Residual Networks and Analyses

He, Kaiming, et al. "Deep Residual Learning for Image Recognition." *arXiv preprint arXiv:1512.03385* (2015).

Andreas el. al. "Residual Networks are Exponential Ensembles of Relatively Shallow Networks." *arXiv* (2016).

Zagoruyko, Sergey, and Nikos Komodakis. "Wide Residual Networks." *arXiv* (2016).

Deep residual networks

152 layers network

1st place on ILSVRM 2015 classification task

1st place on ImageNet detection

1st place on ImageNet localization

1st place on COCO detection

1st place on COCO segmentation

Deeper Network?

Is deeper network always better?

What about vanishing/exploding gradients?

- Better initialization methods / batch normalization / ReLU

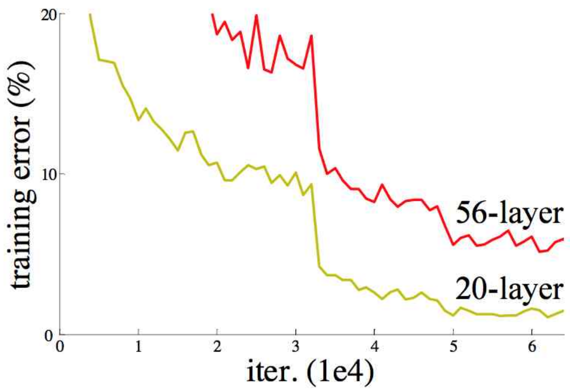
Any other problems?

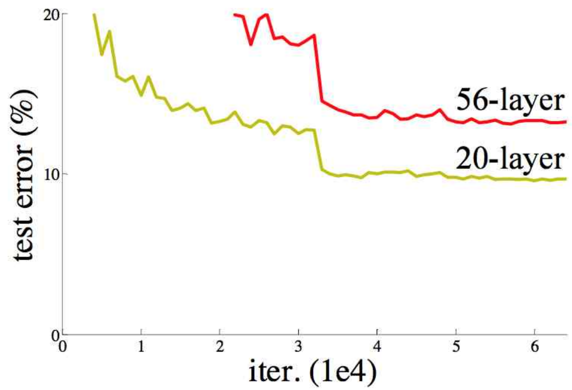
- Overfitting?

