Google Inception

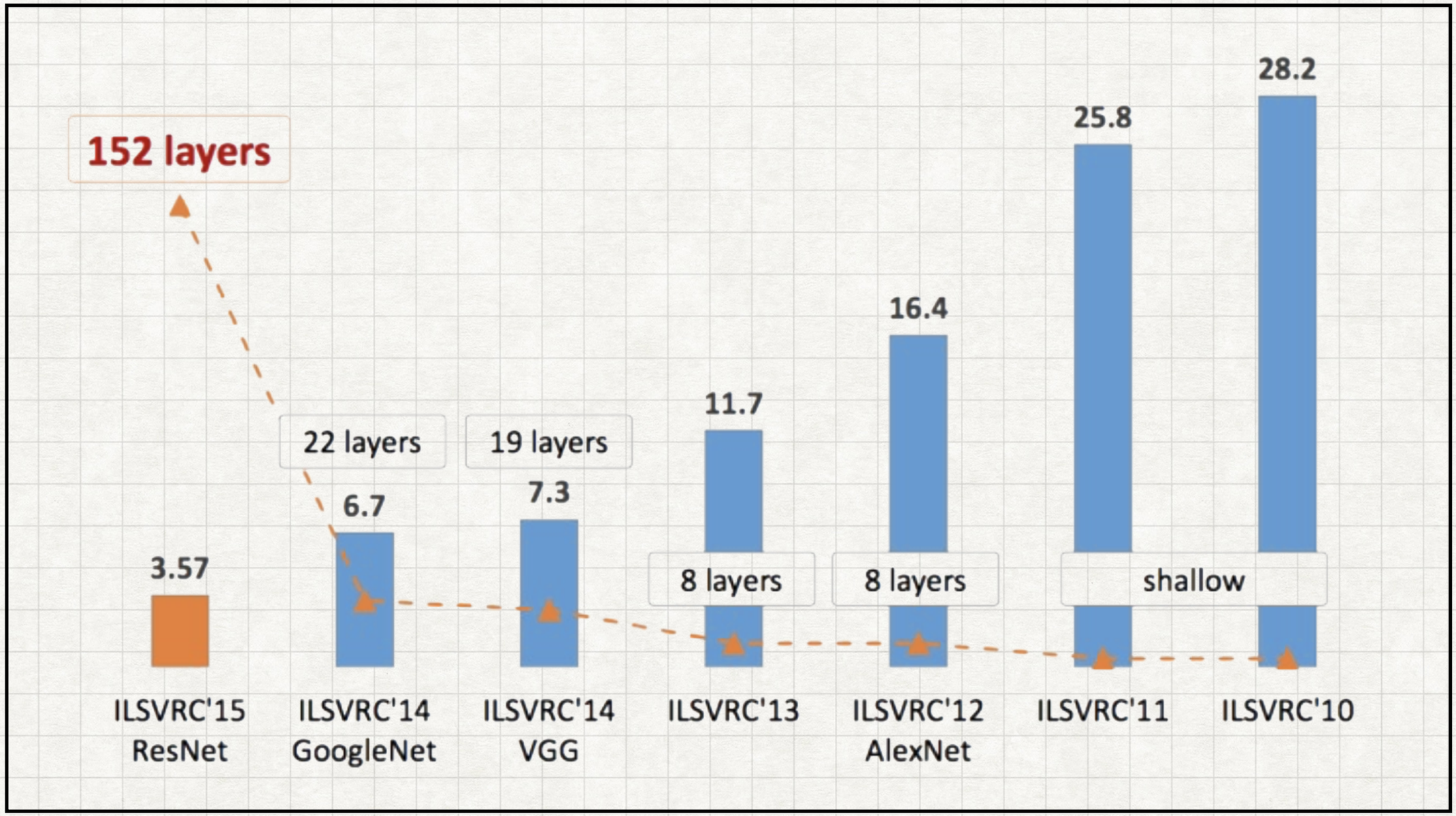
v3 v4

報告人：楊哲寧

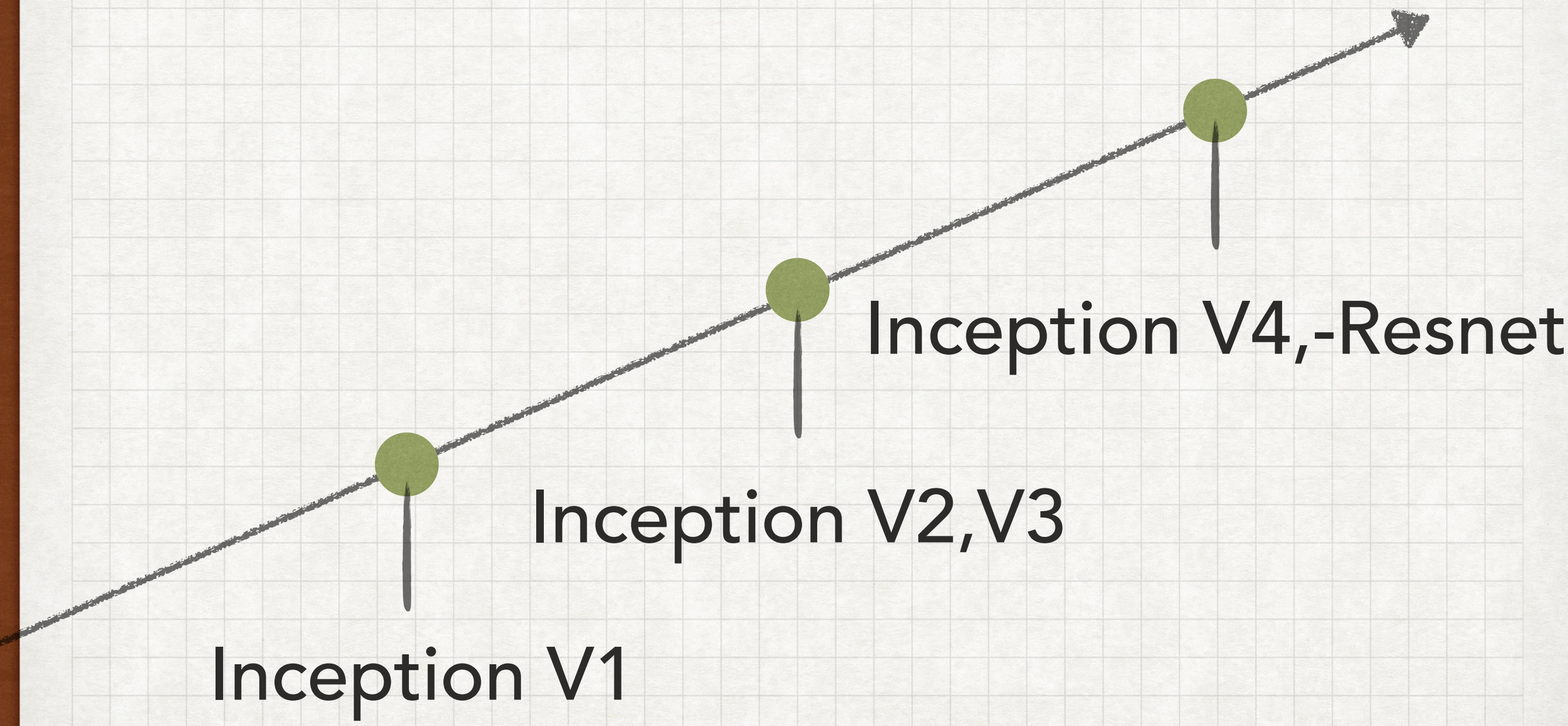
GoogLeNet(V1)是一個22層的深度網絡

Model	Alexnet	Vgg	Googlenet	Resnet
推出年	2012	2014	2014	2015
層數	8	19	22	152
卷積層數	5	16	21	151
卷積核大小	11,5,3	3	7,1,3,5	7,1,3,5
Inception(NIN)	-	-	+	-
全連接層	3	3	1	1
全連接層大小	4096*2,1000	4096*2,1000	1000	1000
Top-5 error	16.4%	7.3%	6.7%	3.57%

