Highway，Residual and Dense Network Learning

赵洲

浙江大学计算机学院

HWN的研究动机

- 深度学习网络梯度消失/爆炸问题（Vanishing and Exploding Gradient）。

$$C = \text{sigmoid}(W_4 * H_3 + b_4)$$

$$H_3 = \text{sigmoid}(W_3 * H_2 + b_3)$$

$$H_2 = \text{sigmoid}(W_2 * H_1 + b_2)$$

$$H_1 = \text{sigmoid}(W_1 * x + b_1)$$

$$\frac{\delta C}{\delta W_1} = \frac{\delta C}{\delta H_3} * \frac{\delta H_3}{\delta H_2} * \frac{\delta H_2}{\delta H_1} * \frac{\delta H_1}{\delta W_1}$$

$$W = W - lr * g(t)$$

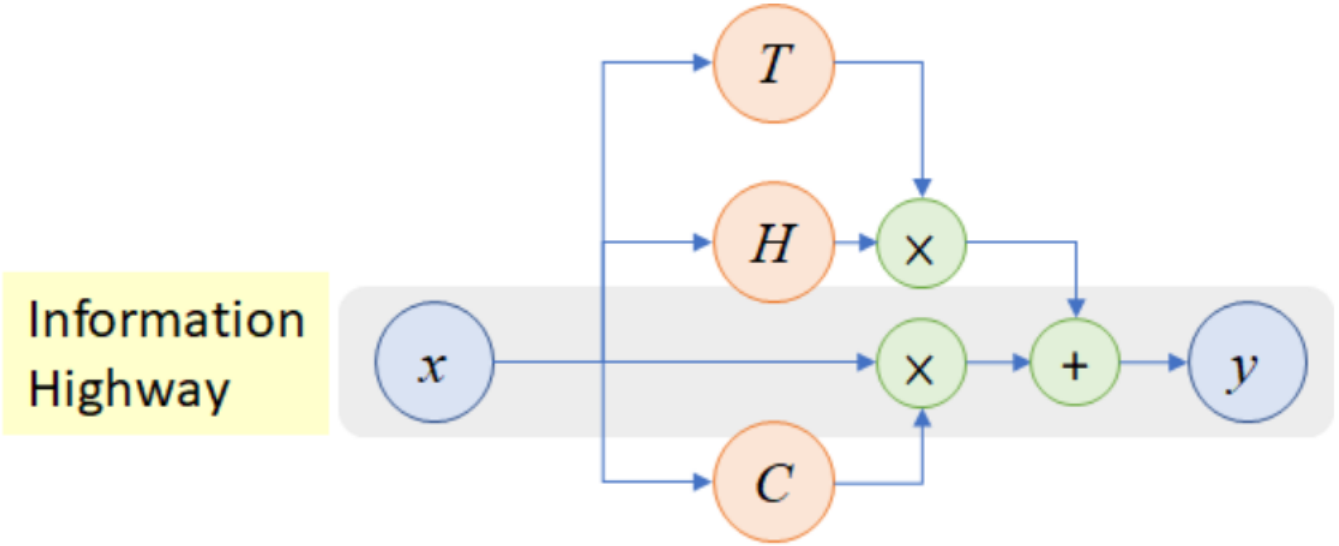
forward $h_l = w_l^T h_{l-1}$

backward $\frac{\partial l}{\partial h_{l-1}} = \frac{\partial l}{\partial h_l} \cdot \frac{\partial h_l}{\partial h_{l-1}} = \frac{\partial l}{\partial h_l} w_l$

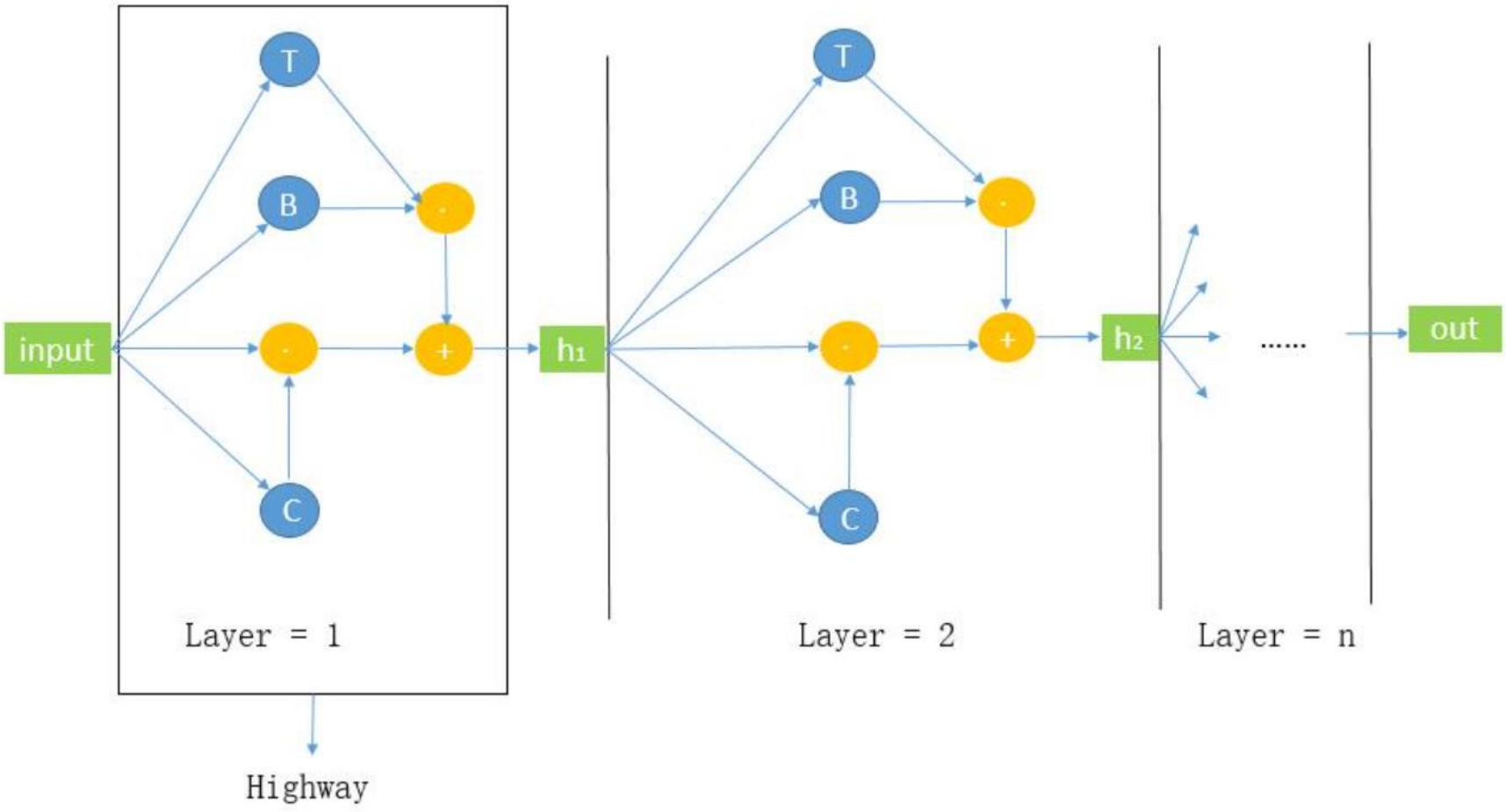
$\Rightarrow \frac{\partial l}{\partial h_k} = \frac{\partial l}{\partial h_t} \underbrace{\prod_{i=k+1}^{t} w_i}$ $\Rightarrow 0.9^{100}$