- Degradation problem: more depth but lower performance

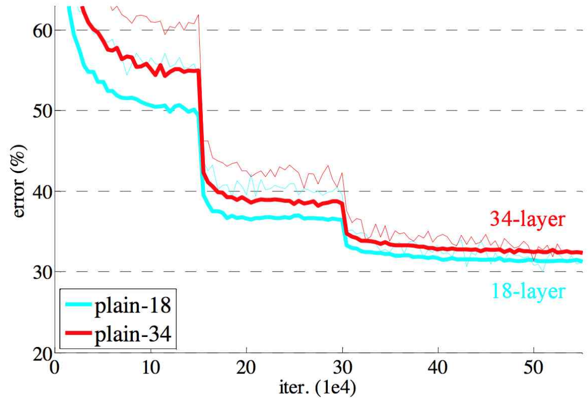
Degeneration problem

CiFAR 100 Dataset

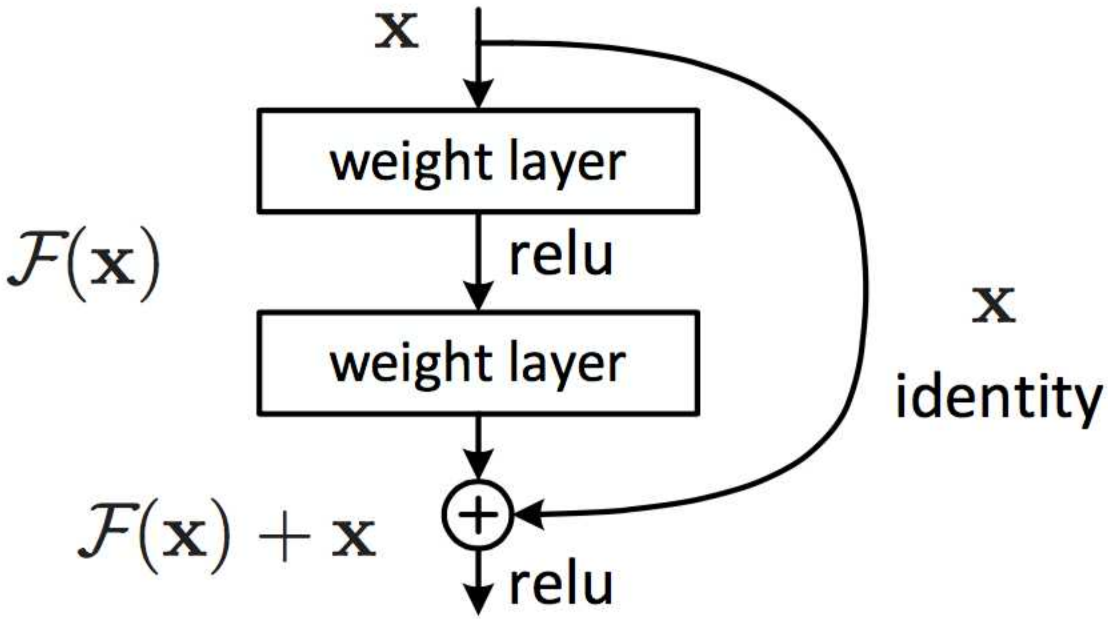




ImageNet

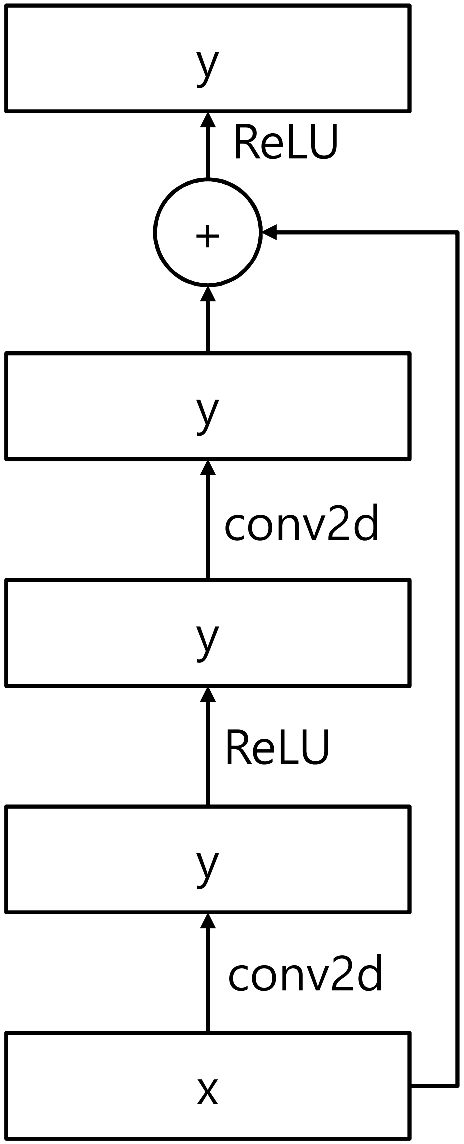


Residual learning building block



Residual learning building block

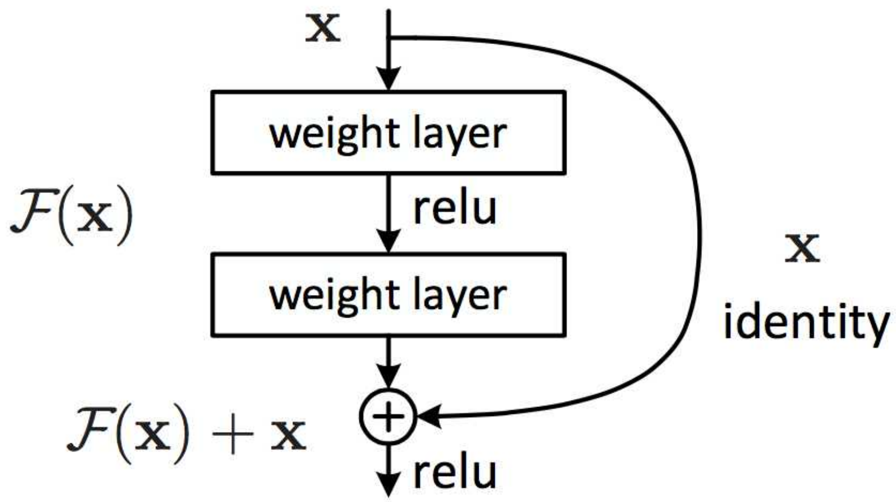
```
def residual_block(x, n_in, n_out, subsample, phase_train, scope='res_block'):
  with tf.variable_scope(scope):
    if subsample:
      y = conv2d(x, n_in, n_out, 3, 2, 'SAME', False, scope='conv_1')
      shortcut = conv2d(x, n_in, n_out, 3, 2, 'SAME',
                 False, scope='shortcut')
    else:
      y = conv2d(x, n_in, n_out, 3, 1, 'SAME', False, scope='conv_1')
      shortcut = tf.identity(x, name='shortcut')
    y = batch_norm(y, n_out, phase_train, scope='bn_1')
    y = tf.nn.relu(y, name='relu_1')
    y = conv2d(y, n_out, n_out, 3, 1, 'SAME', True, scope='conv_2')
    y = batch_norm(y, n_out, phase_train, scope='bn_2')
    y = y + shortcut
    y = tf.nn.relu(y, name='relu_2')
  return y


def conv2d(x, n_in, n_out, k, s, p='SAME', bias=False, scope='conv'):
  with tf.variable_scope(scope):
    kernel = tf.Variable(
      tf.truncated_normal([k, k, n_in, n_out],
        stddev=math.sqrt(2/(k*k*n_in))),
      name='weight')
    tf.add_to_collection('weights', kernel)
    conv = tf.nn.conv2d(x, kernel, [1,s,s,1], padding=p)
    if bias:
      bias = tf.get_variable('bias', [n_out], initializer=tf.constant_initializer(0.0))
      tf.add_to_collection('biases', bias)
      conv = tf.nn.bias_add(conv, bias)
  return conv
```



Why residual?

We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.

Shortcut connections are used.



Why residual?

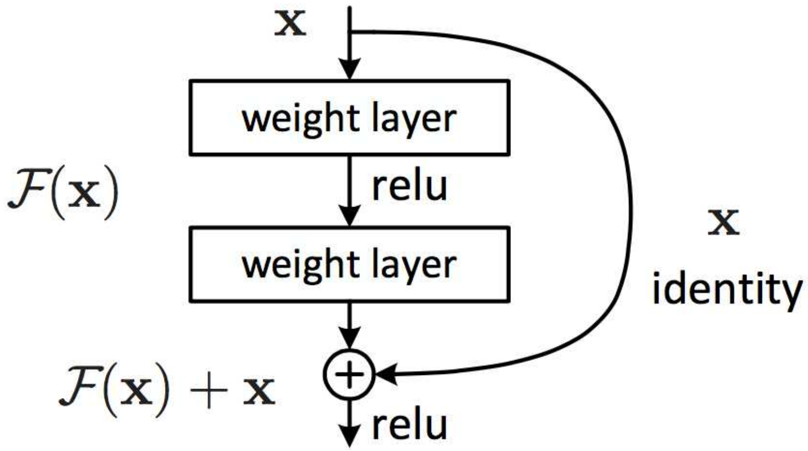
“The extremely deep residual nets are easy to optimize. “

“The deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks. “

Residual mapping

Basic residual mapping (same dim.)

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



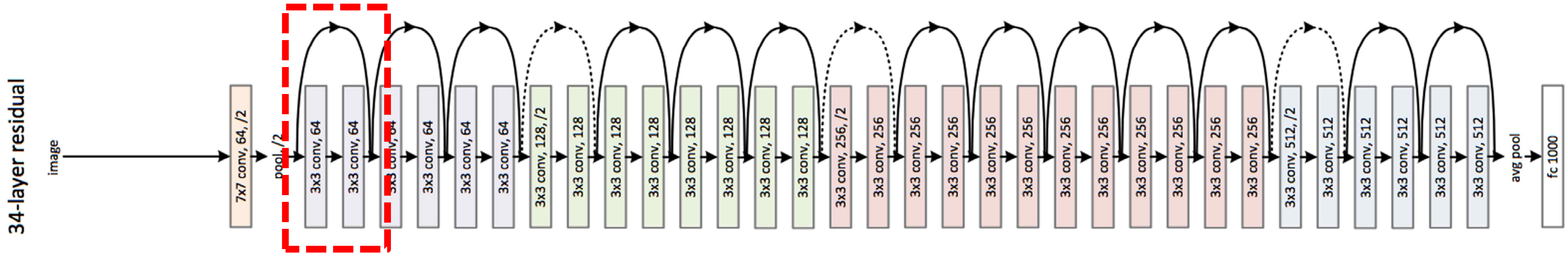
$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$

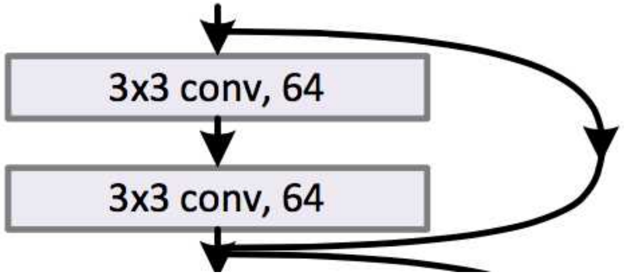
Basic residual mapping (different dim.)

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

"But we will show by experiments that the identity mapping is sufficient for addressing the degradation problem and is economical, and thus W is only used when matching dimensions."

