Dynamic Neural Network: Convolution, Involution, CondConv

KAO Shiu-hong 20657378

Abstract

The traditional architecture of the Convolution Neural Network (CNN) has been one of the most common and powerful neural networks, especially for dealing with image information. However, some of its shortcomings, such as redundant filters and parameters, are believed to exist, contributing to unnecessary computation, and reducing the efficiency of the network. Dynamic neural networks, on the other hand, provide a more flexible architecture based on different samples.

In this report, traditional Convolution as well as the other two pixel-wise dynamic networks, specifically, Involution and Conditionally Parameterized Convolutions (CondConv), are introduced. The basic architectures of these networks are firstly specified. Afterward, an experiment will be conducted to compare the efficiency and performance of distinct networks.

1. Introduction

In a traditional Convolution Neural Network, each filter is used to scan through every pixel of the input and every sample. From current research, as a filter is believed to determine the existence of some characteristics of the sample, redundant computation therefore exists.

Dynamic neural networks, instead of fitting all the inputs with the same kernels, adapt to different samples and are able to generate different kernels based on a given pixel, vector, or region. We will introduce two types of dynamic neural networks, Involution [1.] and Conditionally Parameterized Convolutions (CondConv) [2.], in this report.

In session 2, basic information about each network will be given, including the architecture and the mathematical expression. In Involution Network, the vector of a pixel across different channels is chosen to generate a kernel, which will be used to fit the neighbour region of the pixel. This enables the network to produce a kernel according to the information across the channels, unlike the traditional Convolution Network. In CondConv, example-dependent scalar weights are applied to the kernels using some routing functions related to the input sample, making the kernel conditioned on the input as well as reduce the multiplication complexity.

2. Network Architecture

In this session, we will introduce the Convolution and two dynamic neural networks which we use in the experiment.

2.1 Standard Convolution

In [3.], a clear representation method of Convolution is introduced. In the standard Convolution Neural Network, the operation at the i^{th} pixel of the image with a kernel W can be interpreted as

$$F'_{(.,i)} = \sum_{j \in \Omega(i)} W[p_i - p_j] F_{(.,j)} + b \tag{1}$$

, where $\Omega(i)$ denotes the set of pixels fitted in the filter at the i^{th} pixel, $F_{(.,j)} \in \mathbb{R}^{c}$ denotes the input feature vector at j^{th} pixel, $F'_{(.,i)} \in \mathbb{R}^{c'}$ denotes the output feature vector at the i^{th} pixel; W is $\mathbb{R}^{c' \times c \times k \times k}$ be a $k \times k$ filter, $W[p_i - p_j] \in \mathbb{R}^{c' \times c}$ is the filter at position offset between i^{th} and j^{th} pixels, and $[p_i - p_j] \in \{ \left(-\frac{k-1}{2}, -\frac{k-1}{2}\right), \left(-\frac{k-1}{2}, -\frac{k-1}{2} + 1\right), \dots, \left(\frac{k-1}{2}, \frac{k-1}{2}\right) \}$, and $b \in \mathbb{R}^{c'}$ is a bias vector. W is shared across every pixel in the input.

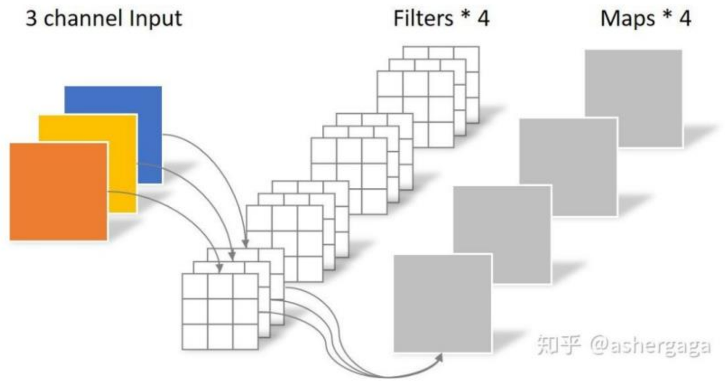

Figure 2.1.1

For example, Figure 2.1.1 shows a standard convolution neural network with $c = 3$, $c' = 4$, and $k = 3$.

