

Podstawy Sztucznej Inteligencji (PSI)

Lab 3 - omówienie

Pytania? Wątpliwości? Uwagi?

Częste kwestie

Nie można zapominać o `loss.backward()`!

- liczy gradienty
- bez tego $\text{gradient} = 0$ -> sieć się nie uczy

Używanie modułów-komponentów

```
self.mlp = nn.Sequential(...)
```

- można wywołać przez:

```
self.mlp(X)
```

- nie trzeba podawać jawnie `.forward()`

Konwolucyjne sieci neuronowe

Computer vision

Classification



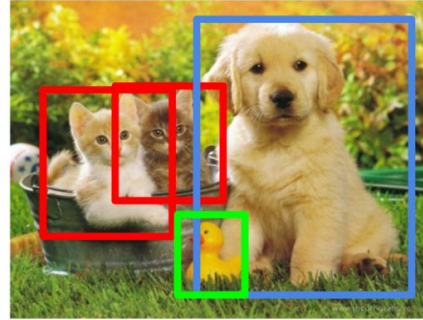
CAT

**Classification
+ Localization**



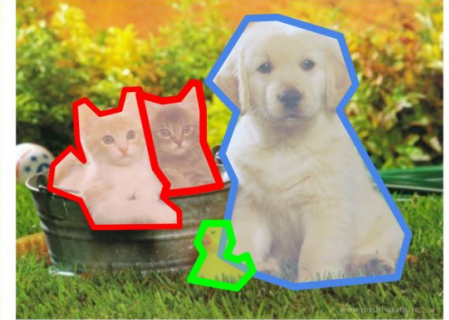
CAT

Object Detection



CAT, DOG, DUCK

**Instance
Segmentation**

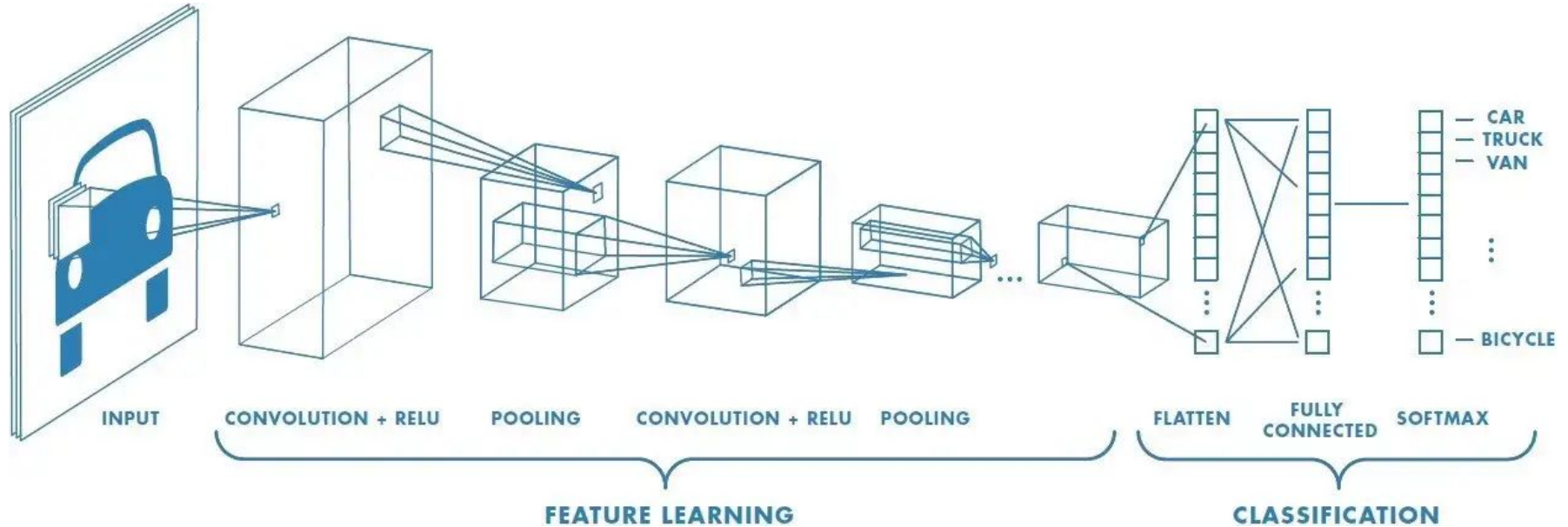


CAT, DOG, DUCK

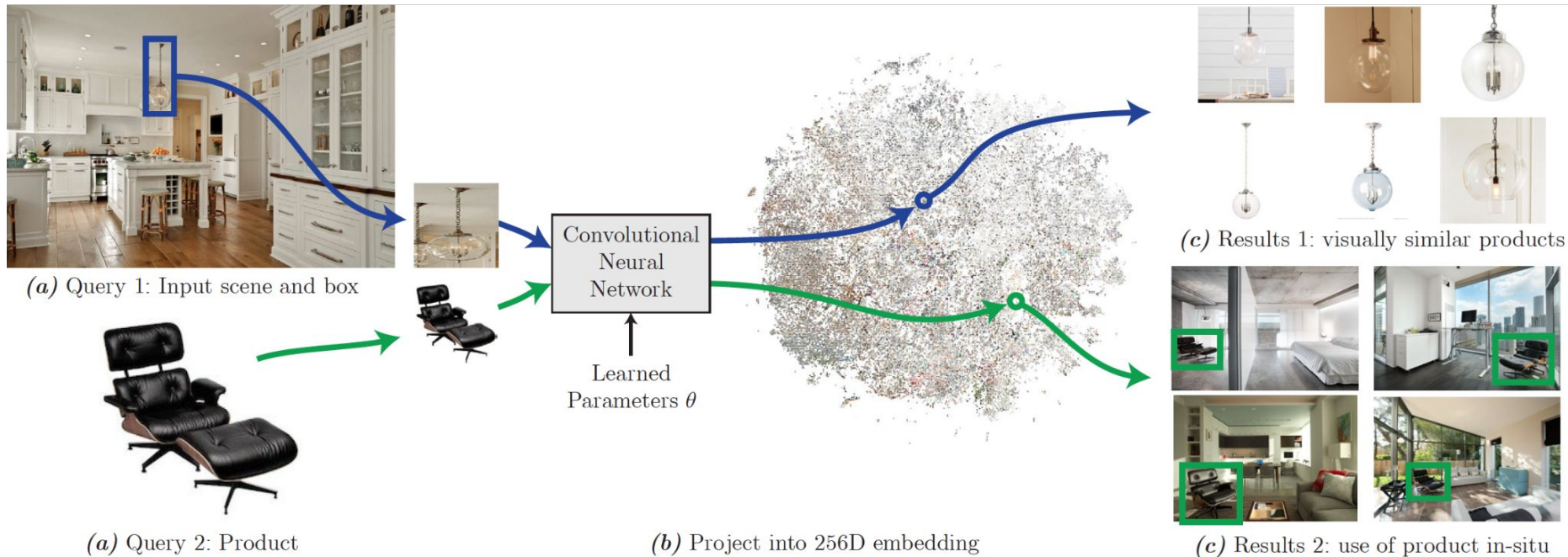
Single object

Multiple objects

Klasyfikacja obrazów



Wyszukiwanie wizualne



Czego chcemy w computer vision?

- **automatyczne wykrywanie cech** - model powinien sam nauczyć się, jakie obiekty na obrazach są przydatne do danego zadania
- **rozsądny koszt obliczeniowy** - dla MLP nie ma szans (np. mały obrazek RGB to $3 * 256 * 256 = \sim 200k$ wejść, 1000 wymiarów na wyjściu = ~ 200 miliardów wag)
- **translational equivariance** - chcemy, żeby cecha na obrazie (np. krawędzie, kolory, proste kształty) była wykrywana w każdym miejscu tak samo
