SZTUCZNA INTELIGENCJA

UCZENIE GŁĘBOKIE I GŁĘBOKIE SIECI NEURONOWE

Deep Learning

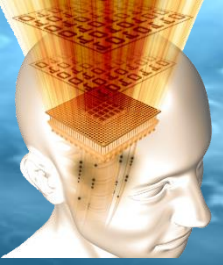
AGH

Adrian Horzyk

Czym jest uczenie głębokie?

Uczenie głębokie (zwane też uczeniem hierarchicznym) jest klasą algorytmów uczenia maszynowego i strategii uczących takich, że:

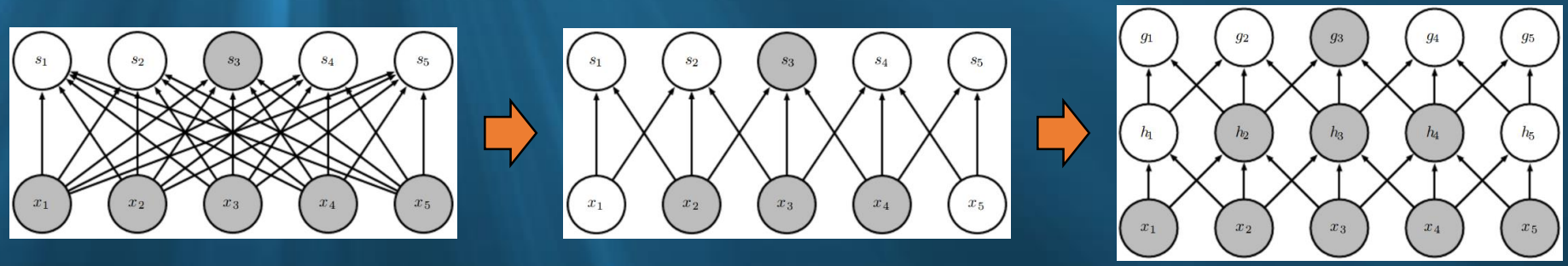
- ✓ Rozwijają strukturę hierarchiczną i reprezentację podstawowych i wtórnych cech, reprezentujących różne poziomy abstrakcji.
- ✓ Wykorzystują kaskadę wielu warstw neuronów (lub innych jednostek obliczeniowych) różnych rodzajów w celu stopniowej ekstrakcji cech i ich transformacji w celu osiągnięcia hierarchii cech wtórnych/pochodnych, które prowadzą do lepszych wyników zbudowanych na ich podstawie sieci neuronowych. W ten sposób próbują określić bardziej skomplikowane cechy na podstawie prostszych cech.
- ✓ Stosują różne strategie uczenia nadzorowanego i nienadzorowanego dla różnych warstw,
- ✓ Stopniowo rozwijają i aktualizują strukturę sieci dopóki występuje znacząca poprawa wyników działania sieci.



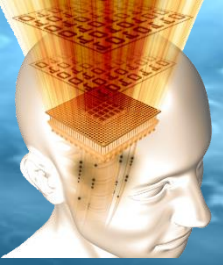
Strategie Uczenia Głębokiego

Strategie uczenia głębokiego zakładają zdolność do:

- aktualizuj tylko wybraną część neuronów, która najlepiej odpowiada na dane wejściowe, tak więc inne neurony nie są aktualizowane i ich parametry (np. wagi i progi) nie są zmieniane,
- unikaj łączenia neuronów na zasadzie każdy-z-każdym pomiędzy kolejnymi warstwami, czyli strategii często stosowanej w sieciach MLP oraz wielu innych, lecz spróbuj pozwolić neuronom specjalizować się w rozpoznawaniu wzorców składowych i cech, które mogą być wyekstrahowane z ograniczonego podzbioru wejść:



- twórz połączenia pomiędzy różnymi warstwami i podsieciami, czyli nie tylko pomiędzy kolejnymi warstwami
- użyj wiele podsieci które mogą być łączone w różny sposób w celu pozwolenia neuronom tych podsieci na specjalizację w definiowaniu i rozpoznawaniu ograniczonego podzbioru cech i wzorców składowych,
- Pozwól neuronom na specjalizację i nie nakładaj reprezentowanych regionów i reprezentacji tych samych cech lub wzorców składowych.

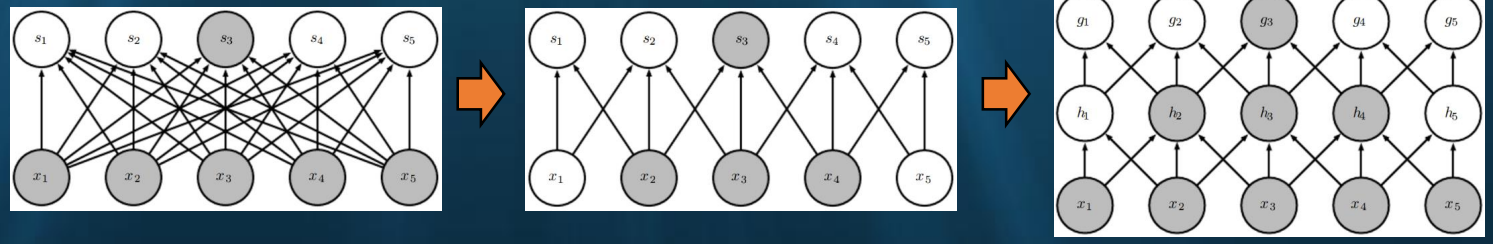


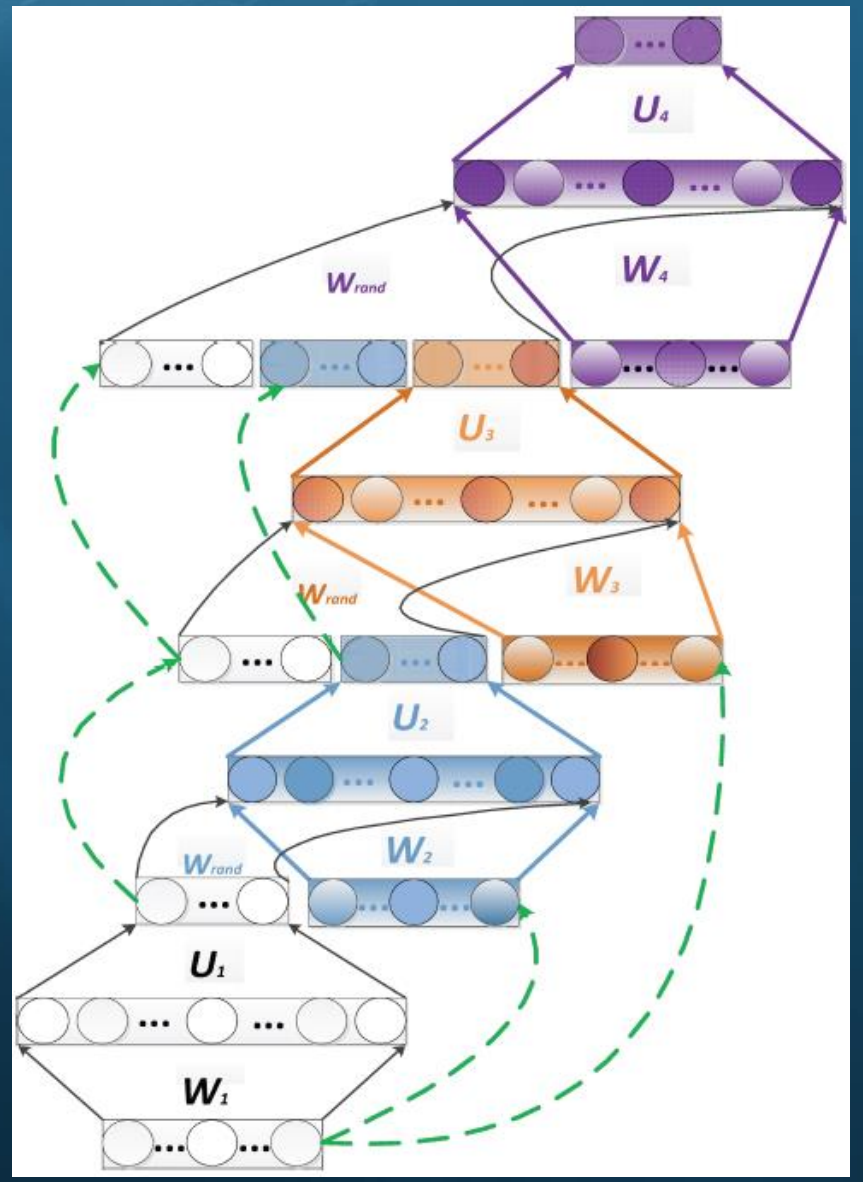
Strategie Uczenia Głębokiego

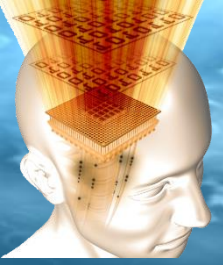
W głębokich strukturach sieci neurony mogą mieć połączenia wejściowe dochodzące z różnych warstw, kombinować ze sobą różne cechy wyekstrahowane wcześniej, w celu obliczenia wyjść tych neuronów.

W trakcie zajęć laboratoryjnych:

Można spróbować wykorzystać tą strategię zamiast klasycznych połączeń każdy-z-każdym stosowanych w sieciach MLP i porównać wyniki działania z przypadkiem, gdy neurony są podłączone tylko do ograniczonej podgrupy neuronów z poprzedniej/poprzednich warstw.



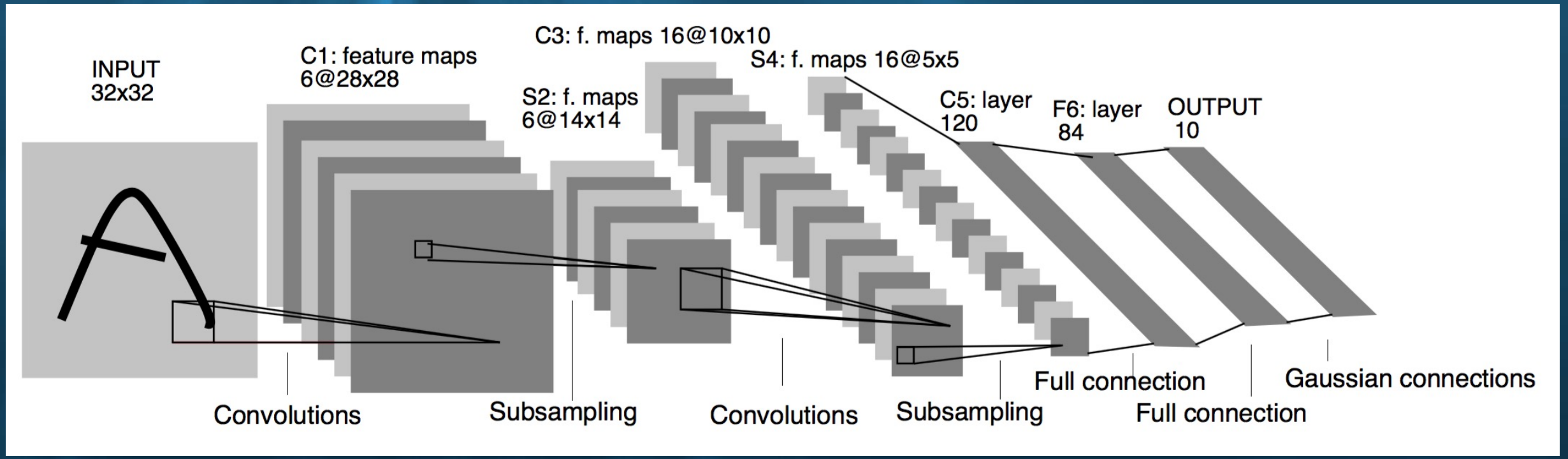


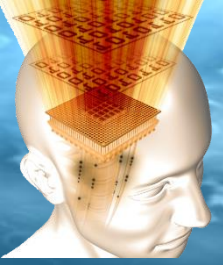


Różnorodność Głębokich Struktur Sieciowych

Architektury głębokie mogą zawierać wiele podsieci i wiele warstw różnych typów połączonych ze sobą:

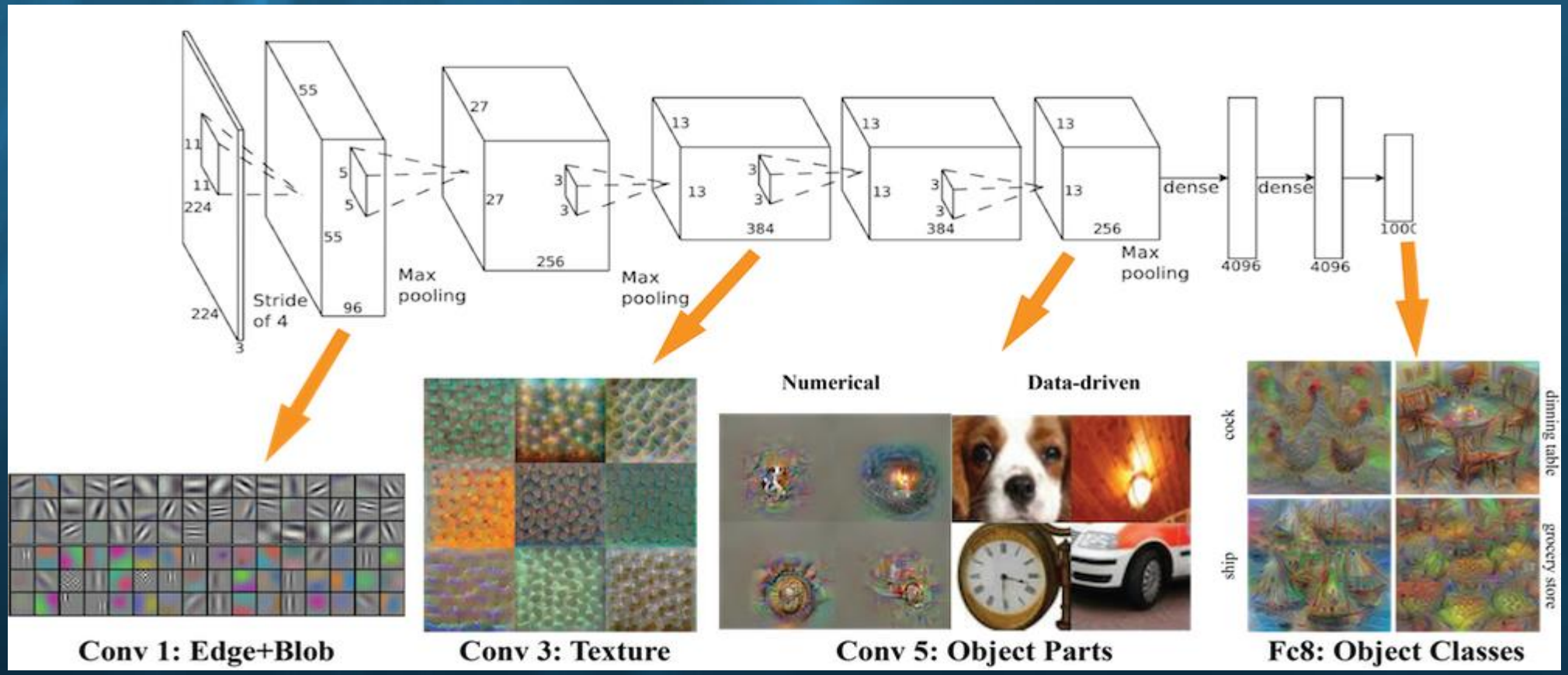
- Warstwy próbkujące (subsampling layers) mogą być skombinowane z warstwami konwolucyjnymi.
- Każda warstwa może zawierać wiele podsieci tego samego rodzaju.

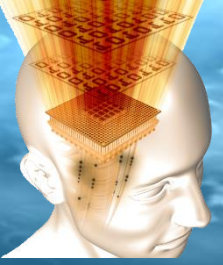




Różnorodność Głębokich Struktur Sieciowych

Architektury głębokie mogą zawierać wiele podsieci specjalizujących się w klasyfikacji lub rozpoznawaniu cech, przetwarzaniu specjalnych rodzajów danych wejściowych lub łączeniu danych z poprzedniej warstwy obliczając np. maksima (max-pooling):

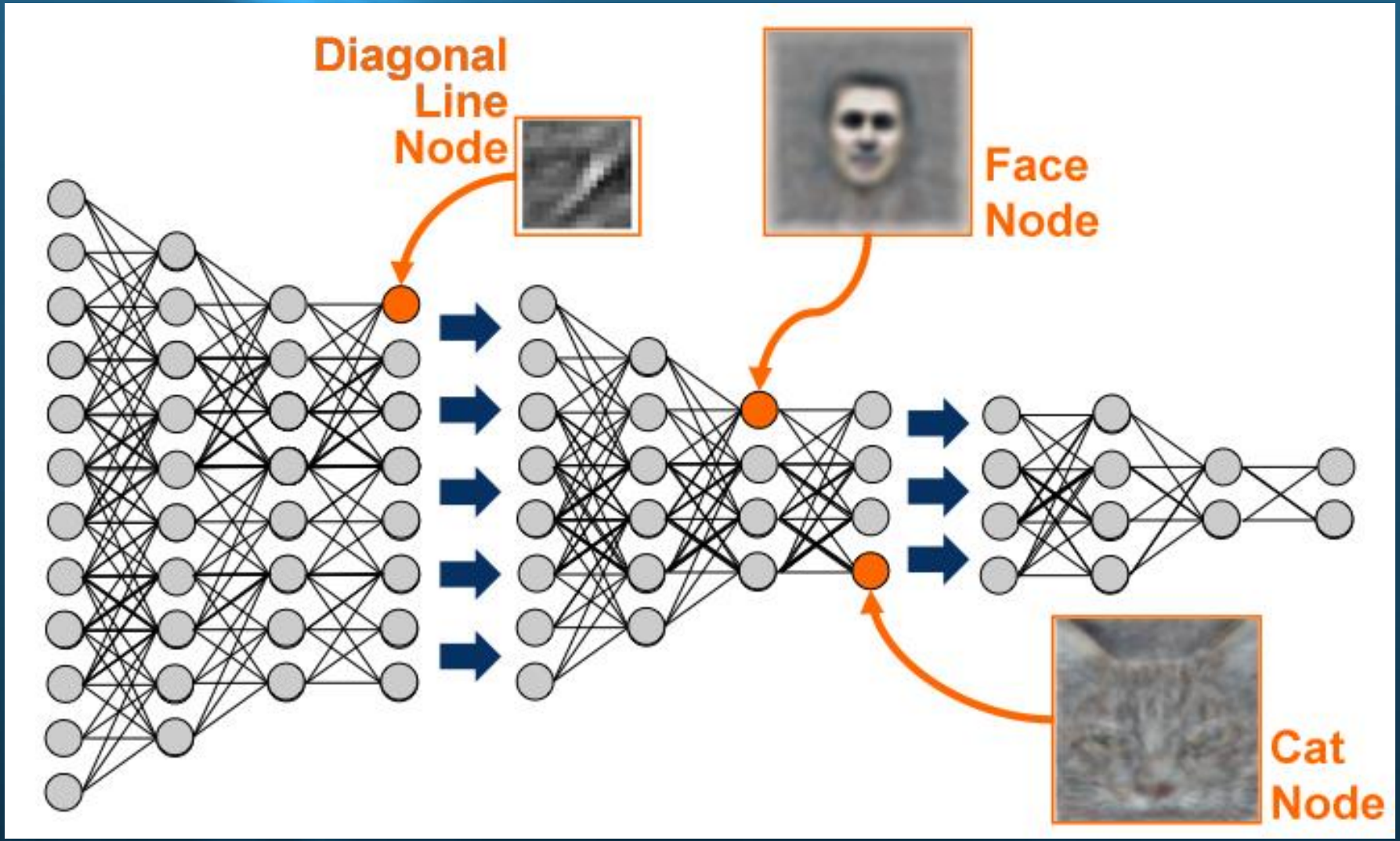


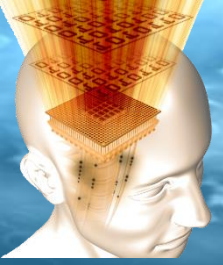


Różnorodność Głębokich Struktur Sieciowych

Ważną częścią każdej sieci głębokiej jest zawsze proces związany z ekstrakcją cech.

