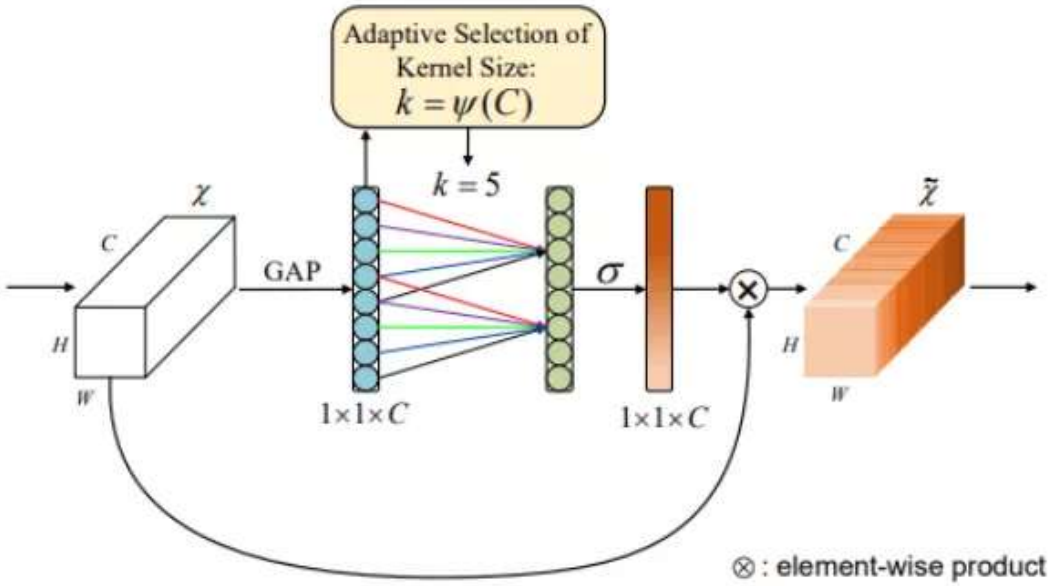
深度学习计算机视觉相关代码可复现论文整理分享

深度学习与NLP 2022-01-06 09:17

收录于话题
#计算机视觉 1 ##干货资源 107



最近在读论文的时候会发现一个问题，有时候论文核心思想非常简单，核心代码可能也就十几行。但是打开作者release的源码时，却发现提出的模块嵌入到分类、检测、分割等代码框架中，导致代码比较冗余，对于特定任务框架不熟悉的我，很难找到核心代码，导致在论文和网络思想的理解上会有一定困难。**因此，作为【论文复现项目】的补充，本项目的宗旨也是让世界上没有难读的论文。**

资源整理自网络，源地址：https://github.com/xmu-xiaoma666/External-Attention-pytorch

目录

- 1. RepMLP Usage
- 2. MLP-Mixer Usage
- 3. ResMLP Usage
- 4. gMLP Usage

- Re-Parameter(ReP) Series
  - 1. RepVGG Usage
  - 2. ACNet Usage
  - 3. Diverse Branch Block(DDB) Usage
- Convolution Series
  - 1. Depthwise Separable Convolution Usage
  - 2. MBConv Usage
  - 3. Involution Usage

内容截图

README.md

Attention Series

- Pytorch implementation of "Beyond Self-attention: External Attention using Two Linear Layers for Visual Tasks---arXiv 2021.05.05"
- Pytorch implementation of "Attention Is All You Need---NIPS2017"
- Pytorch implementation of "Squeeze-and-Excitation Networks---CVPR2018"
- Pytorch implementation of "Selective Kernel Networks---CVPR2019"
- Pytorch implementation of "CBAM: Convolutional Block Attention Module---ECCV2018"
- Pytorch implementation of "BAM: Bottleneck Attention Module---BMCV2018"
- Pytorch implementation of "ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks---CVPR2020"
- Pytorch implementation of "Dual Attention Network for Scene Segmentation---CVPR2019"

- Pytorch implementation of "EPSANet: An Efficient Pyramid Split Attention Block on Convolutional Neural Network---arXiv 2021.05.30"
- Pytorch implementation of "ResT: An Efficient Transformer for Visual Recognition---arXiv 2021.05.28"
- Pytorch implementation of "SA-NET: SHUFFLE ATTENTION FOR DEEP CONVOLUTIONAL NEURAL NETWORKS---ICASSP 2021"
- Pytorch implementation of "MUSE: Parallel Multi-Scale Attention for Sequence to Sequence Learning---arXiv 2019.11.17"
- Pytorch implementation of "Spatial Group-wise Enhance: Improving Semantic Feature Learning in Convolutional Networks---arXiv 2019.05.23"
- Pytorch implementation of "A2-Nets: Double Attention Networks---NIPS2018"
- Pytorch implementation of "An Attention Free Transformer---ICLR2021 (Apple New Work)"
- Pytorch implementation of VOLO: Vision Outlooker for Visual Recognition---arXiv 2021.06.24" 【论文解析】
- Pytorch implementation of Vision Permutator: A Permutable MLP-Like Architecture for Visual Recognition---arXiv 2021.06.23 【论文解析】
- Pytorch implementation of CoAtNet: Marrying Convolution and Attention for All Data Sizes---arXiv 2021.06.09 【论文解析】
- Pytorch implementation of Scaling Local Self-Attention for Parameter Efficient Visual Backbones---CVPR2021 Oral 【论文解析】
- Pytorch implementation of Polarized Self-Attention: Towards High-quality Pixel-wise Regression---arXiv 2021.07.02 【论文解析】
- Pytorch implementation of Contextual Transformer Networks for Visual Recognition---arXiv 2021.07.26 【论文解析】
- Pytorch implementation of Residual Attention: A Simple but Effective Method for Multi-Label Recognition---ICCV2021
- Pytorch implementation of S^2-MLPv2: Improved Spatial-Shift MLP Architecture for Vision---arXiv 2021.08.02 【论文解析】
- Pytorch implementation of Global Filter Networks for Image Classification---arXiv 2021.07.01

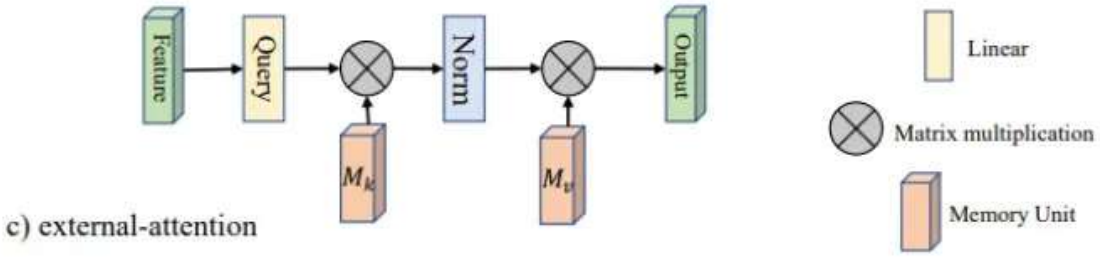
---

1. External Attention Usage

1.1. Paper

"Beyond Self-attention: External Attention using Two Linear Layers for Visual Tasks"

1.2. Overview



1.3. Code

```
from attention.ExternalAttention import ExternalAttention
import torch

input=torch.randn(50,49,512)
ea = ExternalAttention(d_model=512,S=8)
output=ea(input)
print(output.shape)
```

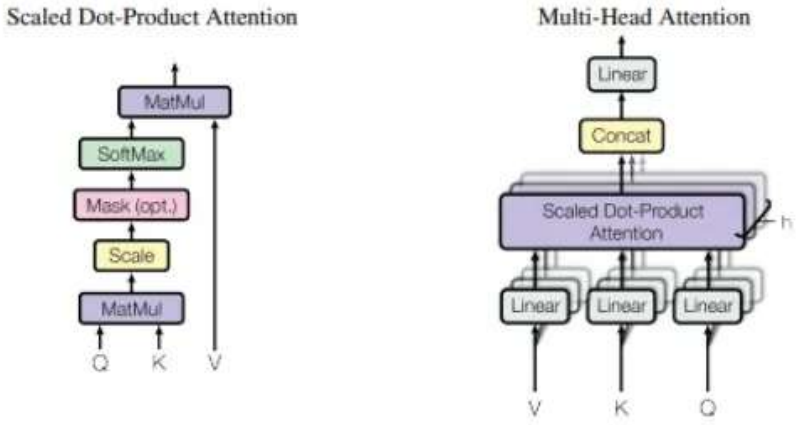
---

2. Self Attention Usage

2.1. Paper

"Attention Is All You Need"

1.2. Overview



1.3. Code

```
from attention.SelfAttention import ScaledDotProductAttention
import torch

input=torch.randn(50,49,512)
sa = ScaledDotProductAttention(d_model=512, d_k=512, d_v=512, h=8)
output=sa(input,input,input)
```

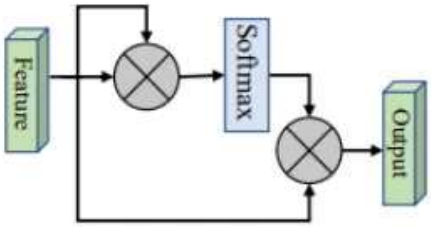
```
print(output.shape)
```

3. Simplified Self Attention Usage

3.1. Paper

None

3.2. Overview



3.3. Code

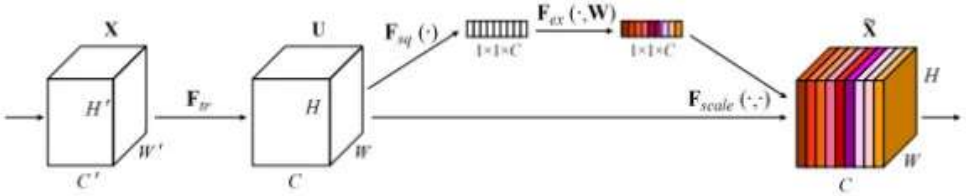
```
from attention.SimplifiedSelfAttention import SimplifiedScaledDotProductAttentic
import torch

input=torch.randn(50,49,512)
ssa = SimplifiedScaledDotProductAttention(d_model=512, h=8)
output=ssa(input,input,input)
print(output.shape)
```

4. Squeeze-and-Excitation Attention Usage

4.1. Paper

"Squeeze-and-Excitation Networks"

4.2. Overview



4.3. Code

```
from attention.SEAttention import SEAttention
import torch
```

```
input=torch.randn(50,512,7,7)
se = SEAttention(channel=512,reduction=8)
output=se(input)
print(output.shape)
```

---

5. SK Attention Usage

5.1. Paper

"Selective Kernel Networks"

5.2. Overview

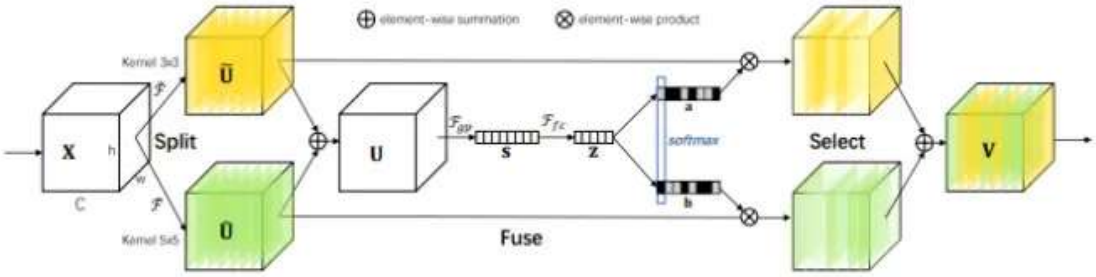


Figure 1. Selective Kernel Convolution.

