. Different from SE, GCT does not use global average pooling (GAP) to aggregate channel context. GAP might fail in some extreme cases. For example, if SE is deployed after the Instance Normalization [37] layer that is popular in style transfer task, the output of GAP will be constant for any inputs since IN fixes the mean of each channel of features. To avoid this problem, we choose ℓ_p-norm instead. It is worth noting that GCT is robust with different ℓ_p-norms. In Sec. 4.5, we compare the performance of some popular ℓ_p-norms and choose the best one, ℓ_2-norm, to be our default setting. Notably, the performance of ℓ_1-norm is very close to ℓ_2-norm, but ℓ_1-norm can be equivalently replaced by GAP when the input of GCT is

consistently non-negative (for example, after ReLU activation in our default setting). In this case, ℓ_1-norm is more computationally efficient as shown in Table 3.

Besides, we use trainable parameters, α_c, to control the weight of each channel because different channels should have different significance. Especially if α_c is close to 0, the channel c will not be involved in the channel normalization. In other words, the gating weight, $\boldsymbol{\alpha}$, make GCT capable of learning the situation that one channel is individual to other channels.

3.2. Channel Normalization

Normalization methods can create competition relationship between neurons (or channels) [23] with lightweight computing resource and a stable training performance (*e.g.*, [19]). Similar to LRN, we use a ℓ_2 normalization to operate across channels, namely channel normalization. Let $\mathbf{s} = [s_1, ..., s_C]$, the formula of channel normalization is:

$$\hat{s}_c = \frac{\sqrt{C} s_c}{||\mathbf{s}||_2} = \frac{\sqrt{C} s_c}{[(\sum_{c=1}^{C} s_c^2) + \epsilon]^{\frac{1}{2}}}, \tag{3}$$

where ϵ is a small constant. The scalar $\sqrt{C}$ is used to normalize the scale of $\hat{s}_c$, avoiding a too small scale of $\hat{s}_c$ when C is large. Compared to the *FC* layers used by SE, our channel normalization has less computational complexity ($O(C)$) compared to the *FC* layers ($O(C^2)$).

3.3. Gating Adaptation

We employ a gating mechanism, namely gating adaptation, to adapt the original feature. By introducing the gating mechanism, our GCT can facilitate both competition and cooperation during the training process. Let the gating weight $\boldsymbol{\gamma} = [\gamma_1, ..., \gamma_C]$ and the gating biases $\boldsymbol{\beta} = [\beta_1, ..., \beta_C]$, we design the following gating function:

$$\hat{x}_c = x_c[1 + \tanh(\gamma_c \hat{s}_c + \beta_c)]. \tag{4}$$

The scale of each original channel x_c will be adapted by its corresponding gate, *i.e.*, $1 + \tanh(\gamma_c \hat{s}_c + \beta_c)$. Due to that the channel normalization is parameter-free, we design the trainable weight and bias, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, for learning to control the activation of gate channel-wisely. LRN benefits from only the competitions among the neurons [23]. However, GCT is able to model more types of relationship (*i.e.*, competition and cooperation) among different channels by combining normalization methods and gating mechanisms. When the gating weight of one channel (γ_c) is activated positively, GCT promotes this channel to compete with the others as in LRN. When the gating weight is activated negatively, GCT encourages the channel to cooperate with the others. We further analyze these adaptive channel relationships in Sec.4.4.

```
def forward(self, x, epsilon=1e-5):
  # x: input features with shape [N,C,H,W]
  # alpha, gamma, beta: embedding weight, gating
      weight, gating bias with shape [1,C,1,1]
  embedding = (x.pow(2).sum((2,3), keepdim=True)
      + epsilon).pow(0.5) * self.alpha
  norm = self.gamma /
      (embedding.pow(2).mean(dim=1,
      keepdim=True) + epsilon).pow(0.5)
  gate = 1. + torch.tanh(embedding * norm +
      self.beta)
  return x * gate
```

Figure 3: An implementation of GCT (ℓ_2) based on PyTorch

Besides, this gate function allows original features to pass to the next layer when the gating weight and biases are zeros, which is

$$\hat{\mathbf{x}} = F(\mathbf{x}|\boldsymbol{\alpha}, \mathbf{0}, \mathbf{0}) = \mathbf{1}\mathbf{x} = \mathbf{x}. \tag{5}$$