HWN网络架构

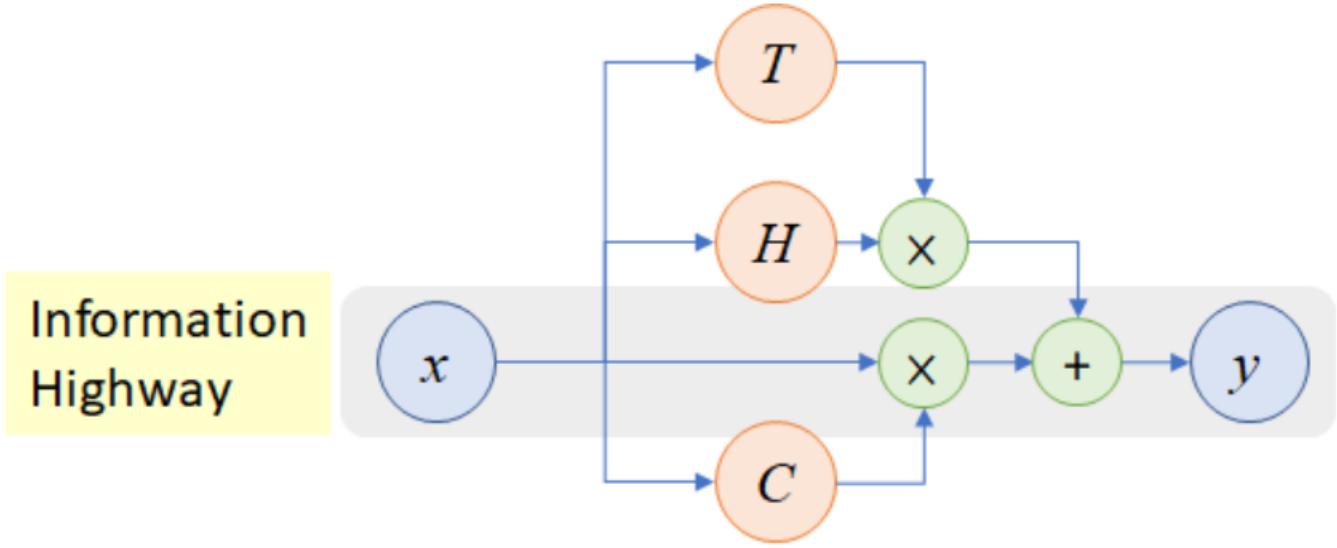
$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C})$$



多层HWN网络架构



HWN网络架构解释

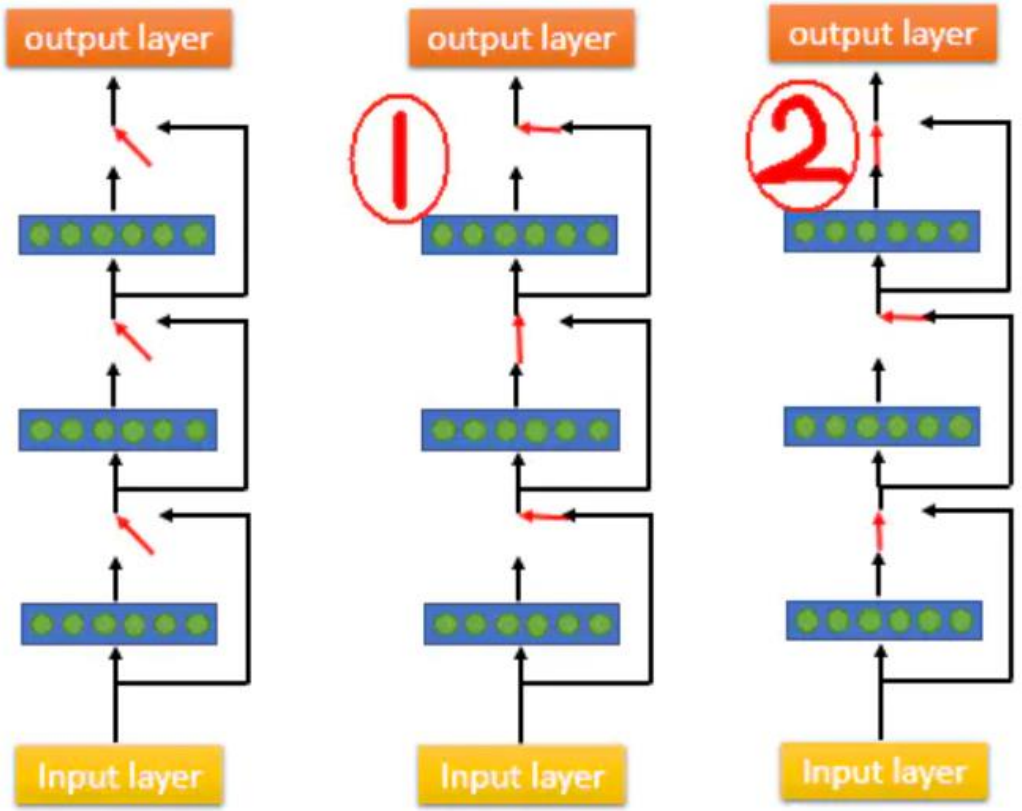


$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$

$$T(\mathbf{x}) = \sigma(\mathbf{W_T}^T \mathbf{x} + \mathbf{b_T})$$

HWN网络功能



$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$

结果对比

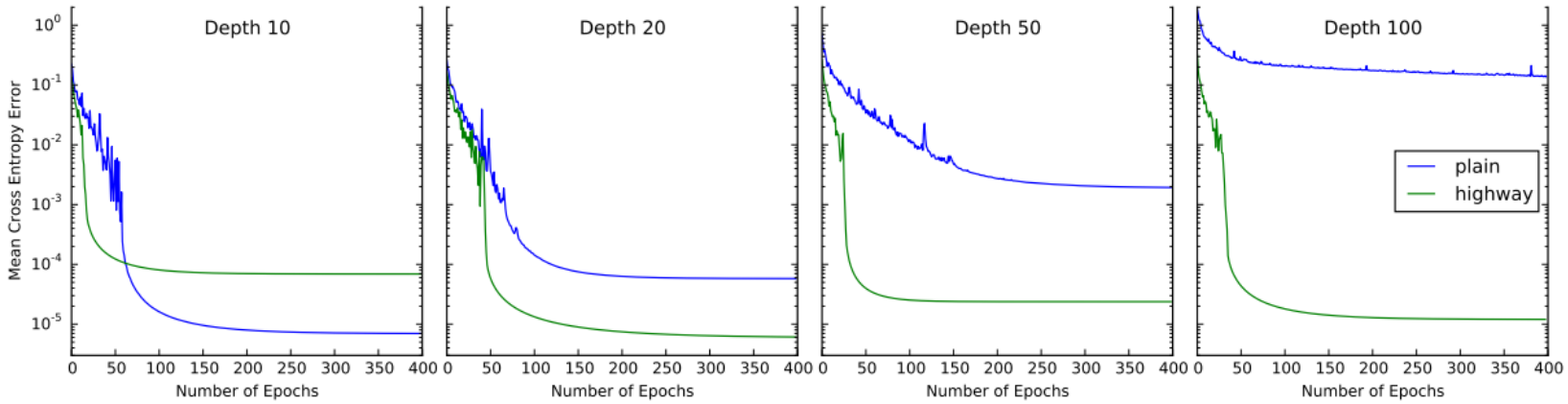


Figure 1. Comparison of optimization of plain networks and highway networks of various depths. All networks were optimized using SGD with momentum. The curves shown are for the best hyperparameter settings obtained for each configuration using a random search. Plain networks become much harder to optimize with increasing depth, while highway networks with up to 100 layers can still be optimized well.

