STRUCTURE COMPOSING LAYER: A NEW TYPE OF LAYER TO DEEP AND SPEED UP CONVOLUTIONAL NEURAL NETWORKS

A PREPRINT

Guangqun Chen
Dot King Technology
dotkingtech@gmail.com

January 30, 2019

ABSTRACT

In this paper a new type of layer named Structure Composing Layer (SCLayer) aims to reduce parameters'number and decrease computation cost is proposed. SCLayer combines simple structures into complex structure with less parameters. Compared to convolution layer with filter kernel size 3×3, SCLayer reduces parameters' number to $\frac{1}{9}$ of convolution layer's. Based on VGG-16[1][1] with SCLayers a nueral network, named Snowflake Net (SfNet), is designed and evaluated on dataset CIFAR-10 and CALTECH-256[3]. Experiment results show that comparing with VGG-16 training time reduces more than 12.9%, classification accuracy increases about 0.18% on CIFAR-10, classification accuracy increases about 2.75% on CALTECH-256. During training process nonlinear phenomenon Accuracy Valley is observed. Influence of nonlinear phenomena to models' generalization is discussed.

***K*eywords** structure compose · speed up · nonlinear · classification · deep learning

1 Introduction

Deep convolutional neural networks have achieved great success in computer vision tasks. Two main reasons for the success are:

- Smaller filter kernel size:
  From AlexNet[2] to VGG the first convolution layer's filter kernel size decreases from 11×11 to 3×3. Stacking multiple convolution layers which have smaller filter kernel size, bigger receptive field can be reached using fewer parameters.
- Deep architecture:
  More layers are used. Networks become deeper. ResNet make it possible to train up to hundreds or even thousands layers[4]. Due to deep architecture richer structure features can be generated, representational ability of networks become more powerful. Richer structure features not only greatly benefit classification task, also benefit other vision tasks too.

Deep CNN have been applying to many vision tasks such as Segmentation, Human Pose Estimation, Image Generation, etc. The success in different fields prove that making network deeper works. As deep CNN have been studying further, many new networks are created, U-Net[9], HourGlass[8], Dual Path Network[10], FractualNet[11], DenseNet[12], etc. These works make deep CNN success further. CNN become deeper, models become more complicated, dealing with overfitting becomes more challenging, computation cost, memory consumption increase. Many methods and different type of layers are proposed to overcome these obstacles. Dropout layer is used to deal with overfitting problem. Batch Normalization layer is introduced to increase training speed and improve accuracy. Symmetries

[1] In this article VGG-16, VGG all refer to VGG-16 with batch normalization layers, linear layers are different for CIFAR-10 and CALTECH-256, See Section3.2.

such as rotation, multi-scaling are studied to improve performance of CNN[14][15]. Attention mechanism is used in multiple object recognition[5]. Models such as mobilenet[16] are designed to reduce complexity and target mobile device. Pruning methods are studied to compress CNN[6]. Constructing low rank basis of filters is used to speed up CNN[7]. Reinforcement learning is being adopted to automate some works currently done by human such as data augment policy selection[13]. Let machine do jobs and it will do better, should be future progress direction for CNN.

In this article SCLayer a new type of layer is proposed. SCLayer is also a good example to explain features' growth and evolution process [2].

The article is organised as follows: The analysis of CNN and it's components are given in Section 2. SCLayer and SfNet are described in Section 3. In Section 4 experiment results to evaluate SCLayer and SfNet are given. In Section 5 conclusions are drawn and Todo list is given.

2 Analysis of CNN and It's Components

2.1 Preparation Knowledge

Mixture and Spread come from physical concepts, they can help us to understand information loss. Second law of thermodynamics tell us that isolated system evolutes in the direction of increasing entropy, structures disppear, system degenerate into more disordered state, atoms or molecules mix with each other, microstates diffuse into uniform states. Liouville theorem states that probability density in phase space is constant along trajectories that obey Hamiltonian equations, quantum theory tell us information is conservative, these appears to contradict the second law, the reason behind this is that Hamiltonian equations and quantum theory are time symmetric, if the current state of system is known, from this state and physical equations both the future and the histroy can be known, so information is conservative. Particles mix, collide, entangle with each other, information spread inside the system, probability denstity spread in phase space, cause entropy increase. Although theoretically state can be calculated, if you want to go back how much works need to be done? it is irreversible, information is not lost, but it can't be retrived, the second law still hold. Mathematically phase space and feature space have no difference, they are high dimensions geometry space, mixture and spread can be apply to machine learning too.

In machine learning a equation which is broadly used is

$$y = \sum_{i=1}^{n} w_i x_i + b \tag{1}$$

Equation1 shows mixture of x_i with weight w_i give y. Variables x_i mix together, project into 1 dimension, this mixture is not reversible, there is no physical equations applied to this mixture, one value y corresponds to infinite sets of x_i, Sets of x_i can't be distinguished just with one value y, they are indistinguishable. Equation1 reduces dimensions, but it causes information loss.