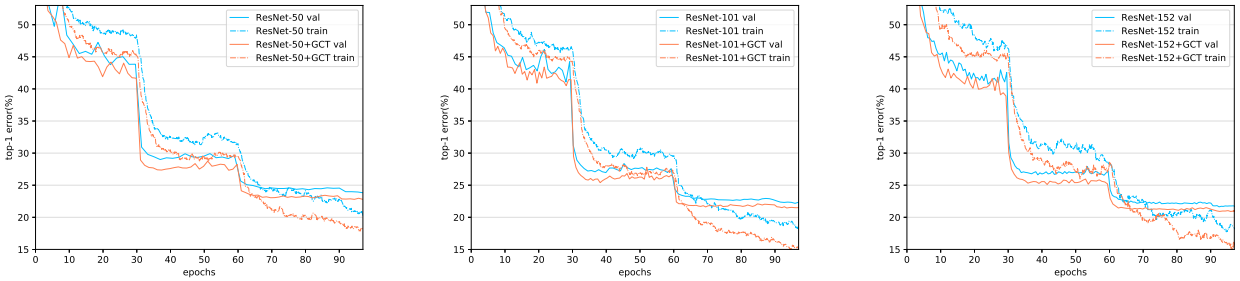
The ability to model identity mapping can effectively improve the robustness of the degradation problem in deep networks. ResNets also benefit from this idea. Therefore, we propose to initialize $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ to 0 in the initialization of GCT layers. By doing this, the initial steps of the training process will be more stable, and the final performance of GCT will be better.

3.4. Learning

Similar to modern normalization layers (*e.g.*, BN), we propose to apply GCT for all convolutional layers in deep networks. However, there are many different points nearby one convolutional layer to employ GCT. In deep networks, each convolutional layer always works together with a normalization layer (*e.g.*, BN) and an activation layer (*e.g.*, ReLU [28]). For this reason, there are three possible points to deploy the GCT layer, which are before the convolutional layer, before the normalization layer, and after the normalization layer. All these methods are effective, but we find it to be better to employ GCT before the convolutional layer. In Sec. 4.5, we compare the performance of these three application methods.

We implement and evaluate our GCT module on some popular deep learning frameworks, including PaddlePaddle [1], TensorFlow [2] and PyTorch [29], and we observe similar improvement by introducing GCT. Fig. 3 shows a simple implementation based on PyTorch.

In the training process, we propose to use 1 to initialize $\boldsymbol{\alpha}$ and use 0 to initialize all $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$. By doing this, GCT will be initialized as an identity mapping module, which will make the training process more stable. Besides, to avoid the bad influence of unstable gradient on the GCT gate in initial training steps, we propose to use warmup method (to start training with a small learning rate). In all the experiments



(a) ResNet-50 (b) ResNet-101 (c) ResNet-152

Figure 4: **Training curve comparisons** for ResNets with different depth on ImageNet.

	original		GCT	
Network	top-1	top-5	top-1	top-5
VGG-16 [31]	26.2	8.3	**25.1**$_{(1.1)}$	**7.5**$_{(0.8)}$
Inception-v3 [34]	24.3	7.3	**23.7**$_{(0.6)}$	**7.1**$_{(0.2)}$
ResNeXt-50 [41]	22.4	6.3	**21.7**$_{(0.7)}$	**6.0**$_{(0.3)}$
ResNet-50 [13]	23.8	7.0	**22.7**$_{(1.1)}$	**6.3**$_{(0.7)}$
ResNet-101 [13]	22.2	6.2	**21.4**$_{(0.8)}$	**5.9**$_{(0.3)}$
ResNet-152 [13]	21.6	5.9	**20.8**$_{(0.8)}$	**5.5**$_{(0.4)}$
ResNet-200* [14]	20.7	5.2	**19.7**$_{(1.0)}$	**4.8**$_{(0.4)}$

Table 1: **Improvement in error performance (%) on ImageNet.** The numbers in brackets denote the improvement in performance over the baselines. ResNet-200* means we follow the strategy in [14] to train this model on 224×224 but evaluate on 320×320.

on ImageNet [30], we start training with a learning rate of 0.01 for 1 epoch. After the warmup, we go back to the original learning rate schedule. Finally, we propose **NOT** to apply weight decay on β parameters, which is possible to reduce the performance of GCT.

