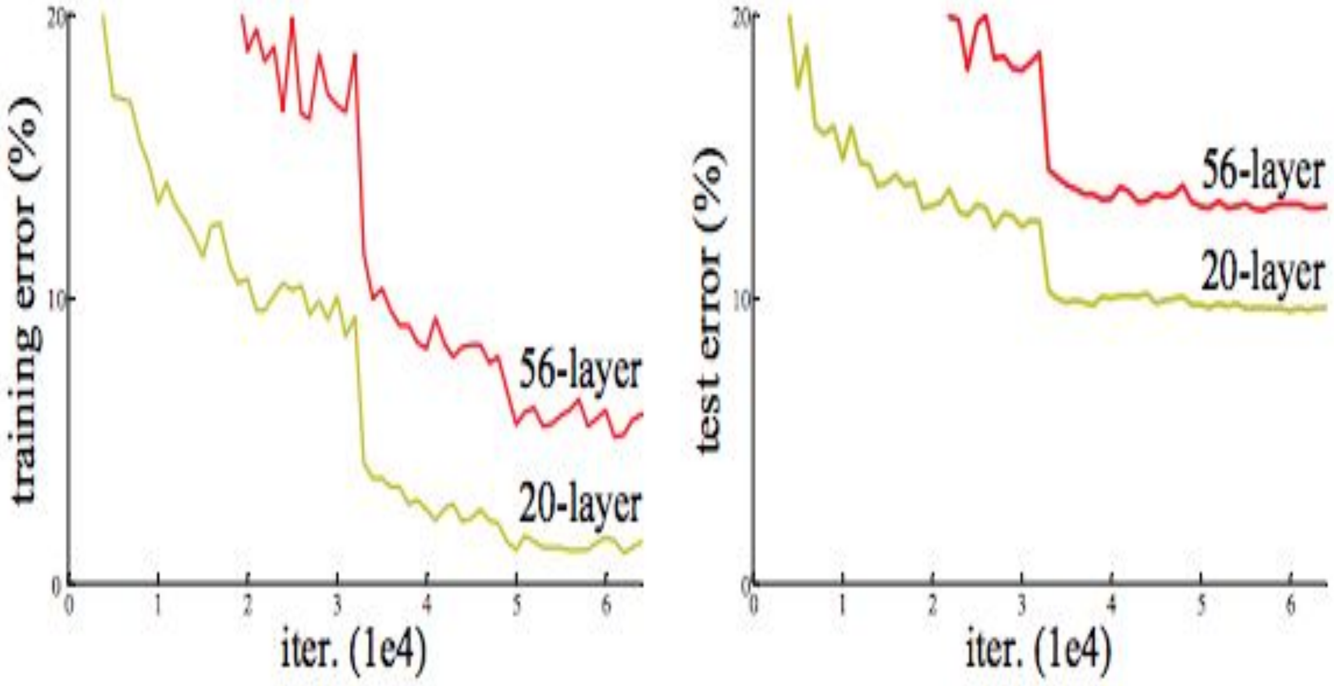
ResNet

Contents

1. Problem
2. Solution
3. Residual block
4. Why resnet work
5. Pattern in this architecture
6. Comparison with plain CNN

Problem

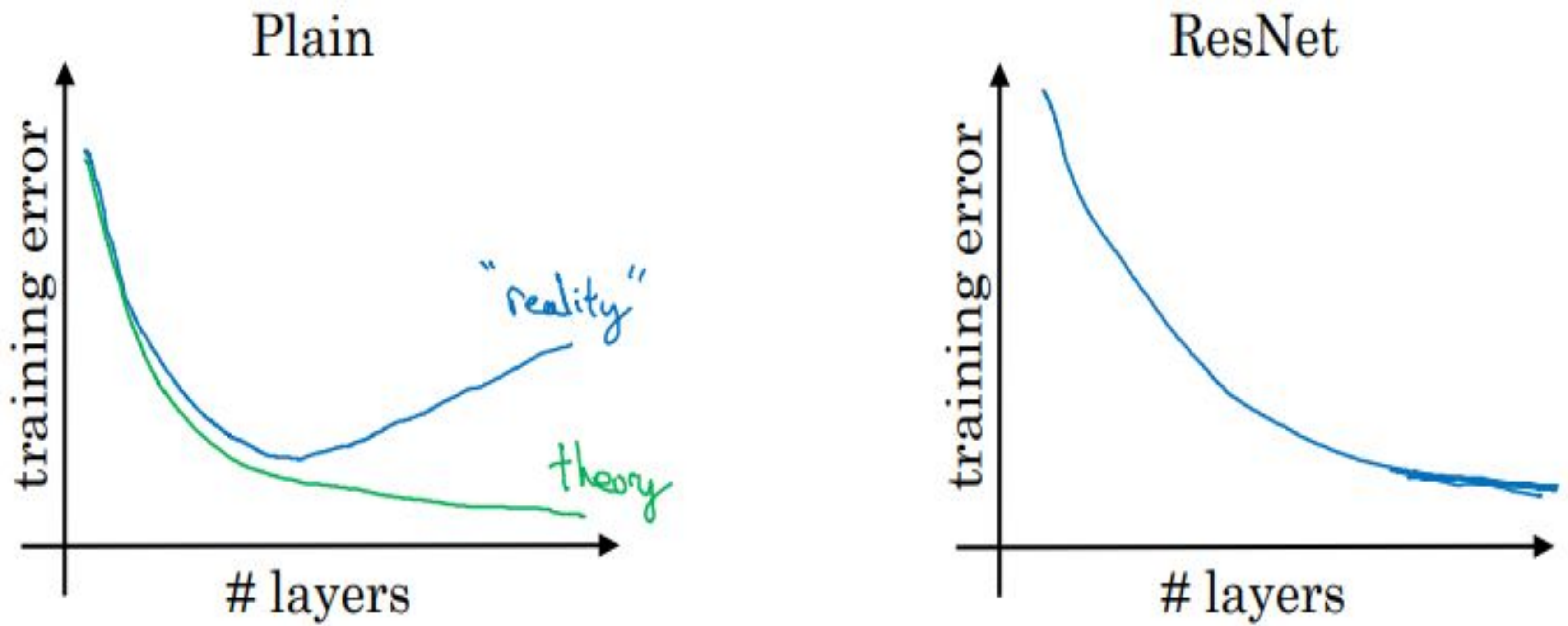
1. It’s Difficult to train very deep neural networks.
    a. Time consuming
    b. Hurts the performance
2. It’s because of vanishing and exploding gradient problem.



Source: towardsdatascience

Solution

1. Performance can be enhanced by residual block
