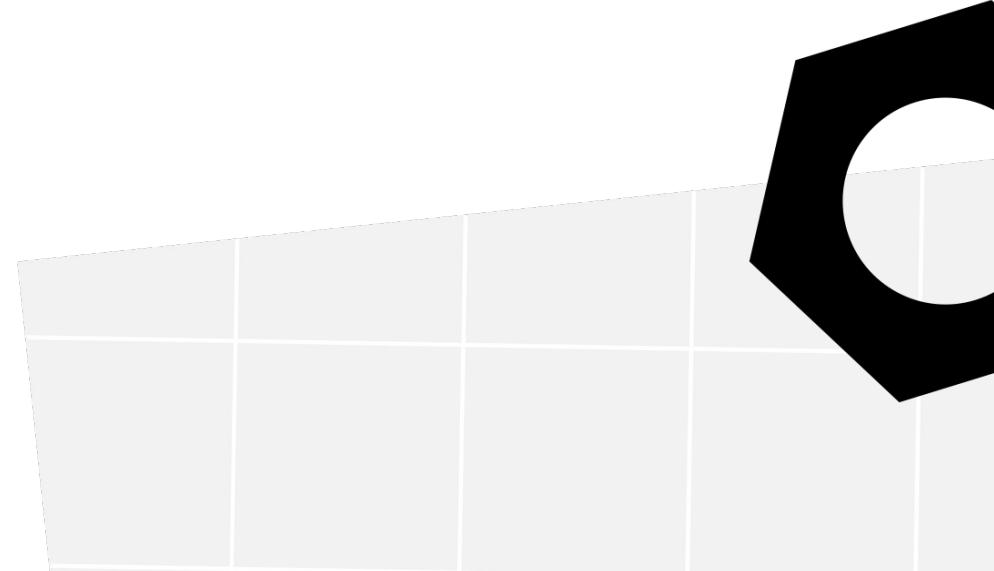




# **Przetwarzanie obrazu - Computer Vision**

**Kurs Data Science**



# Plan prezentacji:

- Computer Vision – ogólnie
- Computer Vision jako proces w ujęciu konwencjonalnym (biblioteka OpenCV)
  - Czym jest obraz?
    - Komputerowa reprezentacja obrazu
    - Rodzaje obrazów
    - Przestrzenie barwowe
  - Przetwarzanie obrazów
    - Filtry splotowe
    - Wyrównanie histogramu
    - Progowanie
    - Operacje morfologiczne
  - Wyciąganie cech i informacji
    - Kontury
    - HuMoments
  - Dokonywanie predykcji na podstawie cech.

# Materiały na zajęcia

pobierz notatnik z kodem do zajęć

[link](#)

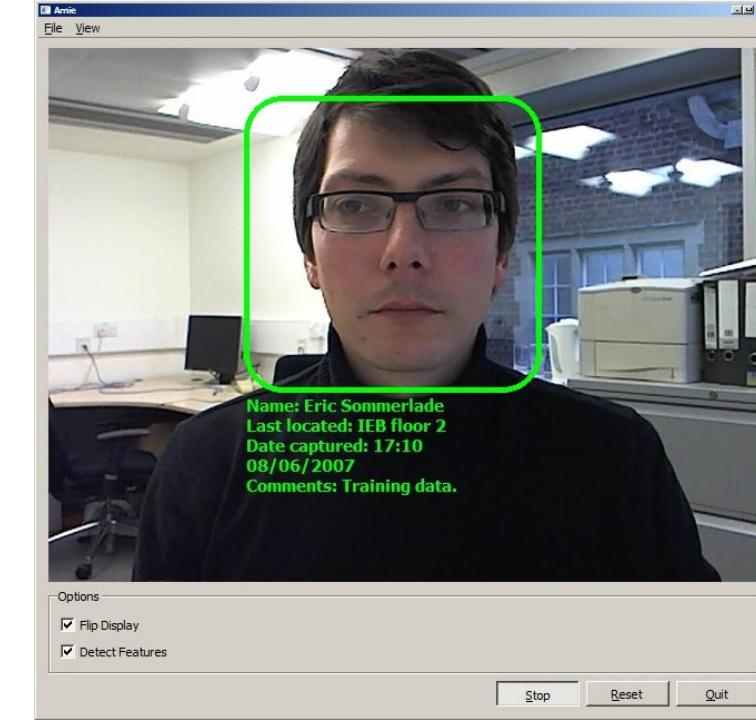
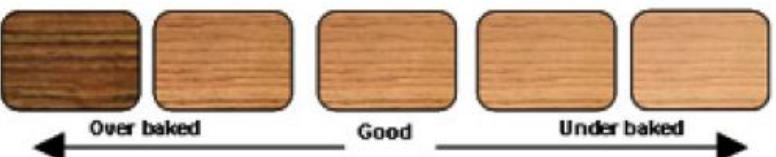
# Co to jest computer vision?

Computer Vision to dziedzina zajmująca się tym, w jaki sposób można zmusić komputery do prawdziwego zrozumienia tego co zawierają w obrazach/filmy. Computer Vision zajmuje się automatycznym wyodrębnianiem i analizą użytecznych informacji z pojedynczego obrazu lub sekwencji obrazów.



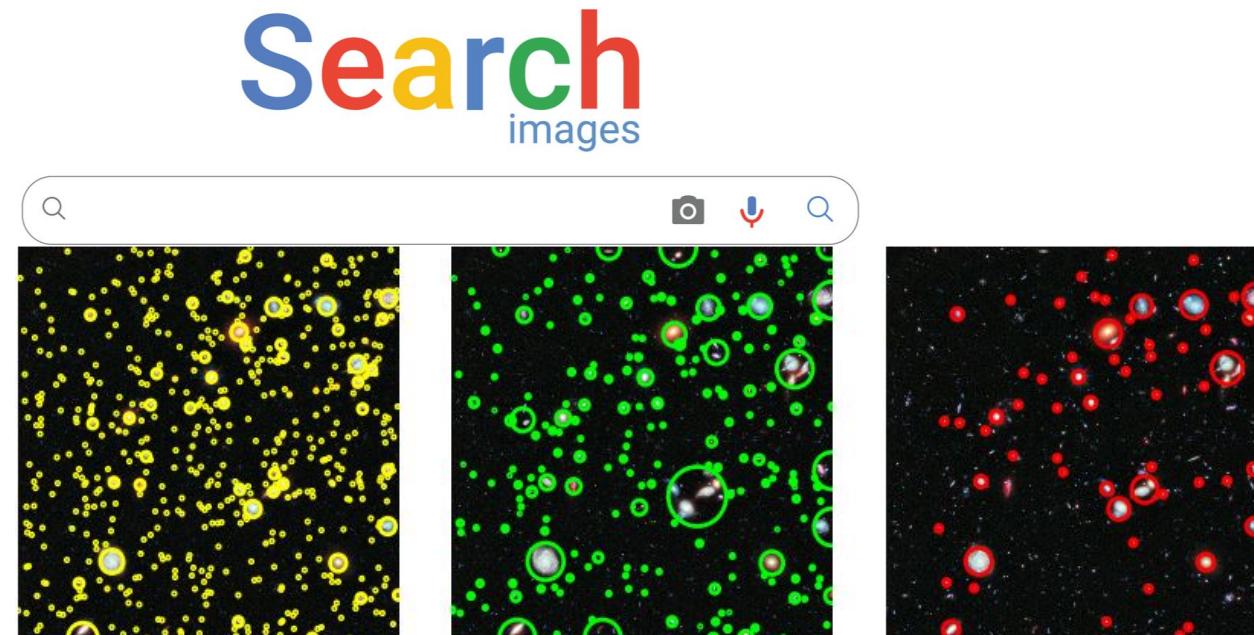
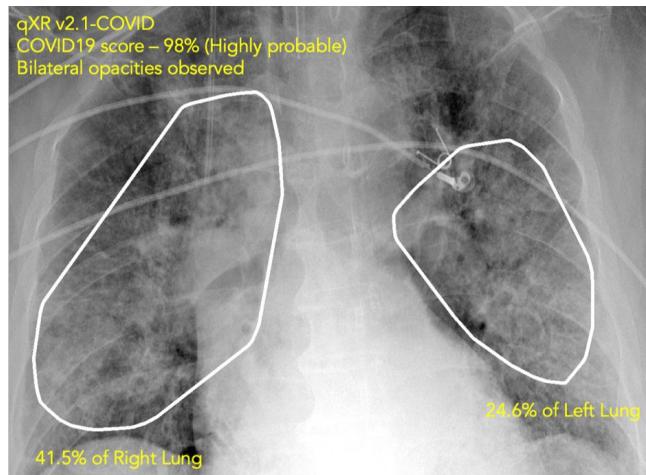
# Przykłady zastosowań

- Systemy sterowania
  - Autonomiczne samochody
  - Kontrola jakości
- Bezpieczeństwo
  - Systemy identyfikacji osób
  - Identyfikacja dokumentów



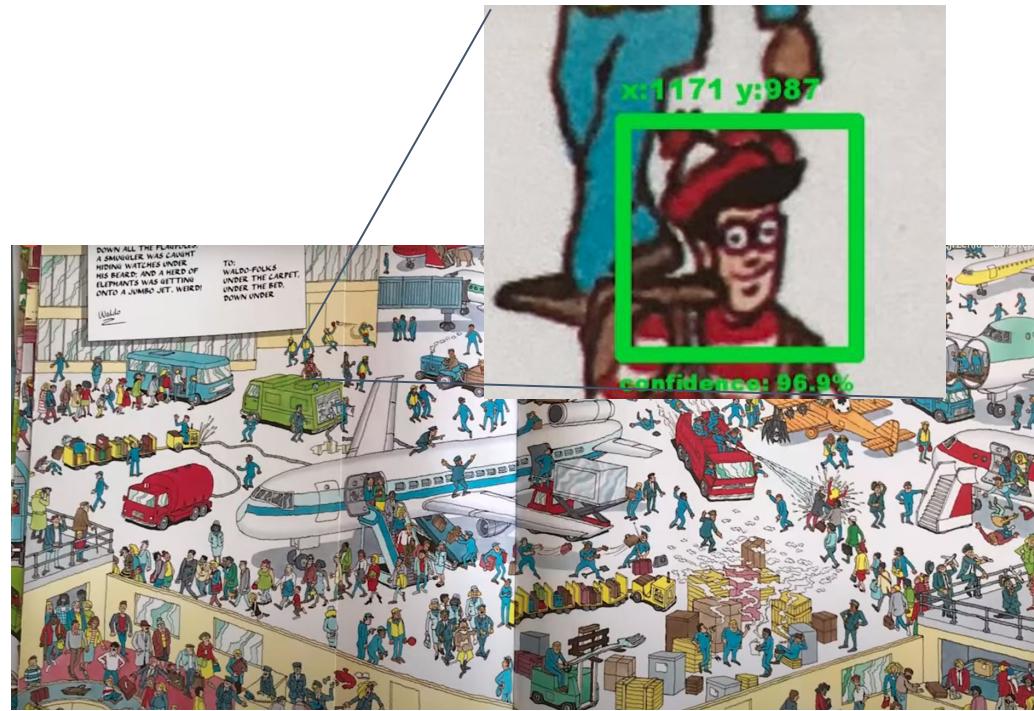
# Przykłady zastosowań

- Medycyna
  - Automatyczna detekcja i analiza komórek pod mikroskopem
  - Diagnostyka
- Wyszukiwanie informacji
  - wyszukiwanie obrazów podobnych do zapytania



# Computer Vision to też:

Algorytmy poszukujące Waldo:



<https://theblue.ai/blog/5-weird-ai-applications/>

Generowanie abstrakcyjnych dzieł sztuki:



Stallone sam w domu



<https://www.youtube.com/c/CtrlShiftFace/videos>

Generator nieistniejących ludzi:



<https://thispersondoesnotexist.com/>

# Proces widzenia maszynowego

Czyli plan na resztę dnia

Raw image - Surowy obraz



Enhanced Image - Ulepszony obraz



Prediction - Predykcja

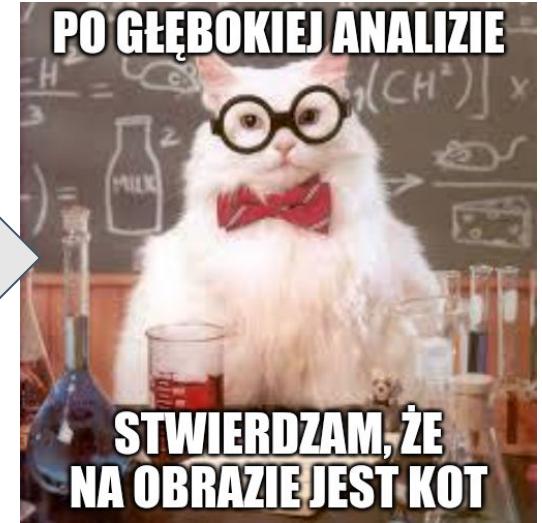


Image preprocessing - wstępne przetwarzanie obrazu

Feature Extraction - ekstrakcja cech

Classification - klasyfikacja (przykładowe zadanie)

Q	A	S	D	F	G	H
1	0	2	3	7	8	6
2	4	5	3	2	1	4
6	8	9	7	5	1	3
2	1	3	3	5	4	3
6	8	7	8	9	1	3
7	4	7	0	9	2	3

Raw image - Surowy obraz



# Pierwszy krok! – Czyli czym właściwie jest obraz?

# Rodzaje obrazów

Zapisane za pomocą wektorów



**vector**



Składające się z pixeli



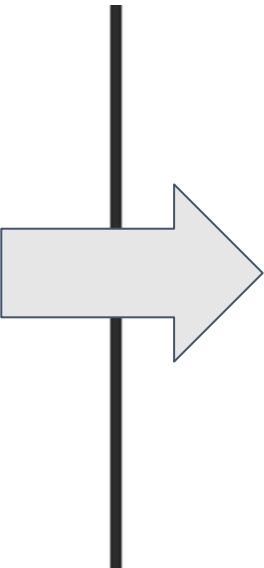
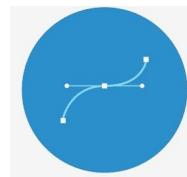
**raster**

# Rodzaje obrazów

W przetwarzaniu obrazów jednak najczęściej wykorzystuje się obrazy rastrowe. Dlatego też to właśnie na nich się skupimy



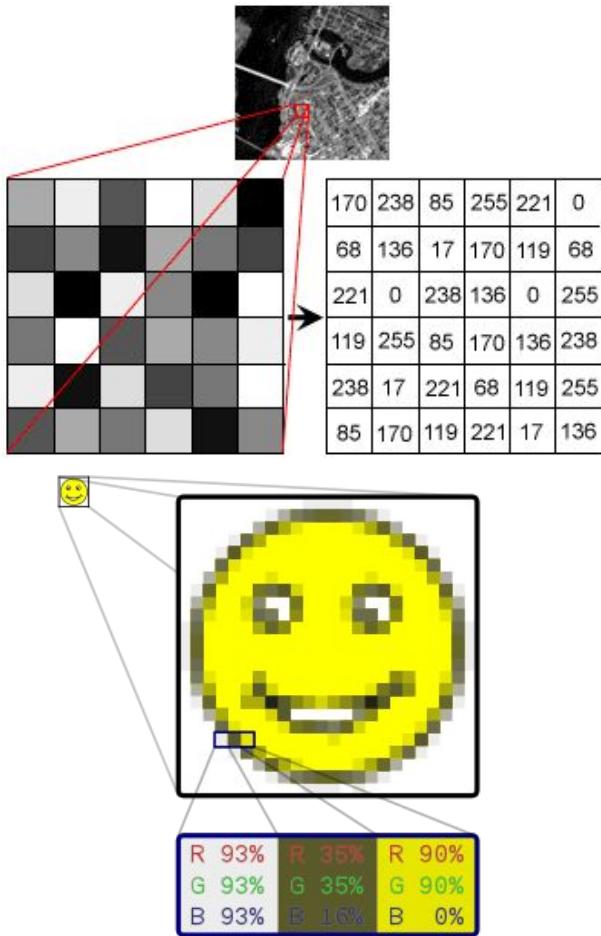
**vector**



**raster**



# Obraz rastrowy



- Obraz rastrowy jest dwuwymiarową tablicą prostokątną (macierzą)
- Pojedynczy element macierzy określany jest jako piksel:
  - czarno-biały,
  - w skali szarości,
  - kolorowy (dodatkowy 'wymiar')

# Podstawowe pojęcia:

- rozdzielcość
- pixel
  - jasność pixela  $\{0, 1\}$ ,  $\{0, 1, \dots, 255\}$  (są też inne możliwości)
  - przeźroczystość pixela (dodatkowy kanał alfa)
  - ilość bitów, na których jest zapisana informacja na temat pixela

Z kilku powodów jasność pixela przyjęto się zapisywać w 2 formatach

- floatowa wartość 0-1
- intowa wartość ograniczona ilością bitów (np 0-255)

# Rozdzielcość



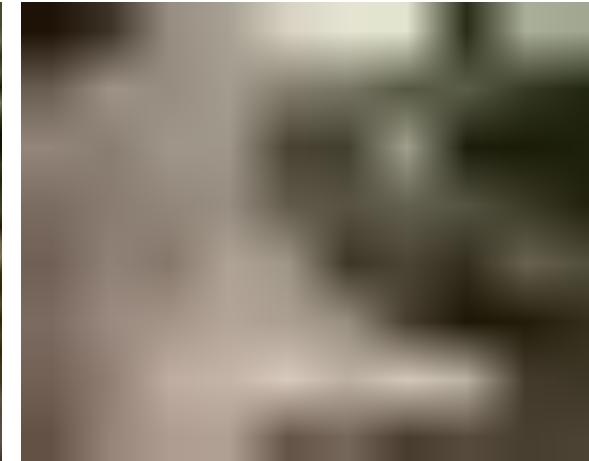
(a)  $400 \times 300$



(b)  $100 \times 75$



(c)  $40 \times 30$



(d)  $10 \times 8$

# Modyfikacja rozdzielczości

## Warto pamiętać

W przypadku zmiany rozdzielczości obrazu (np. w celu ograniczenia zajętości pamięciowej) przeznaczonego do analizy należy posługiwać się stałym współczynnikiem dla osi x i y. W innym przypadku zniekształcony obiekt, może być błędnie interpretowany.

```
# Przykład:

scale_percent = 60 # percent of original size
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)

# resize image
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
```

# Pixeły



(a) RGB,  
256<sup>3</sup> możliwych kolorów



(b) 8-bitów,  
256 możliwych kolorów



(c) 4 bity  
16 możliwych kolorów



(d) 2 bity  
4 możliwe kolory

# Wartości pixeli

## Warto pamiętać

W przypadku wykorzystywania wartości pixeli w modelach uczenia maszynowego jako featurów należy zastosować odpowiednią normalizację.

```
# Przykład gdy mamy wartości pixeli (0-255)
# normalize to the range 0-1
pixels /= 255.0
```

Więcej:

<https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/>

0	0	0	0
0	0	0	0
0	0	0	0

## Obraz czarno-biały

Jest zapisywany w pamięci komputera w postaci jednej macierzy.

Wartości pixeli przyjmują jedynie 2 różne wartości

- [0,1]
- [0, 255]

1	1	1	1
1	1	1	1
1	1	1	1

lub 255\*



# Obraz w skali szarości

Jest zapisywany w pamięci komputera w postaci jednej macierzy.

Wartości pixeli znajdują się w przedziale

- 0 - 1 (float)
- 0 - 255 (integer)

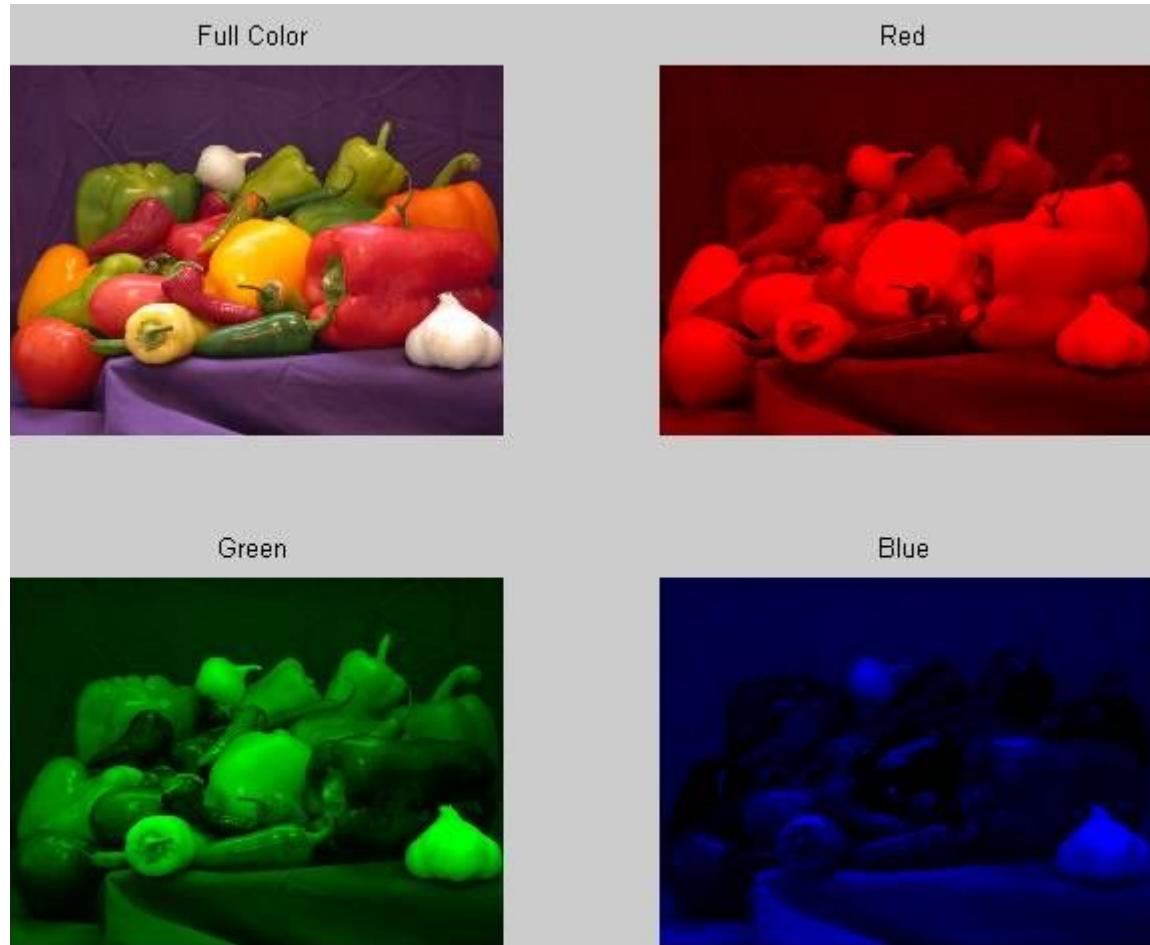


Pani ze zdjęcia doczekała się własnej strony na wikipedii:  
<https://en.wikipedia.org/wiki/Lenna>

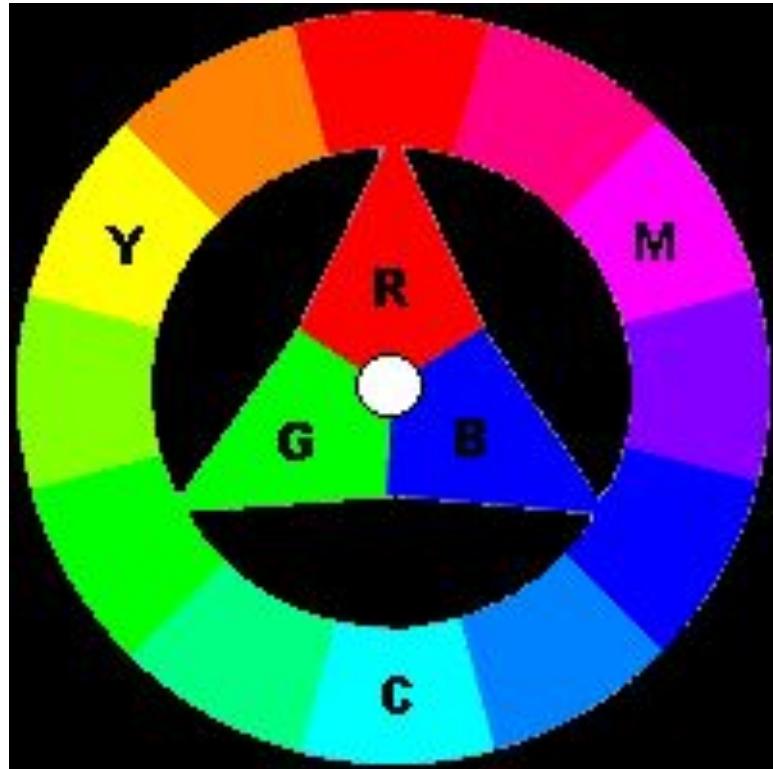
# Obraz kolorowy

Jest zapisywany w pamięci komputera w postaci 3 macierzy, gdzie każda z nich reprezentuje jedną przestrzeń barwową.

To co widzimy ostatecznie, jest tylko zabiegiem bazującym na złożeniu tych kolorów. W jaki sposób możemy składać nasze kolory?

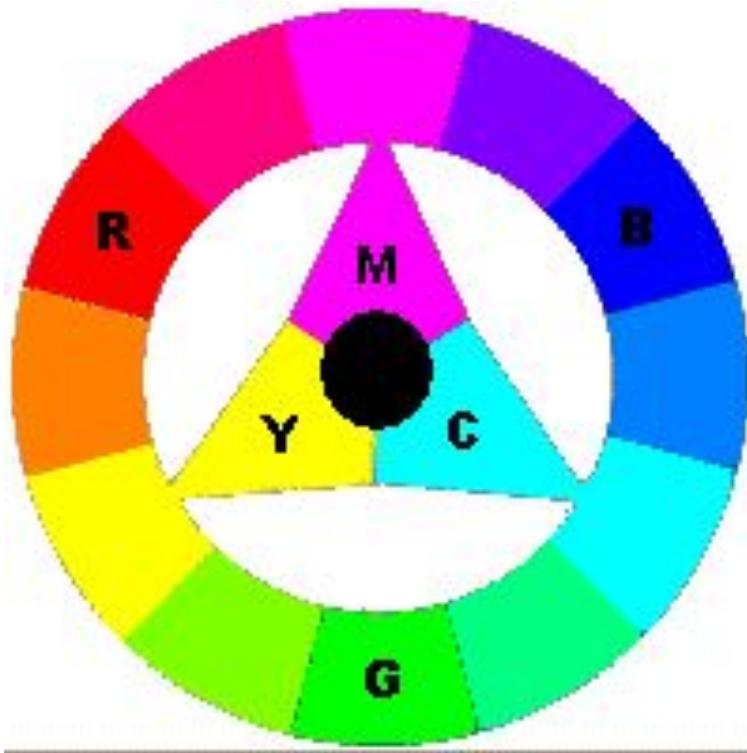


# Additive Colors



- Brak jakiejkolwiek wartości na każdym z kanałów oznacza kolor czarny.
- Im więcej dodanych kolorów, tym jaśniejszy.
- Tworzenie koloru przez dodanie trzech kolorów podstawowych: czerwonego, zielonego i niebieskiego.

# Subtractive Colors



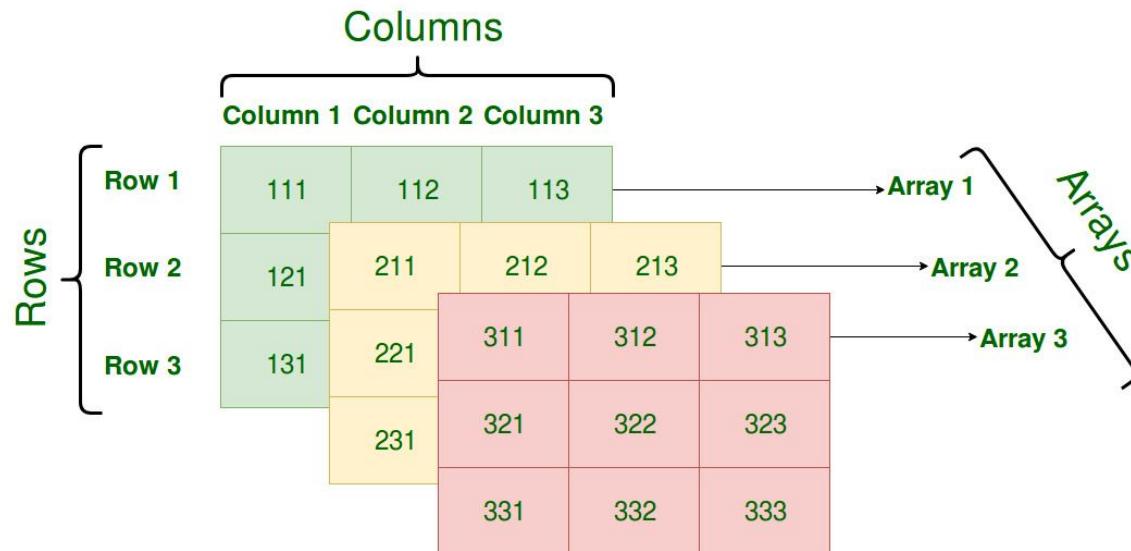
- Brak wartości → Biały kolor
- Odjęcie wszystkim kolorów skutkuje kolorem czarnym.
- Następnie odejmując kolory cyjan, magenta i / lub żółty, możliwe uzyskanie jest wszystkich kolorów.

# Obraz kolorowy

Skoro w pamięci komputera przechowywane są macierze z liczbami reprezentujące wartości pixeli. To skąd wiadomo w jaki sposób te macierze interpretować?

No nie wiadomo.

Musimy niestety dysponować informacją jakiej przestrzeni barw dotyczą.