Top-5 error


152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10

INCEPTION



Inception

感受野（receptive field）

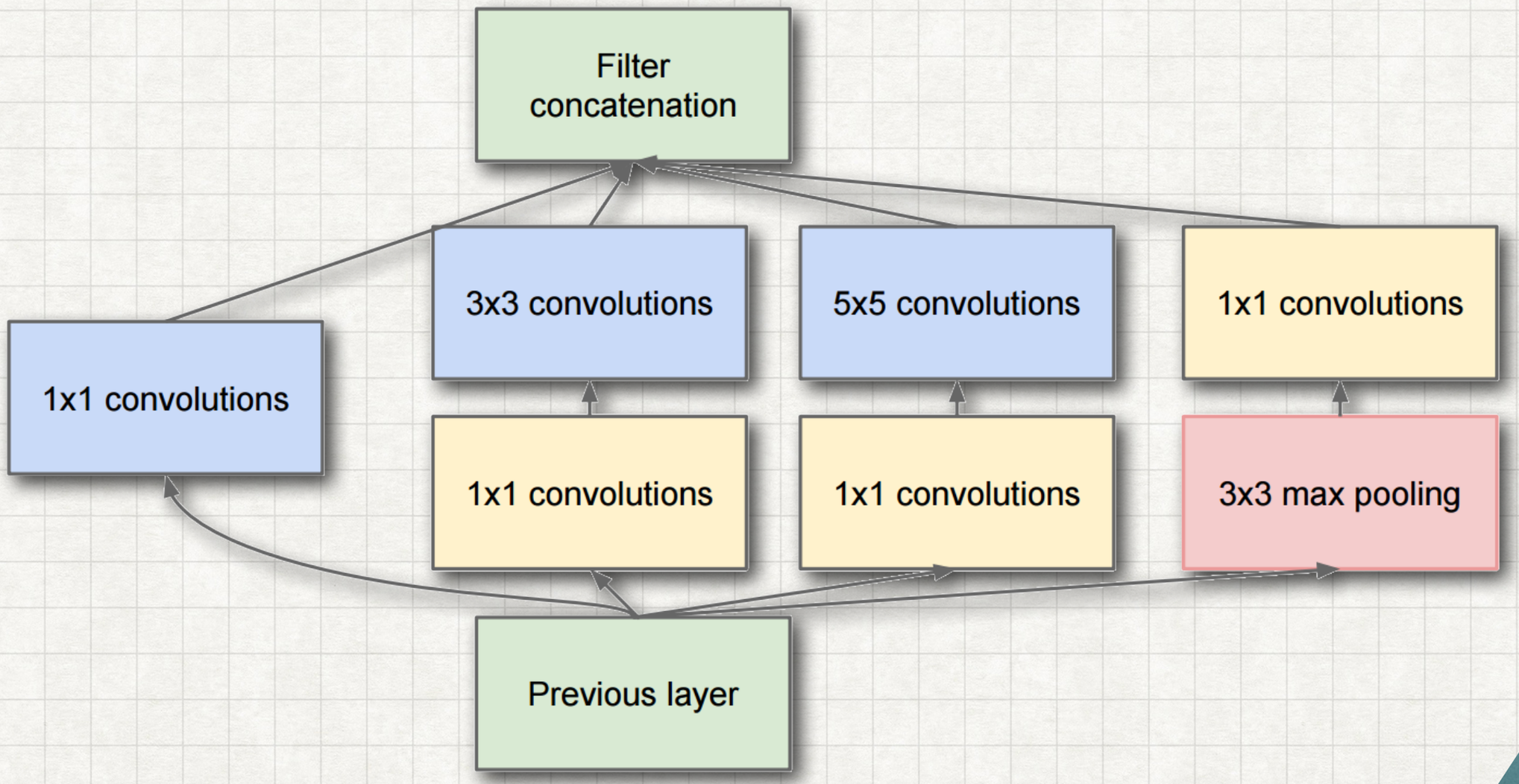
1 * 1, 3 * 3, 5 * 5, 7*7等不同的捲積運算與池化操作可以獲得輸入圖像的不同信息

由於信息位置的巨大差異，為卷積操作選擇**合適的卷積**核大小就比較困難

信息分佈更**全局性的圖**像偏好較**大**的卷積核
信息分佈比較**局部的圖**像偏好較**小**的卷積核

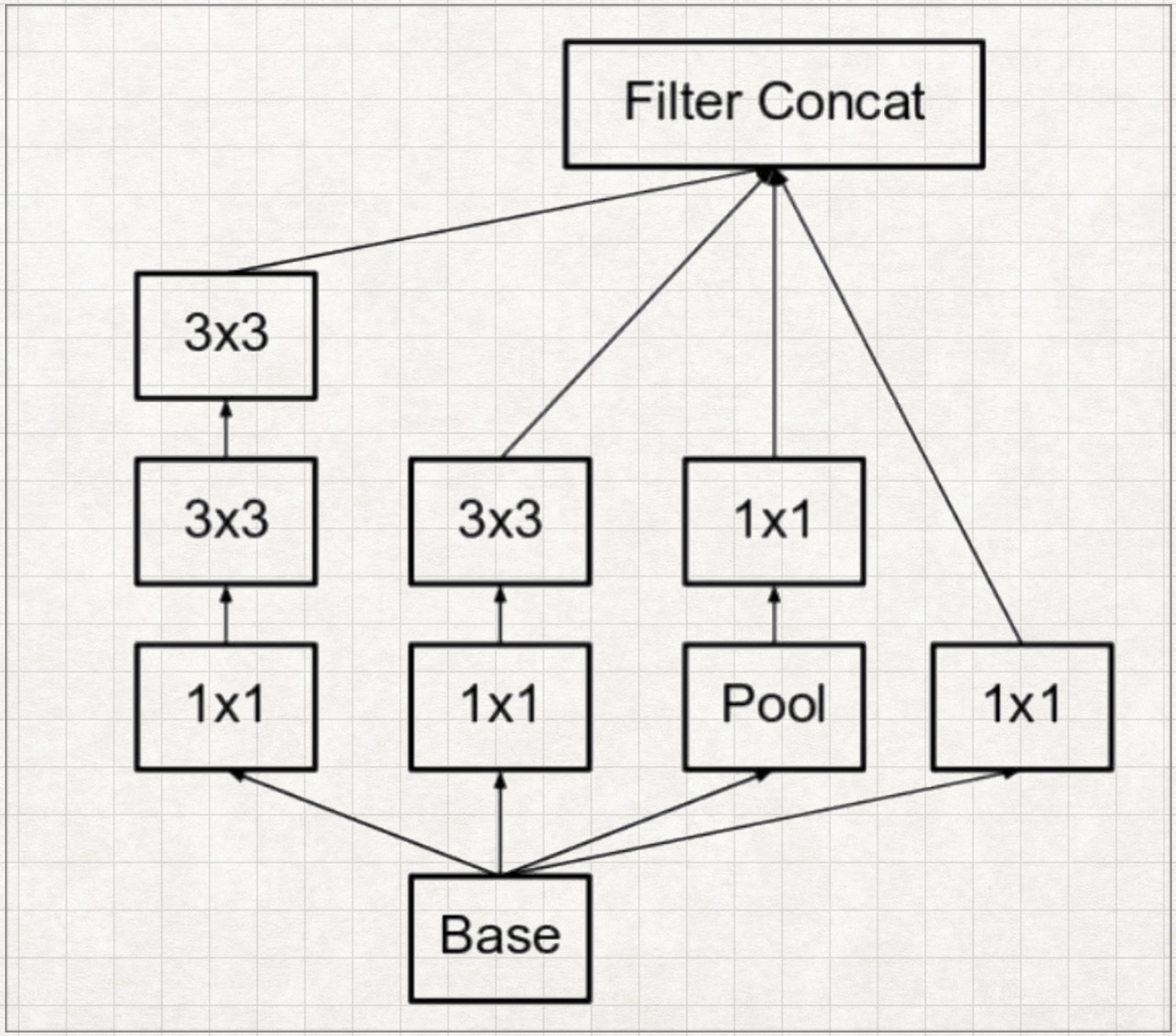
簡單地堆疊較大的卷積層非常**消耗計算資源**

First Inception Structure



當卷積不會大幅度改變輸入維度時，神經網絡可能會執行地更好(InceptionV2)