结果对比

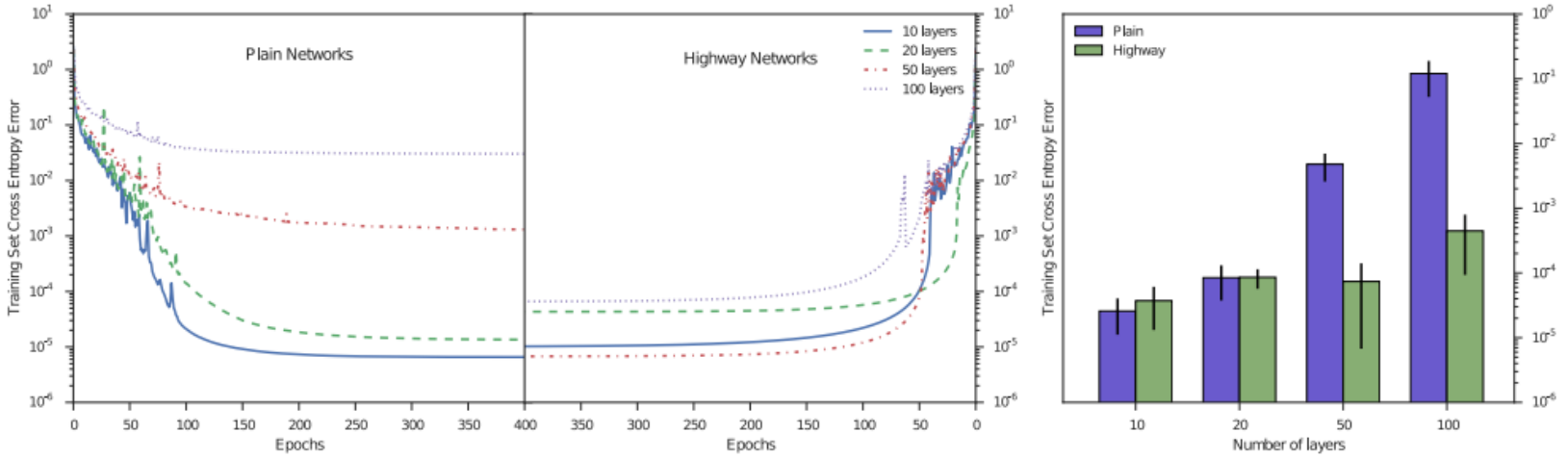


Figure 1: Comparison of optimization of plain networks and highway networks of various depths. *Left:* The training curves for the best hyperparameter settings obtained for each network depth. *Right:* Mean performance of top 10 (out of 100) hyperparameter settings. Plain networks become much harder to optimize with increasing depth, while highway networks with up to 100 layers can still be optimized well. Best viewed on screen (larger version included in Supplementary Material).

DRN的研究动机

- 解决**网络退化问题**。

When we increase the number of layers, there is a common problem in deep learning associated with that called the **Vanishing/Exploding gradient**. This causes the gradient to become 0 or too large. Thus when we increases number of layers, **the training and test error rate also increases**.

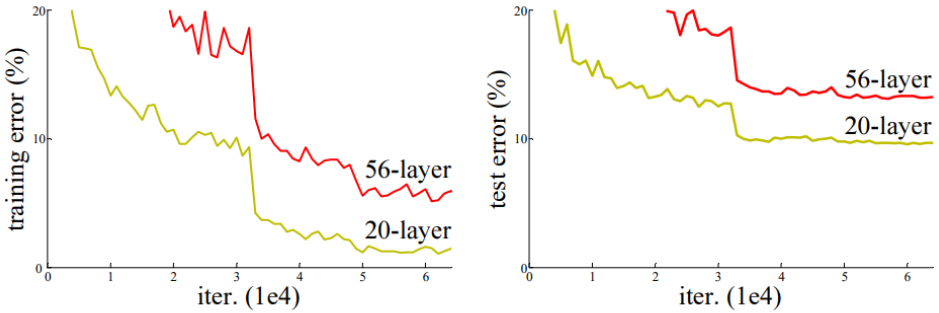
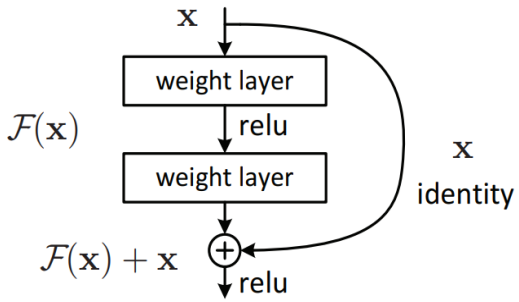


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

"残差在数理统计中是指实际观察值与估计值（拟合值）之间的差。"

DRN网络结构

- 当网络退化时，浅层网络能够达到比深层网络更好的训练效果，这时如果我们把低层的特征传到高层，那么效果应该至少不比浅层的网络效果差。
- Identify Mapping by Shortcuts: $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$



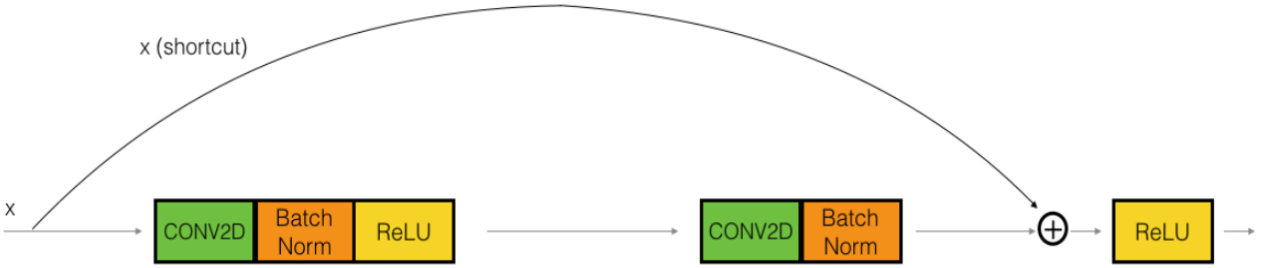
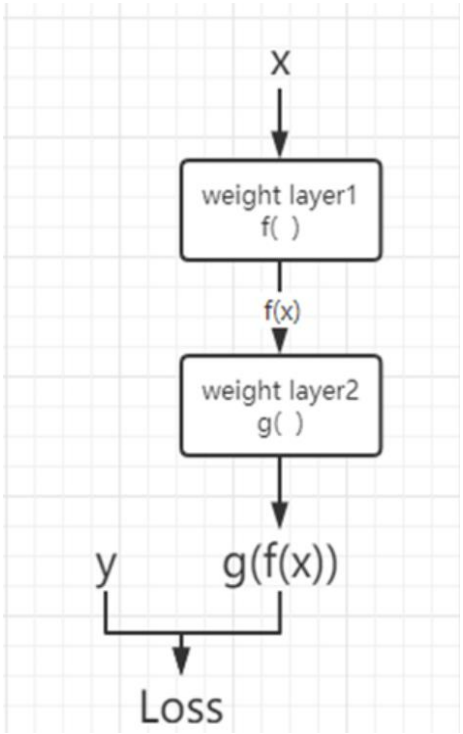


Figure 2. Residual learning: a building block.

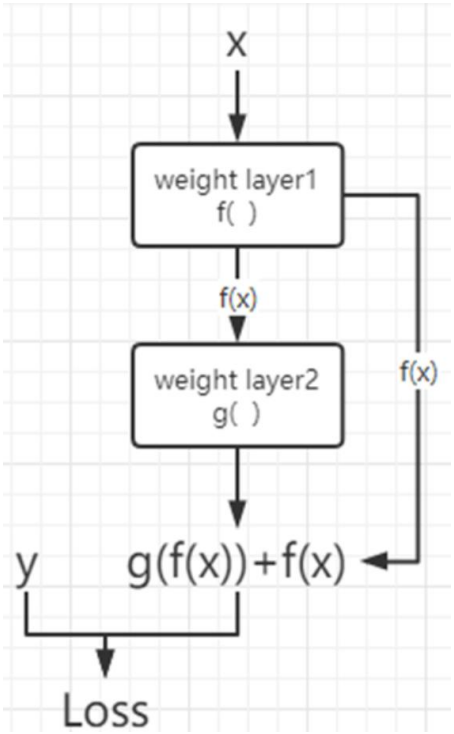
DRN解决梯度消失问题

- 普通网络



$$\frac{\partial Loss}{\partial w1}=\frac{\partial Loss}{\partial g(f(x))}\frac{\partial g(f(x))}{\partial f(x)}\frac{\partial f(x)}{\partial w1},\quad \frac{\partial Loss}{\partial w2}=\frac{\partial Loss}{\partial g(f(x))}\frac{\partial g(f(x))}{\partial w2}$$

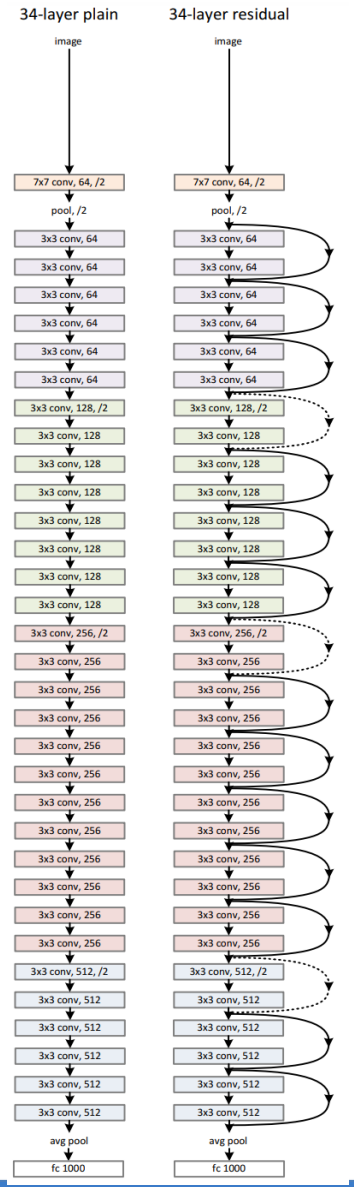
- 残差网络



$$\frac{\partial Loss}{\partial w1}=\frac{\partial Loss}{\partial(g(f(x))+f(x))}\left(\frac{\partial g(f(x))}{\partial f(x)}\frac{\partial f(x)}{\partial w1}+\frac{\partial f(x)}{\partial w1}\right)$$