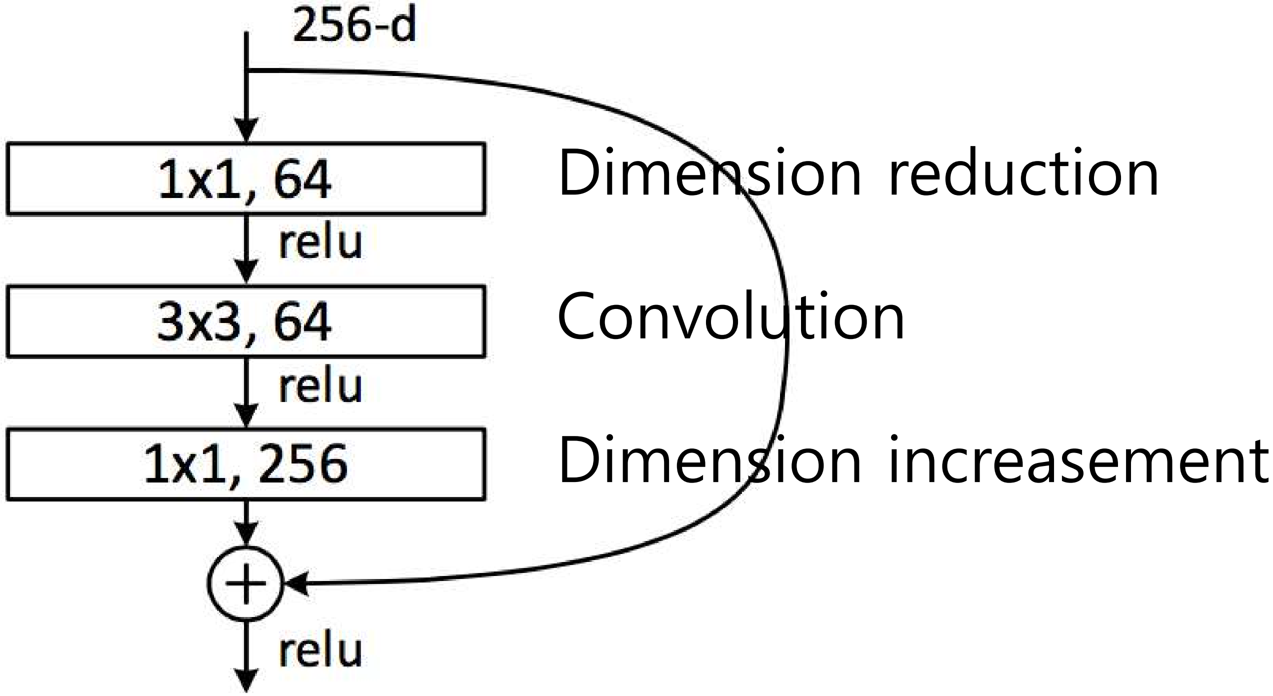
Deep residual network



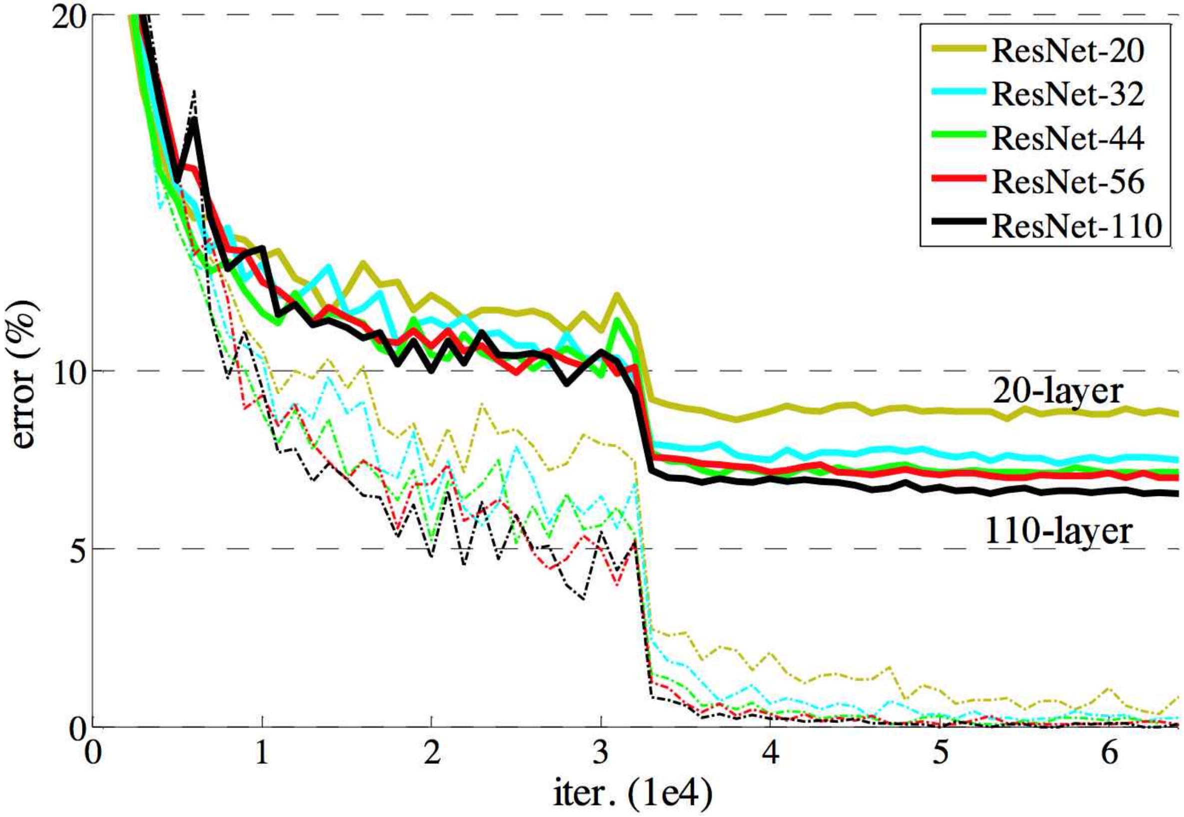



```python
def residual_block(x, n_in, n_out, subsample, phase_train, scope='res_block'):
  with tf.variable_scope(scope):
    if subsample:
      y = conv2d(x, n_in, n_out, 3, 2, 'SAME', False, scope='conv_1')
      shortcut = conv2d(x, n_in, n_out, 3, 2, 'SAME',
                        False, scope='shortcut')
    else:
      y = conv2d(x, n_in, n_out, 3, 1, 'SAME', False, scope='conv_1')
      shortcut = tf.identity(x, name='shortcut')
    y = batch_norm(y, n_out, phase_train, scope='bn_1')
    y = tf.nn.relu(y, name='relu_1')
    y = conv2d(y, n_out, n_out, 3, 1, 'SAME', True, scope='conv_2')
    y = batch_norm(y, n_out, phase_train, scope='bn_2')
    y = y + shortcut
    y = tf.nn.relu(y, name='relu_2')
  return y
```