將5×5的捲積分解為兩個3×3的捲積運算以提升計算速度。

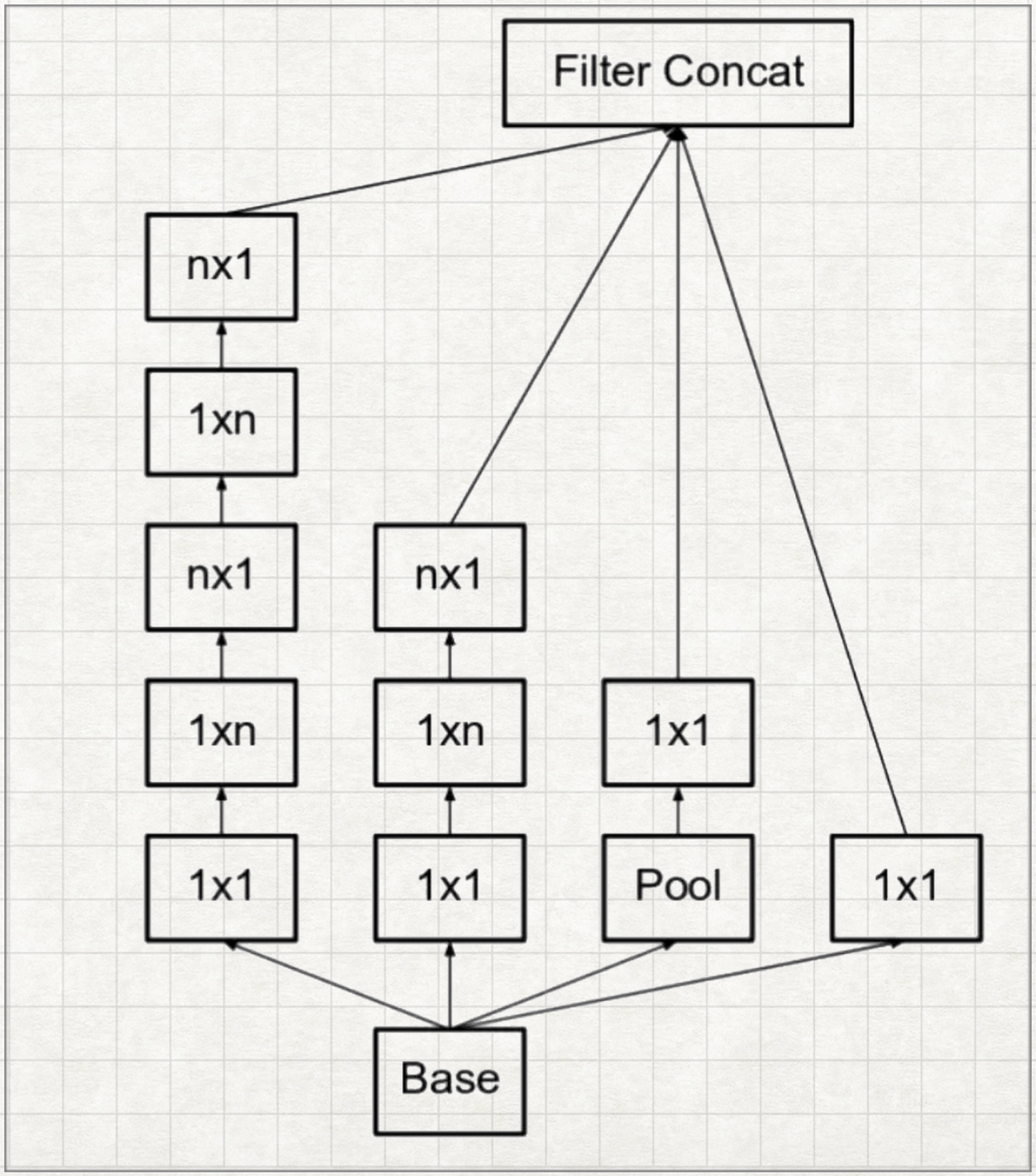


GoogLeNet團隊發現在網絡的前期使用這種分解效果並不好，
在中度大小的特徵圖（特徵圖）上使用效果才會更好（特徵圖大小建議在12到20之間）。

n * n的捲積核尺寸分解為1×n和n×1兩個卷積

加速計算

可以將1個卷積拆成2個卷積，
使得網絡深度進一步增加，**增加了網絡的非線性**（每增加一層都要進行RELU）。



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

訓練集的數據分佈和預測集的數據分佈不一致

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

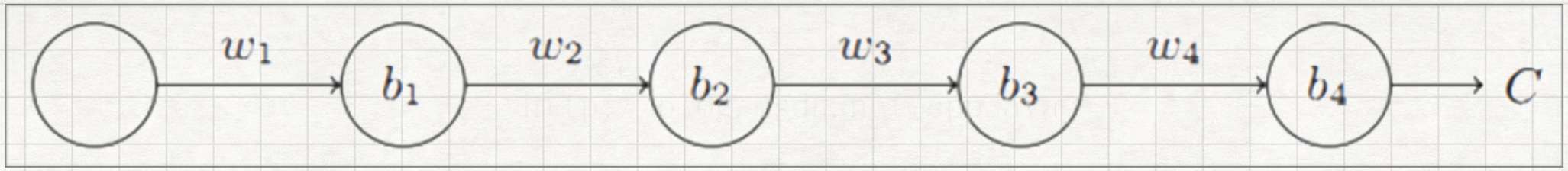
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Vanishing Gradient

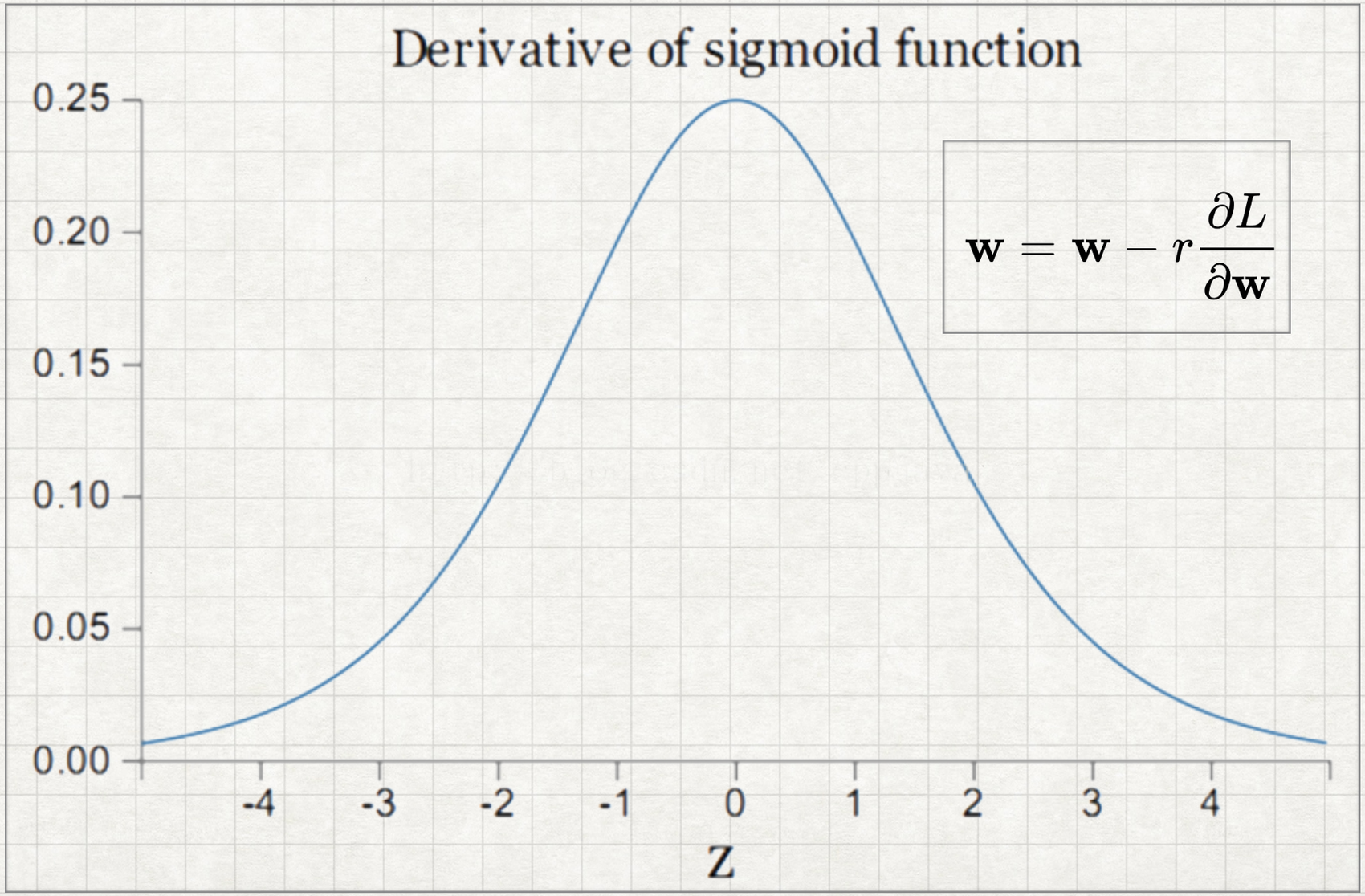
梯度消失

Covariate shift

訓練集的數據分佈和預測集的數據分佈不一致



$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\, w_2 \sigma'(z_2)\, w_3 \sigma'(z_3)\, w_4 \sigma'(z_4)\, \frac{\partial C}{\partial a_4}$$