4. Experiments

We apply GCT for all the convolutional layers in deep networks rather than block-level deployment in SE-Net. In all GCT counterparts, we employ one GCT layer before each convolutional layer. In the Kinetics experiments, we apply GCT at the last two convolutional layers in each ResBlock. More training details are shown in Sec. 3.4.

4.1. ImageNet

We experiment on the ImageNet 2012 dataset [30] with $1,000$ classes. We train all the models on the 1.28M training images and evaluate on the $50,000$ validation images.

Training details. In the training process of all the models, the input image is 224×224 randomly cropped from a resized image using the same augmentation in [33]. We use SGD with a mini-batch size of 256. For ResNet-152 and ResNeXt-50, we use half mini-batch size and double the training steps). The weight decay is 0.0001, and the momentum is 0.9. The base learning rate is 0.1, and we divide it by 10 every 30 epochs. All models are trained for 100 epochs from scratch, using the weight initialization strategy described in [12]. Besides, we start the training process with a learning rate of 0.01 for 1 epoch. After the warmup, we go back to the original learning rate schedule. In all comparisons, we evaluate the error on the single 224×224 center crop from an image whose shorter side is 256. For ResNet-200 [14], we evaluate on 320×320 following [14].

Integration with deep modern architectures. We study the effects of integrating GCT layers with some state-of-the-art backbone architectures, *e.g.*, ResNets and ResNeXts [41], in which we apply GCT before all the convolutional layers. We report all these results in Table 1. Compared to original architectures, we observe significant performance improvements by introducing GCT into networks. Particularly, the top-1 error of GCT-ResNet-101 is $\mathbf{21}.4\%$, which is even better than the ResNet-152 baseline (21.6%) with a deeper network and much more parameters. In addition, GCT is able to bring stable improvement in ResNets with different depth ($\mathbf{1}.\mathbf{1}\%$ top-1 improvement in ResNet-50, $\mathbf{0}.\mathbf{8}\%$ in ResNet-152 and $\mathbf{1}.\mathbf{0}\%$ in ResNet-200). Besides, we observe a smooth improvement throughout the training schedule, which is shown in Fig.4.

We also explore the improvement with GCT in *non-residual* networks (*e.g.*, VGG-16 [31] and Inception-v3 [34]). To stabilize the training process, we employ BN [19] layers after every convolutional layer. Similar to the effectiveness in residual architectures, GCT layers bring promising improvements in *non-residual* structures.

Compared to SE. Following [16], we conduct experiments on ImageNet to compare SE with GCT. In addition, we make comparison fairly in both residual and *non-residual* networks and the results are reported in Table 2. We follow the methods in [17] to integrate SE into VGG-16 [31], Inception [34], ResNet-50 [13] and ResNeXt-50 [41] and train these models in same training schedule. Compared to

	original		+SE [17]		**+GCT (ours)**	
Network	top-1/5	G/P	top-1/5	G/P	top-1/5	G/P
ResNet-50 [13]	23.8/7.0	3.879/25.61	22.9/6.6	**3.893***/28.14	**22.7/6.3**	3.900/**25.68**
ResNeXt-50 [41]	22.4/6.3	3.795/25.10	22.0/6.1	**3.809***/27.63	**21.7/6.0**	3.821/**25.19**
Inception-v3 [34]	24.3/7.3	2.847/23.87	24.0/7.2	**2.851***/25.53	**23.7/7.1**	2.862/**23.99**
VGG-16 [31]	26.2/8.3	15.497/138.37	25.2/7.7	15.525/138.60	**25.1/7.5**	**15.516/138.38**

Table 2: **Compared to SE on ImageNet.** We evaluate the models of error performance (%), GFLOPs (G) and parameters (M). G/P means GFLOPs/parameters. *: In the first three networks, SE is only employed in block-level (Res-Block or Inception-Block) as proposed [17], but GCT is applied for all the convolutional layers. This difference makes that SE uses comparable GFLOPs with GCT. In VGG-16, however, SE is employed for all the convolutional layers in the experiments, which is the same as GCT. Under the same setting, GCT outperforms SE on both complexity and performance.