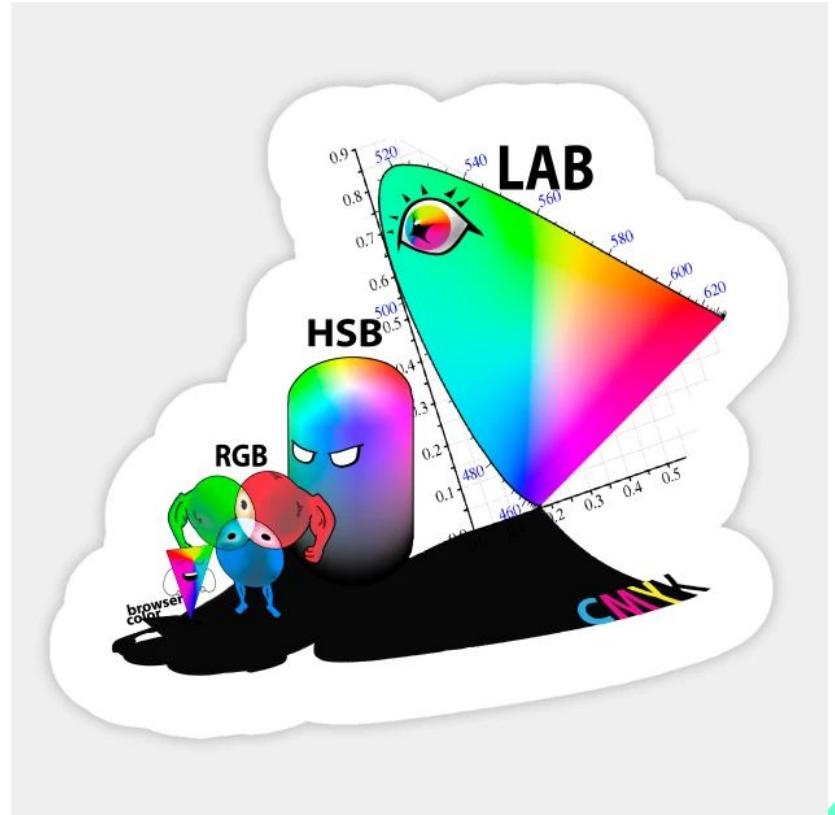
# Przestrzenie barwowe

- Przestrzeń barw to metoda określania, tworzenia i wizualizacji kolorów.
- Kolory są zwykle określane za pomocą trzech atrybutów lub współrzędnych, które reprezentują jego położenie w określonej przestrzeni barw.
- Przestrzeń barw to trójwymiarowe układy współrzędnych, w których każdy kolor jest reprezentowany przez pojedynczy punkt.

# Przestrzenie barwowe

Mamy wiele rodzajów przestrzeni barwowych:

- HSI (Hue, Saturation, Intensity)
- RGB (Red, Green, Blue)
- CMY(K) (Cyan, Magenta, Yellow, Black)
- CIE
- Luminance - Chrominance



# A po co?

Przestrzenie barwowe są nam potrzebne z kilku powodów:

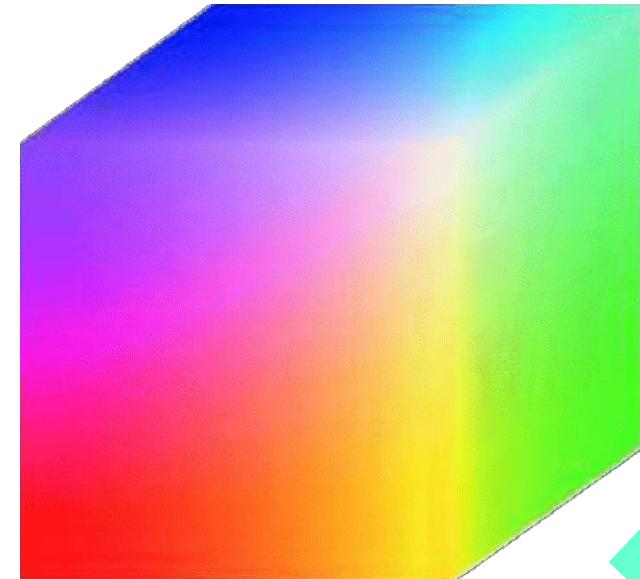
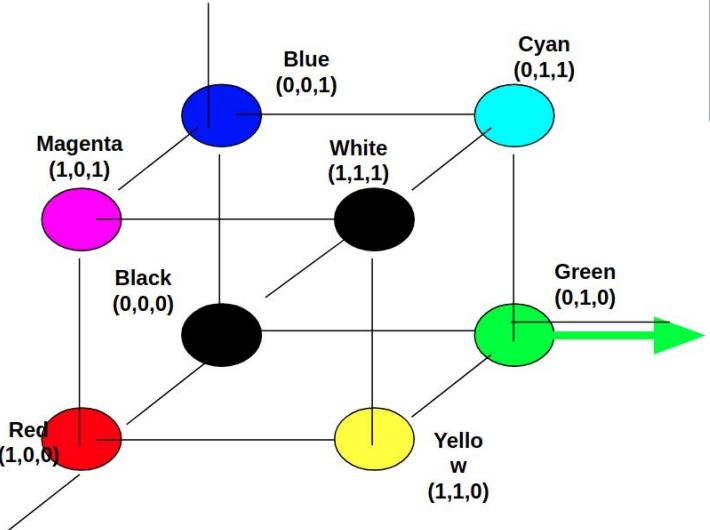
- Dane obrazowe mogą być przechowywane w różnej postaci, warto wiedzieć czym one się różnią
- Część algorytmów wymaga zmiany przestrzeni barwowej do poprawnego działania (Histogram Equalization)
- Wybór przestrzeni barwowej może wpływać na działanie modeli służących do klasyfikacji czy detekcji

Więcej:

<https://towardsdatascience.com/understand-and-visualize-color-spaces-to-improve-your-machine-learning-and-deep-learning-models-4ece80108526>

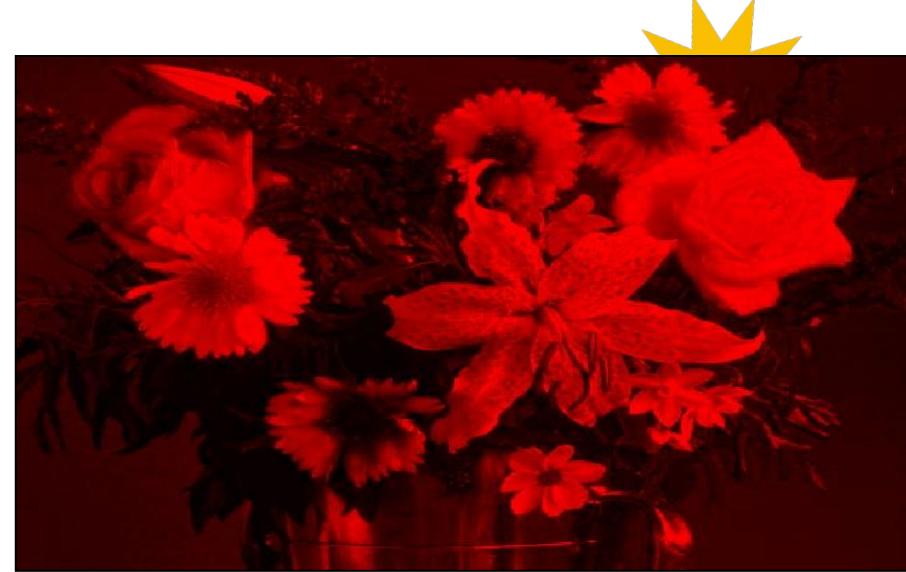
# RGB

- Jeden z najprostszych modeli kolorów. Współrzędne kartezjańskie dla każdego koloru; oś jest przypisana do trzech podstawowych kolorów: czerwonego (R), zielonego (G) i niebieskiego (B).
- Odpowiada zasadom **kolorów addytywnych**.
- Inne kolory są reprezentowane jako dodatkowa mieszanka R, G i B.
- Najczęściej stosowany w przetwarzaniu obrazów. Jednak nie zawsze.





Pełen obraz



Kanał czerwony



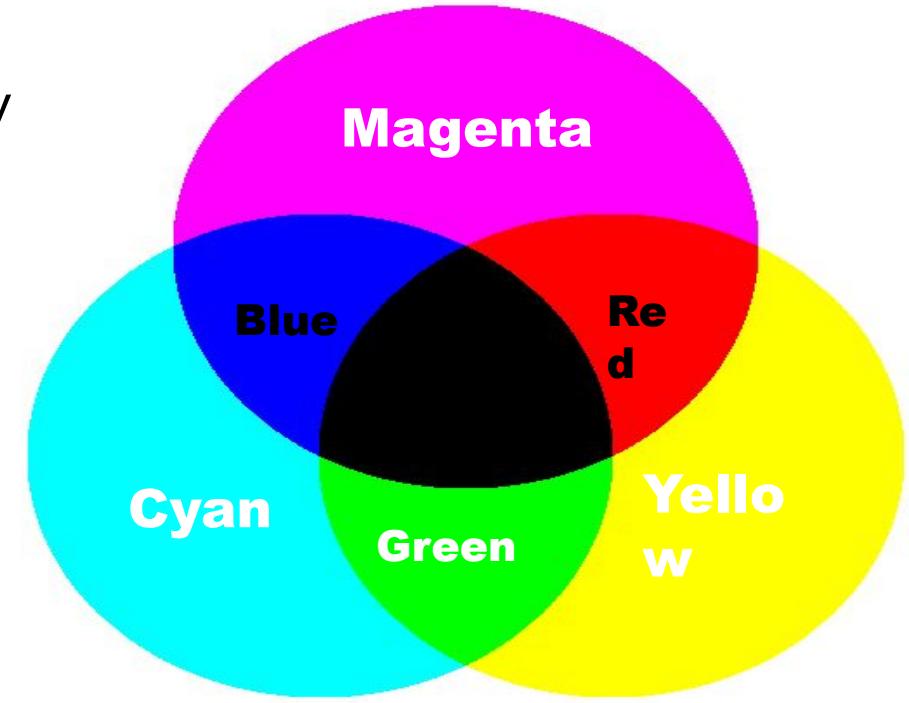
Kanał zielony



Kanał niebieski

# CMYK

- Odpowiada zasadzie subtraktywnych kolorów, wykorzystując trzy kolory: cyjan, magenta i żółty.
- Teoretycznie jednolita mieszanka cyjanu, magenty i żółtego daje czerń (środek obrazu).
- W praktyce wynikiem jest zwykle brudny brązowo-szary odcień (w trakcie wydruku). Dlatego czarny jest często używany jako czwarty kolor.





Pełen obraz



Cyan kanał (1-R)



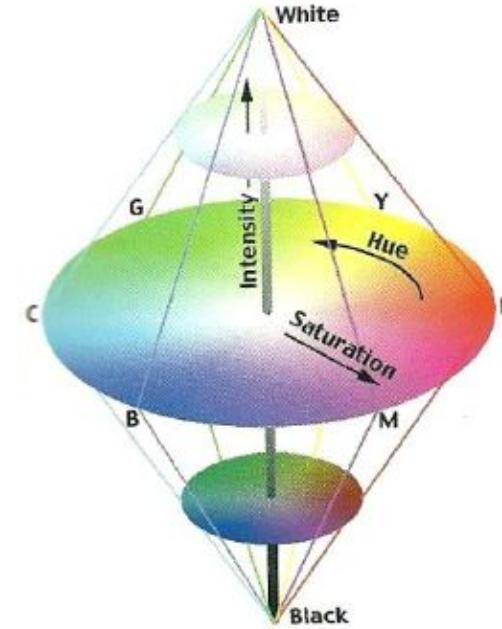
Magenta kanał (1-G)



Żółty kanał (1-B)

# HSI / HSL / HSV

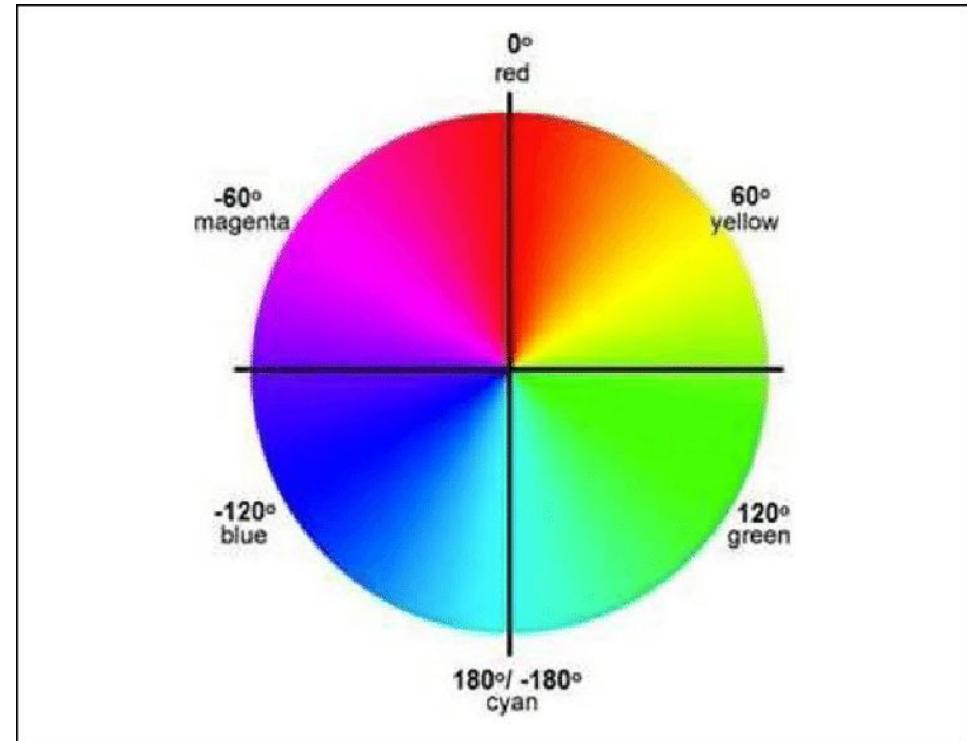
- Stworzony na podobieństwo postrzegania koloru przez człowieka.
- Aplikacje do przetwarzania obrazu, takie jak operacje np. operacje na histogramie, działają tylko na intensywności obrazu i są wykonywane znacznie łatwiej na obrazie w przestrzeni kolorów HSI.



- H=Hue,
- S = Saturation,
- I (Intensity)
- L = Lightness
- V = Value

# HSI / HSL / HSV

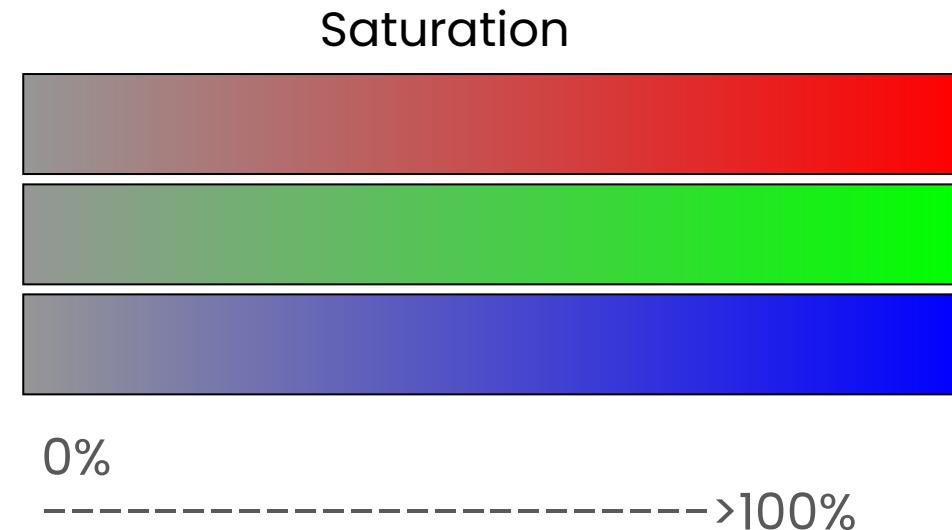
- Hue (Odcień) – część odpowiadająca za barwę, wyrażona jako liczba od 0 do 360 stopni:
  - Czerwony od 0 do 60 stopni.
  - Żółty od 61 do 120 stopni.
  - Zielony od 121 do 180 stopni.
  - Cyjan od 181 do 240 stopni.
  - Niebieski od 241 do 300 stopni.
  - Magenta od 301 do 360 stopni.



# HSI / HSL / HSV

## Saturation (Nasycenie):

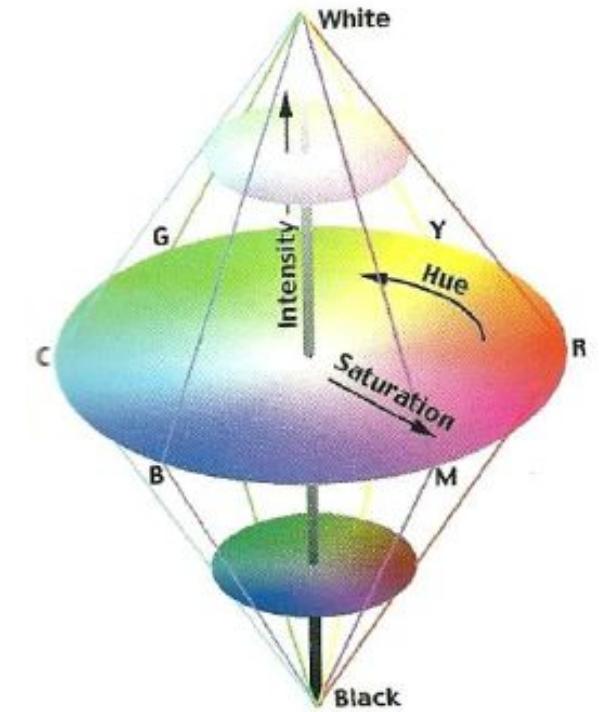
- Stopień, w jakim kolor różni się od neutralnej szarości.
- 100% = pełne nasycenie, wysoki kontrast między innymi kolorami.
- 0% = odcień szarości, niski kontrast.



# HSI / HSL / HSV

## Intesity (Intensywność):

- Jasność każdego koloru jest zdefiniowana przez jego wysokość wzdłuż osi pionowej.
- 100% nasycenia można osiągnąć jedynie przy 50% intensywności.
- Wraz ze wzrostem lub spadkiem intensywności nasycenie spada.
- Biel przy 100% intensywności. Barwa i nasycenie nie istnieją.
- Czerń przy intensywności 0%. Barwa i nasycenie nie istnieją.





Pełen obraz



Hue kanał



Saturation kanał

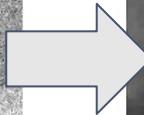


Intensity kanał

Raw image - Surowy obraz



Enhanced Image - Ulepszony obraz



**Image preprocessing –  
wstępne przetwarzanie  
obrazu**

**Idziemy  
dalej!**

# Image processing

Preprocessing obrazu ma za zadanie:

- odseparować szum od obrazu (denoising),
- wyostrzyć obraz,
- poprawić jakość obrazu i jego szczegółowość.

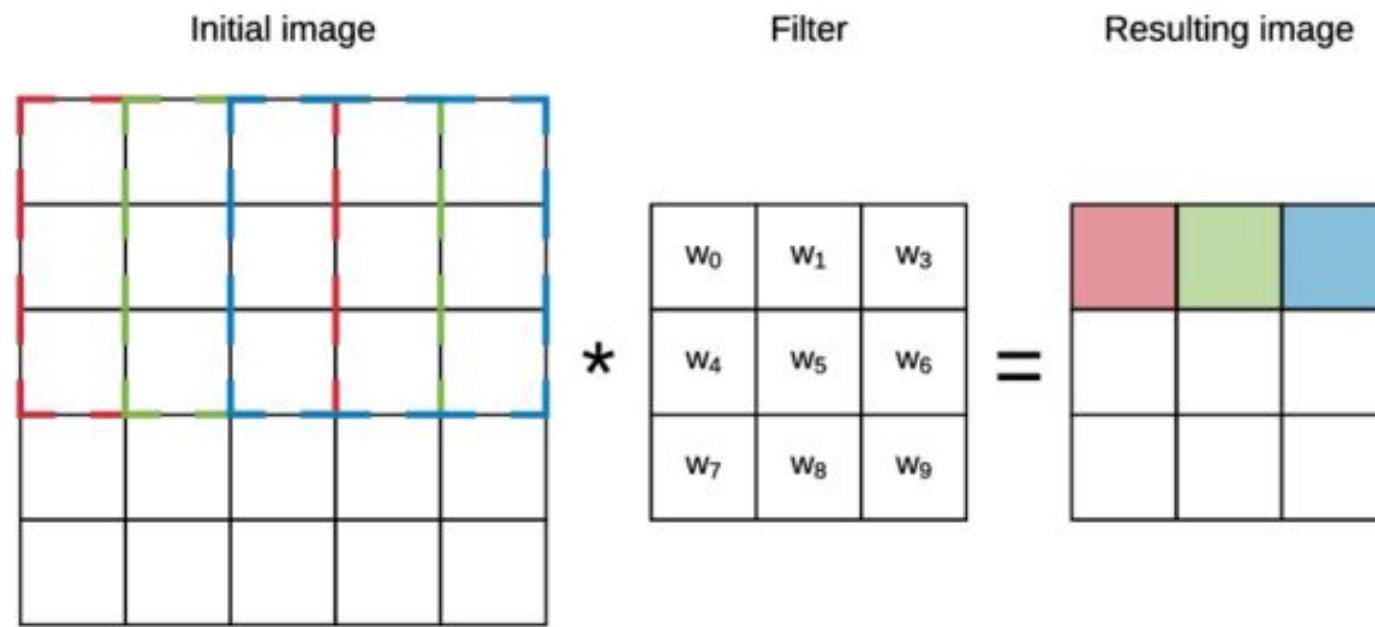
Mozemy tego dokonać poprzez:

- filtry splotowe,
- wyrównanie histogramu,
- progowanie,
- operacje morfologiczne.

# Filtracja splotowa

Bardzo ważna sprawa!

Filtры сplotowe działają na operacji konwolucji, czyli mnożenia poszczególnych pixeli przez odpowiednie wagi w celu uzyskania nowej wartości pixela ...



$$\left( \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2, 2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

# A po co?

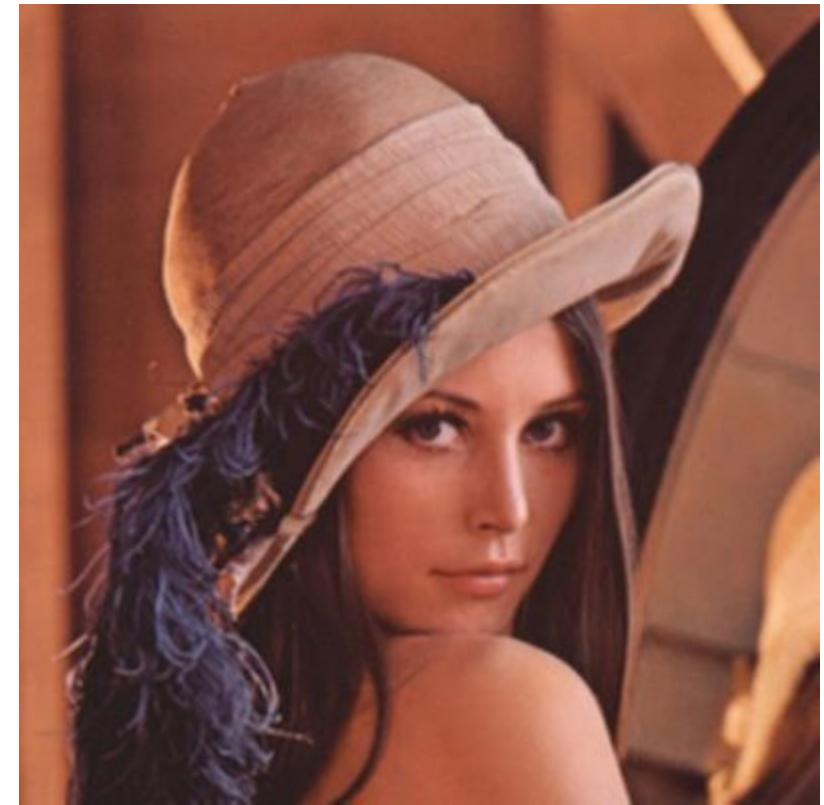
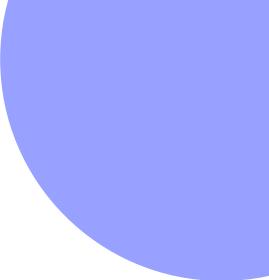
- Filtry splotowe są kwintesencją działania Konwolucyjnych sieci neuronowych (o których na następnych zajęciach)
- Mogą służyć do usuwania szumów w konwencjonalnym podejściu do przetwarzania obrazów
- Mogą służyć do wykrywania krawędzi i ważnych detali występujących na obrazach.
- Są używane w programach do obróbki zdjęć



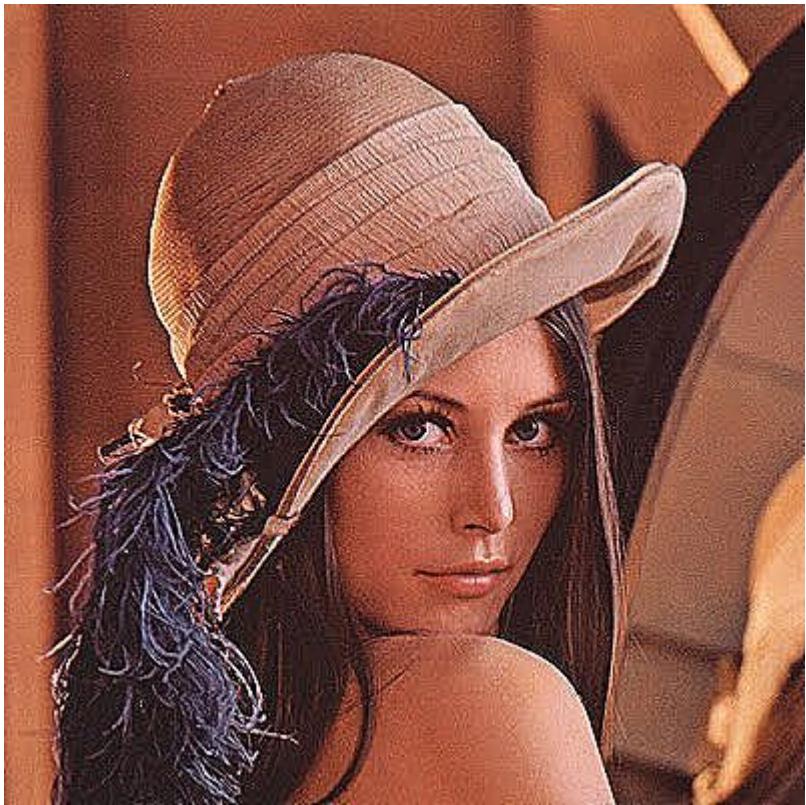
**Identity**  $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$



**Box blur**  $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$



# Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$


# Edge det.

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


# Sobel

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

Left

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Bottom

Full



```
#Filters

import numpy as np

def apply_filter(source, kernel):
    new_image = cv2.filter2D(source, -1, kernel)
    cv2_imshow(new_image)
    return new_image

#identity
identity_kernel = np.array([[0, 0, 0],
                           [0, 1, 0],
                           [0, 0, 0]])
apply_filter(img, identity_kernel)

#blur
blur_kernel = np.ones((3, 3), np.float32) / 9
apply_filter(img, blur_kernel)

#sharpen
sharpen_kernel = np.array([[0, -1, 0],
                           [-1, 5, -1],
                           [0, -1, 0]])
apply_filter(img, sharpen_kernel)
```

**Zobaczmy jak to  
wygląda w Pythonie!**

```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#simply_edge_detection
simply_edge_kernel = np.array([[[-1, -1, -1],
                                [-1, 8, -1],
                                [-1, -1, -1]]])

apply_filter(gray_img, simply_edge_kernel)

#bottom sobel
bottom_sobel_kernel = np.array([[[-1, -2, -1],
                                 [0, 0, 0],
                                 [1, 2, 1]]])

bottom_sobel = apply_filter(gray_img, bottom_sobel_kernel)

#bottom sobel
left_sobel_kernel = np.array([[1, 0, -1],
                             [2, 0, -2],
                             [1, 0, -1]])

left_sobel = apply_filter(gray_img, left_sobel_kernel)

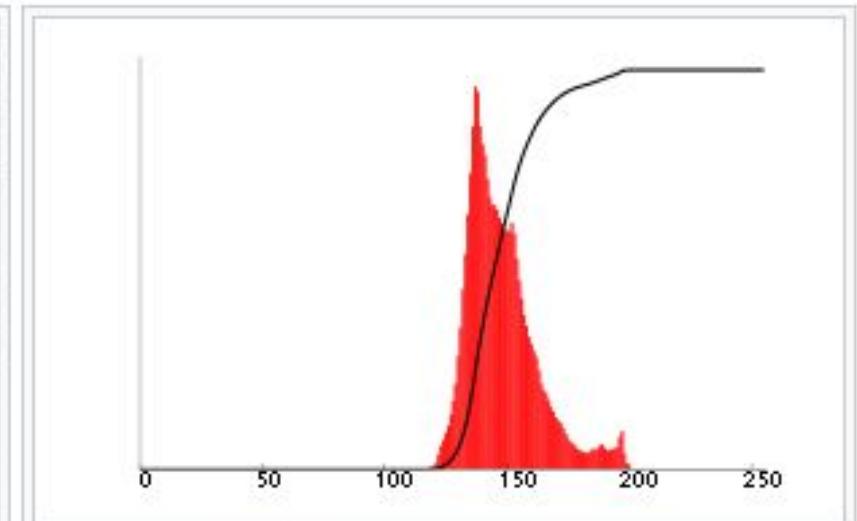
#sobel
sobel = left_sobel/2 + bottom_sobel/2
cv2_imshow(sobel)
```

**Zobaczmy jak to  
wygląda w Pythonie!**

# Histogram obrazu

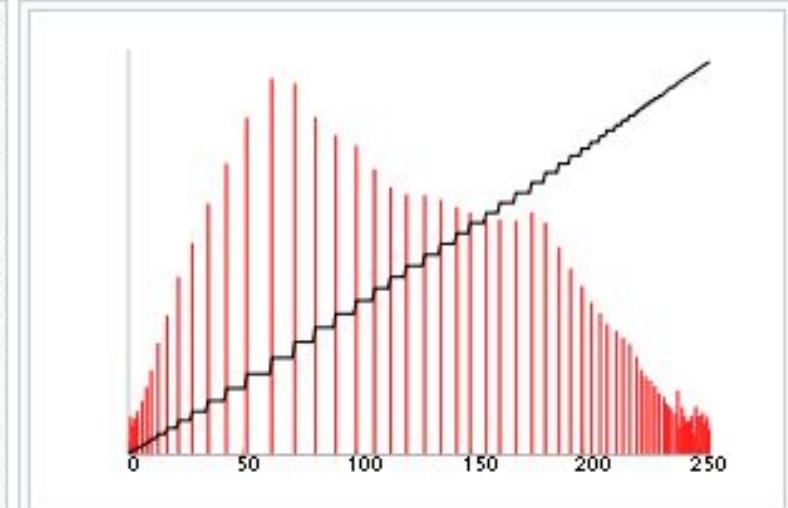
Histogram to graficzna reprezentacja rozkładu intensywności obrazu. Mówiąc prościej, reprezentuje liczbę poszczególnych wartości pixela.