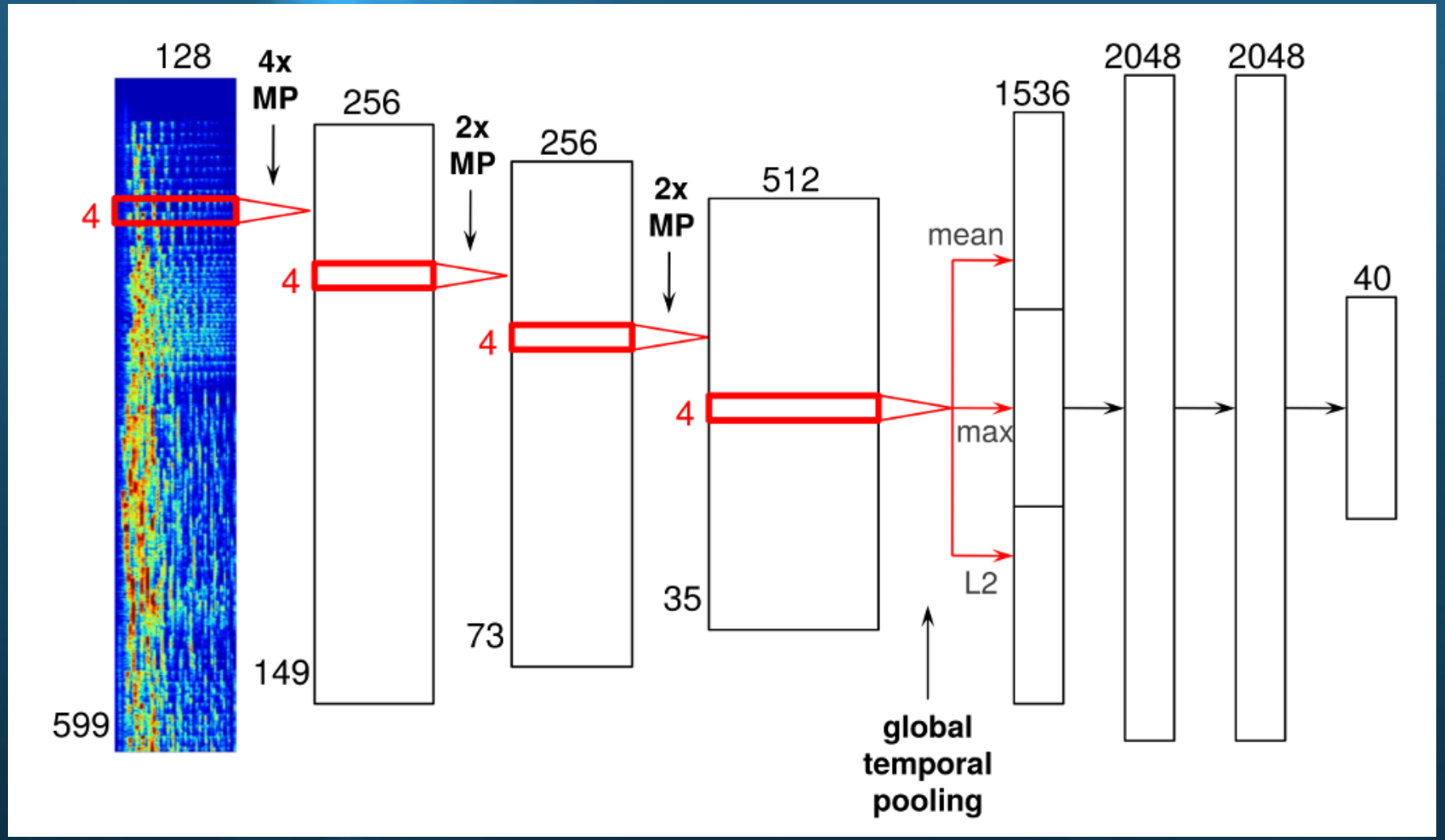
Czasami możliwe jest wskazanie neuronu wyspecjalizowanego w rozpoznawaniu i reagowaniu na specyficzną cechę.

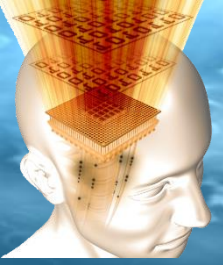




Różnorodność głębokich struktur sieciowych

W innych przypadkach możemy łączyć, kombinować lub wybierać dane wykorzystujące maksima, wartości średnie, sumy ważone, filtry itp.



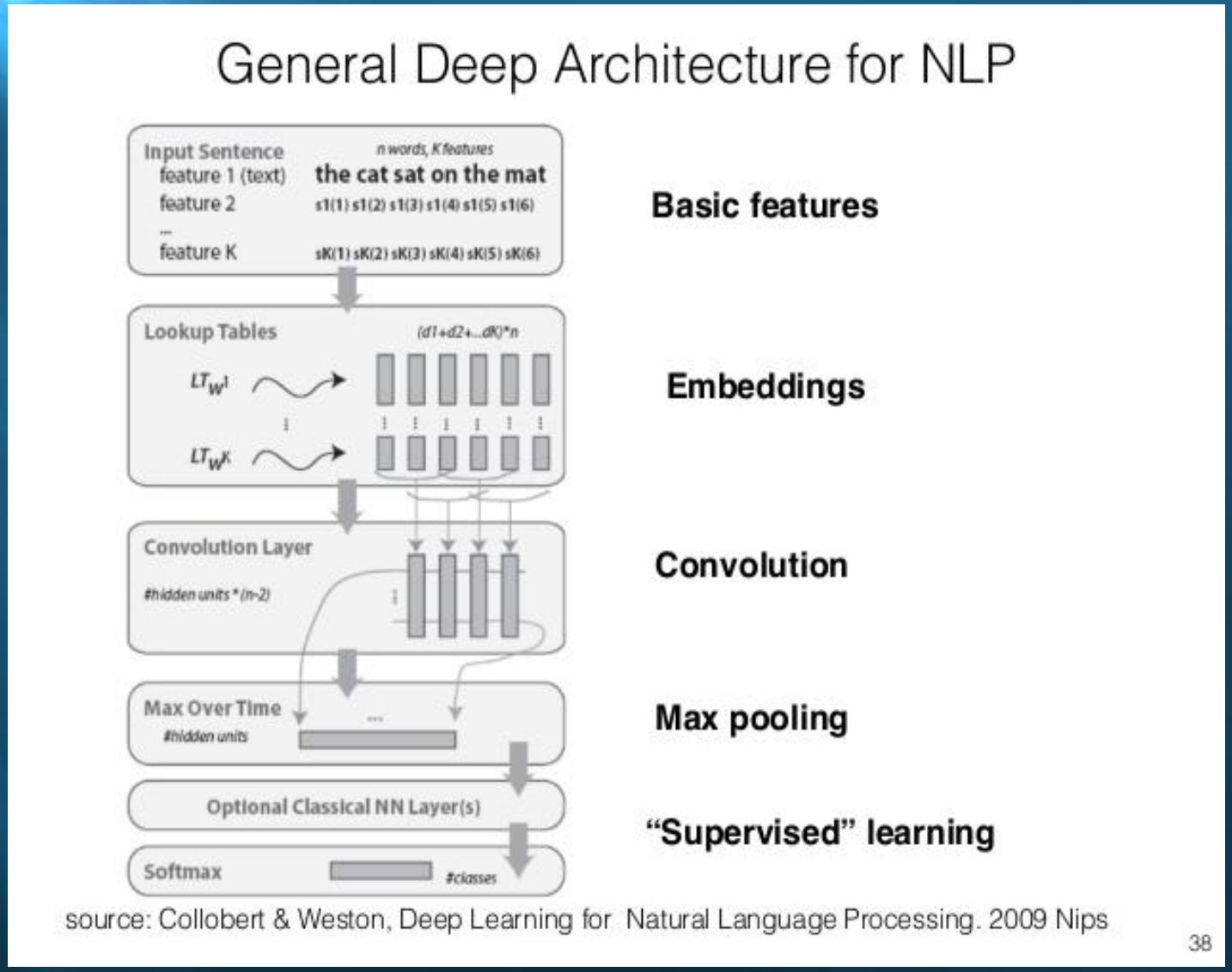


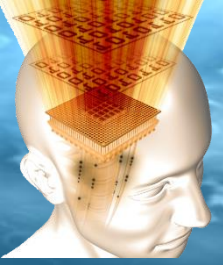
Różnorodność Głębokich Struktur Sieciowych

Sieci głębokie próbują rozdzielić przetwarzanie neuronowe na kilka faz, w trakcie których najpierw, podstawowe, potem wtórne, a następnie bardziej złożone cechy są rozpoznawane.

Sieci te mogą zawierać również warstwy konwolucyjne i łączące.

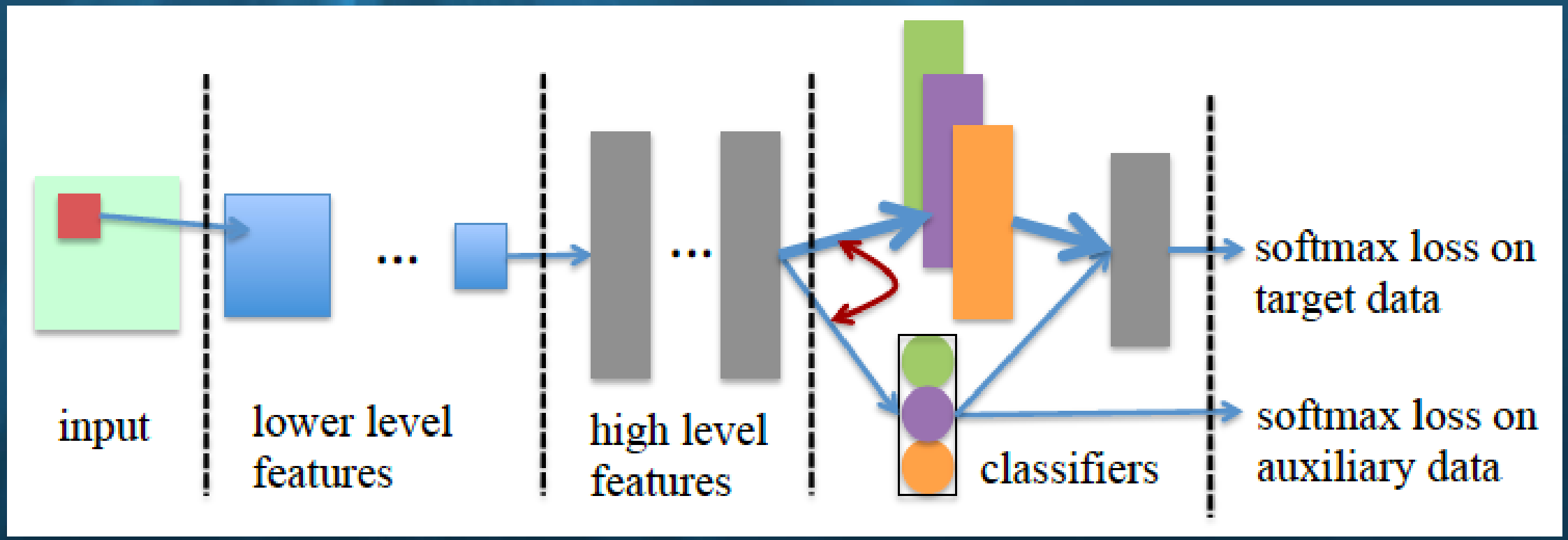
Zwykle ostatnie warstwy uczone są z wykorzystaniem uczenia nadzorowanego, tj. propagacja wsteczna czy spadek gradientu, w celu zebrania i finalnego dostrojenia wyników wyjściowych.

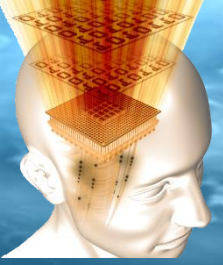




Różnorodność Głębokich Struktur Sieciowych

Sieci głębokie zwykle próbuję wyekstrahować wartościowe cechy na początku, a następnie wykorzystują inne podsieci, żeby je pogrupować i sklasyfikować albo wykorzystać do regresji czy aproksymacji. Finalnie, ostatnie warstwy filtrują najlepsze wyniki stosując funkcje minimum i maksimum albo dokonują finalnej aproksymacji zgodnie z określoną klasą w trakcie uczenia nadzorowanego:

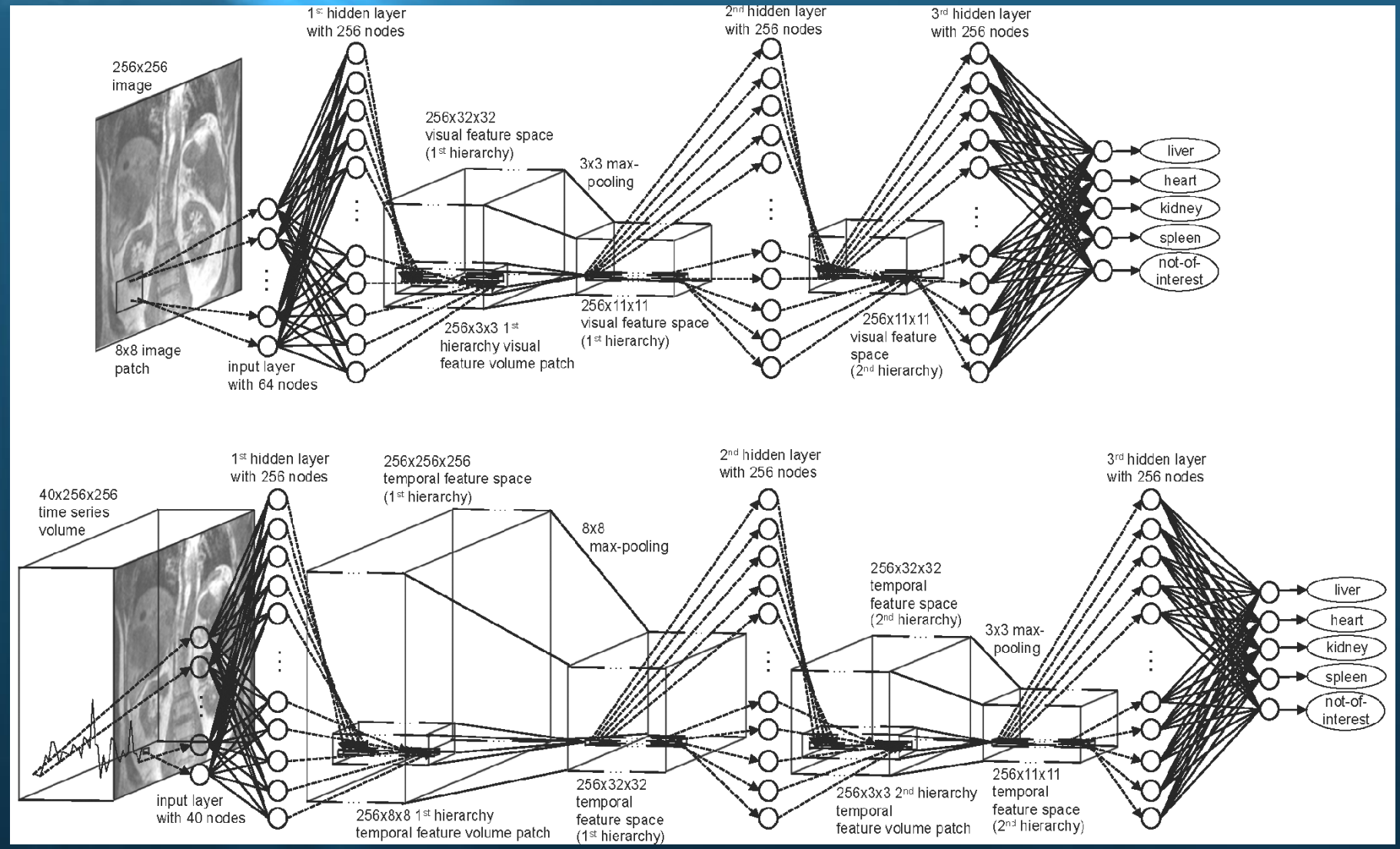


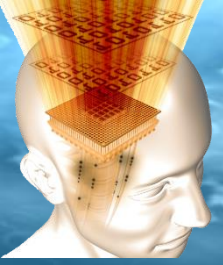


Różnorodność Głębokich Struktur Sieciowych

Tutaj zaprezentowano sieci głębokie wykorzystane do rozpoznawania narządów wewnętrznych człowieka:

