How Deep Is Deep Enough? HDDE9-2S A CNN Network With 12 Layers Reached Higher Accuracy In Image Classification

A PREPRINT

Guangqun Chen
Dot King Technology
dotkingtech@gmail.com

March 4, 2019

ABSTRACT

In this paper I studied SfNet further. I reduced layers of SfNet created a group of networks: HDDE9 series. HDDE9-2 has 11 layers, HDDE9-2S and HDDE9-3 have 12 layers. In my experiments on dataset CALTECH-256, compared to SFNet, the classification accuracy of HDDE9-3 increases about 4.35%, the classification accuracy of HDDE9-2 increases about 3.07%, the classification accuracy of HDDE9-2S increases about 5.55%. Compared to VGG-16, the classification accuracy of HDDE9-3 increases from 83.63% to 90.73%, the classification accuracy of HDDE9-2 increases to 89.81%, the classification accuracy of HDDE9-2S increases to 92.28%. The improvement is due to SCLayer and architecture change.

***K*eywords** SfNet · high accuracy · HDDE9 · image classification · deep learning · machine learning · SCLayer

1 Introduction

Deep convolutional neural networks have achieved great success in computer vision tasks. One reason for the success is deep architecture.

After ResNet[3] came out, hundreds layers networks are not rare. Due to deep architecture, richer structure features can be generated, representational capacity of networks become more powerful, memory consumption, computation cost increase too.

Recently a new type of layer SCLayer[4] was proposed to reduce parameters' numbers, to increase running speed, and a network SfNet[4] using SCLayer was proposed.

In this article I studied SCLayer and SfNet further, proposed networks HDDE9 series.

HDDE named after this question: how deep is deep enough? 9 is the base structure layers number.

The article is organised as follows: Network HDDE9 series are described in Section 2. In Section 3 experiments to evaluate HDDE9 series are given. In Section 4 conclusions are drawn and Todo list is given. In SectionA source code of HDDE9 series in pytorch is listed.

2 HDDE9 Series

2.1 Review of SCLayer and SfNet

The SCLayer uses maxpool to replace 2D cross-correlation operations in order to abandon redundant informations, to reduce parameters, and to reduce computations.

SfNet is created by replacing layers of scale 4 and scale 5 in VGG-16 with MixedSCLayers and SCLayers. The kernel size of SCLayers is 3×3.

2.2 HDDE9 Series

Using smaller kernel size and stacking multiple layers together we can reach bigger receptive field with fewer parameters.

For SCLayer if we use bigger kernel size, it doesn't increase parameters'number. To get bigger receptive field we can increase SCLayer's kernel size without stacking them together. Based on this truth, I reduced layers of SfNet in scale4 and scale5 from 3 to 1, increased SCLayer kernel size to 7, and created the base structure of HDDE91.

When use 3 FC layers classifier just as VGG-16[1] does, it is HDD9-3 which has 12 layers.

When reduced 1 layer from the classifier, it is HDDE9-2 which has 11 layers.

Add one more layers to HDDDE9-2 to increase features numbers to squeeze performance further, it is HDDE9-2S which has 12 layers2.

Table 1: HDDE9 Base Architecture

Scale	HDDE9
1	$[3 \times 3, 64, B, R] \times 2$
	2×2 max pool stride 2
2	$[3 \times 3, 128, B, R] \times 2$
	2×2 max pool stride 2
3	$[3 \times 3, 256, B, R] \times 3$
	2×2 max pool stride 2
4	$[MSC(7 \times 7, 512), B, R] \times 1$
	2×2 max pool stride 2
5	$[MSC(7 \times 7, 512), B, R] \times 1$
	2×2 max pool stride 2

B: BatchNorm layer. R:ReLU layer. MSC: MixedSCLayer. MixedSCLayer smear features number is 16.

Table 2: HDDE9-2 HDDE9-2S and HDDE9-3 Architecture

	HDDE9-2	HDDE9-3	HDDE9-2S
Base	HDDE9		
Other	$[FC - 2048, B, R] \times 1$	$[FC - 2048, B, R] \times 2$	[$conv : 1 \times 1$-2048,B,R] $[FC - 2048, B, R] \times 1$
	FC-256		

3 Experiments

3.1 Experiment Set up

I used the same experiment set up as in the paper[4]. Details see the paper.

