

Sieci konwolucyjne

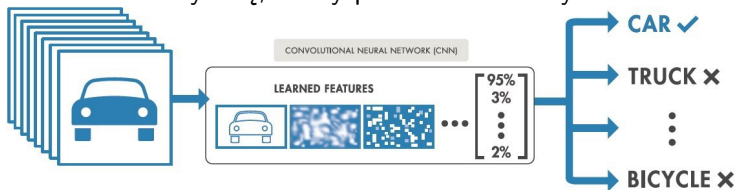
prof. UAM dr hab. Tomasz Górecki

tomasz.gorecki@amu.edu.pl

Uniwersytet im. Adama Mickiewicza w Poznaniu
Wydział Matematyki i Informatyki



Sieci neuronowe znajdują zastosowanie w pracy z danymi w postaci obrazów, mianowicie mogą zostać wykorzystane do klasyfikacji obrazów. Wyjściem takiej sieci byłby rozkład prawdopodobieństwa wykrycia obiektu z danej kategorii na obrazie. Powstaje jednak pytanie dotyczące kwestii odpowiedniego uformowania wejścia sieci. Według podanej wcześniej definicji neuronów i sieci neuronowych wejściem sieci ma być jednokolumnowy wektor liczb rzeczywistych. Niestety, cyfrowa reprezentacja zdjęć jest macierzą pikseli. Co więcej, rozpatrując paletę barw niemonochromatyczną, każdy piksel ma 3 kolory.



Macierz można rozwinąć do jednego, długiego wektora. To podejście ma jednak szereg wad. Po pierwsze: ilość wag takiej sieci na warstwie poprzedzającej wyjściową byłaby bardzo duża. Dla obrazu o rozmiarze 224×224 o 3-bitowej głębi koloru, warstwa wyjściowa miałaby $224 \cdot 224 \cdot 3 = 150\,528$ neuronów. Kolejna warstwa, przyjmując zmniejszenie liczby neuronów nawet o połowę (75 264 neurony) miałaby $75\,264 \cdot (150\,528 + 1) = 11\,329\,414\,656$ wag. Zapisując wagi w postaci zmiennoprzecinkowej na 8 bitach osiągamy rozmiar około 10,5 GB na tylko jednej warstwie. Sieć o takim rozmiarze jest poza możliwościami implementacji na urządzeniach mobilnych. Kolejną wadą tego podejścia jest fakt, że ucząc sieć na sztywnej reprezentacji obrazu, sieć nie nauczy się rozpoznawania obiektów na obrazie umieszczonych w innych miejscach na obrazie niż spotkane w zbiorze uczącym.

Tensor

Uogólnienie pojęcia wektora; wielkość (tablica liczb), której własności pozostają identyczne niezależnie od wybranego układu współrzędnych.

A tensor is an N-dimensional array of data



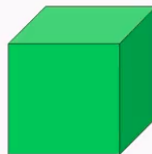
Rank 0
Tensor
scalar



Rank 1
Tensor
vector



Rank 2
Tensor
matrix



Rank 3
Tensor



Rank 4
Tensor

Warstwa konwolucyjna

W celu uniezależnienia pozycji i rotacji rozpoznawanych obiektów od procesu uczenia, należy skonstruować najgłębsze warstwy sieci tak, aby ekstrahowały cechy charakterystyczne obrazu bez względu na ich pozycję. W tym celu stosuje się **warstwy splotowe** (ang. *convolutional*) zwane również **konwolucyjnymi**.

Dla konwolucyjnych sieci neuronowych strukturę danych możemy przyjąć jako trójwymiarowe bloki (tensory). Każdy z wymiarów stanowią odpowiednio wysokość obrazu, jego szerokość oraz liczba warstw – dla obrazu w skali RGB są to trzy warstwy. Dzięki operacji konwolucji nie tylko uzyskujemy interesujące nas szczegóły obrazu ale także redukujemy objętość danych wejściowych.

Operator konwolucji







Konwolucja, zwana również **splotem**, to matematyczna operacja wykonywana na dwóch funkcjach. W wyniku splotu otrzymujemy trzecią funkcję, która jest nazywana tak samo jak operacja – splot.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau.$$

W odniesieniu do konwolucyjnych sieci neuronowych pierwszy argument (funkcja f) odnosi się do wektora danych wejściowych, natomiast drugi argument (funkcja g) jest tak zwanym **filtrem** bądź **jądrem** (*ang. kernel*). Przy tak zdefiniowanych argumentach operację wykonuje się na danych dyskretnych, stąd można zdefiniować konwolucję dla takich danych

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m] = \sum_{m=-\infty}^{\infty} f[n - m]g[m].$$

Idea warstwy konwolucyjnej opiera się na wybieraniu kolejnych małych obszarów danych wejściowych. Każdy filtr stanowi wagę połączenia w tej sieci neuronowej, co oznacza, że każdy neuron jest połączony do niewielkiego obszaru obrazu.