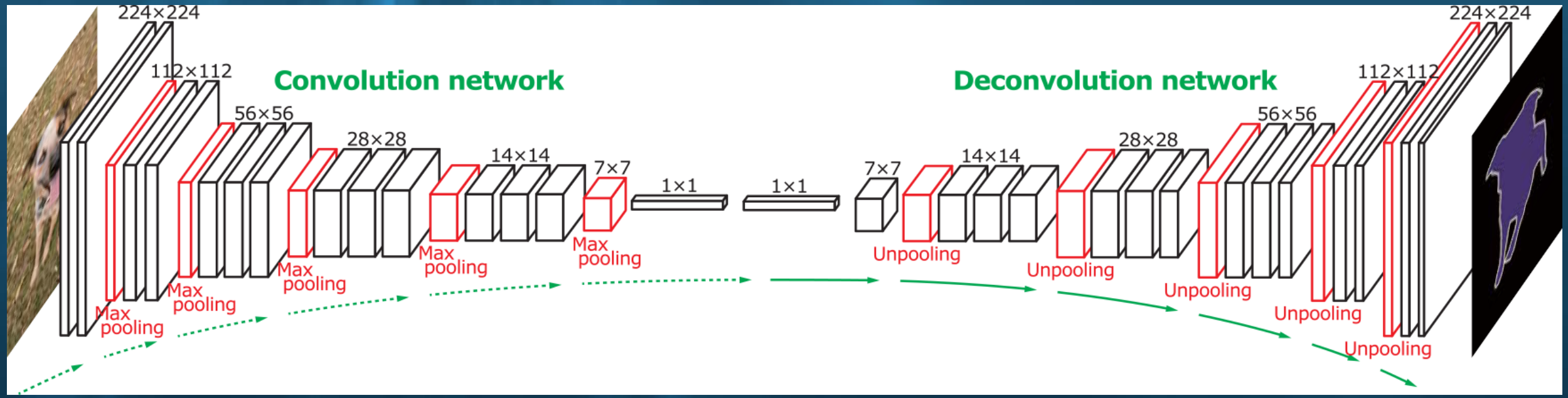
- Wątroba (liver)
- Serce (heart)
- Nerka (kidney)
- Śledziona (spleen)
- i inne organy…





Różnorodność Głębokich Struktur Sieciowych

Możemy wykorzystać niektóre sieci głębokie do definiowania klas w celu ich porównania lub do rekonstrukcji uogólnionych obiektów:



Podsumowanie

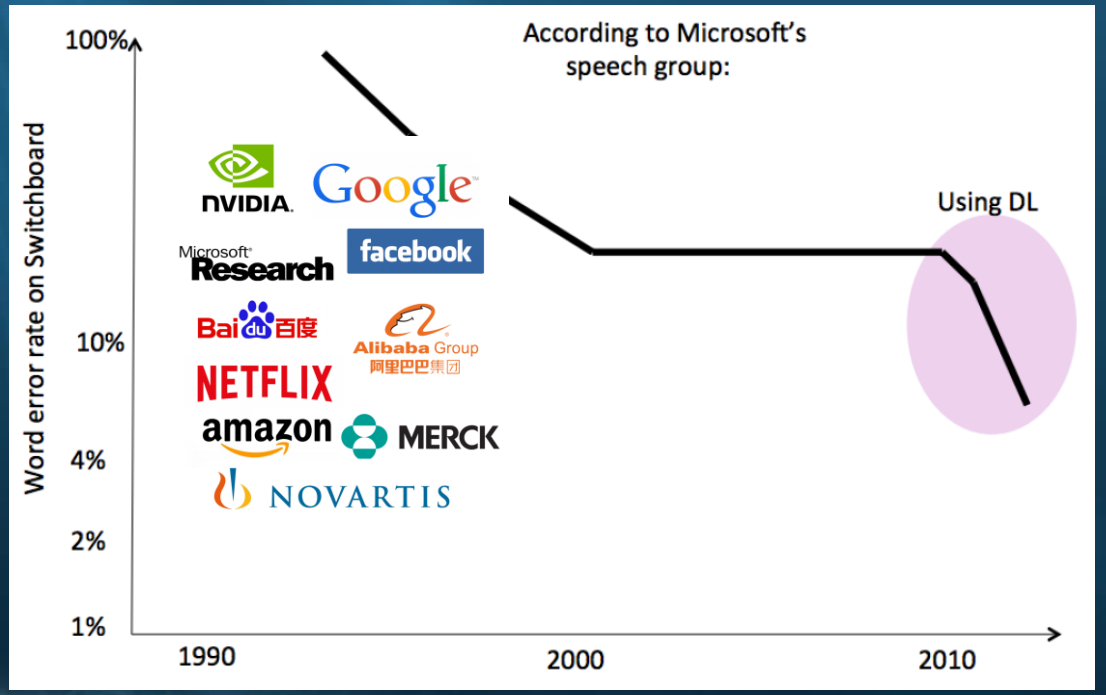
Algorytmy uczenia głębokiego umożliwiają nam:

- ✓ uczenie hierarchicznych kaskad warstw lub podsieci,
- ✓ reprezentację podstawowych, wtórnych i wyższych rzędów cech,
- ✓ wykorzystanie różnych nadzorowanych i nienadzorowanych strategii uczenia poszczególnych warstw,
- ✓ stopniowy rozwój struktury sieci i stopniowe uczenie/adaptację neuronów lub innych jednostek obliczeniowych i kolejnych warstwach,
- ✓ aktualizację wag tylko części neuronów lub innych jednostek obliczeniowych, które najlepiej lub najmocniej odpowiadają na dane wejściowe albo najmocniej różnicują cechy wzorców uczących,
- ✓ łączenie neuronów pomiędzy różnymi warstwami, a nie tylko pomiędzy kolejnymi warstwami,
- ✓ Uzyskanie różnych poziomów abstrakcji dzięki podziałowi przetwarzania obrazu pomiędzy warstwy i ich podsieci (w plastrach).

Głęboko – oznacza poprawę uczenia!

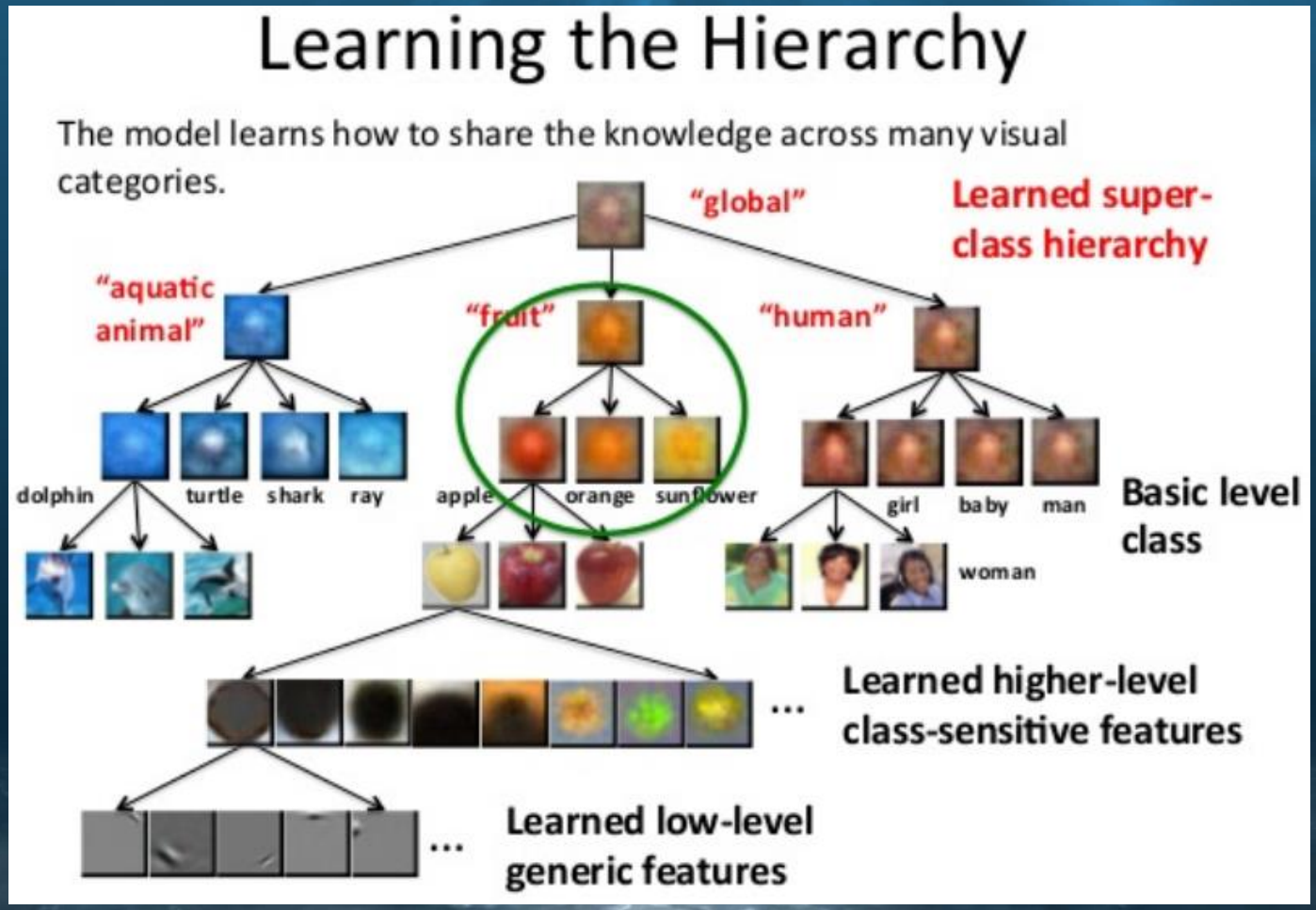
Algorytmy uczenia głębokiego, głębokie sieci i strategie zwykle poprawiają wyniku uczenia i dają lepsze wyniki w porównaniu do innych metod uczenia biorąc pod uwagę różne zastosowania, np. w :

- Systemach wizyjnych i rozpoznawaniu wzorców
- Klasyfikacji i klasteryzacji
- Eksploracji danych i wyszukiwaniu informacji
- Rozpoznawaniu mowy
- Analizie języka naturalnego
- Systemach decyzyjnych i rekomendacyjnych



Głęboko – oznacza poprawę uczenia!

Uczenie głębokie pozwala uczyć sieci różne kategorie i hierarchie cech w celu poprawienia wyników działania sieci, np. wynikowej klasyfikacji:

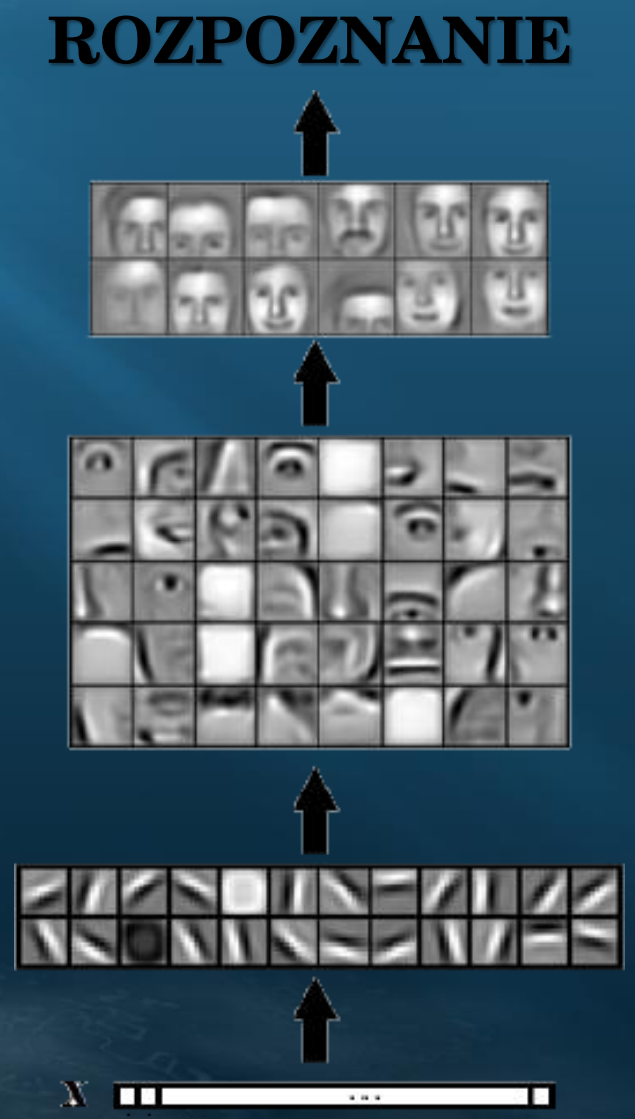


Automatyczna Ekstrakcja Cech

Sieci głębokie poprawiają wyniki uczenia się dzięki stopniowemu i hierarchicznemu procesowi ekstrakcji cech z surowych danych.

Przede wszystkim szukamy cech:

- ✓ różnicujących (dyskryminatywnych)
- ✓ pewnych
- ✓ niezmiennych



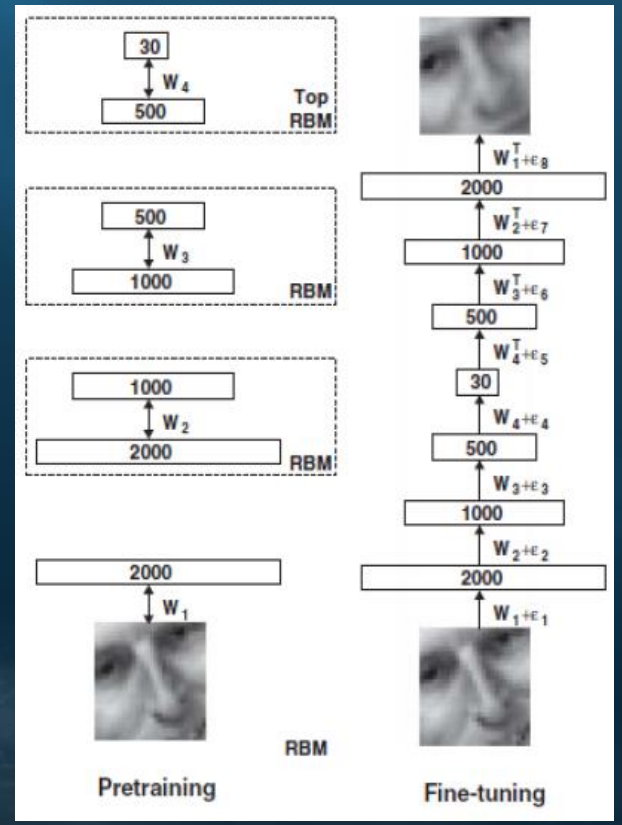
- Lee et al. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, Int. Conf. ICML 2009

Problem Zanikającego Gradientu

W przypadku wykorzystania strategii uczących wykorzystujących metody gradientowe dla wielu warstw (np. sieci MLP) zwykle natrafiamy na **problem zanikającego gradientu**, ponieważ pochodne są z przedziału [0, 1], więc wielokrotne przemnażanie prowadzi do bardzo małych liczb prowadząc do coraz mniejszych zmian wag w warstwach coraz dalej położonych od wyjścia sieci, stosując np. propagację wsteczną błędów.

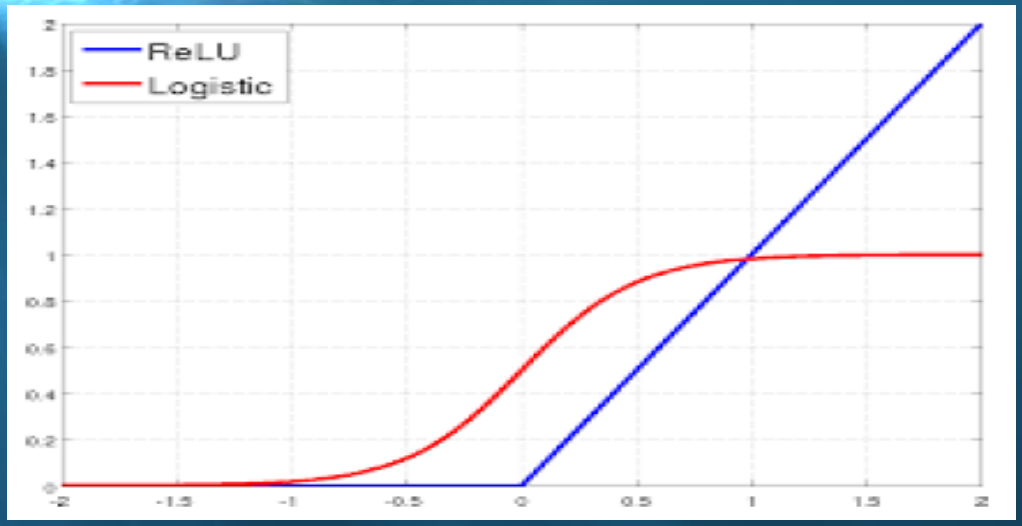
Ten problem może zostać rozwiązany za pomocą treningu wstępnego i strategii końcowego dostrojenia, które najpierw trenują model warstwa po warstwie w nienadzorowany sposób (np. używając głębokiego autoenkodera), a następnie wykorzystujemy algorytm propagacji wstecznej do dostrojenia sieci.

Hinton, Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. Science 2006



Rektyfikowane Jednostki Liniowe (ReLU)

Możemy również wykorzystać jednostki ReLU w celu wyeliminowania problemu zanikającego gradientu.



Jednostki ReLU są zdefiniowane jako:
f(x) = max(0, x) zamiast funkcji logistycznej.

Strategia wykorzystująca jednostki ReLU oparta jest na uczeniu pewnych cech dzięki rzadszym aktywacjom tych jednostek.

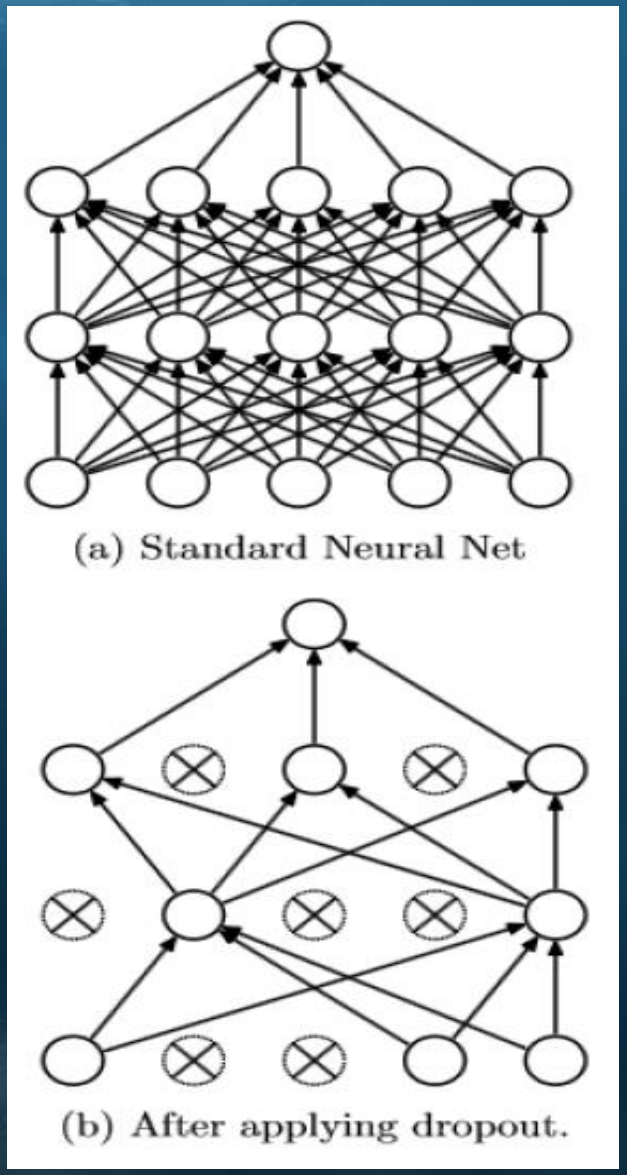
Inną korzyścią jest to, że proces uczenia przebiega zwykle szybciej

Nair, Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. ICML 2010

Technika Regularyzacji Opuszczeń

Można wykorzystać też technikę regularyzacji opuszczeń, która wybiera do aktualizacji tylko te neurony z różnych warstw, które są najlepiej przystosowane.