	original	+SE [17]	+GCT (ℓ_1-norm)	+GCT (ℓ_2-norm)
ResNet-50 [31]	603	**525***	484	425
VGG-16 [13]	313	183	**281**	268

Table 3: **Inference speed (FPS) comparison**. We compare the speed on ImageNet by using one GTX 1080 Ti GPU. GCT is always applied for all the convolutional layers. *: In ResNet-50, SE is applied for each Res-Block as proposed. But, in VGG-16, SE is applied for all the convolutional layers, which leads to better fairness in the same setting with GCT.

SE, GCT always achieves better improvement.

In order to compare computational complexity, we calculate the GFLOPs and the number of parameters. In VGG-16 experiments, SE is employed for all the convolutional layers, which is the same as GCT. Under this fair condition, GCT achieves better performance with less increase in both GFLOPs (**0.019**G *vs*. 0.028G) and parameters (**0.01**M *vs*. 0.23M). Moreover, GCT is much more efficient than SE in run-time as shown in Table 3 (**281**FPS *vs*. 183). In other experiments, SE is employed in block-level (Res-Block or Inception-Block) as proposed [17], which means the number of SE modules is only about $1/3$ of GCT. However, the increase in parameters of GCT is still much less than SE, and the inference speed of GCT is comparable with SE (GCT 484FPS *vs*. SE 525 in ResNet-50). Compared to SE, GCT performs better and is capable of applying for all the convolutional layers while keeping the network efficient.

4.2. COCO

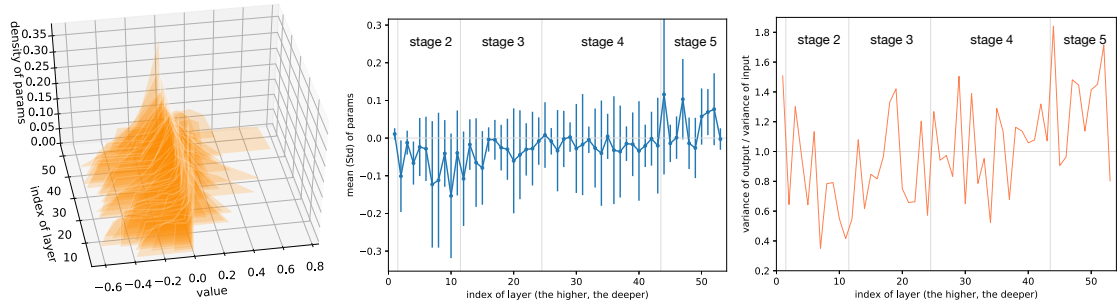
Next we evaluate the generalizability on the COCO dataset [26]. We train the models on the COCO *train2017* set and evaluate on the COCO *eval2017* set (a.k.a *minival*).

Training details. We experiment on the Mask R-CNN baselines [11] and its GN counterparts [40]. All the backbone models are pre-trained on ImageNet using the scale and aspect ratio augmentation in [33] and fine-tune on COCO with a batch size of 16 (2 images/GPU). Besides, all these experiments use the Feature Pyramid Network (FPN) [25]. We also use the same hyperparameters and two training schedules used in [40]. The short schedule includes 90K iterations, in which the learning rate is divided by 10 at 60K and 80K iterations. The long schedule increases the iterations to 270K, in which the learning rate is divided by 10 at 210K and 250K. The base learning rate is 0.02 in both schedules.