2.2 Involution

Different from standard convolution, the involution kernel $\mathcal{H}_i \in \mathbb{R}^{k \times k}$ for the i^{th} pixel of the image is conditioned on the feature vector $X_i \in \mathbb{R}^{c}$. In mathematics,

$$\mathcal{H}_i = \phi(X_i) \tag{2}$$

for ϕ as a kernel generating function. In specific, we take

$$\mathcal{H}_i = \phi(X_i) = W_1 \sigma(W_0 X_i) \tag{3}$$

, where $W_0 \in \mathbb{R}^{\frac{c}{r} \times c}$ and $W_1 \in \mathbb{R}^{k \times k \times \frac{c}{r}}$ for c as the number of channels of the input and r as the control of a reduction ratio, and σ is the Batch Normalization and non-linear activation functions that interleave two linear projections.

The operation at the i^{th} pixel of the image can be then interpreted as:

$$F'_{(.,i)} = \sum_{j \in \Omega(i)} \mathcal{H}_i[p_i - p_j] F_{(.,j)} \tag{4}$$

, where $\Omega(i)$ denotes the set of pixels fitted in the filter at the i^{th} pixel, $F_{(.,j)} \in \mathbb{R}^c$ denotes the input feature vector at j^{th} pixel, $F'_{(.,i)} \in \mathbb{R}^c$ denotes the output feature vector at the i^{th} pixel; $\mathcal{H}_i \in \mathbb{R}^{k \times k}$ is the $k \times k$ filter generating from (2), $\mathcal{H}_i[p_i - p_j] \in \mathbb{R}$ is the weight in filter $\mathcal{H}_i$ position offset between i^{th} and j^{th} pixels, and $[p_i - p_j] \in \{ \left(-\frac{k-1}{2}, -\frac{k-1}{2}\right)$, $\left(-\frac{k-1}{2}, -\frac{k-1}{2} + 1\right), \ldots, \left(\frac{k-1}{2}, \frac{k-1}{2}\right) \}$.

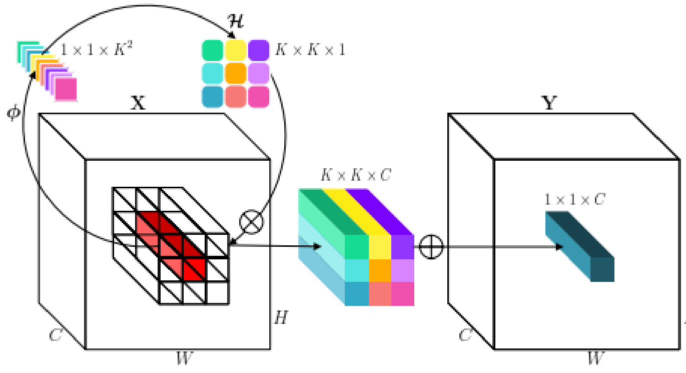

Figure 2.2.1

The example graph illustration is shown in Figure 2.2.1 with ⊗ indicating multiplication broadcast across C channels and ⊕ indicating summation aggregated within the K × K spatial neighborhood.

2.3 CondConv

In standard convolution, an input is fitted with some kernels in size of $k \times k$, and these results are combined afterwards. Each kernel is designed to adapted to every pixel, so the computation complexity is high. In CondConv, however, we parameterize the convolutional kernels by:

$$Output(X) = \sigma((\alpha_1 W_1 + \alpha_2 W_2 + \cdots + \alpha_n W_n) * X) \qquad (5)$$

, where σ is an activation function, $*$ implies the operation of kernel fitting, $\alpha_i = r_i(X)$ is a scalar weight conditioned on the input X computed using a routed function with learned parameters, and W_i is the kernel with the same size as it is in the standard convolution.

This design can achieve the same performance as the standard convolution does since

$$\begin{aligned} Output(X) &= \sigma\big((\alpha_1 W_1 + \cdots + \alpha_n W_n) * X\big) \\ &= \sigma(\alpha_1 (W_1 * X) + \cdots + \alpha_n (W_n * X)) \end{aligned} \qquad (6)$$

, but its computation complexity can be greatly reduced by combining the n kernels together first. Figure 2.3.1 shows the flow of a CondConv layer with an example of $n = 3$.