- **translational invariance** - chcemy, żeby niewielkie wahania pikseli w różne strony nie zmieniały reakcji modelu
- **lokalność** - model powinien uwzględniać, że na obrazie rzeczy blisko siebie są ważniejsze (**spatial locality**)

Narzędzie nr 1: konwolucja

- **uwaga:** każdy mówi "konwolucja", ale tak naprawdę CNNy robią **korelację skrośną (cross-correlation)**; różnią się tylko znakiem, więc w sumie bez różnicy
- **konwolucja (convolution)** to operacja, w której sprawdzamy, jak bardzo **wejście (input)** jest interesujące dla **filtra (filter / kernel)** w poszczególnych miejscach
- tzw. **sliding dot product** - przesuwamy, iloczyn element-wise i sumujemy (plus bias)

Input		Kernel		Output																	
<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	*	<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	=	<table border="1"><tr><td>19</td><td>25</td></tr><tr><td>37</td><td>43</td></tr></table>	19	25	37	43
0	1	2																			
3	4	5																			
6	7	8																			
0	1																				
2	3																				
19	25																				
37	43																				

Konwolucja - przykład

1	0	1
0	1	0
1	0	1

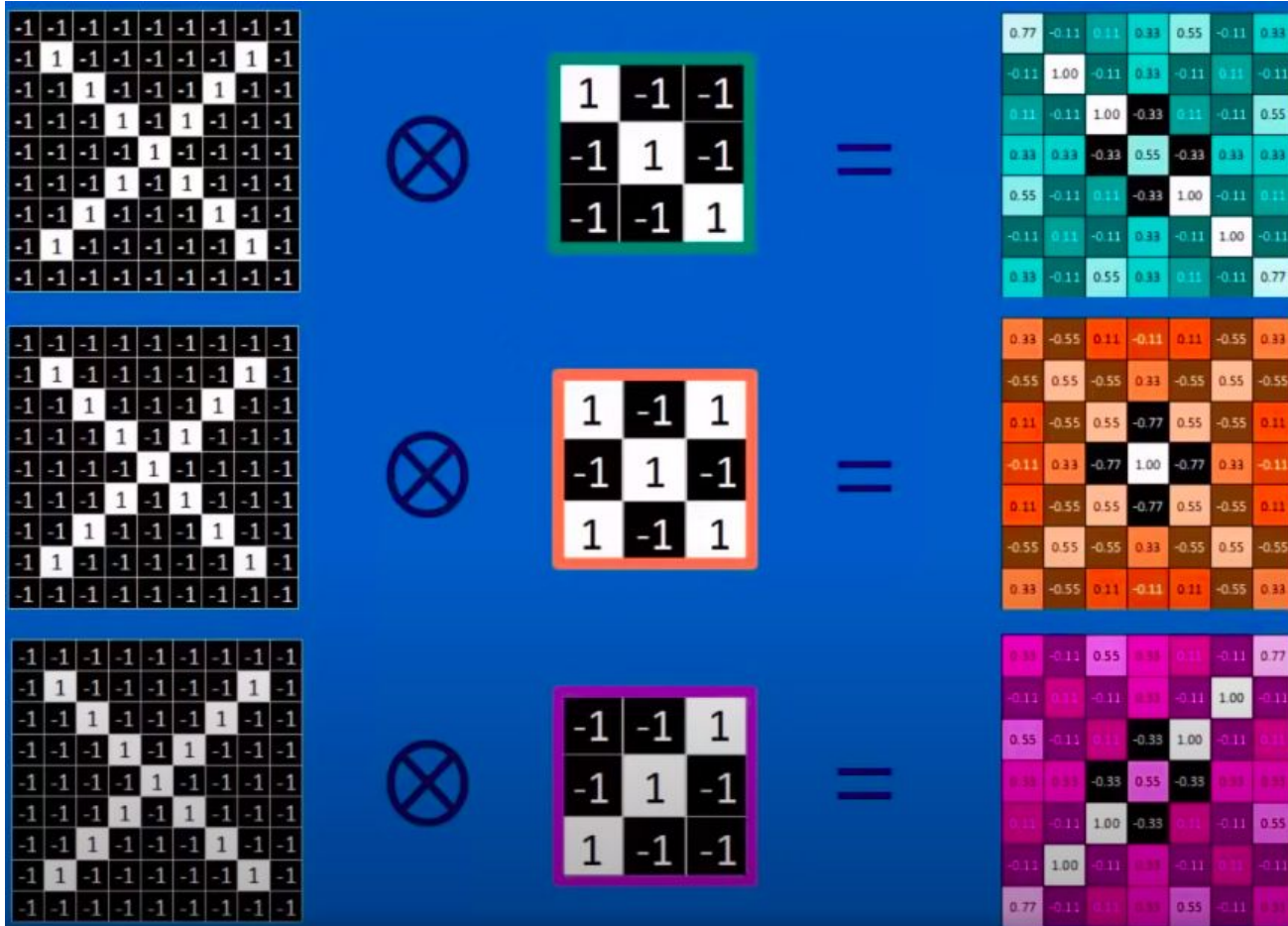
1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