Spread is not always bad. In filters' image See Figure 3 in article[2], filters show structures, structures disperse a little bit in spatial space, spread in spatial space decreases position accurary, but it helps to tolerate deformation, error and noise.

Linearity and Nonlinearity are two important aspects in science and mathematics. A simple linear equation is:

$$y = sx + t \qquad s, t \; are \; constants \tag{2}$$

We can think of Equation 2 as two operations: scale and translation. Equation2 shows: x scales s times and translates distance t, it becomes y. No matter how many times you operate scale and translation, finally it can be represent by one scale and one translation, it keep linearity unchanged. If a geometry object is operated by linear operation, it can becomes bigger or smaller, moves to different position, but it's shape will not change.

Any equations which are not linear are nonlinear, such as quadratic, cubic, exponential. Butterfly Effect is a nonlinear effect, it can be explained as a tiny error or difference can be catastrophically enlarged, this effect make system unpredictable, unstable. Opposite effect of Butterfly Effect is error or difference can be dramatically shrinked, it shrinks errors, shrinks difference, make system stable and uniform different samples. In this article I name it as Inverse Butterfly Effect.

[2]Natural selection is a optimization process, here evolution means features' growth forced by optimizer.

Global Fitting and Piecewise Function Fitting are two kinds of method to fit curve. Global fitting is more difficult to fit than piecewise function fitting. Piecewise function fitting can fit complex curve with simple piecewise functions, error can be small. Piecewise functions are used in fitting and finite element method to solve PDE. If piecewise function covers big enough area, it becomes global fitting. The advantage of piecewise function fitting against global fitting can be extended to small kernel size filters against big kernel size filters.

When use piecewise function to fit, usually simple and uniform function are used, how about fit with multi-form functions, if locally the most fitted function is used, error can be small. What form of piecewise function is most fitted locally? Let machines learn them.

Structure Generation makes natural world so complex. Atoms generate molecules, molecules generate cells, cells generate tissues, organs, animals, plants, etc. As generation process is going, number of types of objects become larger, 3 types of particles generate more than 100 types of atoms, hundred types of atoms generate uncountable types of molecules, structures become richer and more complex. There are two types of generation process, one is same type of elements repeat themself, another is different type of elements combine together become more complex object.

Degree of Freedom is a quatity to evaluate problem's complexity. In a model same amount of parameters as degree of freedom are needed to represent target, if paramters' number is less than degree of freedom it will underfit, if parameters' number is more than degree of freedom it will overfit. A three colors image has $n \times n$ pixels, $3 \times n^2$ variables can fully represent it, $3 \times n^2$ [3] is the degree of freedom of a picture without any constraints. Image is a projection of 3d objects on 2d plane, particles on surface of 3d objects are not free, there are constraints between each other, in the projected image, there are constraints between pixels too. Considering constraints, degree of freedom of a picture is less than $3 \times n^2$. How much is the degree of freedom of a square picture whose resolution is $n \times n$? Degree of freedom of a picture should be

$$3 \times n^2 > f_{real} > 3 \times n^1$$

Write it as:

$$f_{real} = c \times n^{\lambda} \qquad 2 > \lambda > 1 \qquad c = constant. \tag{3}$$

Equation3 is helpful to estimate how degree of freedom change when images' resolution change.

In following parts I will use degree of freedom of pictures' without any constraints to estimate the complexity of problem. The parameters' number of CNN models are more than pictures' degree of freedom. There is room to reduce parameters.

2.2 CNN and It's Components

Deep CNN compose of layers. The most useful layers are:

Convolution layer compose of filter bank, filter is a piecewise function, it can be any form. By moving this piecewise function on inputs, operations such as smoothing, derivative can be done, inputs can be decomposed into different components.

For 2d convolution layer, there are n inputs, function of filter j in filter bank can be described as:

$$out_j = \sum_{k=1}^{n} weight(j,k) * input(k) + bias_j \tag{4}$$

where $*$ is 2D cross-correlation operator.

Rewrite Equation4

$$out_j = \sum_{k=1}^{n} a_k \times \frac{weight(j,k)}{a_k} * input(k) + bias_j$$

$$out_j = \sum_{k=1}^{n} a_k \times weight^{'}(j,k) * input(k) + bias_j \qquad weight^{'}(j,k) = \frac{weight(j,k)}{a_k} \tag{5}$$

Equation5 can be explained as linear combination of n convolution operations.