通过改变模型输出值的构成以在梯度计算时融入加法，使得当个别梯度计算时即使部分项过小/趋近于0时，信息传播也不会消失

网络对比



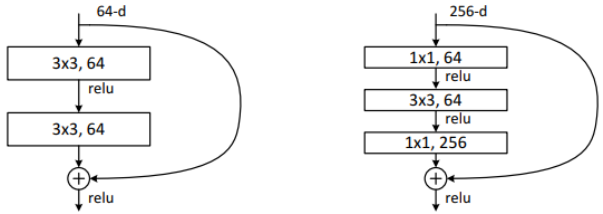


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

结果对比

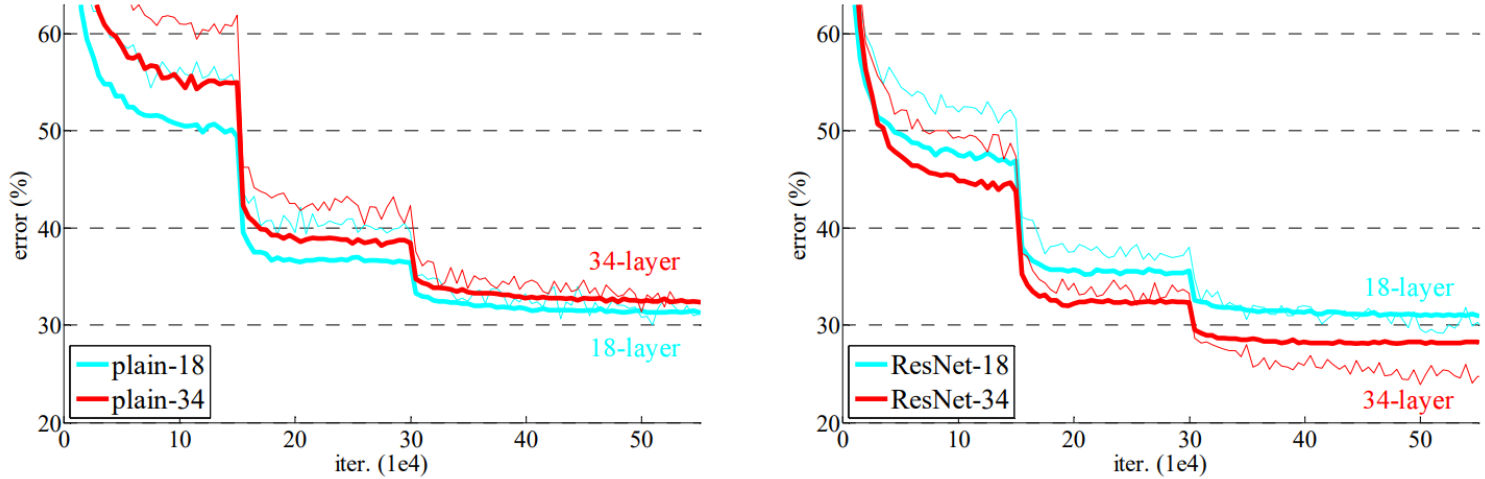


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

层数效果表现

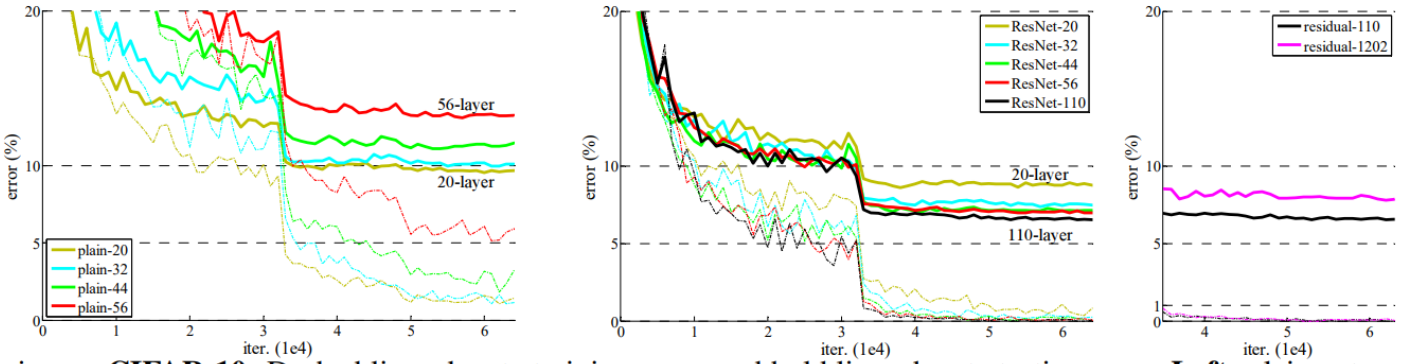


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

层参数激活分析

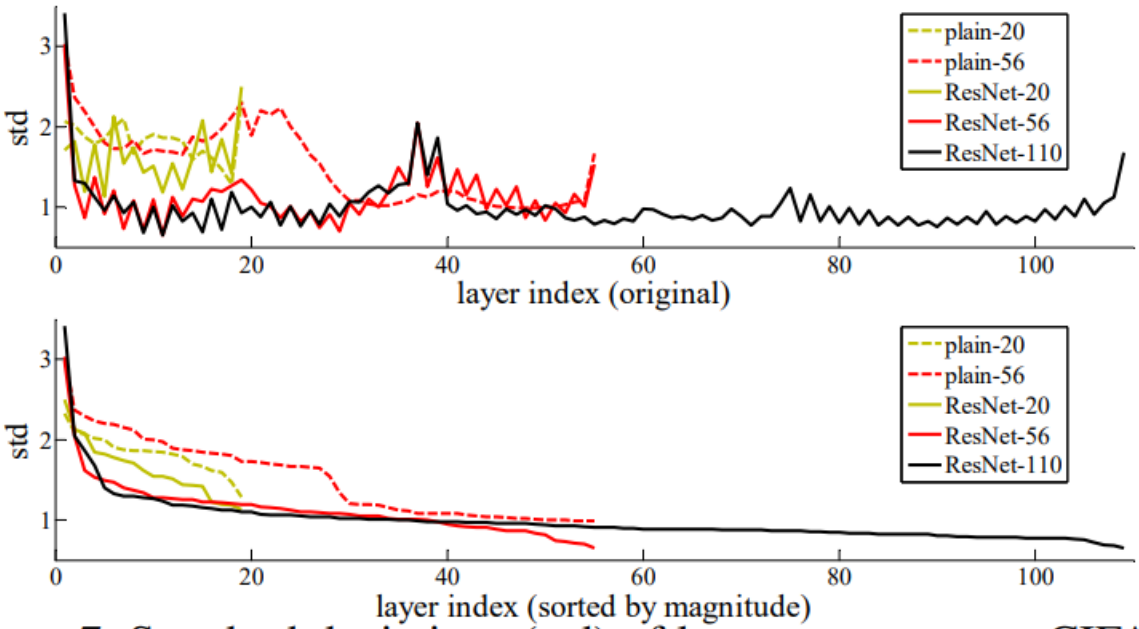


Figure 7. Standard deviations (std) of layer responses on CIFAR-10. The responses are the outputs of each 3×3 layer, after BN and before nonlinearity. **Top**: the layers are shown in their original order. **Bottom**: the responses are ranked in descending order.