Operation	Kernel ω	Image result $g(x,y)$
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)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

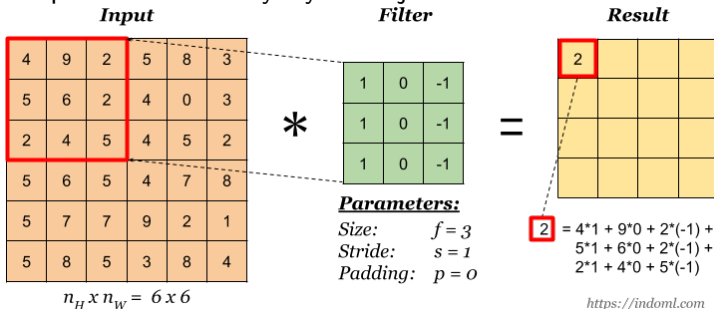
To jakich wymiarów będzie wynik splotu jest ściśle powiązane z danymi przekazywanymi na wejściu sieci oraz opisane przez trzy główne parametry: **głębokość** (*ang. depth*), **krok** – przesunięcie filtra (*ang. stride*) oraz **uzupełnienie obrazu** (*ang. padding*). Głębokość jest ściśle określona przez liczbę zastosowanych filtrów. Wynikiem splotu obszaru obrazu i każdego filtra jest dwuwymiarowa macierz. Gdy mamy kilka filtrów, otrzymamy tyle samo macierzy wynikowych, które zestawiając ze sobą utworzą obraz o głębokości równej ilości zastosowanych filtrów. Ustalenie kroku o jaki przesuwamy filtr po obrazie ma wpływ na wielkość obrazu wynikowego. Im większy krok tym mniejszą macierz otrzymamy.

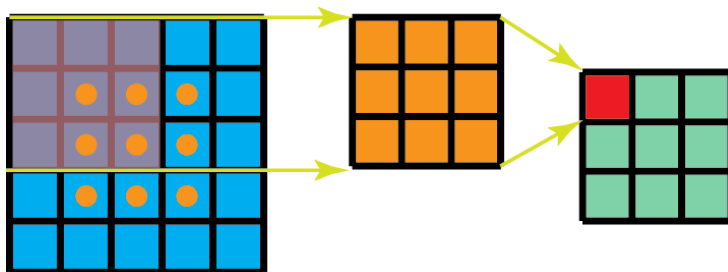
Trzeci parametr – padding, określa uzupełnienie obszaru obrazu wejściowego o dodatkowe piksele. Takie działanie podyktowane jest potrzebą zwiększenia udziału pikseli znajdujących się na krawędzi obrazu w konwolucji. Ze względu na charakter operacji splotu, piksele w centrum obrazu są więcej razy uwzględniane w obliczeniach. Uzupełnienie macierzy wejściowej o dodatkowe piksele zwiększa udział obszarów brzegowych. Rozmiar macierzy wynikowej można obliczyć z następującej zależności

$$\frac{n + 2p - f}{s} + 1,$$

gdzie n oznacza wysokość lub szerokość obrazu wejściowego, p – liczbę pikseli uzupełniających obraz (padding), f rozmiar filtru (wysokość lub szerokość) oraz s – krok, z jakim przesuwamy filtr po obrazie (stride).

Rozważamy dwie macierze o wymiarze 3×3 , elementy o tych samych indeksach mnożymy i wszystkie wyniki sumujemy. Tak otrzymany wynik umieszcza się w macierzy wynikowej. Operację powtarzamy przesuwając filtr wzdłuż macierzy wejściowej o jedną pozycję aż do pokrycia całego obrazu i zapełnienia macierzy wynikowej.