Po lewej mamy obraz, a po prawej na czerwono – histogram, czarna linia – histogram skumulowany



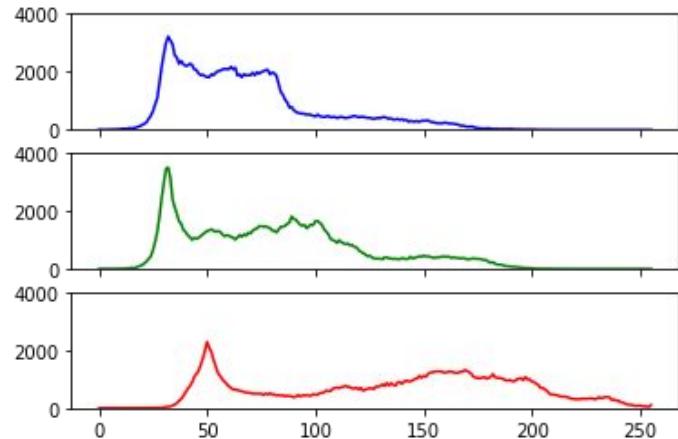
# Wyrównanie histogramu

Osiąga to poprzez efektywne rozłożenie najczęściej występujących wartości intensywności, czyli rozciągnięcie zakresu intensywności obrazu. Ta metoda zwykle zwiększa globalny kontrast obrazów, gdy użyteczne dane są reprezentowane przez bliskie wartości kontrastu. Pozwala to obszarom o niższym lokalnym kontraście uzyskać wyższy kontrast.

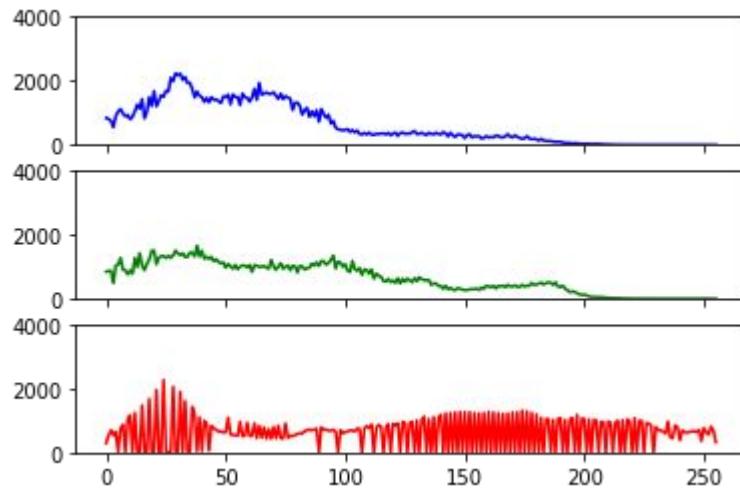
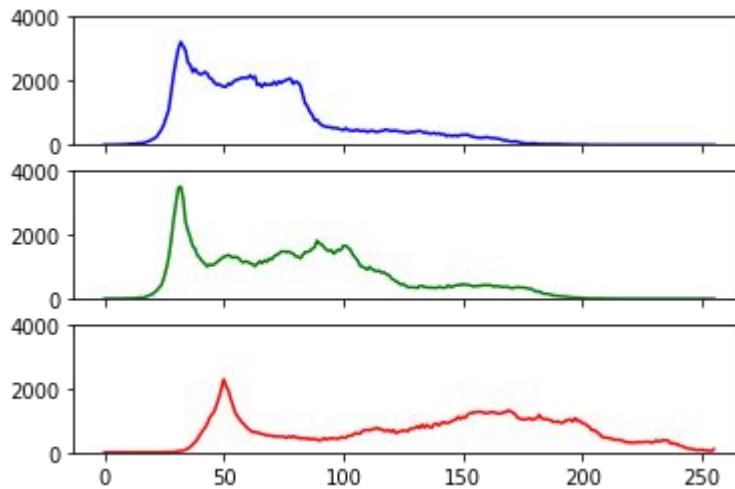


# Histogram kolorowego obrazu

- Histogram kolorowego obrazu należy rozpatrywać osobno dla każdego kanału.
- Korekcja histogramu nie może być zastosowana oddzielnie do składowych czerwonego, zielonego i niebieskiego obrazu, ponieważ prowadzi to do dramatycznych zmian w balansie kolorów obrazu.
- Obraz należy więc wcześniej skonwenterować na obraz HSV/HSL. Gdzie modyfikowana będzie jedynie 3 składowa, która nie wpływa na barwę.



# Histogram kolorowego obrazu



```
import matplotlib.pyplot as plt

def show_histogram(image):
    ax1 = plt.subplot(311)
    plt.xlim([0, 256])
    plt.ylim([0, 4000])
    for i, col in enumerate(['b', 'g', 'r']):
        hist = cv2.calcHist([image], [i], None, [256], [0, 256])
        ax1 = plt.subplot(3,1,i+1, sharey=ax1)
        plt.plot(hist, color = col)

    if i<2:
        plt.setp(ax1.get_xticklabels(), visible=False)

plt.show()

show_histogram(img)
```

# Histogram

```
def show_hsv_equalized(image):
    new_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    H, S, V = cv2.split(new_image)
    eq_V = cv2.equalizeHist(V)
    eq_image = cv2.cvtColor(cv2.merge([H, S, eq_V]), cv2.COLOR_HSV2BGR)
    cv2.imshow(eq_image)
    show_histogram(eq_image)

show_hsv_equalized(img)
```

# Wyrównanie histogramu

# Progowanie

Progowanie (ang. thresholding) – metoda uzyskiwania obrazu binarnego na podstawie obrazu kolorowego lub w odcieniach szarości. Polega na wyznaczeniu dla danego obrazu progu jasności, a następnie piksele jaśniejsze od wyznaczonego progu otrzymują jedną wartość, a ciemniejsze drugą.



# Progowanie



Próg: 50



Próg: 100



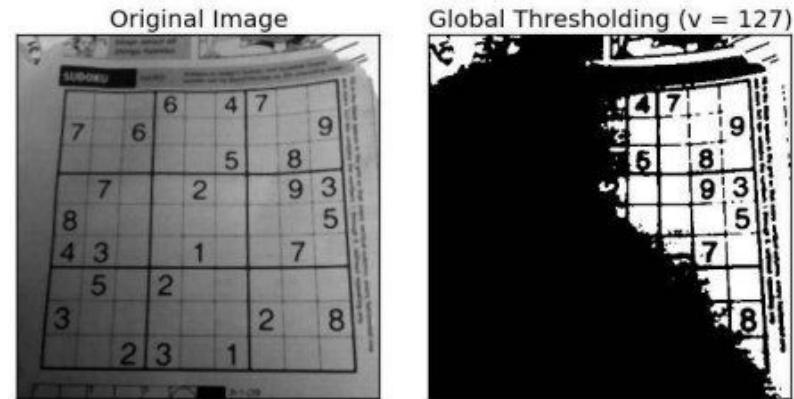
Próg: 150



Próg: 200

# Progowanie adaptacyjne

W poprzedniej sekcji jako progu użyliśmy jednej wartości globalnej. Nie zawsze to dobry pomysł np. jeśli obraz ma różne warunki oświetlenia w różnych obszarach.



W takim przypadku może pomóc adaptacyjne progowanie. Próg dla piksela jest ustalany na podstawie małego obszaru wokół niego. Otrzymujemy więc różne progi dla różnych regionów tego samego obrazu.

# Progowanie adaptacyjne

Adaptive Mean Thresholding



Adaptive Gaussian Thresholding



```
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

def threshold(image):
    @interact(x=widgets.IntSlider(min=0, max=255, step=1, value=100))
    def trackbar(x):
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        _, thresh1 = cv2.threshold(gray_img,x,255,cv2.THRESH_BINARY)
        cv2_imshow(thresh1)
        return x
    threshold(img)
```

# Progowanie

```
def adaptive_threshold(image):
    @interact(x=widgets.IntSlider(min=3, max=15, step=2, value=9), y=True)
def trackbar(x, y):
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

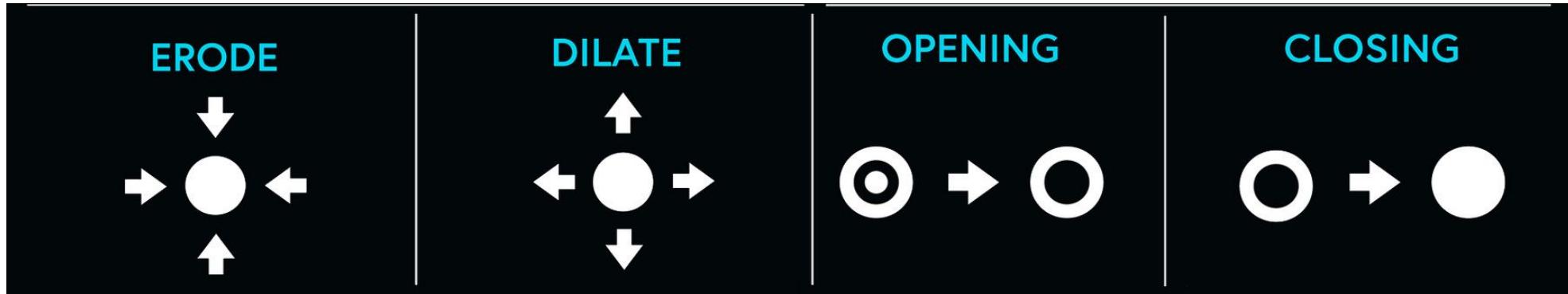
    if y:
        adaptive_method = cv2.ADAPTIVE_THRESH_MEAN_C
    else:
        adaptive_method = cv2.ADAPTIVE_THRESH_GAUSSIAN_C

    thresh1 = cv2.adaptiveThreshold(gray_img, 255, adaptive_method,
                                    cv2.THRESH_BINARY, x, 2)
    cv2_imshow(thresh1)
    return x
adaptive_threshold(img)
```

# Progowanie Adaptacyjne

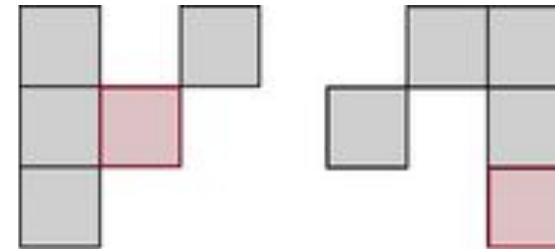
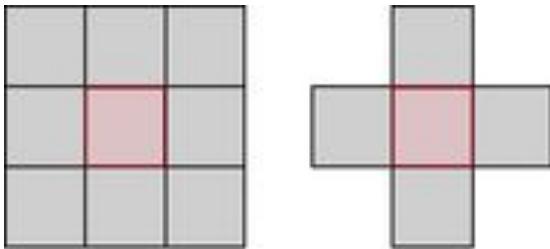
# Operacje morfologiczne

- Są to operacje, które pozwalają na powiększenia lub zmniejszenia obszarów obrazu, a także usunięcia lub wypełnienia pikseli brzegowych jakiegoś obszaru obrazu.
- Z reguły raczej operują na czarno-białych obrazach.
- Wyróżniamy:
  - Erozję
  - Dylatację
  - Otwarcie
  - Zamknięcie



# Element strukturalny

Z pojęciem operacji morfologicznych związany jest również element strukturalny. Składa się on z punktu centralnego oraz obszaru, który jest wokół niego kontrolowany. W Pythonie jest reprezentowany podobnie jak kernel opisany przy okazji filtrów konwolucyjnych. Tutaj jednak algorytm działania jest nieco inny.



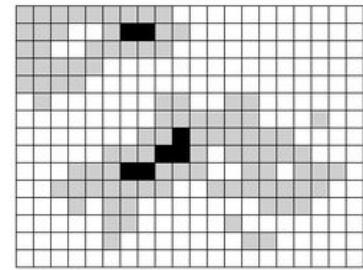
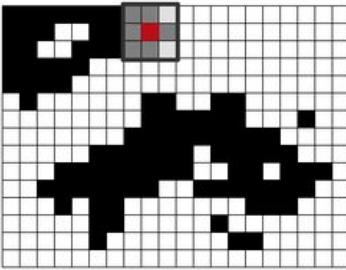
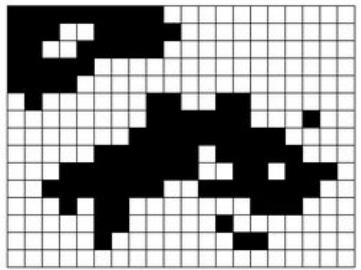
# Algorytm

Ogólny algorytm przekształcenia morfologicznego:

- element strukturalny jest przemieszczany po całym obrazie i dla każdego punktu obrazu dokonywane jest sprawdzanie, czy element strukturalny pasuje/bądź nie
- w przypadku wykrycia zgodności wzorca pikseli obrazu i szablonu elementu strukturalnego następuje wykonanie pewnej operacji na badanym punkcie

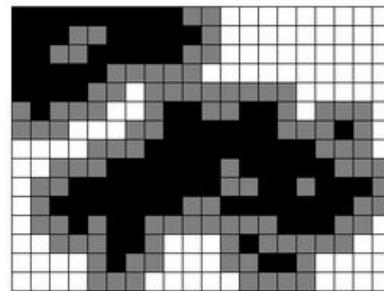
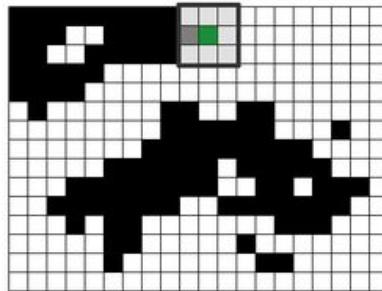
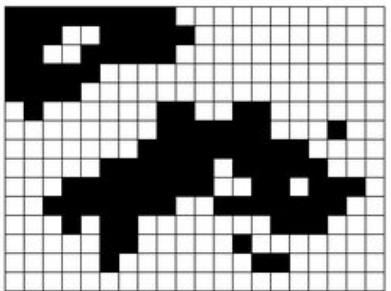
# Erode (erozja)

Operacja ta przykłada element strukturalny do każdego piksela na obrazie. Jeżeli choć jeden piksel z sąsiedztwa objętego elementem ma wartość równą zero, punkt centralny również otrzymuje wartość zero. W przeciwnym wypadku jego wartość nie ulega zmianie.



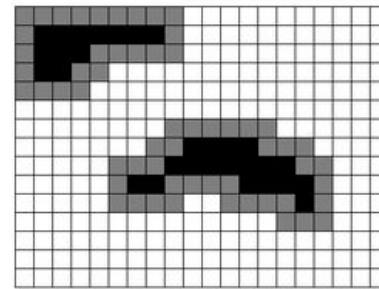
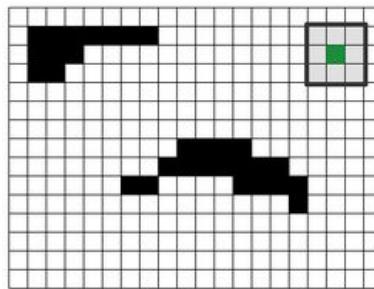
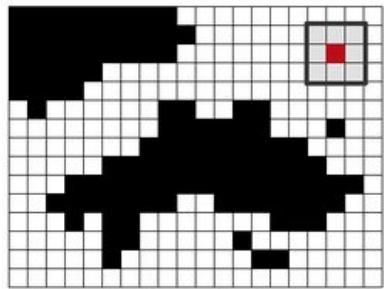
# Dilation (Dylatacja)

Operacja ta przykłada element strukturalny do każdego piksela na obrazie. Jeżeli choć jeden piksel z sąsiedztwa objętego elementem ma wartość równą 1, punkt centralny również otrzymuje wartość 1.



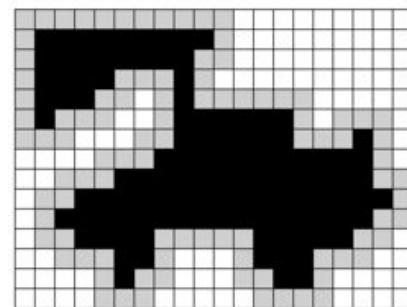
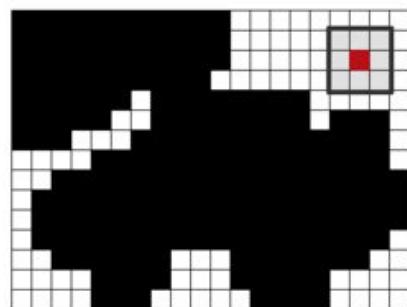
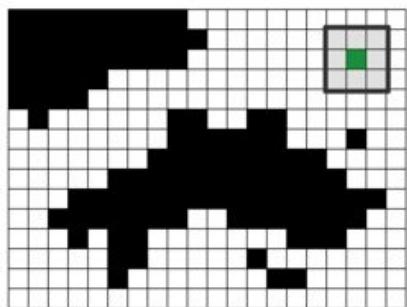
# Open (Otwarcie)

Operacja ta składa się z **erozji** i następującej po niej **dylatacji**. Służy np. do uzupełnienia braków w niezamkniętych pierścieniach.



# Closing (Dokmnięcie)

Operacja ta składa się z **dylatacji** i następującej po niej **erozji**. Służy np. wypełnianiu braków.



# Porównanie:

Sobel +



Erode



Dilation



Opening



Closing



```

def morphological_operation(image):

    def opening(img, kernel):
        return cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

    def closing(img, kernel):
        return cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

    @interact(operation=['Erosion', 'Dilation', 'Opening', 'Closing'],
              struct_el=['MORPH_RECT', 'MORPH_ELLIPSE', 'MORPH_CROSS'],
              size = widgets.IntSlider(min=3, max=15, step=2, value=5))
    def trackbar(operation, struct_el, size):
        operation_dict = {'Erosion': cv2.erode, 'Dilation': cv2.dilate,
                          'Opening': opening, 'Closing': closing}
        struct_el_dict = {'MORPH_RECT':
                           cv2.getStructuringElement(cv2.MORPH_RECT, (size, size)),
                           'MORPH_ELLIPSE':
                           cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (size, size)),
                           'MORPH_CROSS':
                           cv2.getStructuringElement(cv2.MORPH_CROSS, (size, size))}

        result = operation_dict[operation](image, struct_el_dict[struct_el])
        cv2_imshow(result)

    #using sobel output form above
    sobel[sobel>30] = 255 # add threshold
    cv2_imshow(sobel)
    morphological_operation(sobel)

```

# Operacje morfologiczne

# Proces widzenia maszynowego

Czyli plan na resztę dnia

Raw image – Surowy obraz



Enhanced Image – Ulepszony obraz

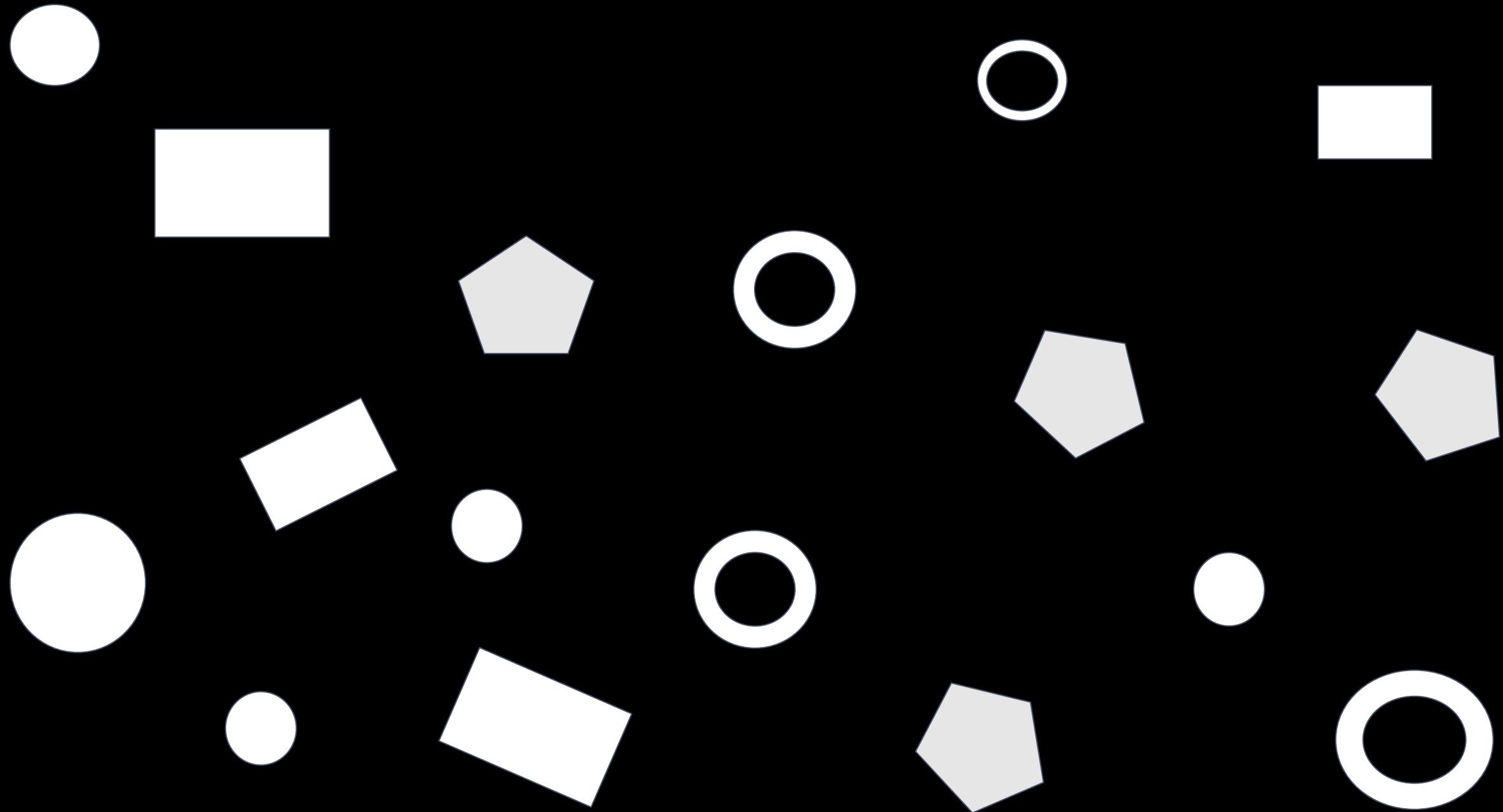


Image preprocessing – wstępne przetwarzanie obrazu

Features – cechy obrazu

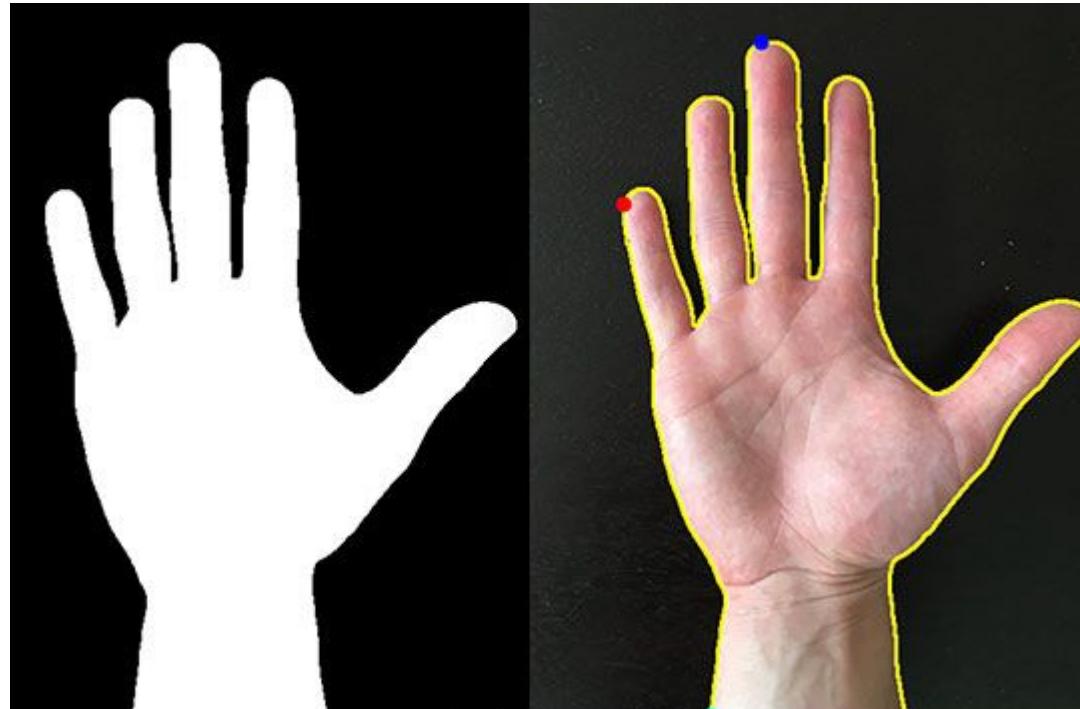
Q	A	S	D	F	G	H
1	0	2	3	7	8	6
2	4	5	3	2	1	4
6	8	9	7	5	1	3
2	1	3	3	5	4	3
6	8	7	8	9	1	3
7	4	7	0	9	2	3

Feature Extraction – ekstrakcja cech



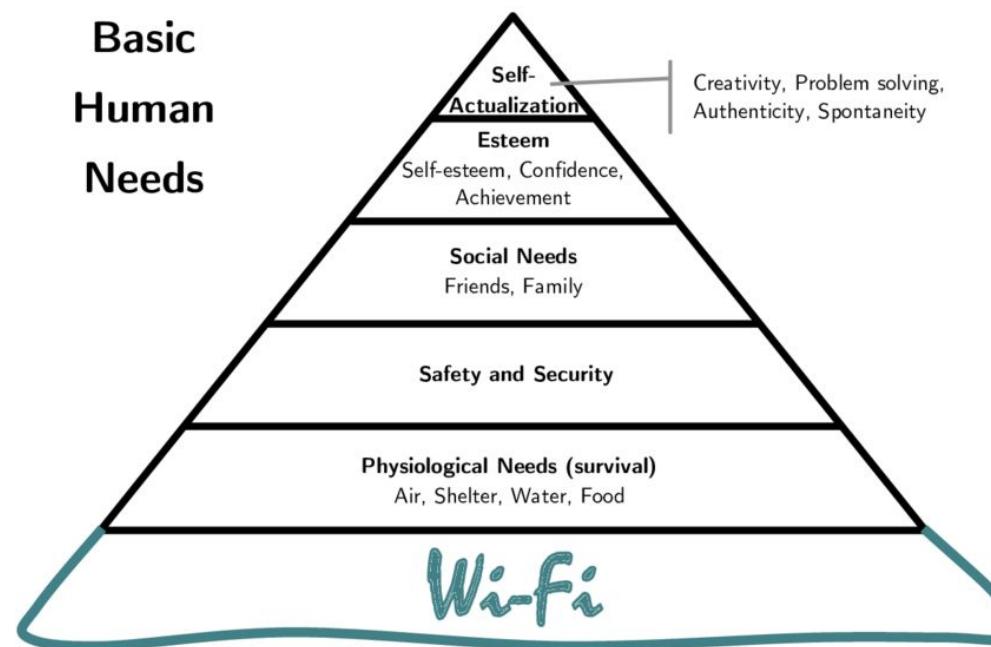
# Contours (kontury)

Kontur to po prostu krzywa łącząca wszystkie ciągłe piksele (wzdłuż granicy), mające tą samą wartość. Kontury są użytecznym narzędziem do analizy kształtów oraz wykrywania i rozpoznawania obiektów.



# Hierarchia

Czasami niektóre kontury znajdują się wewnętrznych innych konturów (zagnieżdżone figury). W tym przypadku zewnętrzny kontur nazywamy rodzicem, a wewnętrzny dzieckiem. Python umożliwia zapis tych relacji, co może być przydatne w identyfikacji obiektów.



# Reprezentacja hierarchii w OpenCV

Każdy kontur posiada informację o tym jaka jest hierarchia, kto jest jego dzieckiem, kto jest jego rodzicem. OpenCV reprezentuje hierarchię jako tablicę czterech wartości:

## [Next, Previous, First\_Child, Parent]

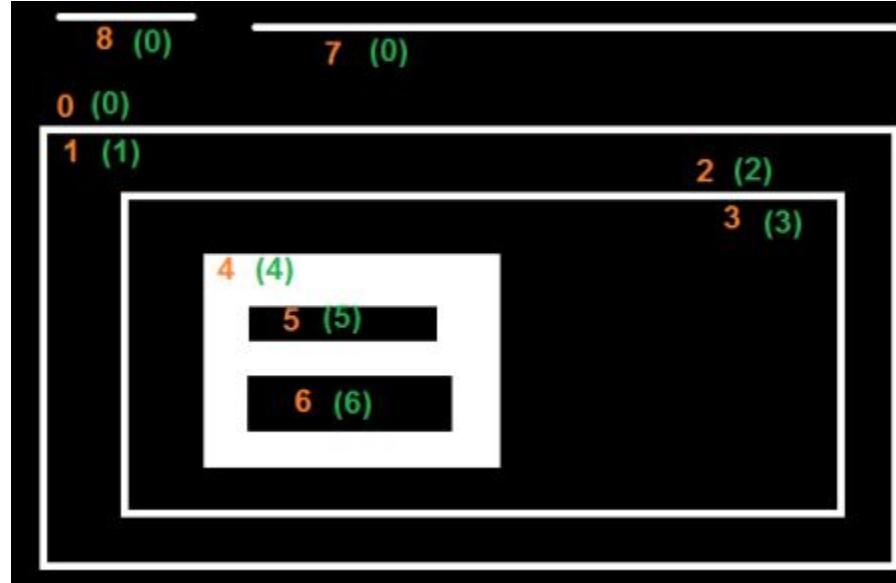
- Next oznacza następny kontur na tym samym poziomie hierarchii.
- Previous oznacza poprzedni kontur na tym samym poziomie hierarchii.
- First\_Child oznacza swój pierwszy kontur potomny.
- Parent oznacza indeks konturu rodzica.