2. Even if we add a residual block it guarantees that it may or maynot improve the performance but it definitely won’t hurt the performance
3. It’s though skip connections:
    a. Adds output from previous layers to layer ahead.

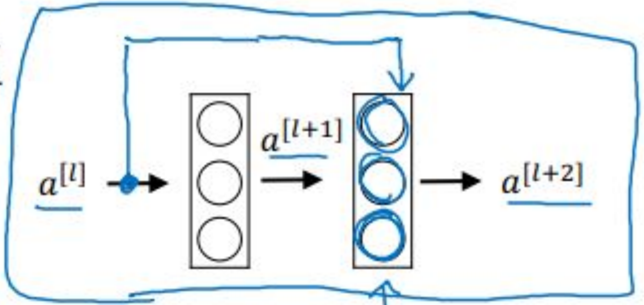


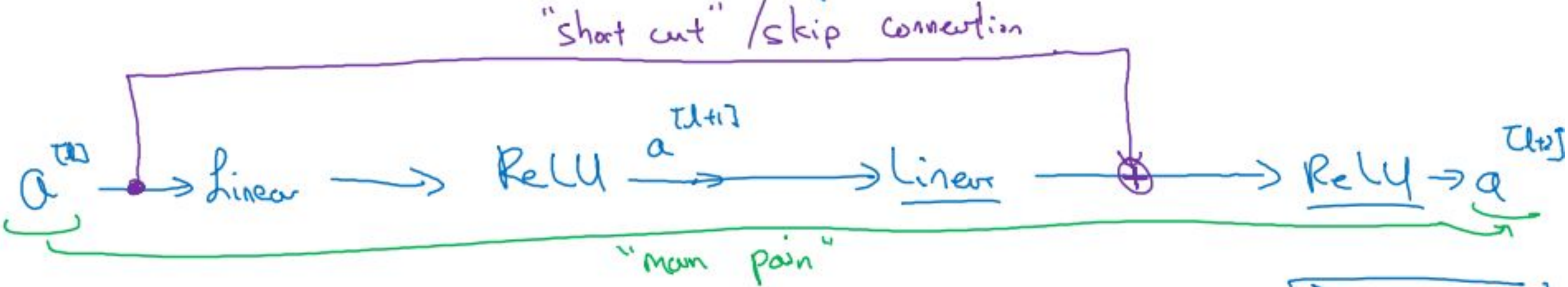
Source: coursera

Residual block

1. Add outputs of previous layers to layers ahead of that layer.
2. 2-layer: Adds outputs of layer x to layer x+2.
3. 3-layer: Adds outputs of layer x to layer x+3.
4. This x is add to outputs of 2nd layer before relu.