Konwolucja - wiele różnych filtrów



Konwolucja - zalety

- **automatyczne wykrywanie cech** - tę operację można różniczkować, a więc można optymalizować spadkiem wzdłuż gradientu, więc wag filtrów można się uczyć
- **translational equivariance** - działa tak samo w każdym miejscu obrazu
- **lokalność** - obejmujemy taki fragment, jak duży jest filtr

Input image



Convolution
Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



Warstwa konwolucyjna

- w sieciach neuronowych **warstwa konwolucyjna (convolutional layer)** składa się z wielu filtrów
- filtry często nazywa się **kanalami (channels)** albo **reception fields**
- inicjalizujemy różnymi małymi liczbami losowymi, a potem uczymy - filtry nauczą się **wyciągać (extract)** różne cechy z obrazu
- wyjście nazywa się często **mapą cech (feature map)**, bo duże liczby zawierają miejsca, gdzie filtr wykrył swoją cechę
- po niej następuje element-wise **aktywacja**, jak w MLP

Konwolucja + ReLU

Feature Map

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	1

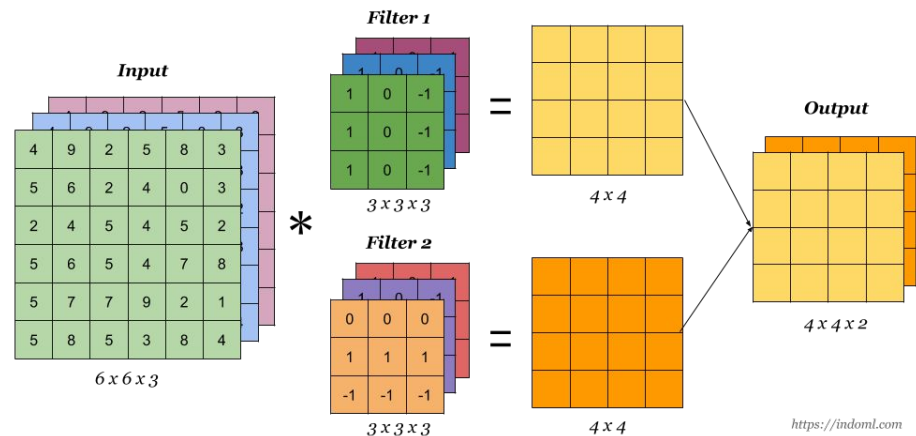
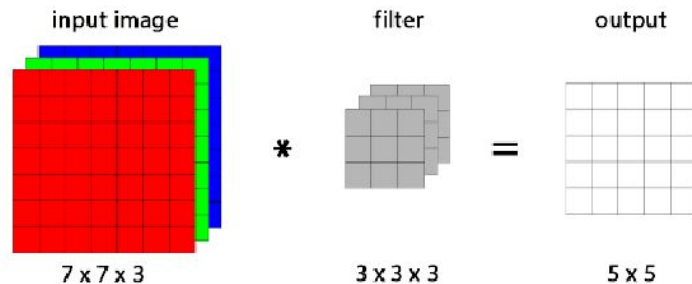


Feature Map, Post ReLU

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

Filtry są wielowymiarowe

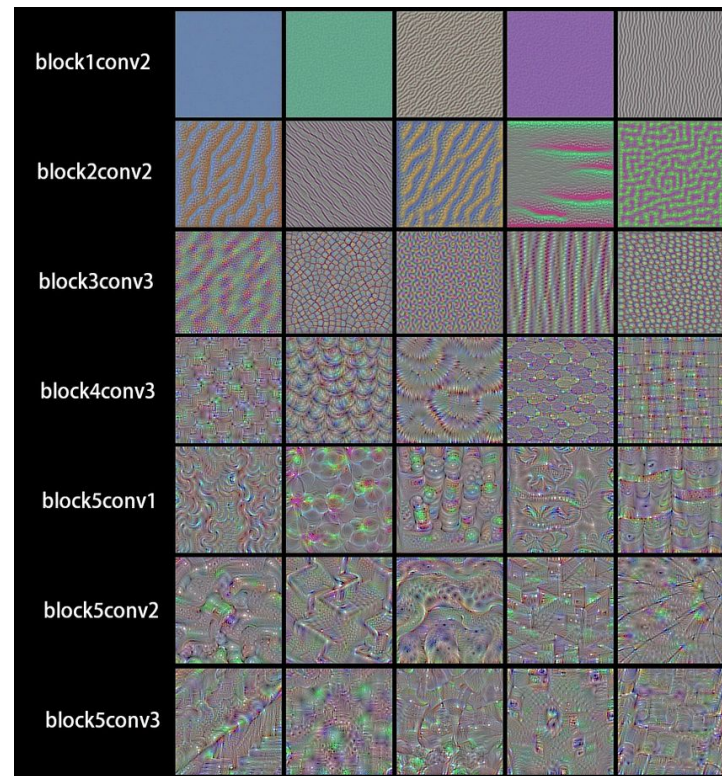
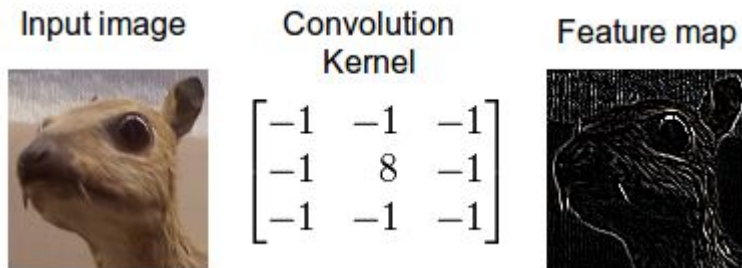
- pojedynczy filtr jest **tensorem 3D**
- ma **tylko kanałów** (głębokość), ile jest kanałów w **warstwie wejściowej**
- w pierwszej warstwie filtry mają więc rozmiar **$F \times F \times 3$** (bo RGB)
- zasada działania bez zmian - iloczyn skalarny (mnożenie point-wise + suma), tylko że w 3D
- liczba filtrów = liczba kanałów na wyjściu**



<https://indoml.com>

Co to robi?