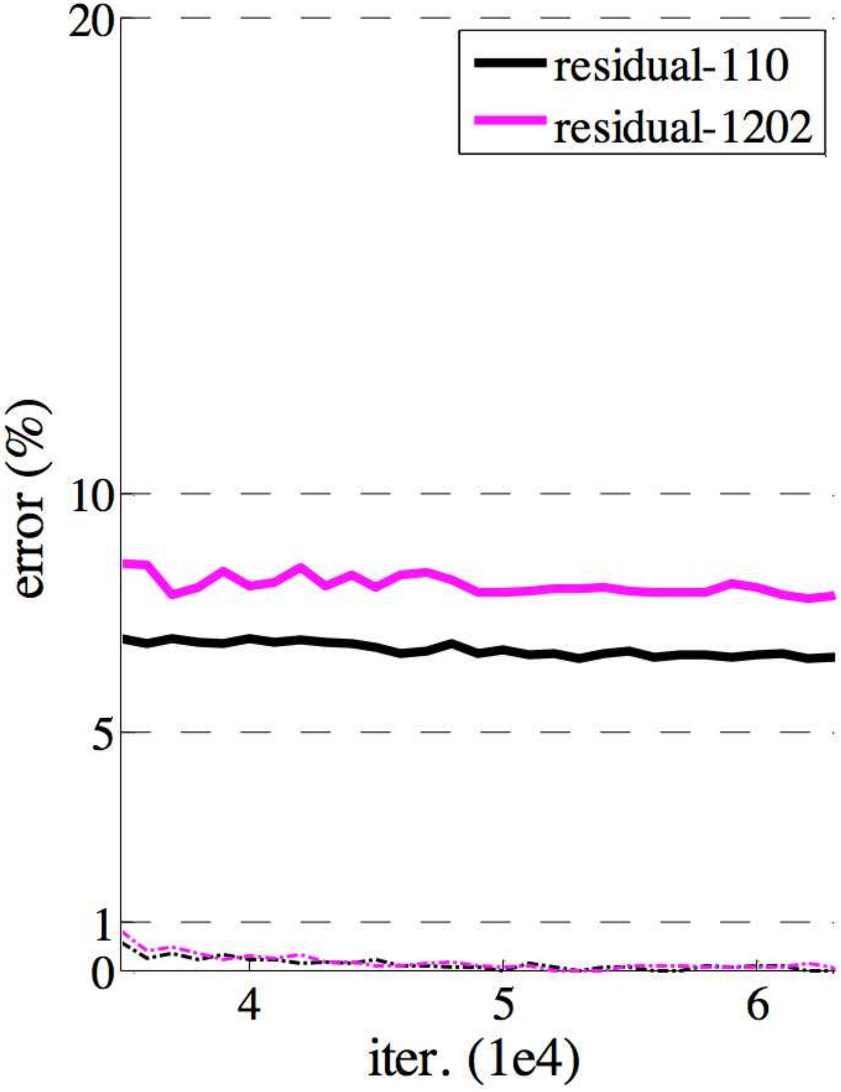
Deeper bottle architecture



Experimental results



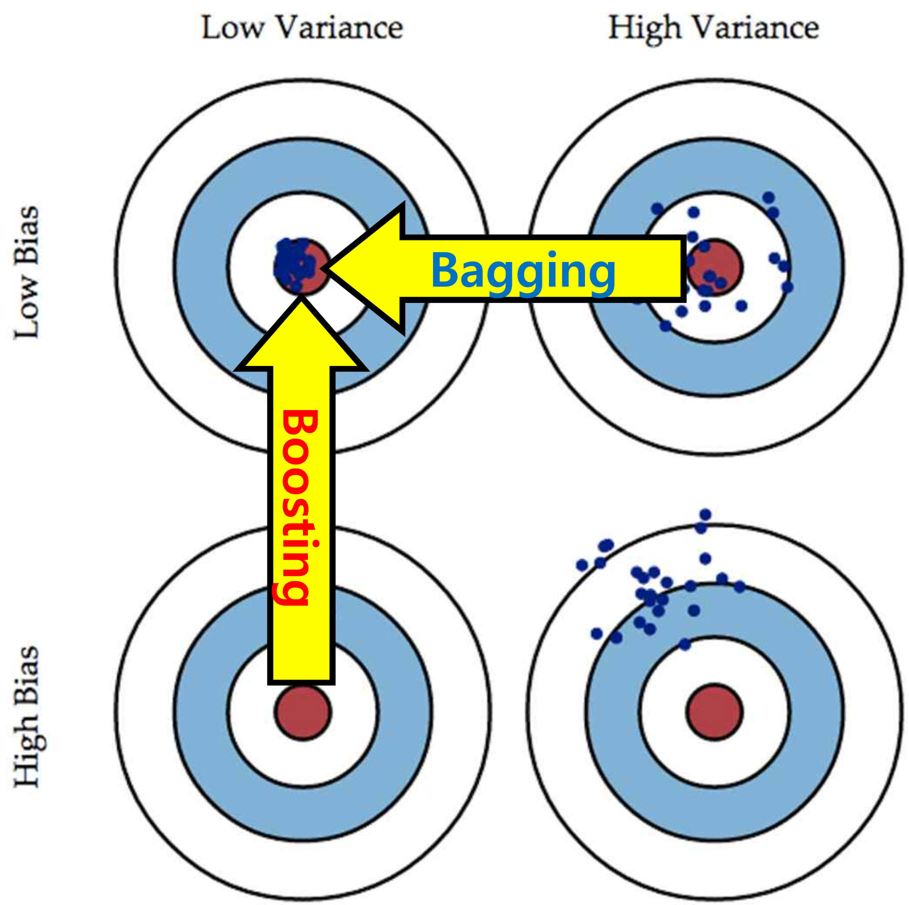
Experimental results



ResNet is an ensemble model

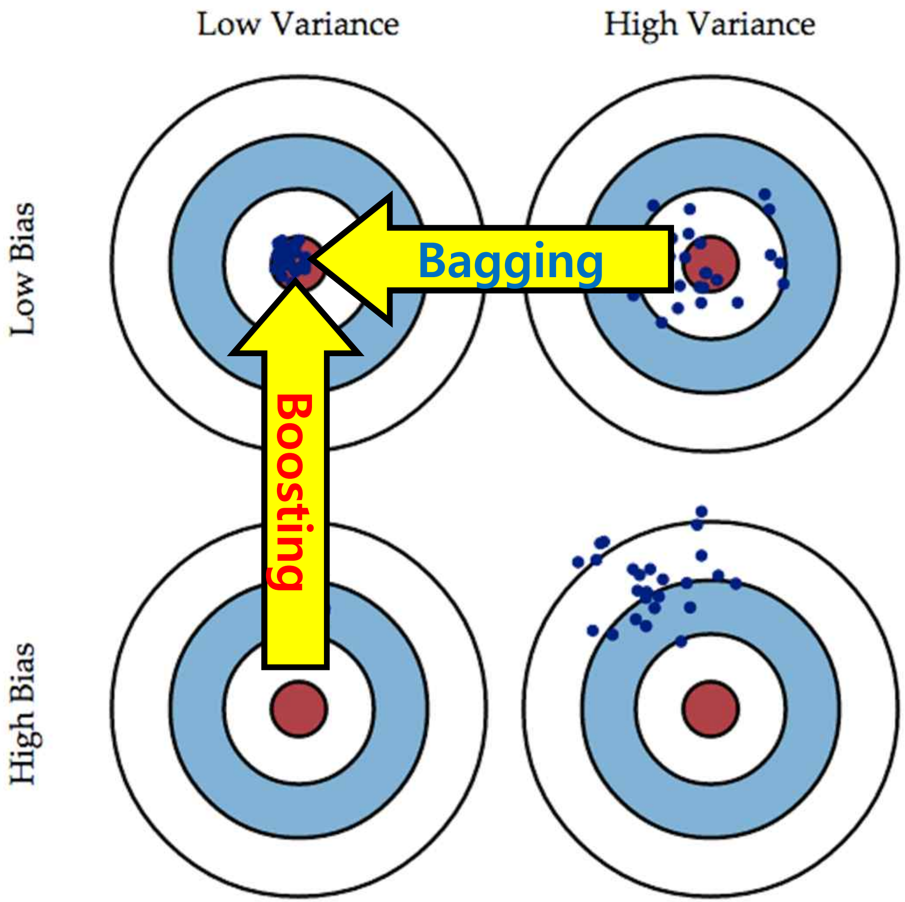
Andreas el. al. "Residual Networks are Exponential Ensembles of Relatively Shallow Networks." *arXiv* (2016).

Ensemble model?