3.2 Experiment Results

Table 3 lists Accuracy and Accuracy Improvement. Data of VGG-16 and SfNet are from the paper[4]. Table4 give training loss and accuracy of each epoch on CALTECH-256[2], Figure1 shows training loss curve and accuracy curve

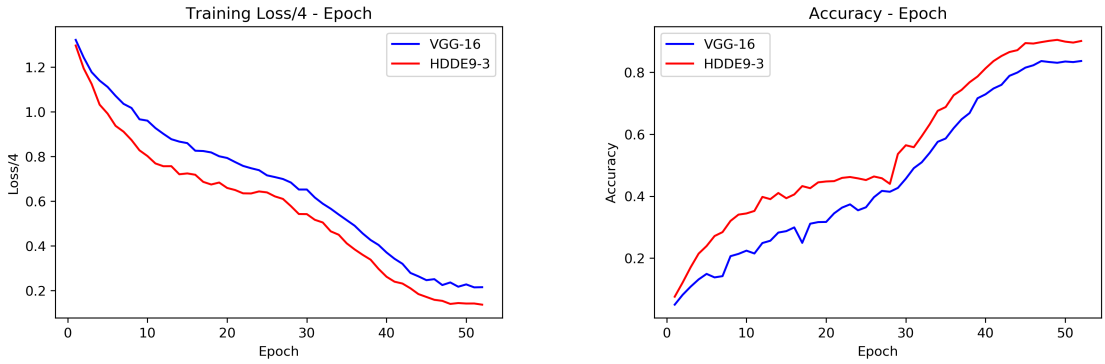

Figure 1: VGG-16 and HDDE9-3 Training Loss and Accuracy Comparison on CALTECH-256.

Table 3: Accuracy of VGG-16, SfNet , HDDE9-3,HDDE9-2 and HDDE9-2S

	VGG-16	SfNet	HDDE9-3	HDDE9-2	HDDE9-2S
CALTECH-256	83.63%	86.74%	90.73%	89.81%	92.28%
Accuracy Improvement	0	3.11%	7.1%	6.18%	8.65%

comparision of VGG-16 and HDDE9-3. I also tested HDDE9-3 with different SCLayer filter's sizes. For dataset CALTECH-256, using size 7 is a good choice.

4 Conclusions and Future Works

In this section I want to correct a conclusion in paper[4]. SCLayer is not a good choice to stack layers and deep CNN, because to increase receptive field, only need to increase kernel size, and kernel size can't be unlimited increased. But I keep this open, maybe some day some one can find methods to stack them together or use it for different purpose.

There are somethings need to study further: How does stride value influence SCLayer's performance? Because of one layer structure in Scale4 and Scale5, maybe multiscale is worth to study, using different stride values? The essence of ideas in the paper[4] is to implement multiform piecewise function fitting, layers before scale4 create set of features can be viewed as multiple function forms, the job in scale 4 and scale 5 is to use these features basis to fit the function, SCLayer uses moving windows to scan the spatial space. Moving window has drawback to handle maximum values, How to implement multiform piecewise function fitting without moving window? attention, multigrid, quadtree? or abandon SCLayer and cluster every features map directly and get maximum value map?

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556v6*, 2015.

[2] Greg Griffin, Alex Holub and Pietro Perona. Caltech-256 Object Category Dataset. http://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. *arXiv:1603.05027v3*, 2016.

[4] Guangqun Chen. Structure Composing Layer: A New Type of Layer to Deep and Speed up Convolutional Neural Networks. *viXra:1901.0463v3* 12. Feb,2019. http://viXra.org/abs/1901.0463?ref=10401489.