$$\mathbf{w} = \mathbf{w} - r\frac{\partial L}{\partial \mathbf{w}}$$

InceptionV3

將7x7分解成兩個一維的捲積（1x7,7x1），3x3也是一樣（1x3,3x1），網絡輸入從224x224變為了299x299。

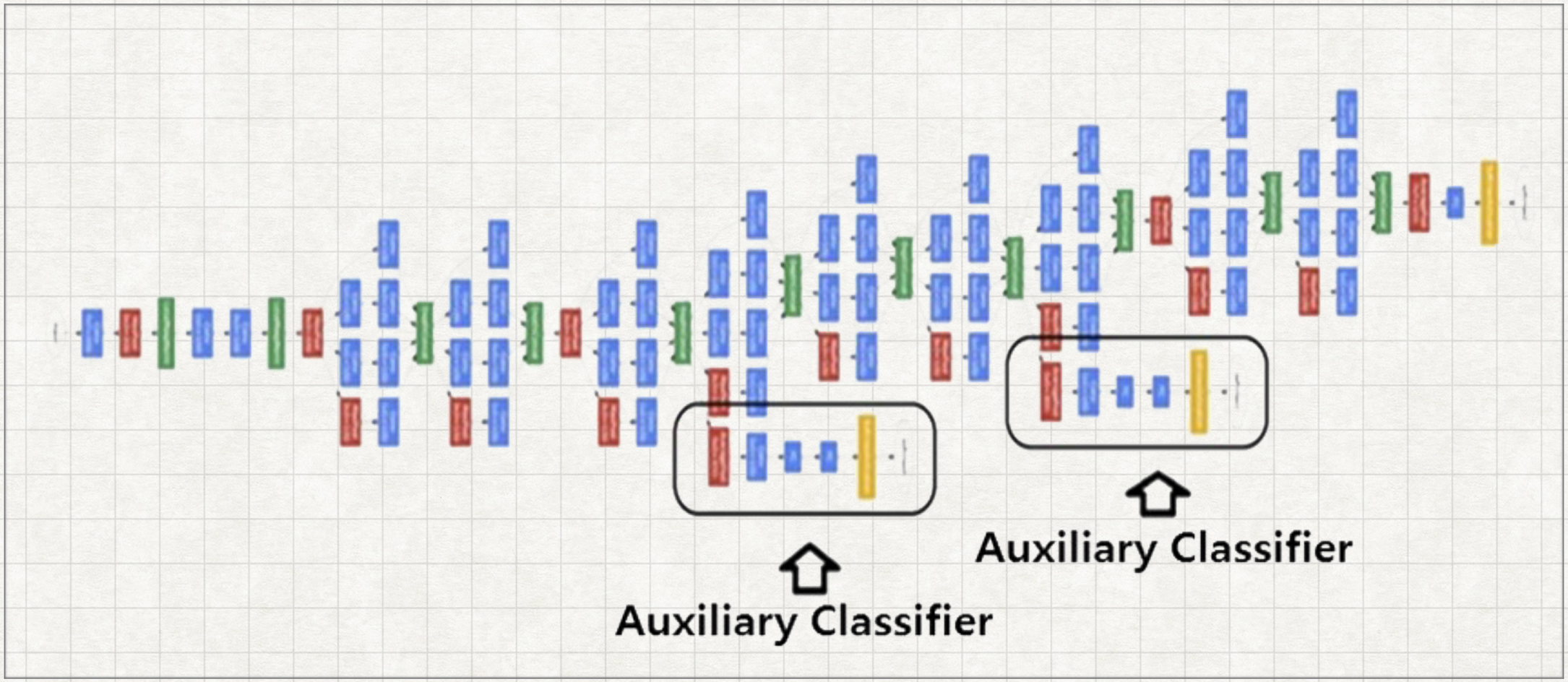
RMSProp優化器

輔助分類器使用了BatchNorm

Auxiliary classifiers :
to overcome the vanishing gradient problem

通過中間層的分類損失，監督網絡，起正則化作用並補充梯度。

並按一個較小的權重（0.3）加到最終分類結果中，這樣相當於做了模型融合，同時給網絡增加了反向傳播的梯度信號，也提供了額外的正則化



作者認為輔助分類器的功能是正則化，尤其是它們具備BatchNorm或Dropout操作時。

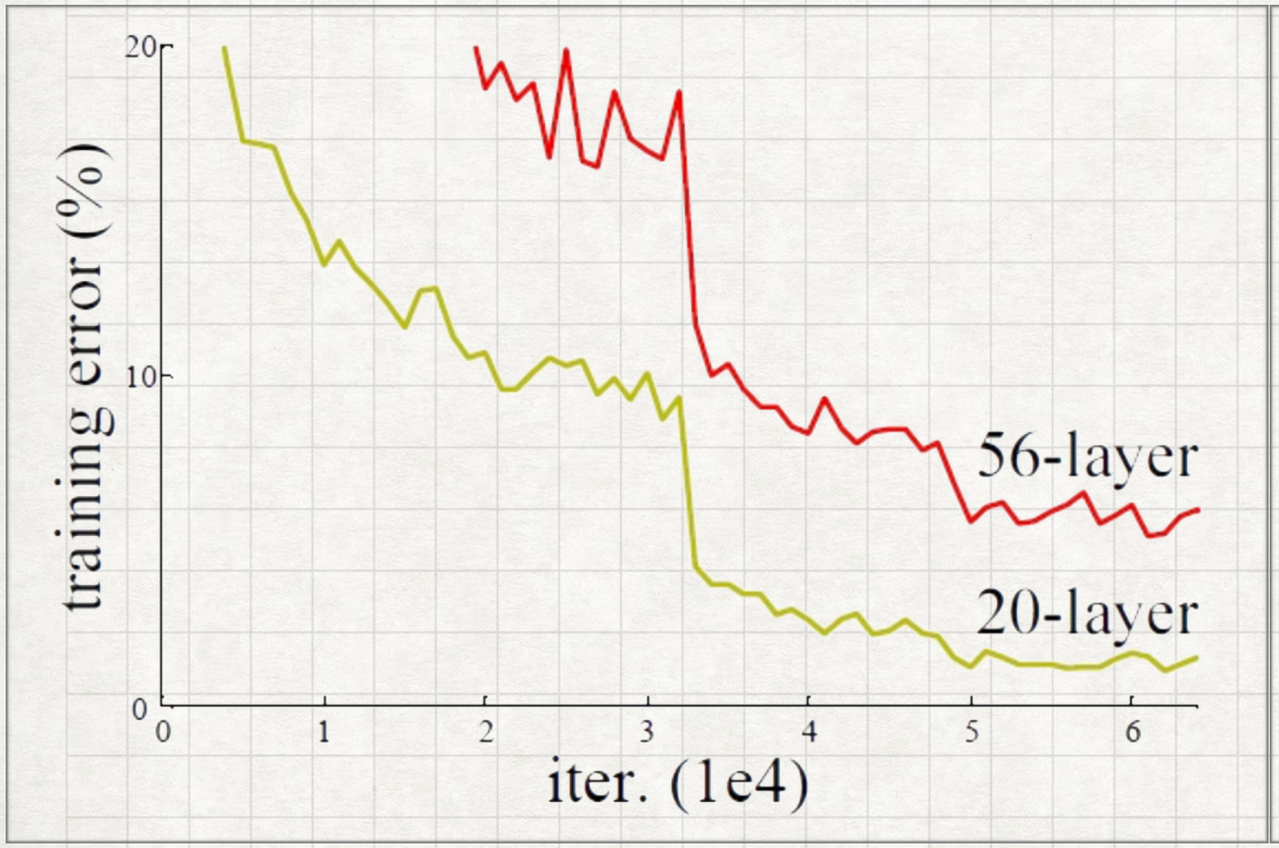
InceptionV4, Inception-Resnet V1, V2

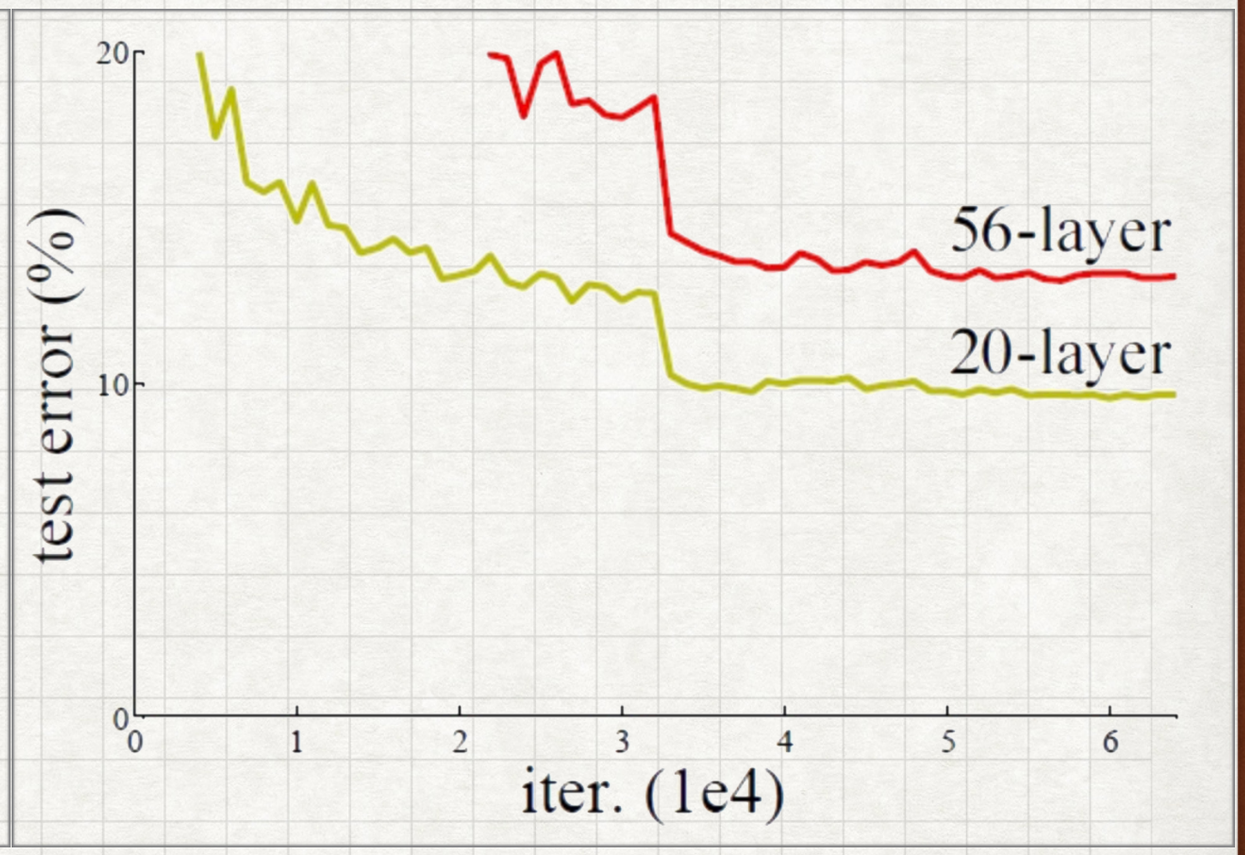
結合殘差連接可以顯著加速Inception的訓練。

也有一些證據表明殘差Inception網絡在相近的成本下略微超過沒有殘差連接的初始網絡。