Residual block





$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$ $a^{[l+1]} = g(z^{[l+1]})$ $z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$ $\boxed{\cancel{a^{[l+2]} = g(z^{[l+2]})}}$

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

Source: coursera

Dimension problem for residual block

1. Used same padding conv layers.
2. When dimensions of a[l] and a[l+2] are not equal(in this architecture dimensions are halved.)
3. A[l] is multiplied with matrix W_s. i.e z[l+2] and W_x.A[l] have same dimensions.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

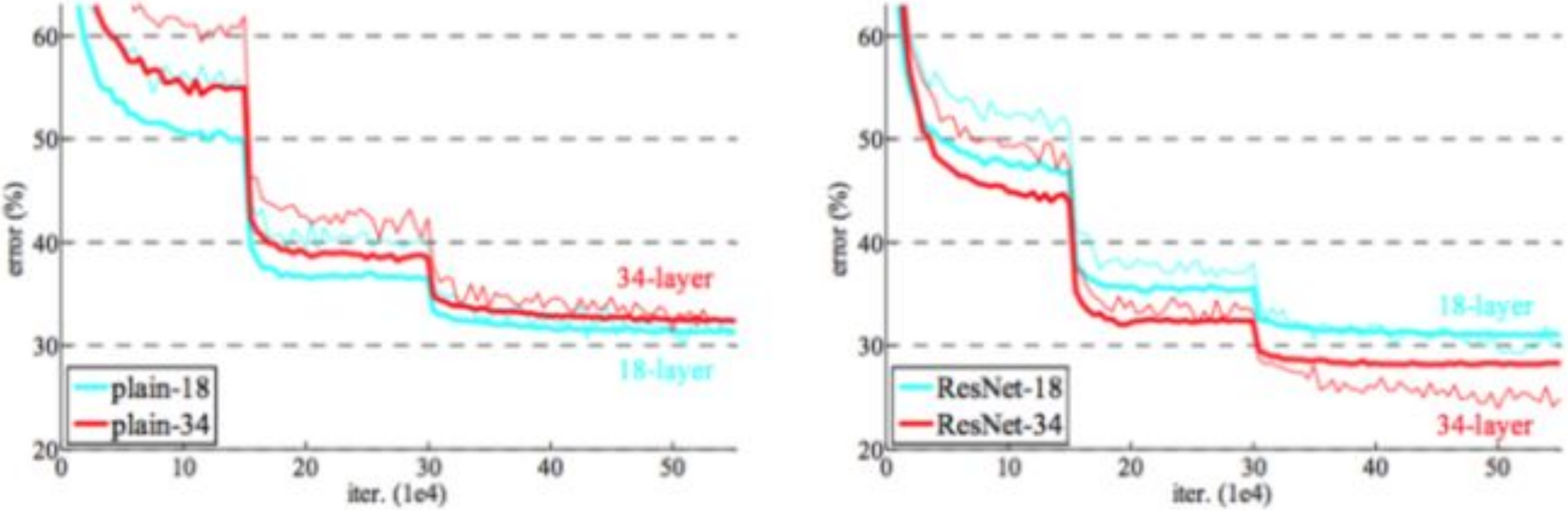
Why resnet work

1. For residual blocks it’s very easy to learn identity function.
   a. Weights are zero and bias is zero.
   b. With this characteristic it has a strict baseline it can output same feature maps of before or it can be a better one’s.

Pattern

1. As we go deeper
   a. height and width of feature maps decreases.
   b. Number of channels increases.
2. When feature maps size is halved then number of channels doubled.
3. Skip connections are arranged in 2-layer blocks

Comparison



Source: coursera

Comparison

1. For plain CNN 34-layers CNN must have less error than 18-layer CNN but it’s not.
2. While with residual blocks It’s true
   a. 18-layer Resnet performance is similar to 18-layer plain CNN
   b. But 34-layer Resnet outperformed 34-layer plain CNN.
   c. This shows adding more layers may improve but won’t hurt the performance.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Vanishing gradient:

1. This problem occurs when we backprop.
2. The gradients w.r.t model parameters would become smaller and smaller when we go from output layers to input layers.
3. **Reason:**
   a. in chain rule we multiply these derivatives right.
   b. So every derivative is much smaller than 1 then multiplying all would result a lower number.
4. This occurs mainly because of sigmoid activation function using at every layer.