5.3. Code

```
from attention.SKAttention import SKAttention
import torch

input=torch.randn(50,512,7,7)
se = SKAttention(channel=512,reduction=8)
output=se(input)
print(output.shape)
```

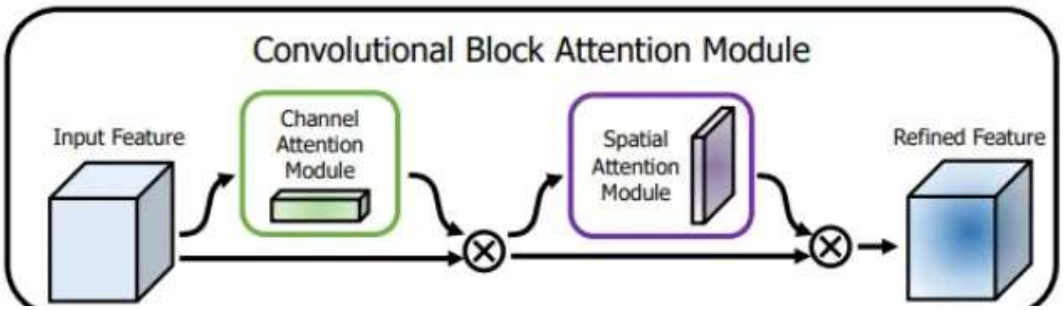
---

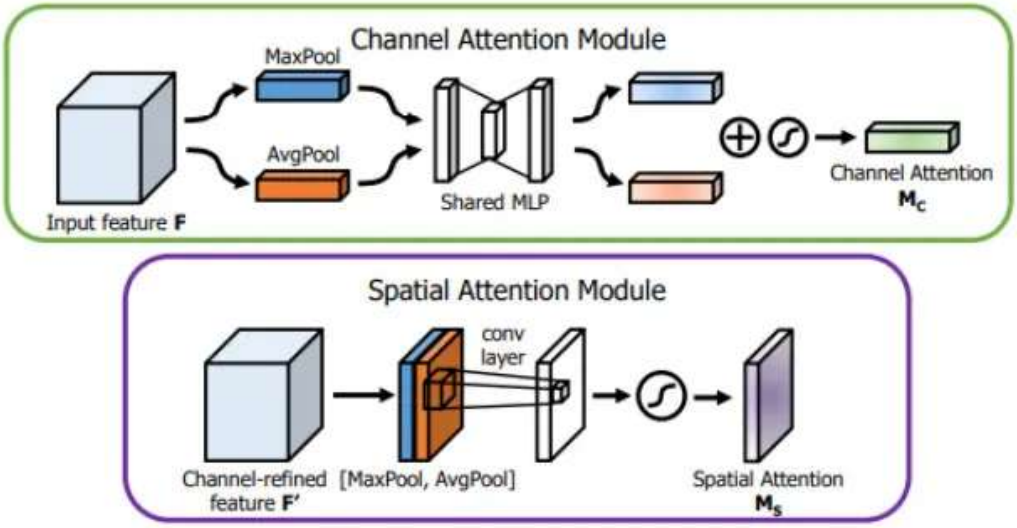
6. CBAM Attention Usage

6.1. Paper

"CBAM: Convolutional Block Attention Module"

6.2. Overview





6.3. Code

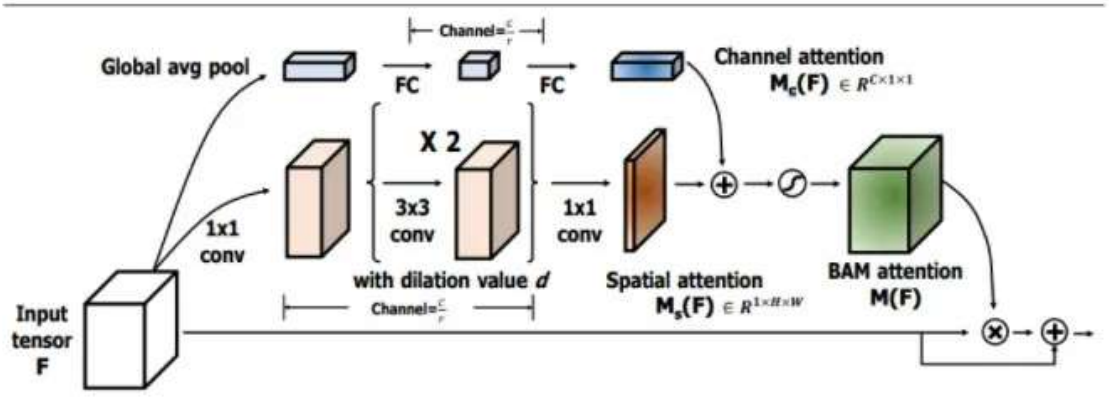
```
from attention.CBAM import CBAMBlock
import torch

input=torch.randn(50,512,7,7)
kernel_size=input.shape[2]
cbam = CBAMBlock(channel=512,reduction=16,kernel_size=kernel_size)
output=cbam(input)
print(output.shape)
```

7. BAM Attention Usage

7.1. Paper

"BAM: Bottleneck Attention Module"

7.2. Overview



7.3. Code

```
from attention.BAM import BAMBlock
```

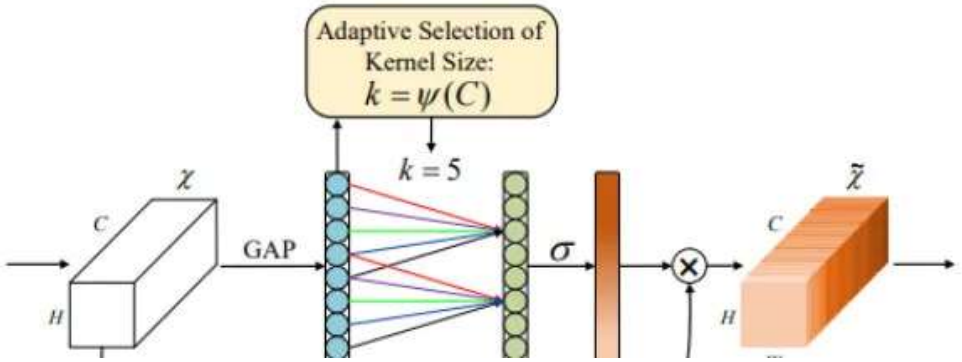
```
import torch

input=torch.randn(50,512,7,7)
bam = BAMBlock(channel=512,reduction=16,dia_val=2)
output=bam(input)
print(output.shape)
```

8. ECA Attention Usage

8.1. Paper

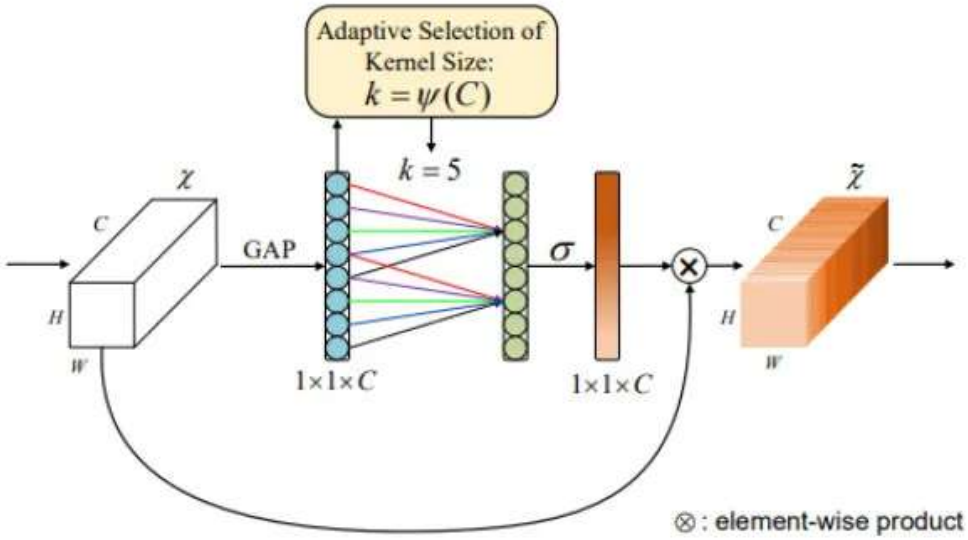
"ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks"

8.2. Overview



README.md

8.2. Overview



8.3. Code

```
from attention.ECAAttention import ECAAttention
import torch

input=torch.randn(50,512,7,7)
```

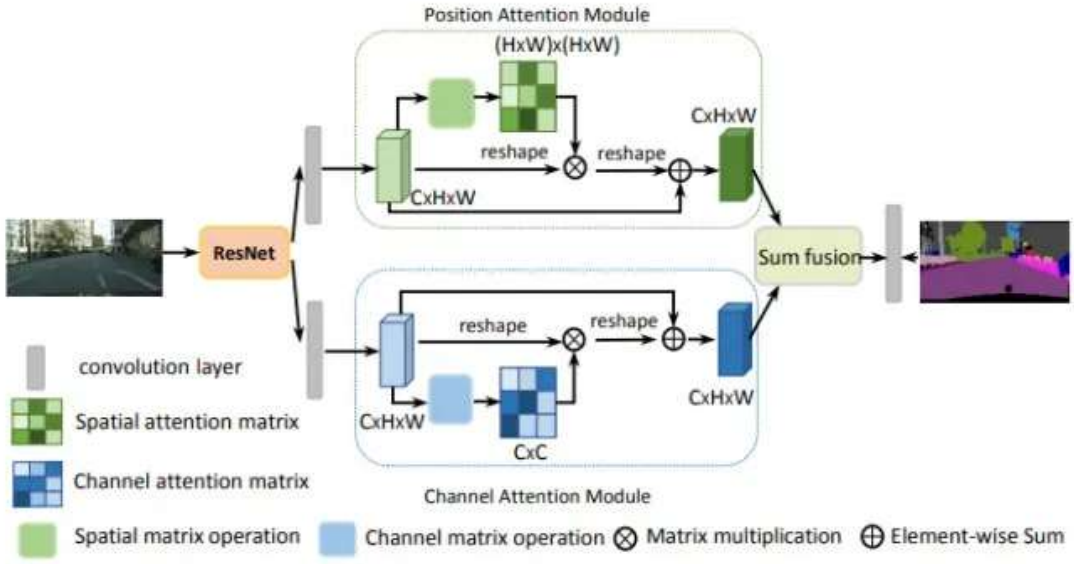
```
input=torch.randn(50,512,7,7)
eca = ECAAttention(kernel_size=3)
output=eca(input)
print(output.shape)
```

9. DANet Attention Usage

9.1. Paper

"Dual Attention Network for Scene Segmentation"

9.2. Overview



9.3. Code

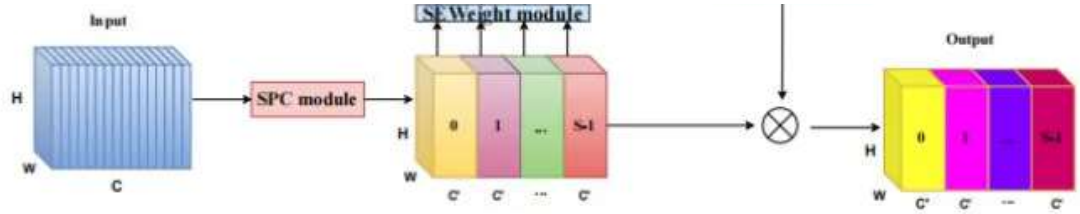
```
from attention.DANet import DAModule
import torch

input=torch.randn(50,512,7,7)
danet=DAModule(d_model=512,kernel_size=3,H=7,W=7)
print(danet(input).shape)
```

10. Pyramid Split Attention Usage

10.1. Paper

"EPSANet: An Efficient Pyramid Split Attention Block on Convolutional Neural Network"

10.2. Overview



10.3. Code

```
from attention.PSA import PSA
import torch

input=torch.randn(50,512,7,7)
psa = PSA(channel=512,reduction=8)
output=psa(input)
print(output.shape)
```