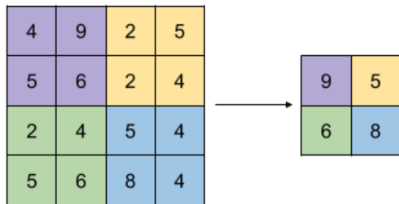


Konwolucje pozwalają na ekstrakcję prostych cech w początkowych warstwach sieci, np. rozpoznają krawędzie o różnej orientacji lub różnokolorowe plamy, a następnie plastry, koła w kolejnych warstwach. Każda warstwa konwolucyjna zawiera cały zbiór filtrów, a każdy z nich generuje osobną mapę aktywacji 2D. **Konwolucyjne sieci neuronowe** (ang. *Convolutional Neural Network – CNN*) składają się z jednej lub wielu warstw konwolucyjnych. Sieci konwolucyjne są łatwe do uczenia, gdyż zawierają mniej parametrów 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).

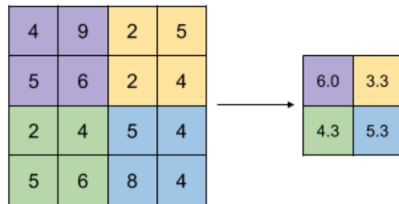
Klasyczną konwulucję przeplata się często warstwami ekstrahującymi najistotniejsze elementy. Zasada działania nie zmienia się w stosunku do warstwy konwulucyjnej: wzdłuż wierszy macierzy przesuwane jest okno o wymiarze n , biorące pod uwagę n^2 elementów. Tym razem jednak nie używamy splotu, a innej funkcji, która dla wszystkich elementów pokrytych przez maskę obliczy skalar. Funkcja ta powinna być bardzo szybka do obliczenia i nie powinna zawierać wyuczalnych parametrów. Przykładem takich funkcji jest maksimum, średnia, mediana. Warstwy takie nazywa się **warstwami łączącymi** lub **zbierającymi** (*ang. pooling*). Podobnie jak przy zwykłej konwulucji, zmniejszany jest rozmiar wynikowej macierzy. Działanie takiej warstwy w praktyce powoduje wyłuskanie z głębszych warstw tylko najistotniejszych właściwości. W tym przypadku następuje utrata informacji – np. dla funkcji maksimum, wszystkie wartości, prócz najwyższej, zostaną pominięte.

Całość procesu odbywa się przy pomocy filtru o wymiarach 2×2 i przesunięciu równym również 2. Pozwala to uniknąć wielokrotnego uwzględniania danych z tych samych obszarów. Operacji dokonujemy na każdej warstwie wzdłuż głębokości. Efektem ma być zmniejszona wysokość i szerokość przy tej samej głębokości.

Max Pooling



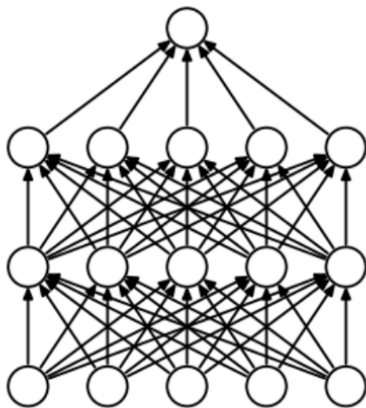
Avg Pooling



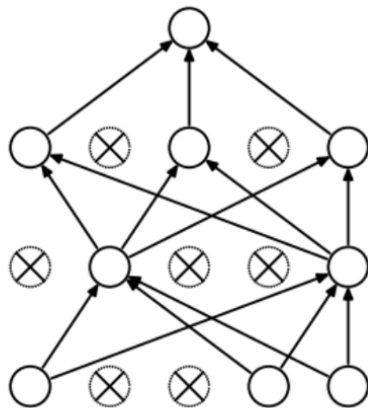
<https://indoml.com>

Polega na losowym usuwaniu z sieci pojedynczych neuronów w trakcie uczenia. Ponieważ skomplikowane sieci (a takie niewątpliwie są głębokie sieci neuronowe), szczególnie dysponujące relatywnie niewielkimi ilościami danych uczących, mają tendencję do dokładnego dopasowywania się do danych, to taki sposób deregulacji zmusza je do uczenia w sposób bardziej zgeneralizowany. W każdej turze uczenia się każdy z neuronów jest usuwany lub zostawiany w sieci. Szanse na usunięcie definiuje się poprzez określenie prawdopodobieństwa, z jakim neuron zostanie usunięty. Zastosowanie dropout w praktyce prowadzi do sytuacji, w której architektura sieci zmienia się dynamicznie i otrzymujemy model, w którym jeden zbiór danych został wykorzystany do nauczania wielu sieci o różniących się architekturach, a następnie został przetestowany na zbiorze testowym z uśrednionymi wartościami wag.