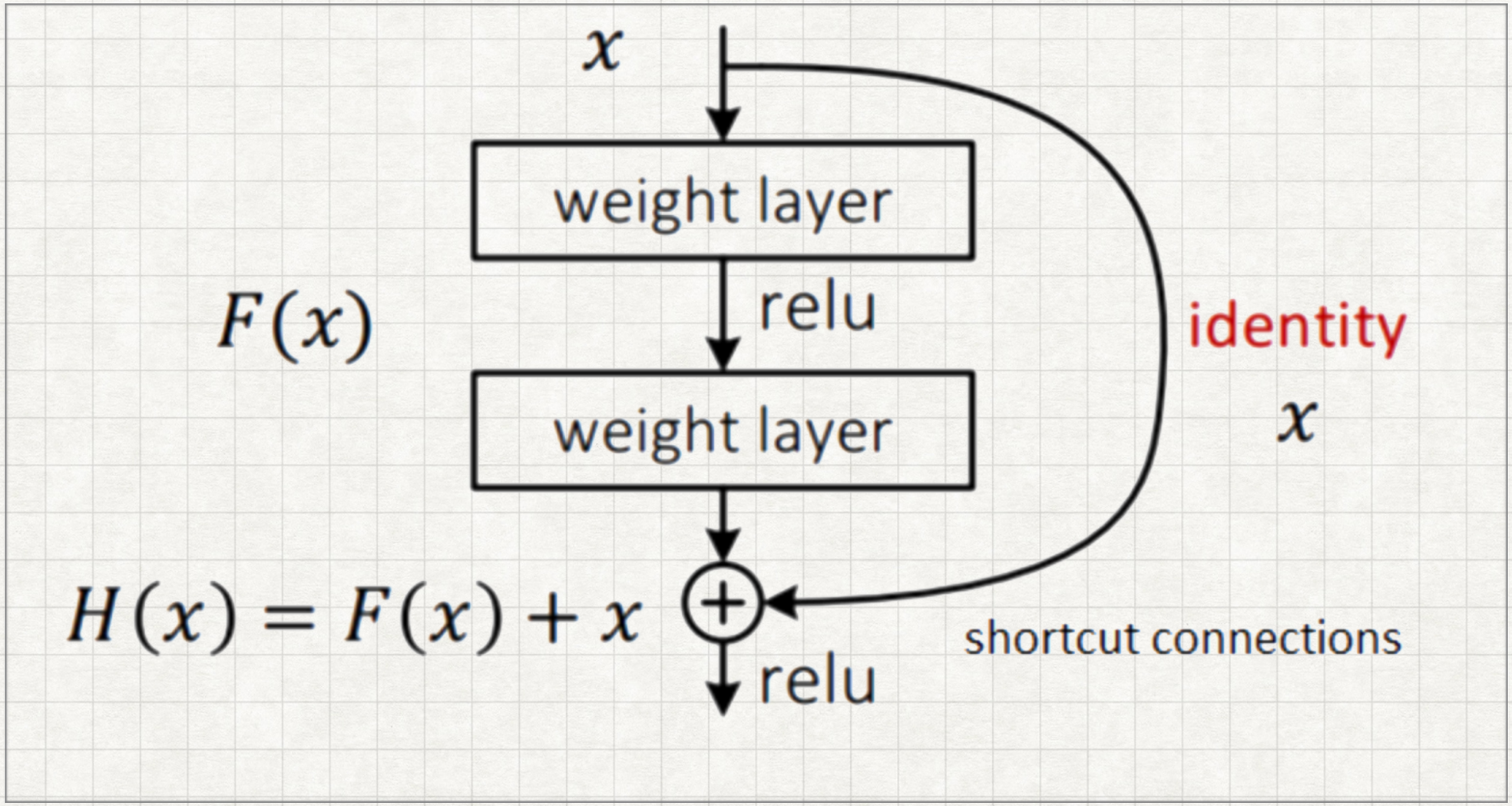
研究者還展示了多種新型殘差和非殘差Inception網絡的簡化架構

梯度消失


20
10
0
training error (%)
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer


20
10
0
test error (%)
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer

ResNet


x
weight layer
relu
$F(x)$
weight layer
identity
x
$H(x) = F(x) + x$
relu
shortcut connections

ResNet

$$y_l = h(x_l) + F(x_l, W_l)$$
$$x_{l+1} = f(y_l)$$

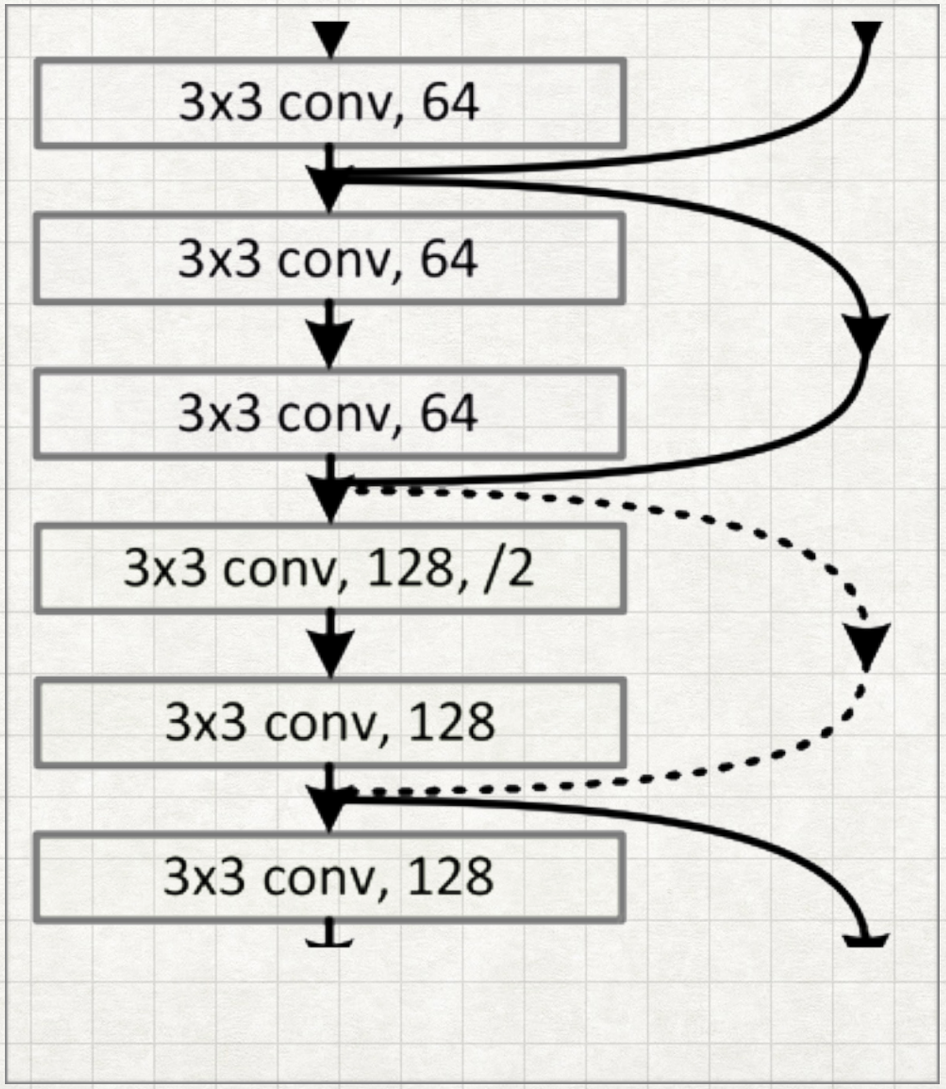
$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \left(1 + \frac{\partial}{\partial x_L} \sum_{i=l}^{L-1} F(x_i, W_i)\right)$$