- **pierwsze warstwy**, blisko obrazu, wyciągają **proste cechy** - kolory, krawędzie pionowe i poziome itp.
- **dalsze warstwy** dostają na wejście feature mapy z poprzednich konwolucji - mogą ekstrahować bardziej **złożone cechy**



VGG - filtry w różnych warstwach

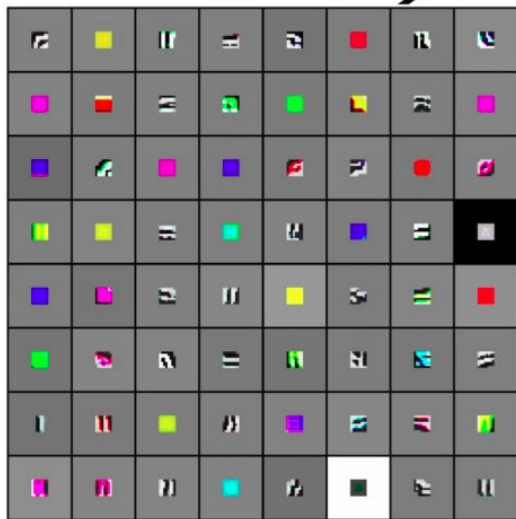


Low-level
features

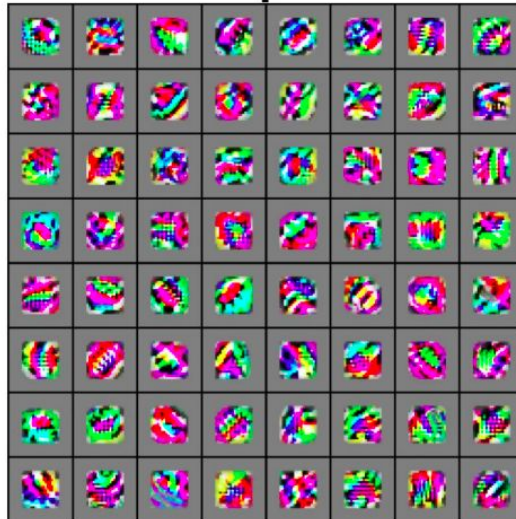
Mid-level
features

High-level
features

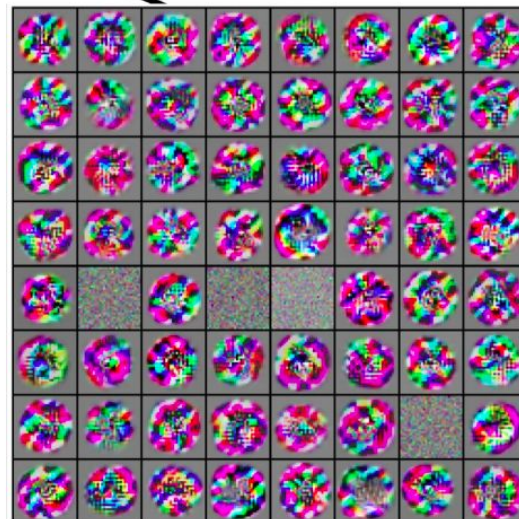
Linearly
separable
classifier



VGG-16 Conv1_1



VGG-16 Conv3_2



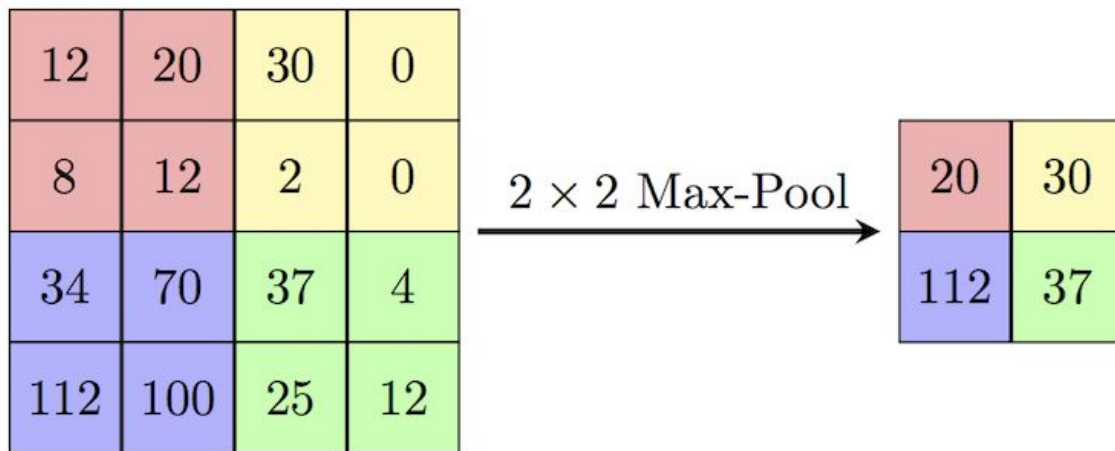
VGG-16 Conv5_3

Warstwa konwolucyjna - hiperparametry

- **rozmiar filtra K** - jak duży obszar obejmujemy, ma bezpośredni wpływ na wielkość cech i koszt obliczeniowy
- **liczba filtrów C** - jak dużo kanałów rozważamy
- **krok (stride) S** - o jak dużo przesuwamy się w prawo i w dół; często 1, zwykle mniejszy od filtra
- **wypełnienie (padding) P:**
 - jak nie wypełnimy krawędzi obrazu, to nie możemy wyjść poza niego -> zmniejszy się
 - często ok - ważne cechy są rzadko blisko krawędzi
 - można wypełnić dookoła np. zerami, albo średnią wartością sąsiednich pikseli