Dropout



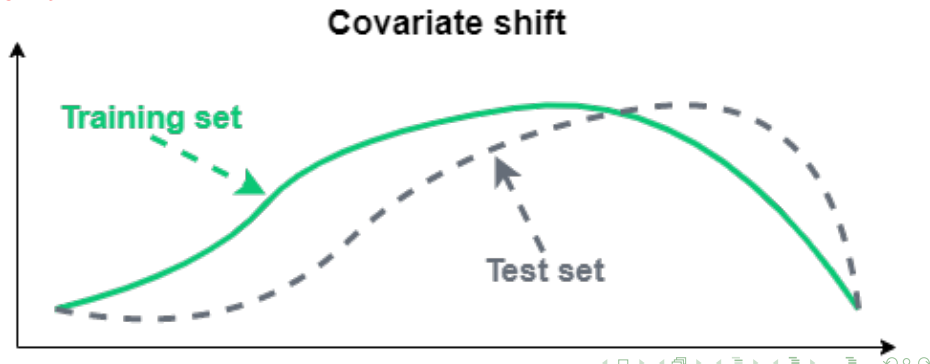
(a) Standard Neural Net



(b) After applying dropout.

Normalizacja batchowa

Normalizacja batchowa ma na celu redukcję tzw. **internal covariate shift**. Mimo że zbiory danych, których używamy do uczenia maszynowego są z reguły dobrze zbilansowane, to jednak podział zbioru na uczący, walidacyjny i testowy powoduje, że zbiory te mają różny rozkład danych wejściowych. Gdybyśmy nałożyli na siebie takie dwa rozkłady, to byłyby one względem siebie przesunięte. To przesunięcie nazywamy **covariate shift**.

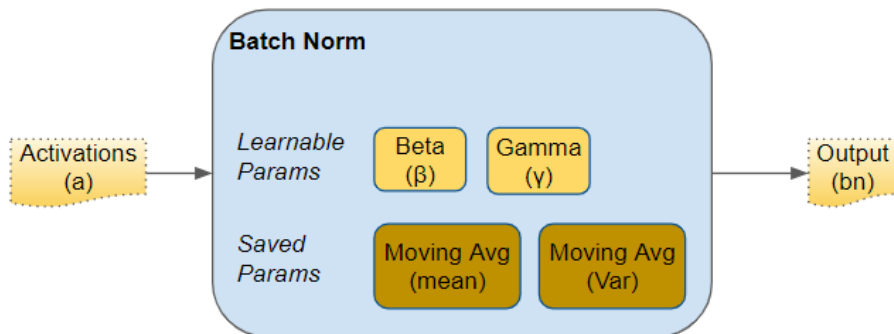


Covariate shift występuje nie tylko przy podziale zbioru lub przy jego wzbogacaniu o nowe dane, ale również w wyniku przechodzenia danych wejściowych przez kolejne warstwy sieci neuronowej. Sieć modyfikuje dane w naturalny sposób, poprzez nakładanie wag przypisanych do połączeń między neuronami w sieci. W konsekwencji każda kolejna warstwa musi się nauczyć danych, które mają nieco inny rozkład niż oryginalne dane wejściowe. Powoduje to nie tylko spowolnienie procesu uczenia, ale również większą podatność sieci na overfitting. Zjawisko przesunięcia rozkładu danych wejściowych w sieci neuronowej zostało nazwane **internal covariate shift**. Jako rozwiązanie tego problemu zaproponowano metodę normalizacji danych wykonywaną pomiędzy warstwami sieci neuronowej, jako część jej architektury.

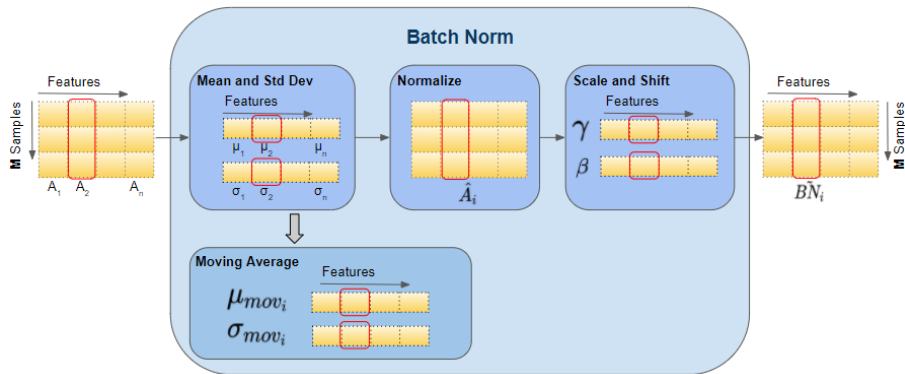
Normalizacja batchowa

Każda warstwa tego typu ma swoje parametry (dla każdej cechy):

- Dwa, które uczymy (β – przesuwanie i γ – skalowanie),
- Dwa, które wyznaczamy z próbki (średnia i wariancja).