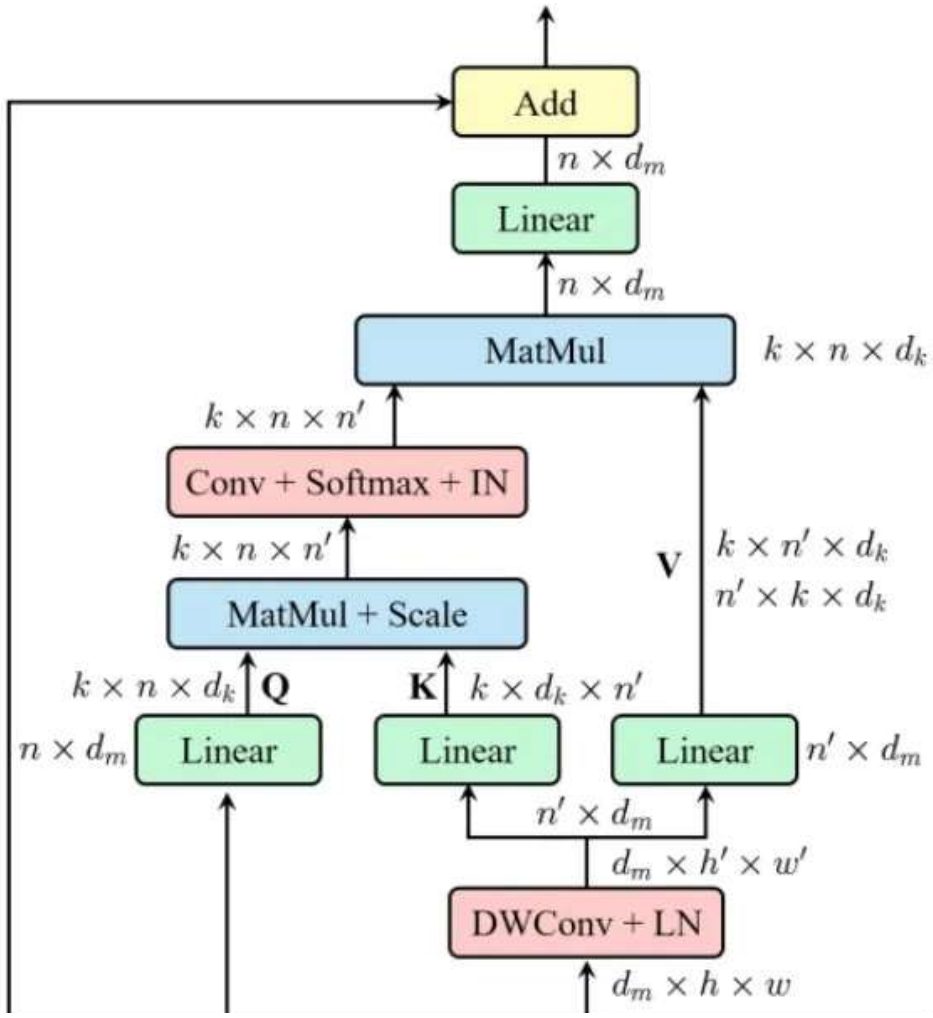
---

11. Efficient Multi-Head Self-Attention Usage

11.1. Paper

"ResT: An Efficient Transformer for Visual Recognition"

11.2. Overview



$$X: n \times d_m$$

11.3. Code

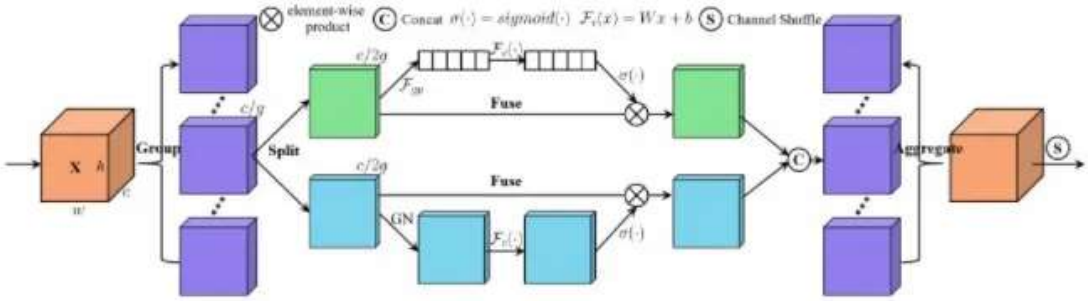
```
from attention.EMSA import EMSA
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,64,512)
emsa = EMSA(d_model=512, d_k=512, d_v=512, h=8,H=8,W=8,ratio=2,apply_transform=T
output=emsa(input,input,input)
print(output.shape)
```

12. Shuffle Attention Usage

12.1. Paper

"SA-NET: SHUFFLE ATTENTION FOR DEEP CONVOLUTIONAL NEURAL NETWORKS"

12.2. Overview



12.3. Code

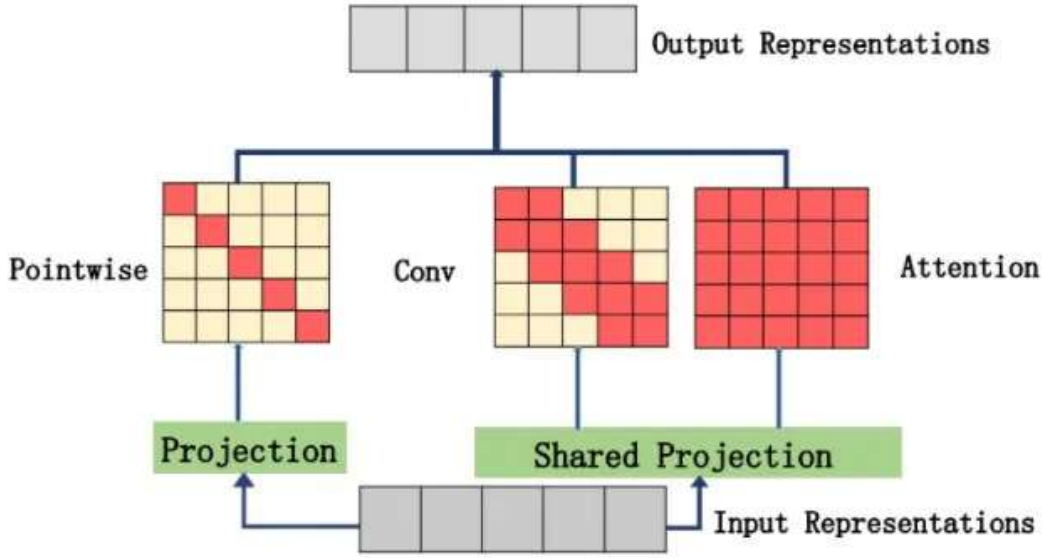
```
from attention.ShuffleAttention import ShuffleAttention
import torch
from torch import nn
from torch.nn import functional as F


input=torch.randn(50,512,7,7)
se = ShuffleAttention(channel=512,G=8)
output=se(input)
print(output.shape)
```

13. MUSE Attention Usage

13.1. Paper

"MUSE: Parallel Multi-Scale Attention for Sequence to Sequence Learning"

13.2. Overview



13.3. Code

```
from attention.MUSEAttention import MUSEAttention
import torch
from torch import nn
from torch.nn import functional as F


input=torch.randn(50,49,512)
sa = MUSEAttention(d_model=512, d_k=512, d_v=512, h=8)
output=sa(input,input,input)
print(output.shape)
```

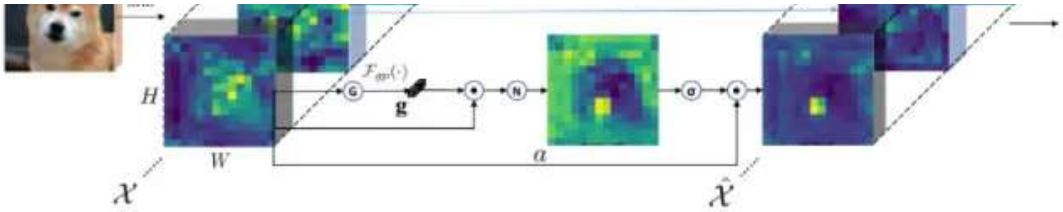
---

14. SGE Attention Usage

14.1. Paper

Spatial Group-wise Enhance: Improving Semantic Feature Learning in Convolutional Networks

14.2. Overview



14.3. Code

```
from attention.SGE import SpatialGroupEnhance
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,512,7,7)
sge = SpatialGroupEnhance(groups=8)
output=sge(input)
print(output.shape)
```

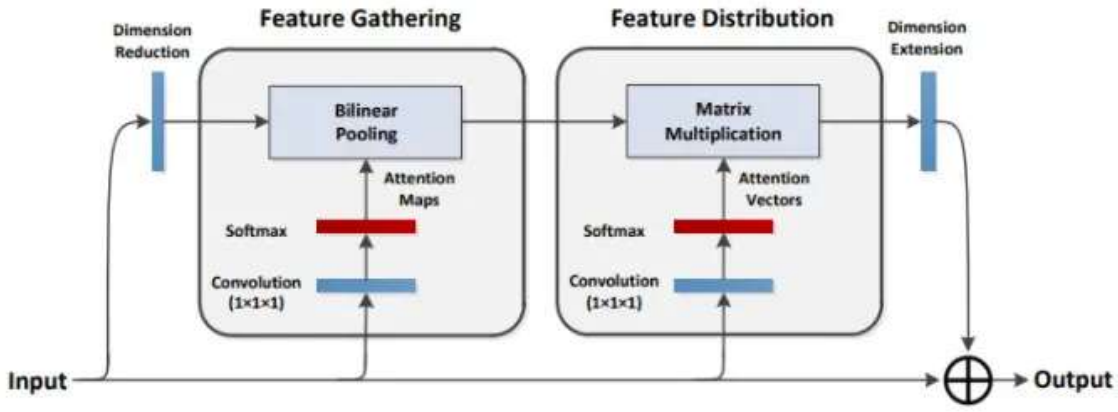
---

15. A2 Attention Usage

15.1. Paper

A2-Nets: Double Attention Networks

15.2. Overview



15.3. Code

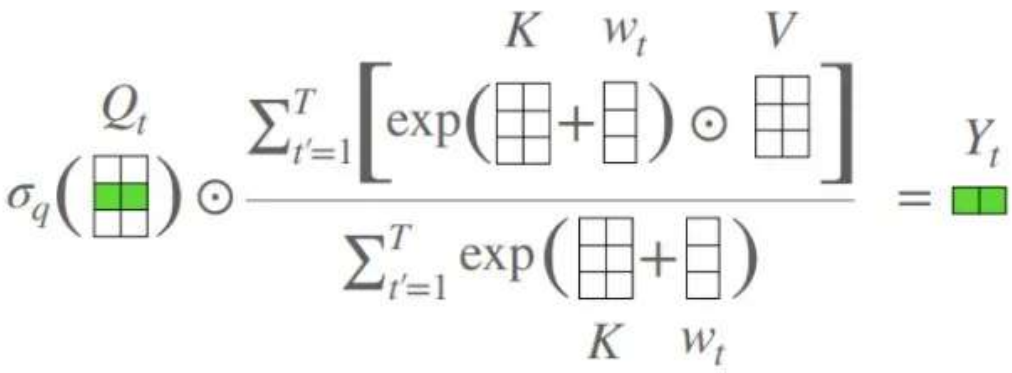
```
from attention.A2Atttention import DoubleAttention
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,512,7,7)
a2 = DoubleAttention(512,128,128,True)
output=a2(input)
print(output.shape)
```