Ta technika zapobiega przeuczeniu sieci neuronowych, zapobiega psuciu wag dobrze przystosowanych do innych wzorców, jak również przyspiesza proces nauki.

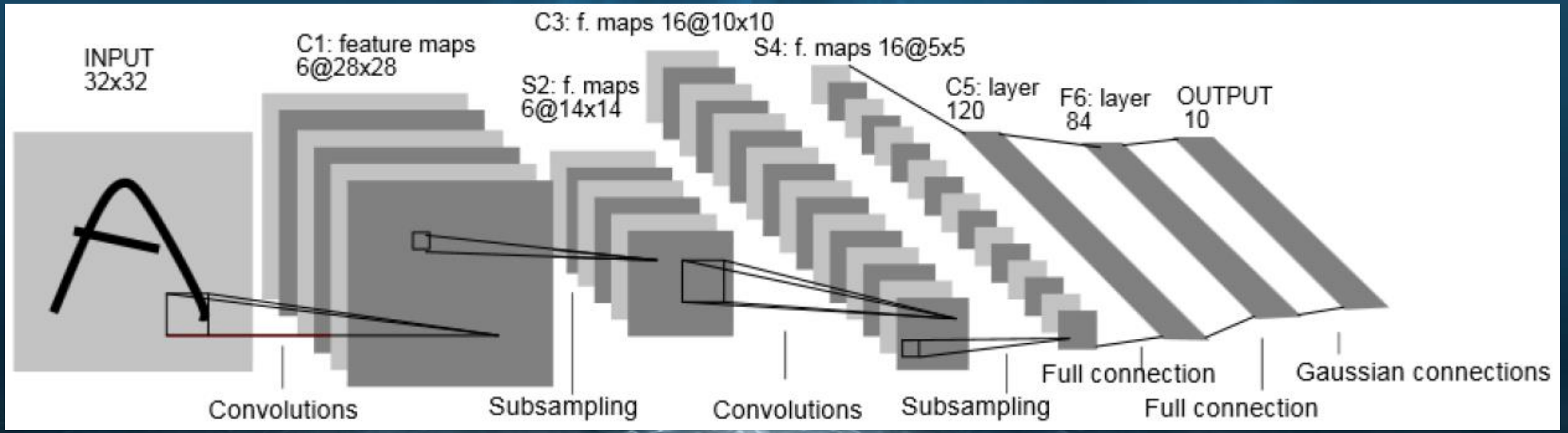


Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR 2014

Głębokie Sieci Konwolucyjne

Głębokie sieci konwolucyjne (CNN) potrafią stopniowo filtrować różne części danych uczących i **wyostrzać ważne cechy** w procesie dyskryminacji wykorzystanym do rozpoznawania lub klasyfikacji wzorców.

LeCun et al. Gradient-Based Learning Applied to Document Recognition. Proc. of IEEE 1998

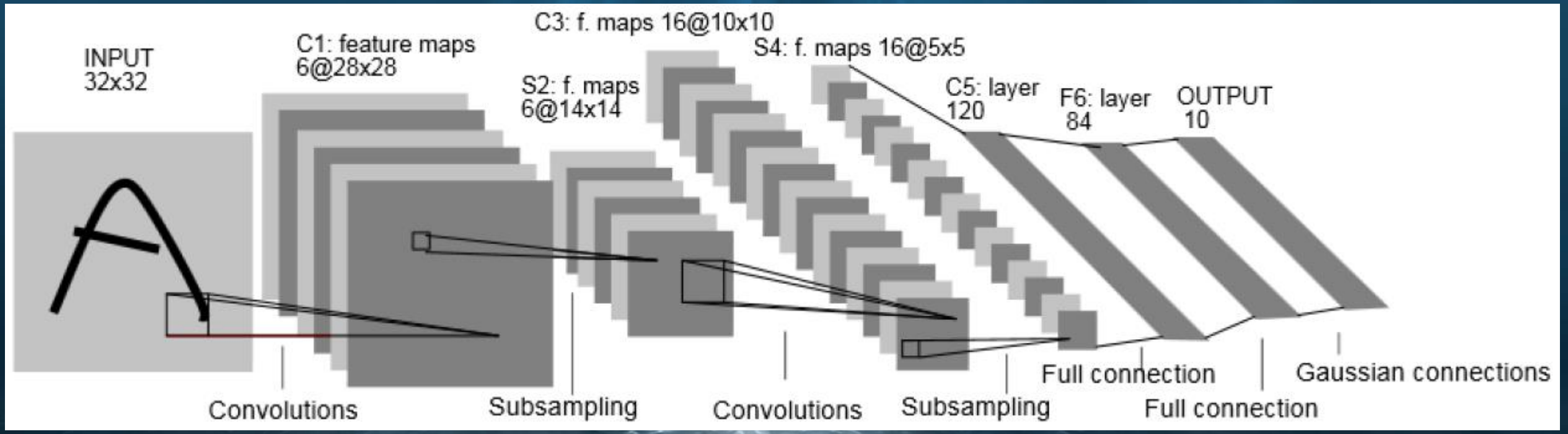


Głębokie Sieci Konwolucyjne

W każdej konwolucji możemy wyróżnić:

- Ilość parametrów w każdej warstwie: Ilość kanałów * Ilość filtrów * szerokość filtra * wysokość filtra
- Ilość jednostek ukrytych w każdej warstwie: Ilość filtrów * szerokość wzorca * wysokość wzorca

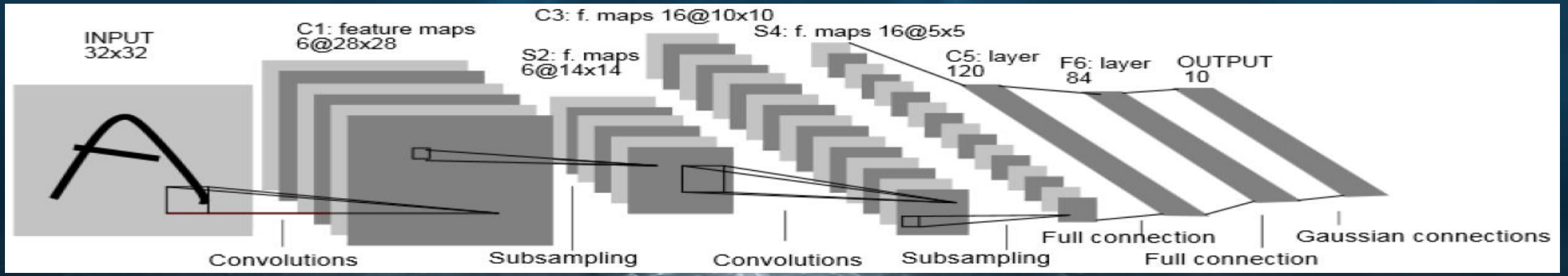
LeCun et al. Gradient-Based Learning Applied to Document Recognition. Proc. of IEEE 1998



Łączenie z Wybieraniem Maksimum

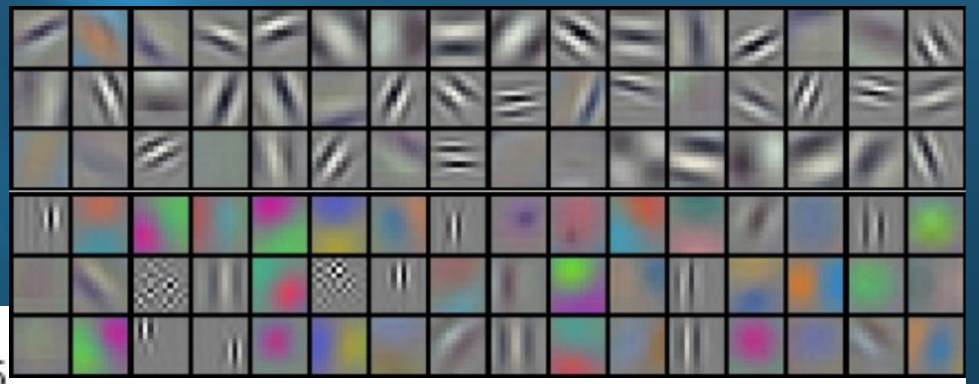
Warstwa łącząca (pooling layer) służy do progresywnej redukcji rozmiaru przestrzennego do zredukowania ilości cech i złożoności obliczeniowej sieci.

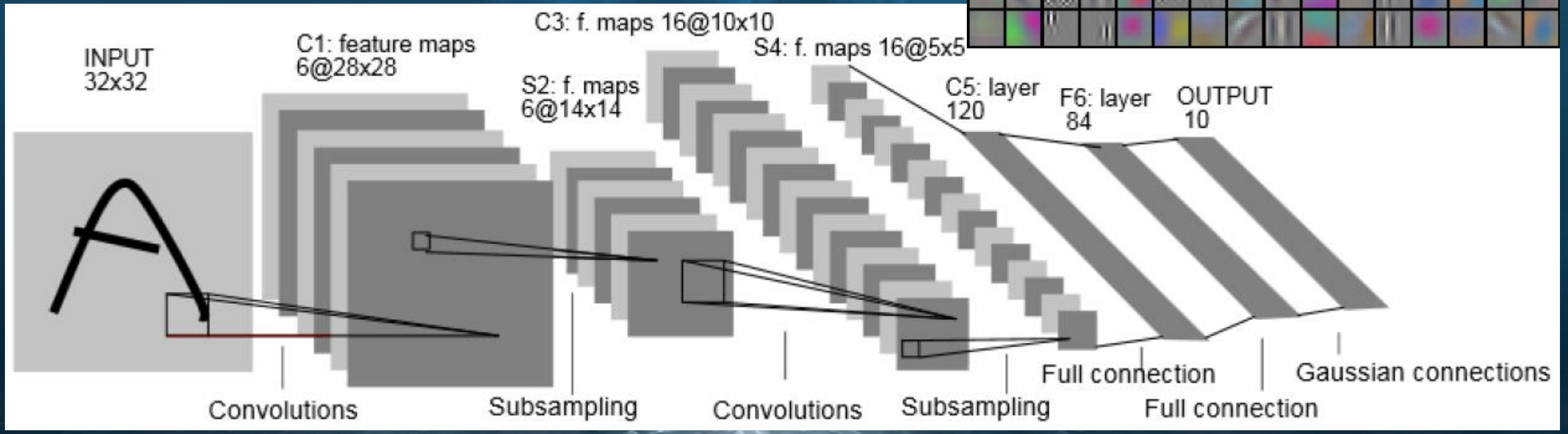
Najczęściej w sieciach konwolucyjnych stosujemy warstwę **MaxPool** która that przesuwa filtry 2x2 przez całą macierz wyciągając największą wartość z okna filtra i zapisuje ją do następnej mapy.
Najważniejszy powód stosowania warstw łączących jest uchronienie modelu przed przeuczeniem.
Czasami stosujemy też warstwę opuszczającą, która zastępuje warstwę łączącą.
Należy być ostrożnym przy stosowaniu warstwy łączącej, szczególnie w zadaniach wizyjnych, gdyż może to spowodować utratę lokalnej wrażliwości modelu mimo zmniejszenia rozmiaru modelu.



Próbkowanie i Konwolucje

Konwolucje pozwalają na ekstrakcję prostych cech w początkowych warstwach sieci, np. rozpoznają krawędzie o różnej orientacji lub różnokolorowe plamy, a następnie plastry, koła w kolejnych warstwach. Każda warstwa konwolucyjna zawiera cały zbiór filtrów (np. 8 filtrów), a każdy z nich generuje osobną mapę aktywacji 2D. Układamy te mapy aktywacyjne na stercie wzdłuż wymiaru głębokości sieci i produkujemy obraz wyjściowy.



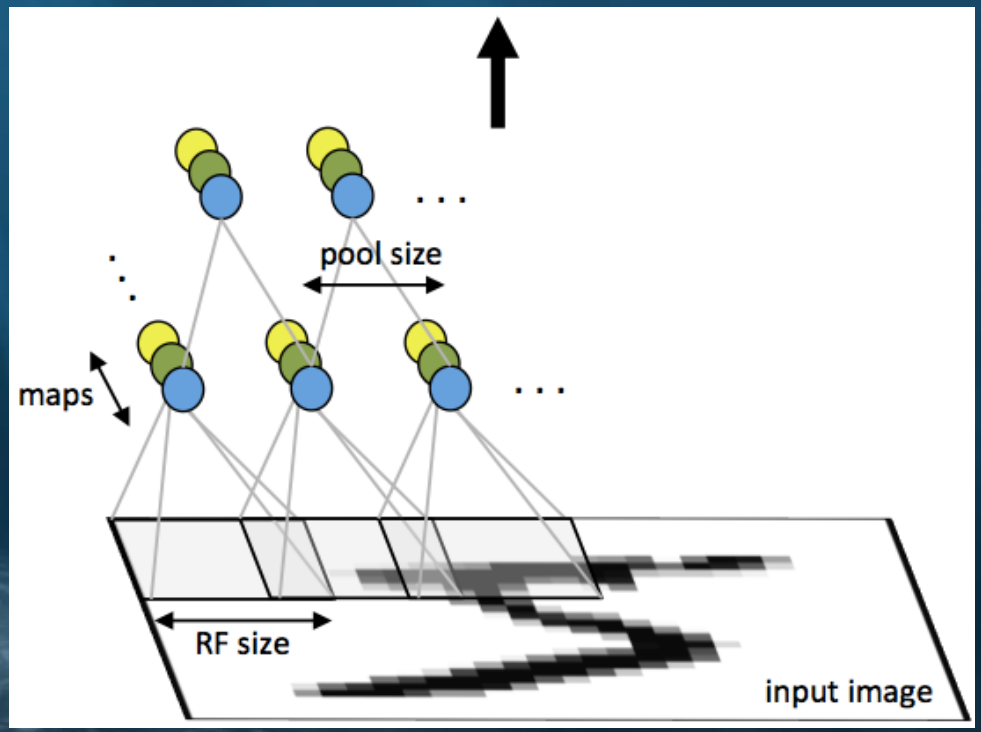


Konwolucyjne Sieci Neuronowe

Konwolucyjne sieci neuronowe (*Convolutional Neural Network - CNN*) składają się z jednej lub wielu warstw konwolucyjnych (typowych dla kroku próbkowania, określającego subwzorce), a następnie przez jedną lub w pełni połączone warstwy tak jak w klasycznej wielowarstwowej sieci, np. MLP, SVM, SoftMax itp. **Głęboka sieć konwolucyjna** zawiera wiele warstw.
Sieci kowolucyjne są łatwe do uczenia, gdyż zawierają mniej parametrów (wykorzystując te same wagi) niż typowe sieci neuronowe z dokładnością do ilości warstw konwolucyjnych i ich rozmiaru.

Ten rodzaj sieci neuronowych jest predestynowany do obliczeń na strukturach 2D (tj. obrazy).

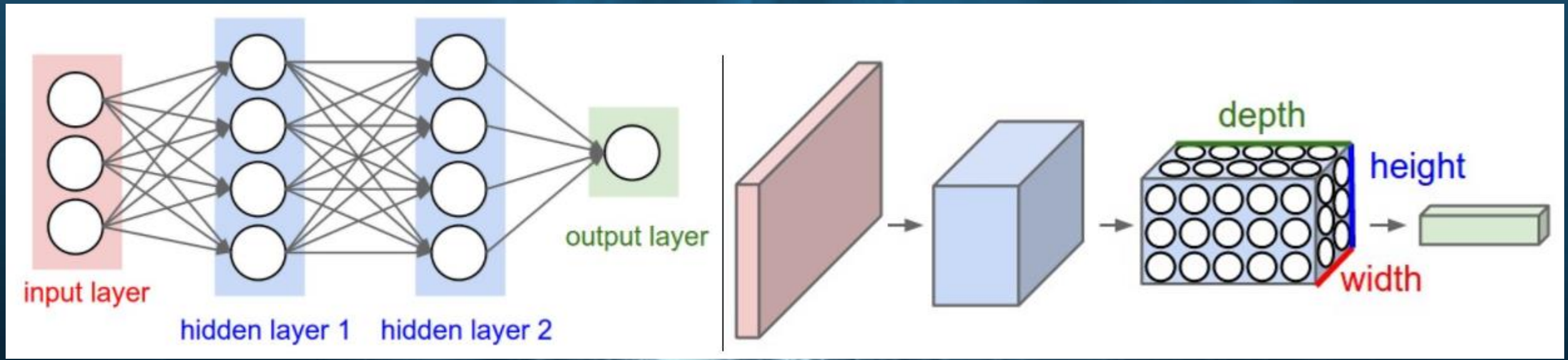
Na rysunku w pierwszej warstwie sieci konwolucyjnego łączenia neurony o tym samym kolorze powiązały wagi i jednostki różno-kolorowe w celu reprezentacji różnych map filtrów.



http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

Convolutional Neural Networks

Sieci konwolucyjne porządkują jednostki obliczeniowe („neurony”) w strukturze 3D: **szerokość, wysokość i głębokość**. Neurony w każdej warstwie są połączone do małego regionu warstwy poprzedniej zamiast do wszystkich, jak temu jest w typowych sieciach neuronowych. Ponadto CNN (e.g. CIFAR-10) redukuje pełne obrazy do pojedynczych wektorów wyjściowych reprezentujących wyniki dla poszczególnych klas, jak przedstawiono na poniższym rysunku.