A HDDE9-2, HDDE9-2S and HDDE9-3 Implementation with Pytorch

```
class MixedSCLayer(nn.Module):
    def __init__(self,input_ch,output_ch,kernel_size,stride,
        padding=1,smear_channels = 16):
        super().__init__()
```

```
        self.avg_conv = nn.Conv2d(input_ch,smear_channels,
        kernel_size,stride,padding)
        self.maxpool = nn.MaxPool2d(kernel_size,stride,padding )
        self.fd_conv=nn.Conv2d(input_ch,output_ch-smear_channels,1,1)
    def forward(self,x):
        y = self.avg_conv(x)
        z = self.maxpool(x)
        return  torch.cat([y,self.fd_conv(z)],1)


class SCLayer(nn.Module):
    def __init__(self,input_ch,output_ch,kernel_size,stride,
        padding=1):
        super().__init__()
        self.maxpool = nn.MaxPool2d(kernel_size,stride,padding )
        self.fd_conv=nn.Conv2d(input_ch,output_ch,1,1)
    def forward(self,x):
        z = self.maxpool(x)
        return  self.fd_conv(z)


# in_size:
# image size 224 x 224  in_size: 7 x 7 = 49
#CALTECH-256 out_class: 256

class HDDE9_BASE(nn.Module):
    def __init__(self):
        super().__init__()

        self.feature = nn.Sequential(
                nn.Conv2d(3, 64, 3, 1, 1 ),
                nn.BatchNorm2d(64),nn.ReLU(True),
                nn.Conv2d(64, 64, 3,1, 1),
                nn.BatchNorm2d(64),nn.ReLU(True),
                nn.MaxPool2d(2,2),
                nn.Conv2d(64, 128,3,1,1),
                nn.BatchNorm2d(128),nn.ReLU(True),
                nn.Conv2d(128, 128,3, 1,1),
                nn.BatchNorm2d(128),nn.ReLU(True),
                nn.MaxPool2d(2,2),
                nn.Conv2d(128, 256,3,1,1),
                nn.BatchNorm2d(256),nn.ReLU(True),
                nn.Conv2d(256, 256,3,1,1),
                nn.BatchNorm2d(256),nn.ReLU(True),
                nn.Conv2d(256, 256,3,1, 1),
                nn.BatchNorm2d(256),nn.ReLU(True),
                nn.MaxPool2d(2,2),
                MixedSCLayer(256,512,7,1,padding = 3),
                nn.BatchNorm2d(512),nn.ReLU(True),
                nn.MaxPool2d(2,2),
                MixedSCLayer(512,512,7,1,padding = 3),
                nn.BatchNorm2d(512),nn.ReLU(True),
                nn.MaxPool2d(2,2))

class HDDE9_3( HDDE9_BASE):
    def __init__(self,in_size,out_class):
        super().__init__()

        self.classifier = nn.Sequential(
```

```
            nn.Linear(in_size*512,2048),nn.BatchNorm1d(2048),
            nn.ReLU(True),
            nn.Linear(2048,2048),nn.BatchNorm1d(2048),
            nn.ReLU(True),nn.Linear(2048,out_class))

    def forward(self,x):
            y = self.feature(x)
            return self.classifier(y.view(y.size(0),-1))

class HDDE9_2( HDDE9_BASE):
    def __init__(self,in_size,out_class):
            super().__init__()

            self.classifier = nn.Sequential(
            nn.Linear(in_size*512,2048),nn.BatchNorm1d(2048),
            nn.ReLU(True),
            nn.Linear(2048,out_class))

    def forward(self,x):
            y = self.feature(x)
            return self.classifier(y.view(y.size(0),-1))

class HDDE9_2S( HDDE12_BASE):
    def __init__(self,in_size,out_class):
            super().__init__()
            self.convsqz = nn.Sequential(
                    nn.Conv2d(512,2048,1,1,padding=0),
                    nn.BatchNorm2d(2048),nn.ReLU(True))

            self.classifier = nn.Sequential(
            nn.Linear(in_size*2048,2048),nn.BatchNorm1d(2048),
            nn.ReLU(True),
            nn.Linear(2048,out_class))
    def forward(self,x):
            y = self.feature(x)
            y = self.convsqz(y)
            return self.classifier(y.view(y.size(0),-1))
```