Jeżeli kontur nie posiada rodzica bądź dziecka, to wartość wynosi -1.

# Hierarchia drzewiasta

Najpopularniejszym sposobem przedstawiania hierarchii konturów jest za pomocą struktury drzewa. Index jest indexem konturu w liście.

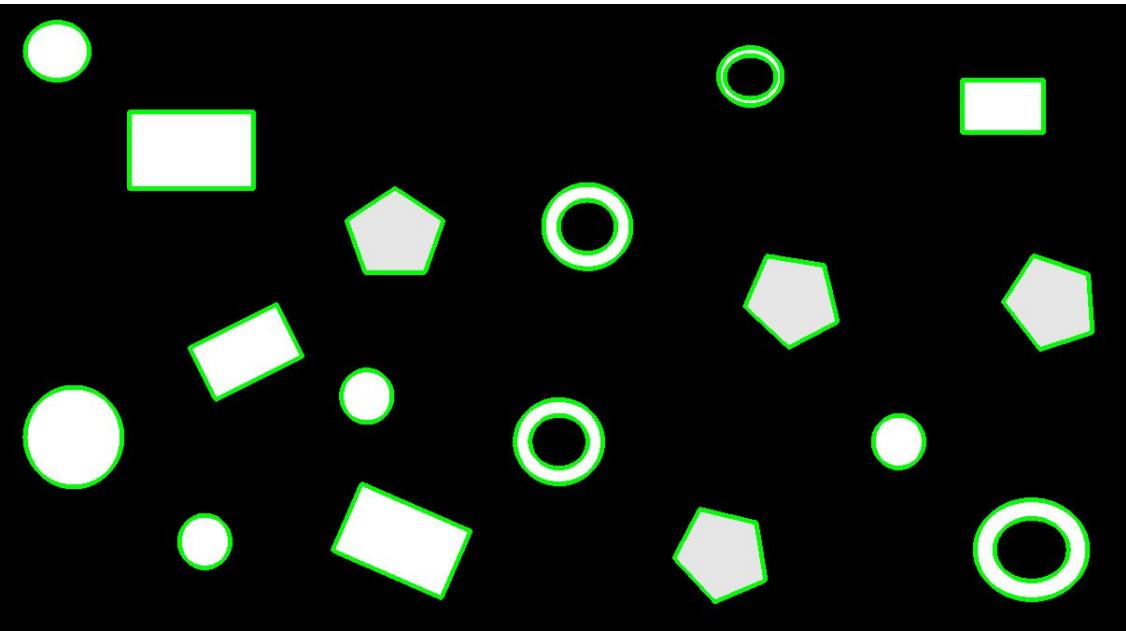
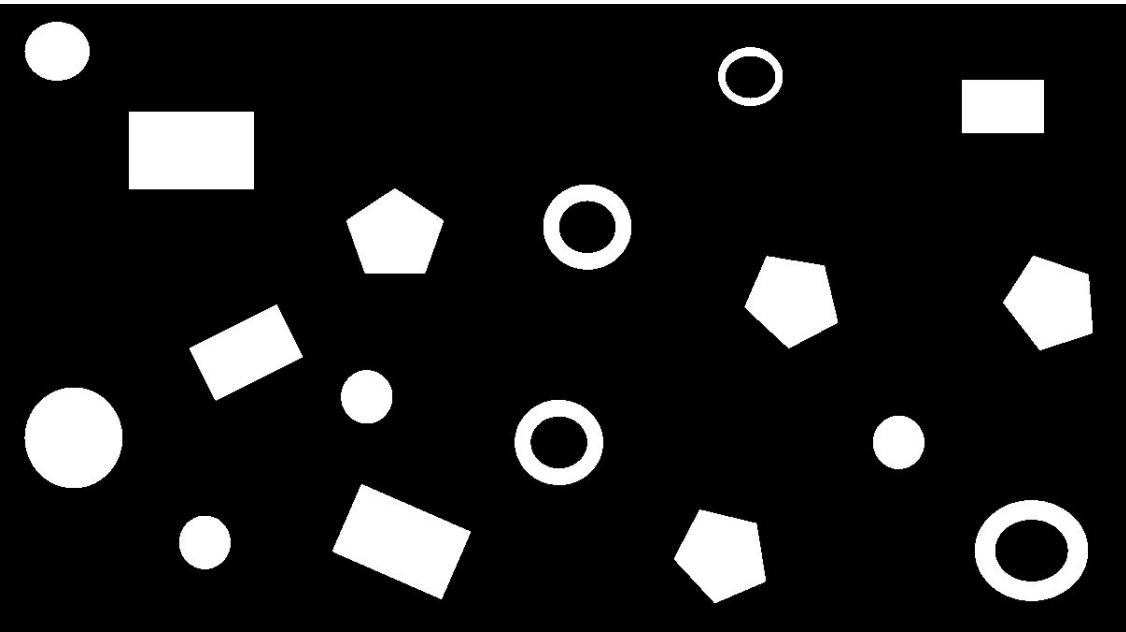
```
array([[[ 7, -1, 1, -1],  
       [-1, -1, 2, 0],  
       [-1, -1, 3, 1],  
       [-1, -1, 4, 2],  
       [-1, -1, 5, 3],  
       [6, -1, -1, 4],  
       [-1, 5, -1, 4],  
       [8, 0, -1, -1],  
       [-1, 7, -1, -1]]])
```



# Kontury

## Hierarchia

```
[[[ 1 -1 -1 -1]
 [ 2  0 -1 -1]
 [ 4  1  3 -1]
 [-1 -1 -1  2]
 [ 5  2 -1 -1]
 [ 6  4 -1 -1]
 [ 8  5  7 -1]
 [-1 -1 -1  6]
 [ 9  6 -1 -1]
 [10  8 -1 -1]
 [11  9 -1 -1]
 [12 10 -1 -1]
 [13 11 -1 -1]
 [14 12 -1 -1]
 [16 13 15 -1]
 [-1 -1 -1 14]
 [17 14 -1 -1]
 [18 16 -1 -1]
 [20 17 19 -1]
 [-1 -1 -1 18]
 [-118 -1 -1]]]
```



# Momenty obrazu

Średnia ważona intensywności pikseli obrazu. W przypadku obrazów czarno-białych po prostu suma białych pikseli.

$$M = \sum_x \sum_y I(x, y)$$

Nie jest to żadne rocket science, ale należy zauważyć, że momenty dla obrazu (1), (2) będą podobne i różne od (3). Co sugeruje potencjał do klasyfikacji obiektów.

(1)

(2)

(2)



# Momenty obrazu

Pełna potęga momentów przejawia się w Hu Moments. Pozwalają one liczyć cechy, które są niezależne od przemieszczenia obiektu, skali czy obrotu. Co pozwala na identyfikację obiektów i ich klasyfikację. Ich wartości wylicza się na podstawie momentów centralnych:

$$h_0 = \eta_{20} + \eta_{02}$$

$$h_1 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

$$h_2 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$h_3 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$h_4 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$h_5 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})]$$

$$h_6 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

# Circularity (okrągłość)

Dodatkową cechą, którą warto pozyskać z konturu jest jego okrągłość.

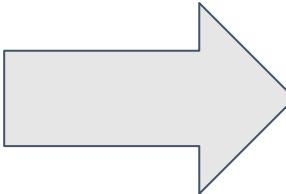
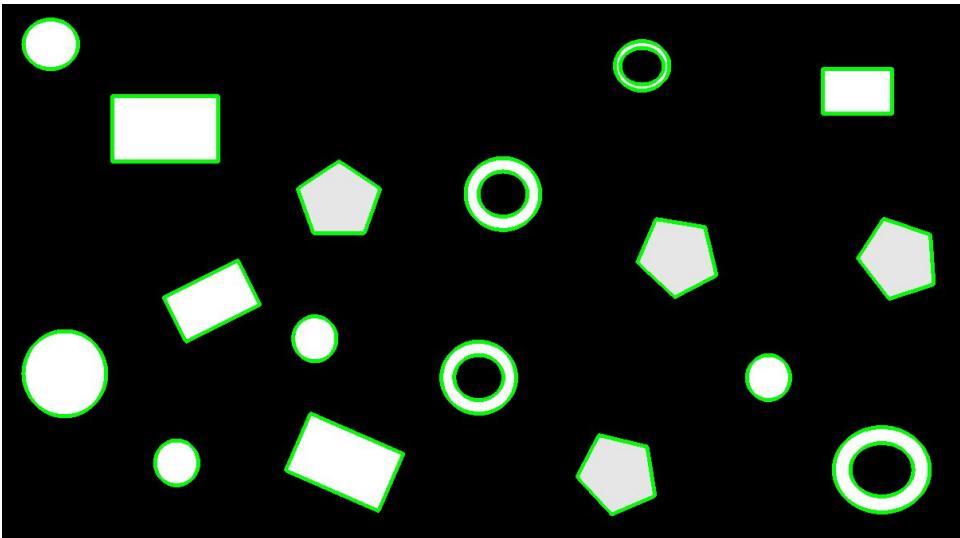
Definiowana jako :

$$\frac{4 * \pi * \text{ContourArea}}{\text{ArcLength}^2}$$

gdzie:

- ContourArea to ilość pikseli znajdująca się wewnątrz konturu
- ArcLength to długość konturu

# Cechy wydobyte!



	0	1	2	3	4	5	6	circularity	has_child
0	0.16	0.00	0.00	0.00	-0.00	-0.00	0.00	0.90	0
1	0.16	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.78	0
2	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.90	1
3	0.19	0.01	0.00	0.00	-0.00	-0.00	0.00	0.64	0
4	0.16	0.00	0.00	0.00	0.00	0.00	-0.00	0.90	0
5	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.90	1
6	0.16	0.00	0.00	0.00	-0.00	-0.00	0.00	0.90	0
7	0.16	0.00	0.00	0.00	-0.00	-0.00	0.00	0.90	0
8	0.19	0.01	0.00	0.00	0.00	-0.00	0.00	0.64	0
9	0.16	0.00	0.00	0.00	-0.00	0.00	-0.00	0.78	0
10	0.16	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.77	0
11	0.16	0.00	0.00	0.00	0.00	-0.00	-0.00	0.78	0
12	0.16	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.90	1
13	0.19	0.01	0.00	0.00	-0.00	0.00	-0.00	0.74	0
14	0.18	0.01	0.00	0.00	0.00	0.00	0.00	0.75	0
15	0.16	0.00	0.00	0.00	0.00	-0.00	0.00	0.90	1
16	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.89	0

[https://en.wikipedia.org/wiki/Central\\_moment](https://en.wikipedia.org/wiki/Central_moment)

```
gray_shapes = cv2.cvtColor(shapes, cv2.COLOR_BGR2GRAY)
_, thresh1 = cv2.threshold(gray_shapes, 50, 255, cv2.THRESH_BINARY)
cv2.imshow(thresh1)

contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
img_with_contours = cv2.drawContours(shapes, contours, -1, (0,255,0), 3)
cv2.imshow(img_with_contours)
print(hierarchy)

# Get rid of internal contours, extract information about childs

filtered_contours = []
has_a_child = []

for contour, h in zip(contours, hierarchy[0]):
    if h[3] == -1:
        filtered_contours.append(contour)
        if h[2] != -1:
            has_a_child.append(1)
        else:
            has_a_child.append(0)

print(has_a_child)
```

# Threshold i wydobycie konturów

```
# Get all usefull information from contours
features_list = []

for contour, childs in zip(filtered_contours, has_a_child):
    moments = cv2.moments(contour)
    hu_moments = cv2.HuMoments(moments)
    features = {x: hu[0] for x, hu in enumerate(hu_moments)}
    contour_area = cv2.contourArea(contour)
    arc_length = cv2.arcLength(contour, True)
    features['circularity'] = 4 * 3.14 * contour_area / arc_length**2
    features['has_child'] = childs
    features_list.append(features)

# Save data as pd.DataFrame
import pandas as pd

pd.options.display.float_format = "{:, .2f}".format
df = pd.DataFrame(features_list)
df
```

## Wydobycie cech konturów i zapis do df

# Ostatni krok do predykcji

Raw image - Surowy obraz



Enhanced Image - Ulepszony obraz



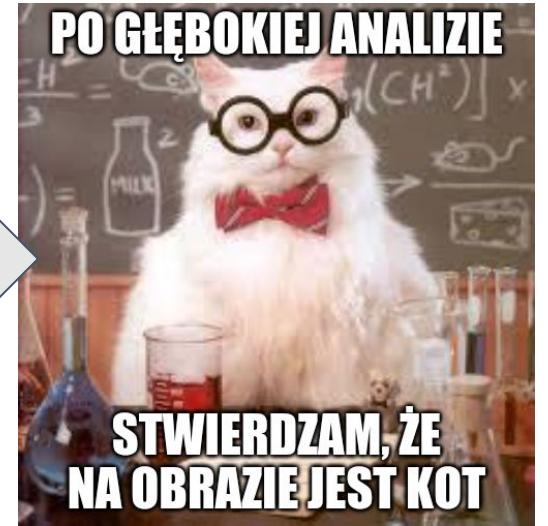
Image preprocessing - wstępne przetwarzanie obrazu

Features - cechy obrazu

Q	A	S	D	F	G	H
1	0	2	3	7	8	6
2	4	5	3	2	1	4
6	8	9	7	5	1	3
2	1	3	3	5	4	3
6	8	7	8	9	1	3
7	4	7	0	9	2	3

Feature Extraction - ekstrakcja cech

Prediction - Predykcja

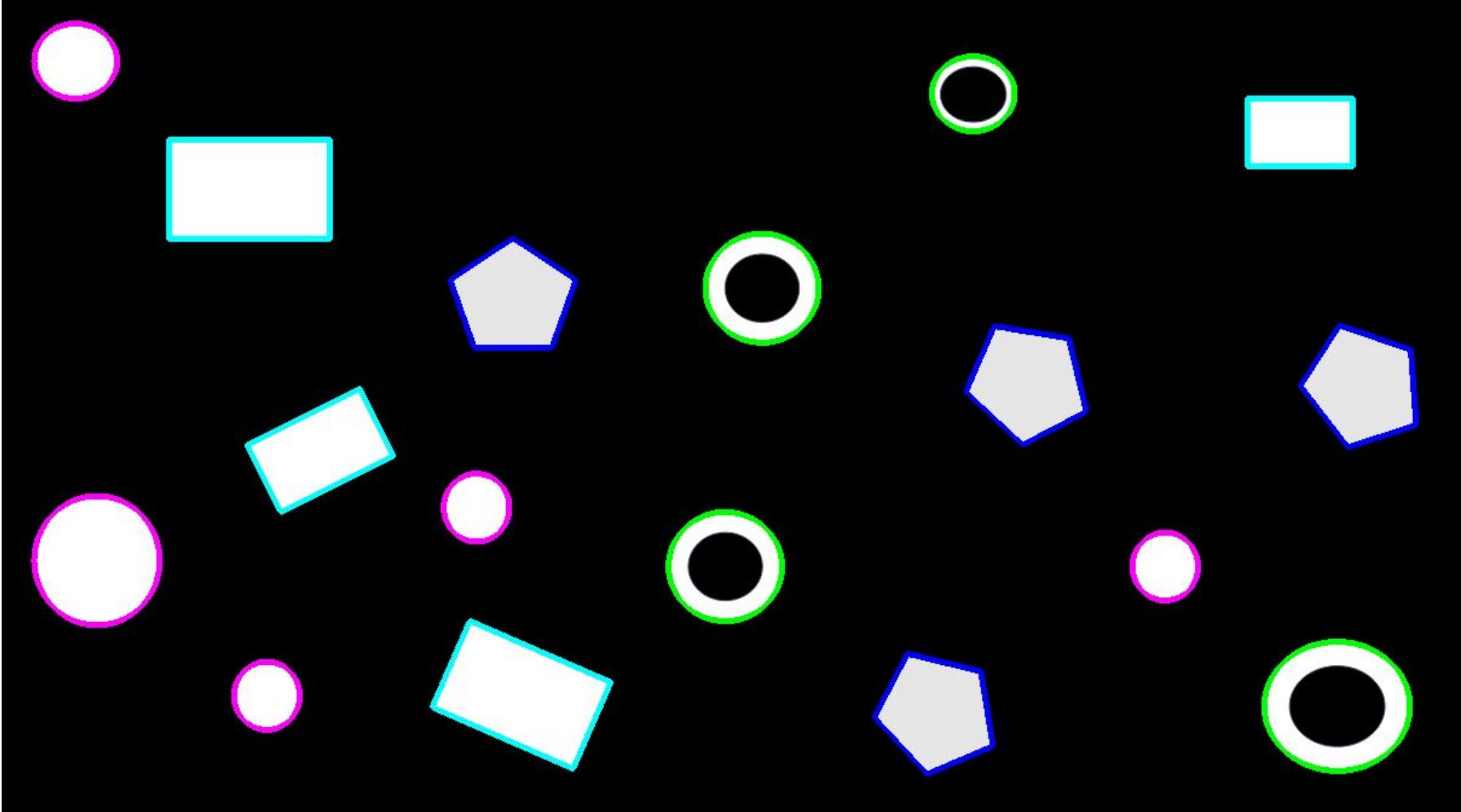


Classification - klasyfikacja  
(przykładowe zadanie)

# Co możemy zrobić z cechami?

- Nie pozostało nic innego jak wykorzystać zebrane dane do predykcji.
- Niestety/stety mamy do czynienia z klasycznym przypadkiem uczenia nienadzorowanego -> nie znamy klas obiektów, a samych danych nie ma zbyt wiele.
- Jednak algorytm K-Means powinien sobie spokojnie poradzić.

# Klasteryzacja



```
from sklearn.cluster import KMeans
import numpy as np

from sklearn import preprocessing

x = df.values #returns a numpy array
normalizer = preprocessing.Normalizer()
x_scaled = normalizer.fit_transform(x)
df = pd.DataFrame(x_scaled)

kmeans = KMeans(n_clusters=4, random_state=0).fit(df)
print(kmeans.labels_)

colors = [(255,0,0), (255,255,0), (255,0,255), (0,255,0)]
for contour, label in zip(filtered_contours, kmeans.labels_):
    classified = cv2.drawContours(shapes, [contour], 0, colors[label], 3)
print(df)
cv2_imshow(classified)
```

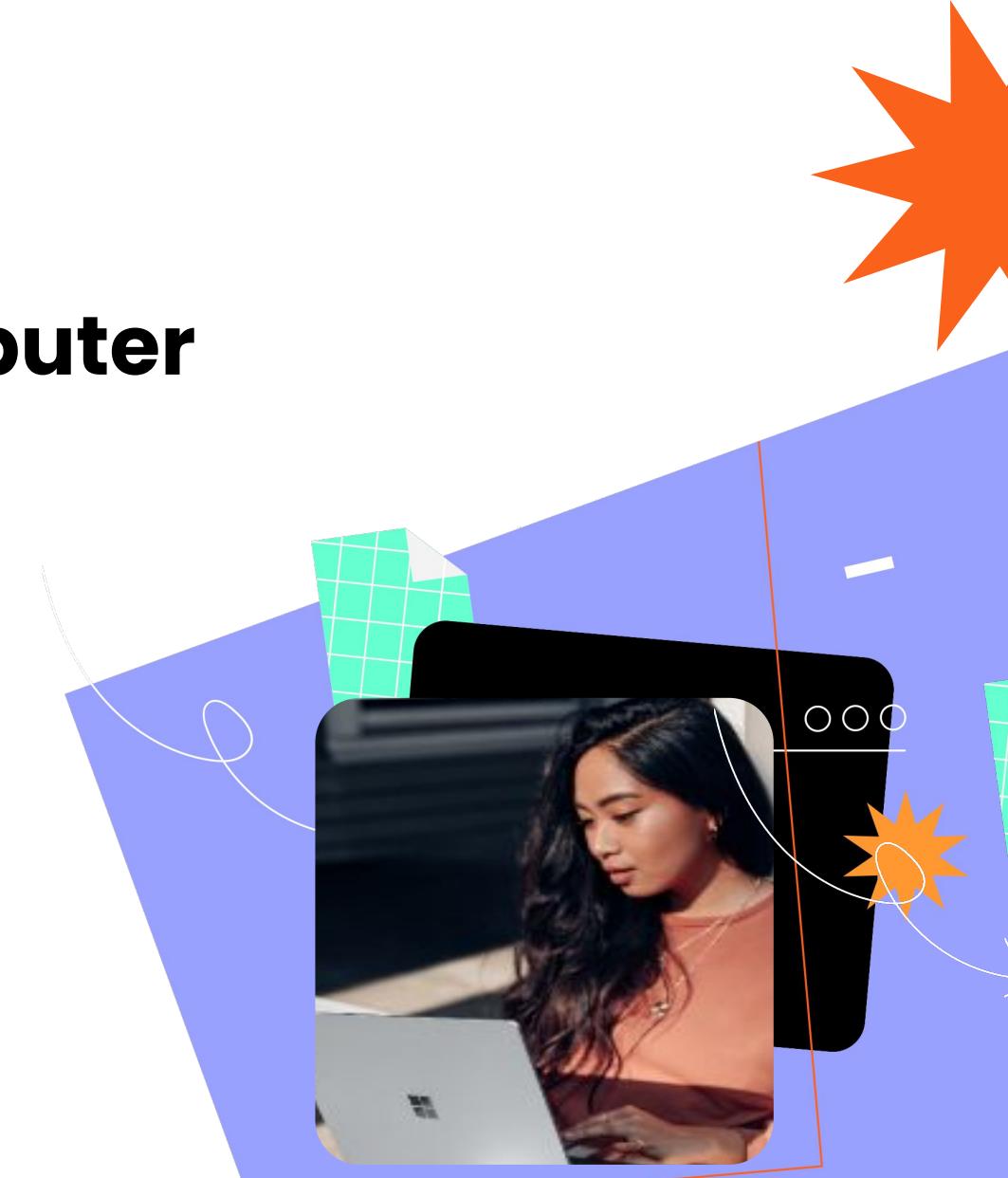
# K-means!

# Koniec dnia pierwszego. Linki:

- <https://opencv.org/>
- Tradycyjne spojrzenie na CV czy właśnie konwencjonalne?
  - <https://arxiv.org/pdf/1910.13796.pdf>
- Poszerz horyzonty:
  - <https://learnopencv.com/otsu-thresholding-with-opencv/>
  - <https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549>
  - [https://opencv-python-tutorials.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_canny/py\\_canny.html](https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html)
  - <https://learnopencv.com/histogram-of-oriented-gradients/>
  - <https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/2013/CS373-IP-04.pdf>

# **Przetwarzanie obrazu – Computer Vision**

**Kurs Data Science – Dzień 2**



# Plan prezentacji:

- Konwolucyjne sieci Neuronowe (CNN)
  - Konwolucja
  - Budowa CNN
  - Pooling, Stride, Padding, Fully Connected
  - Backward propagation
  - Batch Normalization
  - Architektury CNN
- Klasyfikator obrazków z wykorzystaniem warstw konwolucyjnych

# Konwencjonalne - podsumowanie

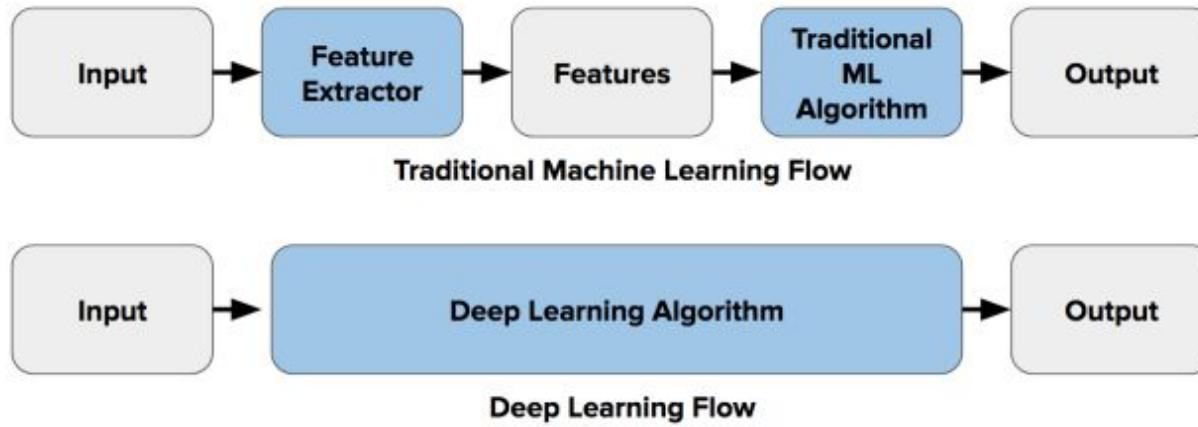
W podejściu konwencjonalnym ręcznie wydobywaliśmy cechy z badanego obrazu i dopiero wtedy wykorzystywaliśmy model, aby coś z tych danych predykować. Można było napotkać następujące problemy:

- Jakich filtrów użyć, aby wydobyć porządkane cechy z obrazu?
- Jak zmieni się działanie algorytmu w przypadku innego oświetlenia/warunków zewnętrznych?

# Podejście konwencjonalne vs DeepLearning

A gdyby tak algorytm sam się nauczył, jakie cechy ma wydobywać i w jaki sposób?

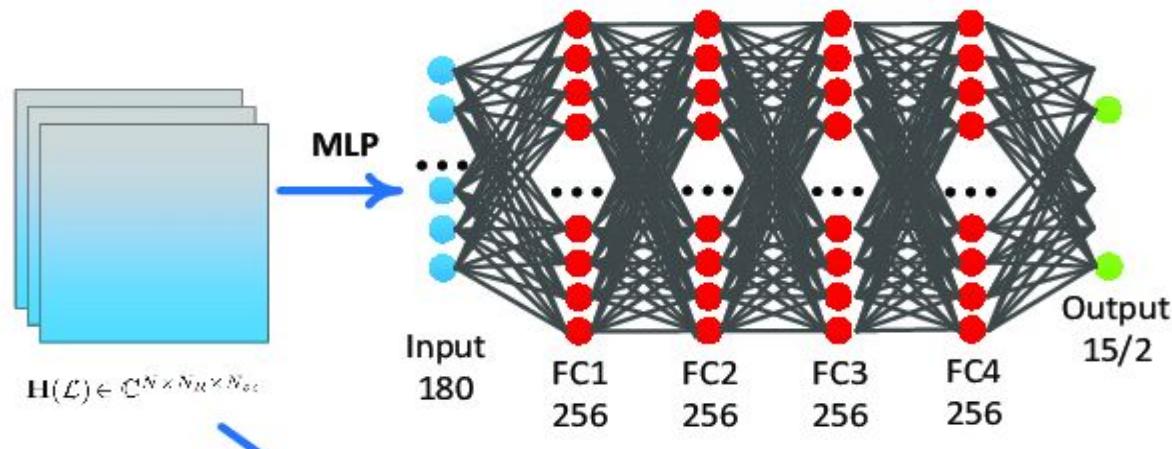
Podejście deeplearningowe zakłada właśnie taki scenariusz!