[3] pixel can be treated as particle. In mechanics N free particles degree of freedom is $6 \times N$, fix particles' position, degree of freedom decrease to $3 \times N$ which correspond to 3 velocity variables. Replace velocity variables with color variables, degree of freedom of N pixels is $3 \times N$

ReLU layer is nonlinear layer.

$$ReLU(x) = max(0, x)$$

Any value less than 0 is replaced by 0. It functions as threshold, threshold value is decided by bias of convolution layer before it, it can be adjusted and learned.

MaxPool layer keep max value inside a window, it tolerates deformation and uniforms difference of input in spatial space locally.

Dropout layer is effective technique for regularization and prevent the co-adaptation of neurons. In a image parts of object can be unclear, missing, because of occlusion, clutter, noise. Dropout layer can be thought as simulation of missing parts of objects.

BatchNormalization layer can accelerate deep network training, increase accuracy.

3 SCLayer and SfNet

In order to clearly explain the ideas of SCLayer, f-image is used to represent feature map, image is used to represent input picture, and pixel represent point on image and f-image, pixel's value can be vector or scalar, image and f-image are rectangle areas in 2D plane filled with pixels, spatial space is used to refer to 2D plane.

Features are split into two groups according to structure generation process:

Smear Feature is generated by simple feature repeating itself in spatial space. Smear feature changes gradually, distributes in spatial space continually and densely, can be smoothed, average value can be used to represent it. Examples of smear features are sky color, water color, skin color .

Structure Feature is generated by combining different simple features together. Structure feature usually has edges, change sharply, tend to distribute sparsely in spatial space, average and smooth will destroy sharp changes of structures, will smear details of structures.

3.1 SCLayer

The design of SCLayer is based on follow analysis of convolution layer:

- Convolution Filters decompose image or f-image into different features, pixel value on f-image represent how much the patch around this pixel match structure, structure is located at highest value pixel's position, neighbor pixels around this pixel partially match structure.
- Pixels on f-image for structure feature distribute sparsely.
- If fit image or f-image with multiform piecewise functions, piecewise functions should be placed at position where pixels have maximum value, neighbor pixels can be ignored.
- In Equation5 cross-correlation operation is linear combination too, it mixes anchor point's value with it's neighbors'value together. For smear feature mixture can average value, get more precise representation, for structure feature mixture will destroy structure. Another linear combination in Equation5 can be viewed as combining simple structures into more complex structure.

Based on these analysis, cross-correlation operation is replaced by maxpool.

$$out_j = \sum_{k=1}^{n} a_k \times maxpool(input(k)) + bias_j \tag{6}$$

Equation6 is explained as combining structures into new structure. The filter is named as Structure Composing Filter. The layer with Structure Composing Filters is SCLayer.

Figure1 show the functional block of SCLayer.

In order to keep some amount of smear features MixedSCLayer is proposed. See Figure2

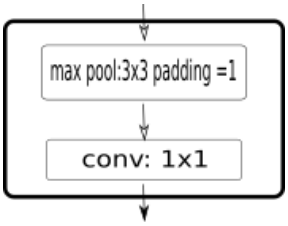

Figure 1: SCLayer Functional Block

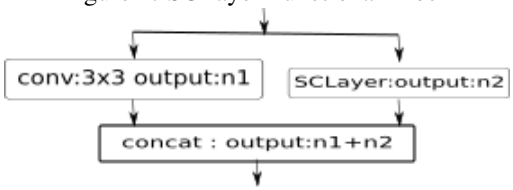

Figure 2: Mixed SCLayer Functional Block

3.2 SfNet

In order to verify SCLayer a network is needed. SfNet is based on VGG-16. Layers of scale 4 and scale 5 are replaced with MixedSCLayers and SCLayers. When enter into new scale, the first layer is replaced by MixedSCLayer, other layers are replaced by SCLayers.

The reason to replace layers of scale 4 and scale 5 is based on this hypothesis: simple structures are generated when image resolution reduce 8 times. Although patch size 8×8 is not bigger enough, receptive field of f-image is bigger than 8×8. From scale 4 SCLayer can be used. 16 smear features are used for MixedSCLayers. I assume smear features should be much less than structure features, number 16 is selected and test on CIFAR-10, and same number of smear features are used on CALTECH-256. For SCLayer Kernel size is 3×3, stride is 1.

Table 1: VGG and SfNet Architecture

Scale		VGG	SfNet
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		$[3 \times 3, 512, B, R] \times 3$	$[SC(3 \times 3, 512), B, R] \times (1 + 2)$
		2×2 max pool stride 2	
5		$[3 \times 3, 512, B, R] \times 3$	$[SC(3 \times 3, 512), B, R] \times (1 + 2)$
		2×2 max pool stride 2	
Linear Layers	CIFAR-10	$[FC - 1024, B, R] \times 2$	
		FC-10	
	CALTECH-256	$[FC - 2048, B, R] \times 2$	
		FC-256	

B: BatchNorm layer. R:ReLU layer. SC: SCLayer or MixedSCLayer. (1+2): 1 MixedSCLayer + 2 SCLayer. MixedSCLayer convolution channels are 16.