The Famous Bias/Variance Tradeoff

Bagging vs. Boosting



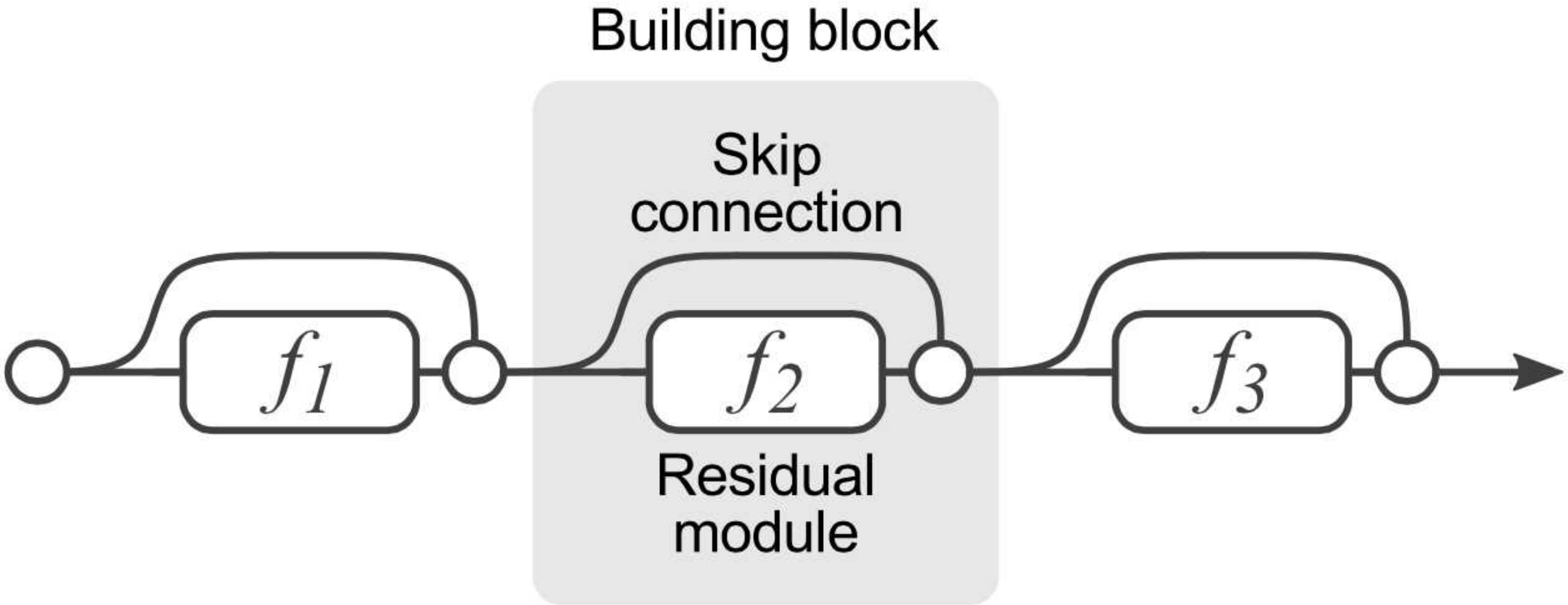
Bagging

- Generate multiple sets of training data (with bootstrapping), multiple predictions and combine them with averaging prediction
- Reduce overall variance using multiple low-bias models
- Example: Random Forest

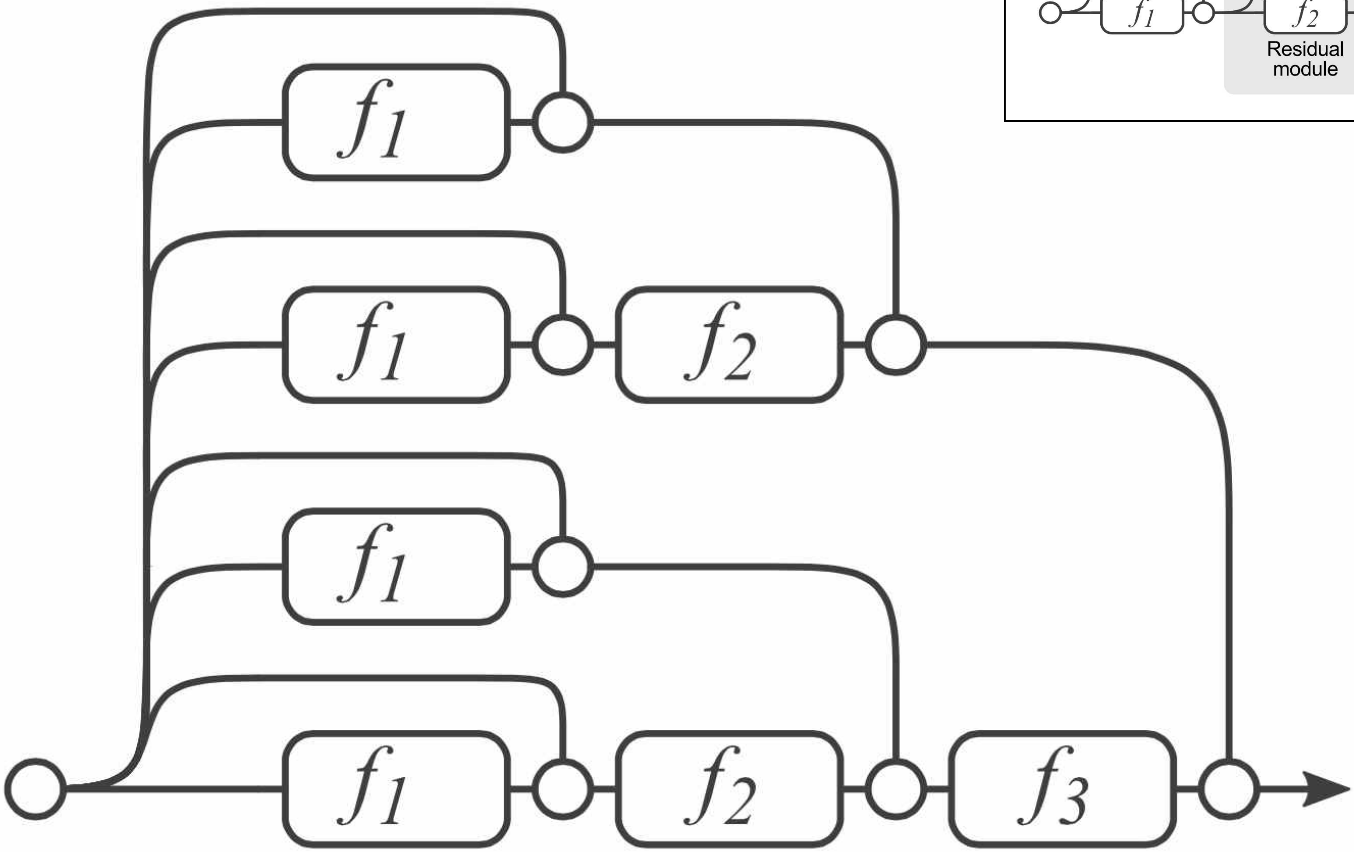
Boosting

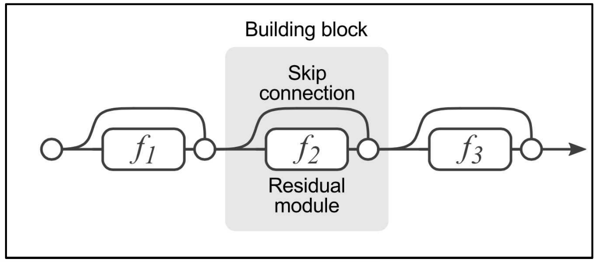
- Generate multiple weak learners and combine them to make a strong learner
- Reduce overall bias using multiple weak learners
- Example: Boosting, AdaBoost

ResNet is an ensemble model?