# Pierwszy pomysł! A gdyby użyć MLP?

W teorii nic nie stoi na przeszkodzie by wykorzystać Multi layer perceptron do analizy obrazu. Każdy pojedynczy pixel może stać się jednym wejściem do sieci. Co w przypadku prostych zbiorów danych (takich jak np. MNIST) może odnieść pozytywne efekty - nawet 98 % accuracy:

<https://share.cocalc.com/share/32b94ee413d02759d719862907bb0a85a76c43f1/2016-11-07-175929.pdf>



```

import numpy as np
from matplotlib import pyplot as plt

from keras.utils import to_categorical

(train_x, train_y), (test_x, test_y) = keras.datasets.mnist.load_data()

print('Training data shape : ', train_x.shape, train_y.shape)
print('Testing data shape : ', test_x.shape, test_y.shape)

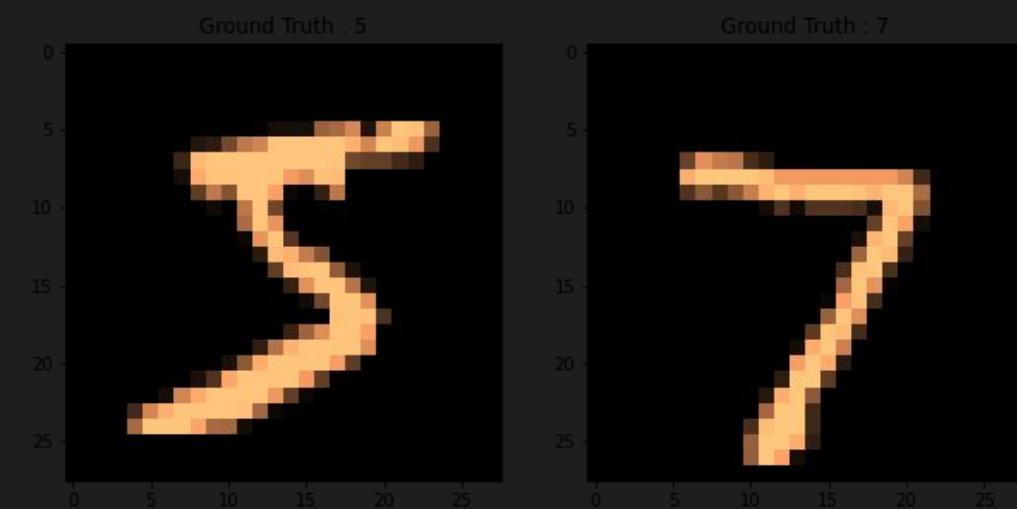
# Find the unique numbers from the train labels
classes = np.unique(train_y)
classes_num = len(classes)
print('Total number of outputs : ', classes_num)
print('Output classes : ', classes)

plt.figure(figsize=[10,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_x[0,:,:], cmap='copper')
plt.title("Ground Truth : {}".format(train_y[0]))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_x[0,:,:], cmap='copper')
plt.title("Ground Truth : {}".format(test_y[0]))

```



```
# Change from matrix to array of dimension 28x28 to array of dimension 784

train_x = train_x.reshape(train_x.shape[0], -1)
test_x = test_x.reshape(test_x.shape[0], -1)

# 0-255 to 0-1
train_x = train_x/255
test_x = test_x/255

# Change the labels from integer to categorical data
train_y_one_hot = keras.utils.to_categorical(train_y)
test_y_one_hot = keras.utils.to_categorical(test_y)
```

# Image processing

# Model i uczenie

```
# Make a model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(train_x.shape[1],)))
model.add(Dense(512, activation='relu'))
model.add(Dense(classes_num, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_x, train_y_one_hot,
                     batch_size=256, epochs=10, verbose=True,
                     validation_data=(test_x, test_y_one_hot))

[test_loss, test_acc] = model.evaluate(test_x, test_y_one_hot)
print(f"Evaluation result on Test Data : Loss = {test_loss}, accuracy = {test_acc}")
```

# Model i uczenie

```
# Make a model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(train_x.shape[1],)))
model.add(Dense(512, activation='relu'))
model.add(Dense(classes_num, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_x, train_y_one_hot,
                     batch_size=256, epochs=10, verbose=True,
                     validation_data=(test_x, test_y_one_hot))

[test_loss, test_acc] = model.evaluate(test_x, test_y_one_hot)
print(f"Evaluation result on Test Data : Loss = {test_loss}, accuracy = {test_acc}")
```

```
def plot_history(history):  
    #Plot the Loss Curves  
    plt.figure(figsize=[8,6])  
    plt.plot(history.history['loss'], 'r', linewidth=3.0)  
    plt.plot(history.history['val_loss'], 'b', linewidth=3.0)  
    plt.legend(['Training loss', 'Validation Loss'], fontsize=18)  
    plt.xlabel('Epochs ', fontsize=16)  
    plt.ylabel('Loss', fontsize=16)  
    plt.title('Loss Curves', fontsize=16)  
  
    #Plot the Accuracy Curves  
    plt.figure(figsize=[8,6])  
    plt.plot(history.history['accuracy'], 'r', linewidth=3.0)  
  
    plt.plot(history.history['val_accuracy'], 'b', linewidth=3.0)  
    plt.legend(['Training Accuracy', 'Validation Accuracy'], fontsize=18)  
    plt.xlabel('Epochs ', fontsize=16)  
    plt.ylabel('Accuracy', fontsize=16)  
    plt.title('Accuracy Curves', fontsize=16)  
  
plot_history(history)
```

# Wyświetlanie rezultatów uczenia

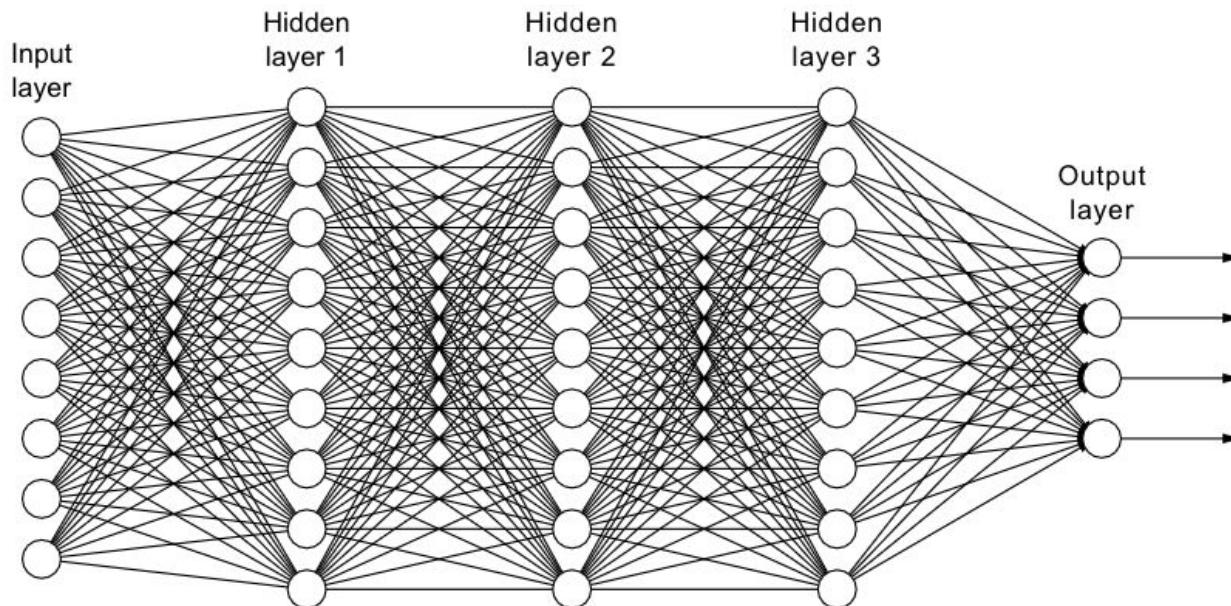
Cifar10

## Ograniczenia MLP



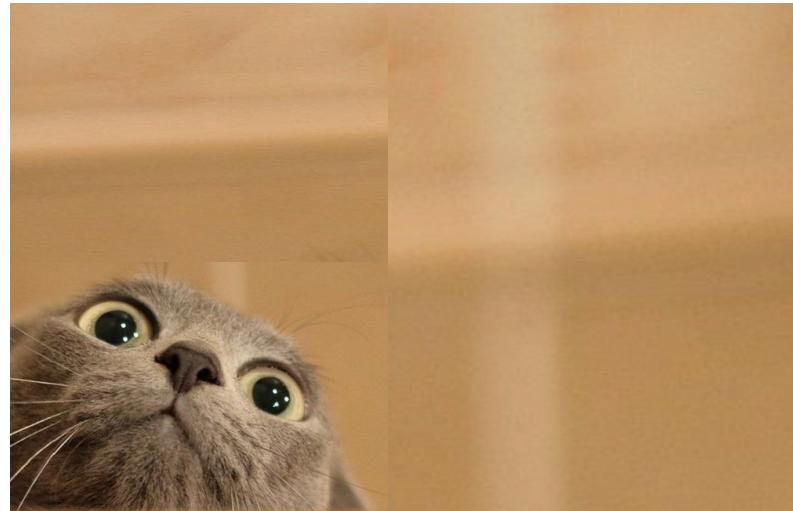
# Problemy z MLP do CV

- Każde wyjście jest przetwarzane przez jeden neuron, a każda następna warstwa jest w pełni połączona - ilość wag szybko staje się niemożliwa do przetwarzania, szczególnie w przypadku dużych obrazów. Nawet przy ograniczeniu warstw taka sieć traci zdolność do generalizacji i szybko się przeucza.



# Problemy z MLP do CV

- MLP również nie potrafi wychwytywać jakichkolwiek przesunięć obiektu w przestrzeni obrazu. Obiekt, który nagle znajduje się w innym miejscu na obrazie niż zazwyczaj, staje się niemożliwy do rozpoznania.



# Problemy z MLP do CV

- Jednak największym problemem MLP jest brak możliwości wychwycania jakichkolwiek przestrzennych zależności obiektów. Pixele są traktowane całkowicie niezależnie od siebie.

# Jak zatem analizować obrazy?

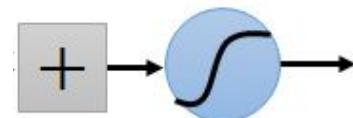
- Cechy i wzorce występujące na obrazie nie zajmują jego całości, są mniejsze i występują w różnych miejscach obrazu.
- Gdyby udało się je wykrywać przez małe detektory takich właśnie cech, a następnie klasyfikować, to można by było na ich podstawie podejmować predykcje co do zawartości obrazu. Jak mogłyby działać?

# Przykład:

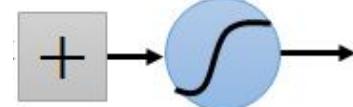
Jak miałoby to działać? Spróbujmy sobie zobrazować taki proces:



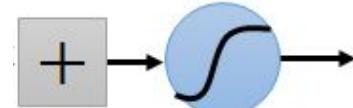
fang detector



beak detector

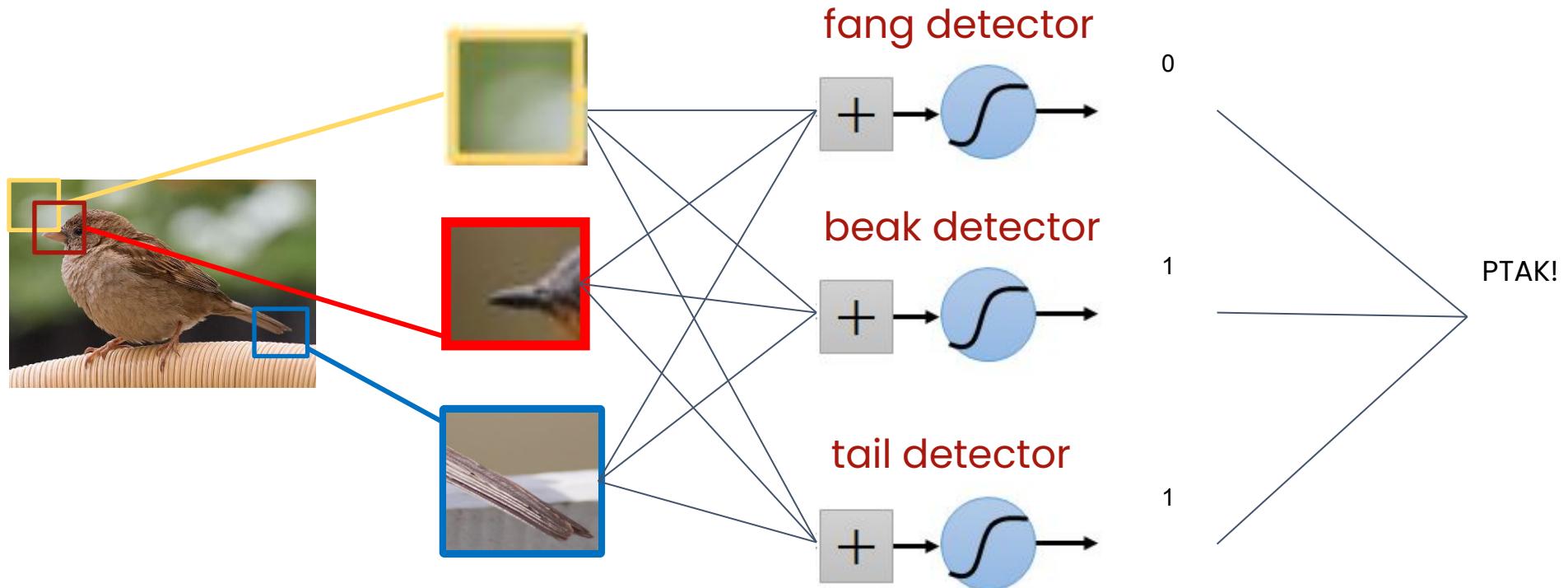


tail detector



# Przykład:

Jak miałoby to działać? Spróbujmy sobie zobrazować taki proces:



# Filtry Konwolucyjne

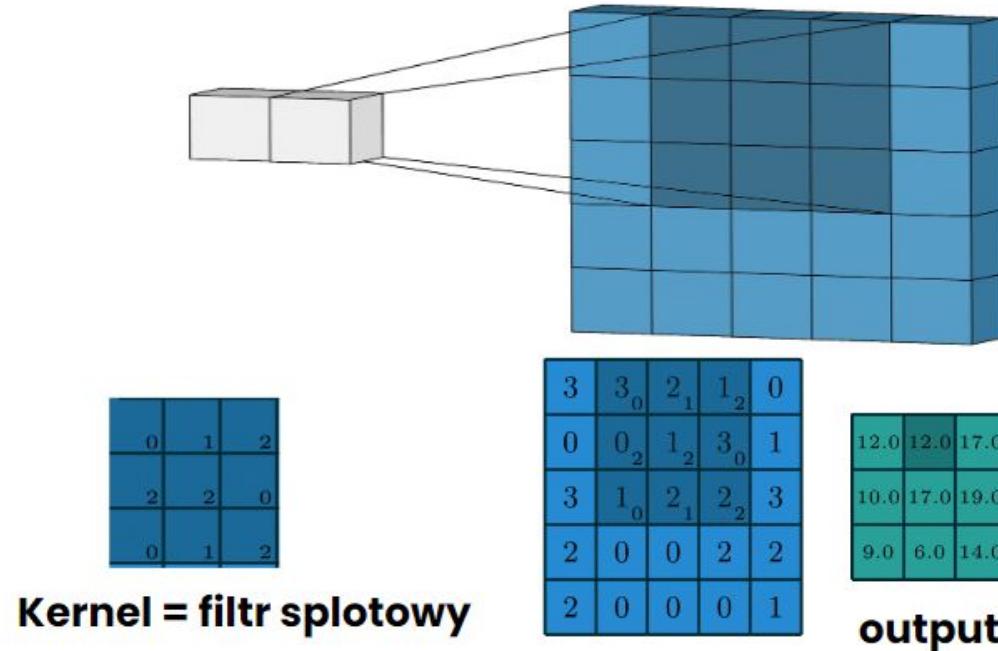
Proces wydobywania konkretnych cech z obrazu ... Ale przecież my to już znamy! (Patrz slajdy 38-42)

## **Tylko jak wykorzystać tutaj DeepLearning?**

A gdyby stworzyć losowe "filtry" (kernele), których wartości będą zmieniane w trakcie procesu nauki, tak aby wydobywały najważniejsze informacje z obrazu?

# Konwolucje!

Na tym pomyśle opiera się cały pomysł konwolucyjnych sieci neuronowych.



[https://aivision.pl/content/images/2020/03/1\\_Fw-ehcNBR9byHtho-Rxbtw.gif](https://aivision.pl/content/images/2020/03/1_Fw-ehcNBR9byHtho-Rxbtw.gif)  
[https://www.optyczne.pl/upload2/232821\\_splot\\_gif.gif](https://www.optyczne.pl/upload2/232821_splot_gif.gif)

# Padding

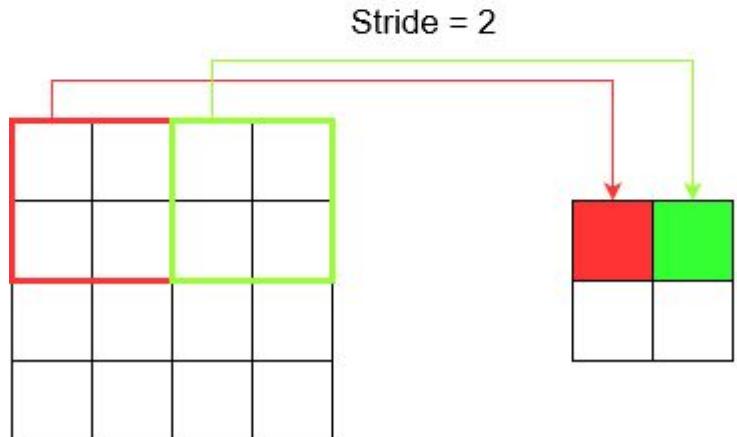
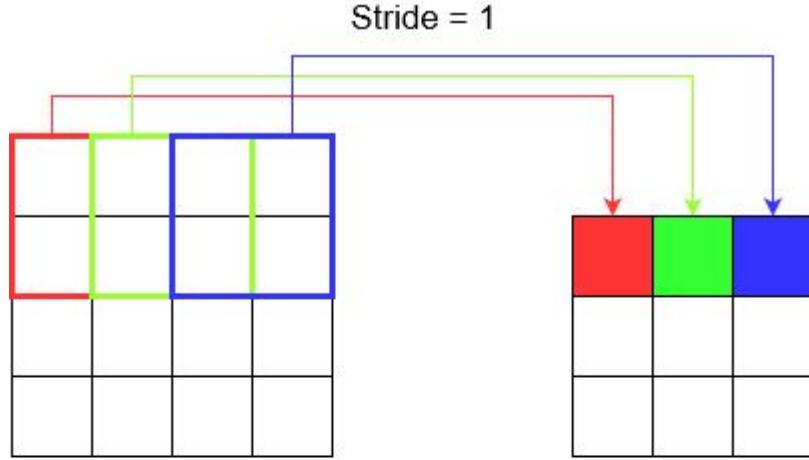
0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

Zero-padding added to image

Na rogach obrazu może być wiele ciekawych informacji, można jednak zauważyc, że krawędziowe pixele są znacznie rzadziej wykorzystywane w trakcie konwolucji. Tym samym ta informacja zanika.

Padding to dodatkowa warstwa, którą możemy dodać do obramowania obrazu. Padding = 1 oznacza dodanie 1 pixela do każdego z boków obrazu.

# Stride



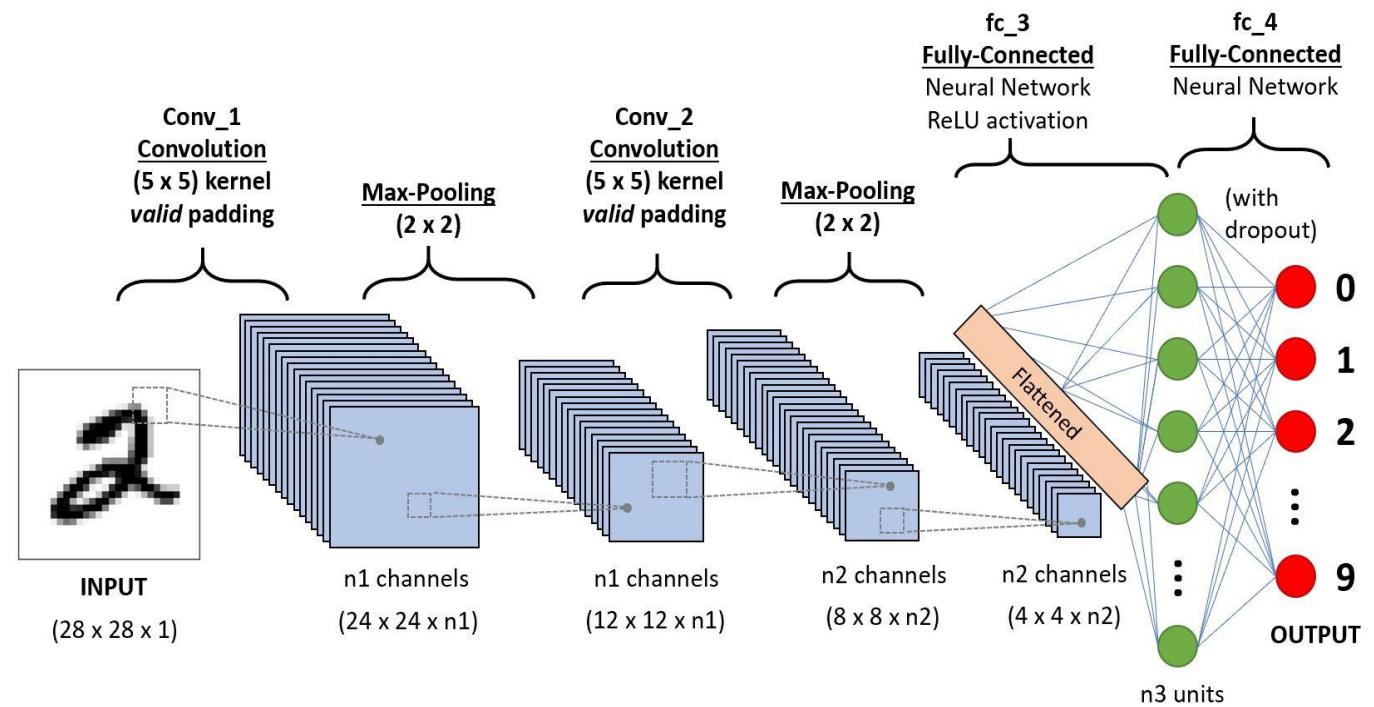
Stride to parametr filtru, który modyfikuje ilość ruchu po obrazie lub wideo. Na przykład, jeśli krok sieci neuronowej jest ustawiony na 2, filtr przesunie się o dwa piksele naraz.

Mожет быть использовано для уменьшения размера выхода при минимальной потере информации.

# Pełna struktura CNN

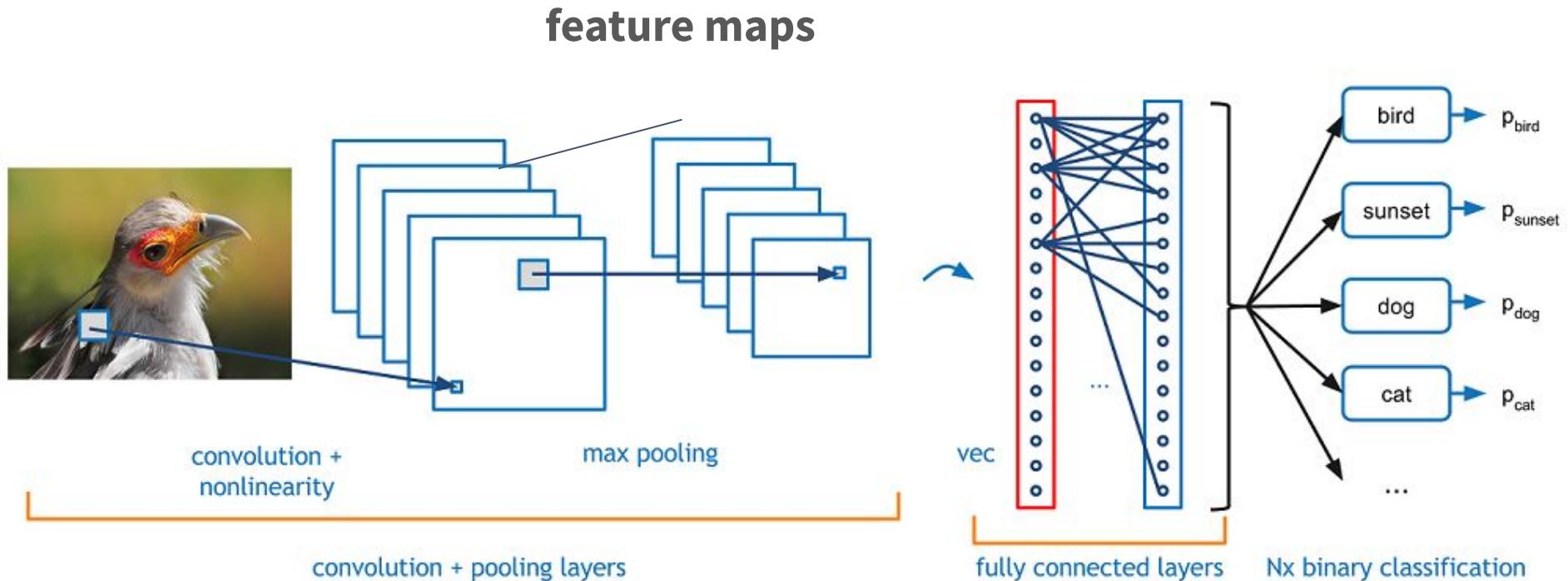
Pełna struktura nie składa się jedynie z konwolucji. CNN to:

- warstwy konwolucyjne
- warstwy pooling
- warstwy fully-connected



# Warstwy konwolucyjne

Ilość warstw feature maps będących outputem konwolucji jest tożsama z ilością różnych filtrów(kerneli) nakładanych na obraz - w tym przypadku 5



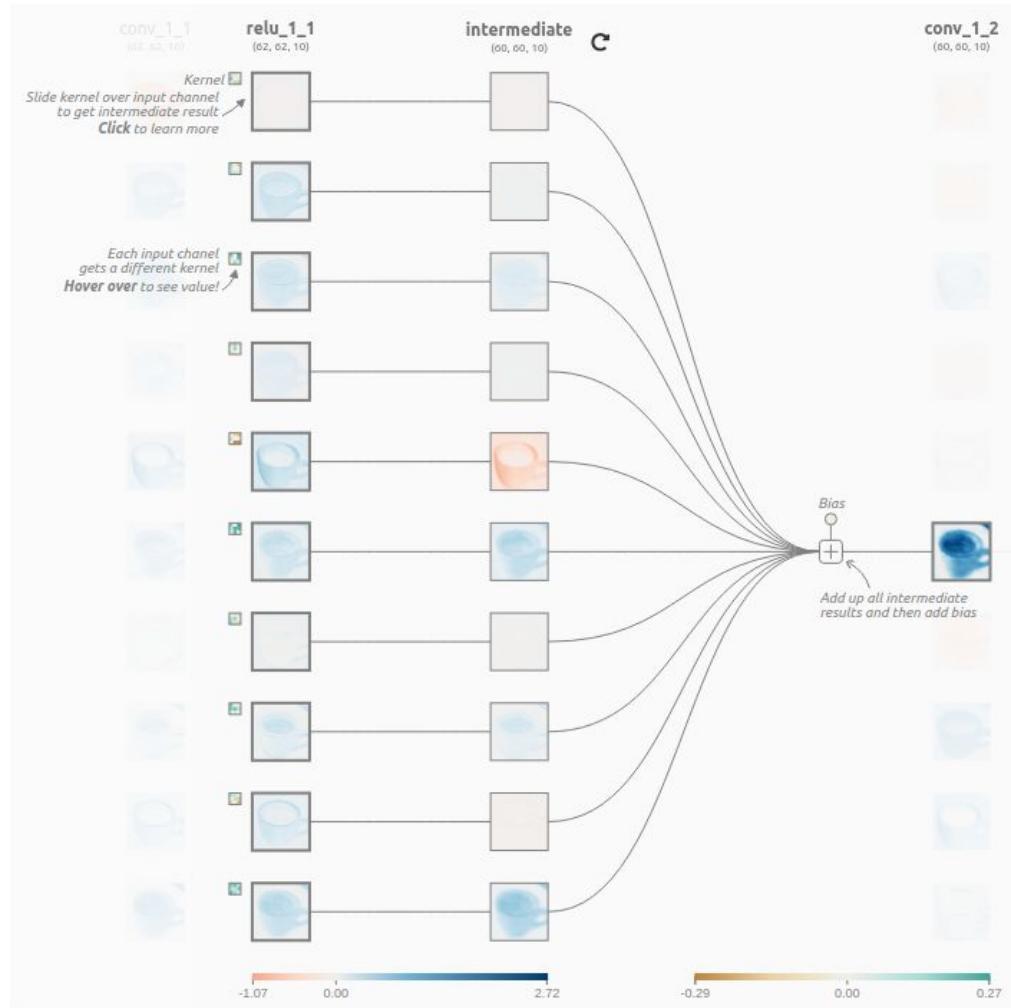
# Jak konwolucja radzi sobie z obrazami wielokanałowymi?

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	
1																										
2	Input					Kernel					Intermediate Output															
4	1	0	1	0	2																					
5	1	1	3	2	1																					
6	1	1	0	1	1																					
7	2	3	2	1	3																					
8	0	2	0	1	0																					
10	1	0	0	1	0																					
11	2	0	1	2	0																					
12	3	1	1	3	0																					
13	0	3	0	3	2																					
14	1	0	3	2	1																					
16	2	0	1	2	1																					
17	3	3	1	3	2																					
18	2	1	1	1	0																					
19	3	1	3	2	0																					
20	1	1	2	1	1																					
21																										

Osobne kernele obsługują każdy z kanałów. Następnie rezultaty takiego przetwarzania są sumowane w jeden output.

Nie należy również zapominać o możliwym biasie pomiędzy pośrednim outputem a finalnym.

# Jak konwolucja radzi sobie z obrazami z kolejnymi warstwami?



Podobnie jak z wejściem, kolejne warstwy konwolucyjne również sumują wyjścia.