4 Experiments

In order to verify SfNet and SCLayer, experiments are designed very carefully. How to choose dataset is the most important part of these experiments. Dataset can demonstrate the idea of structure composing should be chosen, dataset on which experiments can be done without requiring high computation capability should be chosen. dataset's complexity2.1 should be considered. If experiment's results show SfNet's accuracy doesn't drop or drop a little bit, but trade-off of running time and accuracy is good, conclusion of SCLayer and SfNet's success can be drawn.

4.1 Experiment Set up

Two datasets CIFAR-10 and CALTECH-256 were chosen. CIFAR-10's small image size make training very fast, it's possible to repeat experiments many times. Small image size 32×32 is not good for structure composing hypothesis, when image size reduces 8 times, f-image size is 4×4, small f-image size is against requirement of struture features distributing sparsely, when image size reduces 16 times f-image size become 2×2, few parameters model will couple features together, model's flexibility become lower. coupled features' model is more difficult to train than decoupled features' model. Detail information is lost in small size image due to mixture, number of smear features probably increase, these are bad side for SCLayer's construction ideas. How well will the model be on CIFAR-10? Model's success on simple dataset doesn't mean it will success on complex dataset, if model's representation capability is low, complex dataset may cause underfitting. CIFAR-10's degree of freedom is $3 \times (32^2)$. Higher degree of freedom dataset is needed to evaluate SCLayer. Is SfNet's capability enough to represent higher degree of freedom? CALTECH-256 has 256 object categories. 50 training samples and 20 test samples(one group from pre-split groups) were chosen to train models. Fewer samples make it possible to use Kaggle cloud GPU to train. In order to compare experiment results of SfNet with VGG-16's, same hyperparameters, same training method and same number epochs were used. For data augment only random horizontal flip, and normalization were used, on CALTECH-256 in order to get fixed input size 224×224 random resized crop was used. On CIFAR-10, training schedular one-cycle training [17] was used. code is based on fastai library with some modification, batch size: 128, trained 30 epochs, each model trained 10 times. On CALTECH-256 training schedular is linear annealing, batch size: 128, in order to use batch size 128 gradient accumulation was used, batch size for GPU: 32, 128 batch size need 4 times gradient accumulation, each model trained 52 epochs, each model trained 1 time.

4.2 Experiment Results

Table 2 lists Accuracy and Accuracy Improvement. Table3 records training time[4] and accuracy of VGG and SfNet on CIFAR-10. VGG's time fluctuation in Table3 is more than 60 seconds, it is beyond normal range, training time can be split into two groups, slower group and faster group. Training time of SfNet can be split into two groups too. Big flunctuation implies that models were trained in two speed environments, one is little faster than another. Trainings were done in kaggle web site with cloud GPU. They were not done in one shot, it's possible trainings were running in different conditions or at different GPU at different time. Data show: 6 times of VGG's training were running in slower condition, 4 times of SfNet's training were running in slower condition, if average data without any correction, both increase speed up percentage of SfNet. The purpose of collecting training time is to demonstrate SCLayer's and SfNet's speed up. Accurate training time depend on how many SCLayers are used in models, can be calculated precisely. To correct this system error, low bound of time reduced and experiment time reduced are calculated. No matter this error is caused by system error or by statistic error, real percentage of time reduced should be in between low bound time reduced and experiment time reduced. Low bound time reduced is calculated using average training time of VGG's data which are less than 1570 seconds-the average of faster group as VGG's training time, SfNet's average training time is calculated with all experiment data. Experiment time reduced is calculated by averaging data without any correction.

SfNet's percentage of time reduced is:

$$TimeReduced_{low\_bound} = 12.9\%$$
$$TimeReduced_{experiment} = 14.9\%$$
$$14.9\% \geq TimeReduced_{real} > 12.9\%$$

Table4 and Table5 give training loss and accuracy of each epoch on CIFAR-10 and CALTECH-256, CIFAR-10's data is randomly selected 1 from 10 experiment data. Figure3 and Figure4 show training loss curve and accuracy curve, there are some interesting phenomena which are discussed in Section4.3. Training loss in tables and figures of CALTECH-256 are loss divided by 4, this is because of gradient accumulation loss value divided by 4 were recorded, this doesn't influence any conclusion, only curve's shape and relationship of curves matter to the conclusions. If readers want to know CALTECH-256's loss don't forget to multiply 4.

4.3 Discussion

4.3.1 Accuracy Valley Phenomenon

In Figure3 on CALTECH-256 although VGG's accuracy curve has fluctuation, fluctuation is small, the main trend is all the way up, behaves quasi-linear, it matches VGG's traing loss curve which is all the way down. SfNet's loss

[4]Inference time is too short, training time is used for study.