Resnet:

1. **Special in resnets/ Benefits of using resnet over other learning algorithm:**
   a. We can train large deep neural networks without hurting performance.
   b. This avoids vanishing gradient problem.
2. **Reason:**
   a. While backprop gradients passes through this identity mapping.(local gradient between input and output layer is 1)
   b. Hence when multiply gradients at input layer
      i. With the help of identity mapping we could preserve the gradients till the output layer in the input layer.
   c. This identity mapping is used to preserve the gradient.
3. During backprop we back right so while multiplying gradients(<<1) would result in much smaller value so if we have a skip connection then with deteriorating the gradient through residual mapping we can add directly to the previous layers.

SqueezeNet

Contents

1. Introduction
2. Design strategies
3. Fire Module
4. Squeezenet Architecture
5. Evaluation of squeezenet

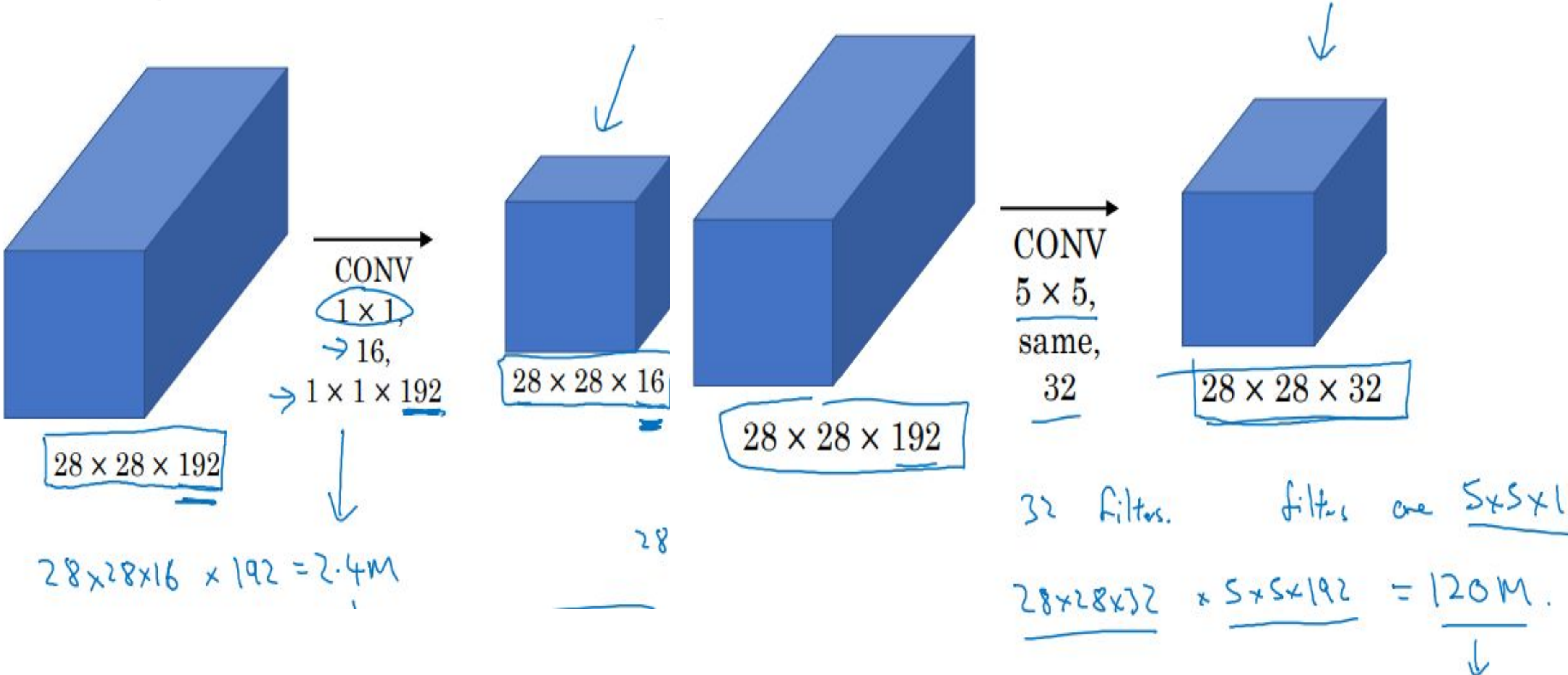
Introduction

1. This architecture was developed by researchers at DeepScale, University of California, Berkeley, and Stanford University
2. Designed mainly to have a architecture with small in storage space.