# **W jaki sposób określić rozmiar outputu warstwy konwolucyjnej?**

Wzór (analogicznie dla szerokości). Natomiast ilość filtrów będzie nam definiować 3 wymiar naszego wyjścia.

$$W_o = [(W_i - K + 2P)/s] + 1.$$

Gdzie:

$W_o$  – Szerokość wyjścia

$W_i$  – Szerokość wejścia

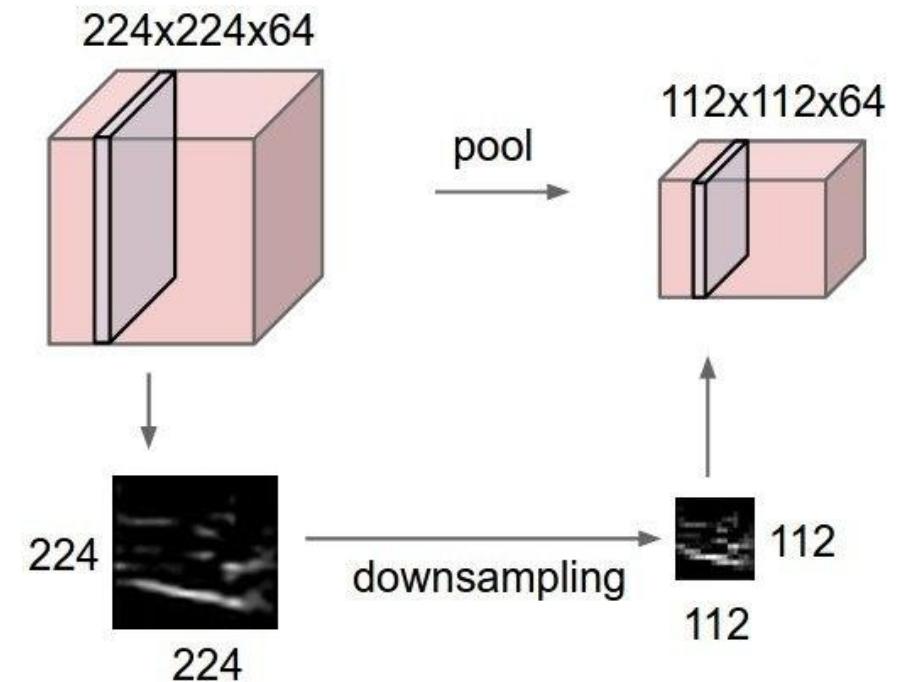
K – Rozmiar filtru

P – Padding

S – Stride

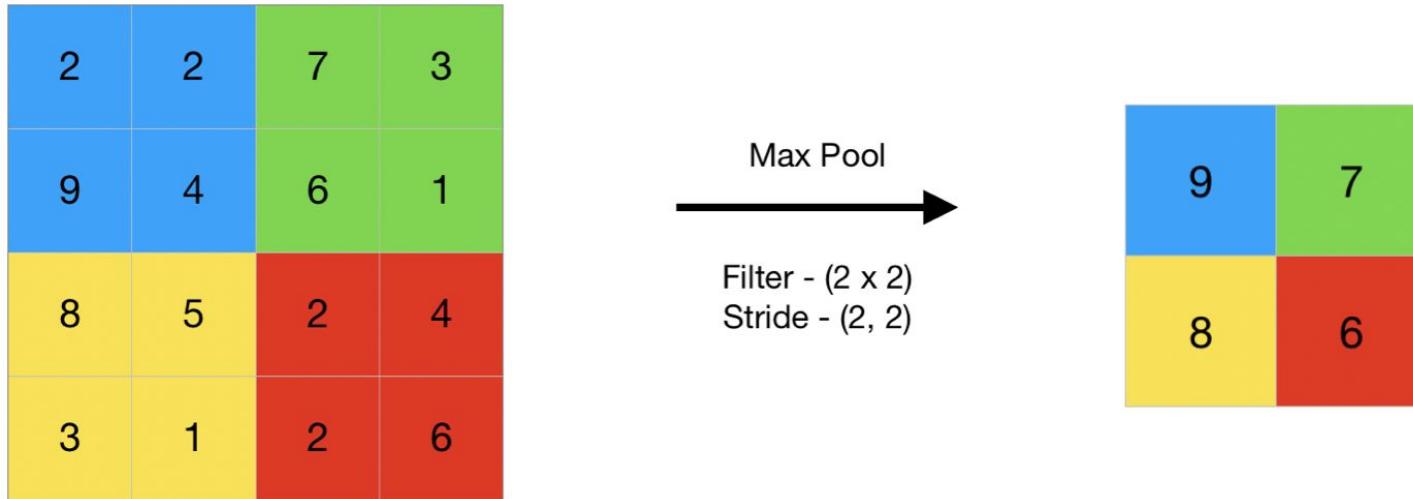
# Pooling

- Zmniejsza liczbę parametrów w modelu - nazywa się to próbkowaniem w dół (downsampling).
- Zbiorczo podsumowuje cechy obecne w poprzedniej warstwie, dzięki czemu model powinien być bardziej odporny na zmianę pozycji cechy na kolejnych obrazach.



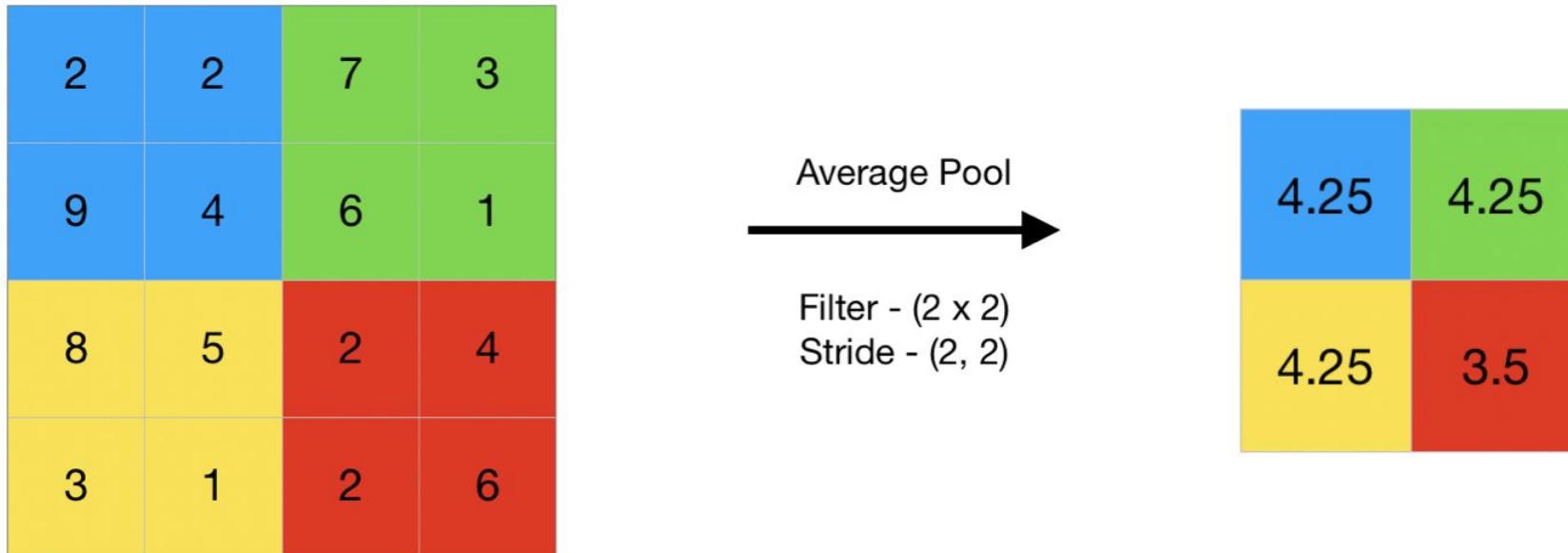
# Max Pooling

Operacja grupowania, która wybiera maksymalny element z obszaru feature mapy objętych filtrem. Zatem wynik po warstwie max-pooling byłby mapą obiektów zawierającą najważniejsze cechy z poprzedniej mapy obiektów.



# Average Pooling

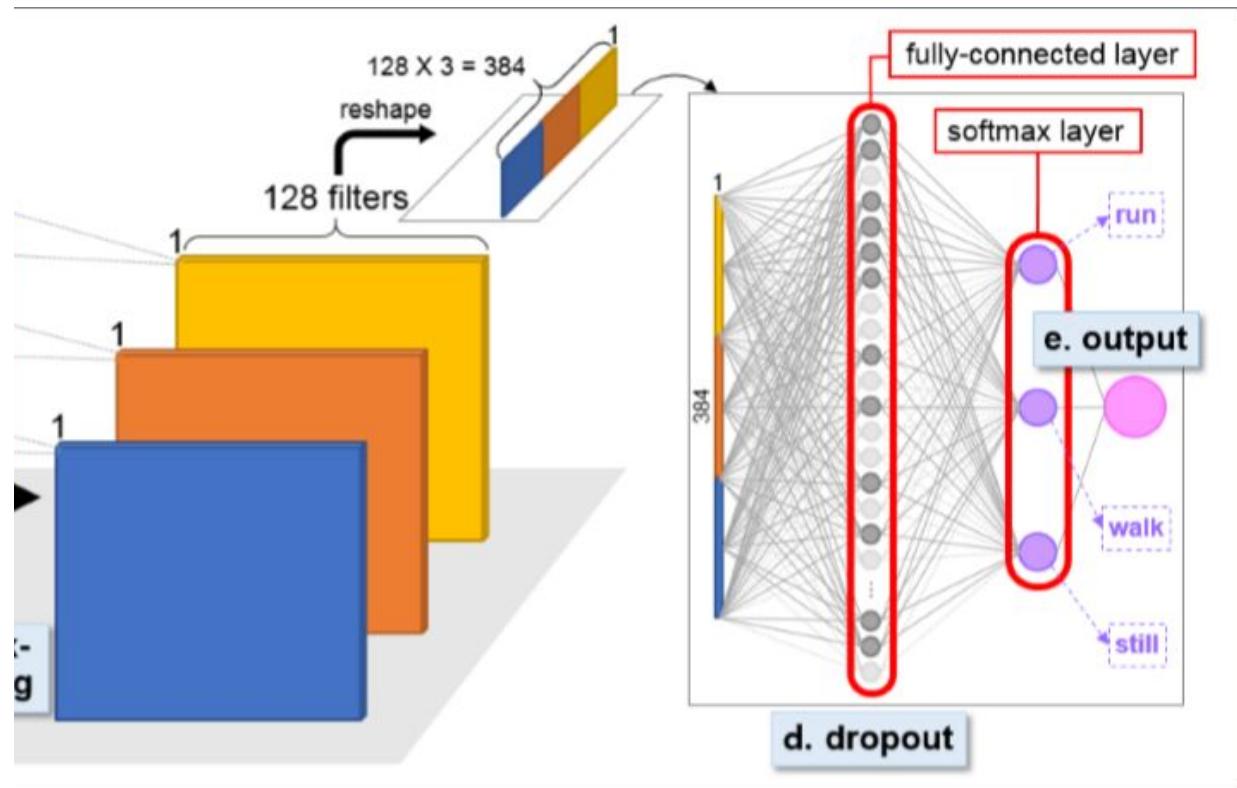
Oblicza średnią elementów obecnych w obszarze feature mapy objętym filtrem. Tak więc, podczas gdy max pooling dają najbardziej widoczną cechę w konkretnym skrawku feature mapy, avarage pooling daje średnią cech obecnych w danym obszarze.



# Fully-connected layer

Na samym końcu sieci występuje klasyczna struktura w pełni połączona umożliwiająca predykcję na podstawie wydobytych w poprzednich warstwach cech.

- Ostatnia warstwa to softmax. Modeluje rozkład prawdopodobieństwa klas.
- Każde połączenie ma wagę, którą można trenować.



# Pierwszy CNN

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))

# compile model
opt = keras.optimizers.SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])

# fit model
history = model.fit(train_x, train_y_one_hot, epochs=30, batch_size=64,
validation_data=(test_x, test_y_one_hot), verbose=1)
```

# Parametry i hiperparametry

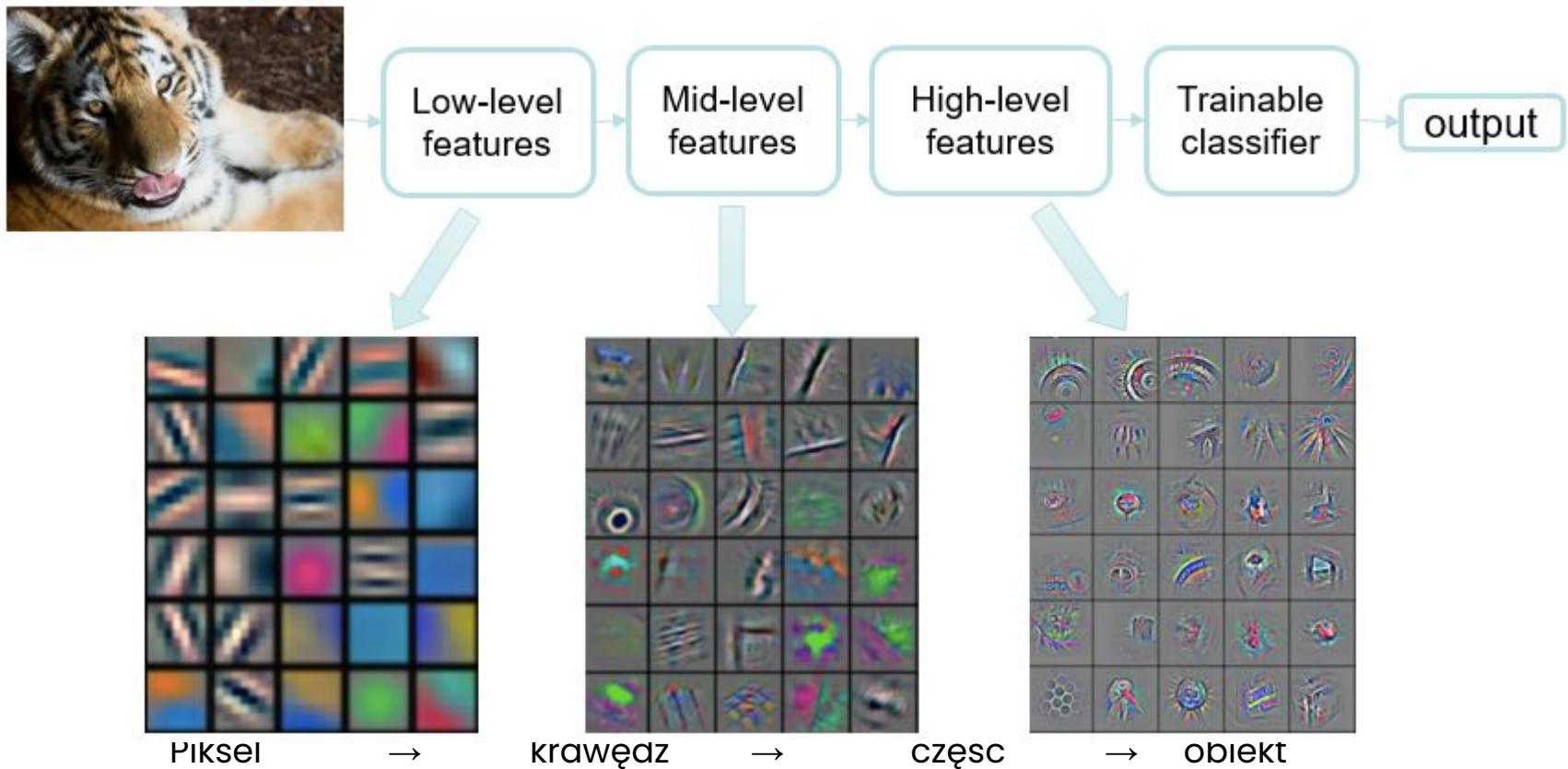
## Parametry

- Wagi oraz bias

## Hiperparametry

Rozmiar kernela: 3x3, 5x5, 11x11  
Stride  
Padding  
Ilość warstw  
Ilość filtrów  
Współczynniki uczenia

# Hierarchia wydobywanych cech



Kolejne warstwy konwolucyjne cechują się wzrostem poziomu abstrakcji.

# Proces uczenia sieci CNN

Tradycyjnie.

W forward propagation informacja jest przesyłana warstwa po warstwie według wstępnie zainicjowanych wag.

W backward propagation wagi są zmieniane zgodnie z gradientem funkcji strat. Błąd propaguje się warstwa po warstwie.

# Fully connected layers

## Forward propagation

Dokładnie tak samo jak w MLP

## Backward propagation

Dokładnie tak samo jak w MLP

# Pooling layer – brak procesu uczenia

## Forward propagation

Daje w wyniku poolingu blok, który jest zredukowany do pojedynczej wartości:

- zwycięskiego neuronu (jego indeks musi być odnotowany podczas przejścia do przodu) – max pooling
- średniej wartości neuronów – avarage pooling

## Backward propagation

- Max Pooling: zwycięskiej jednostce (zapamiętanej w FP) przypisany jest błąd przypisany dla danego neuronu.
- Średnia sumowania błędu jest obliczona poprzez  $1 / (N \times N)$  i ta sama wartość jest przypisywana do wszystkich neuronowych wejściowych.

# Convolutional layer

## Forward propagation

$$\begin{aligned} z_{x,y}^{l+1} &= w^{l+1} * \sigma(z_{x,y}^l) + b_{x,y}^{l+1} \\ &= \sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x-a,y-b}^l) + \end{aligned}$$

## Backward propagation

$$\begin{aligned} \delta_{x,y}^l &= \frac{\partial C}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z_{x',y'}^{l+1}} \frac{\partial z_{x',y'}^{l+1}}{\partial z_{x,y}^l} \\ &= \delta^{l+1} * w_{-x,-y}^{l+1} \sigma'(z_{x,y}^l) \end{aligned}$$

# Batch normalization

Służy do normalizacji wyników poprzednich warstw. Warstwa jest dodawana do modelu sekwencyjnego w celu ujednolicenia danych wejściowych lub wyjściowych. Może być używany w kilku punktach pomiędzy warstwami modelu. Często jest umieszczany tuż po zdefiniowaniu modelu sekwencyjnego i po warstwach splotu i pulowania.

# Dropout

Technika regularyzacji używana do zapobiegania nadmiernemu dopasowaniu w modelu. Dropout jest dodawany do losowego procentu neuronów w sieci. Kiedy neurony są wyłączone, sieć musi uczyć się predykować wyniki bez ich użycia, a tym samym potrafi wykrywać więcej powiązań pomiędzy wynikiem a odkrytymi cechami.

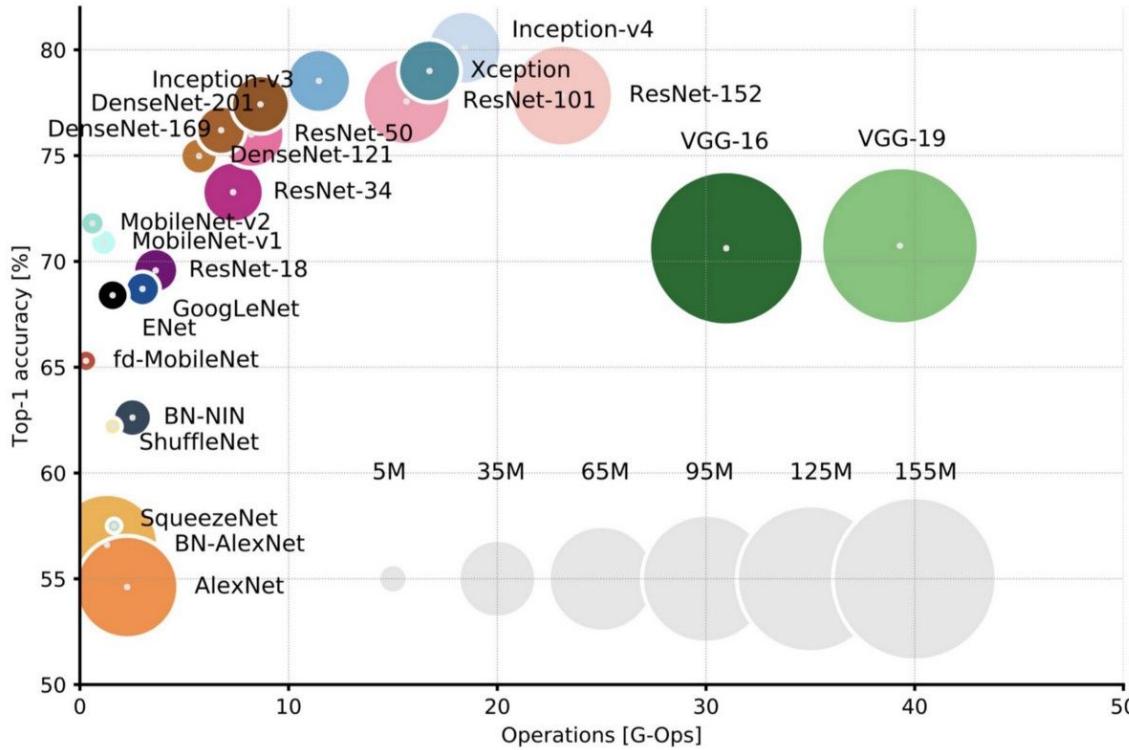
```
model = Sequential ()  
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same', input_shape=(32, 32, 3)))  
model.add(BatchNormalization ())  
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same'))  
model.add(BatchNormalization ())  
model.add(MaxPooling2D ((2, 2)))  
model.add(Dropout (0.2))  
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same'))  
model.add(BatchNormalization ())  
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same'))  
model.add(BatchNormalization ())  
model.add(MaxPooling2D ((2, 2)))  
model.add(Dropout (0.3))  
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same'))  
model.add(BatchNormalization ())  
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same'))  
model.add(BatchNormalization ())  
model.add(MaxPooling2D ((2, 2)))  
model.add(Dropout (0.4))  
model.add(Flatten ())  
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))  
model.add(BatchNormalization ())  
model.add(Dropout (0.5))  
model.add(Dense(10, activation='softmax'))
```

# Dodajemy BN i Dropout!

# Architektury CNN

- MobileNet
- DarkNet
- Inception
- ResNet
- DenseNet
- AlexNet
- VGG
- GoogleNet

# Zestawienie różnych architektur



Model	Top-1	Top-5	Ops	GPU	CPU	Cfg	Weights
AlexNet	57.0	80.3	2.27 Bn	3.1 ms	0.29 s	cfg	238 MB
Darknet Reference	61.1	83.0	<b>0.96 Bn</b>	<b>2.9 ms</b>	<b>0.14 s</b>	cfg	28 MB
VGG-16	70.5	90.0	30.94 Bn	9.4 ms	4.36 s	cfg	528 MB
Extraction	72.5	90.8	8.52 Bn	4.8 ms	0.97 s	cfg	90 MB
Darknet19	72.9	91.2	7.29 Bn	6.2 ms	0.87 s	cfg	80 MB
Darknet19 448x448	76.4	93.5	22.33 Bn	11.0 ms	2.96 s	cfg	80 MB
Resnet 18	70.7	89.9	4.69 Bn	4.6 ms	0.57 s	cfg	44 MB
Resnet 34	72.4	91.1	9.52 Bn	7.1 ms	1.11 s	cfg	83 MB
Resnet 50	75.8	92.9	9.74 Bn	11.4 ms	1.13 s	cfg	87 MB
Resnet 101	77.1	93.7	19.70 Bn	20.0 ms	2.23 s	cfg	160 MB
Resnet 152	77.6	93.8	29.39 Bn	28.6 ms	3.31 s	cfg	220 MB
ResNeXt 50	77.8	94.2	10.11 Bn	24.2 ms	1.20 s	cfg	220 MB
ResNeXt 101 (32x4d)	77.7	94.1	18.92 Bn	58.7 ms	2.24 s	cfg	159 MB
ResNeXt 152 (32x4d)	77.6	94.1	28.20 Bn	73.8 ms	3.31 s	cfg	217 MB
Densenet 201	77.0	93.7	10.85 Bn	32.6 ms	1.38 s	cfg	66 MB
Darknet53	77.2	93.8	18.57 Bn	13.7 ms	2.11 s	cfg	159 MB
Darknet53 448x448	<b>78.5</b>	<b>94.7</b>	56.87 Bn	26.3 ms	7.21 s	cfg	159 MB

# MobileNet

Mobilna sieć do użycia nawet na przenośnych urządzeniach.

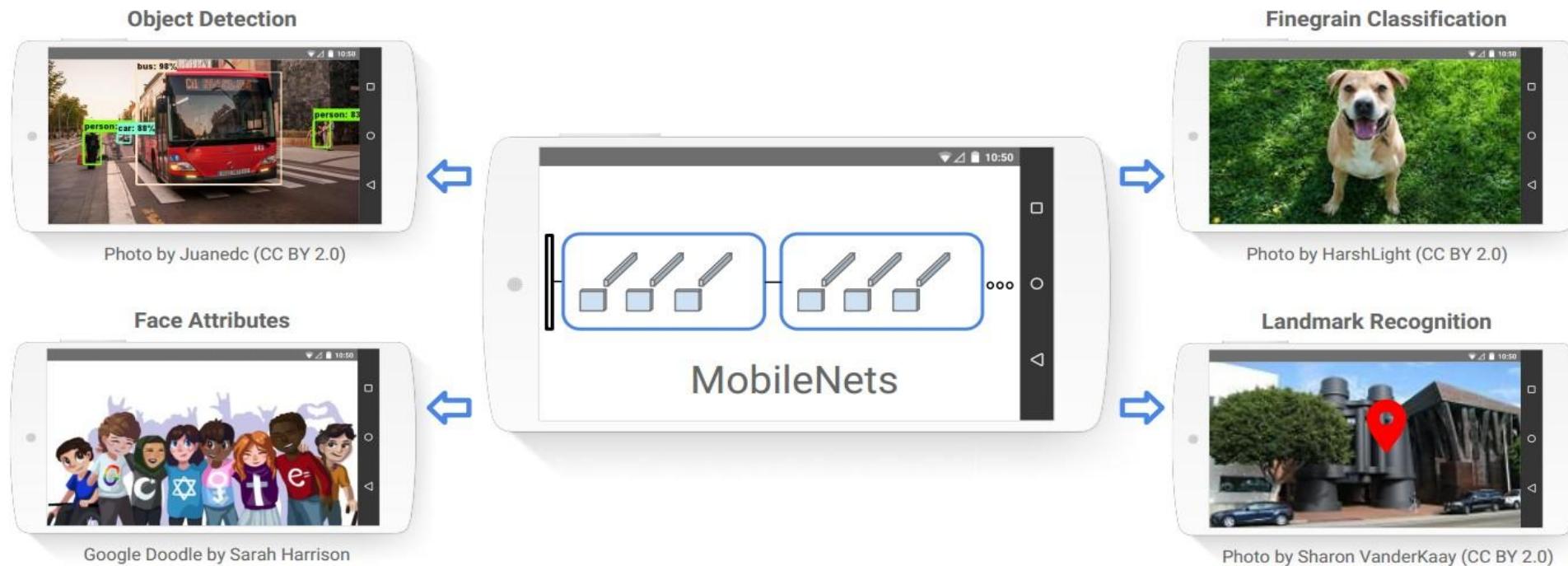


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

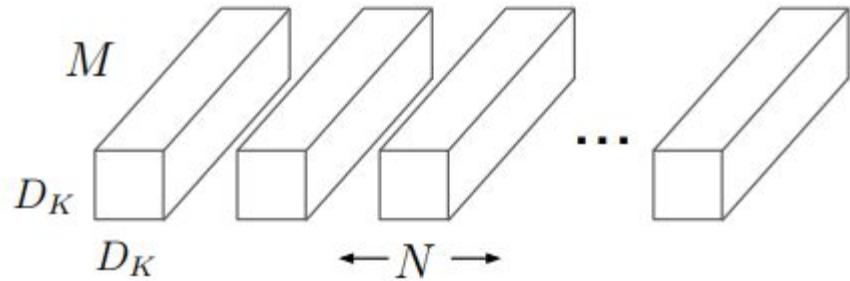
# MobileNet

Jak stworzyć wydajną sieć konwolucyjną?

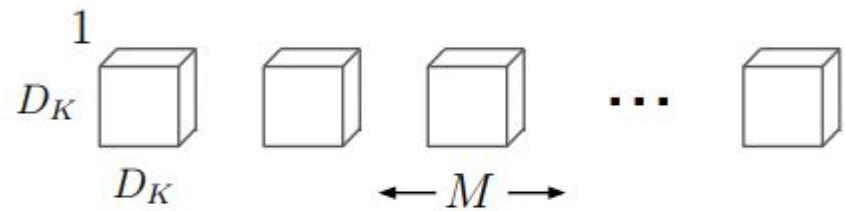
- Zredukować Fully connected layers
- Zmniejszyć kernele (z popularnego 3x3 do 1x1) – pointwise convolution
- Zredukować kanały
- Depthwise Separable Convolutions

# Depthwise Convolutions

- Są to konwolucje z wykorzystaniem filtrów, który w odróżnieniu od standardowych kerneli mają trzeci wymiar.
- Umożliwia to na jednokrotnie procedowanie wielu kanałów.
- Na wyjście otrzymuje się zamiast pojedynczej wartości, wektor.

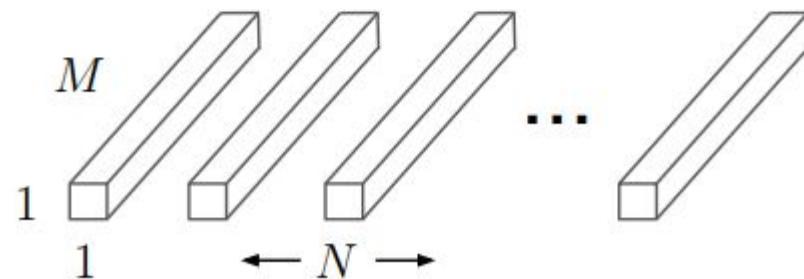


(a) Standard Convolution Filters



# Pointwise Convolutions

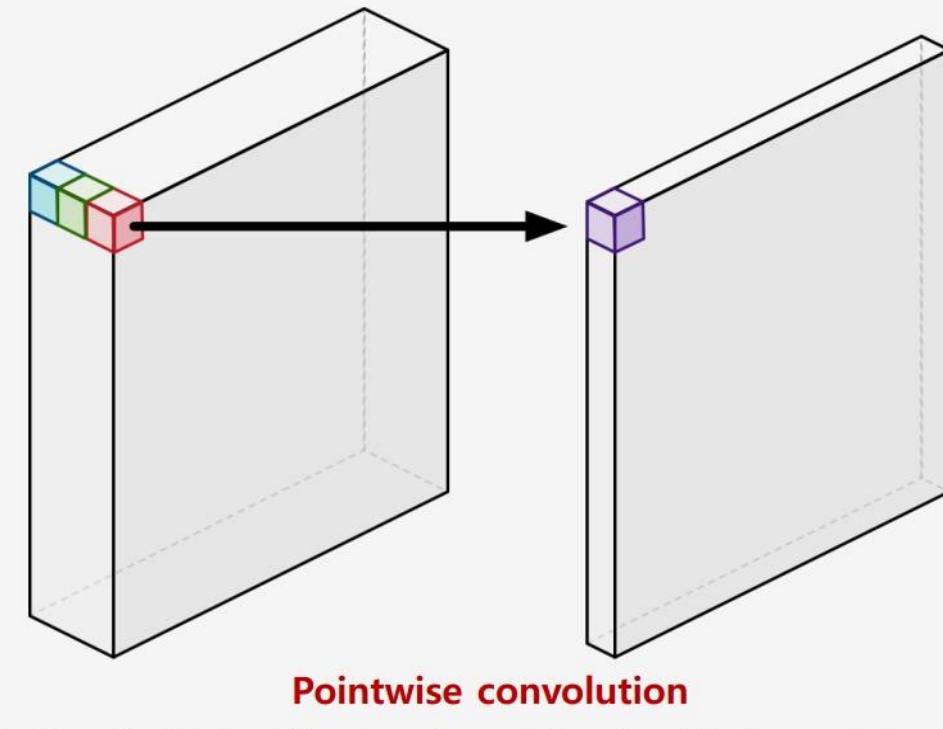
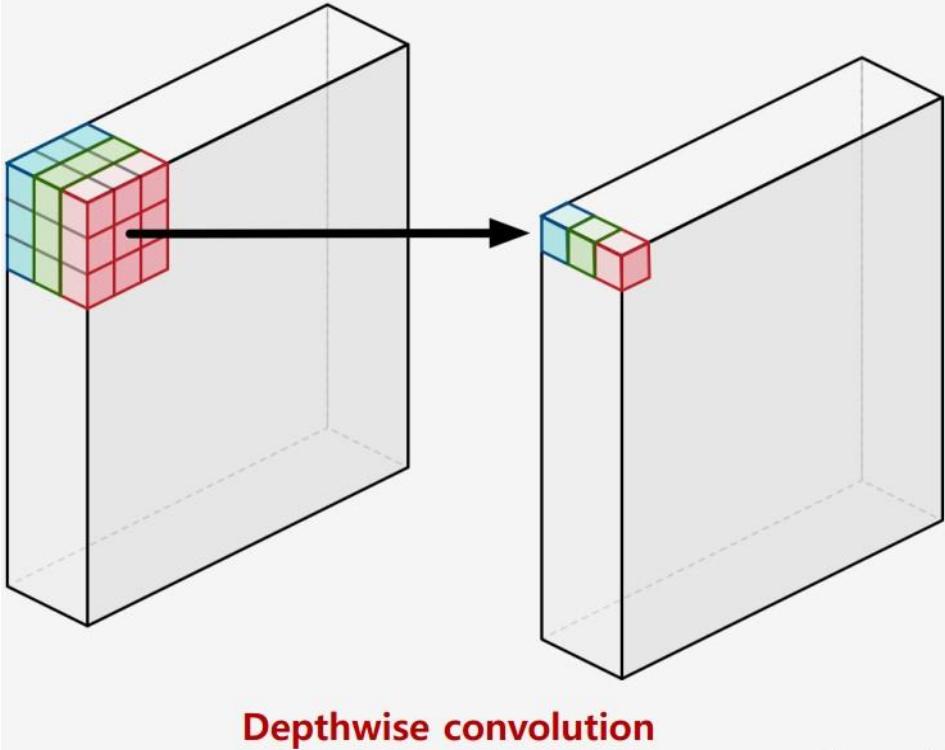
- Następnie stosuje się konwolucje  $1 \times 1$ , która “łączy ze sobą” kanały wejściowy za pomocą różnych wag.



(c)  $1 \times 1$  Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

# Depthwise separable Convolutions

- Depthwise Convolution + Pointwise Convolution( $1 \times 1$  convolution)



Figures from <http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/>

# Warstwy MobileNetu

Finalnie zastępuje się standardowe warstwy konwolucyjne warstwami, które wykorzystują obie te techniki.

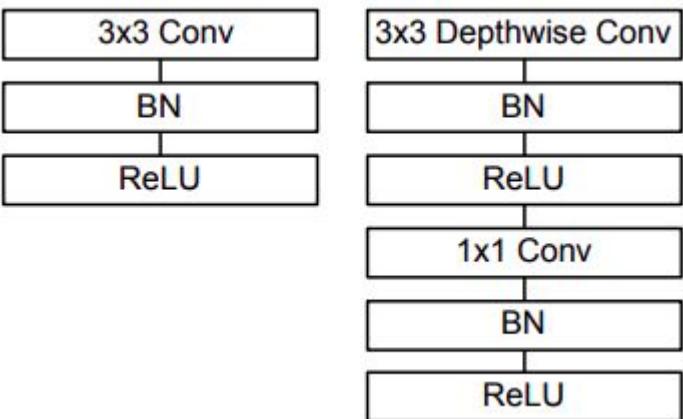


Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool $7 \times 7$	$7 \times 7 \times 1024$
FC / s1	$1024 \times 1000$	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

# MobileNet

W rezultacie otrzymujemy sieć, która posiada znacznie mniej parametrów koniecznych do wyuczenia.

Ilość operacji w standardowej konwolucji:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

**Depthwise:**  $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$

**Pointwise:**  $M \cdot N \cdot D_F \cdot D_F$

Rezultat:

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

