Deep Residual Learning for Image Recognition

—

Wiktor Jeziorski

Głębokie Sieci

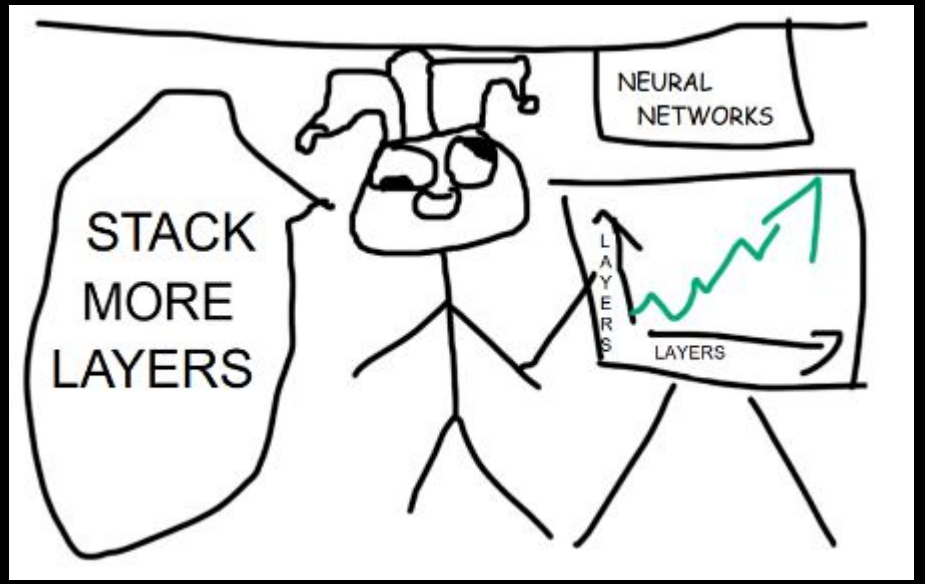
- uczą się cech hierarchicznych

- liczb “poziomów” cech jest tym większa im więcej warstw

- głębokość sieci jest kluczowa dla osiągnięcia najlepszych wyników (np. ImageNet sieci mające od 16 do 30 warstw)

hipoteza

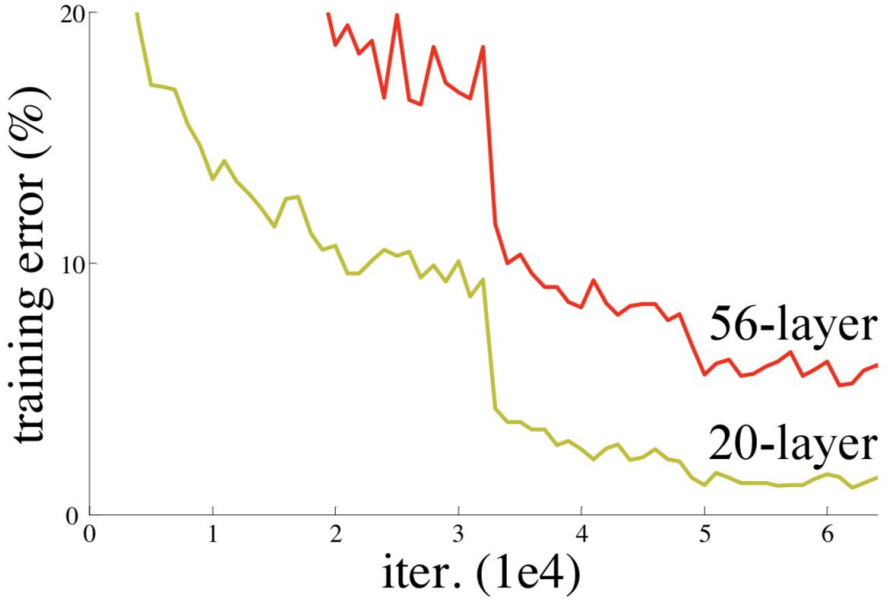
więcej warstw = lepszy model ?