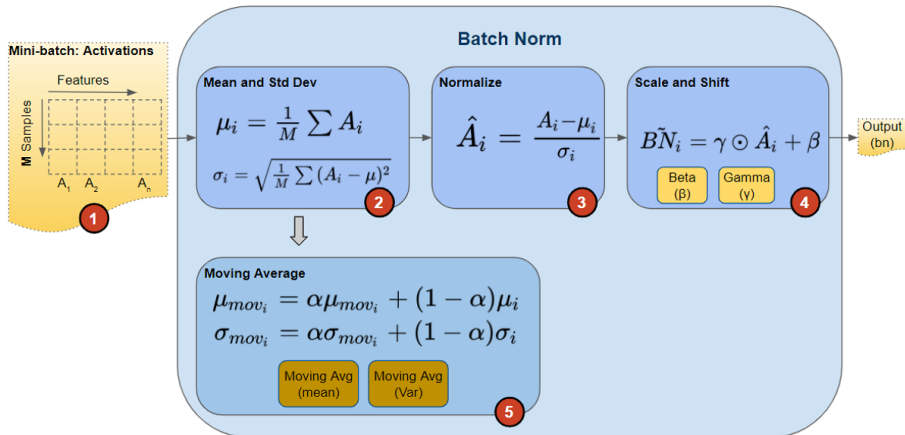


Normalizacja batchowa



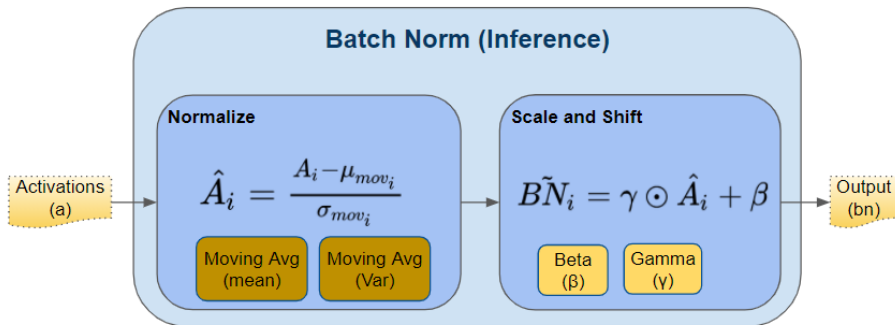
Normalizacja batchowa

Podsumowując: normalizacja batchowa przyspiesza uczenie – pozwala przy mniejszej liczbie iteracji uzyskać takie same rezultaty jak sieci nieposiadające normalizacji batchowej. Pomaga również eliminować przeuczenie.



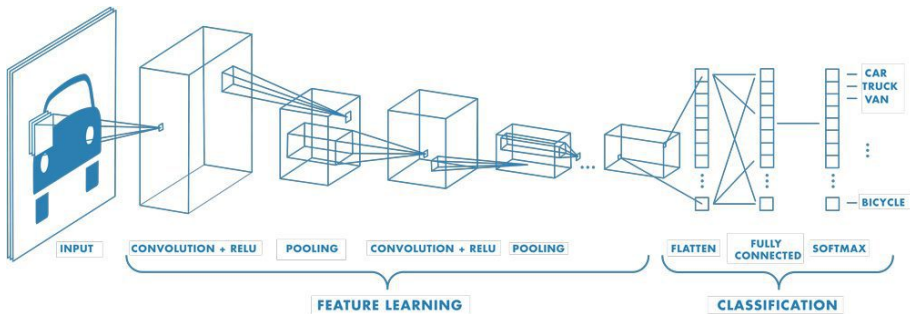
Normalizacja batchowa

Zauważmy, że wykorzystywane średnie i wariancje dla każdej próbki różnią się pomiędzy próbkami i nie można ich bezpośrednio wykorzystać w fazie testowej. Najczęściej wykorzystuje się do tego średnie ruchome (wygładzanie wykładnicze), które wyznaczone są w fazie uczącej, a wykorzystywane w fazie testowej.