```
from keras.models import Model
from keras.applications import MobileNetV2

(train_x, train_y), (test_x, test_y) = keras.datasets.cifar10.load_data ()
# 0-255 to 0-1
train_x = train_x/ 255
test_x = test_x/ 255

# Change the labels from integer to categorical data
train_y_one_hot = to_categorical (train_y)
test_y_one_hot = to_categorical (test_y)

model = MobileNetV2 (input_shape= (train_x.shape [1], train_x.shape [2], 3),
                     classes= 10, include_top= False, weights= None)

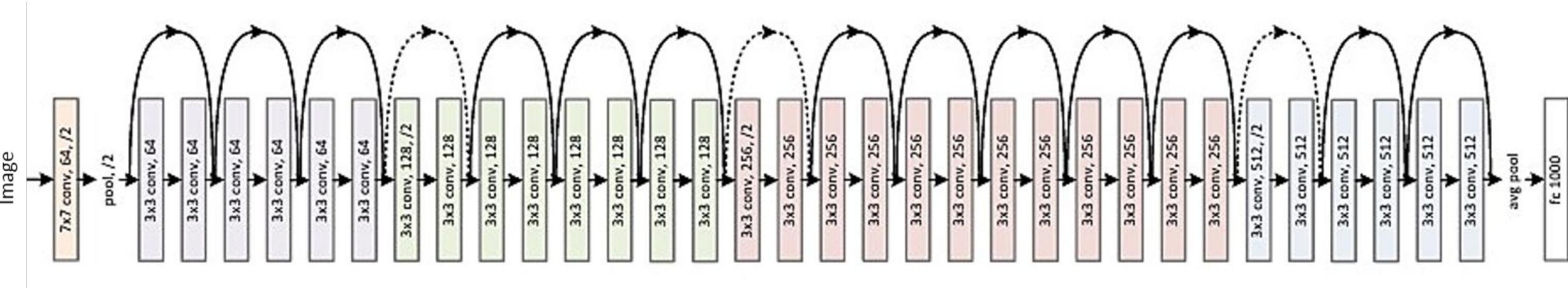
x=model.layers [-1].output
x=keras.layers.BatchNormalization (axis=-1, momentum=0.99, epsilon=0.001 ) (x)
flatten=Flatten () (x)
predictions=Dense (10, activation='softmax') (flatten)
model = Model (inputs=model.input, outputs=predictions )
# compile model
opt = keras.optimizers.SGD (lr=0.001, momentum= 0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=[ 'accuracy'])