Backbone	box head	box AP	mask AP
ResNet-50 BN*	-	37.8	34.2
ResNet-50 BN*+SE [6]	-	$38.2_{(0.4)}$	$34.7_{(0.5)}$
ResNet-50 BN*+**GCT**	-	$\mathbf{39.8}_{(2.0)}$	$\mathbf{36.0}_{(1.8)}$
ResNet-101 BN*	-	40.1	36.1
ResNet-101 BN*+**GCT**	-	$\mathbf{42.0}_{(1.9)}$	$\mathbf{37.7}_{(1.6)}$
$^{+}$ResNet-50 BN*	-	38.6	34.5
$^{+}$ResNet-50 GN	GN	$40.8_{(2.2)}$	$36.1_{(1.6)}$
$^{+}$ResNet-50 BN*+**GCT**	GN	$\mathbf{41.6}_{(3.0)}$	$\mathbf{37.1}_{(2.6)}$
$^{+}$ResNet-50 BN*+**GCT**	GN+**GCT**	$\mathbf{41.8}_{(3.2)}$	$\mathbf{37.3}_{(2.8)}$
$^{+}$ResNet-101 BN*	-	40.9	36.4
$^{+}$ResNet-101 GN	GN	$42.3_{(1.4)}$	$37.2_{(0.8)}$
$^{+}$ResNet-101 BN*+**GCT**	GN	$\mathbf{43.1}_{(2.2)}$	$\mathbf{38.3}_{(1.9)}$

Table 4: **Improvement on COCO with Mask R-CNN framework [11].** BN* means BN is frozen. $^{+}$ means increasing the training iterations from 90K to 270K. When using GN, we follow the original strategy in [40].

Improvements on Mask R-CNN [11]. Table 4 shows the comparison of BN* (frozen BN), GN and BN*+GCT (using GCT before all the convolutional layers of the backbones). First, we use a short training schedule to compare baselines and GCT counterparts. GCT shows stable and significant improvement in both ResNet-50 and ResNet-101. In ResNet-50, GCT improves box AP by $\mathbf{2.0}$ and mask AP by $\mathbf{1.8}$, which significantly outperforms SE (0.4 box AP / 0.5 mask AP). Moreover, in ResNet-101, GCT also improves detection AP by $\mathbf{1.9}$ and segmentation AP by $\mathbf{1.6}$. Then, we use the long schedule to compare GN and BN*+GCT. GN is more effective than BN when batch size is small as



(a) distribution of γ (b) mean (std) of γ (c) ratio of the variance of output and input

Figure 5: **Analysis**. The visulization of parameters of γ (Fig.(a), (b)), and the ratio of variance of GCT output and input feature (Fig.(c)) in all the GCT layers in ResNet-50 on ImageNet.

Backbone	NL-Net [39]	**+GCT**
ResNet-50	74.6	**75.1**$_{(0.5)}$
ResNet-101	75.7	**76.2**$_{(0.5)}$

Table 5: **Improvement in top-1 accuracy (%) over the state-of-the-art method on Kinetics.**

in this case of detection and segmentation using Mask R-CNN. However, we deploy GCT together with BN into the backbone, and these BN*+GCT counterparts achieve much better performance than GN backbones. Compared to GN in ResNet-101, BN*+GCT improves detection AP by **0.8** and segmentation AP by **1.1**. In particular, ResNet-101 with BN*+GCT trained in the short schedule achieves better segmentation AP (**37.7**) than the GN counterpart (37.2) trained with the long schedule. This GN counterpart also uses GN in the backbone, the box heads, and the FPN. We also explore to combine GCT with GN by introducing GCT into GN box head. The results show GN+GCT achieves a better performance. It demonstrates the benefits of integrating GCT with GN. We now have shown the effectiveness of GCT in working with both BN and GN.

4.3. Kinetics

Our previous experiments demonstrate the effectiveness of GCT on image-related tasks. We now evaluate the generalizability in video understanding task of action recognition on a large scale dataset, Kinetics-400 [22]. We employ the ResNet-50 (3D) and ResNet-101 (3D) as the backbone and apply GCT in the last two convolutional layers in each Res-Block. The backbone networks are pre-trained on ImageNet [30].

We compare with the state-of-the-art Non-Local Networks (NL-Net) [39]. The results show that GCT counterparts consistently improves the recognition accuracy over both the ResNet-50 and ResNet-101 baselines, as shown in Table 5. Because of our limited memory resource, we can **NOT** apply GCT in all the convolutional layers, which we believe can further improve the performance.

In summary, extensive experiments demonstrate that GCT is effective across a wide range of modern architectures, visual tasks, and datasets.