Problem degradacji

- zwiększamy liczbę warstw
- “wysycenie” accuracy
- szybka degradacja
- nie jest to overfitting
- większy błąd TRENINGOWY
- głębsze sieci są trudniejsze do optymalizowania

Problem degradacji



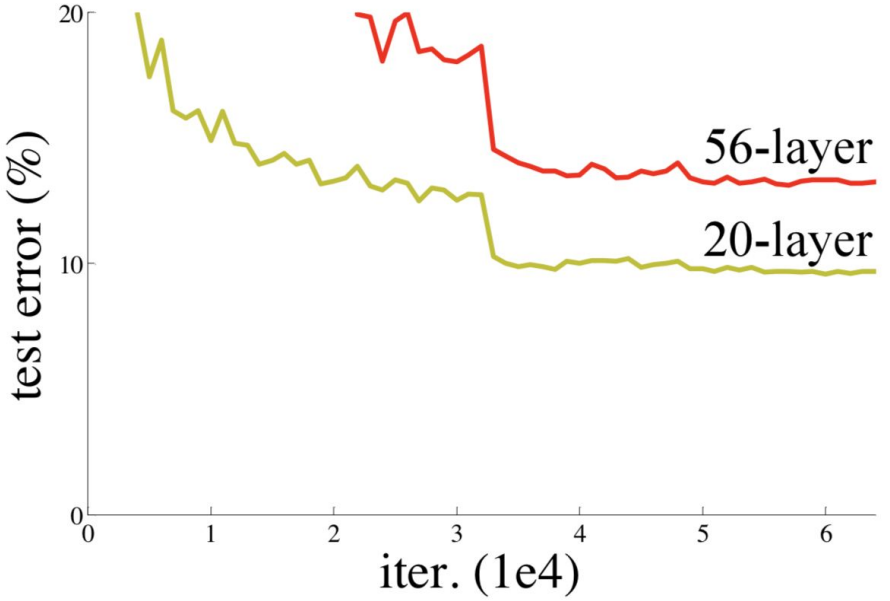
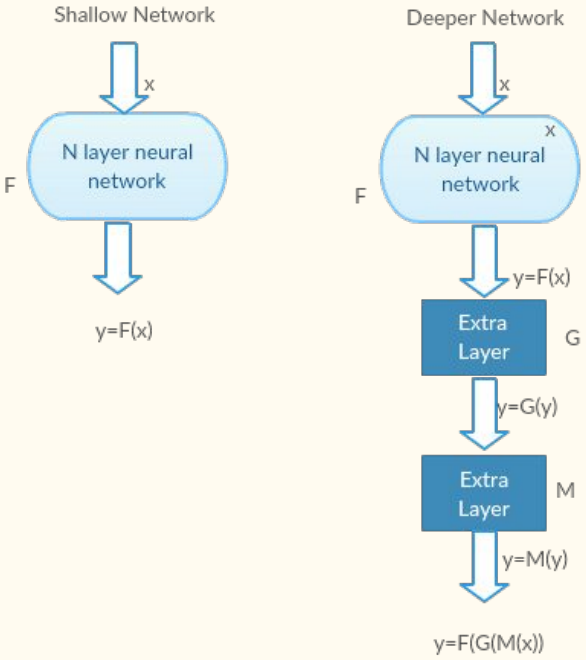


Figure 1. Training error (left) and test error (right) on CIFAR-10

“Konstrukcja” głębszego modelu



G and M act as Identity Functions. Both the Networks Give same output

Propagowanie identyczności jest trudne!

Residual Learning

- rozważamy kilka kolejnych warstw sieci
- **x** - wejście dla pierwszej z warstw
- **H(x)** - funkcja, którą mają realizować te warstwy

przeformułowanie problemu jako funkcja residualna:

- **F(x) = H(x) - x**
- wtedy: **H(x) = F(x) + x**

Residual Learning

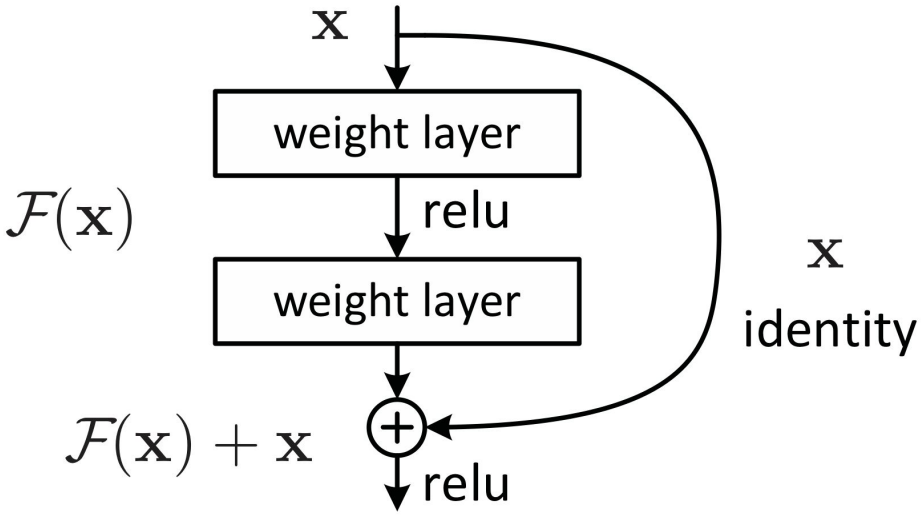


Figure 2. Residual learning: a building block.