ResNet有2個基本的塊

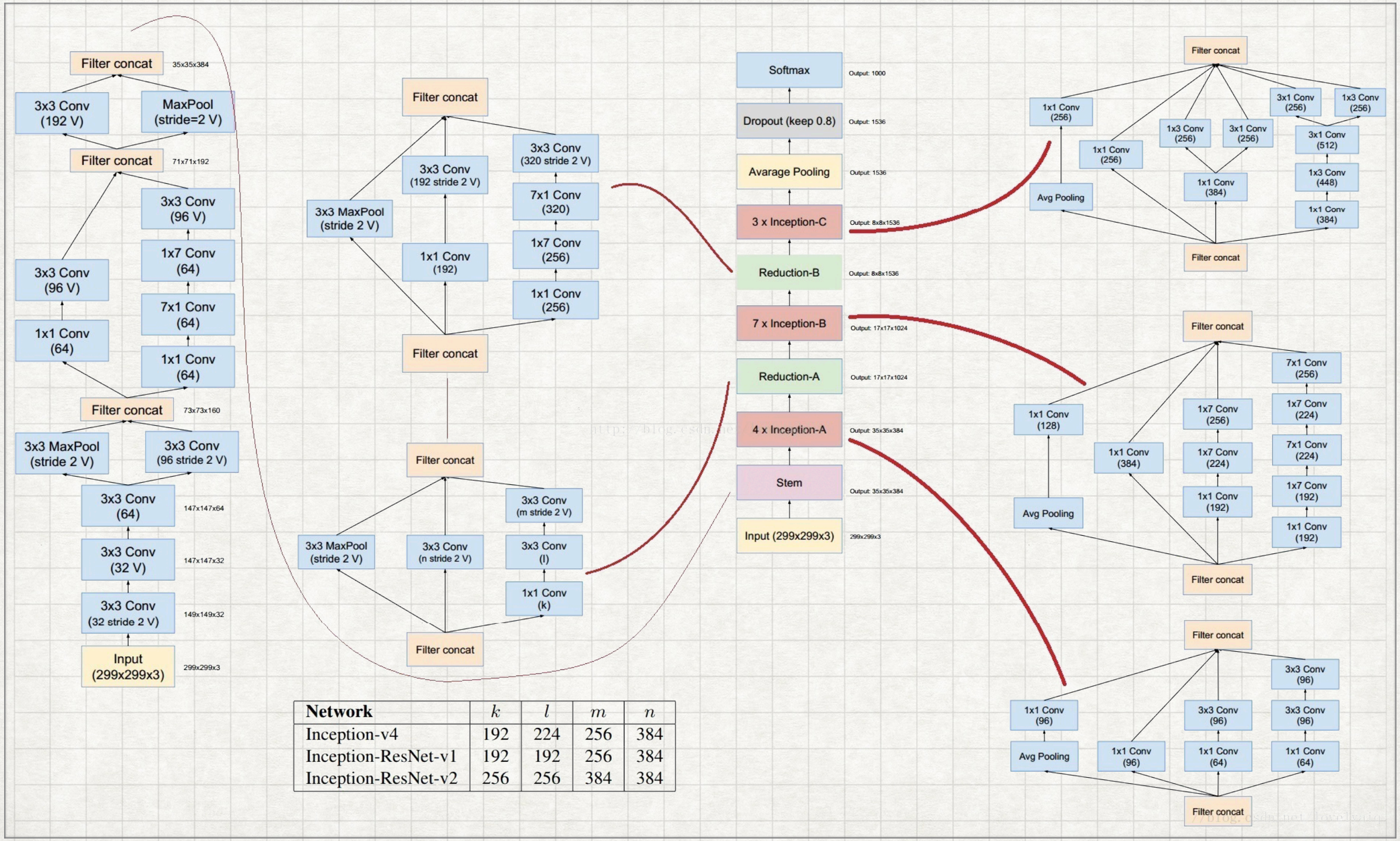
Identity Block，輸入和輸出的維度是一樣的

Conv Block，輸入和輸出的維度是不一樣的



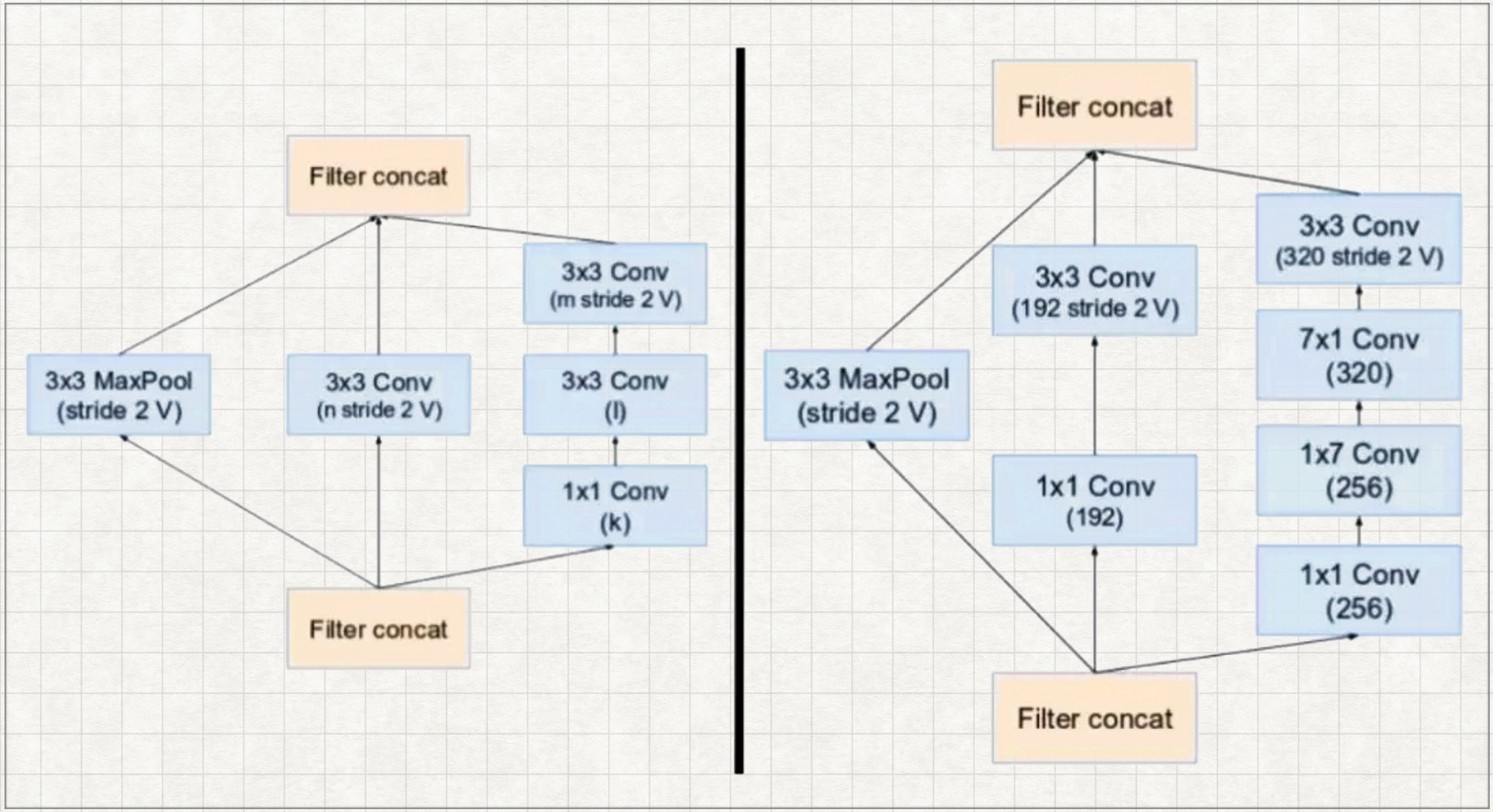
```python
def convolutional_block(self, X_input, kernel_size, in_filter,
                                out_filters, stage, block, training, stride=2):
        """

        # defining name basis
        block_name = 'res' + str(stage) + block
        with tf.variable_scope(block_name):
            f1, f2, f3 = out_filters

            x_shortcut = X_input
            #first
            W_conv1 = self.weight_variable([1, 1, in_filter, f1])
            X = tf.nn.conv2d(X_input, W_conv1,strides=[1, stride, stride, 1],padding='VALID')
            X = tf.layers.batch_normalization(X, axis=3, training=training)
            X = tf.nn.relu(X)

            #second
            W_conv2 = self.weight_variable([kernel_size, kernel_size, f1, f2])
            X = tf.nn.conv2d(X, W_conv2, strides=[1,1,1,1], padding='SAME')
            X = tf.layers.batch_normalization(X, axis=3, training=training)
            X = tf.nn.relu(X)

            #third
            W_conv3 = self.weight_variable([1,1, f2,f3])
            X = tf.nn.conv2d(X, W_conv3, strides=[1, 1, 1,1], padding='VALID')
            X = tf.layers.batch_normalization(X, axis=3, training=training)

            #shortcut path
            W_shortcut = self.weight_variable([1, 1, in_filter, f3])
            x_shortcut = tf.nn.conv2d(x_shortcut, W_shortcut, strides=[1, stride, stride, 1], padding='VALID')

            #final
            add = tf.add(x_shortcut, X)
            add_result = tf.nn.relu(add)

        return add_result
```

Inception V4



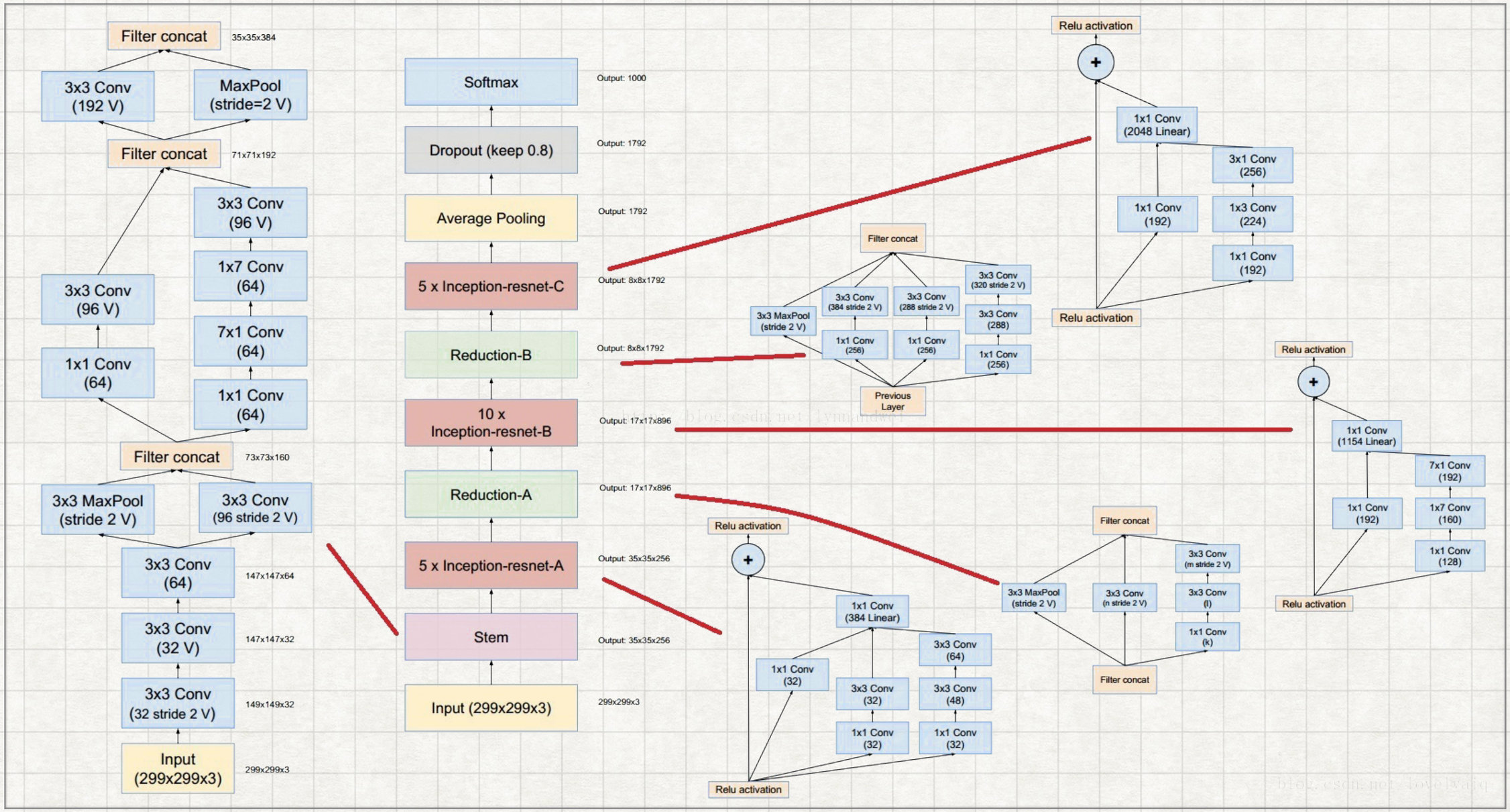
Network	k	l	m	n
Inception-v4	192	224	256	384
Inception-ResNet-v1	192	192	256	384
Inception-ResNet-v2	256	256	384	384