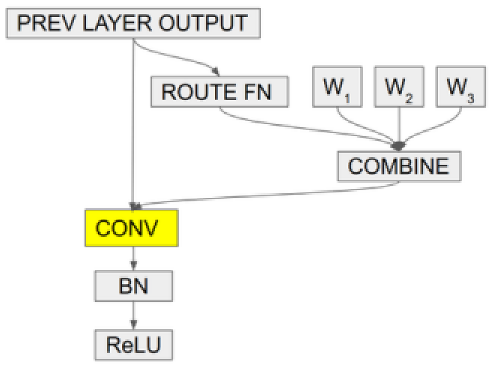

Figure 2.3.1

3. Experiment

All the networks below are built using the PyTorch library, and GPU is used. The source code is provided on Github https://github.com/DanielSHKao/Convolution-CondConv-and-Involution. We conduct experiments on dataset Cifar-10 to compare the performance and speed of these three networks, Convolution, Involution, and CondConv. The Cifar-10 dataset is composed with of 60,000 images in 32 × 32 × 3 in 10 classes, each of which consists of 6,000 examples. Some samples are shown in Figure 3.0.1.

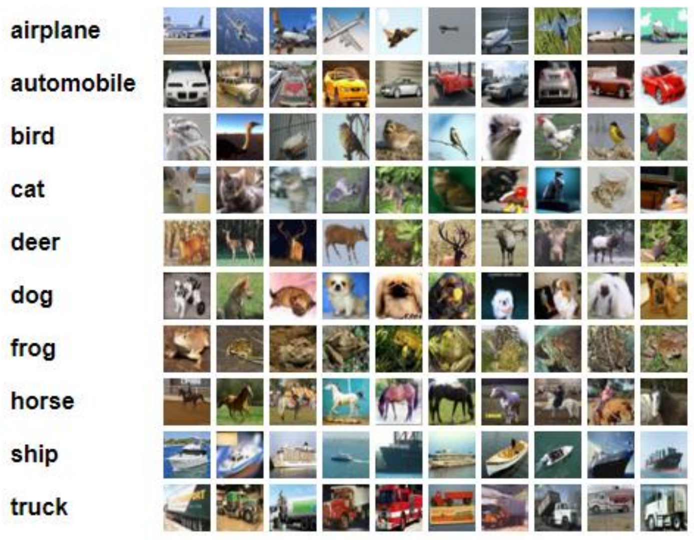

Figure 3.0.1

ResNet is one of the most powerful models for image recognition. Its architecture is shown in Figure 3.0.2.

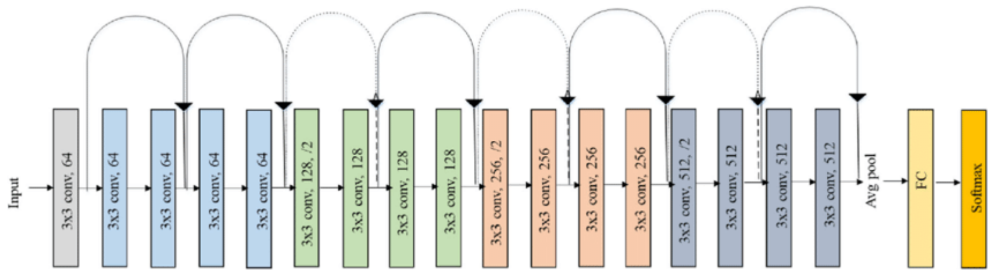

Figure 3.0.2

We apply ResNet18 to the dataset in the case of Convolution and adjust the source code of ResNet18 to conduct Involution and CondConv. In specific, the second layer of the basic block in ResNet18 is substituted with Involution layer or CondConv.

3.1 ResNet with Convolution

There are 11.2M trainable parameters in the model, with estimated size in 44.696 MB. We set batch size to be 64, learning rate to be 0.05 and max epoch to be 30. After 30 epochs, the test accuracy is 91.6%. Related test profiler shows that the mean duration of model forward is 0.00256 second and each training epoch takes 150.19 seconds.

3.2 ResNet with Involution

We adjust the ResNet basic block by replacing the second Convolution layer of each block with Involution layer. Figure 3.2.1 shows the architecture of the model we use.