Residual Learning/Skip Connections - zalety

- dużo łatwiejsza nauka identyczności i odwzorowań bliskich identyczności
- brak dodatkowych parametrów

- dużo łatwiejsza optymalizacja bardzo głębokich sieci
- zwiększanie ilości warstw zwiększa accuracy

Residual Learning

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

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

$$\sigma(\mathbf{y})$$

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

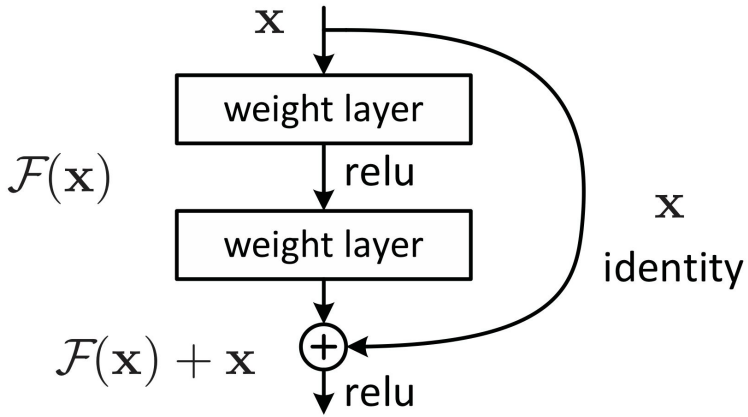


Figure 2. Residual learning: a building block.

Residual Learning

- postać funkcji residualnej F jest płynna
- w eksperymentach w pracy F ma postać 2-3 warstw
- można próbować więcej
- F składające się z 1 warstwy jest niezalecane

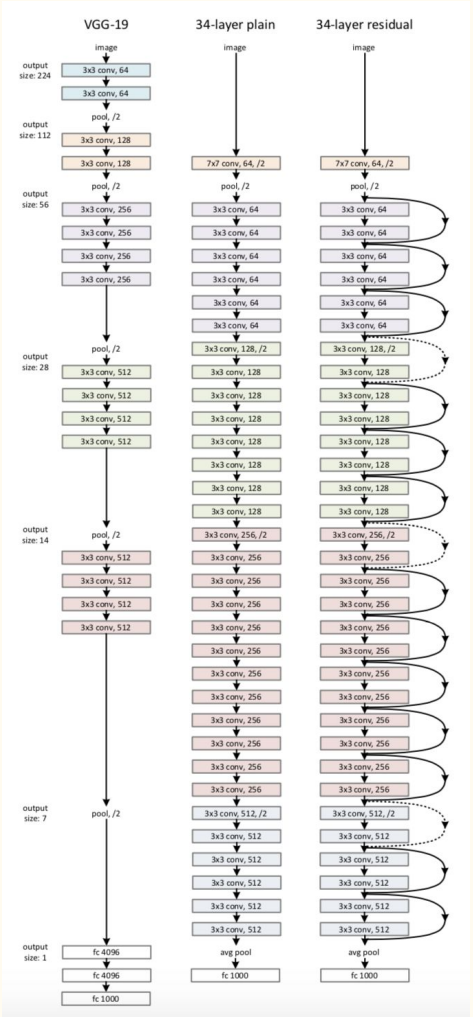
$$\mathbf{y} = W_1\mathbf{x} + \mathbf{x}$$

postać równania podobna do warstwy liniowej, autorzy nie zauważyli żadnych korzyści z tej postaci

Architektura

na obrazku:

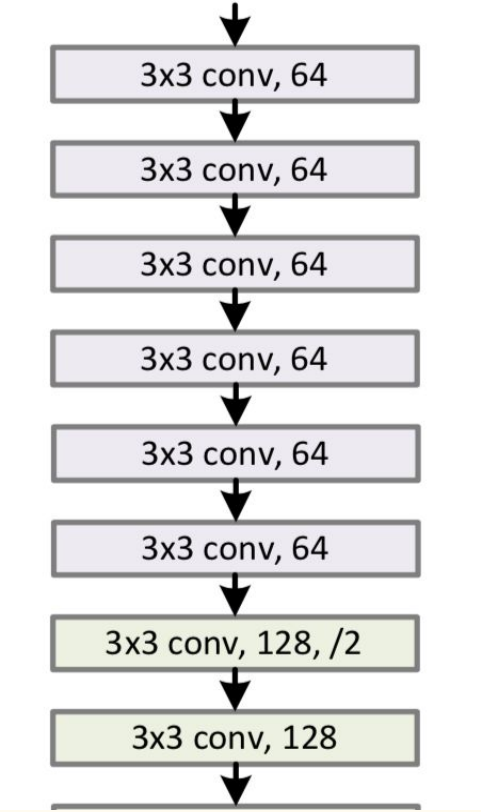
- VGG-19
- 34-warstwowa “zwykła” sieć
- 34-warstwowa sieć residualna



Architektura Sieci “zwykłej”

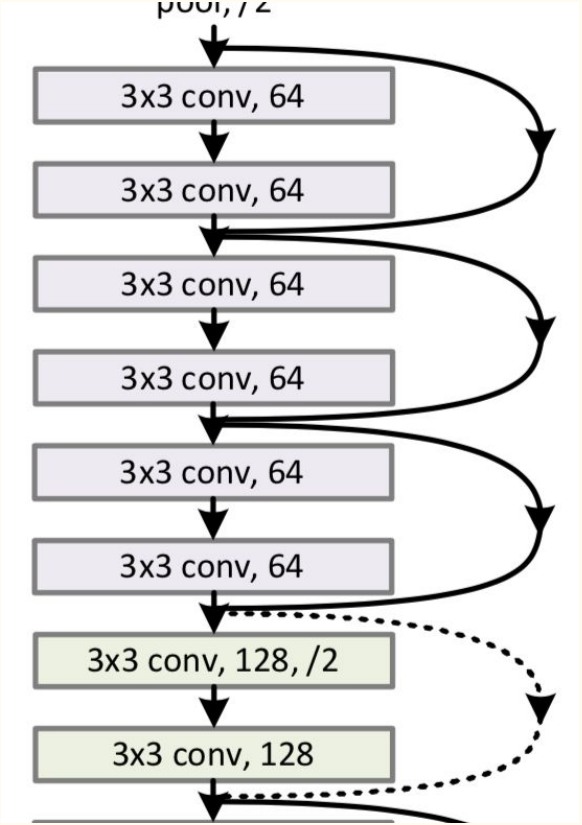
inspirowana przez VGG

- warstwy z tymi samymi wymiarami mają taką samą liczbę filtrów
- gdy wymiary są zmniejszane o połowę podwajamy liczbę filtrów
- downsampling realizowany poprzez konwolucje ze stridem 2



Architektura Sieci Residualnej

- oparta na sieci “zwykłej”
- dodano skip-connections
- padding zerami dla skip-connections zwiększających wymiar



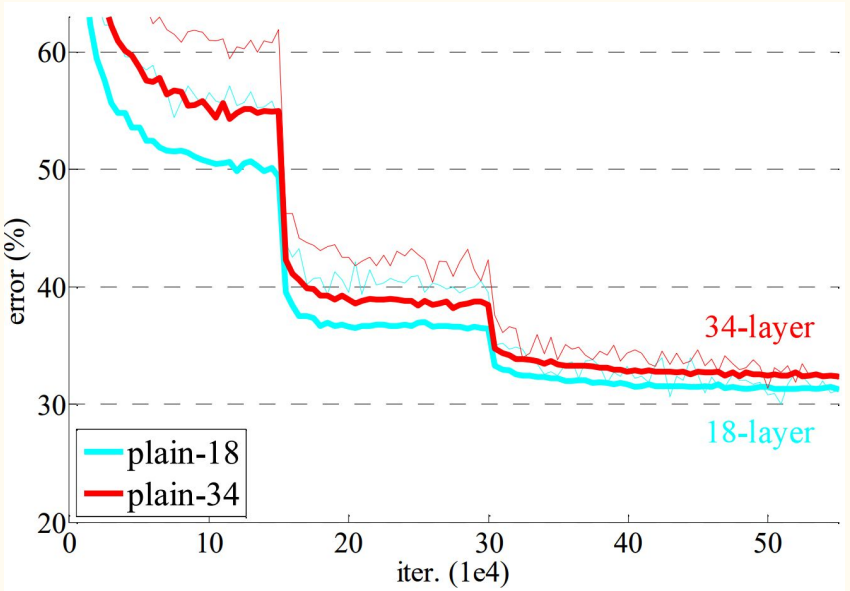
Implementacja

Zastosowano standardowy preprocessing obrazków jak dla VGG i AlexNet

- Batch normalization
- learning rate 0.1
- 60e4 iteracji
- Weight decay 0.0001
- Momentum 0.9

