Dynamic Convolution for Image Classification
(10-11-2020 - 12-12-2020)

Paper: Dynamic Convolution: Attention over Convolution Kernels

Introduction :

- Deep neural networks showed high performance but require high computational load. Lightweight neural networks, on the other hand, have reduced performance due to their low computational load constraints. Dynamic convolution improves the expressive ability of the model without increasing the depth and width of the network with negligible extra FLOPs.
- Dynamic convolution achieves a balance between network performance and computational load by integrating multiple parallel convolution cores according to the input into a dynamic core, which is data-dependent. Multi-core integration has stronger feature expression capability as the cores are fused in a non-linear form through the attention mechanism.
- Dynamic convolution can be easily embedded to replace the convolution of existing network architectures such as 1x1,3x3, group convolutions.I employed the same as part of this project for Resnet.
- We use a set of K parallel convolution kernels instead of using a single convolutional kernel per layer and aggregate them dynamically for each individual input image via input dependent attention. These kernels are assembled differently for different input images.

Dynamic Perceptron:

- Dynamic perceptron is obtained by aggregating (K) linear functions:

$$\boldsymbol{y} = g(\tilde{\boldsymbol{W}}^T(\boldsymbol{x})\boldsymbol{x} + \tilde{\boldsymbol{b}}(\boldsymbol{x}))$$

$$\tilde{\boldsymbol{W}}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x})\tilde{\boldsymbol{W}}_k, \ \tilde{\boldsymbol{b}}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x})\tilde{\boldsymbol{b}}_k$$

$$\text{s.t.} \quad 0 \leq \pi_k(\boldsymbol{x}) \leq 1, \sum_{k=1}^{K} \pi_k(\boldsymbol{x}) = 1,$$

  Here, Πk is attention weight for the kth linear function, g is the activation function.
- The aggregated weight and bias are functions of input and share the same attention.
- Additional calculations in Dynamic perceptron :
    1. *Attention weight calculation* (π_k)

       Constraints:

1. $\sum_{k} \pi_{k}(x) = 1$. This normalisation simplifies the learning of π k (x), significantly.

2.Flattening attention (near-uniform) in early training epochs to facilitate the learning of convolution kernels.

We integrate these two keys by using softmax with a large temperature for kernel Attention. As temperature increases, the output is less sparse facilitating better learning.

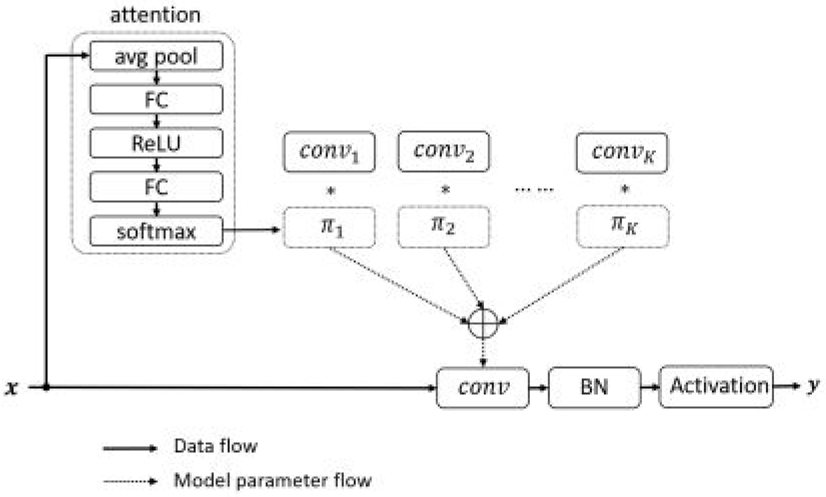
2. *Dynamic weight fusion:*

Dynamic CNN does not increase the depth or width of the network because the parallel kernels share the output channels by aggregation. It introduction of extra computational cost to compute attentions $\{\pi_{k}(x)\}$ and aggregate kernels, which is negligible when compared to convolution.

$$O(\tilde{W}^{T} x + \tilde{b}) >> O(\sum \pi_{k} \tilde{W}_{k}) + O(\sum \pi_{k} \tilde{b}_{k}) + O(\pi(x))$$

Dynamic Convolutional layer :

- Similar to a dynamic perceptron, dynamic convolution also has K cores.