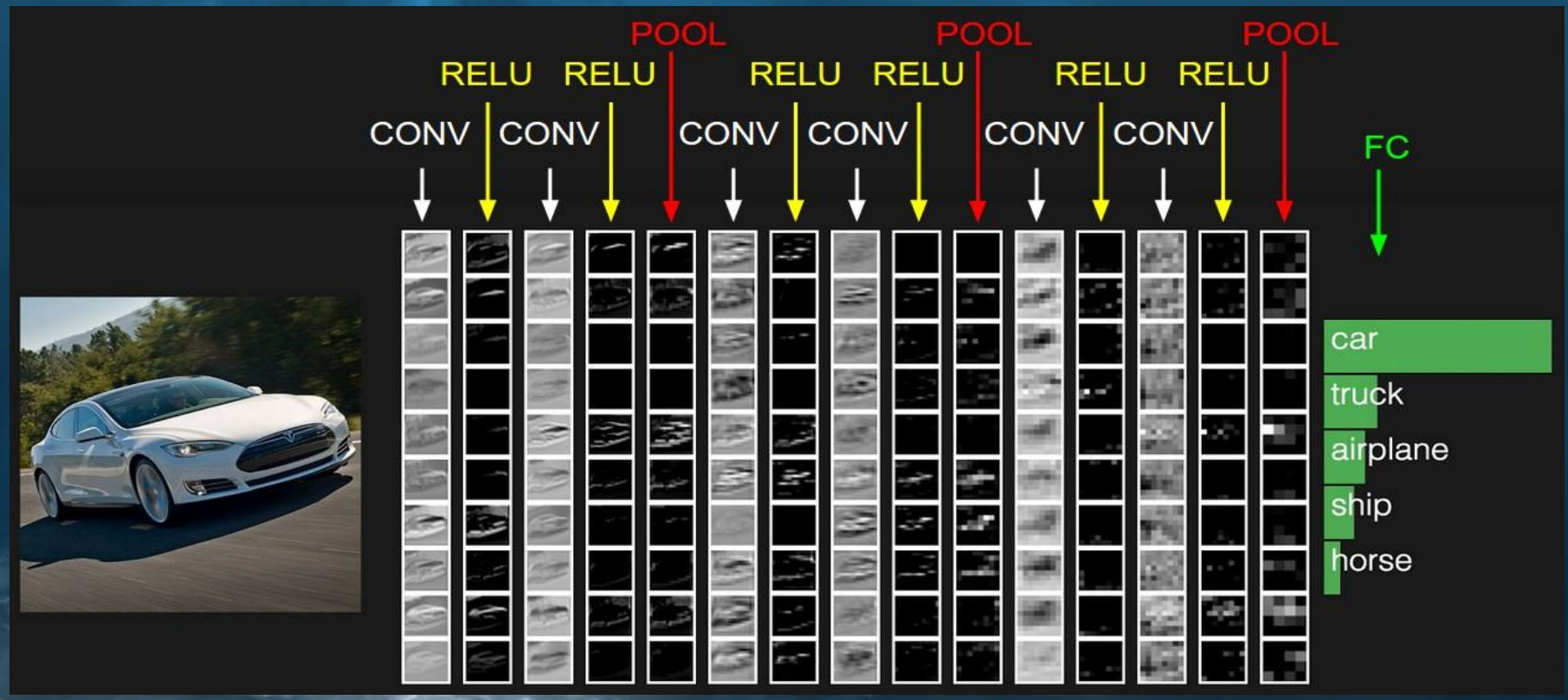
Rysunek przedstawia porównanie typowych i głębokich konwolucyjnych architektur sieci.

Konwolucyjne Sieci Neuronowe

Sieć konwolucyjna jest zwykle sekwencją warstw, które transformują jeden obraz danych do drugiego poprzez funkcję różniczkowalną (w celu umożliwienia wykorzystania algorytmu propagacji wstecznej błędów do dostrojenia parametrów sieci neuronowej.
Sieci konwolucyjne CNNs (ConvNet) zwykle składają się z trzech typowych warstw:

- **Warstwa konwolucyjna zawierająca zbiór adaptowalnych filtrów,** np. [5x5x3]
- **Warstwę łączenia**,
- **W pełni połączoną sieć** implementującą sieci MLP, SVM, czy SoftMax.

[Demonsracja ConvNetJS dla danych CIFAR-10.]



Konwolucyjne Sieci Neuronowe

Przykład sieci konwolucyjnej:

1. **Obraz wejściowy** [32x32x3], gdzie trzeci parametr koduje kolor dla poszczególnych składowych from R, G, i B.
2. **Warstwa konwolucyjna (CONV)** oblicza wyjście dla neuronów podłączonych do lokalnych regionów w obrazie wejściowym, a każda warstwa wyznacza **iloczyn skalarny (*dot product*)** pomiędzy ich wagami i małym regionem w obrazie wejściowym. To może prowadzić do wyników zapisanych w np. 8 warstwach konwolucyjnych o tym samym rozmiarze, co zapisujemy jako [32x32x8] gdzie 8 to ilość filtrów konwolucyjnych.
3. **Warstawa ReLU (RELU)** stosuje funkcję ReLU (określoną jako max(0,x)) z zerowym progiem. Wymiar klastra macierzy dla tej warstwy pozostaje niezmieniony wielkości, czyli [32x32x8] dla powyższego przykładu.
4. **Warstwa łączenia (POOL)** przeprowadza **operację próbkowania przestrzennego** względem wysokości i szerokości (wyznaczając np. maksimum dla MaxPooling), co skutkuje wynikowym wymiarem np. [16x16x8].
5. **Warstwa połączeń każdy-z-każdym (FCNN)** oblicza **wynik klasyfikacji**, co skutkuje wymiarem wyjściowym, np. [1x1x5], dla 5 klas/kategorii, czyli po jednym neuronie wyjściowym dla każdej klasy. Ta warstwa uczona jest metodą uczenia nadzorowanego (zwykle jakąś metodą gradientową, np. propagacją wsteczną błędów) i jest połączona z poprzednią warstwą na zasadzie każdy-z-każdym.

Obliczanie Iloczynu Skalarnego

Iloczyn skalarny (*dot product*) jest operacją algebraiczną, która wykorzystuje dwie tej samej długości sekwencje danych (zwykle wektory) i zwraca pojedynczą liczbę będącą sumą iloczynów poszczególnych odpowiadających sobie wartości w tych sekwencjach:

Załóżmy, że mam dwa wektory:

$$A = [a_1, a_2, \dots, a_n] \quad \text{and} \quad B = [b_1, b_2, \dots, b_n]$$

Produkt skalarny dla tych dwóch wektorów wyznaczamy na podstawie następującej zależności:

$$A \cdot B = \sum_{i=1}^{n} a_i \cdot b_i$$

Uczenie Warstw Konwolucyjnych

Wykorzystujemy zmodyfikowany **algorytm propagacji wstecznej błędów** albo **algorytm stochastycznego spadku gradientu** do adaptacji wag warstw konwolucyjnych.
Błąd delta jest zwykle propagowany wstecz tylko dla zwycięskiego neuronu w danej macierzy warstwy (zwykle stosując **zasadę zwycięzca bierze wszystko *WTA – winner takes all***).
Filtr konwolucyjny (gdyż warstwa konwolucyjna dokonuje takiej adaptacyjnej filtracji wejść) dla j-tego plastra (macierzy) w l-tej konwolucyjnej warstwie wyznacza swoje wyjścia według następującej zależności:

$$z_j^{l+1} = \sum_{i=1}^{n} w_{j,i}^{l} \cdot a_i^{l}$$

Następnie definiujemy **funkcję błędu** zwykle jako:

Błąd całkowity = $\sum$ ½ (docelow prawdopodobieństwo – wyjściowe prawdopodobieństwo) ^{2}

Szczegóły implementacji neuronowych sieci konwolucyjnych można znaleźć w artykule Liu „Implementation of Training Convolutional Neural Networks”

Współdzielone Wagi i Współczynniki Odchyleń

Każdy **plaster (macierz, slice)** w warstwie wykorzystuje te same wagi i współczynniki odchyleń (*biases*) dla wszystkich neuronów. W praktyce każdy neuron wyznacza gradient dla wszystkich wag w trakcie propagacji wstecznej, ale te gradienty zostają dodane w każdym plastrze (macierzy), a aktualizacji wag dokonujemy tylko w pojedynczym zwycięskim plastrze. W taki sposób, wszystkie neurony w pojedynczym plastrze wykorzystują te same wektory wagowe. Warstwa konwolucyjna wykorzystuje te wektory do obliczenia konwolucji wag z obrazu danych wejściowych. Ponieważ ten sam zbiór wag jest wykorzystywany do obliczenia wartości wyjściowych w plastrze, dlatego można takie przekształcenie nazwać **filterem adaptacyjnym** powodującym przekształcenie danych wejściowych na skalar wyjściowy.

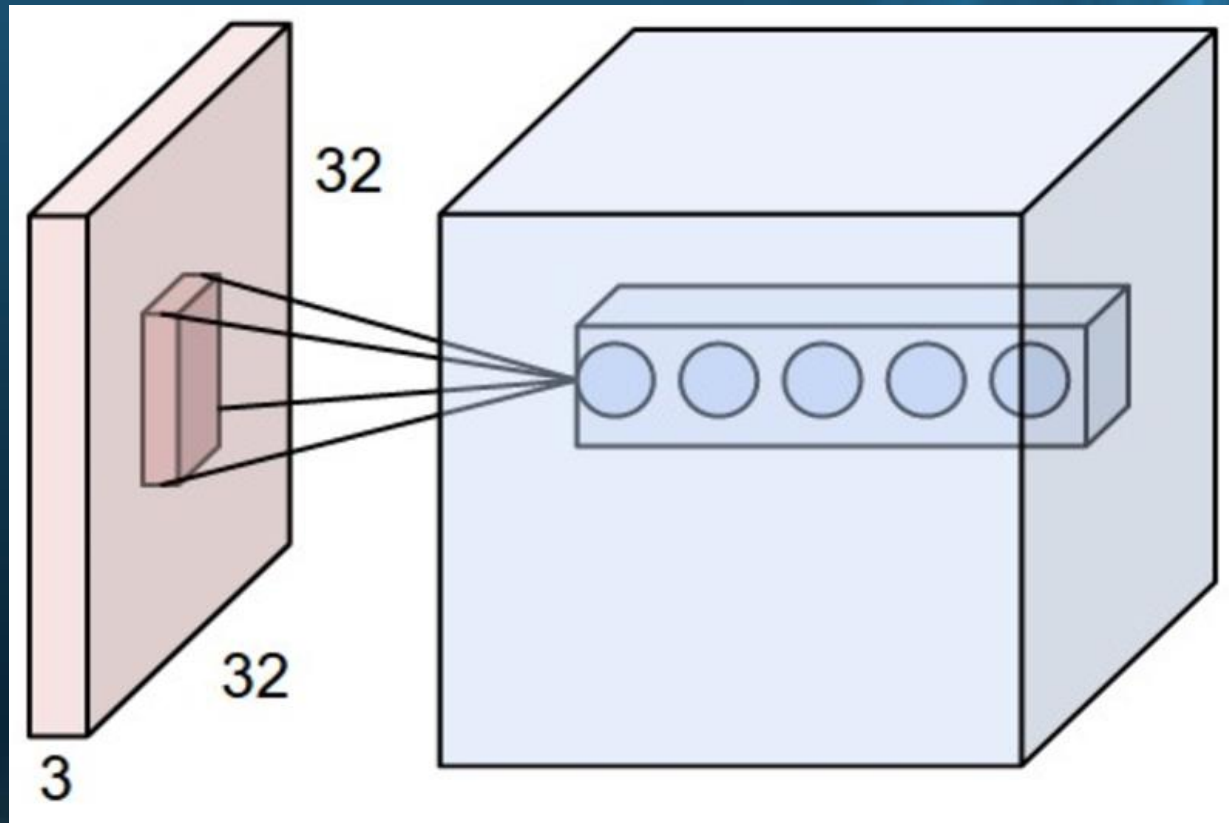
Przykład 96 filtrów [11x11x3] nauczonych przez Krizhevsky at al. Każdy taki filtr posiada 55x55 neuronów w każdym plastrze.

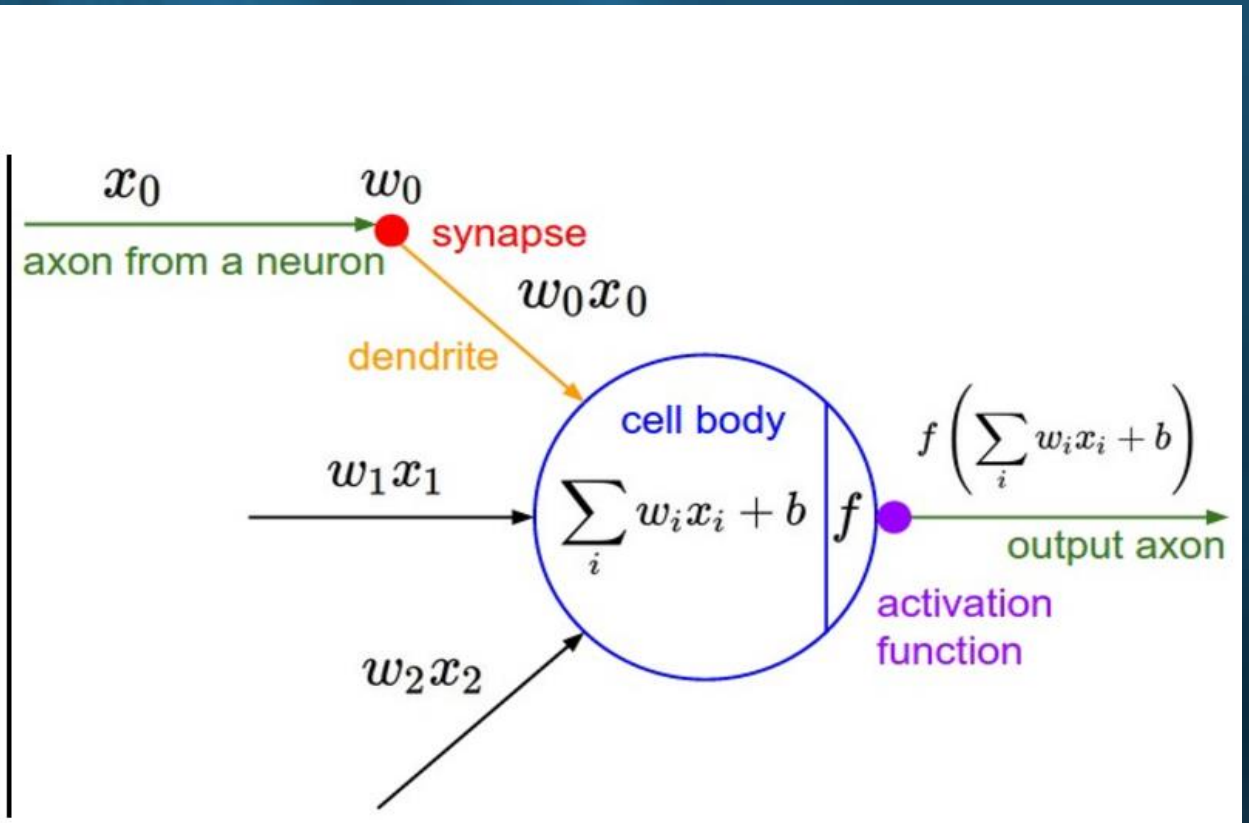
Jeżeli np. wykrywamy linie wertykalne w pewnej lokalizacji obrazu, powinno to być przydatne również w innych lokalizacjach zgodnie z niezmienniczo-natywną strukturą obrazów.

Therefore, **dlatego nie potrzebujemy ponownie nauczyć detekcji linii wertykalnych w każdej lokalizacji warstwy konwolucyjnej** obrazu wyjściowego.

Warstwy konwolucyjne w sieci CNN

W tym przykładzie możemy zauważyć, że mamy różne neurony (5 z nich oblicza iloczyn skalarny ich wag z wydzielonym kawałkiem wejścia) dla całej głębokości (kolejnych plastrów), z których wszystkie są podłączone do tego samego regionu w obrazie wejściowym, gdzie połączenia są ograniczone do lokalnego wycinka obrazu (przestrzeni wejściowej):



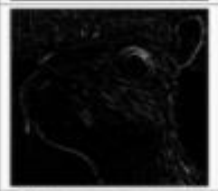


Filtracja adaptacyjna

Warstwa konwolucyjna działa jak **filtr adaptacyjny**, które pozwalają na wyznaczenie wartości w takich macierzach filtracji:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

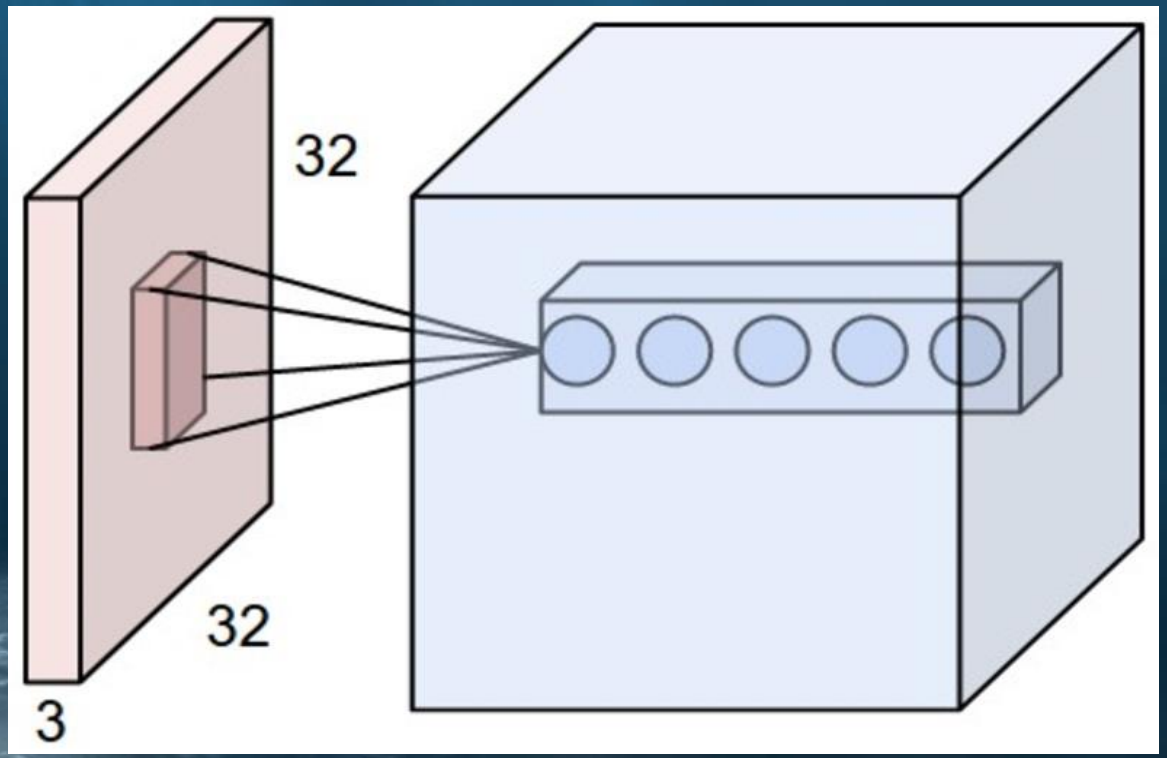
Wykorzystując również inne dobrze znane filtry możemy przekształcać obraz wejściowy tak jak pokazano na rysunku, czyli metodami znanymi z metod przetwarzania obrazów:

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Ilość Filtrów w Warstwach Konwolucyjnych

Głębokość wolumenu wyjściowego jest hiperparametrem, który jest związany z **ilością filtrów**, które chcemy wykorzystać. Każdy filtr uczy się rozpoznawać coś innego w obrazie wejściowym, np. pierwsza warstwa konwolucyjna bierze jako obraz wejściowy jedną składową koloru R, G lub B surowego obrazu, a różne neurony względem wymiaru głębokości tej sieci (które tworzą kolumnę nazywaną **włóknem/filamentem**) mogą aktywować się na skutek prezentacji różnie zorientowanych krawędzi lub kolorowych plam.