Design strategies

1. Replace 3×3 filters with 1×1 filters
2. Decrease the number of input channels to 3×3 filters
3. Downsample late in the network so that convolution layers have large activation maps

Strategy1



28 × 28 × 192
CONV
1 × 1,
16,
1 × 1 × 192
28 × 28 × 16
28x28x16 x 192 = 2.4M
28 × 28 × 192
CONV
5 × 5,
same,
32
28 × 28 × 32
32 filters.
filters are 5x5x1
28x28x32 x 5x5x192 = 120M.

3*3 conv

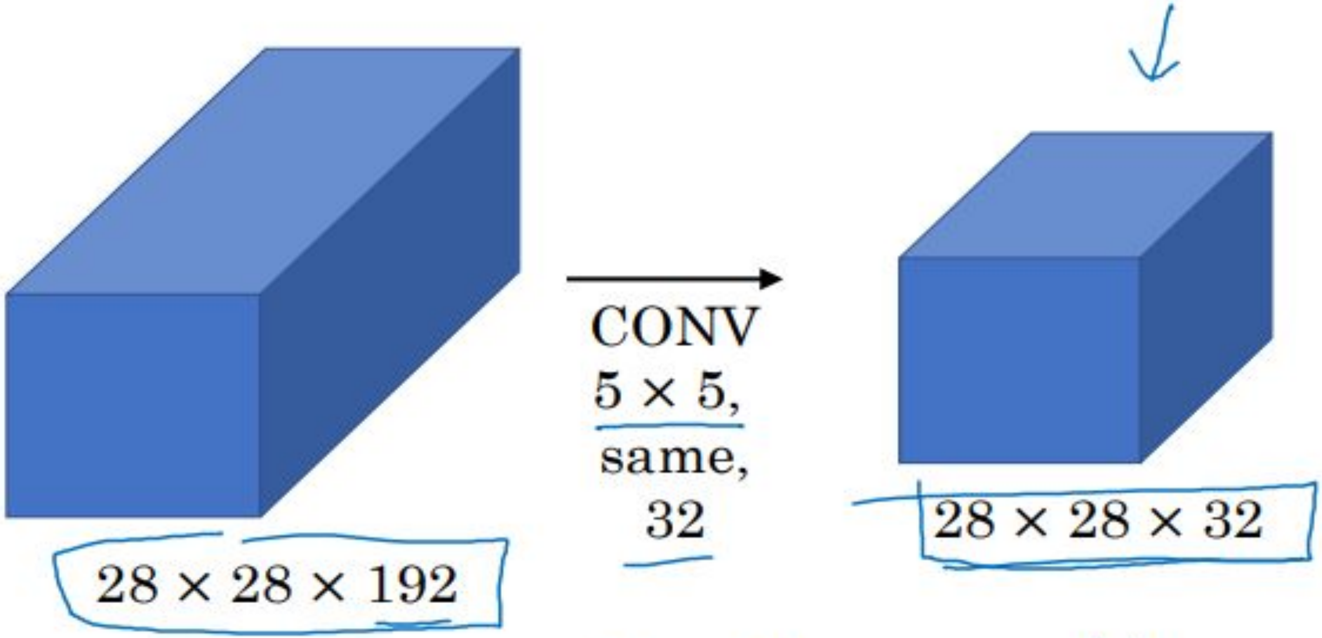
1. **Input:** 28*28*192
2. **Output:** 28*28*32
3. If we do with 3*3 same conv
    a. **Number of filters=**32
    b. **Filter size=**3*3*192
    c. **Padding=** same
    d. **Number of operations required to produce one bit** =192*9
    e. **Total number of bits=** 28*28*32
    f. **Total Number of Operations=** c*d=2.4M*9

1*1 conv

1. **Input:** 28*28*192
2. **Output:** 28*28*32
3. If we do with 1*1 conv
    a. **Number of filters=**32
    b. **Filter size=**1*1*192
    c. **Number of operations required to produce one bit =**192
    d. **Total number of bits=** 28*28*32
    e. **Total Number of Operations=** c*d=2.4M