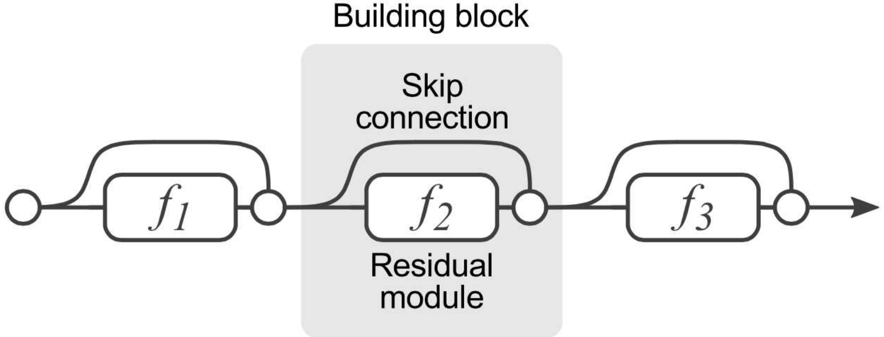
ResNet is an ensemble model?

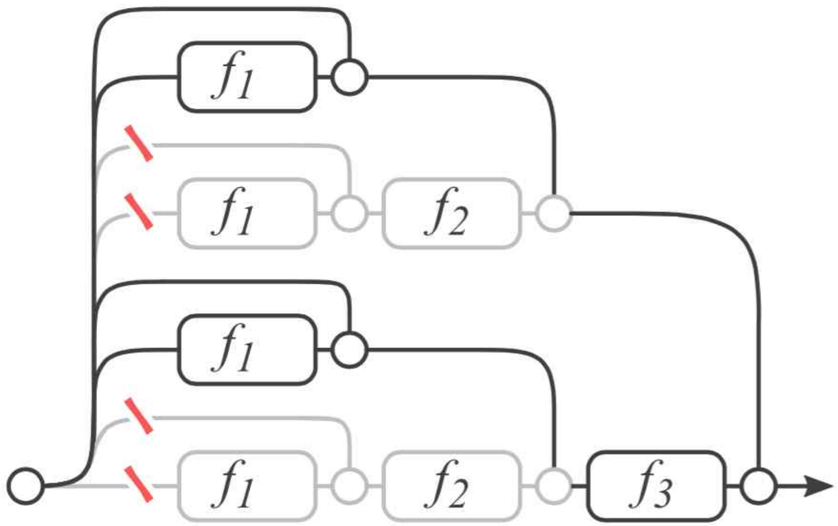




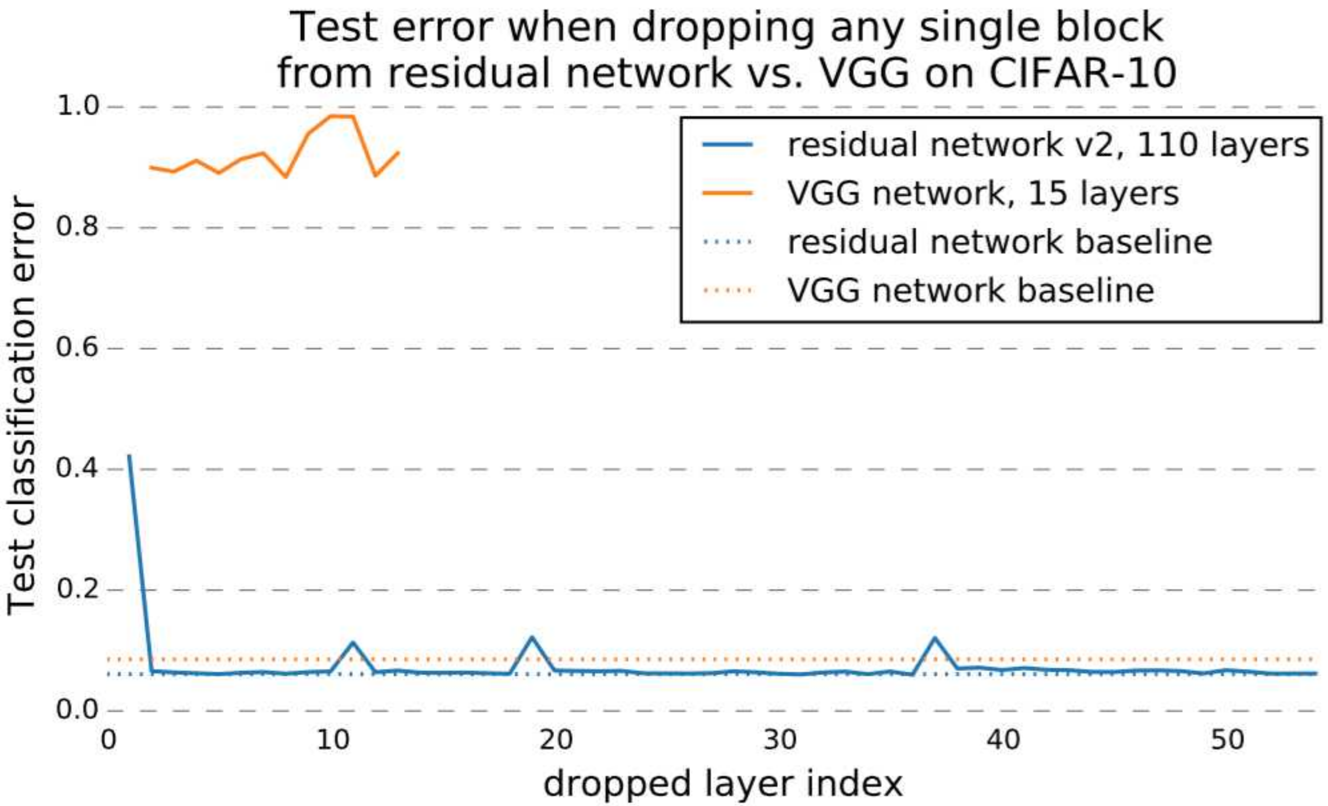
Remove a layer?

What happens if we remove the second layer?