Table 4: Training Loss and Accuracy On CALTECH-256

	HDDE9-2		HDDE9-3		HDDE9-2S	
Epoch	Train Loss/4	Accuracy	Train Loss/4	Accuracy	Train Loss/4	Accuracy
1	1.272336	0.091953	1.296351	0.075625	1.236775	0.113672
2	1.162998	0.146328	1.1941	0.121563	1.12729	0.173047
3	1.081903	0.189531	1.124216	0.170703	1.044858	0.218203
4	1.015719	0.219141	1.031879	0.214609	0.968252	0.259141
5	0.935644	0.267266	0.991913	0.239141	0.925845	0.295
6	0.894574	0.3075	0.937135	0.271406	0.863451	0.323125
7	0.865745	0.329062	0.911137	0.284062	0.837343	0.354687
8	0.83167	0.354375	0.873676	0.320156	0.800482	0.384141
9	0.80824	0.370625	0.82763	0.340469	0.774907	0.398047
10	0.75988	0.38125	0.802067	0.344297	0.779521	0.38875
11	0.740447	0.401172	0.768959	0.352344	0.736874	0.428125
12	0.727211	0.411484	0.756494	0.397656	0.736316	0.404453
13	0.698482	0.41625	0.756652	0.390469	0.694557	0.442109
14	0.70289	0.424375	0.720479	0.410234	0.693691	0.444609
15	0.688377	0.433437	0.724295	0.393437	0.685152	0.442422
16	0.669621	0.438359	0.718334	0.405625	0.661875	0.435937
17	0.654422	0.419687	0.686519	0.432578	0.668239	0.419844
18	0.654638	0.468047	0.674747	0.425781	0.64005	0.450625
19	0.652848	0.442891	0.683555	0.444609	0.639138	0.459453
20	0.626847	0.442578	0.659299	0.4475	0.615908	0.474766
21	0.628143	0.464375	0.650055	0.448594	0.626693	0.461953
22	0.600529	0.495	0.635258	0.459062	0.612314	0.473438
23	0.618979	0.455156	0.634644	0.461875	0.591948	0.502031
24	0.595544	0.490391	0.643759	0.4575	0.606047	0.410625
25	0.595263	0.465078	0.639382	0.452109	0.598789	0.487266
26	0.602039	0.506094	0.621449	0.463594	0.584803	0.511328
27	0.563718	0.523203	0.610296	0.457813	0.566332	0.514687
28	0.550391	0.544375	0.578076	0.439766	0.52905	0.553594
29	0.528523	0.533203	0.542707	0.536016	0.522948	0.577656
30	0.516095	0.509062	0.542245	0.564375	0.490098	0.569375
31	0.494375	0.588125	0.516901	0.558047	0.459255	0.630781
32	0.478008	0.597656	0.504873	0.593906	0.445357	0.6625
33	0.444806	0.637734	0.465274	0.632266	0.415244	0.669687
34	0.424125	0.661719	0.450035	0.675469	0.391407	0.712422
35	0.395492	0.696406	0.41167	0.687734	0.355696	0.725078
36	0.356541	0.733125	0.384388	0.725781	0.339713	0.733906
37	0.333843	0.751484	0.360346	0.743125	0.307329	0.778203
38	0.316675	0.776484	0.338524	0.767656	0.268842	0.813047
39	0.305807	0.791484	0.297259	0.786328	0.251511	0.83375
40	0.270329	0.805809	0.262192	0.812578	0.234622	0.845078
41	0.246138	0.822344	0.240077	0.835938	0.209833	0.861016
42	0.236792	0.840234	0.231743	0.852422	0.197723	0.878672
43	0.214403	0.857031	0.210449	0.865234	0.175501	0.890469
44	0.18536	0.869609	0.184623	0.87125	0.159794	0.896016
45	0.173079	0.8775	0.171792	0.894062	0.13458	0.908125
46	0.169135	0.881953	0.159001	0.892578	0.126098	0.913281
47	0.166386	0.890078	0.154278	0.897188	0.117184	0.917891
48	0.153694	0.893828	0.140579	0.901406	0.113734	0.918828
49	0.149883	0.893437	0.144507	0.904453	0.1209	0.922813
50	0.148773	0.894375	0.142339	0.89875	0.113186	0.918438
51	0.154007	0.892578	0.142538	0.895703	0.111709	0.917578
52	0.15312	0.897266	0.137298	0.900859	0.124566	0.920781
53	0.151751	0.898125	0.138287	0.903281	0.108883	0.918359
54	0.14351	0.894375	0.135479	0.907266	0.112298	0.920781
55	0.157251	0.895234	0.14	0.900156	0.116717	0.917891