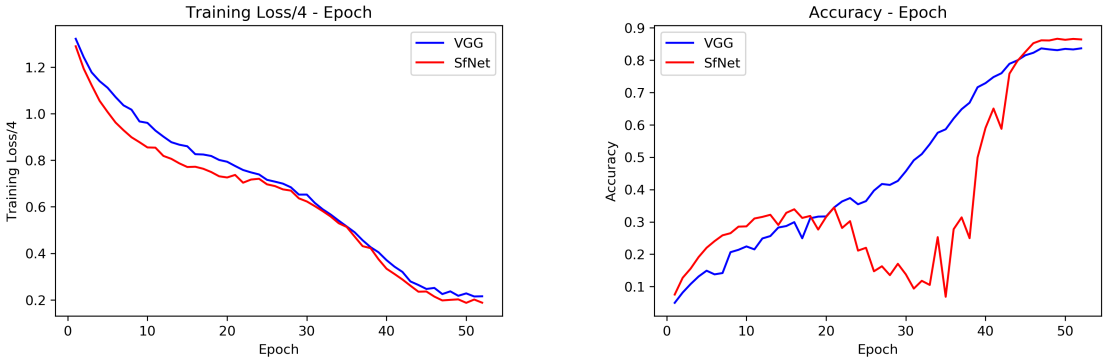

Figure 3: VGG and SfNet Training Loss and Accuracy Comparison on CALTECH-256.

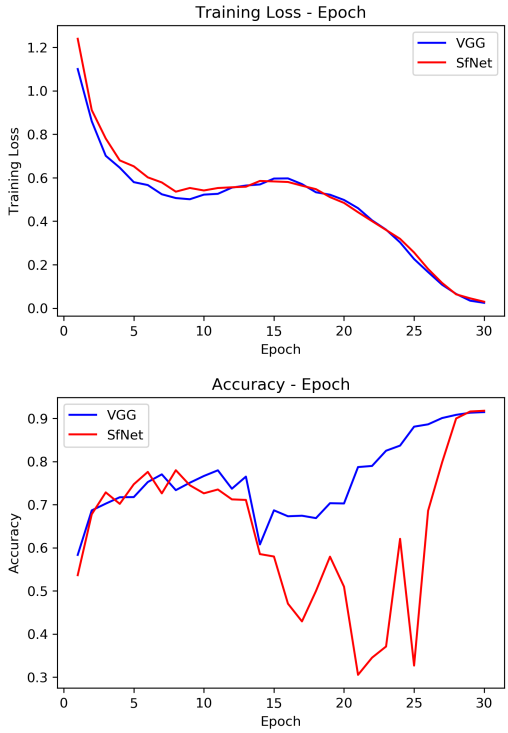

Figure 4: VGG and SfNet Training Loss and Accuracy Comparison on CIFAR-10.

Table 2: Accuracy and Accuracy Improvement of VGG and SfNet

Dataset	Accuracy of VGG-16	Accuracy of SfNet	Accuracy Improvement
CIFAR-10	91.32%	91.5%	0.18%
CALTECH-256	83.625%	86.375%	2.75%

Table 3: 10 Times Training Time and Accuracy On CIFAR-10

	VGG-16		SfNet	
Times	Train Time(second)	Accuracy	Train Time(second)	Accuracy
1	1577	0.9104	1315	0.9164
2	1578	0.9146	1314	0.9134
3	1518	0.9146	1312	0.9123
4	1516	0.9139	1313	0.9134
5	1514	0.9146	1313	0.9177
6	1516	0.9099	1311	0.9171
7	1573	0.9165	1331	0.9142
8	1571	0.9148	1330	0.9131
9	1571	0.9139	1332	0.9153
10	1570	0.9087	1330	0.9166
Average:	1550	0.9132	1320	0.9150

Table 4: Training Loss and Accuracy On CIFAR-10

	VGG-16		SfNet	
Epoch	Train Loss	Accuracy	Train Loss	Accuracy
1	1.100122	0.5834	1.239298	0.5364
2	0.860124	0.6866	0.911267	0.6776
3	0.700860	0.7023	0.780454	0.7284
4	0.646266	0.7171	0.679718	0.7018
5	0.579767	0.7175	0.652583	0.7472
6	0.566333	0.7524	0.601780	0.7759
7	0.523926	0.7702	0.578357	0.7261
8	0.506523	0.7336	0.535902	0.7798
9	0.501077	0.7508	0.552934	0.7449
10	0.522267	0.7664	0.541448	0.7262
11	0.526031	0.7796	0.552678	0.7351
12	0.554514	0.7369	0.556112	0.7122
13	0.564115	0.765	0.558704	0.7109
14	0.569203	0.6077	0.585268	0.5853
15	0.596227	0.6868	0.583023	0.5797
16	0.596998	0.673	0.580476	0.4703
17	0.570715	0.6743	0.563354	0.4292
18	0.5331	0.6687	0.547381	0.4997
19	0.521963	0.7033	0.510906	0.5793
20	0.497751	0.7027	0.484276	0.5097
21	0.460091	0.7871	0.441383	0.3053
22	0.404032	0.7897	0.400366	0.3451
23	0.361152	0.825	0.359721	0.371
24	0.302598	0.8369	0.31891	0.6207
25	0.226056	0.8808	0.256153	0.3268
26	0.16598	0.8862	0.180338	0.6859
27	0.107846	0.9008	0.115959	0.7981
28	0.065549	0.9082	0.063677	0.8998
29	0.034429	0.9132	0.045304	0.916
30	0.024602	0.9146	0.029650	0.9177