Reduction block

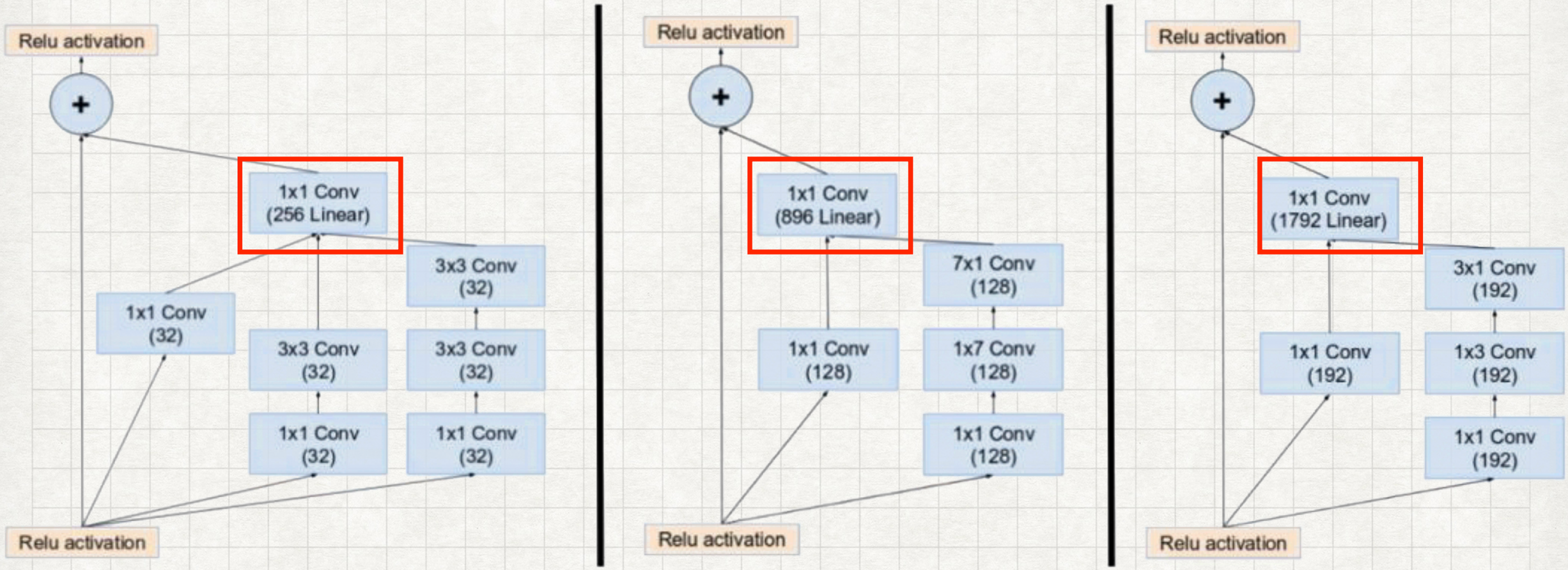


從35x35到17x17的尺寸縮減）和縮減塊B（從17x17到8x8的尺寸縮減）

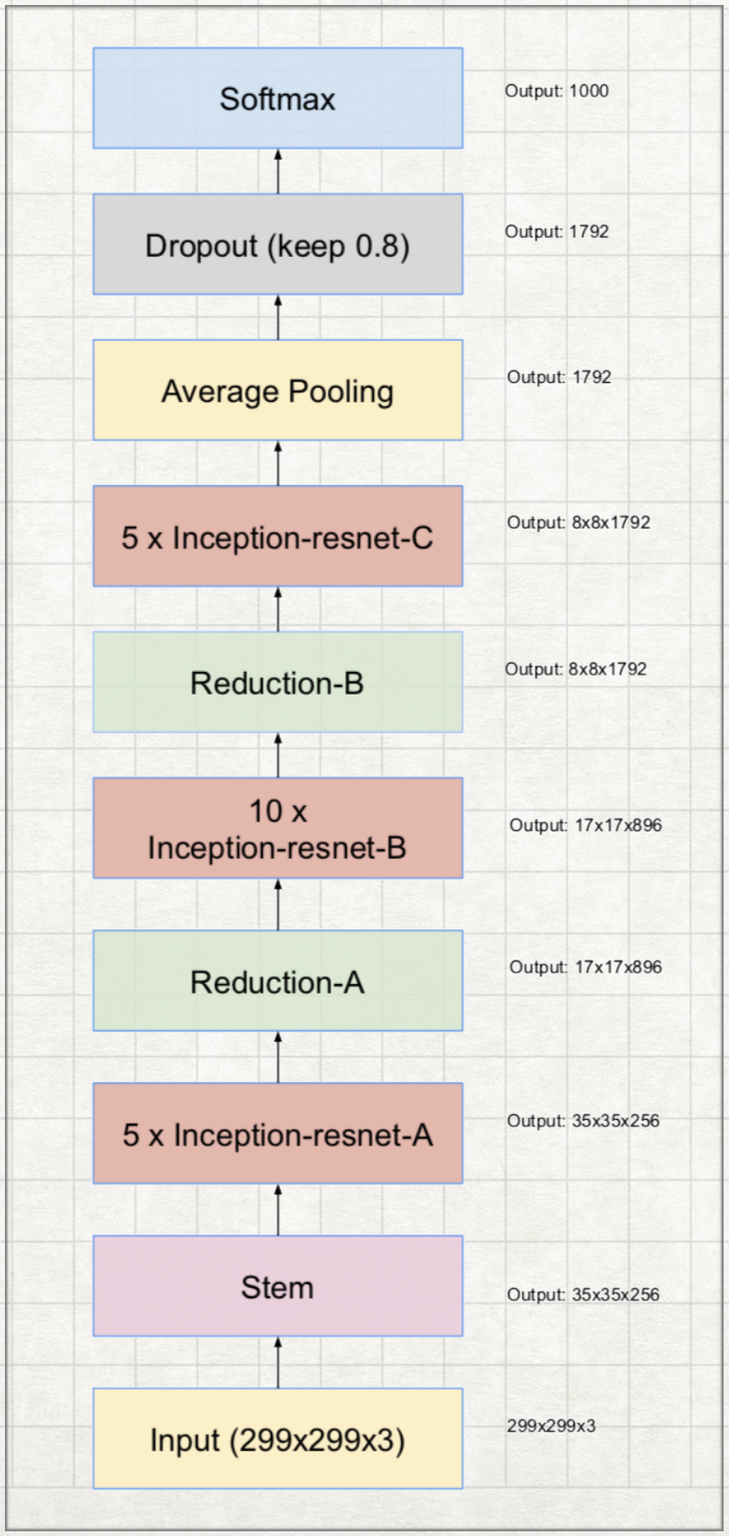
Inception-Resnet



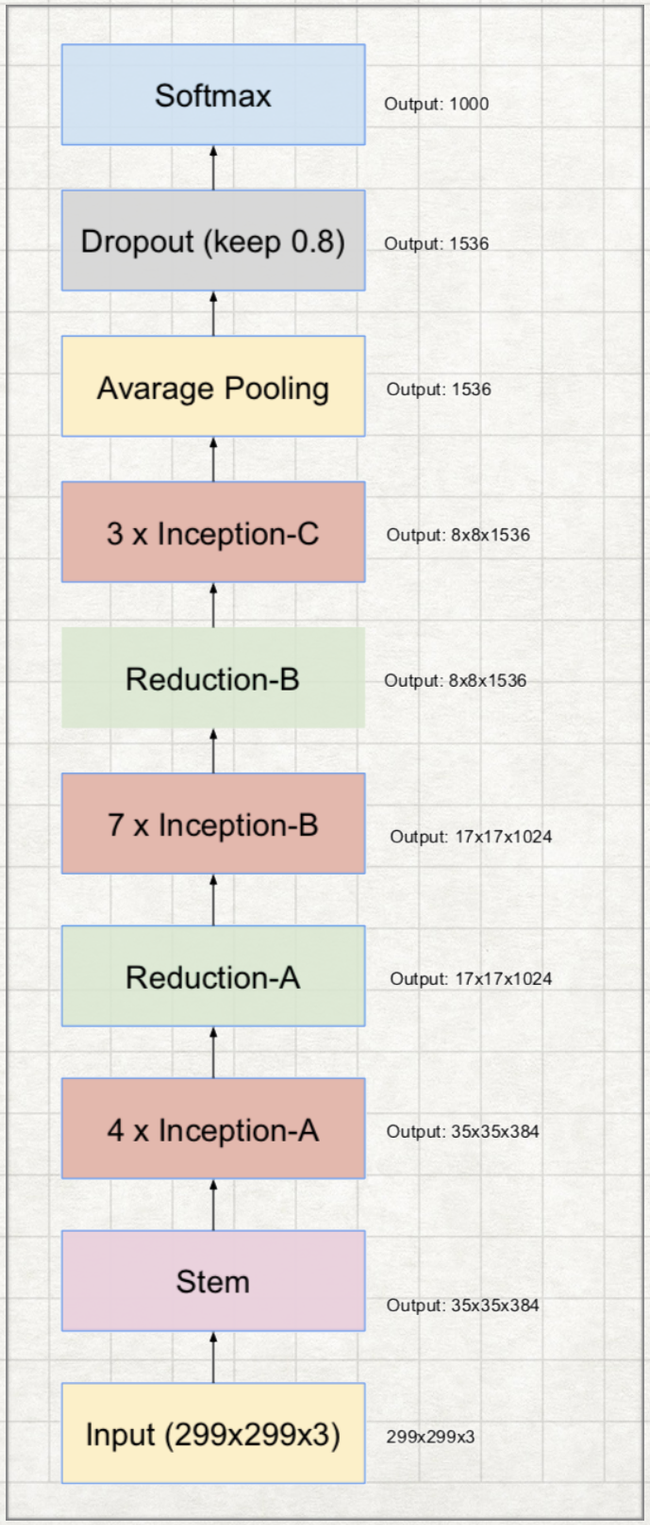
Inception-Resnet-A、B、C



Inception-Resnet V1



Inception V4



Inception-ResNet v1的計算成本和Inception v3的接近。

Inception-ResNetv2的計算成本和Inception v4的接近。

Stem

Inception-Resnet V1

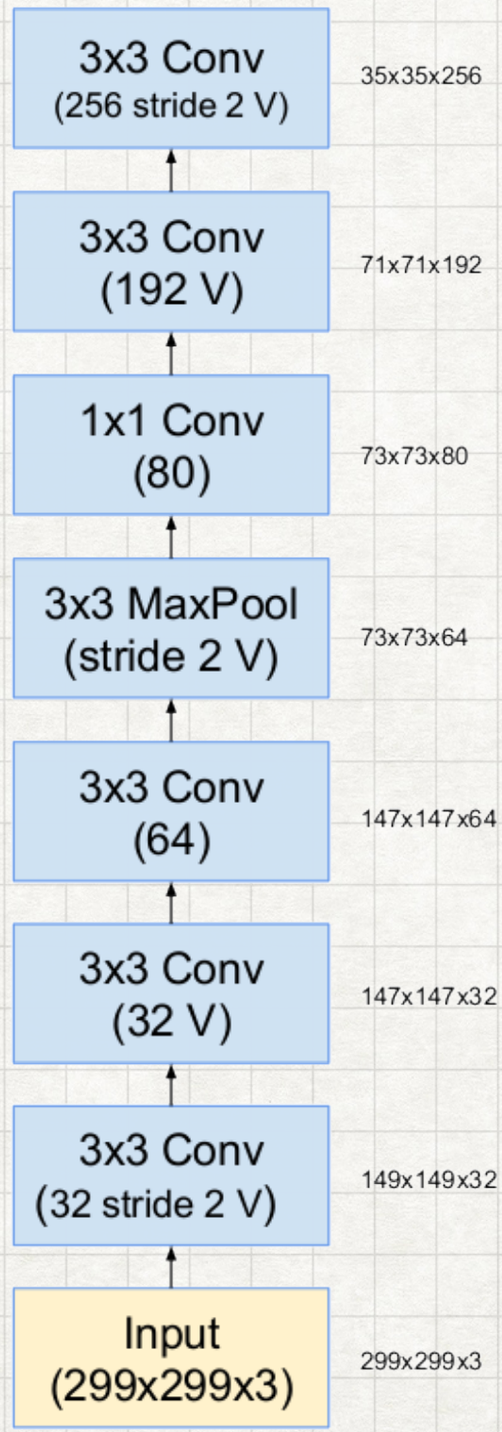