Warstwa zbierająca (pooling)

- **pooling** agreguje informacje z sąsiednich pikseli
- zwykle **max pooling** - bierzemy maksymalną wartość po aktywacji z sąsiednich pikseli
- zapewnia **translation invariance**, bo wiemy tylko, że mniej więcej w danym obszarze coś wystąpiło, nie w danym pikselu
- prawie zawsze bez overlapu (kernel size K = stride S), z bardzo małym filtrem (2-3) - zmniejsza obraz



Współdzielenie wag

- jeden filtr działa tak samo na całym obrazie - mamy **współdzielenie wag (weight sharing)**
- zapewnia to też **translation equivariance**, bo filtr o tych samych wagach jest używany w różnych miejscach
- dzięki temu CNN ma też **rozsądny koszt obliczeniowy**

CNN - koszt obliczeniowy

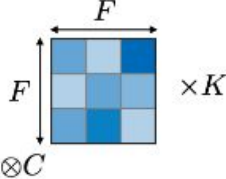
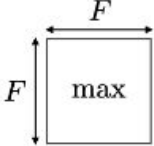
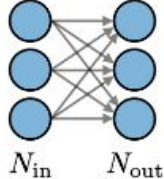
Obraz: $3 \times 256 \times 256$

Liczba parametrów:

MLP: ~200 miliardów

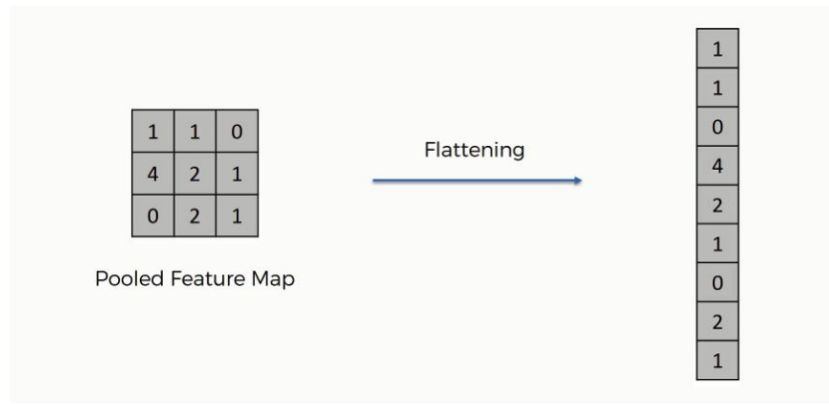
CNN: 64 filtry 3×3

$(64 * 64 * 3 + 1) * 64 =$
~800k

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	<u>$(F \times F \times C + 1) \cdot K$</u>	0	<u>$(N_{in} + 1) \times N_{out}$</u>
Remarks	<ul style="list-style-type: none"> • One bias parameter per filter • In most cases, $S < F$ • A common choice for K is $2C$ 	<ul style="list-style-type: none"> • Pooling operation done channel-wise • In most cases, $S = F$ 	<ul style="list-style-type: none"> • Input is flattened • One bias parameter per neuron • The number of FC neurons is free of structural constraints

Warstwa spłaszczająca (flatten)

- CNN do ekstrakcji cech nazywa się często **backbone**, a MLP do klasyfikacji **głową (head)**
- wyjściem CNNa są feature maps, wejściem MLP wektory
- **warstwa spłaszczająca (flattening layer)** łączy wiersze po kolei w każdej feature mapie, a później konkatenuje to dla każdego kanału



Architektura prostych CNN do klasyfikacji obrazów

- **wejście:** batch N obrazów RGB, kształtu np. (N, C, H, W)
- **sekwencja** warstw konwolucyjnych i poolingowych
 - zwykle kilka (np. 2-3) konwolucje między poolingami
- **spłaszczenie** do wektora
- **MLP**
 - zwykle małe, góra 2 warstwy
 - bardzo często sama warstwa liniowa + softmax
 - drogie + zakładamy, że backbone sam w sobie da dobre cechy

Hiperparametry

Konwolucja:

- liczba warstw
- liczba filtrów
- rozmiar filtrów
- stride
- padding

Pooling:

- liczba warstw - co ile konwolucji robić?
- rozmiar filtra

W PyTorchu

- robi się zwykle 1 klasę, która w `__init__()` tworzy 2 atrybuty, np. `backbone` i `classifier`
- między nimi robi się `nn.Flatten()` (uwaga - funkcja `flatten()` działa inaczej!)
- **kształt tensorów to (N, C, H, W):**
 - N - liczba obrazów w batchu
 - C - liczba kanałów
 - H, W - wymiary obrazu

Transfer learning

Trening sieci do computer vision

- **computer vision** to wymagające zadanie, wymagające modeli o dużej pojemności
- duża pojemność = potrzeba dużego zbioru danych
- np. ImageNet (~1,3 miliona obrazów treningowych, 1000 klas)
- **problem:** przy większości problemów mamy małe zbiory danych
- **obserwacja:** większość zadań wymaga podobnych umiejętności (rozpoznawanie kształtów, kolorów itp.), więc może dałoby się wykorzystać wiedzę z jednego zadania w drugim?