Table 5: Training Loss and Accuracy On CALTECH-256

	VGG-16		SfNet	
Epoch	Train Loss/4	Accuracy	Train Loss/4	Accuracy
1	1.321664	0.049922	1.289359	0.075469
2	1.242926	0.081797	1.193261	0.127344
3	1.177199	0.108047	1.121924	0.155781
4	1.139434	0.131406	1.054908	0.191406
5	1.111174	0.149219	1.007099	0.220078
6	1.07127	0.137891	0.961842	0.240547
7	1.035505	0.141953	0.928787	0.258516
8	1.017021	0.206406	0.898571	0.265078
9	0.966407	0.21375	0.8774	0.285313
10	0.960288	0.224219	0.854855	0.286172
11	0.927241	0.215078	0.853897	0.310469
12	0.901423	0.248828	0.818462	0.315469
13	0.877376	0.256250	0.805783	0.321953
14	0.866563	0.282578	0.786488	0.290469
15	0.859791	0.287187	0.770808	0.327969
16	0.825866	0.299219	0.771879	0.338984
17	0.824479	0.249375	0.76325	0.312266
18	0.818137	0.311172	0.749189	0.318437
19	0.801011	0.316328	0.730862	0.276328
20	0.793377	0.316875	0.725784	0.315391
21	0.775164	0.344609	0.736712	0.344453
22	0.757903	0.363203	0.703390	0.281250
23	0.747579	0.373594	0.716911	0.302344
24	0.738825	0.354453	0.720253	0.211250
25	0.715711	0.364219	0.696391	0.22
26	0.707876	0.396719	0.688808	0.147422
27	0.699341	0.417109	0.674703	0.162891
28	0.683572	0.414297	0.669408	0.135469
29	0.652274	0.426641	0.636157	0.170547
30	0.652334	0.456406	0.622759	0.137891
31	0.616565	0.490391	0.602708	0.093906
32	0.588990	0.509453	0.581074	0.117969
33	0.566424	0.540078	0.557861	0.105234
34	0.540403	0.575469	0.529233	0.252812
35	0.515421	0.586172	0.513280	0.068672
36	0.490407	0.619453	0.471845	0.2775
37	0.456062	0.648203	0.430779	0.313984
38	0.426462	0.668281	0.4222	0.249609
39	0.404337	0.716016	0.374314	0.498828
40	0.370718	0.728828	0.333877	0.590547
41	0.342697	0.7475	0.311149	0.650234
42	0.320031	0.759375	0.287453	0.587578
43	0.279197	0.788438	0.260885	0.757969
44	0.263976	0.799063	0.235154	0.797266
45	0.246928	0.814844	0.236037	0.825313
46	0.251367	0.8225	0.214192	0.852109
47	0.224978	0.836094	0.197659	0.861016
48	0.236679	0.832969	0.200228	0.860547
49	0.217581	0.830547	0.202241	0.865859
50	0.22792	0.834609	0.187346	0.862656
51	0.21466	0.832812	0.201548	0.865391
52	0.21545	0.83625	0.187967	0.86375

curve shape is similar to VGG's, value is less than VGG's. SfNet's accuracy curve is up in the beginning, higher than VGG's, then it turns down to very lower value, then turns back up, rises sharply and is higher than VGG's again, a valley-Accuracy Valley appears.

In Figure4 on CIFAR-10, SfNet's accuracy is rising in the begining, then drops sharply, is lower than all the values before, forms a double bottom then turns back up, rises sharply, a W in the diagram appears, although shape is different from the one on CALTECH-256, it is called Accuracy Valley too. Accuracy and training loss in Accuracy Valley's area diverge each other, loss goes lower, accuracy doesn't go up instead it goes down. Accuracy Valley is not fluctuation. It is not caused by overfitting, models haven't fitted training samples well yet, training loss is still high. Is this due to SfNet's sensitivity or some singular values in parameters or gradient?