16. AFT Attention Usage

16.1. Paper

An Attention Free Transformer

16.2. Overview



16.3. Code

```
from attention.AFT import AFT_FULL
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,49,512)
aft_full = AFT_FULL(d_model=512, n=49)
output=aft_full(input)
print(output.shape)
```

README.md

16.3. Code

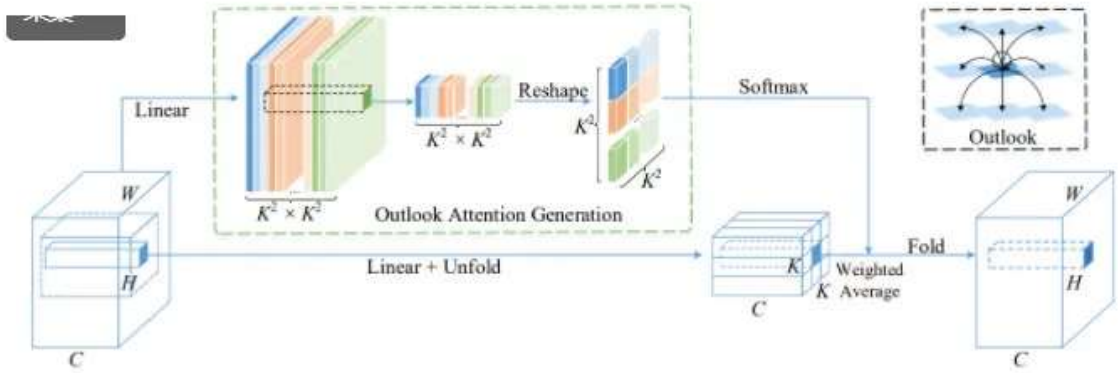
```
from attention.AFT import AFT_FULL
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,49,512)
aft_full = AFT_FULL(d_model=512, n=49)
output=aft_full(input)
print(output.shape)
```

17. Outlook Attention Usage

17.1. Paper

VOLO: Vision Outlooker for Visual Recognition"

17.2. Overview



17.3. Code

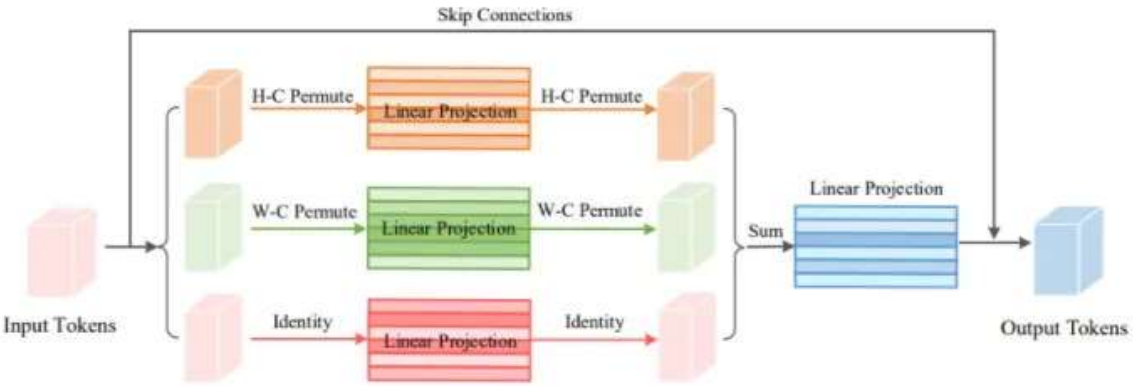
```
from attention.OutlookAttention import OutlookAttention
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,28,28,512)
outlook = OutlookAttention(dim=512)
output=outlook(input)
print(output.shape)
```

18. ViP Attention Usage

18.1. Paper

Vision Permutator: A Permutable MLP-Like Architecture for Visual Recognition"

18.2. Overview



18.3. Code

```
from attention.ViP import WeightedPermuteMLP
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(64,8,8,512)
seg_dim=8
vip=WeightedPermuteMLP(512,seg_dim)
out=vip(input)
print(out.shape)
```

19. CoAtNet Attention Usage

19.1. Paper

CoAtNet: Marrying Convolution and Attention for All Data Sizes"

19.2. Overview

None

19.3. Code

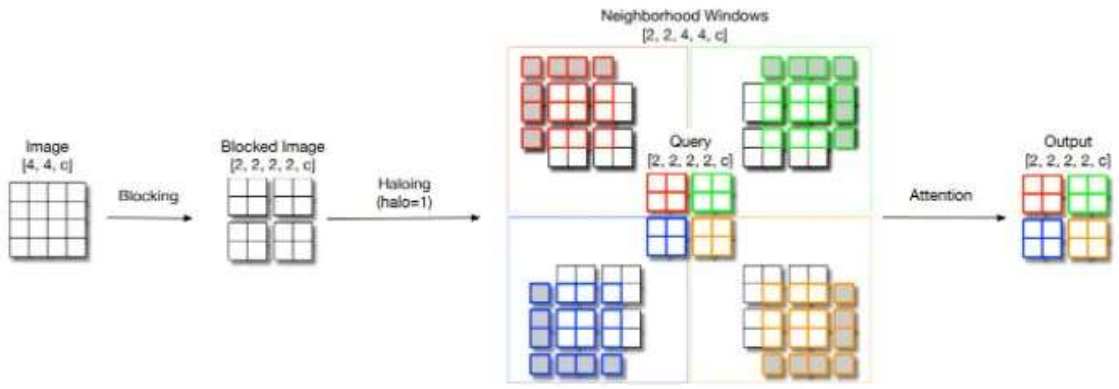
```
from attention.CoAtNet import CoAtNet
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,3,224,224)
mbconv=CoAtNet(in_ch=3,image_size=224)
out=mbconv(input)
print(out.shape)
```

20. HaloNet Attention Usage

20.1. Paper

Scaling Local Self-Attention for Parameter Efficient Visual Backbones"

20.2. Overview



20.3. Code

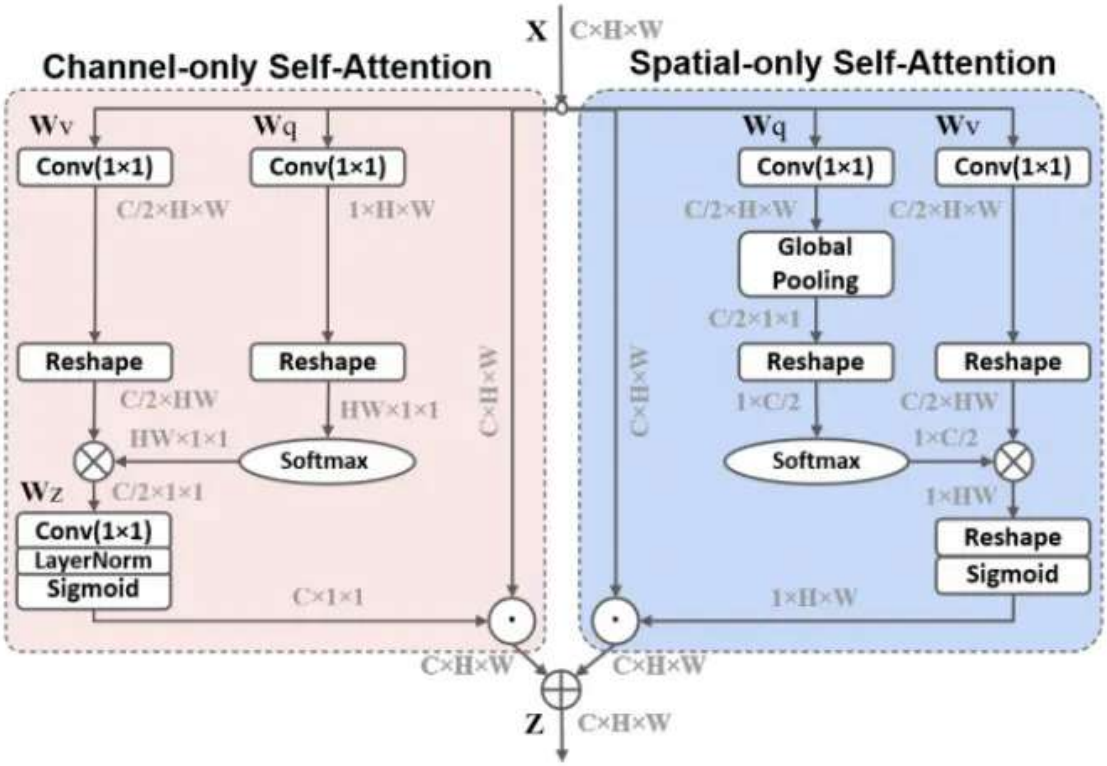
```
from attention.HaloAttention import HaloAttention
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,512,8,8)
halo = HaloAttention(dim=512,
    block_size=2,
    halo_size=1,)
output=halo(input)
print(output.shape)
```

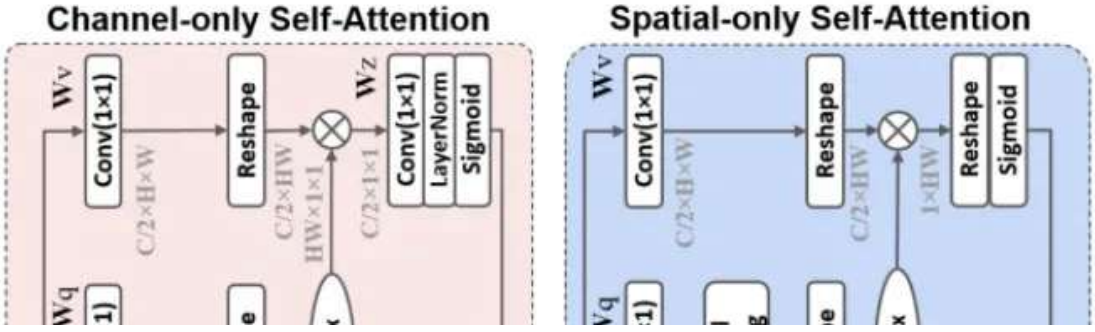
21. Polarized Self-Attention Usage

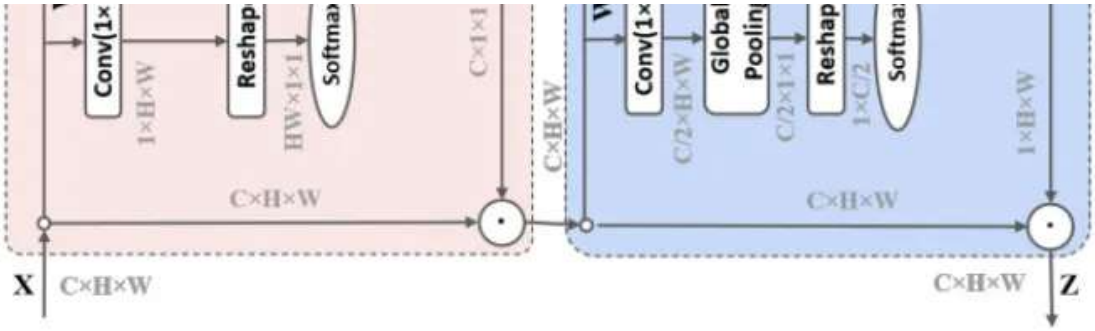
21.1. Paper

Polarized Self-Attention: Towards High-quality Pixel-wise Regression"