Podejścia do transfer learningu

- **zamrożony backbone**

- zamrażamy (freeze) wagi CNNa i trenujemy tylko nowy head
- bardzo obniża koszt obliczeniowy - tak naprawdę to zwykła klasyfikacja tabelaryczna
- często działa dobrze, szczególnie gdy mamy względnie podobną dziedzinę do ImageNet

- **trening całości**

- normalnie trenujemy całą sieć, pretrening traktujemy jak inicjalizację wag
- kosztowne, ale pozwala lepiej przenosić się do znacząco nowych domen

- **pośrednio**

- większy learning rate na końcu sieci, mniejszy w głębszych warstwach, freeze na początku
- złoty środek, ale nieco trudniejsze implementacyjnie + dodatkowe hiperparametry

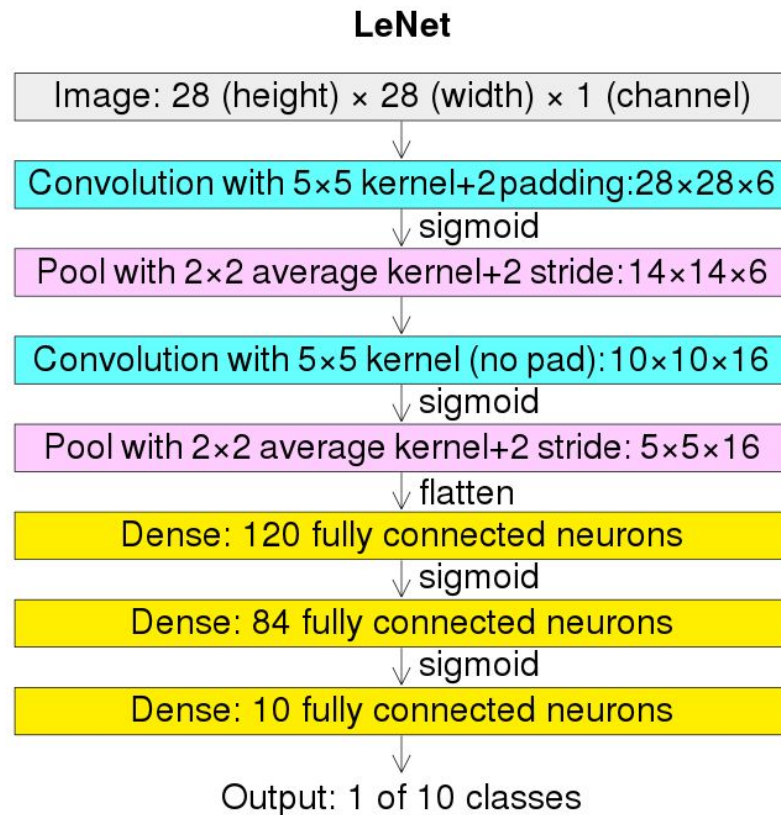
Typowy pipeline w computer vision

- bierzemy **pretrenowane wagi**, np. z Torchvision albo TIMM (Torch IMage Models)
- ucinamy głowę, robimy nową
- **dotrenowujemy** na naszym zbiorze
- ostrożny trening, bo łatwo przeuczyć:
 - mały learning rate, np. $1e-5$ - $1e-3$
 - mała liczba epok (kilka-kilkanaście)
 - freeze na cały backbone albo dużo warstw

Ważne architektury CNN

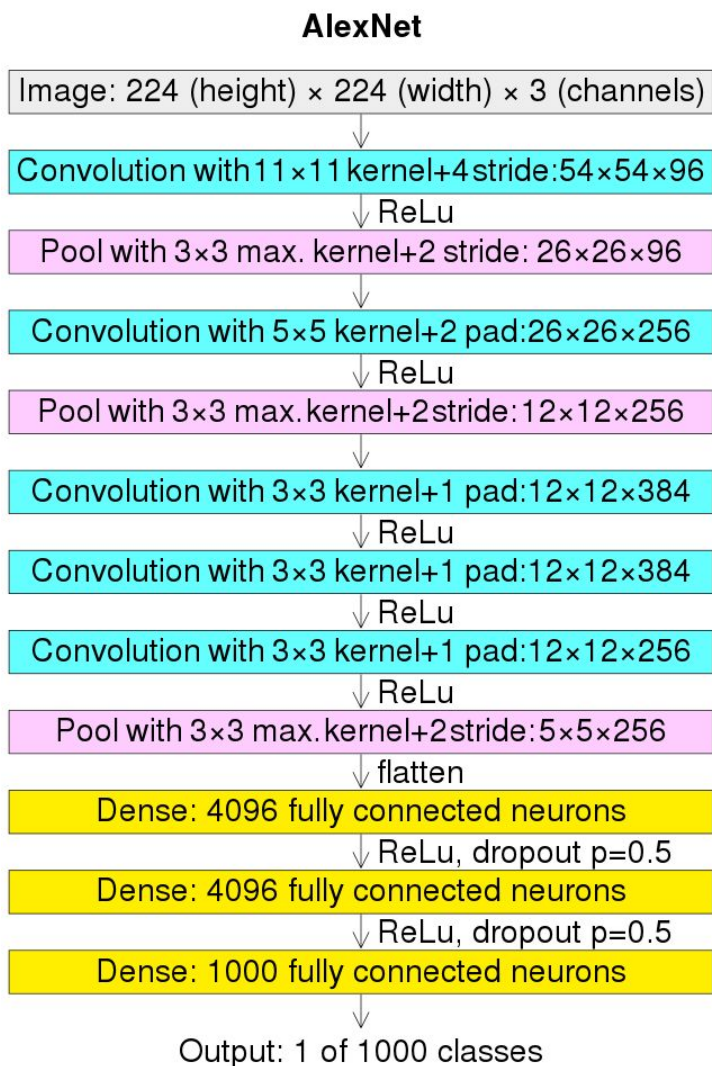
LeNet

- pierwsza udana architektura CNN (1989)
- autor: Yann LeCun
- zastosowanie: rozpoznawanie liczb (kody pocztowe)
- **cechy:**
 - prostota
 - dość duże konwolucje 5x5
 - duży head
 - average pooling
 - aktywacja sigmoidą



AlexNet

- pierwsza bardzo znana architektura CNN
- autorzy: Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton
- wygrała konkurs ImageNet 2012, osiągając błąd testowy 15.3% (drugie miejsce oparte o SVM ok. 25%)
- **cechy:**
 - bardzo duża i dość głęboka sieć neuronowa
 - trening na GPU
 - max pooling
 - aktywacja ReLU