4.4. Analysis

To analyze the behavior of GCT in different layers, we visualize the distribution of the gating weight (γ) of each GCT layer in ResNet-50 on ImageNet. Further, we sort these distributions according to their layer index in 3D space (Fig. 5a). The bigger layer index means it is closer to the network output. To make the visualization clearer, we re-scale the vertical z axis with $log(1+z)$, which corresponds to the percentage density of γ. We also calculate the mean and standard deviation (std) of γ in each layer and show them in a bar chart (Fig. 5b). As shown in Fig. 5a and 5b, the mean of γ tends to be less than 0 in the GCT layers far from the network output. Oppositely, in the layers close to the output, the mean tends to be greater than 0.

According to Eq. 3 & 4, the adaptation of channel x_c is related to $\hat{s}_c$, which corresponds to the ratio of the weighted ℓ_2-norm of x_c (*i.e.*, s_c) and the average of all the s_c. When the gating weight γ_c is greater than 0, the adaptation is positively correlated to $\hat{s}_c$ and increases the variance between x_c and others; When γ_c is lower than 0, the adaptation is negatively correlated and reduces the variance.

Based on the analysis and the results we observe, we suppose that GCT tends to reduce the difference among channels in layers far away from the output. This behavior is helpful to encourage cooperation among channels and relieve overfitting. Apart from this, GCT tends to increase the difference among channels when close to the output. Here, GCT acts like attention mechanisms that focus on creating competition.

To further validate our hypothesis, we calculate the ratio of the variance of output and input feature of each GCT layer, which we show in Fig. 5c. Generally, the shallow con-

Norm	top-1	top-5
ℓ_∞	23.1	6.7
ℓ_1	22.8	6.3
ℓ_2	**22.7**	**6.3**

(a) Embedding operator.

Normalization	top-1	top-5
M & V	23.7	7.1
ℓ_1	22.9	6.4
ℓ_2	**22.7**	**6.3**

(b) Normalization operator.

Adaptation	top-1	top-5
$Sigmoid$	22.9	6.5
$1 + ELU$	22.7	6.4
$1 + tanh$	**22.7**	**6.3**

(c) Adaptation operator.

Position	top-1	top-5
after BN	23.1	6.6
before BN	23.1	6.5
before Conv	**22.7**	**6.3**

(d) Application position.

Table 6: **Ablation experiments.** We evaluate error performance in GCT-ResNet-50 on ImageNet (%). The ResNet-50 baseline achieves a top-1 of 23.8 and a top-5 of 7.0. M & V denotes the mean and variance normalization.

volutional layers learn low-level attributes to capture general characteristics like textures, edges, and corners. Here, GCTs reduce the feature variances to avoid missing some attributes. Besides, the feature variances become larger in deeper layers, where the high-level features are more discriminative and task-related. As expected, in the layers close to network output, GCT tends to magnify the variance of input feature (the ratio is always greater than 1), but in the layers far away from the output, GCT tends to reduce the variance (the ratio is always less than 1). This phenomenon is consistent with our previous hypothesis and shows that GCT is effective in creating both competition and cooperation among channels. Our observation validates that GCT can adaptively learn the channel relationships at different layers.

As shown, for the stages far away from the network output, the proposed GCT layer tends to reduce the variance of input feature, which encourages cooperation among channels and avoids excessive activation values or loss of useful features. On the contrary, for those stages close to the output, GCT tends to magnify the variance. These phenomena are consistent with the observation in SE and its following work [38], *i.e*., the attention weights are shared in shallower layers, but more discriminative in deeper layers.

4.5. Ablation Studies

In this section, we conduct a serial of ablation experiments to explain the relative importance of each operator in the GCT. At last, we show how the performance changes with regards to the GCT position in a network.

Embedding component. To explain the importance of ℓ_p-norm in the embedding module, we compare embedding operators with different ℓ_p norm. We report the results in Table 6a, which shows all the ℓ_p-norms are effective, but the ℓ_2-norm is slightly better than ℓ_1-norm. Besides, we make a clock time comparison between SE and GCT with different embedding components. As shown in Table 3, ℓ_2-norm is computationally similar to ℓ_1-norm and GCT is much more efficient than SE. The results demonstrate that ℓ_p-norms are robust and perform better.