Strategy2

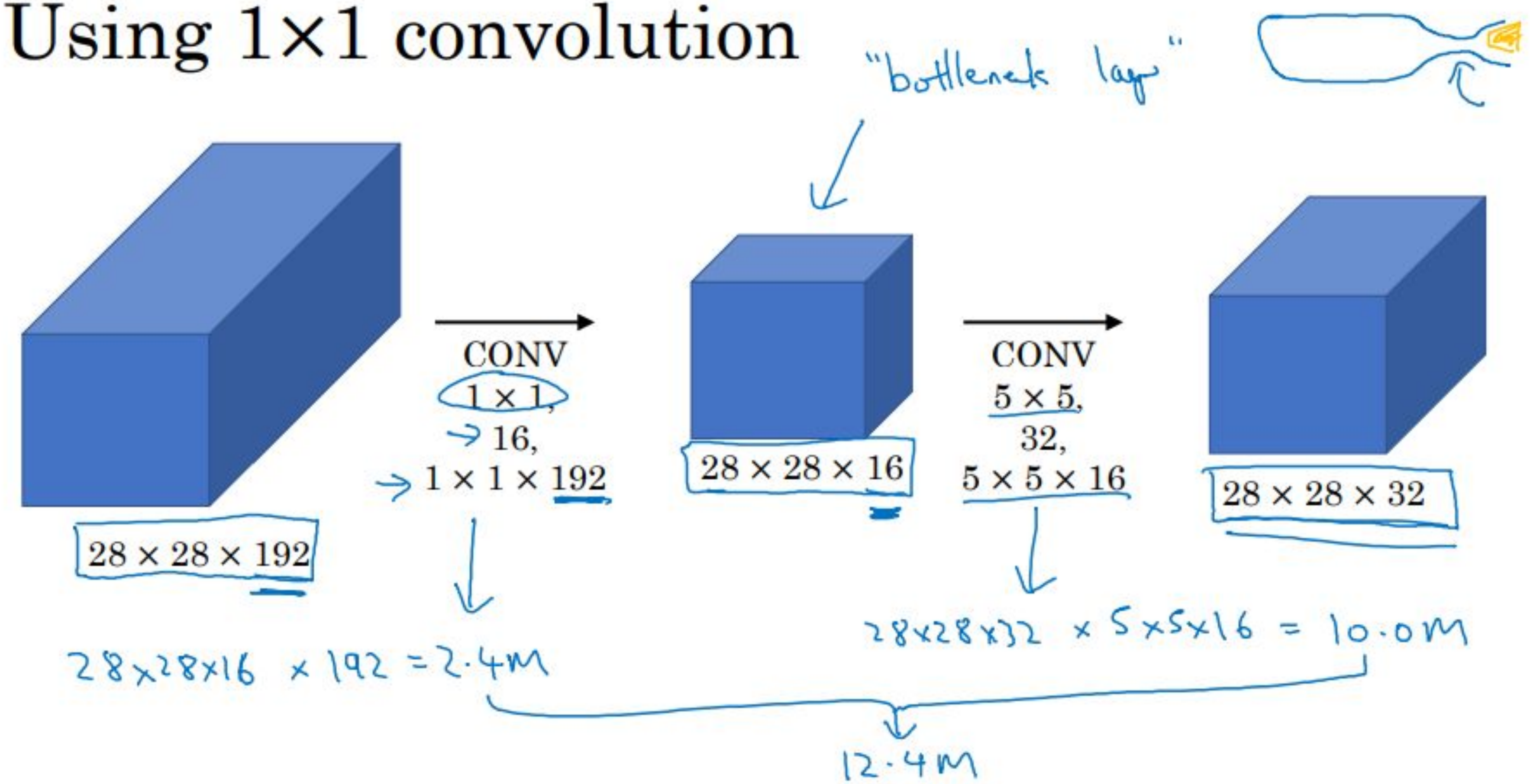
1. This strategy is used for decrease the number of computations that takes place.
2. This strategy is helpful because if this strategy doesn’t follow then
   a. **Filter size=3*3*32**
   b. **Number of operations required to produce one bit=** 3*3*32
   c. So if this 3*3 gets lesser input channels then total number operation required would become much lesser.



32 filters. filters are $\underline{5 \times 5 \times 192}$.