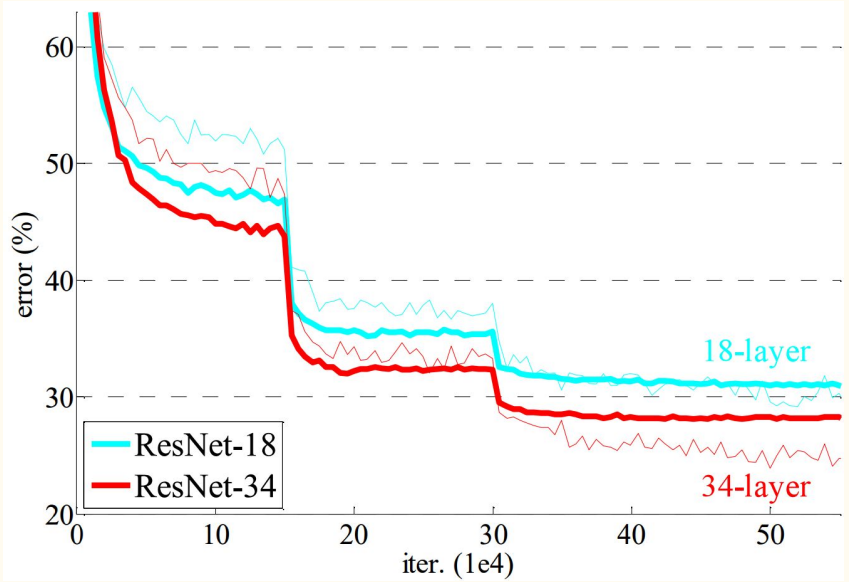
ImageNet - sieć “zwykła”