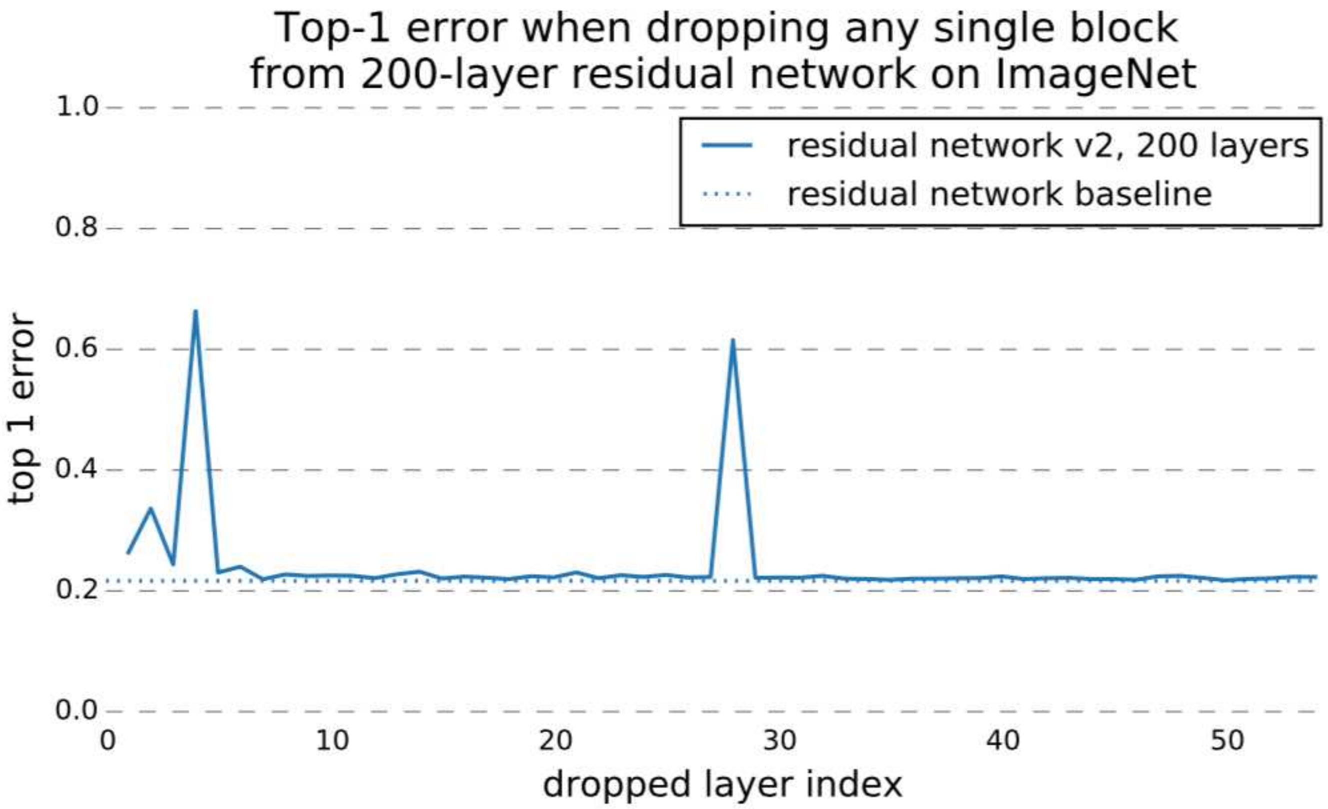




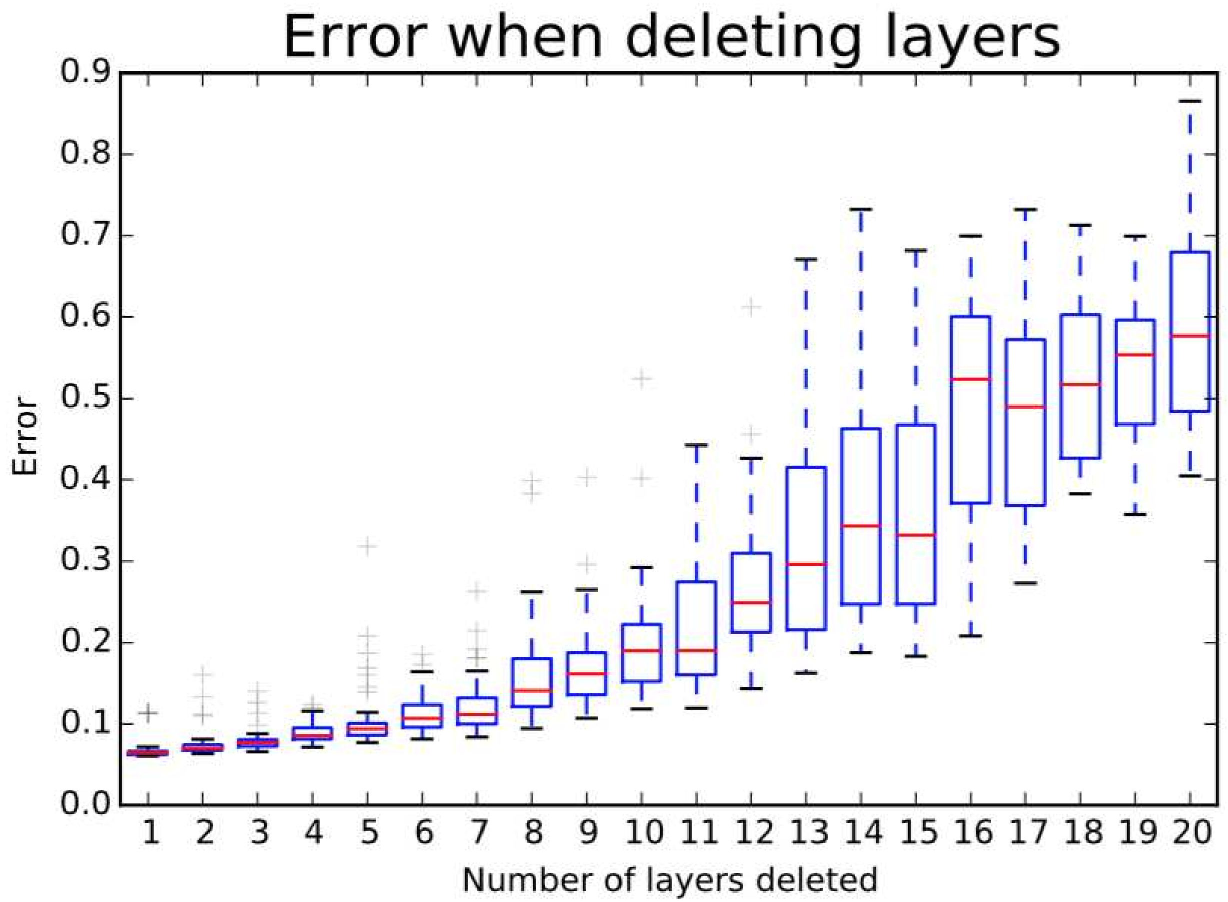
Performance?



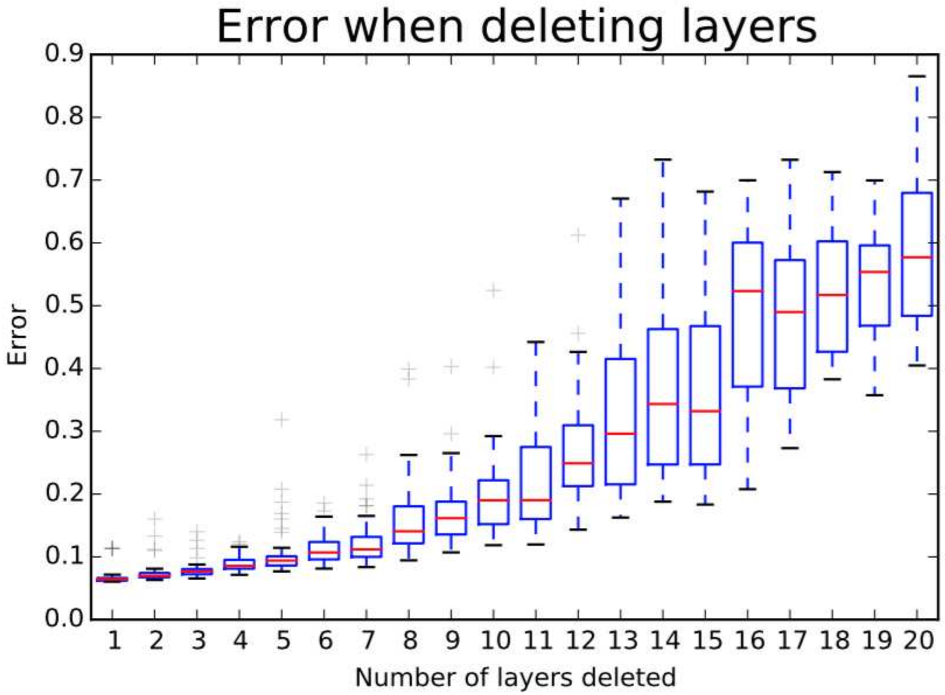
Performance?