21.2. Overview







21.3. Code

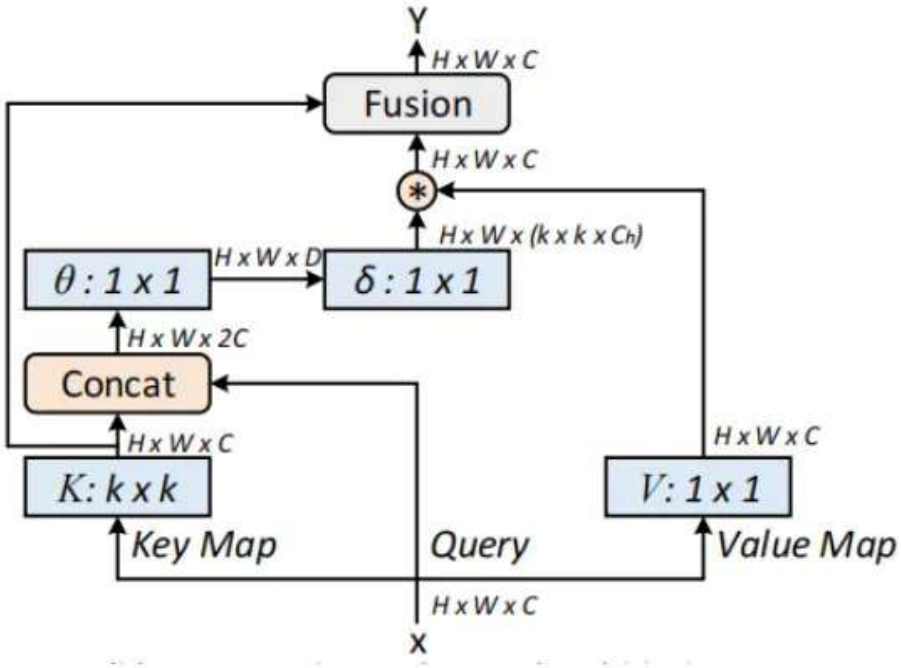
```
from attention.PolarizedSelfAttention import ParallelPolarizedSelfAttention,Sequ
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,512,7,7)
psa = SequentialPolarizedSelfAttention(channel=512)
output=psa(input)
print(output.shape)
```

22. CoTAttention Usage

22.1. Paper

Contextual Transformer Networks for Visual Recognition---arXiv 2021.07.26

22.2. Overview



22.3. Code

```
from attention.CoTAttention import CoTAttention
```

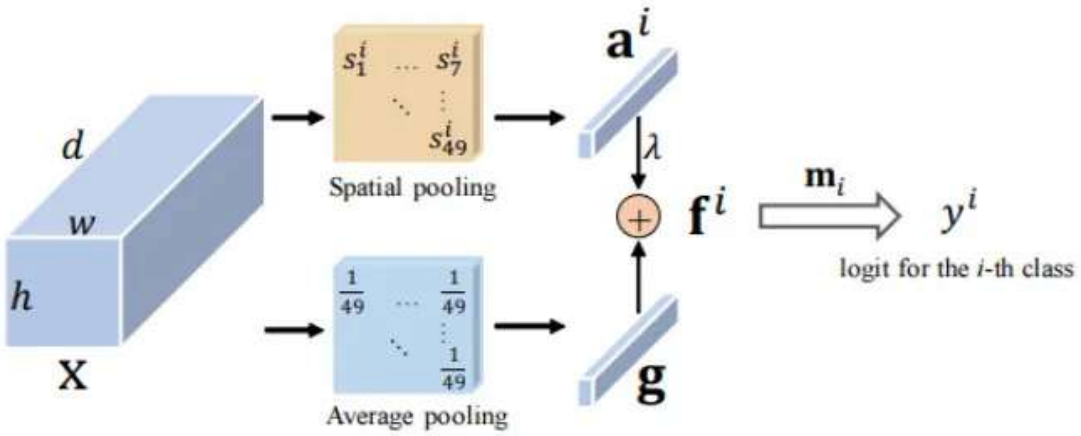
```
from attention.CoTAttention import CoTAttention
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,512,7,7)
cot = CoTAttention(dim=512,kernel_size=3)
output=cot(input)
print(output.shape)
```

23. Residual Attention Usage

23.1. Paper

Residual Attention: A Simple but Effective Method for Multi-Label Recognition---ICCV2021

23.2. Overview



23.3. Code

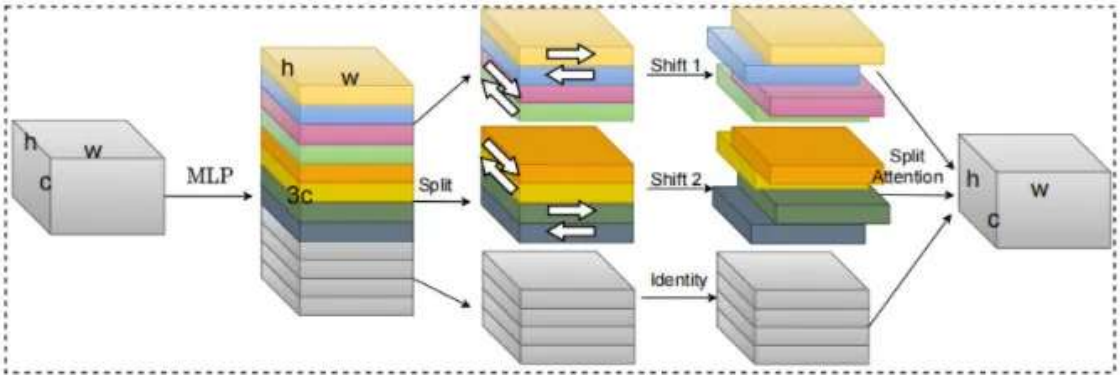
```
from attention.ResidualAttention import ResidualAttention
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,512,7,7)
resatt = ResidualAttention(channel=512,num_class=1000,la=0.2)
output=resatt(input)
print(output.shape)
```

24. S2 Attention Usage

24.1. Paper

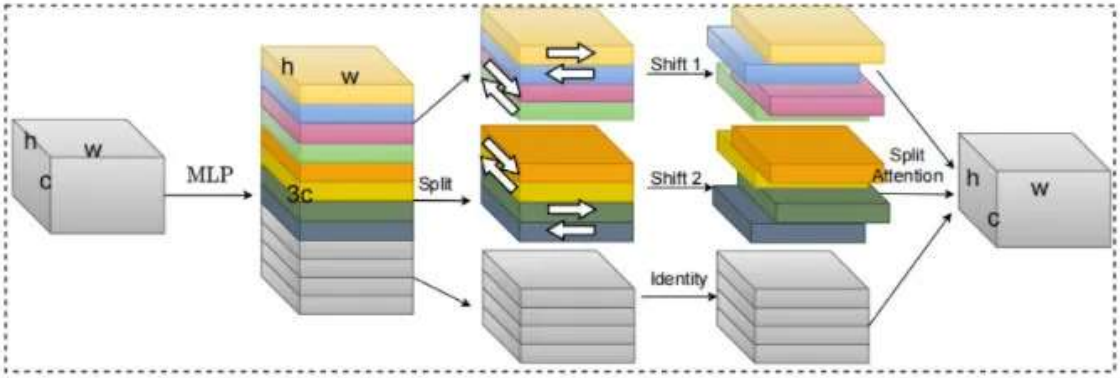
S²-MLPv2: Improved Spatial-Shift MLP Architecture for Vision---arXiv 2021.08.02

24.2. Overview



README.md

24.2. Overview



24.3. Code

```
from attention.S2Attention import S2Attention
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(50,512,7,7)
s2att = S2Attention(channels=512)
output=s2att(input)
print(output.shape)
```

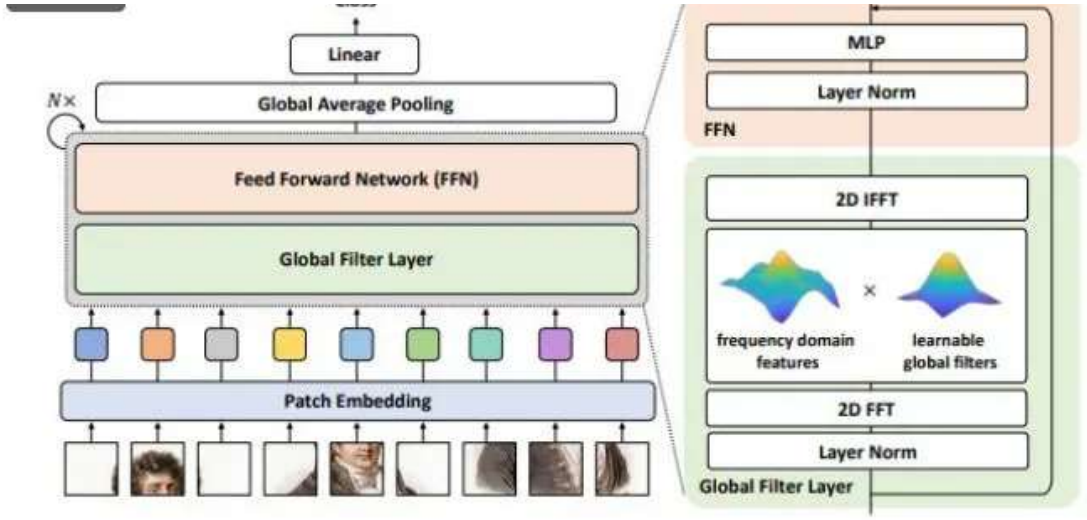
25. GFNet Attention Usage

25.1. Paper

Global Filter Networks for Image Classification---arXiv 2021.07.01

25.2. Overview

采集



25.3. Code - Implemented by 原作者（赵文亮）

```
from attention.gfnet import GFNet
import torch
from torch import nn
from torch.nn import functional as F

x = torch.randn(1, 3, 224, 224)
gfnet = GFNet(embed_dim=384, img_size=224, patch_size=16, num_classes=1000)
out = gfnet(x)
print(out.shape)
```

Backbone CNN Series

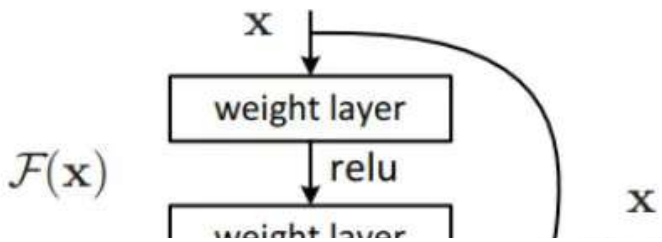
- Pytorch implementation of "Deep Residual Learning for Image Recognition---CVPR2016 Best Paper"
- Pytorch implementation of "Aggregated Residual Transformations for Deep Neural Networks---CVPR2017"