34-warstwowa sieć ma większy błąd poprzez cały trening

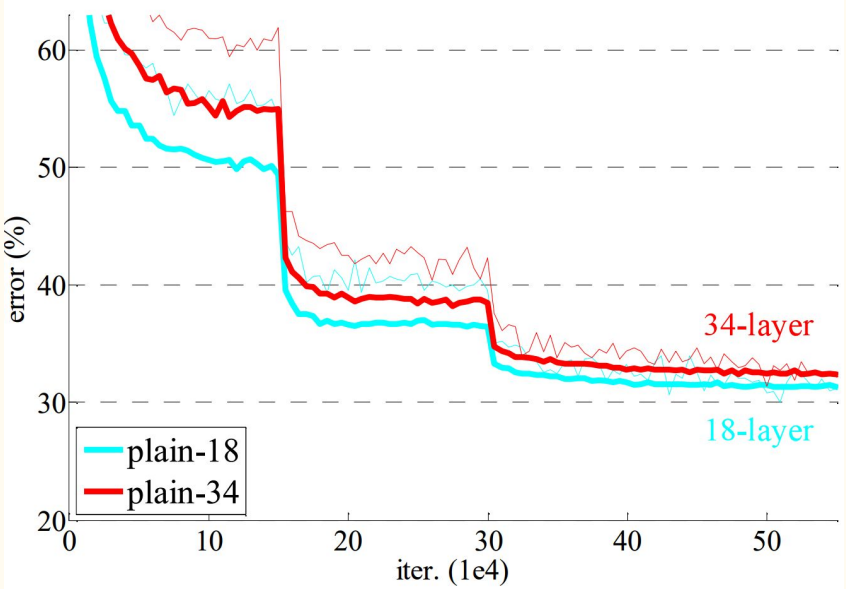


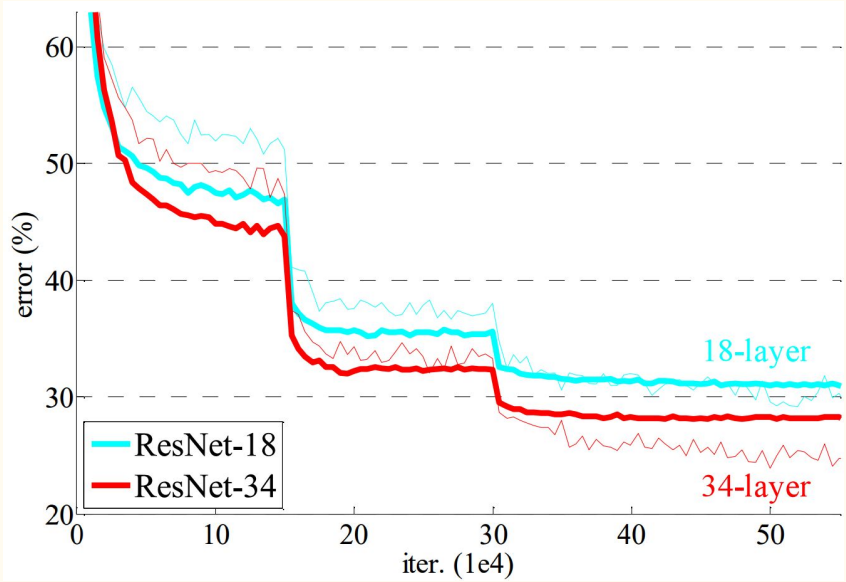
ImageNet - sieć residualna

34-warstwowa sieć po dodaniu skip-connections ma mniejszy błąd



ImageNet - porównanie





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

dla 18-warstwowej podobne wyniki, ale ResNet-18 zbiega szybciej

Identity vs Projection Shortcuts

A - padding zerami, gdzie potrzeba

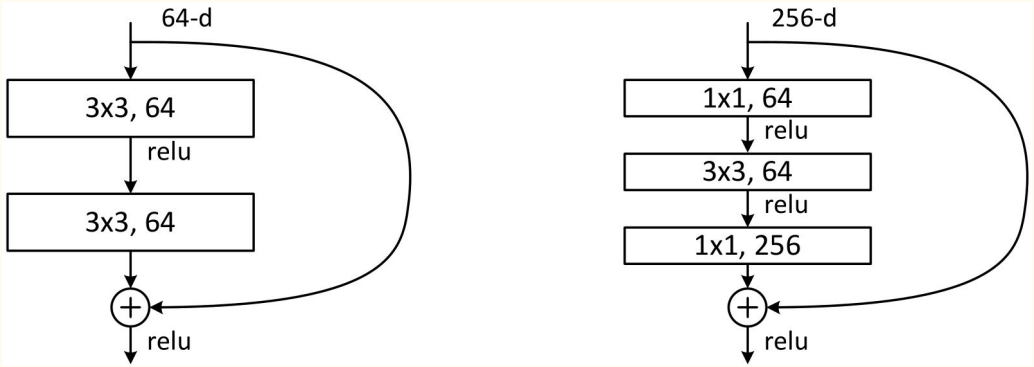
B - projekcje, gdzie potrzeba

C - projekcje wszędzie

plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40

opcja C daje nieznaczną różnicę

Bottleneck Architecture



- konwolucje 1x1 używane dla zredukowania, a potem zwiększenia liczby kanałów

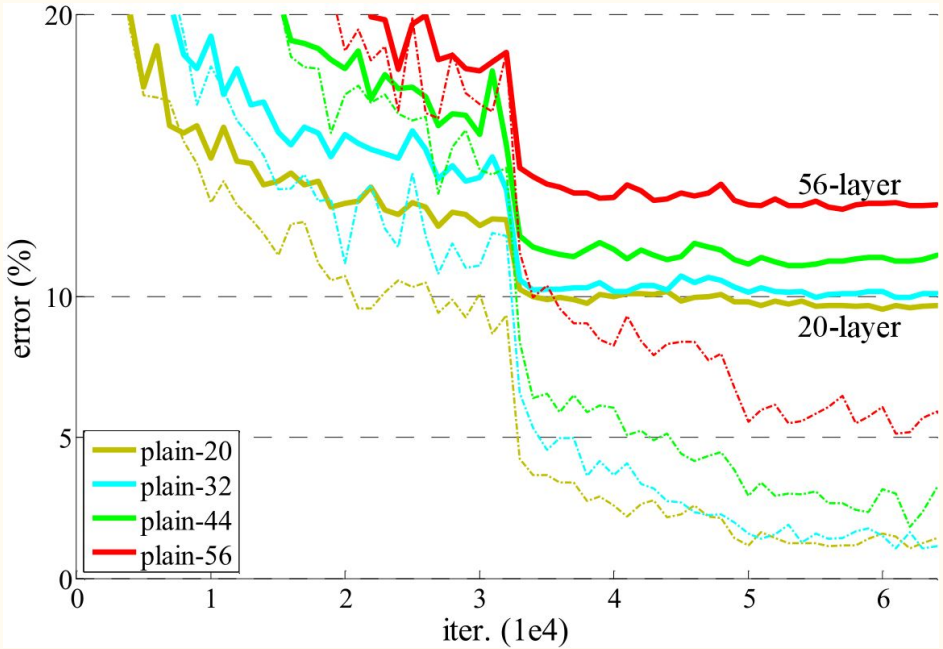
Bottleneck Architecture

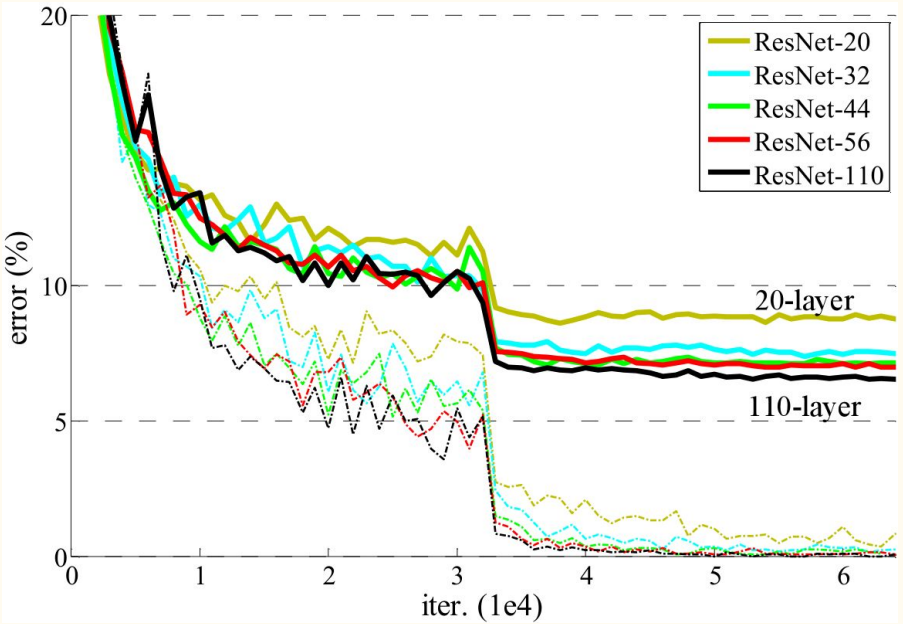
layer name	output size	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stride 2		
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax		
FLOPs		3.8×10^9	7.6×10^9	11.3×10^9

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43[†]
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