Przesuwamy każdy filtr w obrazie wejściowym według zdefiniowanego **parametru kroku (stride):** Kiedy krok jest równy 1 wtedy przesuwamy filtry o jeden piksel, a kiedy 2, wtedy o dwa piksele itp. po całym obrazie wejściowym. To zawsze tworzy mniejszy obraz wyjściowy niż był obraz wejściowy (o mniejszym wymiarze). Czasami wygodne jest dodanie zerowej otoczki do obrazu wejściowego, gdyż wtedy obraz wyjściowy i wejściowy mogą być tego samego rozmiaru.

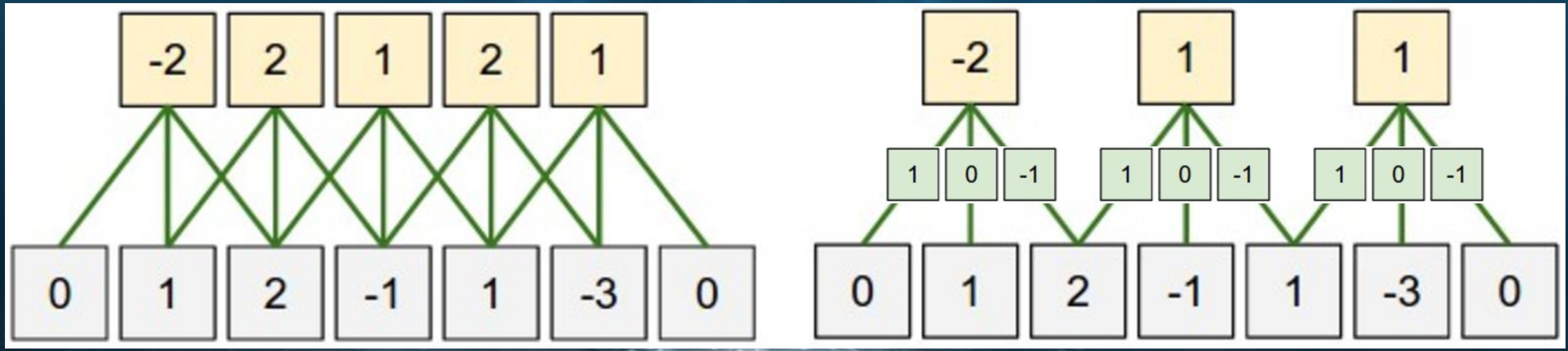


Rozmiar Obrazu Wyjściowego Zależny od Kroku

Możemy obliczyć rozmiar przestrzenny obrazu wyjściowego wg **(W-F+2·P)/S+1** jako funkcję **wielkości obrazu wejściowego W**, **wielkości pola recepcyjnego** warstwy konwolucyjnej neuronów **F**, **kroku S**, i ilości zer tworzących **otoczkę (padding) P** na krawędziach obrazu, np.:

- Jeśli mamy wejście 7x7 i filtr 3x3 z **krokiem 1** i otoczką 0, wtedy otrzymamy wyjście 5x5: (7-3+2·0)/1+1=5
- Jeśli mamy wejście 7x7 i filtr 3x3 z **krokiem 2** i otoczką 0, wtedy otrzymamy wyjście 3x3: (7-3+2·0)/2+1=3

Graficzna prezentacja tego dla jednego wymiaru (szerokości lub wysokości) i takich samych zestawów wag (w zielonych prostokątach) współdzielonych przez wszystkie (żółte) neurony:



Przykład Konwolucji

Załóżmy, że mamy 3 oddzielne tablice przedstawiające 3 plastry obrazu wejściowego w wolumenie 3D o rozmiarze [5x5x3], z których każdy reprezentuje jedną składową koloru obrazu (R, G lub B).

Obraz wejściowy znajduje się we fioletowych tablicach, wagi w czerwonych tablicach, a obraz wyjściowy w zielonych tablicach.

Wykorzystamy następujące parametry w zaprezentowanej warstwie konwolucyjnej:

K = 2 (ilość filtrów),
F = 3 (wielkość filtrów 3x3),
S = 2 (krok),
P = 1 (otoczka), która poszerza obraz wejściowy o otoczkę z zer (na szaro).

Stąd wymiar obrazu wyjściowego jest: $(5 - 3 + 2 \cdot 1) / 2 + 1 = 3$

W przedstawionej na następnych slajdach wizualizacji dokonujemy iteracji po kolejnych polach obrazu wyjściowego, pokazując jak wartości wyjściowe są obliczane poprzez przemnożenie wag przez wybrany obszar danych (określony przez wielkość filtra) z obrazu wejściowego, przesuwając wynik względem odchylenia (bias).

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Przykład Konwolucji

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Porównanie Sztucznych i Konwolucyjnych Sieci Neuronowych

Typowa **sztuczna sieć neuronowa** wykorzystująca połączenia neuronów na zasadzie każdy-z-każdym może w łatwy sposób ulegać **przeuczeniu (overfit)** dla średniej wielkości i dużych obrazów, gdyż np. obraz o wielkości 100x100x3 (gdzie 100x100 to wielkość obrazu w pikselach, a 3 to ilość kanałów kolorów R, G i B) ponieważ uzyskujemy potężną ilość parametrów wolnych, które muszą zostać wytrenowane: (100*100*3 = 30000) w stosunku do ilości uczonych obrazów (wzorców, obiektów). Ponadto taka reprezentacja modelu jest nieoszczędna pamięciowo i droga obliczeniowo!

W **strukturze konwolucyjnej CNN**, każdy neuron połączony jest tylko do lokalnego regionu obrazu wejściowego. **Lokalne obszary (regiony)** zdefiniowane są przez szerokość i wysokość, zaś głębokość przebiega przez całą warstwę obrazu wejściowego. Zakres połączeń wzdłuż osi głębokości CNN jest zawsze równy głębokości obrazu wejściowego (warstwy poprzedniej).

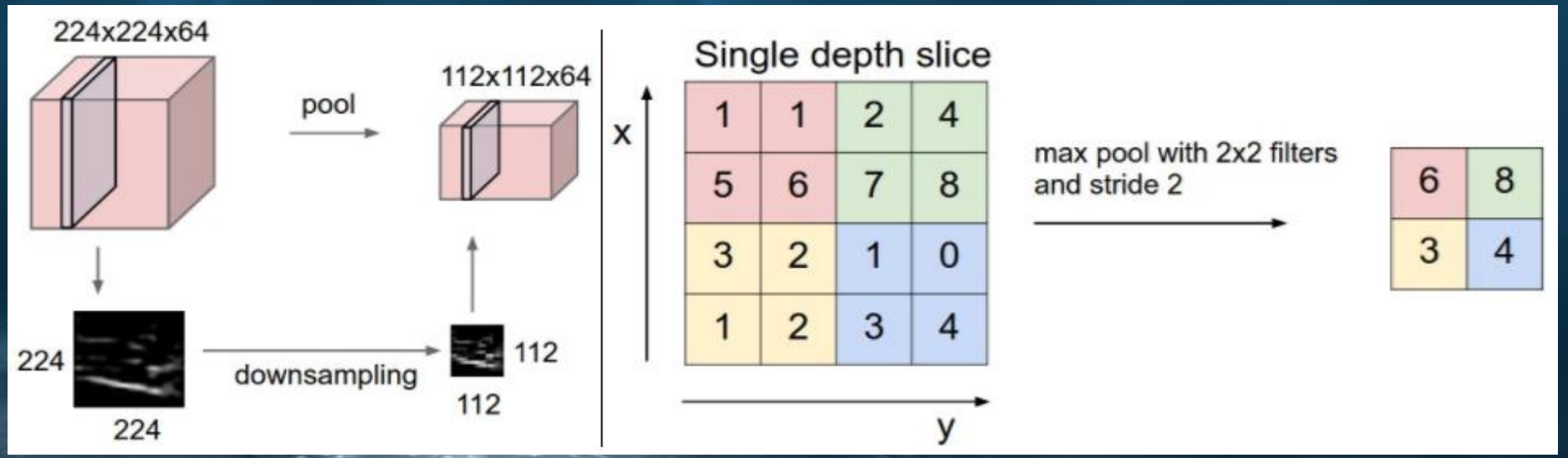
Ten ograniczony parametr hiperpołączeń nazywany jest **polem recepcyjnym**, np.:

Załóżmy, że obraz wejściowy ma rozmiar [32x32x**3**]. Jeśli pole recepcyjne ma rozmiar 5x5, wtedy każdy neuron warstwy konwolucyjnej będzie posiadał połączenia do [5x5x**3**] regionów obrazu wejściowego, czyli (5*5*3=75 wag + 1 parameter odchylenia (bias)). **Głębokość** obrazu wejściowego tutaj będzie równa **3**.

Warstwa Łącząca i Operacja Maksimum

Bardzo często w sieciach konwolucyjnych okresowo wstawiamy **warstwę łączącą (pooling)** pomiędzy kolejnymi **warstwami konwolucyjnymi**. Jej główną funkcją jest **stopniowa redukcja wymiaru i ilości parametrów, jak również nakładu obliczeniowego**. Pozwala również kontrolować **przeuczenie się sieci** ponieważ mniejsza ilość parametrów rzadziej prowadzi do przeuczenia. Warstwa łączenia typowo wykorzystuje **operację Maksimum** niezależnie dla każdego plastra wejściowego **i przeskalowuje go przestrzennie (zmniejszając jego rozmiar)**.

Najczęstszą formą łączenia jest wykorzystanie filtrów o wielkości 2x2 z krokiem 2, próbkujących każdy plaster wejściowy i redukując jego rozmiar o czterokrotnie (każdy wymiar o połowę), odrzucając 75% aktywacji, ponieważ zawsze wybieramy jedno maksimum z 4 aktywacji w regionie wejściowym 2x2 w każdym plastrze. Głębokość jest zaś zachowana.

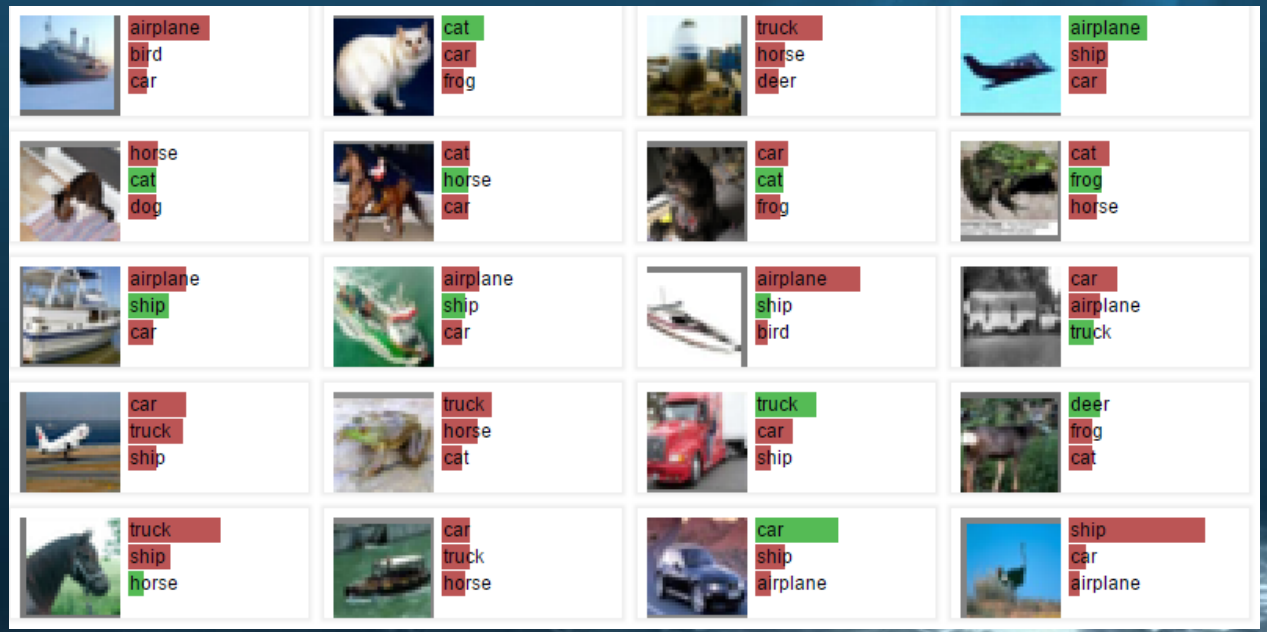


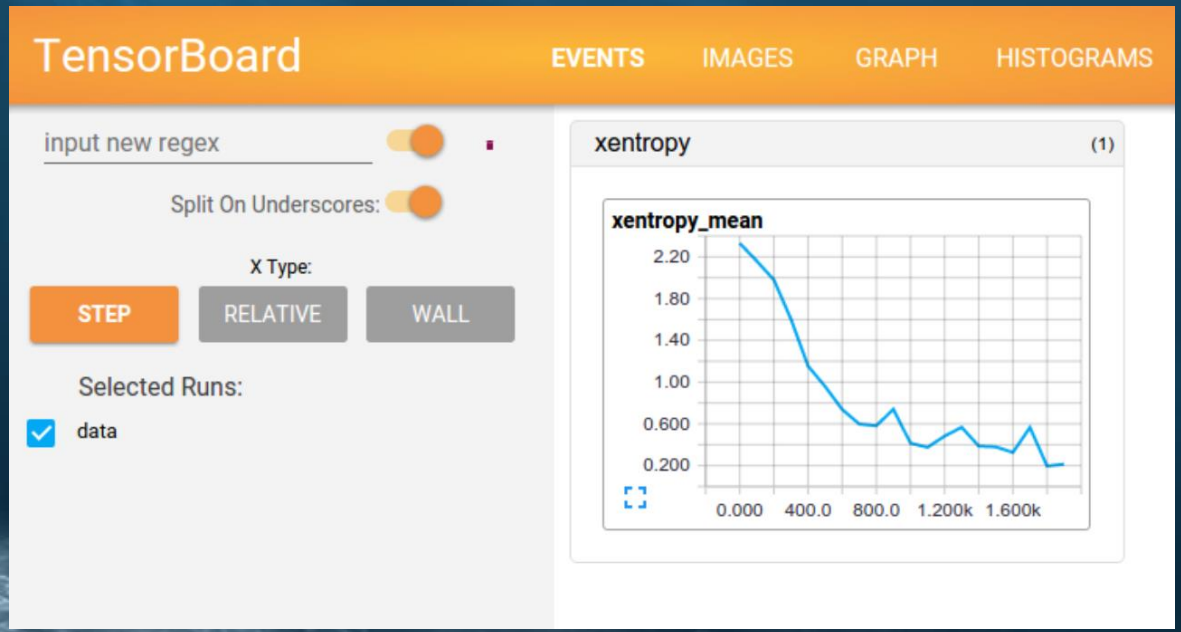
Przykłady Architektur Konwolucyjnych

Przykładami architektur konwolucyjnych CNN są: AlexNet, GoogLeNet, LeNet, ResNet, VGGNet.

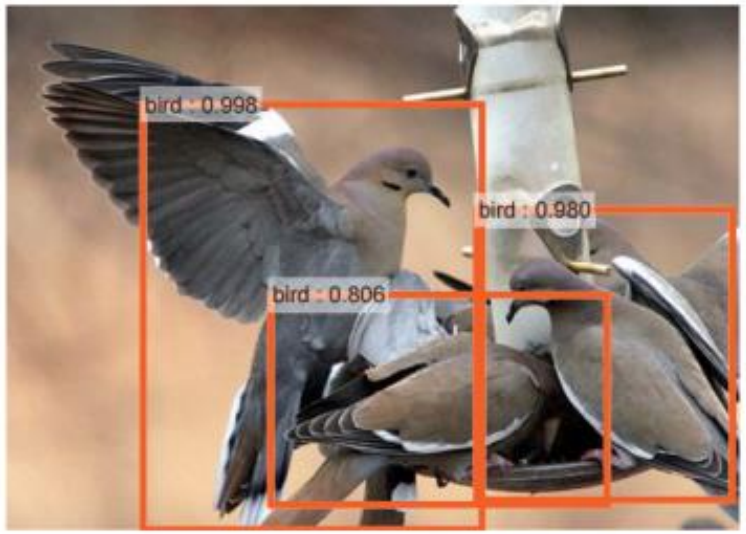
Bechmarkowe zbiory uczące do porównań: MNIST, CIFAR-10, CIFAR-100, STL-10, and SVHN.

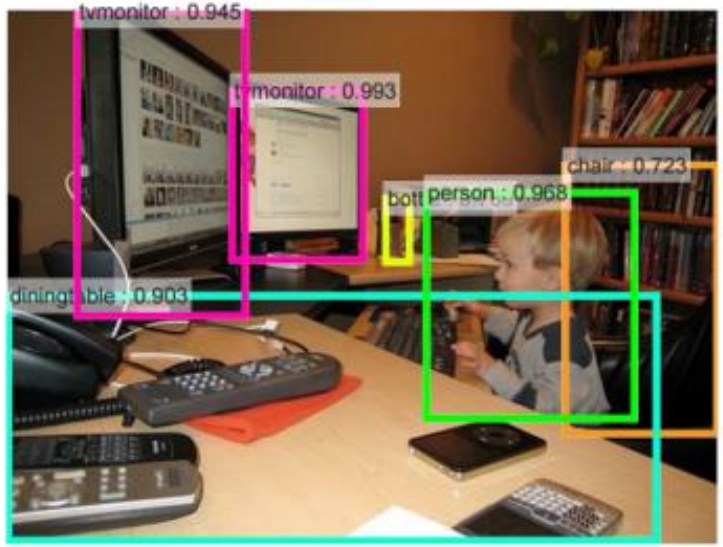
Narzędzia do pracy z sieciami CNN: Theano, PyLearn2, Lasagne, Caffe, Torch7, Deeplearning4j, TensorFlow.