Removing multiple layers?

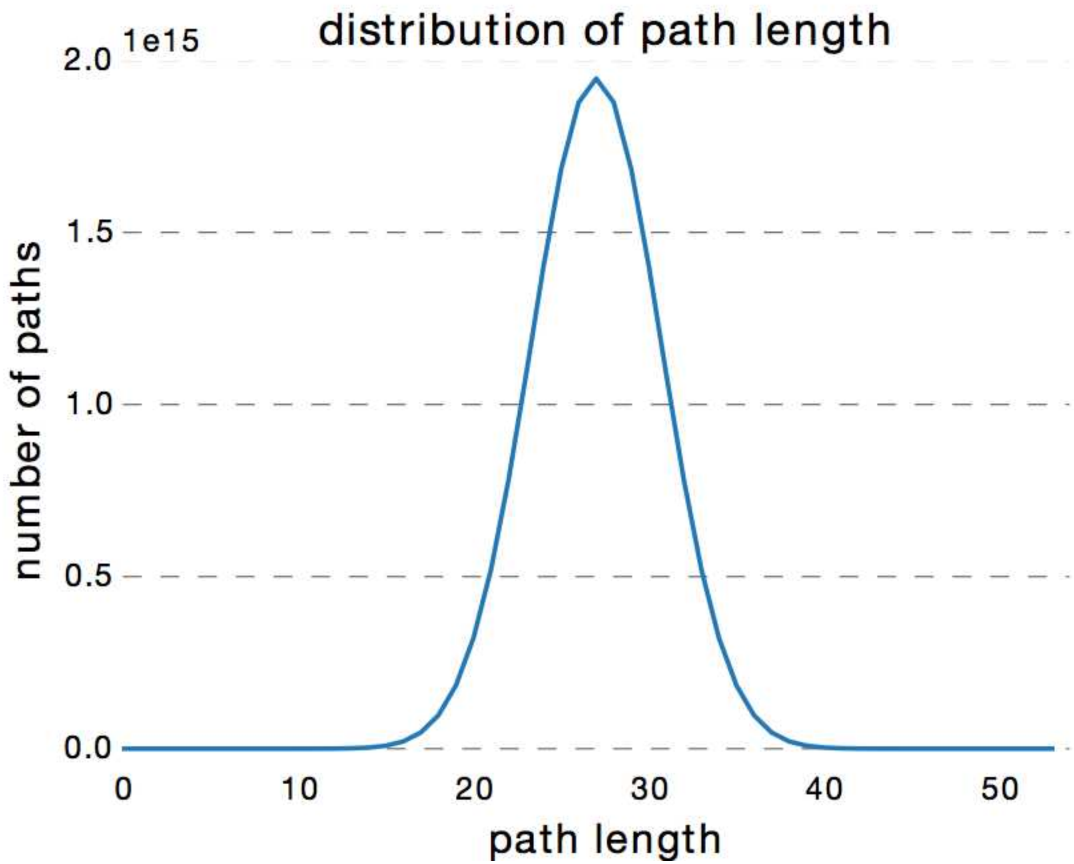


What does this mean?



"One of the characteristics of ensembles is that their performance depends smoothly on the number of members. …. This means **error should decrease smoothly as we delete more residual modules**. "

Ensemble of shallow nets?



Path distribution follows a Binomial distribution.

More than 95% of paths go through 19 to 35 modules.

Depth is NOT ~~that~~ important

Zagoruyko, Sergey, and Nikos Komodakis. "Wide Residual Networks." *arXiv* (2016).

Depth vs. width

“The authors of residual networks tried to make them as **thin** as possible in favor of increasing their depth and having less parameters, and even introduced a «bottleneck» block which makes ResNet blocks even thinner.”

“We note, however, that the residual block with identity mapping that allows to train very deep networks is at the same time a **weakness** of residual networks. As gradient flows through the network there is nothing to force it to go through residual block weights and it can avoid learning anything during training, so it is possible that there is either only a few blocks that learn useful representations, or many blocks share very little information with small contribution to the final goal.”

Experimental results

depth	k	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

Table 4: Test error (%) of various wide networks on CIFAR-10 and CIFAR-100.

Experimental results

depth	k	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

Table 4: Test error (%) of various wide networks on CIFAR-10 and CIFAR-100.

Experimental results

	depth-*k*	# params	CIFAR-10	CIFAR-100
NIN [18]			8.81	35.67
DSN [17]			8.22	34.57
FitNet [22]			8.39	35.04
Highway [26]			7.72	32.39
ELU [5]			6.55	24.28
original-ResNet[9]	110	1.7M	6.43	25.16
	1202	10.2M	7.93	27.82
stoc-depth[12]	110	1.7M	[illegible]	[illegible]
	1202	10.2M	[illegible]	[illegible]
pre-act-ResNet[11]	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.7M	4.97	22.89
	16-8	11.0M	4.81	22.07
	28-10	36.5M	**4.17**	**20.50**

8 times faster to train

Table 5: Test error of different methods on CIFAR-10 and CIFAR-100 with moderate data augmentation (flip/translation). We don't use dropout for these results. In the second column *k* is a widening factor. Results for [11] are shown with minibatch size 128 (as ours), and 64 in parenthesis. Our results are based on 1-time runs. We will update the paper with 5-time run statistics.

Experimental results

	depth-*k*	# params	CIFAR-10	CIFAR-100
NIN [18]			8.81	35.67
DSN [17]			8.22	34.57
FitNet [22]			8.39	35.04
Highway [26]			7.72	32.39
ELU [5]			6.55	24.28
original-ResNet[9]	110	1.7M	6.43	25.16
	1202	10.2M	7.93	27.82
stoc-depth[12]	110	1.7M	5.23	24.58
	1202	10.2M	4.91	-
pre-act-ResNet[11]	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.7M	4.97	22.89
	16-8	11.0M	4.81	22.07
	28-10	36.5M	**4.17**	**20.50**

Table 5: Test error of different methods on CIFAR-10 and CIFAR-100 with moderate data augmentation (flip/translation). We don't use dropout for these results. In the second column *k* is a widening factor. Results for [11] are shown with minibatch size 128 (as ours), and 64 in parenthesis. Our results are based on 1-time runs. We will update the paper with 5-time run statistics.

Summary

Widening consistently improves performance across residual networks of different depth

Increasing both depth and width helps until the number of parameters becomes too high and stronger regularization is needed

Wide networks can successfully learn with a 2 or more times larger number of parameters than thin ones, which would re-quire doubling the depth of thin networks, making them infeasibly expensive to train.