4.3.2 Nonlinear Crisis and Models' Generalization

High similarity of SfNet's accuracy curve to stock chart[5] in Figure3 make me realize Accuracy Valley is nonlinear phenomenon. Nonlinear phenomenon need to be explained from nonlinear point of view. SfNet's accuracy value at bottom of valley is about 6.8%, See Table5 at epoch 35. It means unpredictable, more than 93% predictions are wrong, Why loss and accuracy diverge so huge?

It's Butterfly Effect. The difference between training samples and test samples are catastrophically enlarged. Butterfly Effect brings unpredictability, Butterfly Effect brings crisis, how's generalization of models? Neural network is nonlinear network, if Butterfly Effect exists how do you know model works well at working point[6]? Even if test samples work well at working point, what if change to other samples? how can you assume the divergence between them is not enlarged? It is nonlinear crisis for neural network.

4.3.3 The Universality of Accuracy Valley Phenomenon

Accuracy Valley phenomenon clearly exists in CALTECH-256's diagram of SfNet. Same dataset VGG doesn't show Accuracy Valley phenomenon. There is Accuracy Valley in CIFAR-10's diagram of SfNet, shape is different from the one on CALTECH-256. For VGG on dataset CIFAR-10 there is Accuracy Valley, a shallow version is floating on top of the diagram(blue color), shape is similar to SfNet's shape on CALTECH-256, details are different (5 wave counts in rising phase disppears),See Figure4.

Similar curve shape different model tell us Accuracy Valley's universality. It implies a conclusion: Accuracy Valley is a popular phenomenon, although what trigger it is unknown, when parameters'value fall into some range, Accuracy Valley appears. Same models (SfNet are not toatally same, linear layers are different) have different shape when work on different dataset. Probably Accuracy Valley exists in all nueral networks.

4.3.4 Inverse Butterfly Effect: A Solution to Nonlinear Crisis

Don't ignore a tiny hole, if it is a wormhole, it will lead you into another universe.

Analysis in Section4.1 tells small image size is not good for SfNet, Figure4 proves this, during training SfNet's loss value is all the way higher than VGG's.

The tiny increase of SfNet's accuracy on CIFAR-10 is another problem to bother me. It is so small. Even many times of training can't make me feel confident about this increase. I averaged many times training to exclude fluctuation. Accuracy do increase a little bit. Although accuracy is little higher, but SfNet's training loss is little higher than VGG's, See Figure4. How to explain this? Overfitting? SfNet has fewer parameters, VGG's overfitting causes it has smaller loss but lower accuracy? VGG's data doesn't show any overfitting. Compared to VGG higher loss and higher accuracy show up in other experiment results too.

My explanation of this phenomenon is Inverse Butterfly Effect, Difference between test samples and training samples shrink. If at working point, model is quasi-linear, or has Inverse Butterfly Effect, then you can feel confident about prediction. Inverse Butterfly Effect is a solution to nonlinear crisis.

[5] According to Wave Principle[18] 3 of 5 waves have been formed, wave 4th is forming, in wave 3th (sharp rising one) a clear 5 sub-waves' patten appears. Curiosity make me wonder if continuing training will 5th wave appear? Learning process itself reflects model's property it need to be studied.

[6] On training curve, the point at which parameters are used for inference

As for SfNet on CALTECH-256 at working point, I can't prove it works in quasi-linear state or Inverse Bufferfly Effect state or not. From practical view just like all other models do, I assume if test samples' results are good, then the model can be used for other samples.

5 Conclusions and Future Works

Experiment results and construction method of SfNet clearly show that:

- SfNet has fewer parameters, runs faster than VGG. For classification task, SfNet has higher accuracy than VGG.
- SCLayer can be used as a basic element to construct networks to speed up and deep CNN.
- Accuracy Valley phenomenon exists in both VGG and SfNet.
- A strong guess is given bravely: Accuracy Valley is a popular phenomenon, it exists in any nueral network model. At some condition, it will show up.
- Nonlinear models' behavior diverge hugely when parameters' value is different.
- Inverse Butterfly Effect can shrink samples' divergence improve accuracy.
- Some models' learning process (optimising process) itself try to reproduce natural developing process or growth process.

Items need to be studied further are:

- Apply SCLayer to other networks such as ResNet.
- Apply SCLayer to other computer vision tasks.
- Training SfNet on dataset ImageNet.
- How is SCLayer affected by stride value and kernel size?
- Instead of linear combination, compose structures with boolean operation + normalization.
- Can learning process(optimising process) mimic stock growth or other time series event with some models?
- Can learning process itself be used to predict?

Hope more useful models can be designed with SCLayer, and SCLayer itself can be tested broadly.

References

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

[2] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems (NIPS)*, 2012.

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

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

[5] Jimmy Lei Ba,Volodymyr Mnih, and Koray Kavukcuoglu. Multiple Object Recognition with Visual Attention. *arXiv:1412.7755v2* 23. Apr,2015.