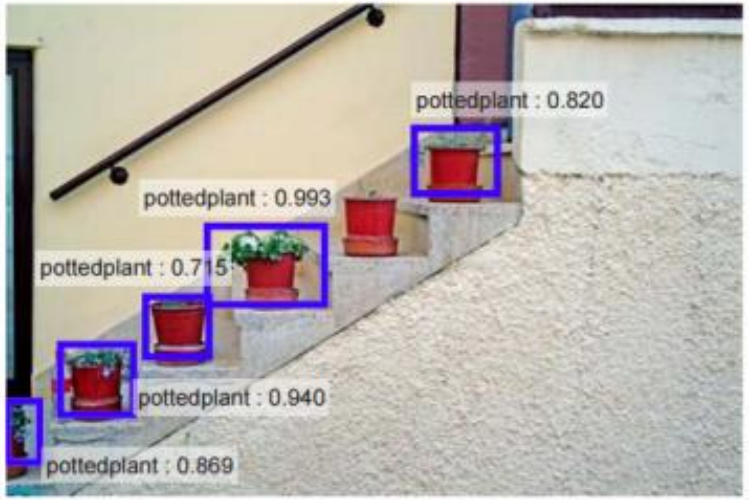


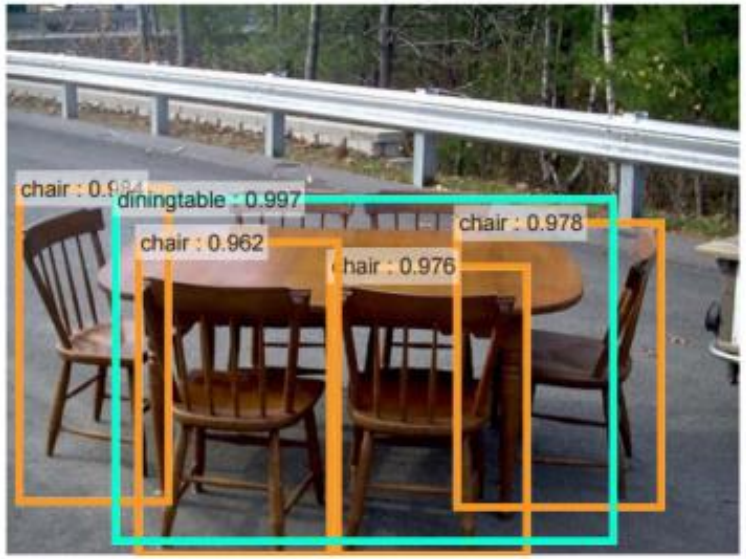


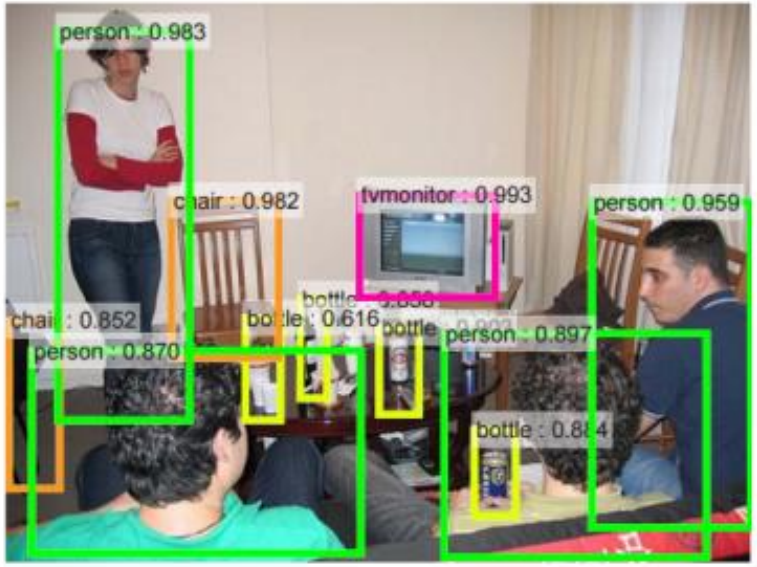
Przykłady Działania Sieci Konwolucyjnych



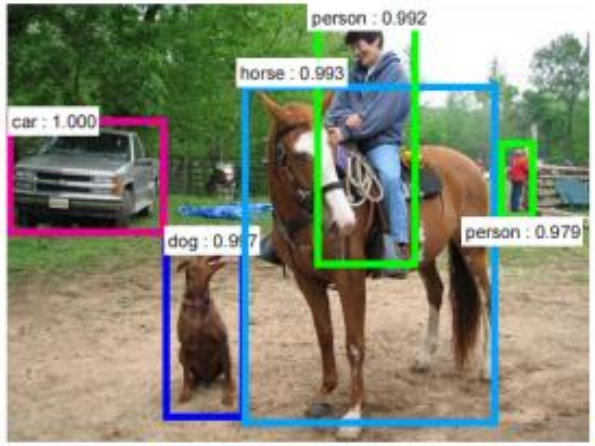


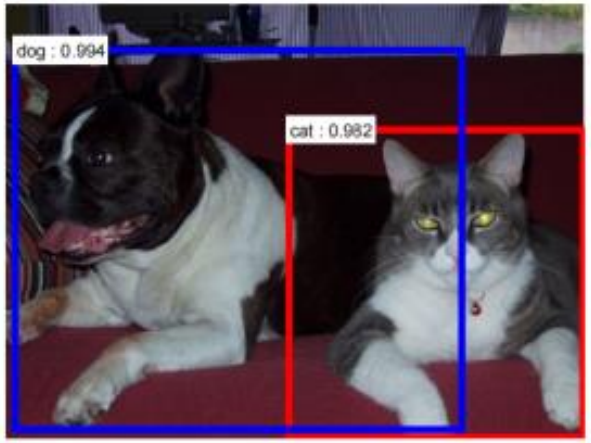


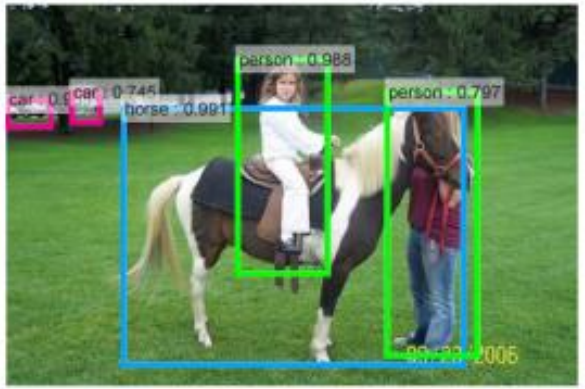


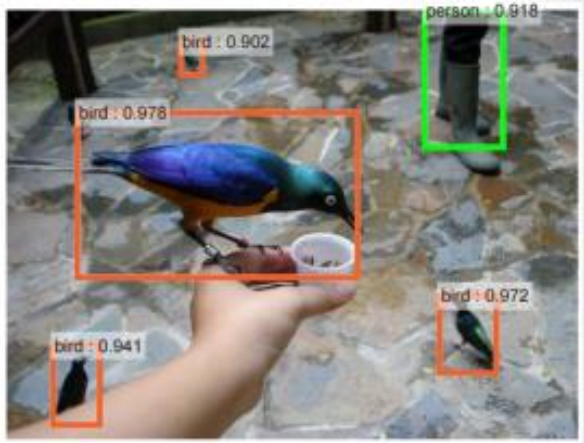


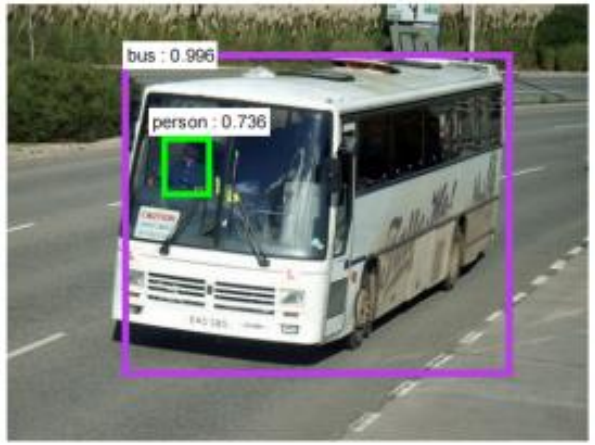
Przykłady Działania Sieci Konwolucyjnych

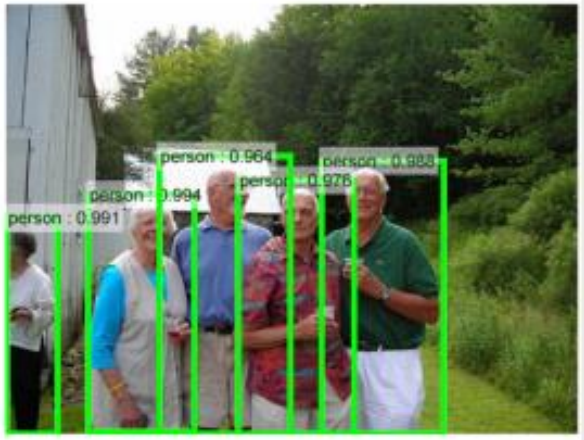












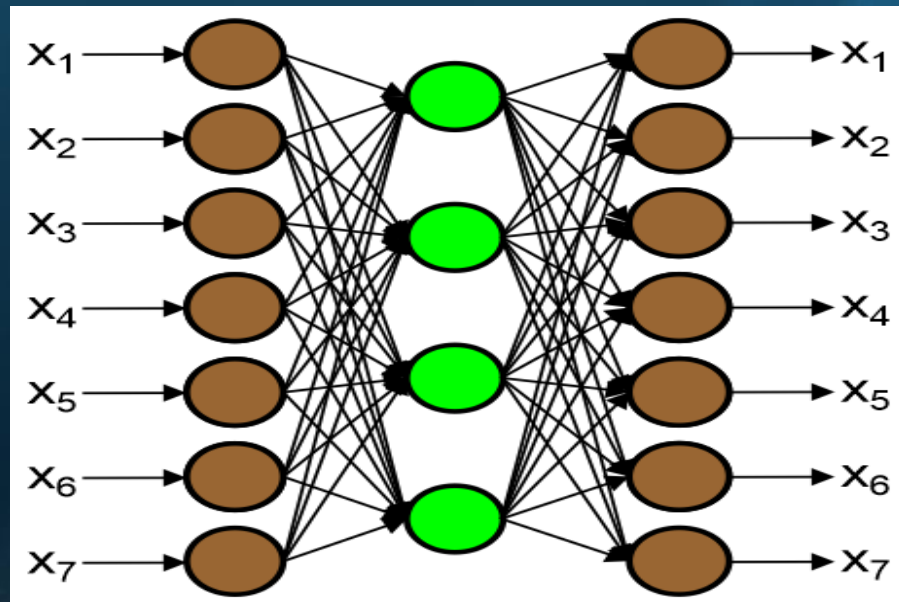
Przykład Reprezentacji Obrazu po jego Odtworzeniu Konwolucyjnym

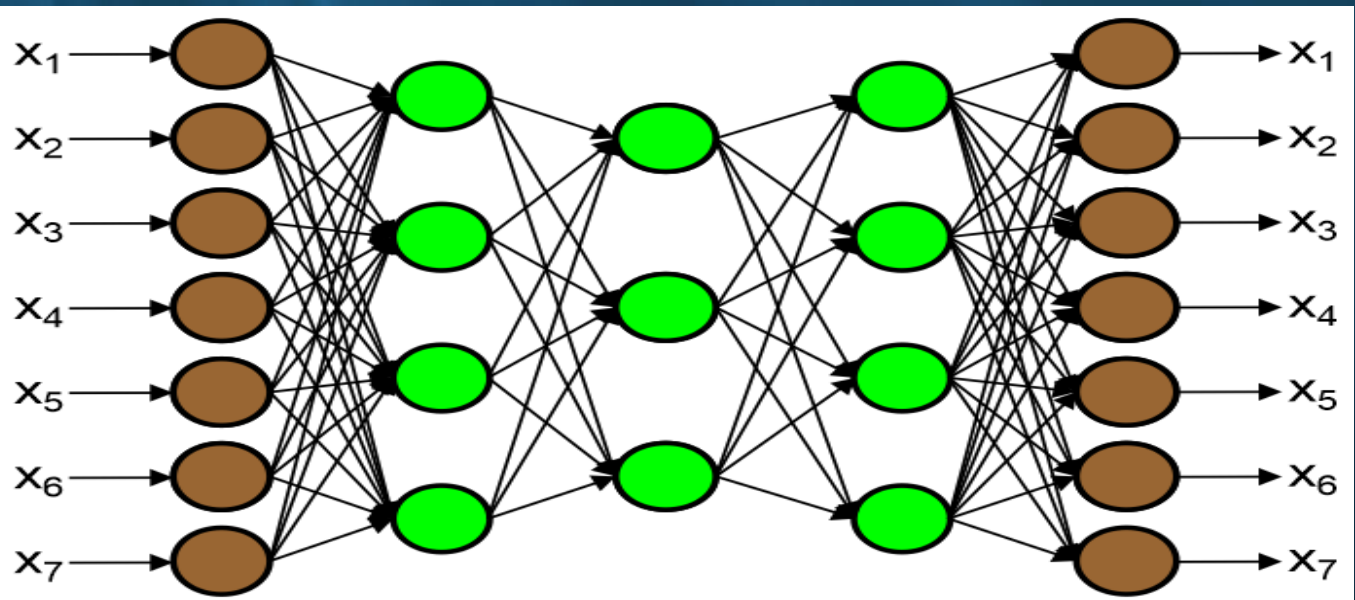
Input

Feature Map

Autoenkoder – Autoasocjator

Głębokie sieci neuronowe często wykorzystują **autoenkodery**, które są rodzajem sztucznych sieci neuronowych wykorzystywanych do **uczenia nienadzorowanego** i **efektywnego kodowania**. Wykorzystujemy tutaj zbiór wzorców wejściowych do nauczenia identycznych wyjść sieci. **Podstawowym celem** takiego uczenia jest odnalezienie zredukowanej liczby neuronów w warstwie ukrytej (warstwach ukrytych) w stosunku do wymiaru danych wejściowych, która pozwala na odtworzeniu na wyjściu wzorca wejściowego **bez zniekształceń**. Pozwala to na **redukcję wymiaru** i taką reprezentację zbioru wzorców wejściowych, która wymusza tworzenie się reprezentacji cech:



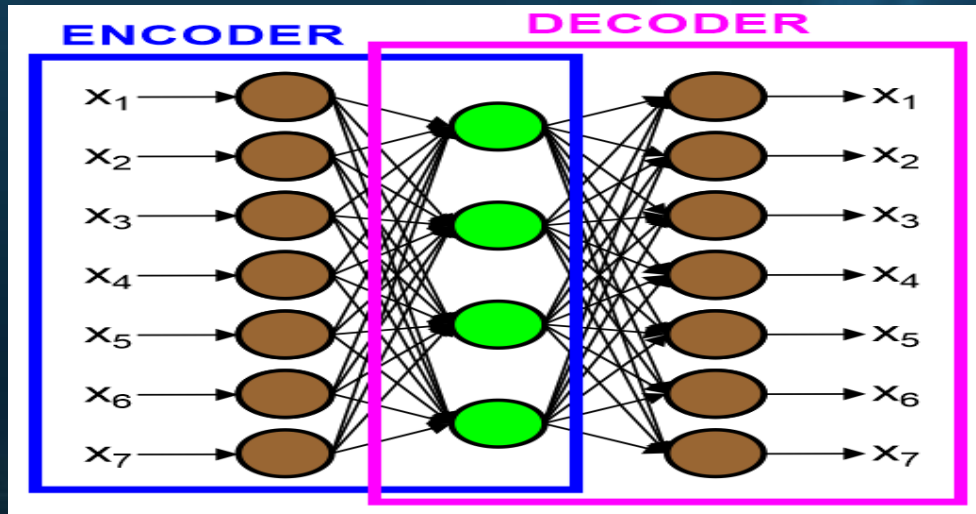


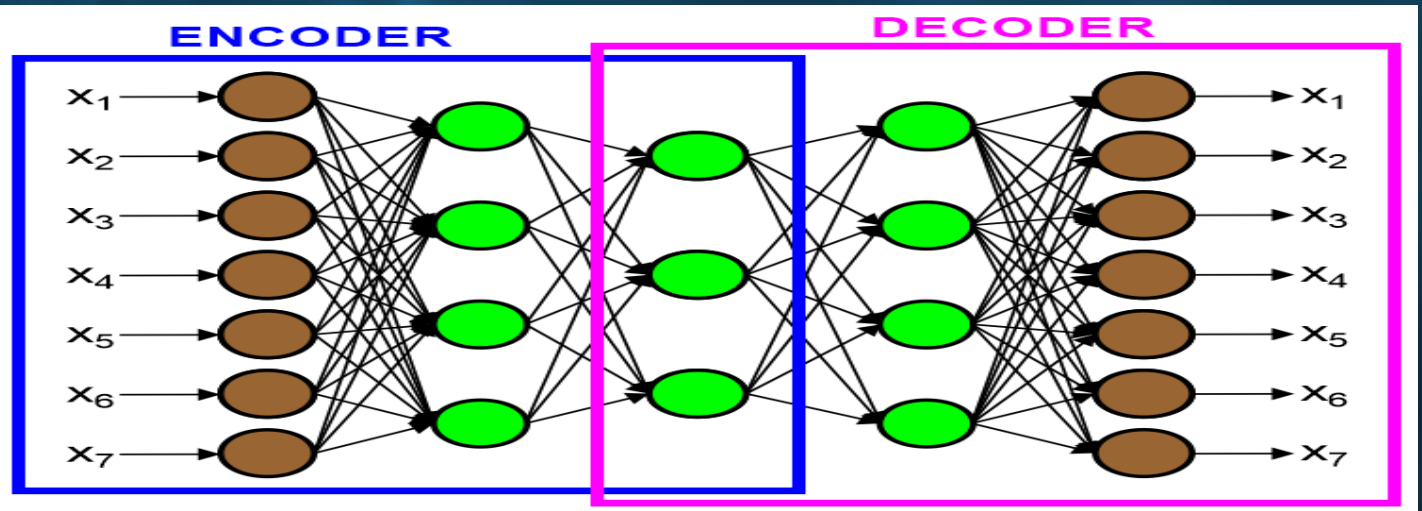
Uczenie Autoenkoderów

Autoenkodery mogą być uczone metodami gradientowymi wykorzystywanymi w uczeniu nadzorowanym ponieważ wyjścia są identyczne z wejściami sieci, więc z łatwością możemy wyliczyć błąd wyjściowy, a następnie wykorzystać np. metodę propagacji wstecznej błędu do adaptacji wag sieci autoenkodera.