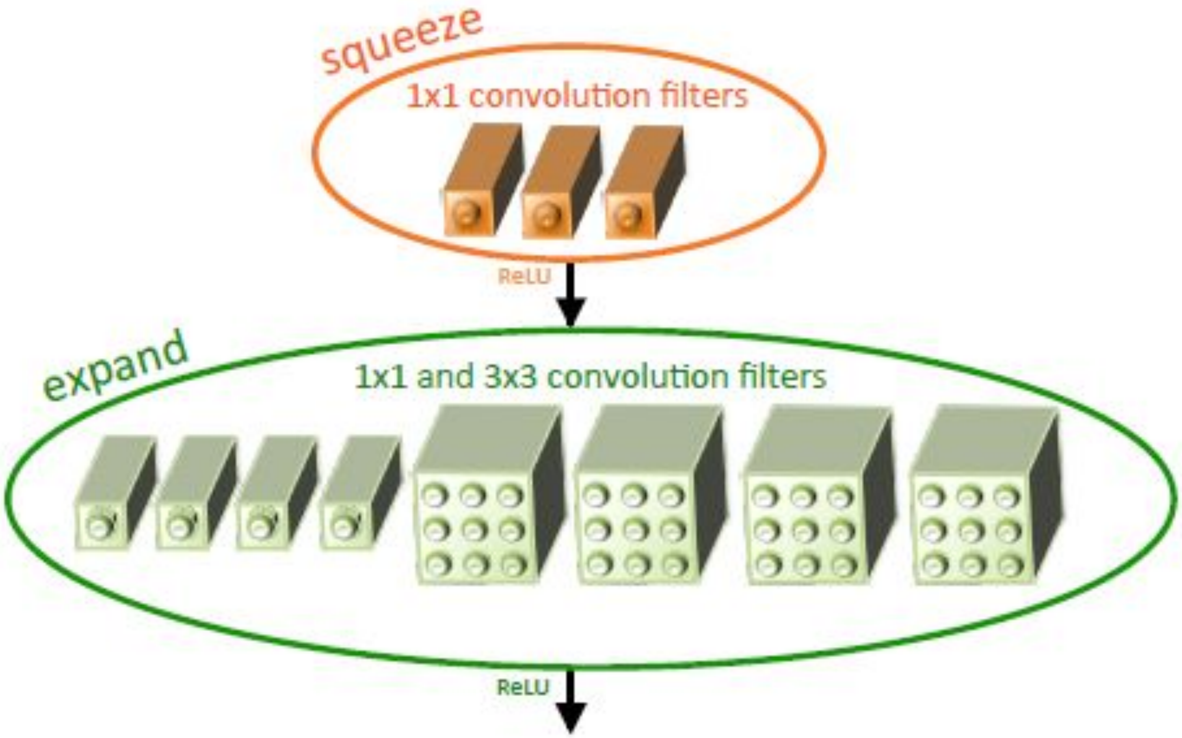
$\underline{28 \times 28 \times 32} \times \underline{5 \times 5 \times 192} = \underline{120M}$.