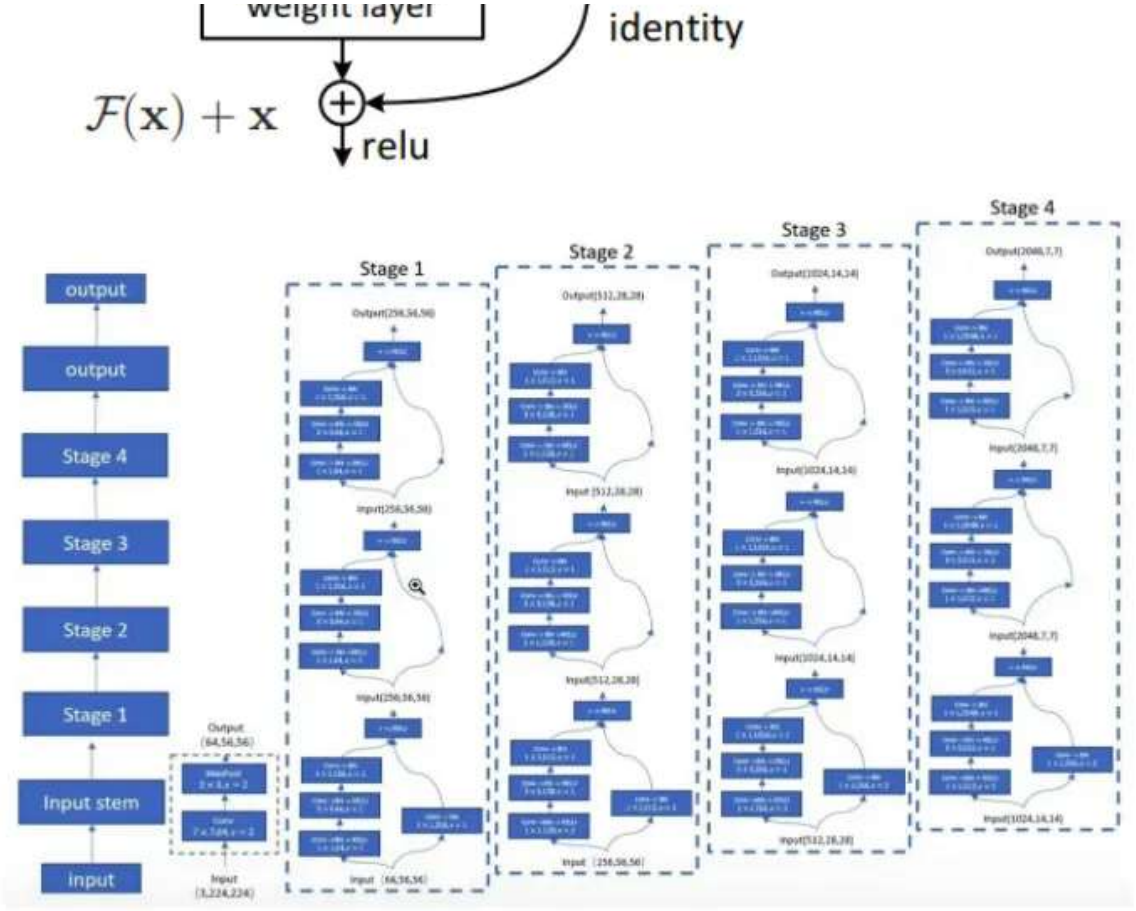
1. ResNet Usage

1.1. Paper

"Deep Residual Learning for Image Recognition---CVPR2016 Best Paper"

1.2. Overview





1.3. Code

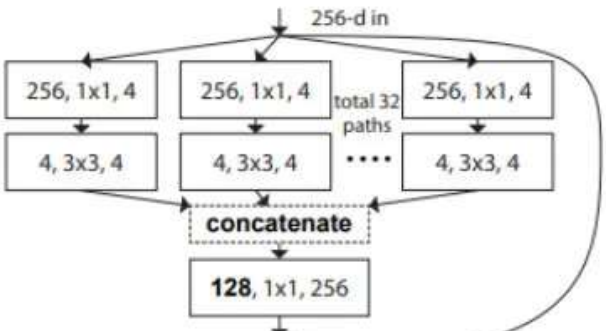
```
from backbone_cnn.resnet import ResNet50,ResNet101,ResNet152
import torch
if __name__ == '__main__':
    input=torch.randn(50,3,224,224)
    resnet50=ResNet50(1000)
    # resnet101=ResNet101(1000)
    # resnet152=ResNet152(1000)
    out=resnet50(input)
    print(out.shape)
```

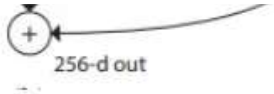
2. ResNeXt Usage

2.1. Paper

"Aggregated Residual Transformations for Deep Neural Networks---CVPR2017"

2.2. Overview





2.3. Code

```
from backbone_cnn.resnext import ResNeXt50,ResNeXt101,ResNeXt152
import torch

if __name__ == '__main__':
    input=torch.randn(50,3,224,224)
    resnext50=ResNeXt50(1000)
    # resnext101=ResNeXt101(1000)
    # resnext152=ResNeXt152(1000)
    out=resnext50(input)
    print(out.shape)
```

MLP Series

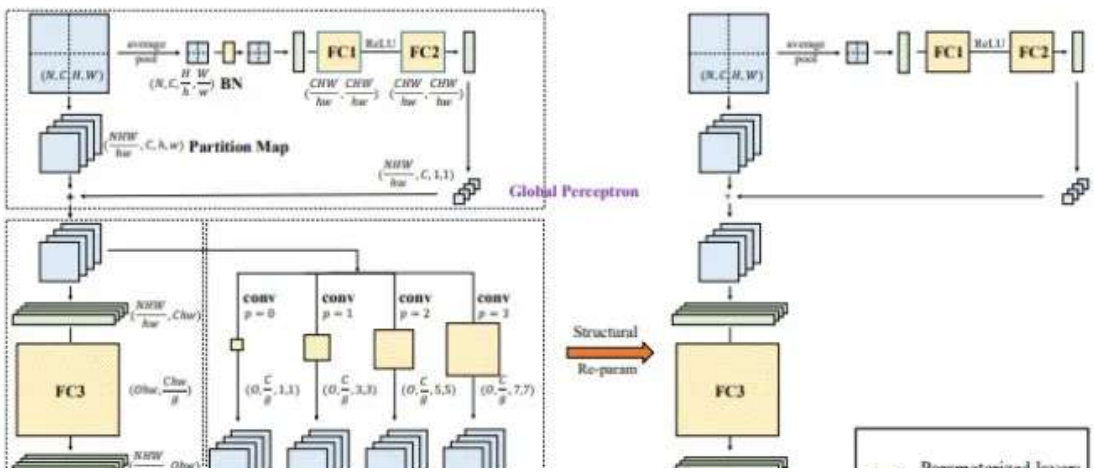
- Pytorch implementation of "RepMLP: Re-parameterizing Convolutions into Fully-connected Layers for Image Recognition---arXiv 2021.05.05"
- Pytorch implementation of "MLP-Mixer: An all-MLP Architecture for Vision---arXiv 2021.05.17"
- Pytorch implementation of "ResMLP: Feedforward networks for image classification with data-efficient training---arXiv 2021.05.07"
- Pytorch implementation of "Pay Attention to MLPs---arXiv 2021.05.17"

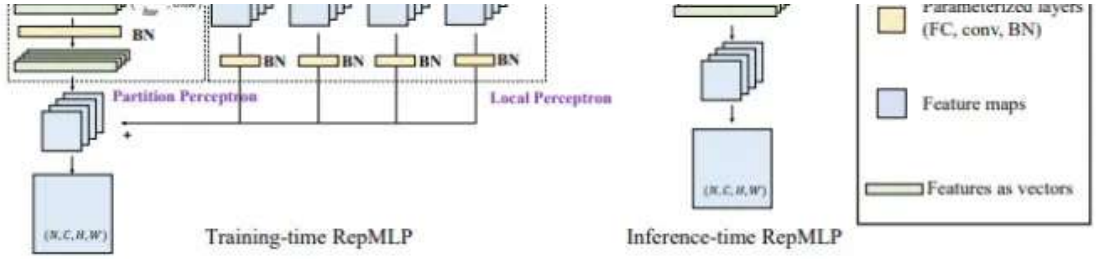
1. RepMLP Usage

1.1. Paper

"RepMLP: Re-parameterizing Convolutions into Fully-connected Layers for Image Recognition"

1.2. Overview





1.3. Code

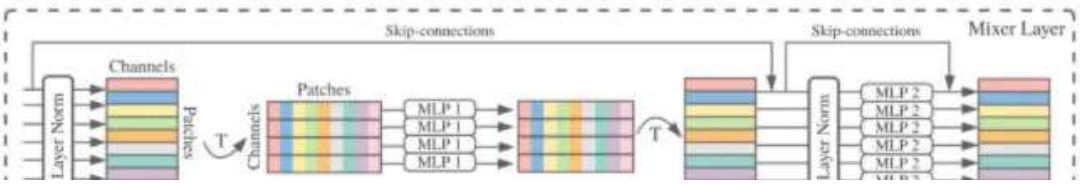
```
from mlp.repmlp import RepMLP
import torch
from torch import nn

N=4 #batch size
C=512 #input dim
O=1024 #output dim
H=14 #image height
W=14 #image width
h=7 #patch height
w=7 #patch width
fc1_fc2_reduction=1 #reduction ratio
fc3_groups=8 # groups
repconv_kernels=[1,3,5,7] #kernel list
repmlp=RepMLP(C,O,H,W,h,w,fc1_fc2_reduction,fc3_groups,repconv_kernels=repconv_k
x=torch.randn(N,C,H,W)
repmlp.eval()
for module in repmlp.modules():
    if isinstance(module, nn.BatchNorm2d) or isinstance(module, nn.BatchNorm1d):
        nn.init.uniform_(module.running_mean, 0, 0.1)
        nn.init.uniform_(module.running_var, 0, 0.1)
        nn.init.uniform_(module.weight, 0, 0.1)
        nn.init.uniform_(module.bias, 0, 0.1)

#training result
out=repmlp(x)
#inference result
repmlp.switch_to_deploy()
deployout = repmlp(x)

print(((deployout-out)**2).sum())
```

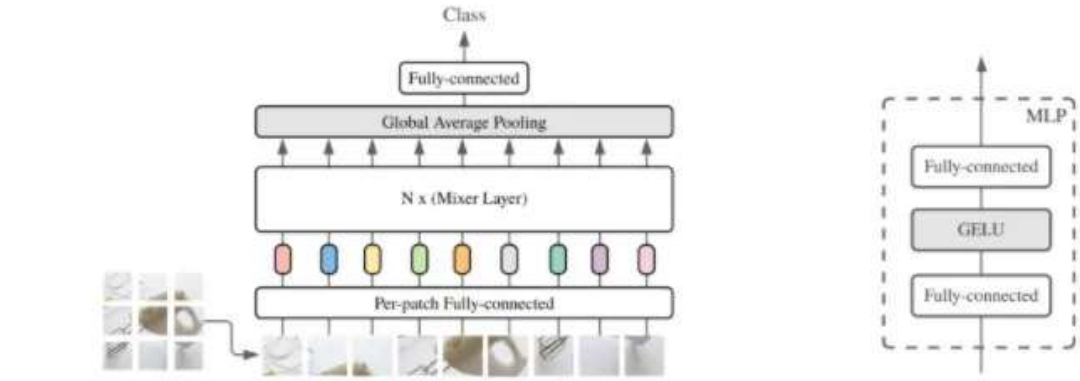
2. MLP-Mixer Usage

2.1. Paper

"MLP-Mixer: An all-MLP Architecture for Vision"

2.2. Overview





2.3. Code

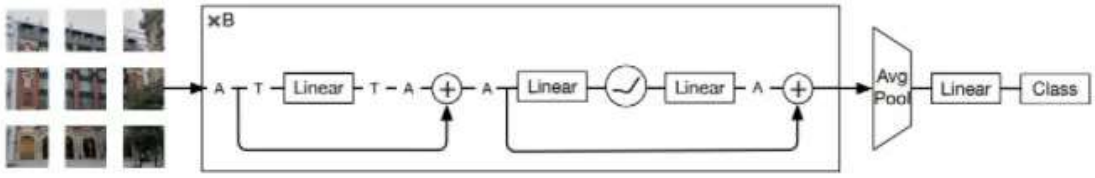
```
from mlp.mlp_mixer import MlpMixer
import torch
mlp_mixer=MlpMixer(num_classes=1000,num_blocks=10,patch_size=10,tokens_hidden_di
input=torch.randn(50,3,40,40)
output=mlp_mixer(input)
print(output.shape)
```

3. ResMLP Usage

3.1. Paper

"ResMLP: Feedforward networks for image classification with data-efficient training"

3.2. Overview



3.3. Code

```
from mlp.resmlp import ResMLP
import torch

input=torch.randn(50,3,14,14)
resmlp=ResMLP(dim=128,image_size=14,patch_size=7,class_num=1000)
out=resmlp(input)
print(out.shape) #the last dimention is class_num
```