Zasadniczym pytaniem jest, po co uczyć sieć autoasocjacji - czyli projekcji identycznościowej?

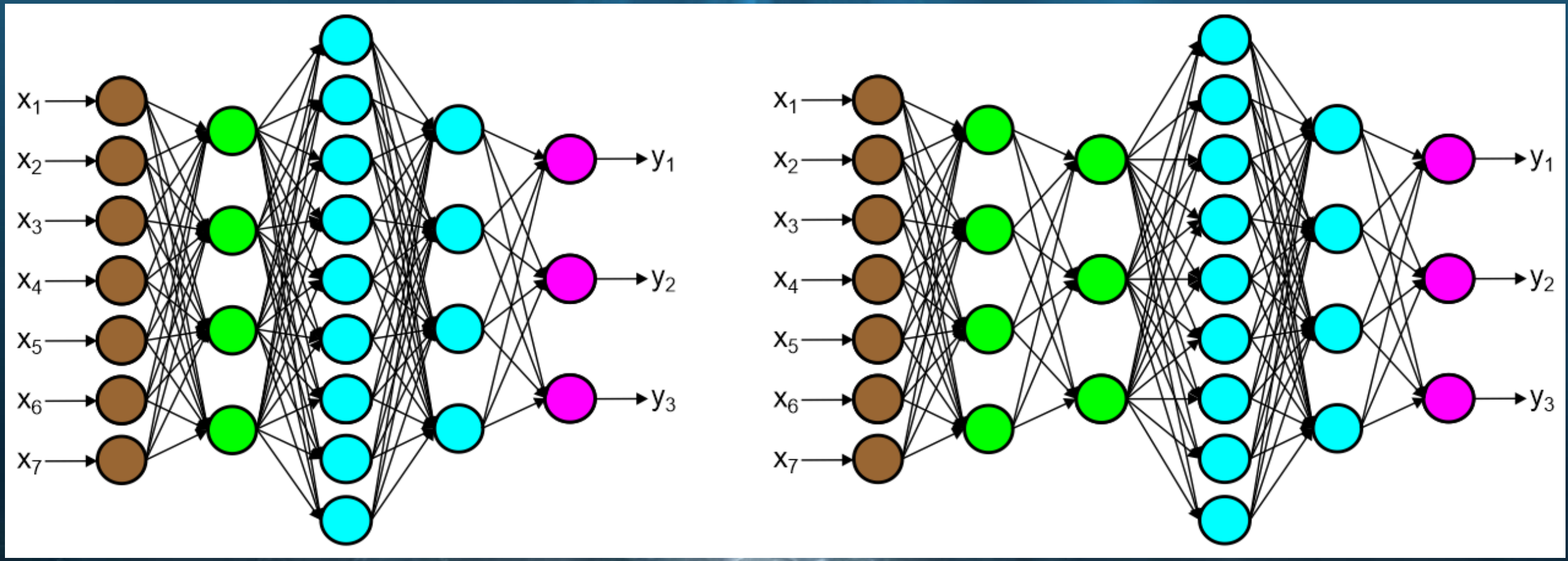
Głównym powodem są neurony w warstwie ukrytej, które uczą się oszczędnie reprezentować dane wejściowe tak, by je móc znowu odtworzyć. Żeby to było możliwe, muszą reprezentować pewne wspólne i podobne cechy uczonych wzorców, a dzięki temu wymuszamy na tej sieci tworzenie reprezentacji cech wzorców uczących. Dochodzi więc do neuronowej kompresji reprezentacji wzorców.





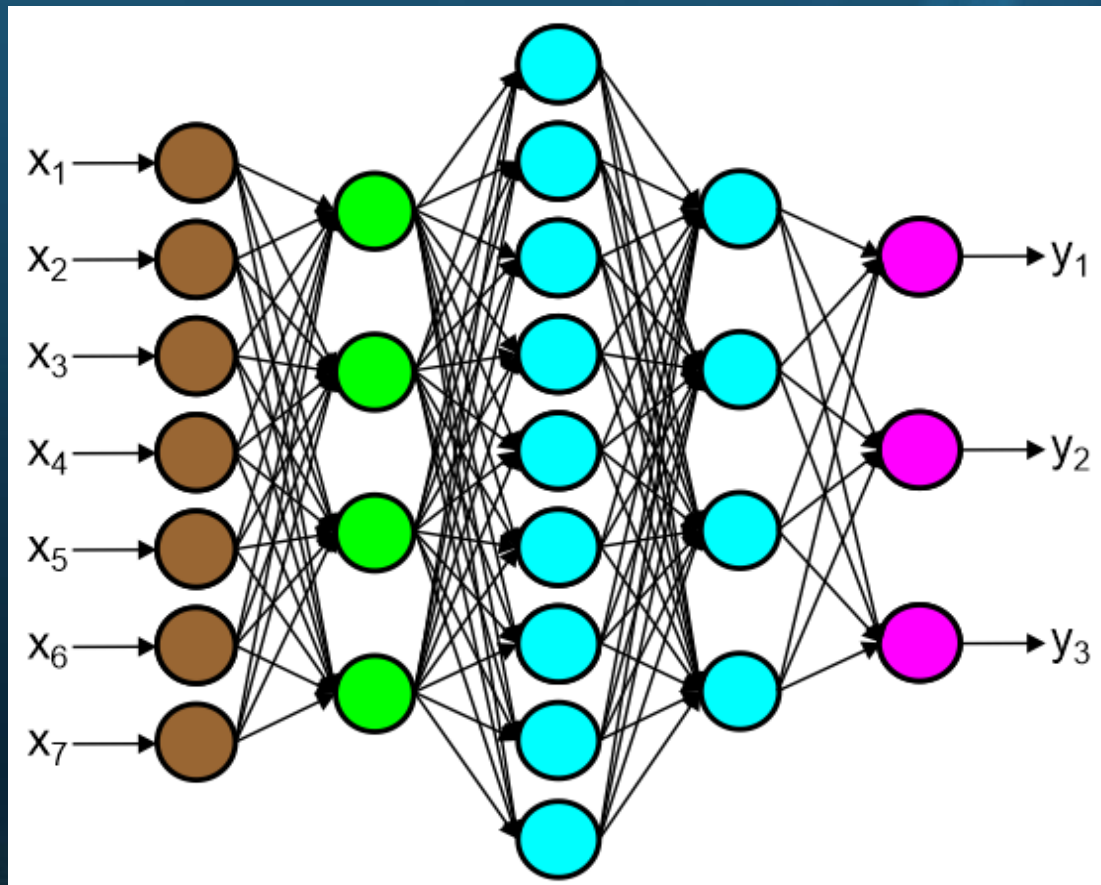
Wykorzystanie Autoenkoderów w Strukturach Sieci Głębokich

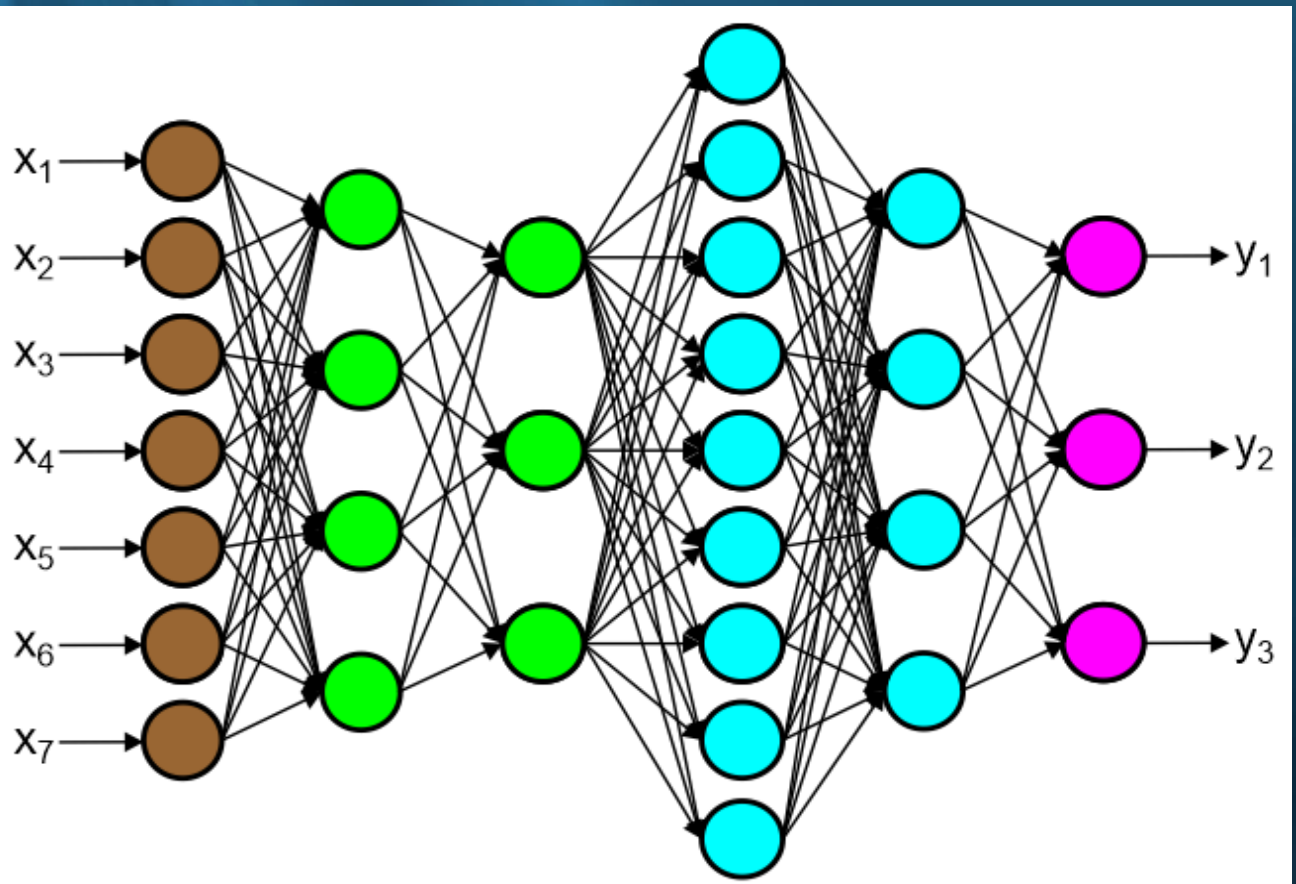
Autoenkodery są często wykorzystywane w sieciach głębokich jako warstwy nie wymagające uczenia nadzorowanego w celu określenia zasadniczych cech wzorców uczących, które mogą być dalej wykorzystane do klasyfikacji poprzez odcięcie ostatniej warstwy autoenkoderów:



Wykorzystanie Autoenkoderów w Strukturach Sieci Głębokich

Autoenkodery są najpierw uczone, a następnie odcinamy ostatnie warstwy i wykorzystujemy pozostałą część sieci (warstwę wejściową i warstwy ukryte) do dalszej budowy sieci głębokiej, która w końcowych warstwach uczona jest w sposób nadzorowany i może również dostrajać warstwy enkoderów:

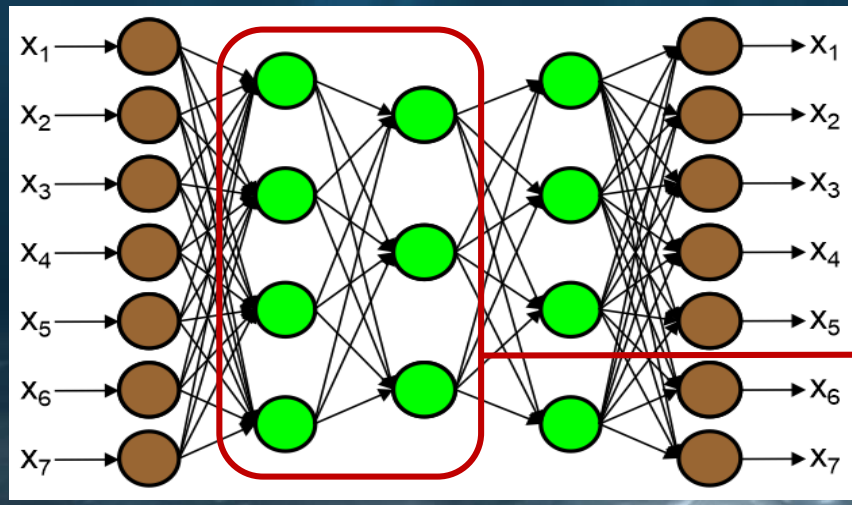


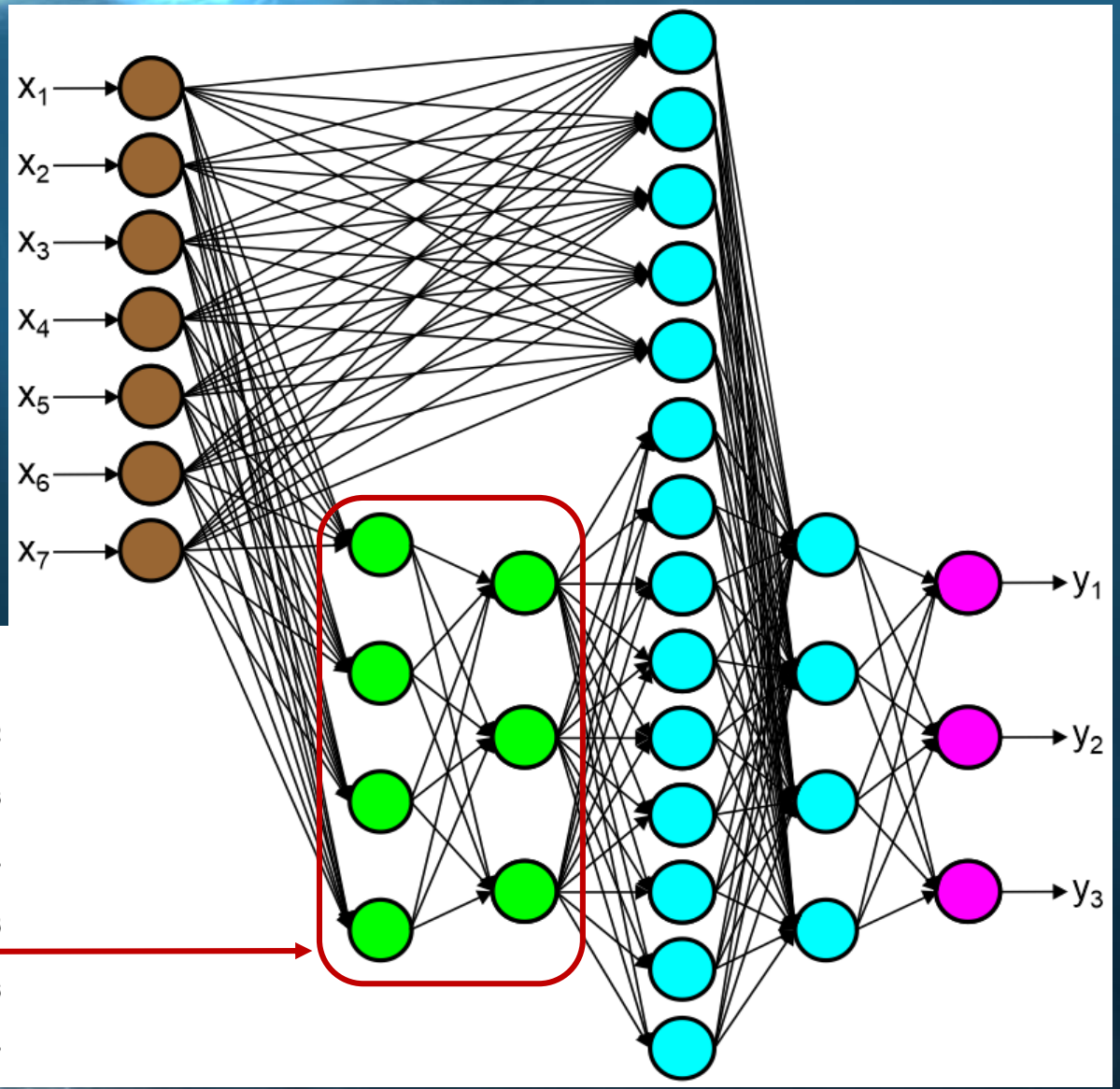


Hybrydowe Architektury Głębokie Zawierające Autoenkodery

Autoenkodery mogą być wykorzystane w różnych hybrydowych architekturach sieci neuronowych do ekstrakcji cech, podobnie jak pierwsze warstwy sieci konwolucyjnych CNN.

Różne zadania **klasyfikacji i rozpoznawania** wymagają jakościowej **ekstrakcji cech (mocnych, niezmniennych i dobrze dyskryminujących)**, gdyż dalsze wyniki klasyfikacji czy rozpoznawania są zależne od jakości tych cech!





BIBLIOGRAFIA I LITERATURA

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press book, 2016.
- DeepMind Video - How it works?
- Convolutional Neural Networks for Visual Recognition
- Convolutional Neural Network (Stanford)
- ImageNet Classification with Deep CNNs
- Intuitive explanation of ConvNets
- Image Style Transfer Using Convolutional Neural Networks, Leon A. Gatys, Alexander S. Ecker, Matthias Bethge.
- Visualizing and Understanding Convolutional Networks, Zeiler, Fergus, ECCV 2014
- Pattern Recognition and Machine Learning (Information Science and Statistics), Bishop, Christopher M., 2006
- Neural Networks and Deep Learning, Michale A. Nielsen, Determination Press, 2015
- An Intuitive Explanation of Convolutional Neural Networks
- Convolutional Neural Networks (LeNet)
- Tianyi Liu, Shuangsang Fang, Yuehui Zhao, Peng Wang, Jun Zhang Implementation of Training Convolutional Neural Networks
- Neural Networks and Deep Learning
- Unsupervised Feature Learning and Deep Learning
- Theano Convolution Arithmetic Tutorial
- Backpropagation In Convolutional Neural Networks