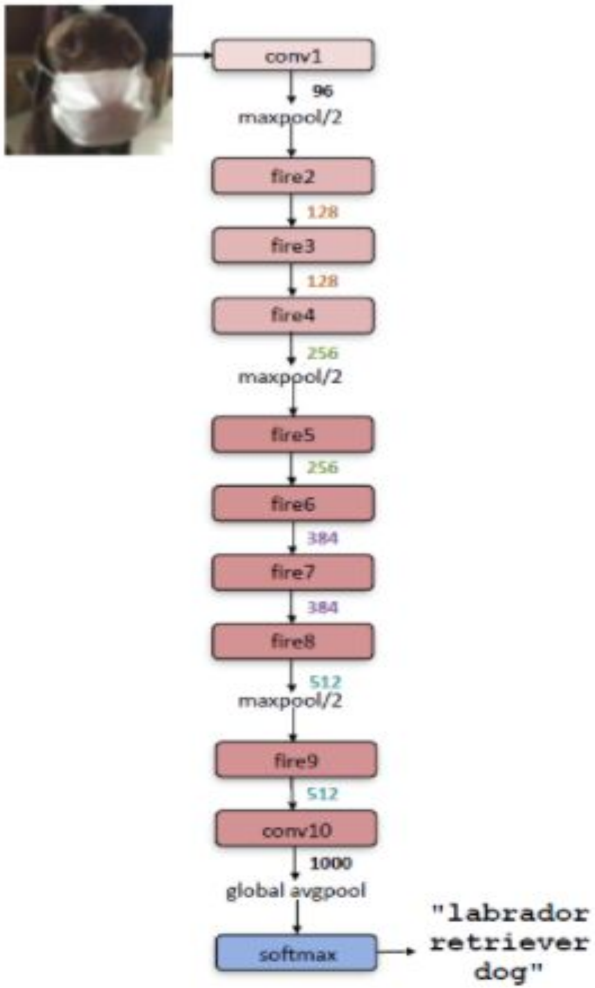
Using 1×1 convolution



Fire module

1. Inspired from inception which is developed by google.
2. Squeeze Layer with 1*1 conv layers(bottleneck).
3. Expand layer with 1*1 and 3*3 conv layers.
4. In This fire module Number channels decreases and then increases.
5. **Terminology:**
   a. s1*1: Number of 1*1 in squeeze layers
   b. e1*1: Number of 1*1 in expand layers
   c. e3*3: Number of 3*3 in expand layers


squeeze
1x1 convolution filters
ReLU
expand
1x1 and 3x3 convolution filters
ReLU


conv1
96
maxpool/2
fire2
128
fire3
128
fire4
256
maxpool/2
fire5
256
fire6
384
fire7
384
fire8
512
maxpool/2
fire9
512
conv10
1000
global avgpool
softmax
"labrador retriever dog"

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1					100% (7x7)		6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1					**20%** (3x3)		6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations	parameters					compression info				1,248,424 (total)	**421,098** (total)

Squeezenet:

Input image: 224 x 224 x 3

Outputs of layers:

Conv1: 111 x 111 x 96

- Filtersize: 7 x 7 x 3
- Stride: 2
- # of filters: 96
- depth: 1
- $\frac{224 - 7}{2} + 1 = 111$

Maxpool1: 55 x 55 x 96

- Filtersize: 3 x 3
- Stride: 2
- # of filter: 96
- depth: 0
- $\frac{111 - 3}{2} + 1 = 55$

Fire3: 128 channels (6

Fire4: 256 channels (1

Maxpool4: 27 x 27 x 256

Fire5: 256 channels

Fire6: 384 channels

Fire7: 384 channels

Fire8: 512 channels

Maxpool8: 13 x 13 x 5

Fire9: ~~13 x 13~~ 512

Conv10: 13 . 100

Fire2: (Fire module)

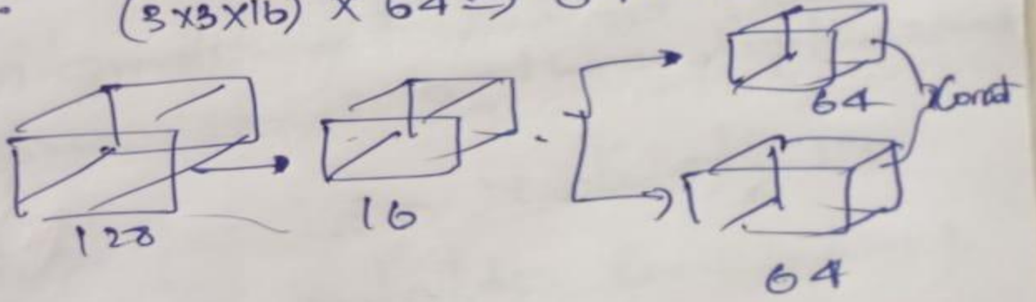
(55×55×128)

→ s(1×1) = 16 : → (1×1×128)×16 ⇒ 16 featuremaps.

→ e(1×1) : 64 : (1×1×16) × 64 ⇒ 64 featuremaps

→ e(3×3) : 64 : (3×3×16) × 64 ⇒ 64 featuremaps

→ Depth = 2



55 + ● − 3 + 1 + 2

Fire3: 128 channels (64+64)

Fire4: 256 channels (128+128)

55 + ~~0~~ - 3 + 1 + 2

Fire3: 128 channels (64+64)

Fire4: 256 channels (128+128)

Maxpool4: 27×27×256

Fire5: 256 channels (128+128)

Fire6: 384 channels (192+192)

Fire7: 384 channels (192+192)

Fire8: 512 channels (256+256)

downsampled after 4 Firemodule

Maxpool8: 13 ×13 ×512

Fire9: 1~~3×13×~~ 512 channel (256+256)

Conv10: 1~~3~~ · 1000 channels (1×1 Convo 1000 filters)

99
17
2

Evaluation

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%