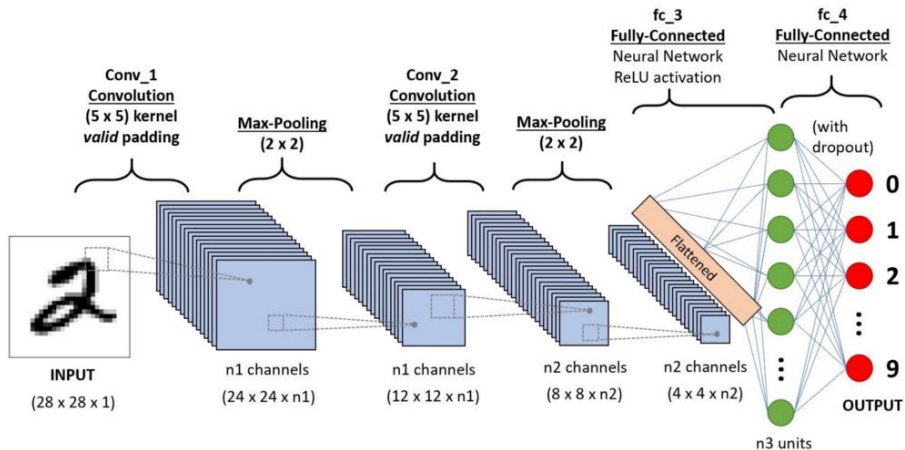


Ostatnia warstwa swoją nazwę wzięła od pełnego połączenia każdego neuronu w jednej warstwie z neuronem w warstwie sąsiedniej. Tworzona jest przez zestawienie przetworzonych danych z warstw wcześniejszych w jednokolumnowy wektor.

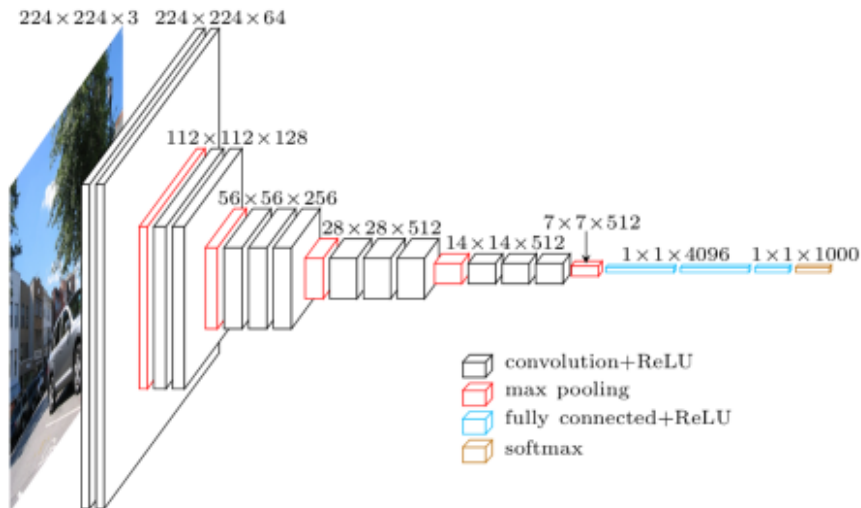
Przykładowe architektury



Przykładowe architektury



Przykładowe architektury



- Skorzystaj ze sprawdzonej architektury sieci i dostosuj ją do swoich potrzeb.
- Zaczynj od overfittingu i wprowadź regularyzację.
- Umieszczaj dropout w warstwie gęsto połączonej, a normalizację batchową wprowadź do podsieci konwolucyjnej.
- Kernel size powinien być dużo mniejszy niż rozmiar obrazka.
- Eksperymentuj z różnymi ustawieniami hiperparametrów.
- Korzystaj z GPU (Google Colaboratory).
- Zgromadź tyle danych uczących, ile to możliwe. Jeżeli nie możesz zgromadzić większej ilości danych, skorzystaj z generatora danych (augmentacja).
- Bardzo głęboka i rozbudowana sieć będzie miała silne skłonności do przeuczenia. Skorzystaj z tak płytkiej sieci, jak to możliwe. W szczególności nie przesadzaj z ilością neuronów i warstw w podsieci gęsto połączonej.
- Upewnij się, że zbiory uczące i testowe są dobrze zbalansowane i mają zbliżone rozkłady. W przeciwnym wypadku na zbiorze testowym będziesz zawsze uzyskiwał dużo gorsze wyniki niż na zbiorze uczącym.