[6] Jian-Hao Luo, and Jianxin Wu. An Entropy-based Pruning Method for CNN Compression. *arXiv:1706.05791V1* 19. Jun, 2017.

[7] Max Jaderberg, Andrea Vedaldi and Andrew Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. *arXiv:1405.3866* 15. May, 2014.

[8] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked Hourglass Networks for Human Pose Estimation. *arXiv:1603.06937* 26. Jul, 2016.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597* 18. May, 2015.

[10] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, Jiashi Feng. Dual Path Networks. *arXiv:1707.01629* 6. Jul, 2017.

[11] Gustav Larsson, Michael Maire, Gregory Shakhnarovich. FractalNet: Ultra-Deep Neural Networks without Residuals. *arXiv:1605.07648* 26. May, 2017.

[12] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993* 28. Jan, 2018.

[13] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V. Le. AutoAugment: Learning Augmentation Policies from Data. *arXiv:1805.09501* 9. Oct, 2018.

[14] Junying Li, Zichen Yang, Haifeng Liu, Deng Cai. Deep Rotation Equivariant Network. *arXiv:1705.08623* 28. Feb, 2018.

[15] Fisher Yu, and Vladlen Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *Published as a conference paper at ICLR* 2016.

[16] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861* 17. Apr, 2017.

[17] Leslie N. Smith. A Disciplined Approach To Neural Network Hyperparameters:Part 1-Learning rate,Batch size,Momentum, And Weight Decay. *arXiv:1803.09820v2*, 24. Apr, 2018.

[18] Robert R. Prechter, JR. R. N. Elliott's Masterworks The Definitive Collection *ISBN:0-932750-37-0 Published by New Classics Library* September 1994

SCLayer and SfNet Implementation with Pytorch in My Experiment

```
class MixedSCLayer(nn.Module):

    def __init__(self,input_ch,output_ch,kernel_size,stride,padding=1):
        super().__init__()
        self.avg_conv = nn.Conv2d(input_ch,16,kernel_size,stride,padding)

        # maxpool_bias can be removed, it is needless,
        # sc_conv- use convolution layer, default bias is enabled
        #but my experiment code have this  #maxpool_bias, so I keep it

        self.maxpool_bias = torch.randn(1,requires_grad = True,
        device=torch.device("cuda" if torch.cuda.is_available() else "cpu"))
        self.maxpool = nn.MaxPool2d(kernel_size,stride,padding )
        self.sc_conv=nn.Conv2d(input_ch,output_ch-16,1,stride)

    def forward(self,x):
        y = self.avg_conv(x)
        z = self.maxpool(x)
        z += self.maxpool_bias
        return  torch.cat([y,self.sc_conv(z)],1)

class SCLayer(nn.Module):

    def __init__(self,input_ch,output_ch,kernel_size,stride,padding=1):
        super().__init__()

        # maxpool_bias can be removed, it is needless,
        #my experiment code have this  #maxpool_bias, so I keep it in this code

        self.maxpool_bias = torch.randn(1,requires_grad = True,
        device=torch.device("cuda" if torch.cuda.is_available() else "cpu" ))
        self.maxpool = nn.MaxPool2d(kernel_size,stride,padding )
        self.sc_conv=nn.Conv2d(input_ch,output_ch,1,stride)
```

```
    def forward(self,x):
        z = self.maxpool(x)
        z += self.maxpool_bias
        return  self.sc_conv(z)

# in_size: 1 for image size ${32\times 32}
# image size 224 x 224  in_size: 7 x 7 = 49
#CIFAR-10 out_class: 10
#CALTECH-256 out_class: 256

class SfNet(nn.Module):

    def __init__(self,in_size,out_class):
        super().__init__()

        # when enter into new scale, 1 mixed layer
        #followed by no-mixed layers

        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(256,512,3,1),nn.BatchNorm2d(512),nn.ReLU(True),
                SCLayer(512,512,3,1),nn.BatchNorm2d(512),nn.ReLU(True),
                SCLayer(512,512,3,1),nn.BatchNorm2d(512),nn.ReLU(True),
                nn.MaxPool2d(2,2),
                MixedSCLayer(512,512,3,1),nn.BatchNorm2d(512),nn.ReLU(True),
                SCLayer(512,512,3,1),nn.BatchNorm2d(512),nn.ReLU(True),
                SCLayer(512,512,3,1),nn.BatchNorm2d(512),nn.ReLU(True),
                nn.MaxPool2d(2,2))
        self.classifier = nn.Sequential(nn.Linear(in_size*512,1024),nn.BatchNorm1d(1024),
        nn.ReLU(True),
        nn.Linear(1024,1024),nn.BatchNorm1d(1024),nn.ReLU(True),
        nn.Linear(1024,out_class))
        # next parts is for caltech-256
        #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))
```