Normalization component. We also explore the significance of ℓ_p-norm in the channel normalization by comparing ℓ_p normalization with mean and variance normalization. The mean and variance normalization will normalize mean to 0 and variance to 1, which is widely used in normalization layers (*e.g*., [19]). We show all these results in Table 6b. Particularly, mean and variance normalization achieves a top-1 error of 23.7%, which is only slightly better than the ResNet-50 baseline (23.8%). Both ℓ_1 and ℓ_2 normalization make more promising improvements, and ℓ_2 performs slightly better. ℓ_p normalization is better at representation learning in channel normalization.

Adaptation component. We replace the activation function of the gating adaptation with a few different non-linear activation functions and show the results in Table 6c. Compare to the baseline (top-1 of 23.8%), all the non-linear adaptation operator achieves promising performance, and $1 + tanh$ achieves a slightly better improvement. Both $1 + tanh$ and 1+ELU [8], which employ residual connection and can model identity mapping, achieve better results than $Sigmoid$. These results show that the residual connection is important for the gating adaptation.

Application position. To find the best way to deploy GCT layers, we conduct experiments in ResNet-50 architecture on ImageNet by separately applying GCT after all the BN layers, before all the BN layers, and before all the convolutional layers. The results are reported in Table 6d. All the placement methods are effective in using GCT to improve the representational power of networks. However, it is better to employ GCT before all the convolutional layers, which is similar to the strategy in [23] (normalization after ReLU).

5. Conclusion and Future Work

In this paper, we propose GCT, a novel layer that effectively improves the discriminability of deep CNNs by leveraging the relationship among channels. Benefit from the design of combining normalization and gating mechanisms, GCT can facilitate two types of neuron relations, *i.e*., competition and cooperation, with negligible complexity of parameters. We conduct extensive experiments to show the effectiveness and robustness of GCT across a wide range of modern CNNs and datasets. Except for CNNs, recurrent networks (*e.g*., LSTM [15]) are also a popular branch in deep neural networks area. In future work, we will study the feasibility of applying GCT into recurrent networks.

Acknowledgements. This work is supported by ARC DP200100938.

References

[1] Parallel distributed deep learning: Machine learning framework from industrial practice. https://www.paddlepaddle.org.cn/.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation)*, pages 265–283, 2016.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Théodore Bluche. Joint line segmentation and transcription for end-to-end handwritten paragraph recognition. In *NIPS*, 2016.

[5] Chunshui Cao, Xianming Liu, Yi Yang, Yinan Yu, Jiang Wang, Zilei Wang, Yongzhen Huang, Liang Wang, Chang Huang, Wei Xu, et al. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. In *ICCV*, 2015.

[6] Yue Cao, Jiarui Xu, Stephen Lin, Fangyun Wei, and Han Hu. Gcnet: Non-local networks meet squeeze-excitation networks and beyond. In *ICCV Workshops*, pages 0–0, 2019.

[7] Joon Son Chung, Andrew Senior, Oriol Vinyals, and Andrew Zisserman. Lip reading sentences in the wild. In *CVPR*, 2017.

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016.

[9] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *ICML*, 2017.

[10] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *ICML*, 2017.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, pages 630–645. Springer, 2016.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-excite: Exploiting feature context in convolutional neural networks. In *NIPS*, pages 9401–9411, 2018.

[17] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[18] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[20] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015.

[21] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *TPAMI*, 35(1):221–231, 2012.

[22] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[24] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In *NIPS*, 2010.

[25] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*. Springer, 2014.

[27] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *NIPS*, 2014.

[28] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[30] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[32] Bharat Singh, Mahyar Najibi, and Larry S Davis. Sniper: Efficient multi-scale training. In *NIPS*, pages 9333–9343, 2018.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *ICCV*, 2015.

[34] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[35] Antonio Torralba. Contextual priming for object detection. *IJCV*, 53(2):169–191, 2003.

[36] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, pages 4489–4497, 2015.

[37] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[38] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. Eca-net: Efficient channel attention for deep convolutional neural networks. In *CVPR*, 2020.

[39] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

[40] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[41] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[42] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.