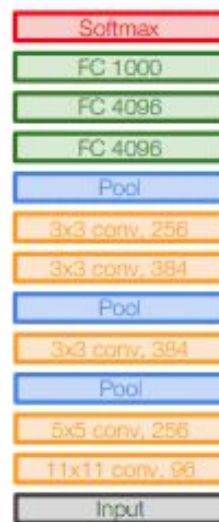
VGG

- pierwsze bardzo głębokie architektury CNN
- autorzy: Visual Geometry Group, University of Oxford
- **obserwacja:** 3 konwolucje 3x3 mają większą nieliniowość niż jedna 7x7 i mniej parametrów (mniejszy overfitting)
- **cechy:**
 - głębokie sieci
 - mniejsze receptive fields
 - duży head
 - budowa blokowa

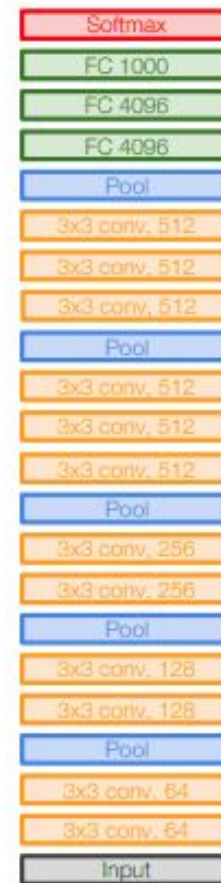
conv(kernel size)-(num channels)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

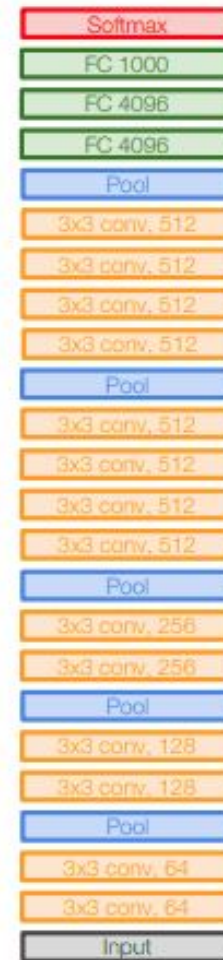
AlexNet vs VGG



AlexNet



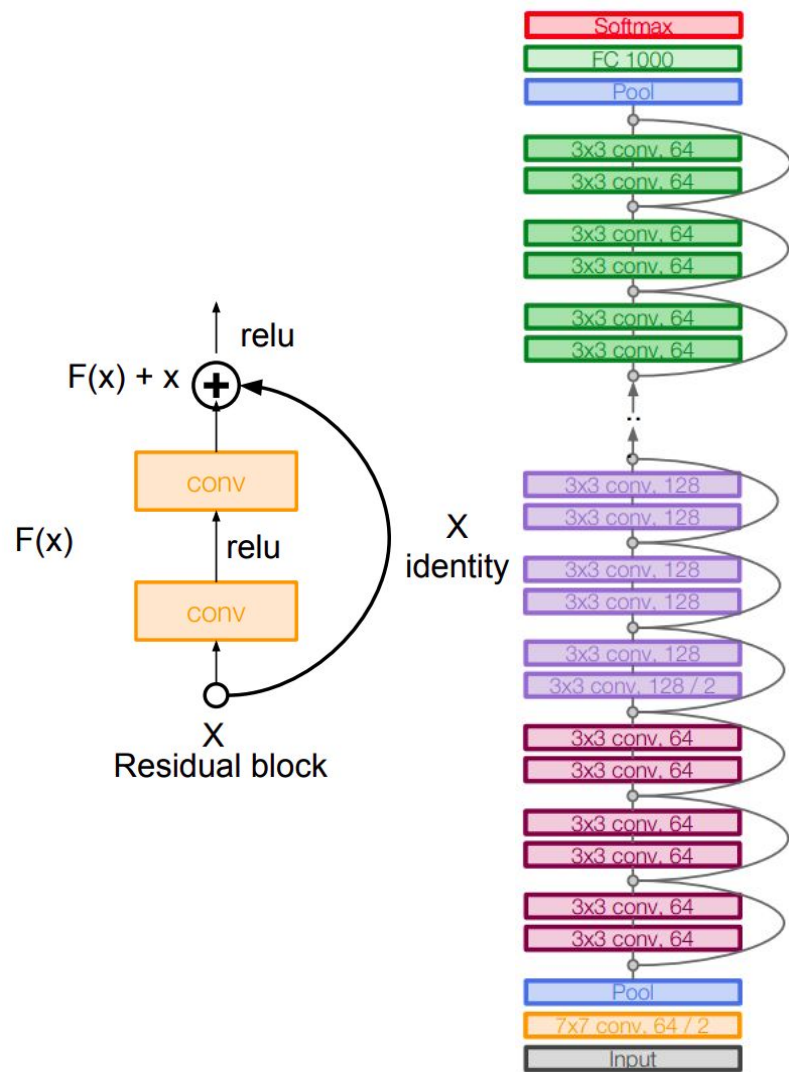
VGG16



VGG19

ResNet

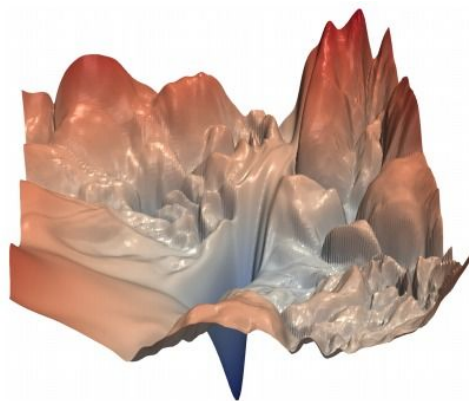
- pierwsze masywnie głębokie sieci
- autorzy: Kaiming He et al.
- **obserwacja:** dodawanie warstw potrafi pogorszyć wynik, ale nie powoduje overfittingu (błąd treningowy i testowy rosną)
- **hipoteza:** sieci mają problem z uczeniem się identyczności
- **residual connections** - wejście "przeskakuje" konwolucje, co daje dostęp do "prostszych" cech wyżej w sieci + usprawnia przepływ gradientu
- okazało się, że powyższe skutkuje też lepszą powierzchnią funkcji kosztu