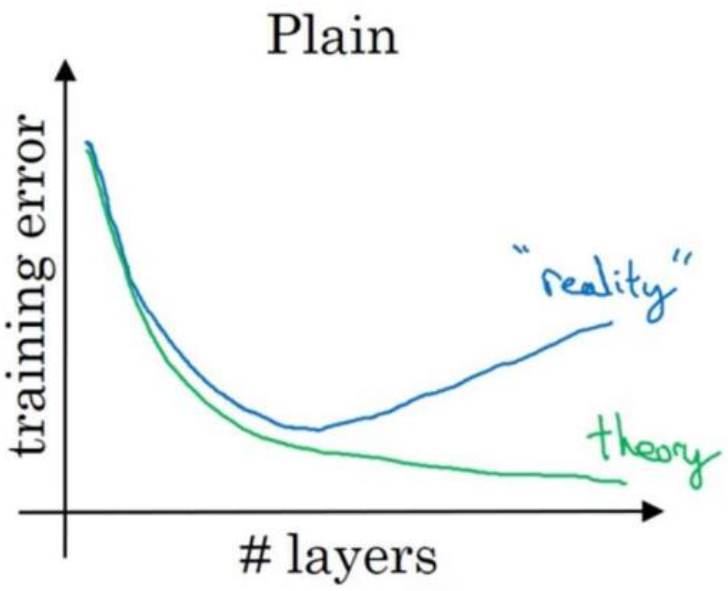
DRN代码实现（基本）

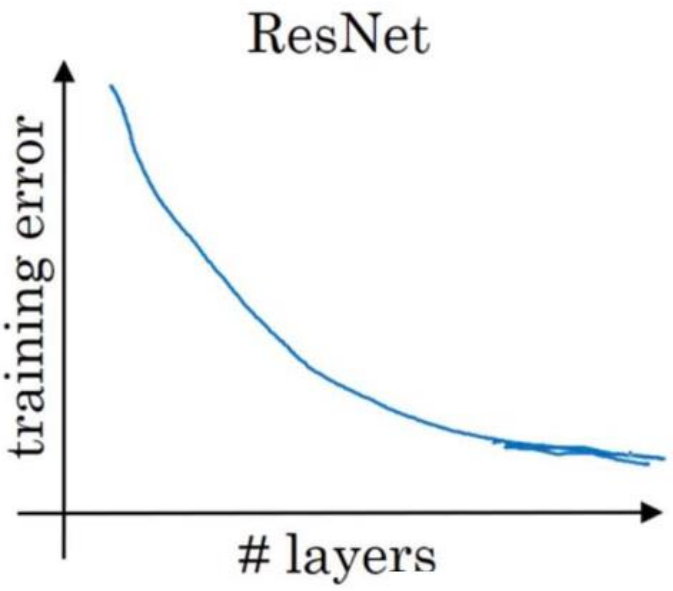
```
import torch
import torch.nn as nn
import numpy as np


class ResNet(nn.Module):
    def __init__(self):
        super(ResNet, self).__init__()
        self.conv = nn.Conv2d(3, 6, 3, 1, padding=1)

    def forward(self, x):
        # 这里的维度要一致
        return x.repeat(1, 2, 1, 1) + self.conv(x)

if __name__ == '__main__':
    net = ResNet()
    x = torch.randn(1, 3, 6, 6)
    print(net(x).shape)
```