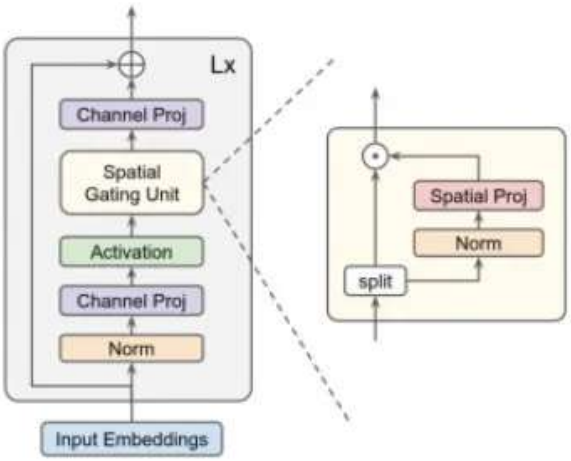
4. gMLP Usage

4.1. Paper

4.1. Paper

"Pay Attention to MLPs"

4.2. Overview



4.3. Code

```
from mlp.g_mlp import gMLP
import torch

num_tokens=10000
bs=50
len_sen=49
num_layers=6
input=torch.randint(num_tokens,(bs,len_sen)) #bs,len_sen
gmlp = gMLP(num_tokens=num_tokens,len_sen=len_sen,dim=512,d_ff=1024)
output=gmlp(input)
print(output.shape)
```

Re-Parameter Series

- Pytorch implementation of "RepVGG: Making VGG-style ConvNets Great Again---

https://github.com/xmu-xiaoma666/External-Attention-pytorch/blob/master/img/GFNet.jpg

README.md

- Pytorch implementation of "RepVGG: Making VGG-style ConvNets Great Again---CVPR2021"
- Pytorch implementation of "ACNet: Strengthening the Kernel Skeletons for Powerful CNN via Asymmetric Convolution Blocks---ICCV2019"
- Pytorch implementation of "Diverse Branch Block: Building a Convolution as an Inception-like Unit---CVPR2021"

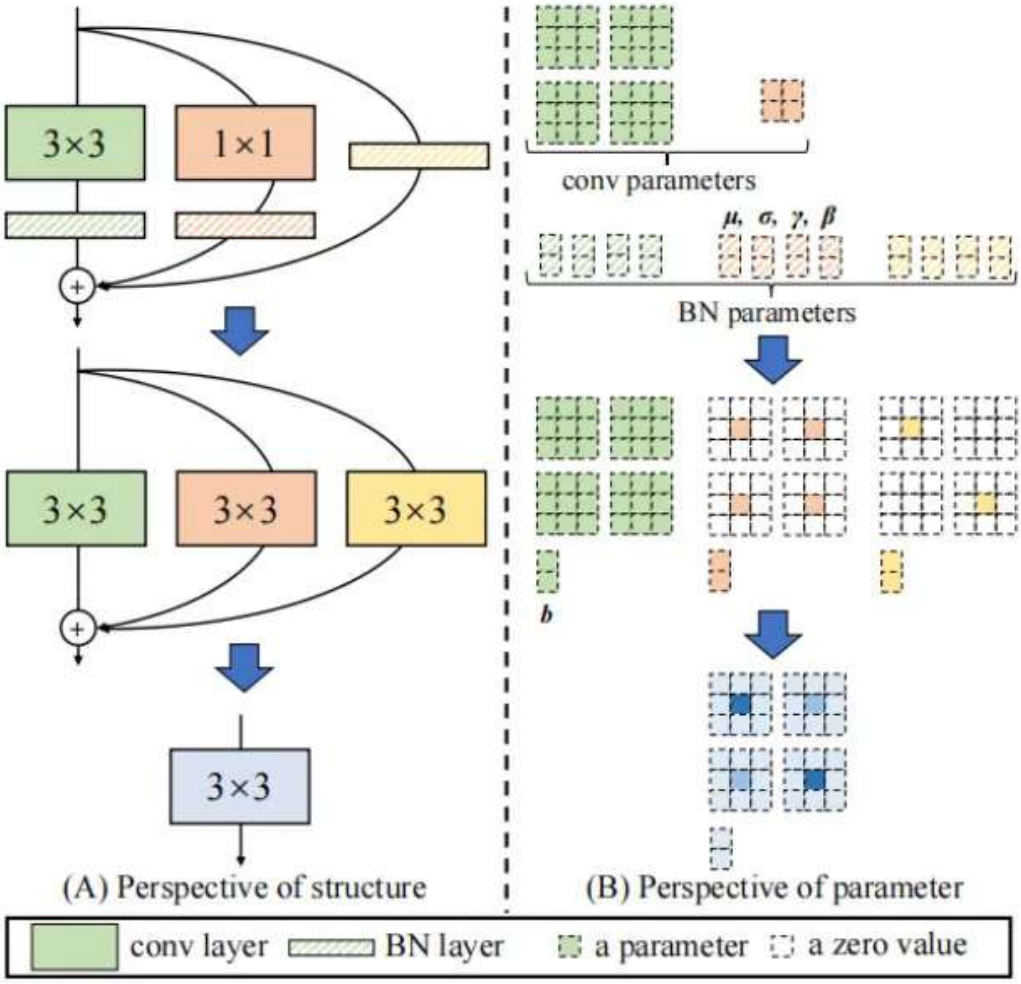
1. RepVGG Usage

1. RepVGG Usage

1.1. Paper

"RepVGG: Making VGG-style ConvNets Great Again"

1.2. Overview



1.3. Code

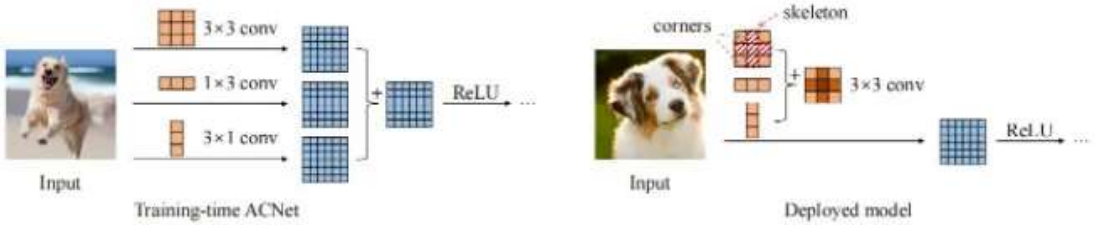
```
from rep.repvgg import RepBlock
import torch


input=torch.randn(50,512,49,49)
repblock=RepBlock(512,512)
repblock.eval()
out=repblock(input)
repblock._switch_to_deploy()
out2=repblock(input)
print('difference between vgg and repvgg')
print(((out2-out)**2).sum())
```

2. ACNet Usage

2.1. Paper

"ACNet: Strengthening the Kernel Skeletons for Powerful CNN via Asymmetric Convolution Blocks"

2.2. Overview



2.3. Code

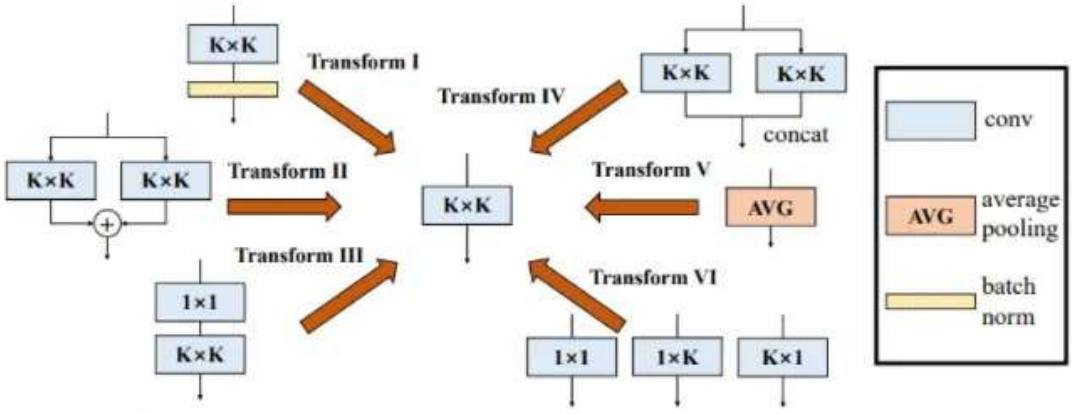
```
from rep.acnet import ACNet
import torch
from torch import nn

input=torch.randn(50,512,49,49)
acnet=ACNet(512,512)
acnet.eval()
out=acnet(input)
acnet._switch_to_deploy()
out2=acnet(input)
print('difference:')
print(((out2-out)**2).sum())
```

2. Diverse Branch Block Usage

2.1. Paper

"Diverse Branch Block: Building a Convolution as an Inception-like Unit"

2.2. Overview



2.3. Code

2.3.1 Transform I

```
from rep.ddb import transI_conv_bn
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,64,7,7)
#conv+bn
conv1=nn.Conv2d(64,64,3,padding=1)
bn1=nn.BatchNorm2d(64)
bn1.eval()
out1=bn1(conv1(input))

#conv_fuse
conv_fuse=nn.Conv2d(64,64,3,padding=1)
conv_fuse.weight.data,conv_fuse.bias.data=transI_conv_bn(conv1,bn1)
out2=conv_fuse(input)

print("difference:",((out2-out1)**2).sum().item())
```

2.3.2 Transform II

```
from rep.ddb import transII_conv_branch
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,64,7,7)

#conv+conv
conv1=nn.Conv2d(64,64,3,padding=1)
conv2=nn.Conv2d(64,64,3,padding=1)
out1=conv1(input)+conv2(input)

#conv_fuse
conv_fuse=nn.Conv2d(64,64,3,padding=1)
conv_fuse.weight.data,conv_fuse.bias.data=transII_conv_branch(conv1,conv2)
out2=conv_fuse(input)

print("difference:",((out2-out1)**2).sum().item())
```

2.3.3 Transform III

```
from rep.ddb import transIII_conv_sequential
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,64,7,7)

#conv+conv
conv1=nn.Conv2d(64,64,1,padding=0,bias=False)
conv2=nn.Conv2d(64,64,3,padding=1,bias=False)
out1=conv2(conv1(input))
```

```
#conv_fuse
conv_fuse=nn.Conv2d(64,64,3,padding=1,bias=False)
conv_fuse.weight.data=transIII_conv_sequential(conv1,conv2)
out2=conv_fuse(input)

print("difference:",((out2-out1)**2).sum().item())
```

2.3.4 Transform IV

```
from rep.ddb import transIV_conv_concat
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,64,7,7)

#conv+conv
conv1=nn.Conv2d(64,32,3,padding=1)
conv2=nn.Conv2d(64,32,3,padding=1)
out1=torch.cat([conv1(input),conv2(input)],dim=1)

#conv_fuse
conv_fuse=nn.Conv2d(64,64,3,padding=1)
conv_fuse.weight.data,conv_fuse.bias.data=transIV_conv_concat(conv1,conv2)
out2=conv_fuse(input)

print("difference:",((out2-out1)**2).sum().item())
```

2.3.5 Transform V

```
from rep.ddb import transV_avg
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,64,7,7)

avg=nn.AvgPool2d(kernel_size=3,stride=1)
out1=avg(input)

conv=transV_avg(64,3)
out2=conv(input)

print("difference:",((out2-out1)**2).sum().item())
```

2.3.6 Transform VI

```
from rep.ddb import transVI_conv_scale
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,64,7,7)
```

```
#conv+conv
conv1x1=nn.Conv2d(64,64,1)
conv1x3=nn.Conv2d(64,64,(1,3),padding=(0,1))
conv3x1=nn.Conv2d(64,64,(3,1),padding=(1,0))
out1=conv1x1(input)+conv1x3(input)+conv3x1(input)

#conv_fuse
conv_fuse=nn.Conv2d(64,64,3,padding=1)
conv_fuse.weight.data,conv_fuse.bias.data=transVI_conv_scale(conv1x1,conv1x3,cor
out2=conv_fuse(input)

print("difference:",((out2-out1)**2).sum().item())
```