- Since the cores are small, the integration of cores is computationally efficient.

Difficulties:

- Difficult to train as they need joint optimisation of all convolution kernels and attention across multiple layers.
- With the same number of output channels, the model is now bigger with more parameters.
- Training converges slowly with softmax.

Using Grouped convolution to achieve :

- **WHY ? :**
    1. Because the idea of grouped convolution is similar to what we are trying to do via attention of different cores.
    2. Grouped convolutions enable efficient model parallelism.

Implementation details:

- **Attention :**
  Global avg pool followed by two fully connected layers with RELU between them and a softmax to generate normalised attention weights for K convolutional kernels. The first fully connected layer reduces the dimension by 4.
- Temperature is reduced from 30 to 1 linearly in first 10 epochs.(Near-uniform attention in early epochs is crucial).
- Why not softmax? : It only allows a small subset of kernels across layers to be optimised because of its near-hot output.
- Why not sigmoid? : It provides near-uniform attention in early epochs but has a significantly large kernel space than softmax. Thus learning becomes difficult.
- We used dynamic convolution for all the layers except the first layer.
- Each layer has K = 4 (Hyper parameter)convolutional kernels.As K increases,optimisation becomes difficult. Batch size = 128.
- For DY- Resnet: When total epochs are n, initial learning rate is 0.1 and drops by 10 at epochs 0.5 x n, 0.75 x n, 0.85 x n.
- Optimiser : SDG optimiser with 0.9 momentum,dropoutrate = 0.1 before the last layer of Dy-Resnet18.
- Dynamic convolution is applied for all layers for better accuracy.

Implementation of Dy-Resnet:

We replace conv2d in resnet by Dynamic_conv2d:

```
# conv1x1 for dynamic convolution
def conv1x1(in_planes,out_planes,stride = 1):
    return Dynamic_conv2d(in_planes,out_planes,kernel_size = 1,stride =
stride,bias = False,)
# conv3x3 for dynamic convolution
```

```
def conv3x3(in_planes,out_planes,stride = 1,groups = 1,dilation = 1):
    return Dynamic_conv2d(in_planes,out_planes,kernel_size = 3,stride =
stride,padding = dilation,groups = groups,bias = False,dilation = dilation)
```

Implementing Dy_conv :

Attention :

The fully connected layer reduces the dimension by 4.(Hyperparameter)

```
if in_planes != 3: # ratios = 0.25
            hidden_planes = int(in_planes * ratios) + 1
        else:
            hidden_planes = K

        self.fc1   = nn.Conv2d(in_planes,hidden_planes,1,bias = False)
        # self.relu  = nn.ReLU()
       self.fc2   = nn.Conv2d(hidden_planes,K,1,bias = True)
```

Conv layer with groups to achieve dynamic convolution:

```
       aggregate_weight =
torch.mm(softmax_attention,weight).view(-1,self.in_planes,self.kernel_size
,self.kernel_size)# expects two matrices (2D tensors)
       if self.bias is not None:
           aggregate_bias = torch.mm(softmax_attention,self.bias).view(-1)
           output = F.conv2d(x,weight = aggregate_weight,bias =
aggregate_bias,stride = self.stride,padding = self.padding,
                               dilation=self.dilation, groups=self.groups *
batch_size)
       else:
           output = F.conv2d(z,weight = aggregate_weight,bias =
None,stride = self.stride,padding = self.padding,
                               dilation=self.dilation, groups=self.groups *
batch_size)
       output = output.view(batch_size, self.out_planes,
output.size(-2),output.size(-1))
```

Here, we consider batch as a dimensional variable and perform group convolution, because the weight of group convolution is different and the weight of dynamic convolution is different.
We generate weights of dynamic convolution. We get “batch_size” convolution parameters.
(each parameter is different)

Why groups = batch_size?

- In group convolution, the convolutional layers are divided into groups. So, taking groups = batch_size prevent minute loses is data.
  Note: groups is a hyperparameter though.

Dataset and accuracy :

- **CIFAR-10 data set :**
  It has 60,000 colour images (32x32) divided into 10 classes. There are 50,000 training images and 10,000 test images. It has an **accuracy** of **82%** when trained on Dynamic Resnet18 for 40 epochs. Each epoch took 4-5 mins.