DRN代码实现

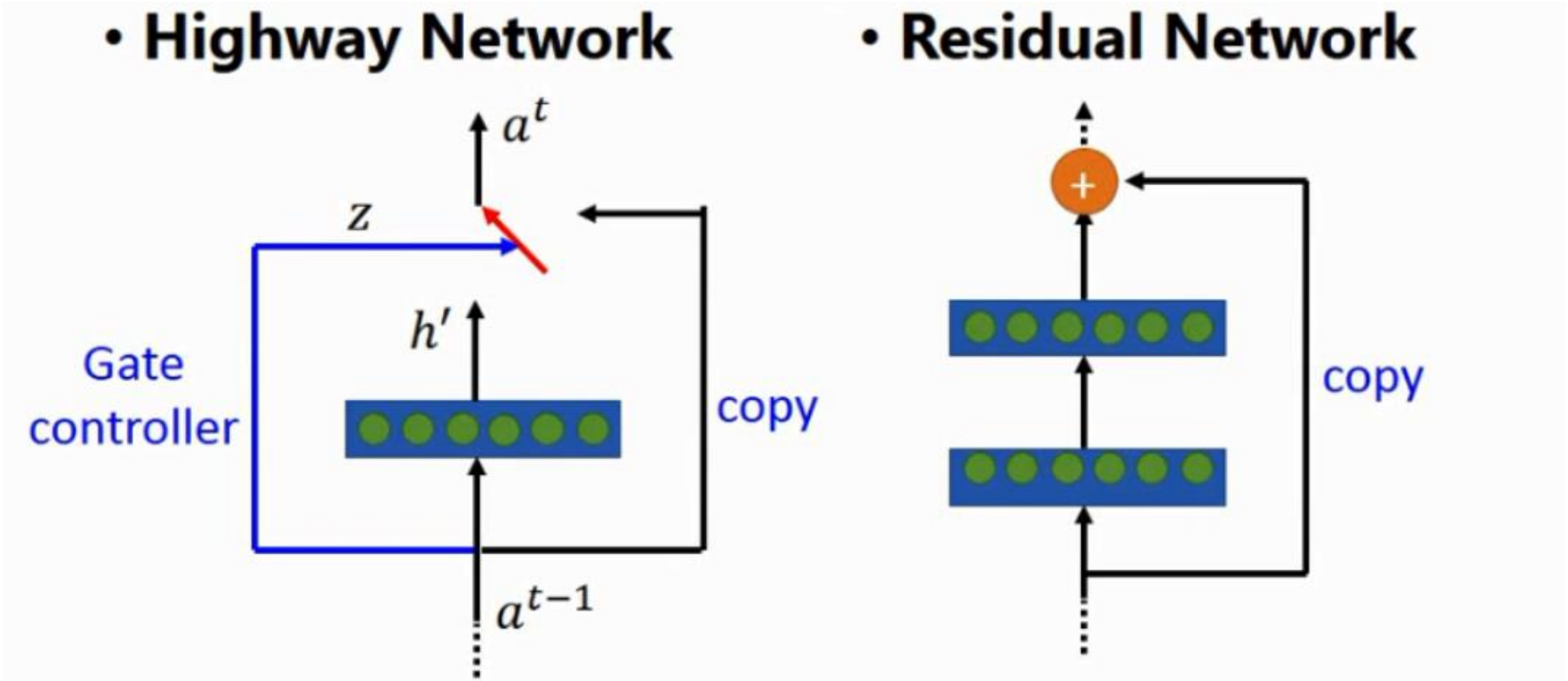
```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_feature, out_feature, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_feature, out_feature, kernel_size=3, stride=stride, paddi
        self.bn1 = nn.BatchNorm2d(out_feature)
        self.conv2 = nn.Conv2d(out_feature, out_feature, kernel_size=3, stride=1, padding=1
        self.bn2 = nn.BatchNorm2d(out_feature)
        self.shortcut = nn.Sequential()
        """
        经过处理后的x要与x的维度相同(尺寸和深度)
        如果不相同，需要添加卷积+BN来变换为同一维度
        """
        if stride != 1 or in_feature != self.expansion * in_feature:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_feature, self.expansion*in_feature, kernel_size=1, stride=stri
                nn.BatchNorm2d(self.expansion * in_feature)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

总结


Plain
training error
"reality"
theory
layers


ResNet
training error
layers

网络对比


• Highway Network
a^t
z
h'
Gate
controller
copy
a^{t-1}