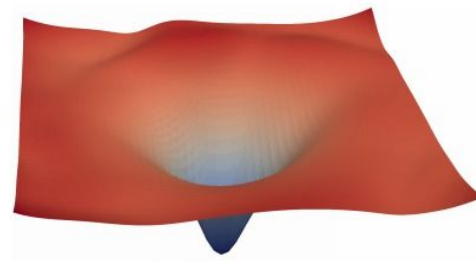
ResNet

- cechy:

- grupa naprawdę głębokich sieci (ResNet18, 34, 50, 101, 152)
- residual connections
- mały head (warstwa liniowa + softmax)

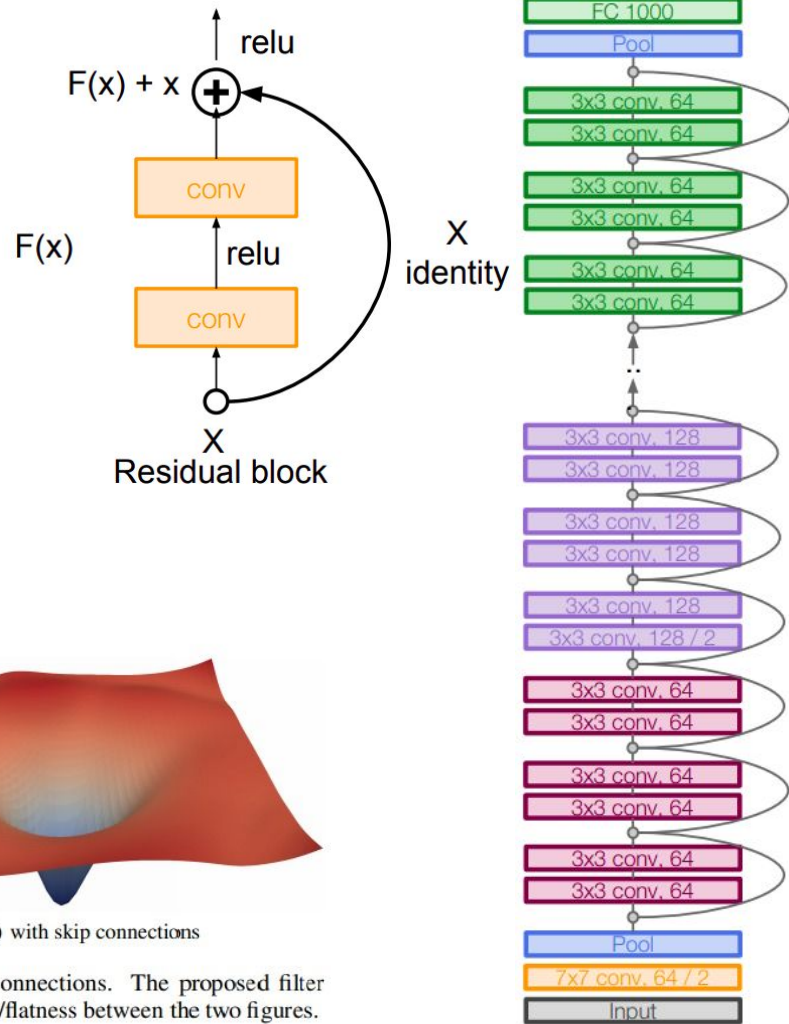


(a) without skip connections



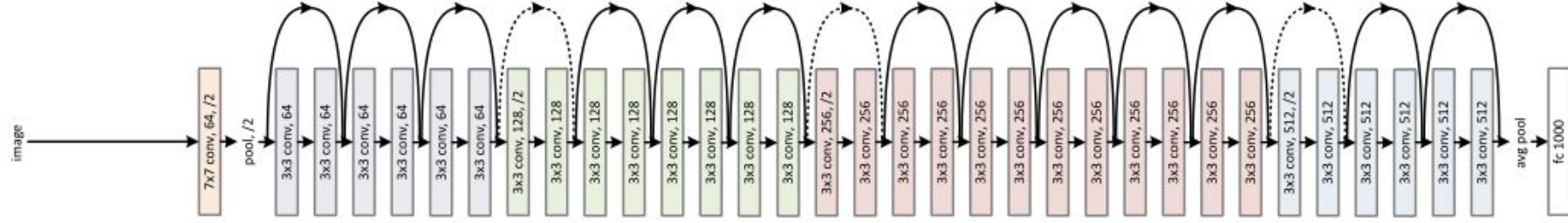
(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.



ResNet34 vs VGG

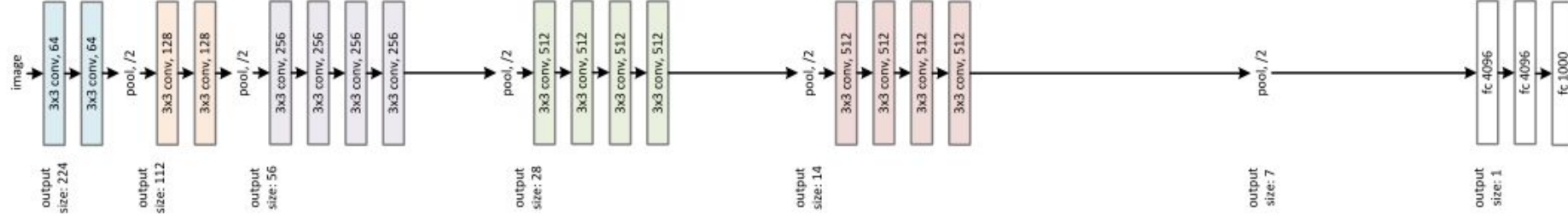
34-layer residual



34-layer plain



VGG-19



Inne ważne architektury

- **MobileNetV1, V2, V3:**

- lekkie sieci, pierwotnie do zastosowań mobilnych / low-latency
- dużo optymalizacji

- **InceptionNetV1, V2, V3, V4 (GoogLeNet):**

- głębokie, potężne sieci o zaawansowanej budowie blokowej
- dużo różnych optymalizacji do redukcji kosztu i regularyzacji

- **DenseNet:**

- wyjście bloku konwolucji ma skip connections do wszystkich kolejnych bloków konwolucji + do wyjścia
- maksymalne rozwinięcie skip connections + kompresja konwolucji, żeby koszt obliczeń nie wybuchł

Pytania?