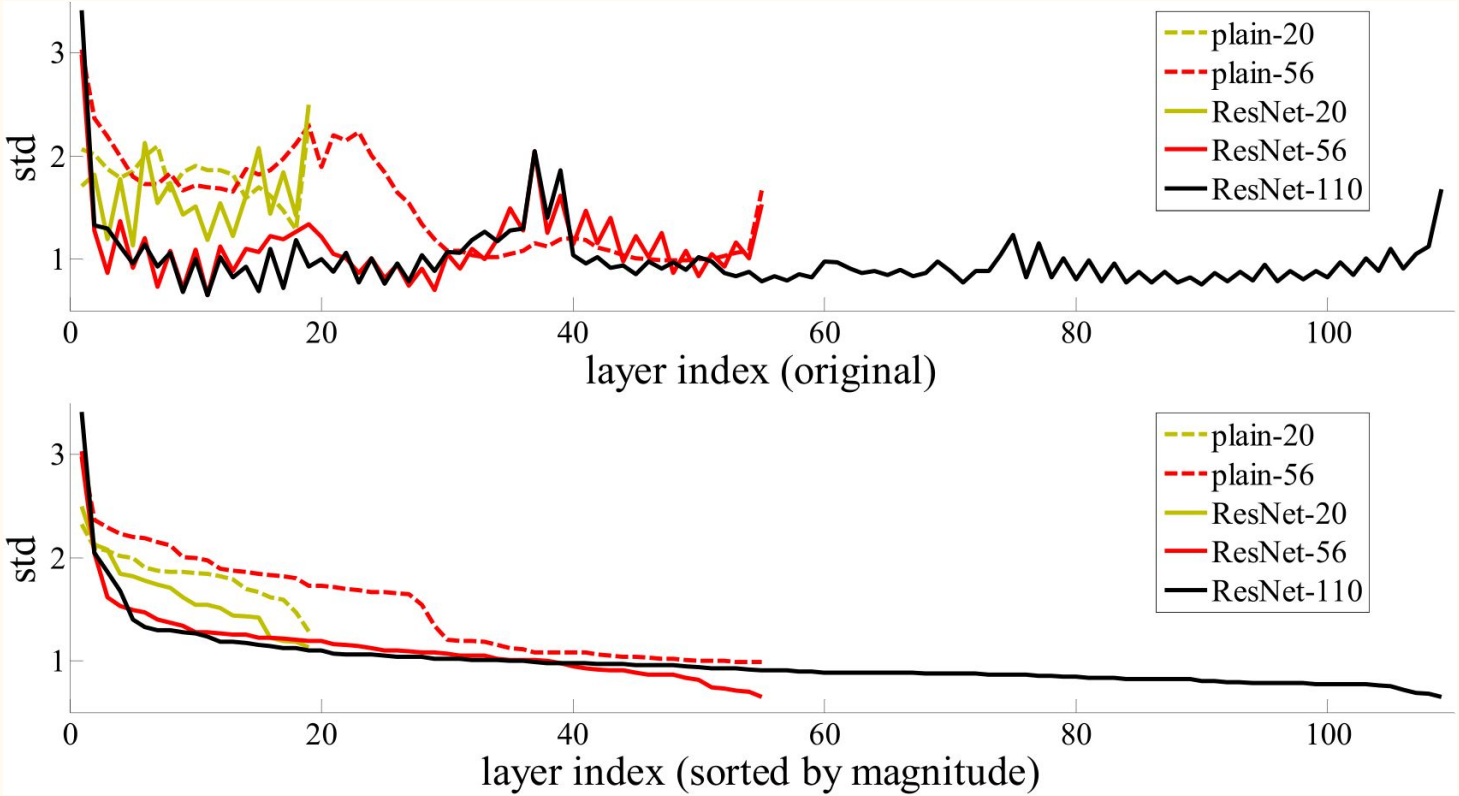
ResNet-152 nadal mniej skomplikowany niż VGG

CIFAR-10

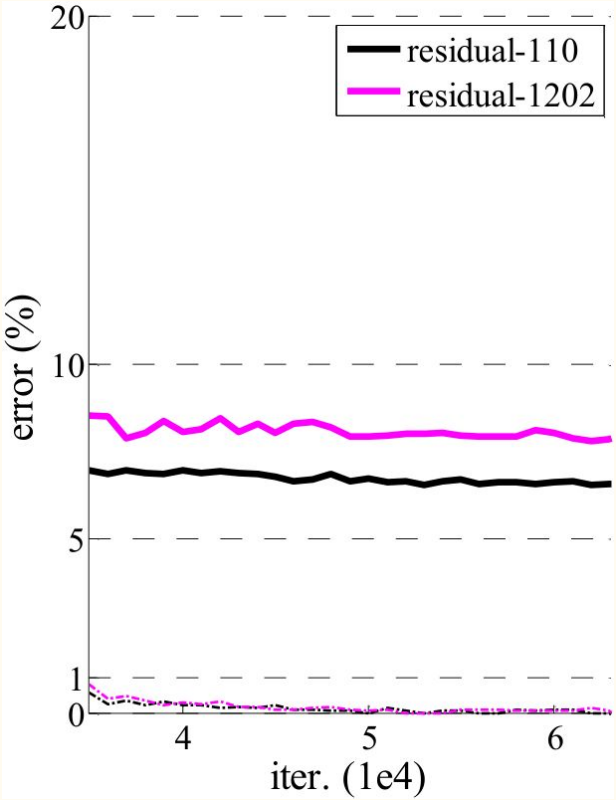

20
10
5
0
error (%)
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer
plain-20
plain-32
plain-44
plain-56


20
10
5
0
error (%)
0
1
2
3
4
5
6
iter. (1e4)
ResNet-20
ResNet-32
ResNet-44
ResNet-56
ResNet-110
20-layer
110-layer

CIFAR-10


plain-20
plain-56
ResNet-20
ResNet-56
ResNet-110
std
layer index (original)
plain-20
plain-56
ResNet-20
ResNet-56
ResNet-110
std
layer index (sorted by magnitude)

CIFAR-10 - 1202 warstwy



Nie ma problemów z optymalizacją,
ale overfituje

	# layers	# params	
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Dziękuję za uwagę