• Residual Network
+
copy

DenseNet的研究动机

- Highway, Residual网络通过**恒等映射**，缓解梯度消失和模型退化问题（“create short paths from early layers to later layers”）。
- Dense网络采用的思路为“与其多次学习冗余特征，特征复用是一种更好地特征提取方式”，即每一层的输入来自前面所有层的输出。

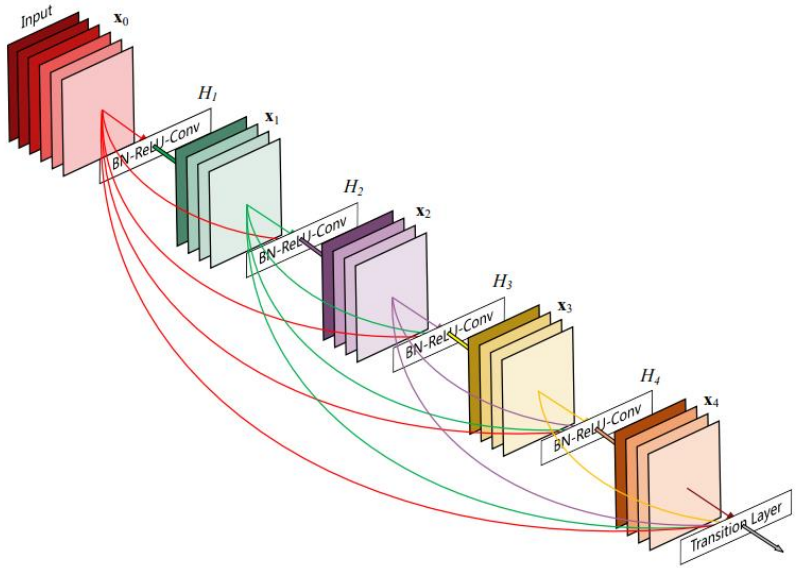
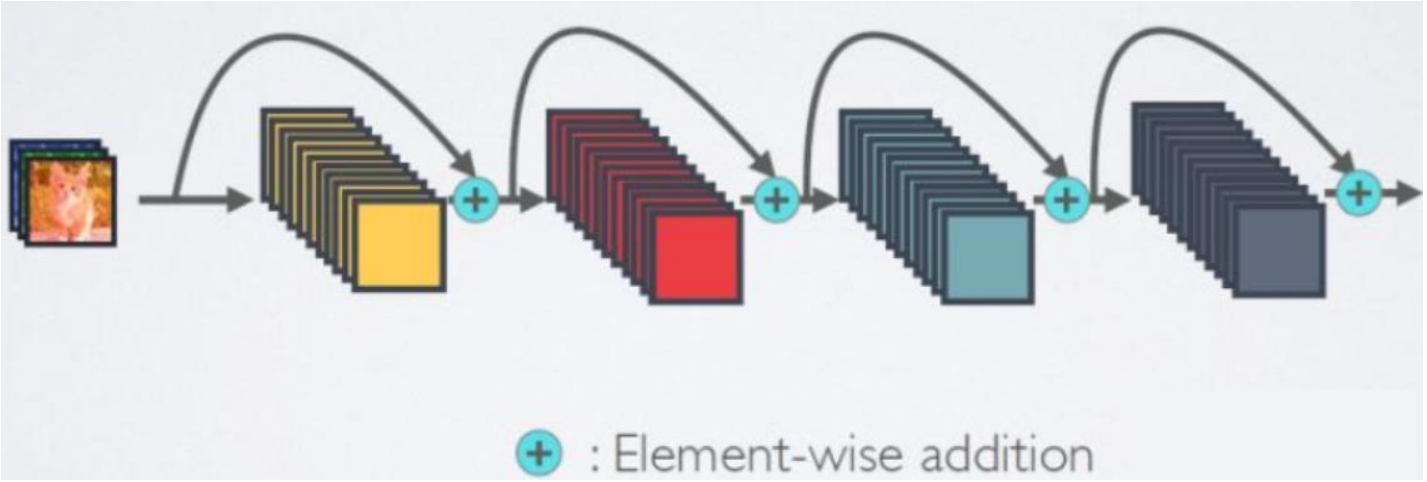
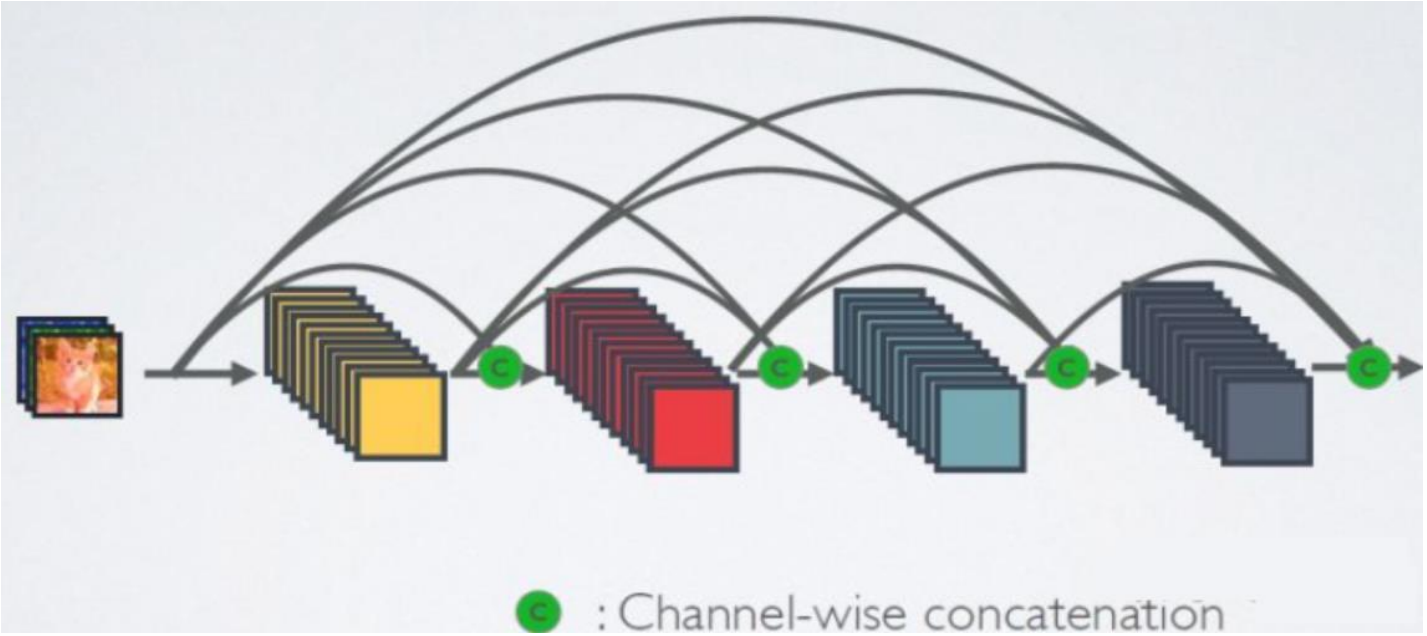


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

对比ResNet网络





DenseNet网络实现

- 普通网络：

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1})$$

- ResNet:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}$$

- DenseNet:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$$

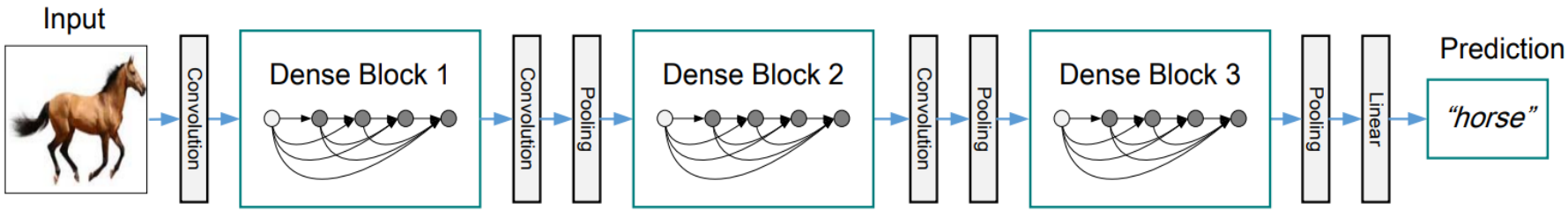
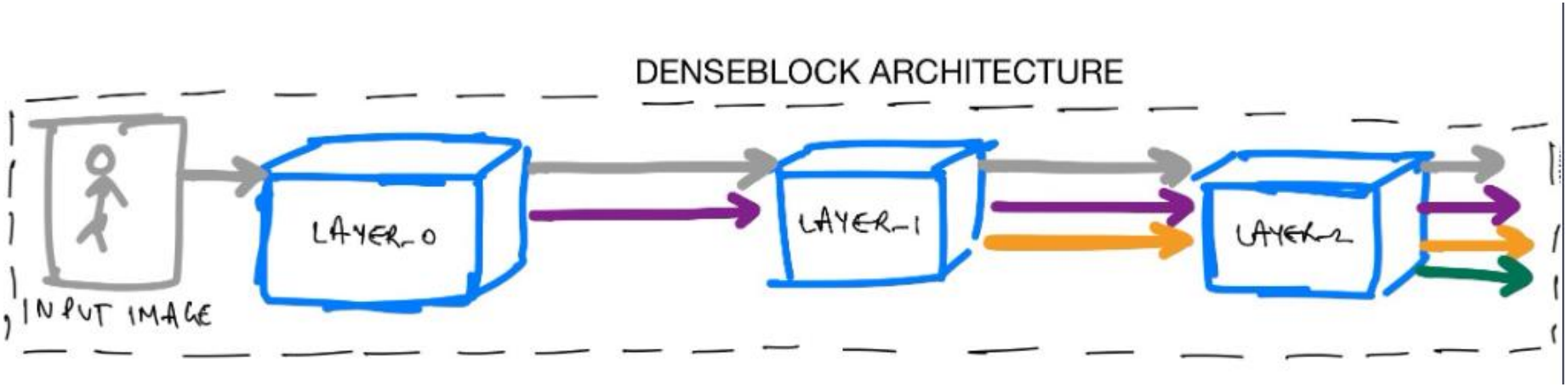
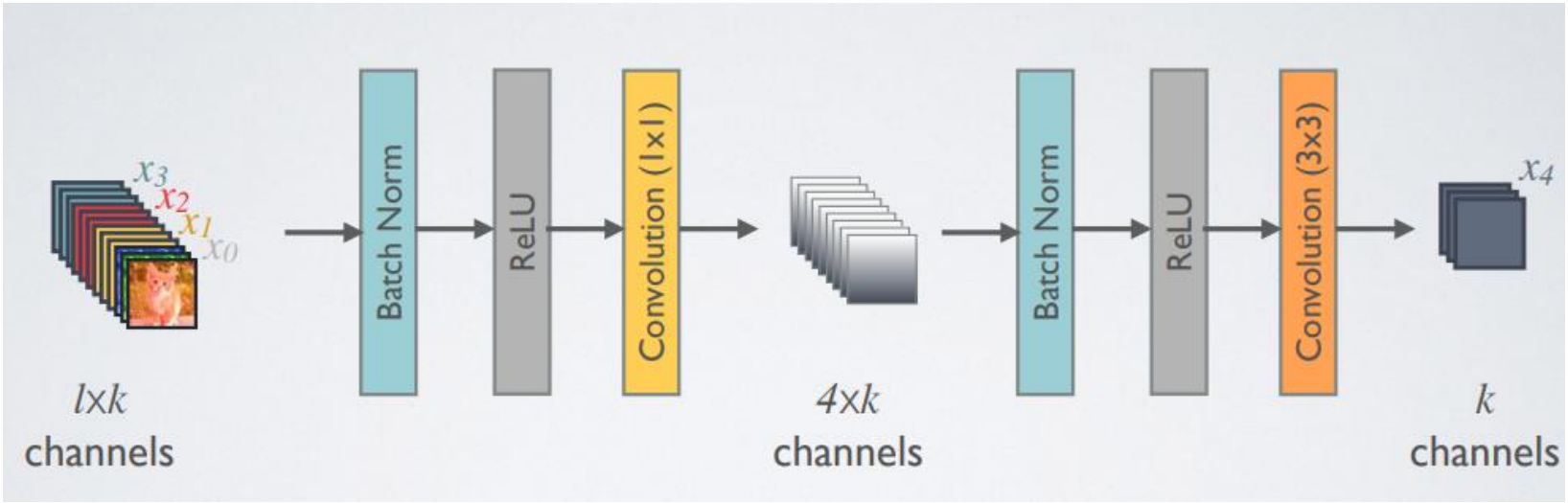


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

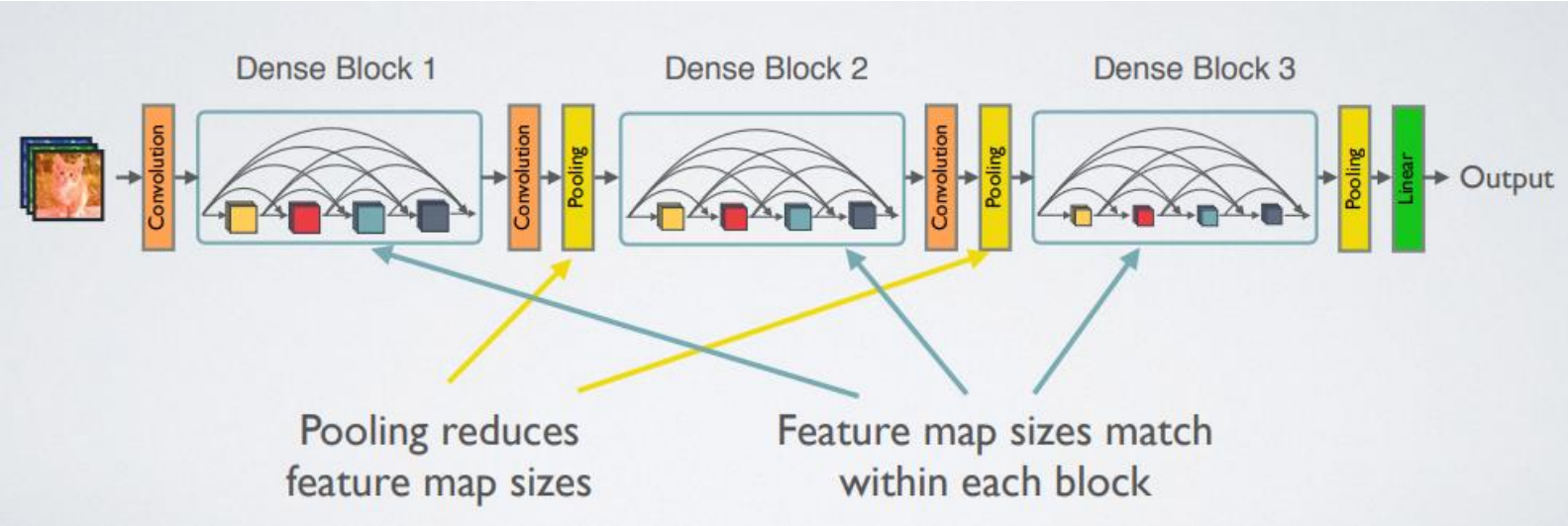
DenseBlock实现

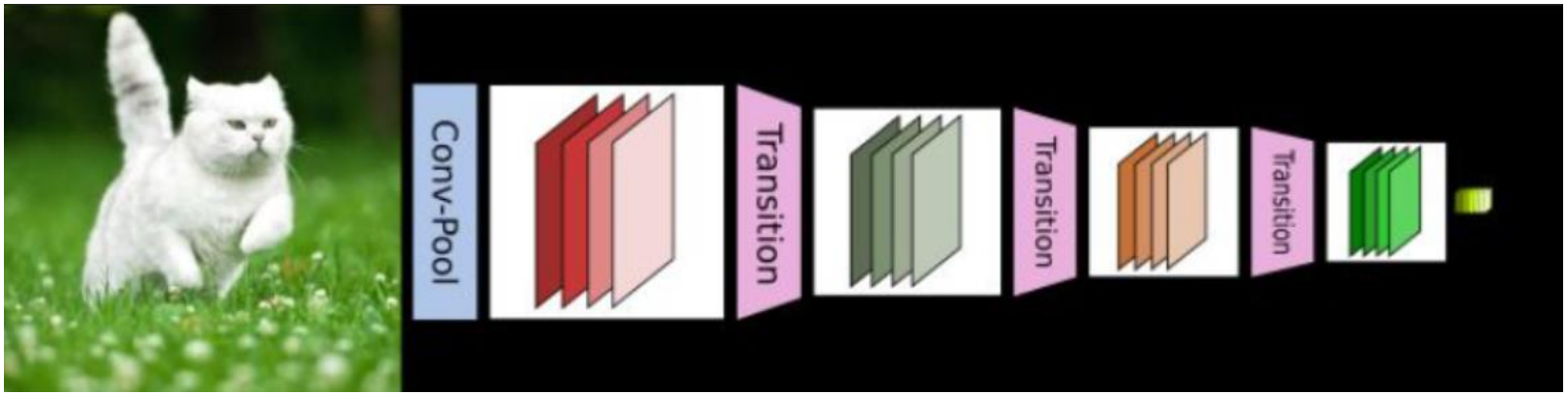
$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$$





DenseNet网络结构





DenseLayer实现