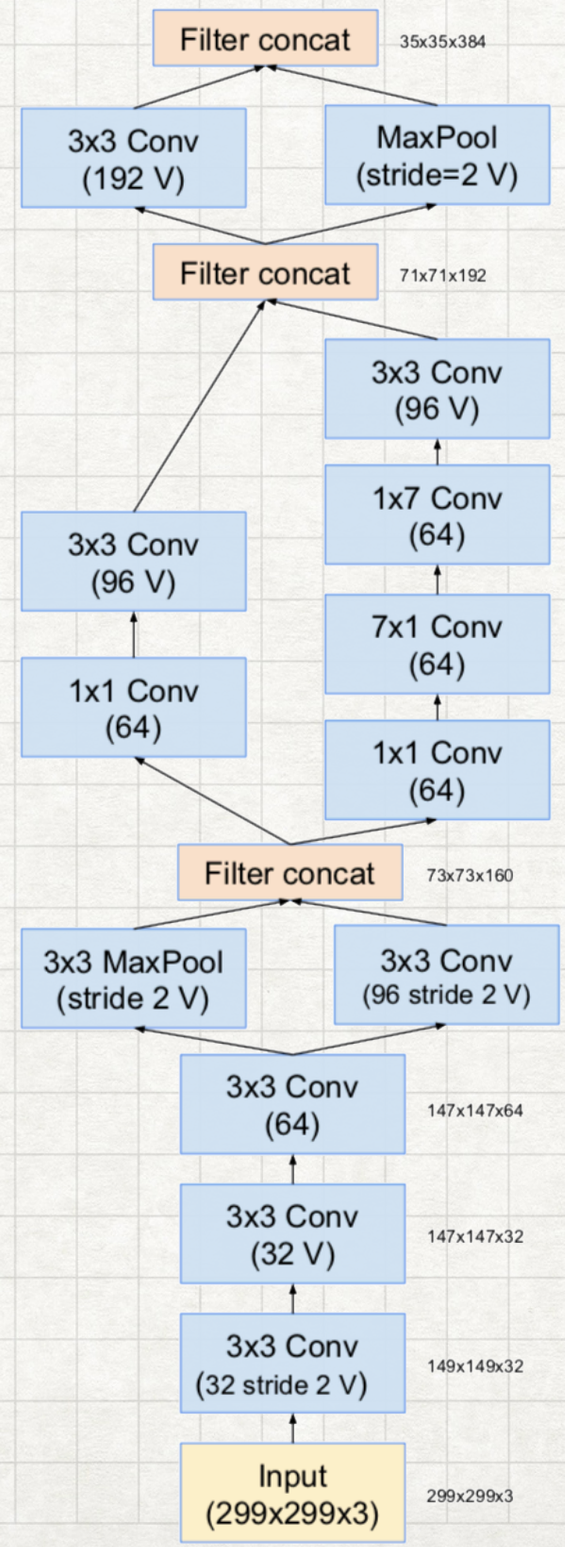


Figure 14. The stem of the Inception-ResNet-v1 network.

http://blog.csdn.net/love1yang

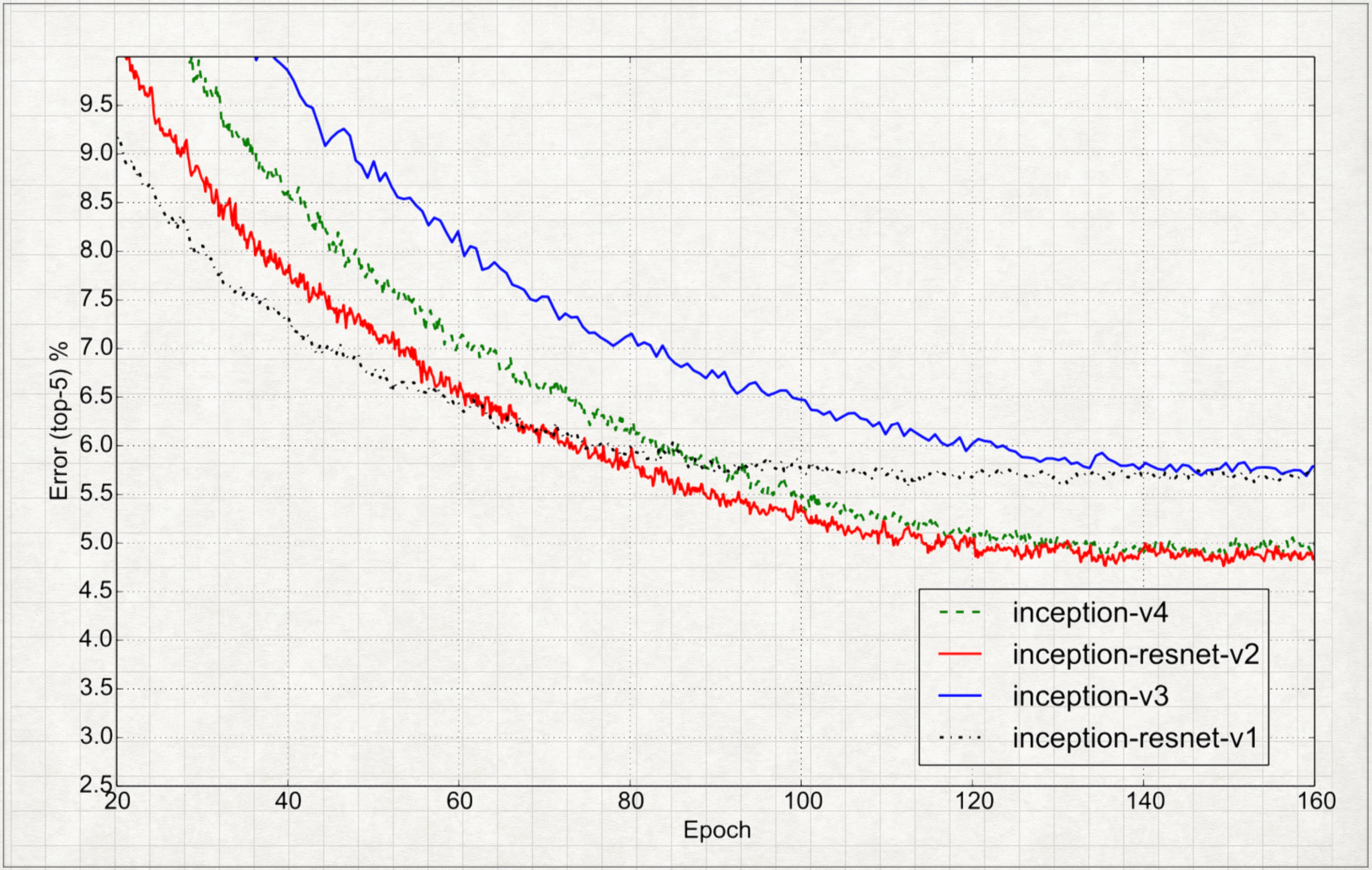
Inception V4



Inception-ResNet v1

Inception-ResNetv2

兩個子版本都有相同的模塊A，B，C和縮減塊結構。唯一的不同在於**超參數設置**。


Error (top-5) %
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
20
40
60
80
100
120
140
160
Epoch
inception-v4
inception-resnet-v2
inception-v3
inception-resnet-v1