Convolution Series

- Pytorch implementation of "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications---CVPR2017"
- Pytorch implementation of "Efficientnet: Rethinking model scaling for convolutional neural networks---PMLR2019"
- Pytorch implementation of "Involution: Inverting the Inherence of Convolution for Visual Recognition---CVPR2021"

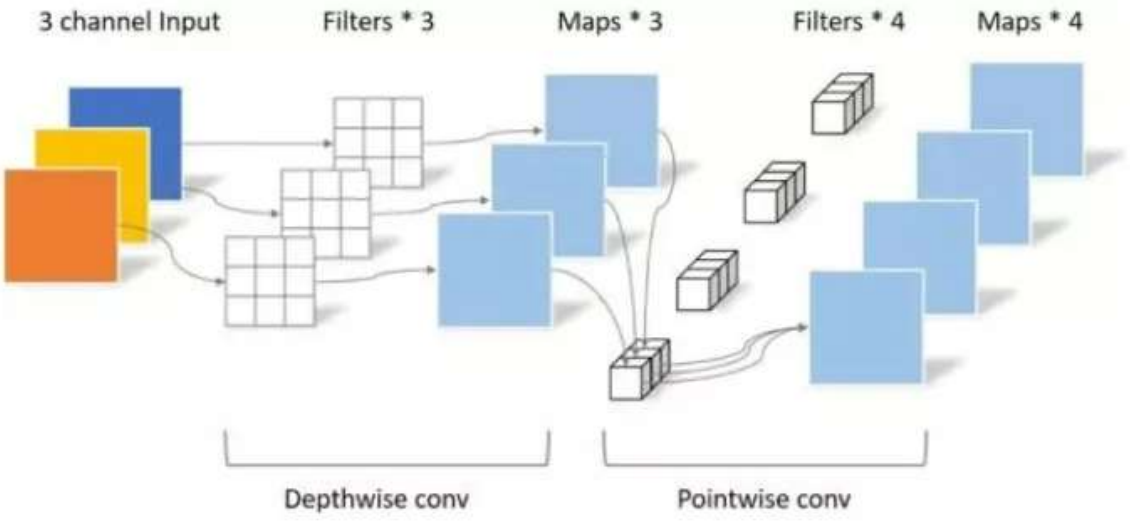
---

1. Depthwise Separable Convolution Usage

1.1. Paper

"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"

1.2. Overview



1.3. Code

```
from conv.DepthwiseSeparableConvolution import DepthwiseSeparableConvolution
```

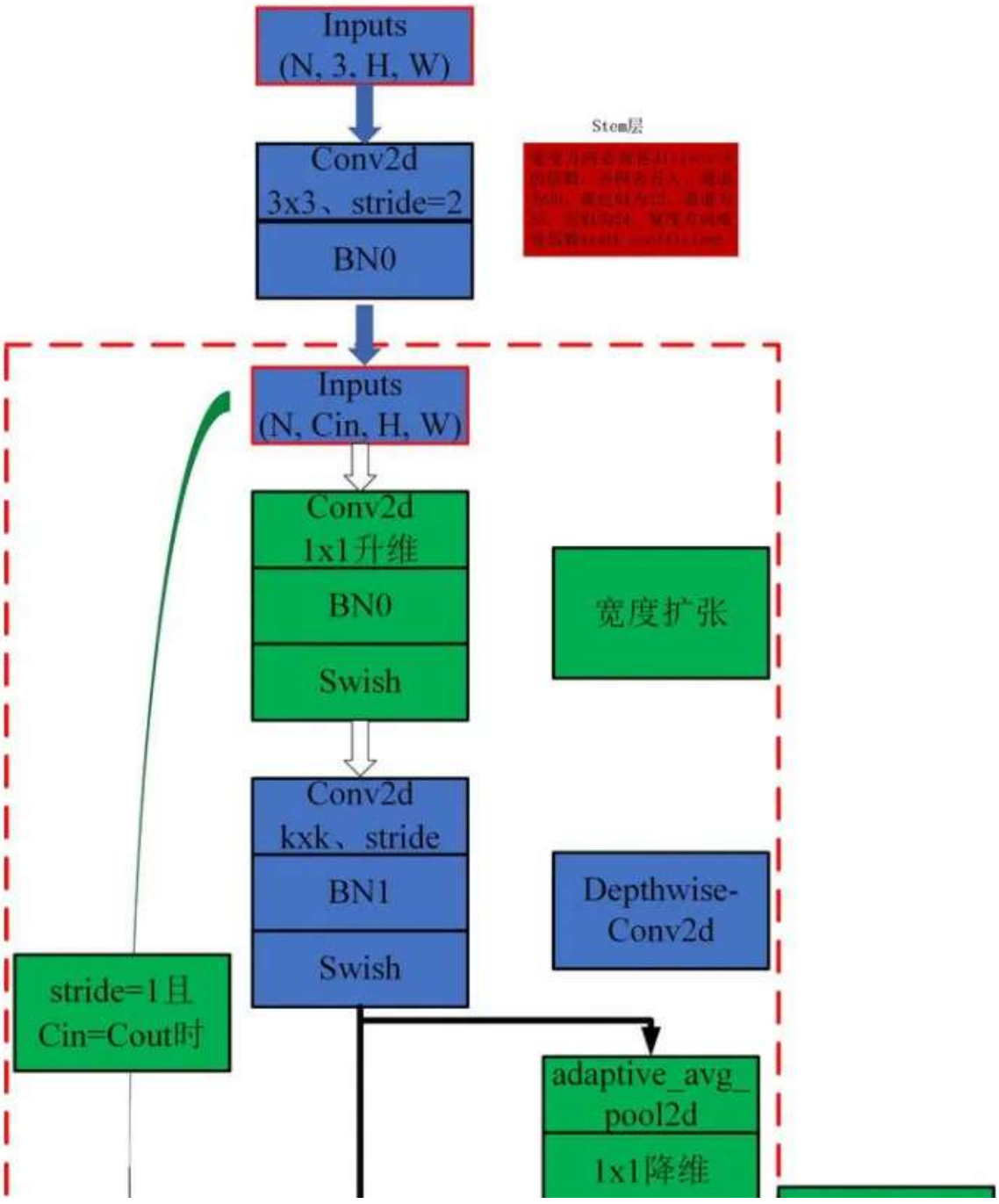
```
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,3,224,224)
dsconv=DepthwiseSeparableConvolution(3,64)
out=dsconv(input)
print(out.shape)
```

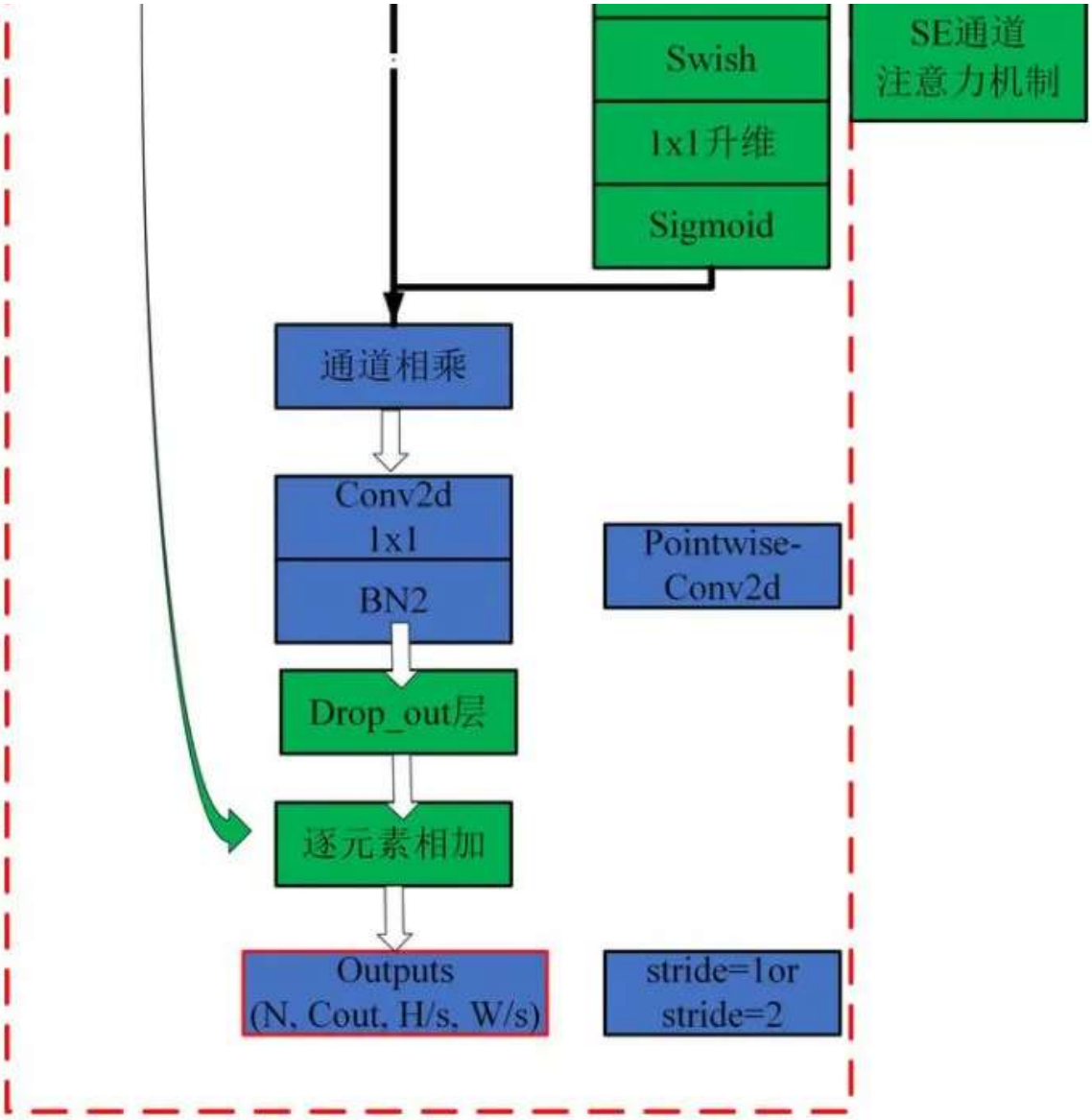
2. MBConv Usage

2.1. Paper

"Efficientnet: Rethinking model scaling for convolutional neural networks"

2.2. Overview





2.3. Code

```
from conv.MBConv import MBConvBlock
import torch
from torch import nn
from torch.nn import functional as F

input=torch.randn(1,3,224,224)
mbconv=MBConvBlock(ksize=3,input_filters=3,output_filters=512,image_size=224)
out=mbconv(input)
print(out.shape)
```

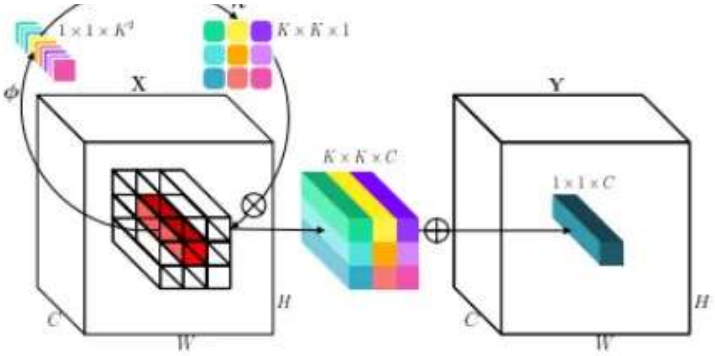
---

3. Involution Usage

3.1. Paper

"Involution: Inverting the Inherence of Convolution for Visual Recognition"

3.2. Overview



3.3. Code

```
from conv.Involution import Involution
```

https://github.com/xmu-xiaoma666/External-Attention-pytorch/blob/master/img/repvgg.png

阅读原文