```
class _DenseLayer(nn.Sequential):
    """Basic unit of DenseBlock (using bottleneck layer) """
    def __init__(self, num_input_features, growth_rate, bn_size, drop_rate):
        super(_DenseLayer, self).__init__()
        self.add_module("norm1", nn.BatchNorm2d(num_input_features))
        self.add_module("relu1", nn.ReLU(inplace=True))
        self.add_module("conv1", nn.Conv2d(num_input_features, bn_size*growth_rate,
                                           kernel_size=1, stride=1, bias=False))
        self.add_module("norm2", nn.BatchNorm2d(bn_size*growth_rate))
        self.add_module("relu2", nn.ReLU(inplace=True))
        self.add_module("conv2", nn.Conv2d(bn_size*growth_rate, growth_rate,
                                           kernel_size=3, stride=1, padding=1, bias=False))
        self.drop_rate = drop_rate

    def forward(self, x):
        new_features = super(_DenseLayer, self).forward(x)
        if self.drop_rate > 0:
            new_features = F.dropout(new_features, p=self.drop_rate, training=self.training)
        return torch.cat([x, new_features], 1)
```

DenseBlock和Transition实现

- DenseBlock实现

```
class _DenseBlock(nn.Sequential):
    """DenseBlock"""
    def __init__(self, num_layers, num_input_features, bn_size, growth_rate, drop_rate):
        super(_DenseBlock, self).__init__()
        for i in range(num_layers):
            layer = _DenseLayer(num_input_features+i*growth_rate, growth_rate, bn_size,
                                drop_rate)
            self.add_module("denselayer%d" % (i+1,), layer)
```

- Transition Layer实现

```
class _Transition(nn.Sequential):
    """Transition layer between two adjacent DenseBlock"""
    def __init__(self, num_input_feature, num_output_features):
        super(_Transition, self).__init__()
        self.add_module("norm", nn.BatchNorm2d(num_input_feature))
        self.add_module("relu", nn.ReLU(inplace=True))
        self.add_module("conv", nn.Conv2d(num_input_feature, num_output_features,
                                          kernel_size=1, stride=1, bias=False))
        self.add_module("pool", nn.AvgPool2d(2, stride=2))
```

实验结果

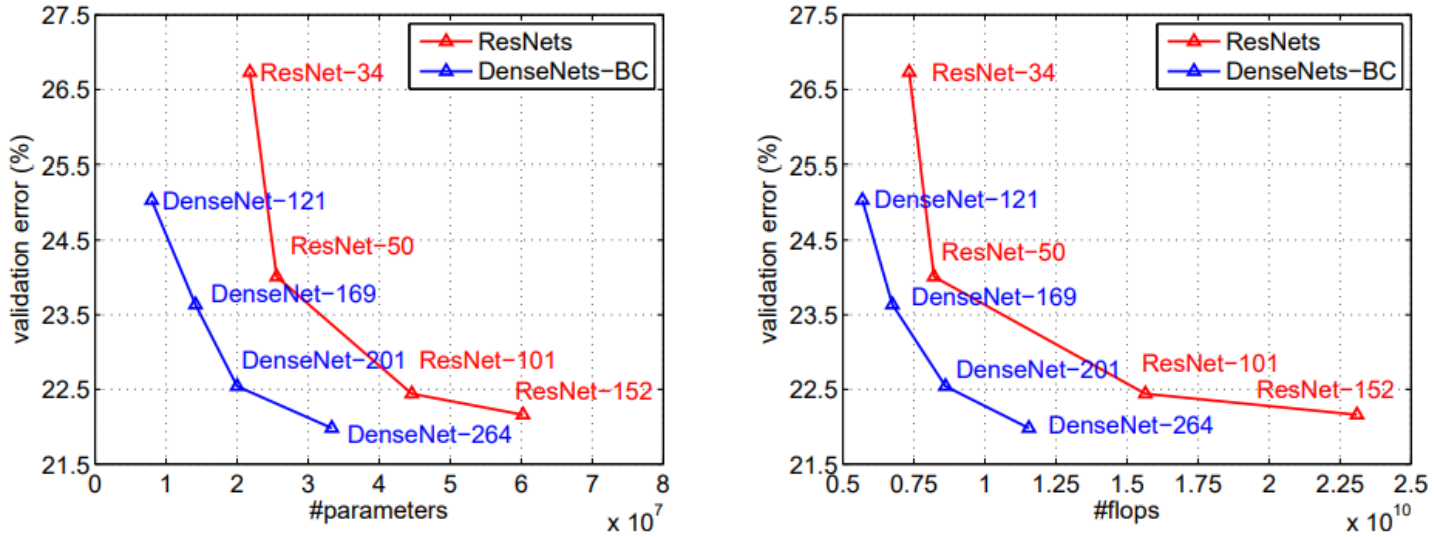
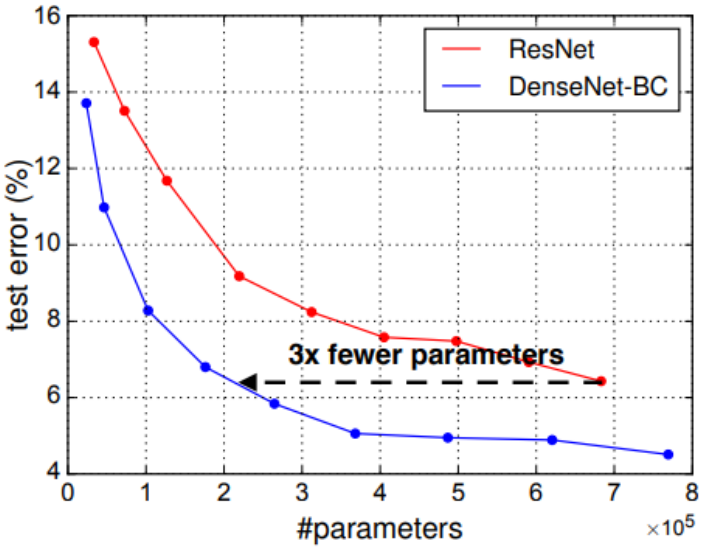
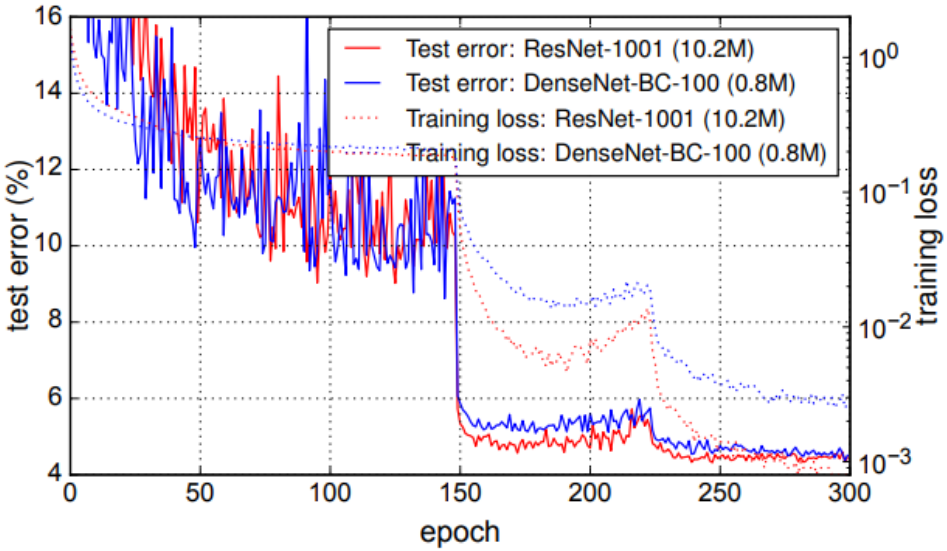


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

实验结果


ResNet
DenseNet-BC
3x fewer parameters
test error (%)
#parameters
×10^5


Test error: ResNet-1001 (10.2M)
Test error: DenseNet-BC-100 (0.8M)
Training loss: ResNet-1001 (10.2M)
Training loss: DenseNet-BC-100 (0.8M)
test error (%)
training loss
epoch

总结

- Highway, ResNet和DenseNet主要解决深度网络学习中的梯度消失和模型退化问题。
- Highway, ResNet采用**恒等映射**的思想，DenseNet采用**多尺度学习**的思想。
- **恒等映射**的思路也被用于Transformer Block中。