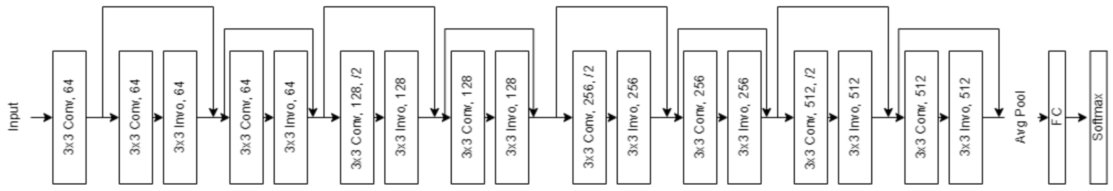

Figure 3.2.1

There are 5.1M trainable parameters with estimated size in 20.467 MB. We set batch size to be 64, learning rate to be 0.05 and max epoch to be 30. After 30 epochs, the test accuracy is 90.8%. Related test profiler shows that the mean duration of model forward is 0.00616 second and each training epoch takes about 34.27 seconds.

We conclude that Involution can significantly increase the training speed without greatly reducing the performance. The reduction of accuracy may originate from the loss of cross-channel information in the samples.

3.3 ResNet with CondConv

We adjust the ResNet basic block by replacing the second Convolution layer of each block with Involution layer. Figure 3.3.1 shows the architecture of the model we use.

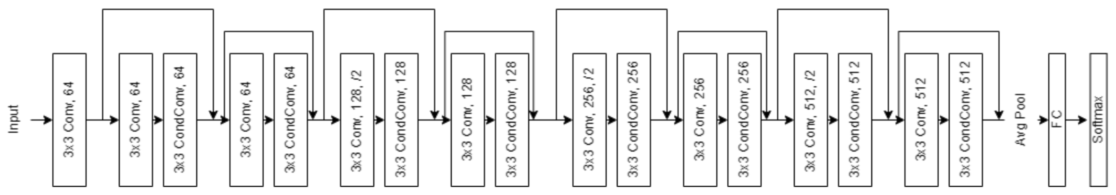

Figure 3.3.1

There are 30.0M trainable parameters with estimated size in 119.929 MB. We set batch size to be 64, learning rate to be 0.05 and max epoch to be 30. After 30 epochs, the test accuracy is 91.2%. Related test profiler shows that the mean duration of model forward is 0.00679 second and each training epoch takes about 74.53 seconds. Almost double number of parameters are used in the model compared to the Convolution ResNet, but half of the training time is used, and the forward time is similar, too.

4. Conclusion

Model	Trainable Params	Training Epoch	Forward	Performance
Convolution	11.2 M	150.19 (s)	0.00256 (s)	91.6 %
Involution	5.1 M	34.27 (s)	0.00616 (s)	90.8 %
CondConv	30.0 M	74.53 (s)	0.00679 (s)	91.2 %

Table 3.1

Table 3.1 shows the comparison of some main properties in the networks. We can derive that the performance of Involution and CondConv is not significantly better than that of Convolution on Cifar-10. Some possible reasons may be the complexity of the dataset and the parameters we set on the network. The advantages of Involution and CondConv may have their optimal architecture instead of ResNet. Also, the samples in Cifar-10 are in size of 32*32*3 (width*height*channels) and in ten classes. Some more complicated datasets such as ImageNet and COCO are used in [2.].

Reference

[1.] Involution: Inverting the Inherence of Convolution for Visual Recognition, https://arxiv.org/abs/2103.06255

[2.] CondConv: Conditionally Parameterized Convolutions for Efficient Inference, https://arxiv.org/pdf/1904.04971

[3.] Decoupled Dynamic Filter Networks, https://arxiv.org/abs/2104.14107

[4.] Dynamic Neural Networks: A Survey, https://arxiv.org/abs/2102.04906

[5.] Dynamic Filter Networks, https://arxiv.org/abs/1605.09673

[6.] Dynamic Region-Aware Convolution, https://arxiv.org/abs/2003.12243

[7.] Pixel-Adaptive Convolutional Neural Networks, https://arxiv.org/abs/1904.05373