# fit model
history = model.fit (train_x, train_y_one_hot, epochs=30, batch_size=64,
                     validation_data= (test_x, test_y_one_hot), verbose=1)
# evaluate model
_, acc = model.evaluate (test_x, test_y_one_hot, verbose=1)
```

# Cifar10, a MobileNet?

# ResNet

Są sieciami wyjątkowo głębokimi, które mogłyby się wydawać nie powinny działać ze względu na ilość parametrów. Jednak dzięki zastosowaniu residualnych połączeń jest to możliwe.



# ResNet

Głównym z problemów rozwiązywanych przez ResNety jest vanishing gradient. Dzieje się tak, ponieważ gdy sieć jest zbyt głęboka, gradienty, z których obliczana jest funkcja straty, zmniejszają się do zera (użycie ReLu).

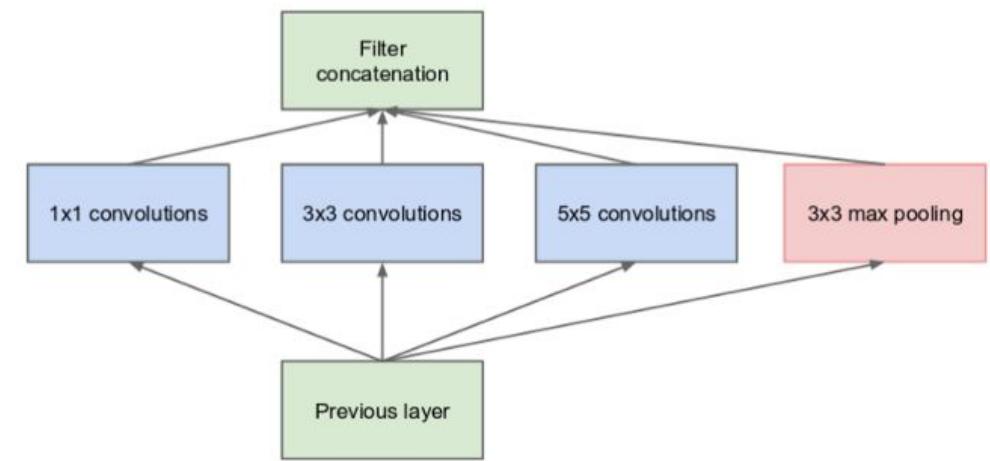
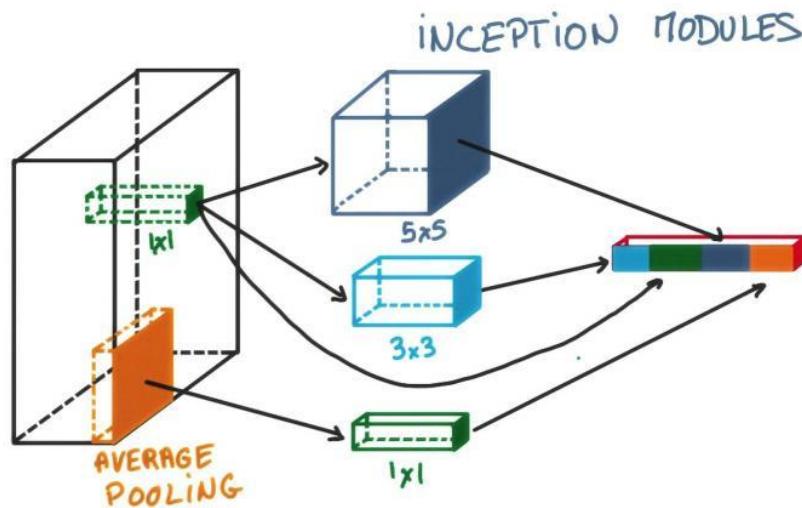
Dzięki sieciom ResNets gradienty mogą przepływać bezpośrednio przez połączenia wstecz z późniejszych warstw do początkowych filtrów.

# Inception - idea

- Istotne części obrazu mogą mieć różny rozmiar. Na przykład obraz przedstawiający psa może być zdjęciem z bliska, ale też wykonanym z pewnej odległości. Na każdym zdjęciu obszar zajmowany przez psa jest inny. -> Większe cechy, muszą być wykrywane przez większe kernele.
- Bardzo głębokie sieci neuronowe, które mogłyby analizować zarówno duże i małe cechy mają skłonność do overfittingu (przez zbyt dużą ilość parametrów).

# Inceptionv1 - GoogleNet

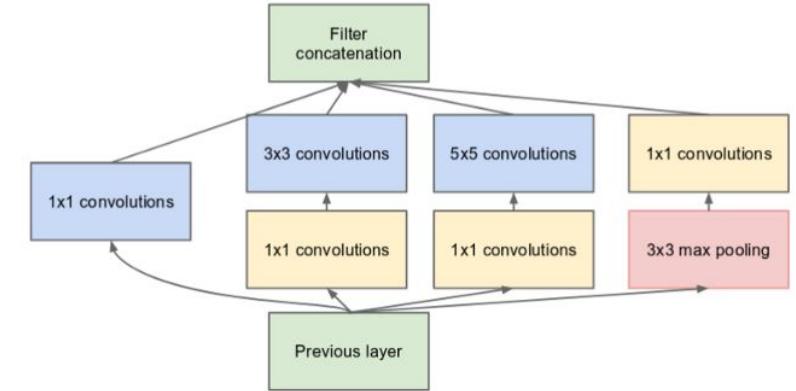
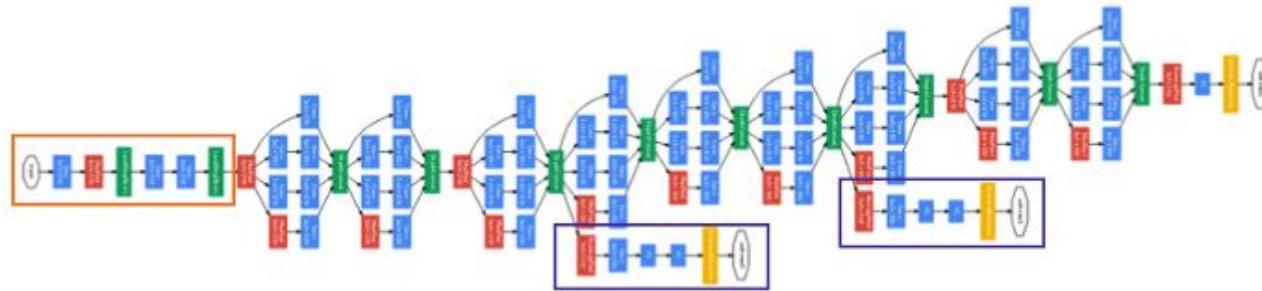
Jedynym rozwiązaniem wydaje się dokonanie konwolucji wieloma różnymi filtrami jednocześnie. Jednakże to rozwiązanie było mocno obliczeniowo nieefektywne.



(a) Inception module, naïve version

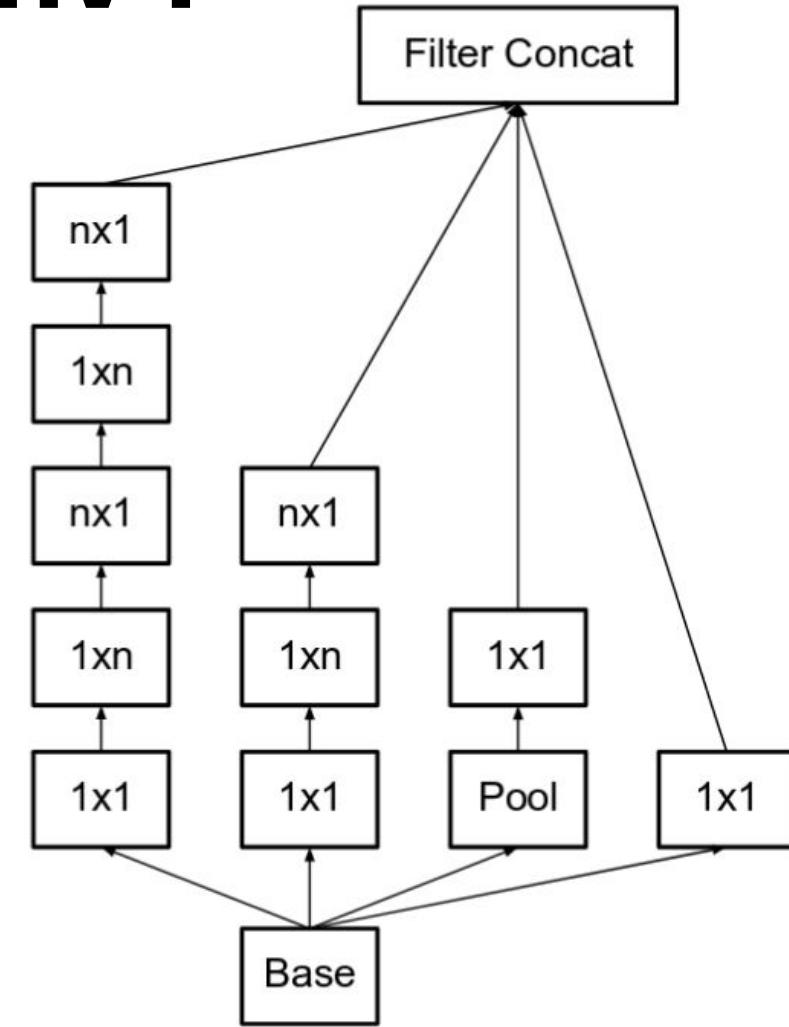
# Inceptionv1 - GoogleNet

Ostatecznie postanowiono ograniczyć ilość obliczeń poprzez zastosowanie pixelwise concatenation. W ten sposób powstała pierwsza wersja Incepcji.



# Inceptionv1 → Inceptionv4

Z czasem dokonano wiele usprawnień czego wyrazem było stworzenie Incepjiv4. Jednym z nich była optymalizacja obliczeń w sieci poprzez rozkład konwolucji z jednokrotnego użycia filtru  $n \times n$  na użycie dwóch o wymiarach  $1 \times n$  i  $n \times 1$ .



# Inceptionv4

Ostatecznie architektura zakłada obecność następujących warstw:

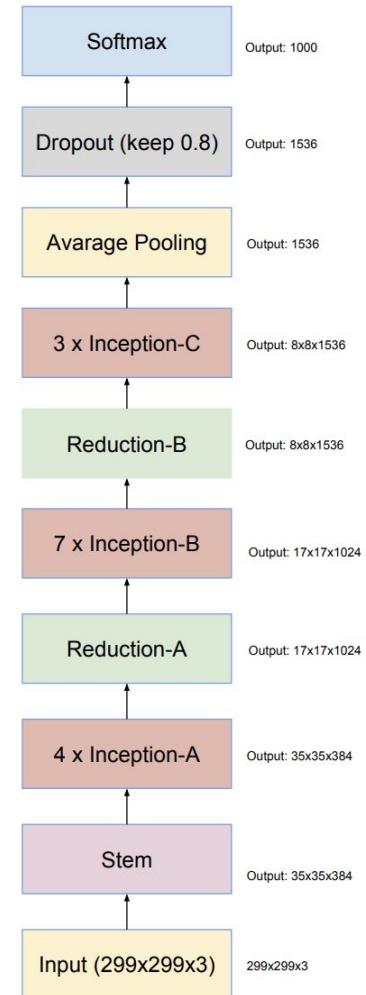
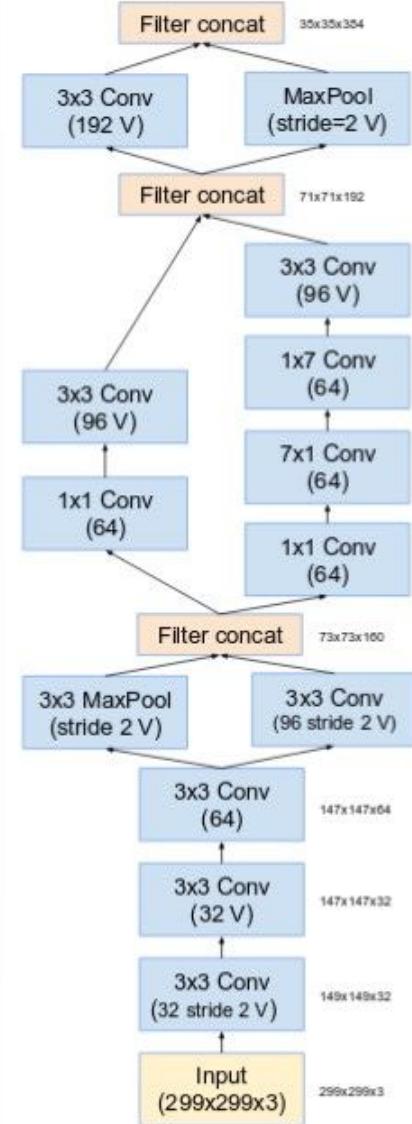


Figure 9. The overall schema of the Inception-v4 network. For the detailed modules, please refer to Figures 3, 4, 5, 6, 7 and 8 for the detailed structure of the various components.

# Inceptionv4

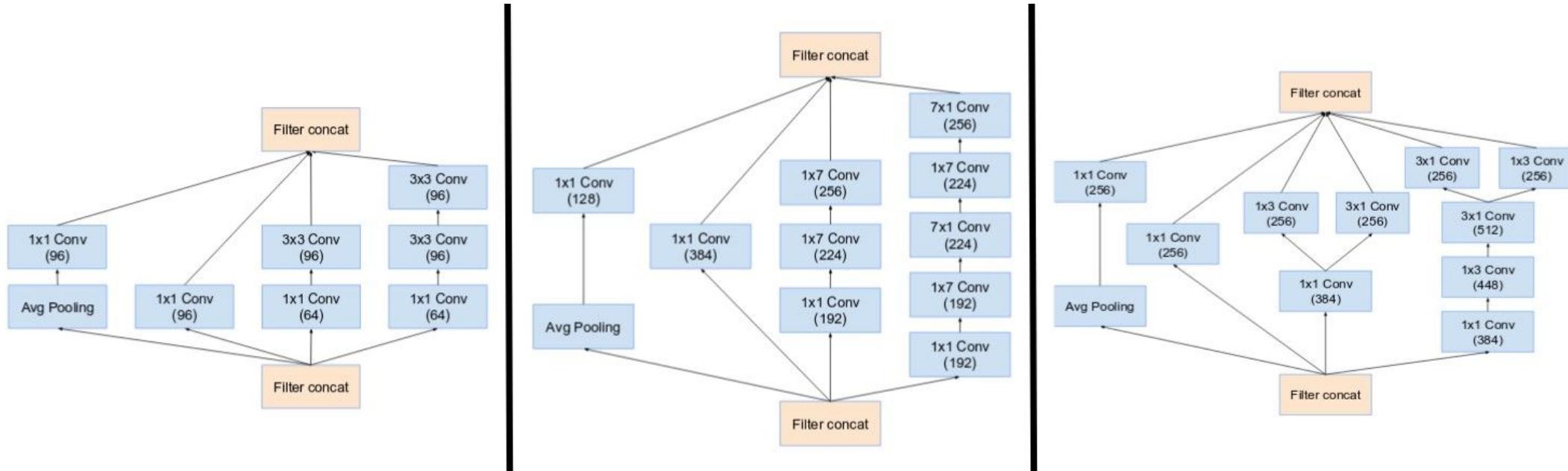
Blok stem:

Jest blokiem zawierającym operacje dokonywane przed blokiem incepcji.



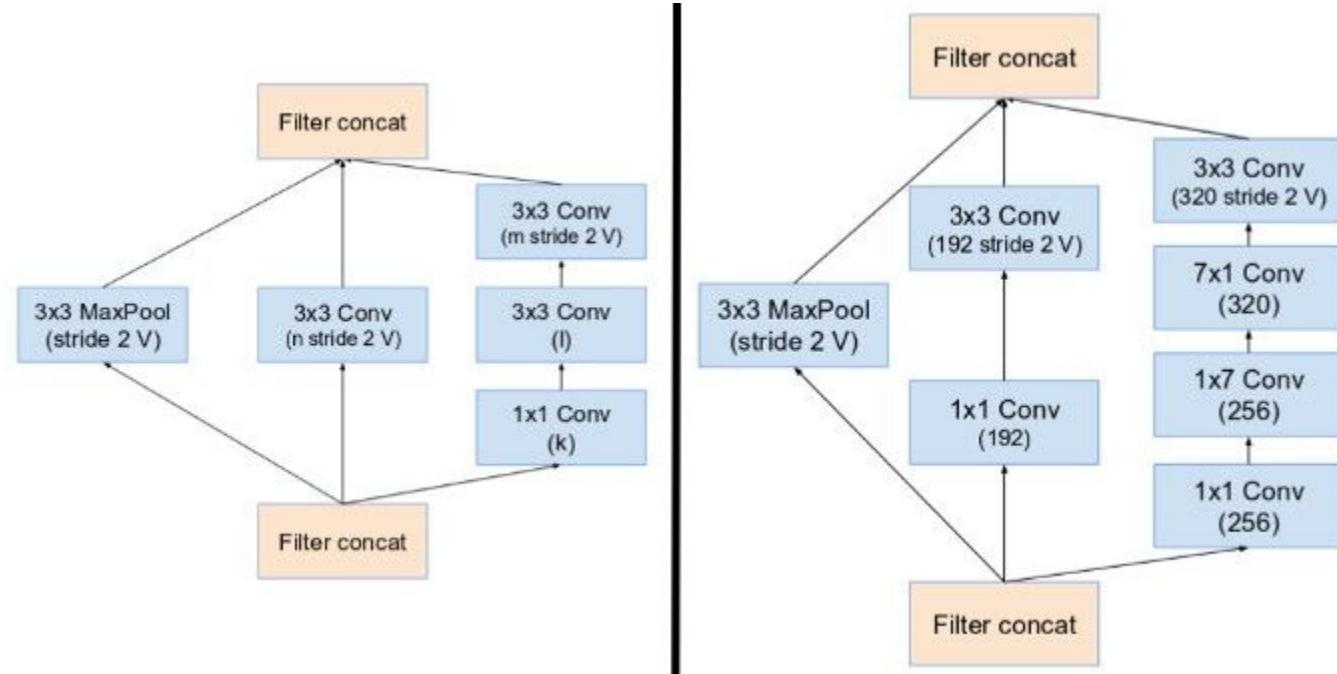
# Inceptionv4

Trzy główne moduły inception nazwane A, B i C. Utworzone na wzór pierwowzoru z Incepçjvi. Zoptymalizowane pod kątem obliczeniowym.



# Inceptionv4

I finalnie bloki “Reduction Blocks”, które służą do zmiany szerokości feature map.

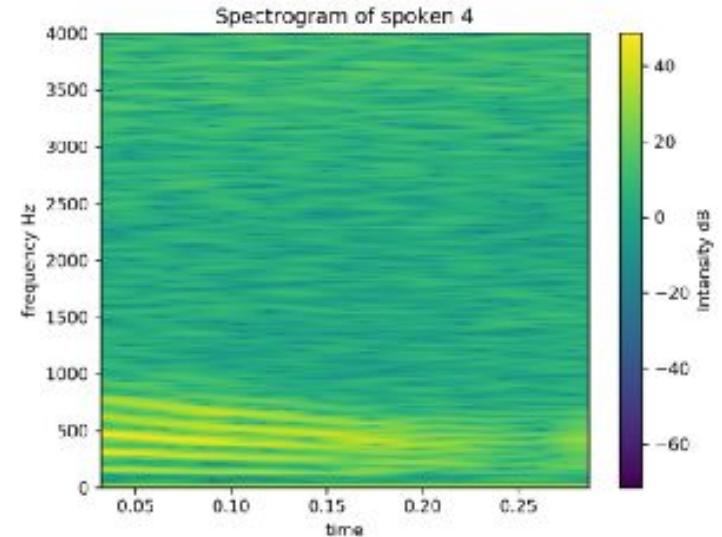


# Przykłady nietypowego zastosowania CNN

Analiza dźwięku - spektrogramów:

Sieci CNN można wykorzystać do klasyfikacji dźwięku.

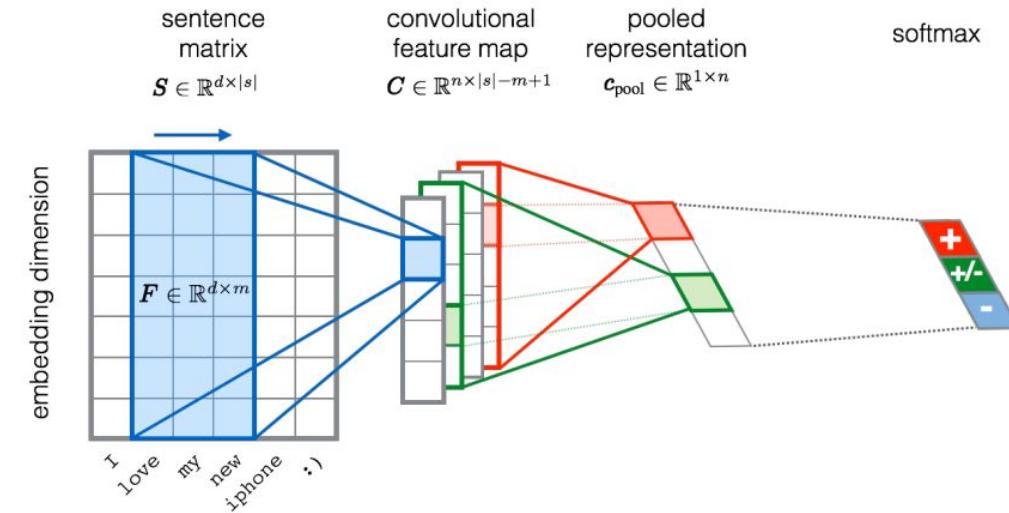
Np. do rozpoznawania rodzaju muzyki.



# Przykłady nietypowego zastosowania CNN

## Klasyfikacja tekstu

Poprzez rozbicie wyrazów na wektory i utworzenie z nich macierzy można pokusić się o klasyfikację tekstu.



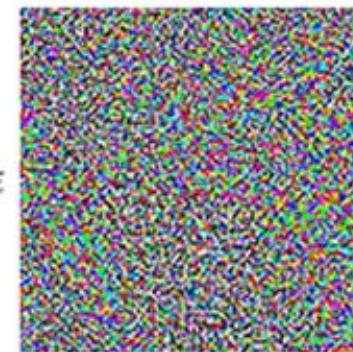
# Jak oszukać CNN?

Okazuje się, że wystarczy dodanie odpowiedniego szumu do obrazu, żeby oszukać CNN.



“panda”

57.7% confidence



+  $\epsilon$



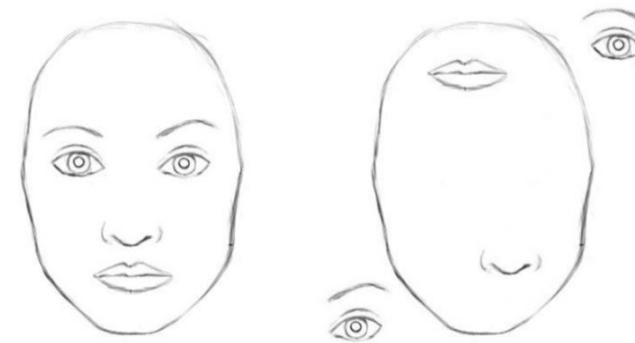
“gibbon”

99.3% confidence

# Jak oszukać CNN?

Sieci konwolucyjne mają również problem z rotacją i deformacją obiektów.

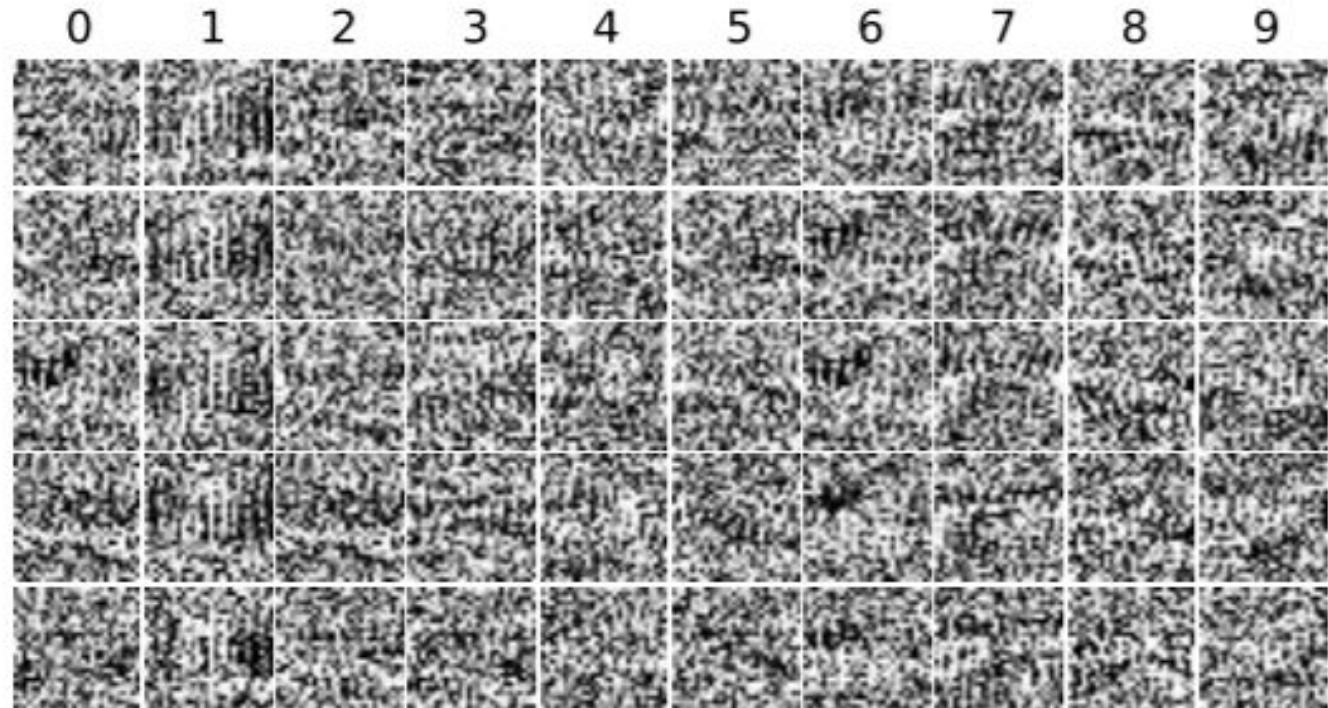
- Rotacja – sieci konwolucyjne zapamiętują konkretną orientację cech na obrazie, bez odpowiednio skonstruowanego zbioru ciężka jest predykcja obiektów obróconych względem tego jak najczęściej wyglądają na obrazach.
- Deformacja – sieci konwolucyjne badają wystąpienia konkretnych cech na obrazie, a nie ich względową orientację między sobą.



# Jak oszukać CNN?

Tworzone są również algorytmy, których celem jest właśnie oszukiwanie sieci CNN.

Po prawej Algorytm genetyczny wypracował szумy, które wytrenowane na zbiorze MNIST były fałszywie kwalifikowane z 99% skutecznością.



# Koniec dnia drugiego. Linki:

Konwencjonalne vs DeepLearning:

<https://medium.com/discover-computer-vision/deep-learning-vs-traditional-techniques-a-comparison-a590d66b63bd>

Pobaw się CNN!:

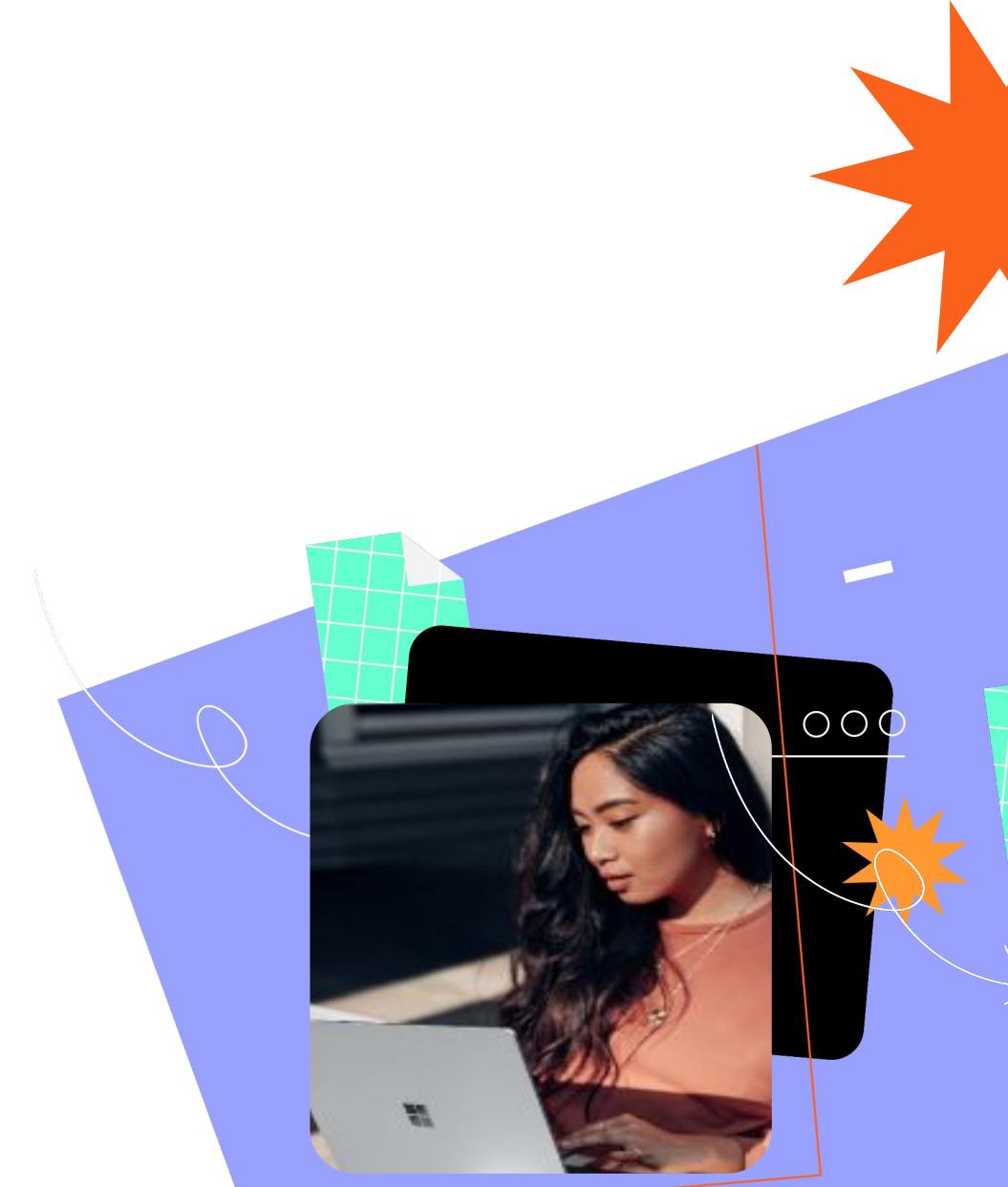
<https://poloclub.github.io/cnn-explainer/>

Zestawienie różnych architektur:

<https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d>

# **Przetwarzanie obrazu – Computer Vision**

**Kurs Data Science – Dzień 3**

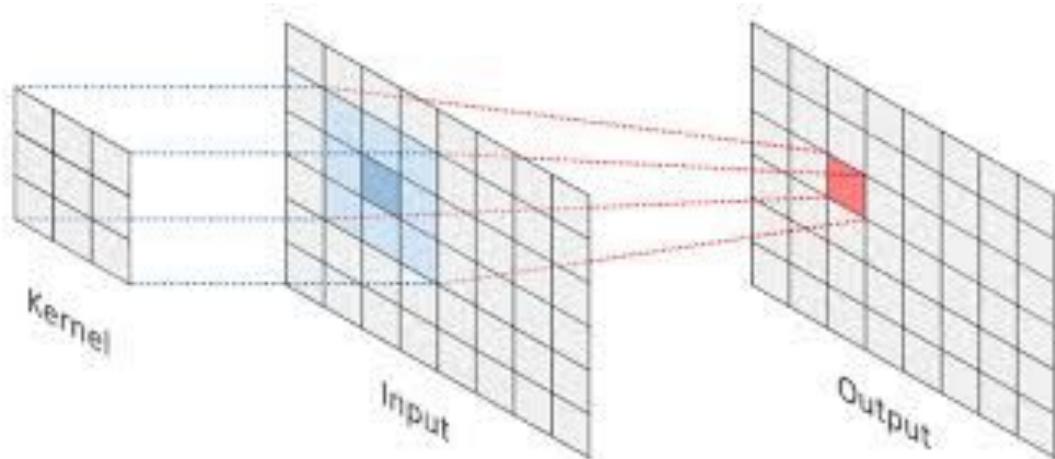


# Plan prezentacji:

- Zadania realizowane w trakcie analizy obrazów
  - Klasyfikacja
  - Detekcja
  - Segmentacja
  - Sekwencje
  - Tworzenie
- Data Augmentation
- Transfer learning

# Wstęp

Poznane dotychczas metody skupiały się na problemie klasyfikacji obrazów. Najważniejsze było zrozumienie zasad kryjących się za działaniem **sieci konwolucyjnych** oraz różnych mechanizmów, które starają się usprawnić ich działanie. Ten sposób przetwarzania cech występuje w praktycznie **każdym** problemie przetwarzania obrazu.



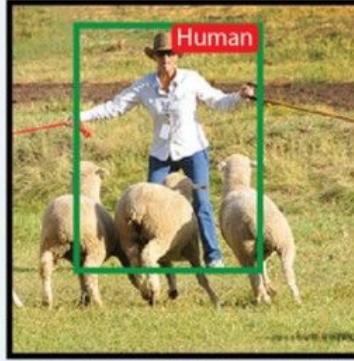
# Zadania związane z przetwarzaniem obrazu



## Image Classification

Classify an image based on the dominant object inside it.

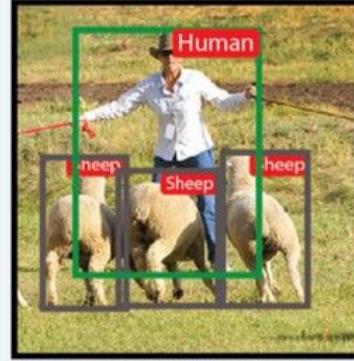
**datasets:** MNIST, CIFAR, ImageNet



## Object Localization

Predict the image region that contains the dominant object. Then image classification can be used to recognize object in the region

**datasets:** ImageNet



## Object Recognition

Localize and classify all objects appearing in the image. This task typically includes: proposing regions then classify the object inside them.

**datasets:** PASCAL, COCO



## Semantic Segmentation

Label each pixel of an image by the object class that it belongs to, such as human, sheep, and grass in the example.

**datasets:** PASCAL, COCO



## Instance Segmentation

Label each pixel of an image by the object class and object instance that it belongs to.

**datasets:** PASCAL, COCO



## Keypoint Detection

Detect locations of a set of predefined keypoints of an object, such as keypoints in a human body, or a human face.

**datasets:** COCO

+ Image generation

# Klasyfikacja

Problem klasyfikacji polega na przypisaniu obrazowi **jednej** z dostępnych etykiety kategorii.

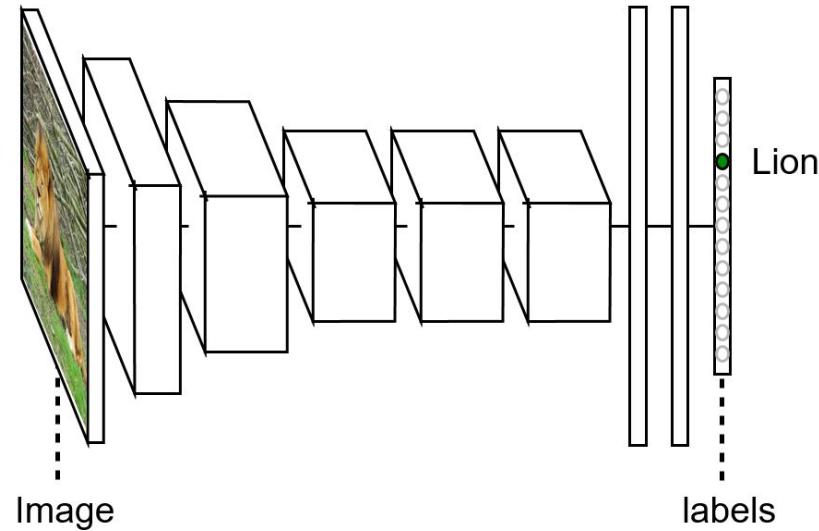


Image classification  
(what? I don't care where)

[http://deeplearning.csail.mit.edu/instance\\_ross.pdf](http://deeplearning.csail.mit.edu/instance_ross.pdf)

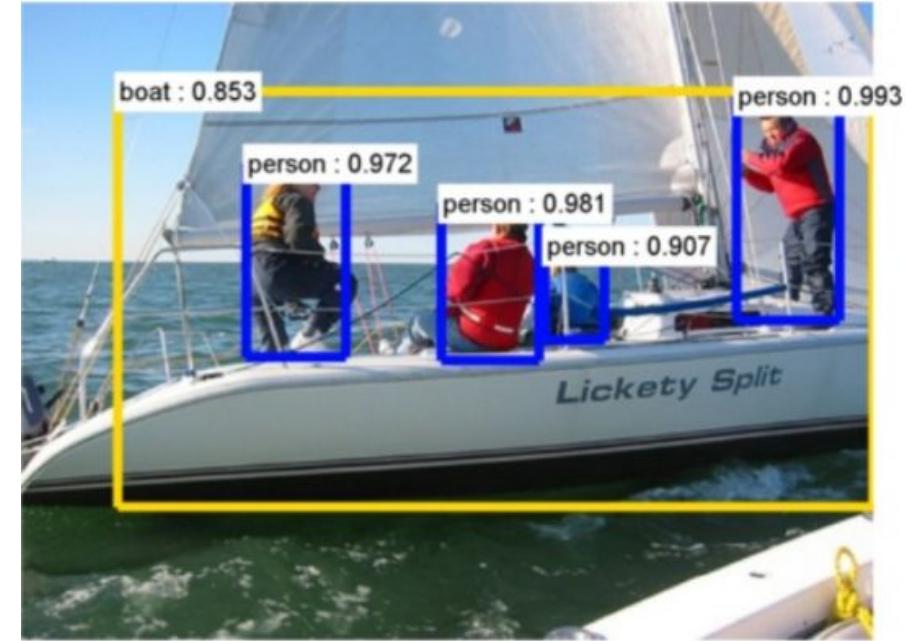
# Klasyfikacja - krótkie powtóżenie

- Warstwy konwolucyjne w tym przypadku kończą się warstwą w pełni połączoną z softmaxem na jej końcu.
- W niektórych zastosowaniach można wykorzystać SVM zamiast warstwy fully-connected.



# Detekcja

Detekcja obiektów na obrazie wymaga **znalezienia obiektu na obrazie** i określenia najmniejszej ramki otaczającej w całości dany obiekt.



Object detection  
(what + where?)

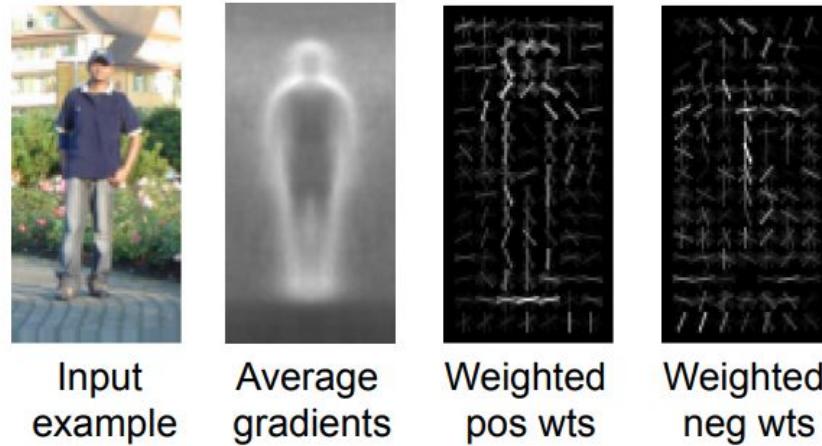
[http://deeplearning.csail.mit.edu/instance\\_ross.pdf](http://deeplearning.csail.mit.edu/instance_ross.pdf)

# Detekcja - Metody

- Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN, cascade R-CNN.)
- Single Shot MultiBox Detector (SSD)
- You Only Look Once (YOLO)
- Single-Shot Refinement Neural Network for Object Detection (RefineDet)
- Retina-Net
- Deformable convolutional networks

# Object detection

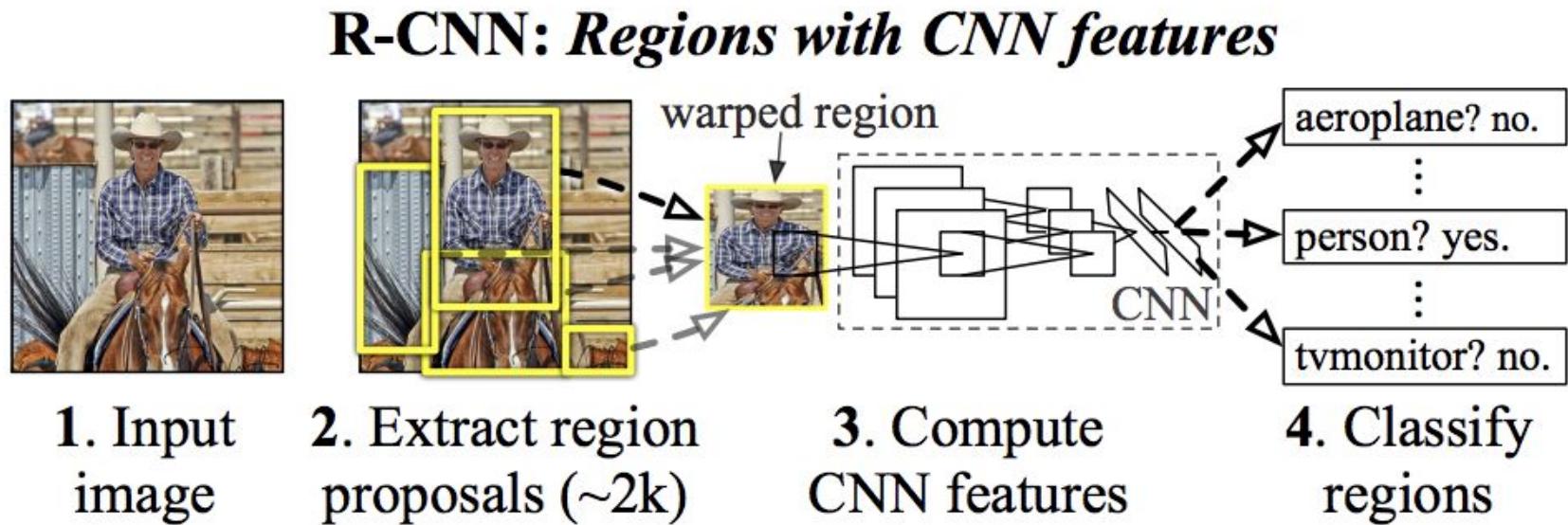
- Wcześniej wykrywanie obiektów na obrazie opierało się o bardziej prymitywne metody np. **HOG**
- Pierwsze deeplearningowe podejście **R-CNN - Regions with CNN features** (2014)



[http://vision.stanford.edu/teaching/cs231b\\_spring1213/slides/HOG\\_2011\\_Stanford.pdf](http://vision.stanford.edu/teaching/cs231b_spring1213/slides/HOG_2011_Stanford.pdf)

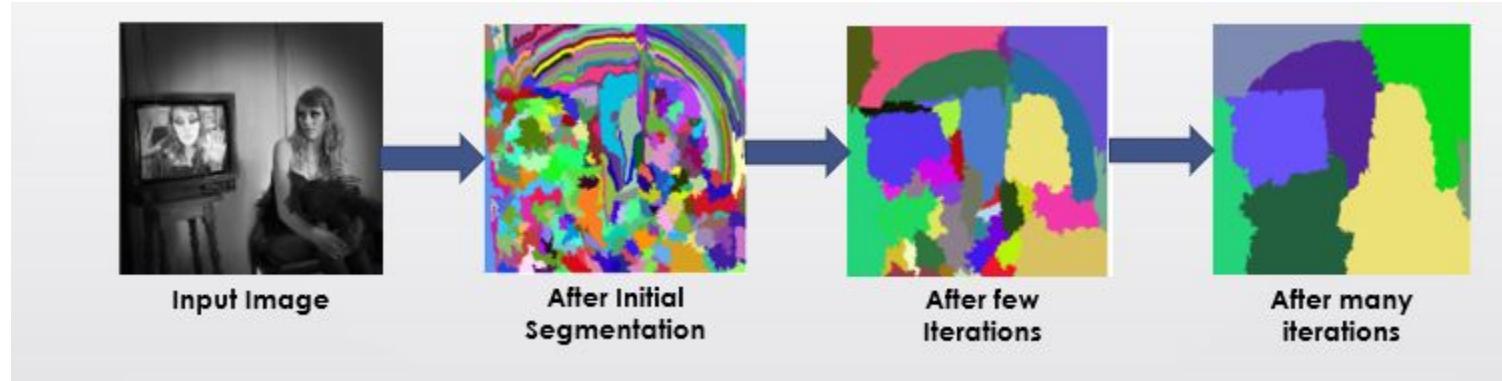
# R-CNN - Działanie

- Moduł 1: **Region Proposal.** Generuje i wyodrębnia propozycje regionów niezależnie od kategorii – kandydatów na ramki ograniczające.
- Moduł 2: **Feature Extractor.** Wyodrębnia cechy z każdego regionu kandydującego przy użyciu CNN.
- Moduł 3: **Classifier.** Klasyfikuje obiekty jako jedną ze znanych klas, np. SVM albo Fully Connected.



# R-CNN - Cechy

- Regional proposal najczęściej był implementowany za pomocą algorytmu "selective search"
  - Stwórz sobie początkowe regiony.
  - Z zestawu regionów wybierz dwa najbardziej podobne (podobieństwo może być określone na wiele sposobów - patrz link).
  - Połącz je w jeden, większy region.
  - Powtórz powyższe kroki dla wielu iteracji do osiągnięcia zadanej ilości regionów.



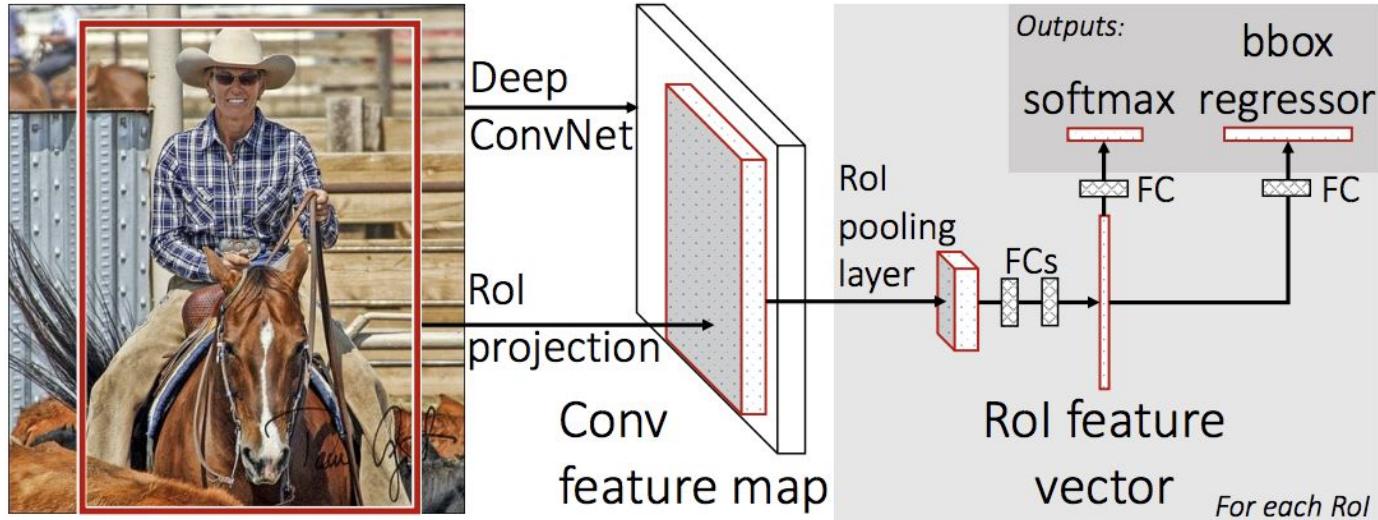
<https://www.geeksforgeeks.org/selective-search-for-object-detection-r-cnn/>

# R-CNN - Problemy

- Trenowanie to wieloetapowy proces. Obejmuje przygotowanie i trenowanie oddzielnie trzech modeli.
- Trenowanie było w związku z tym kosztowne pod względem pamięciowym i czasowym. Same warstwy konwolucyjne w zakresie tak wielu (~2000) propozycji regionów wymagały ogromnych zasobów.
- Sam proces detekcji również nie należy do najszybszych i daleko odbiega od detekcji w czasie rzeczywistym nawet na bardzo dobrym sprzęcie (zestawienie dalej).

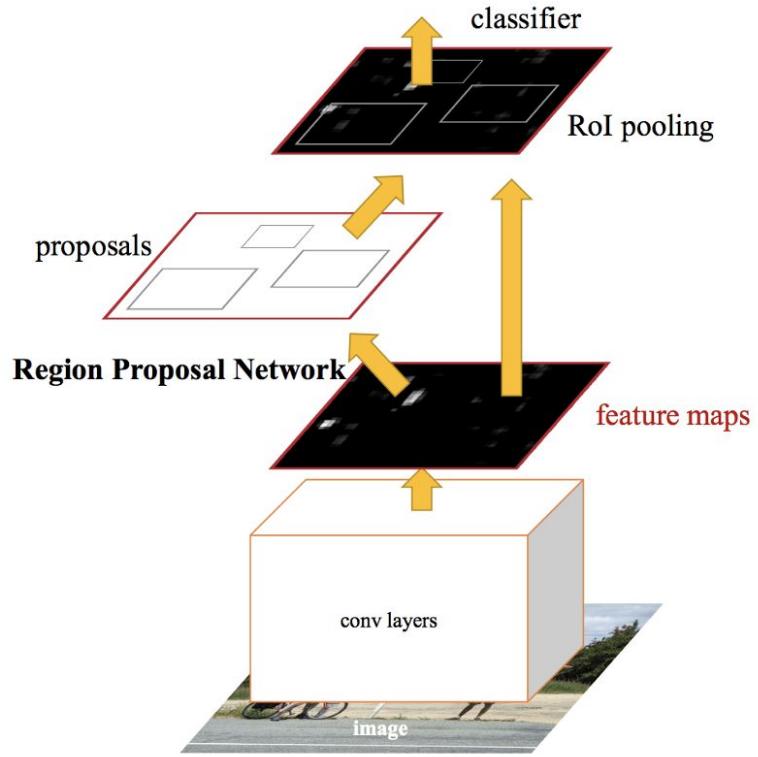
# Fast R-CNN

- Jeden wspólny model do bezpośredniego uczenia się, generowania regionów i klasyfikacji.
- Pretrenowana sieć konwolucyjna.
- Fully-connected rozdzielona na predykcje klasy oraz parametrów bbox'a (ramki okalającej obiekt).



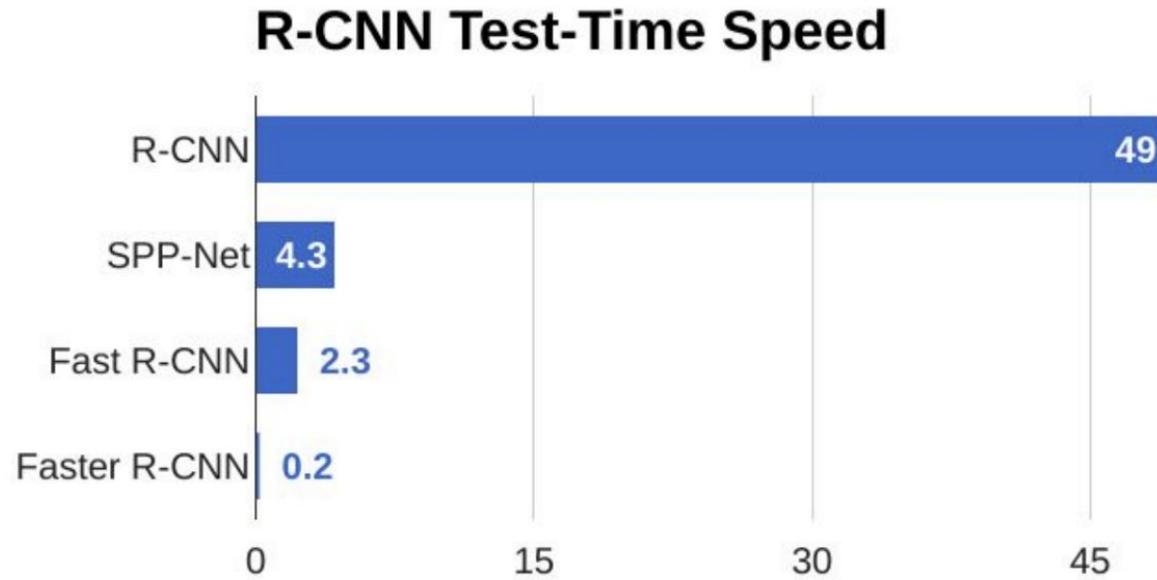
# Faster R-CNN

Oba powyższe algorytmy (R-CNN i Fast R-CNN) wykorzystują selektywne wyszukiwanie w celu odnalezienia obszarów kandydujących. W Faster R-CNN zastąpiono ten algorytm siecią neuronową. Dzięki temu algorytm poszukujący ramek ograniczających zyskał możliwość uczenia się.



# R-CNN - Wersje

Porównując prędkości działania poszczególnych wersji algorytmu należy zwrócić uwagę, że nawet Faster R-CNN umożliwia przetwarzanie jedynie 5 klatek na sekundę (oczywiście na odpowiednim sprzęcie).



# Single Shot MultiBox Detector (SSD)

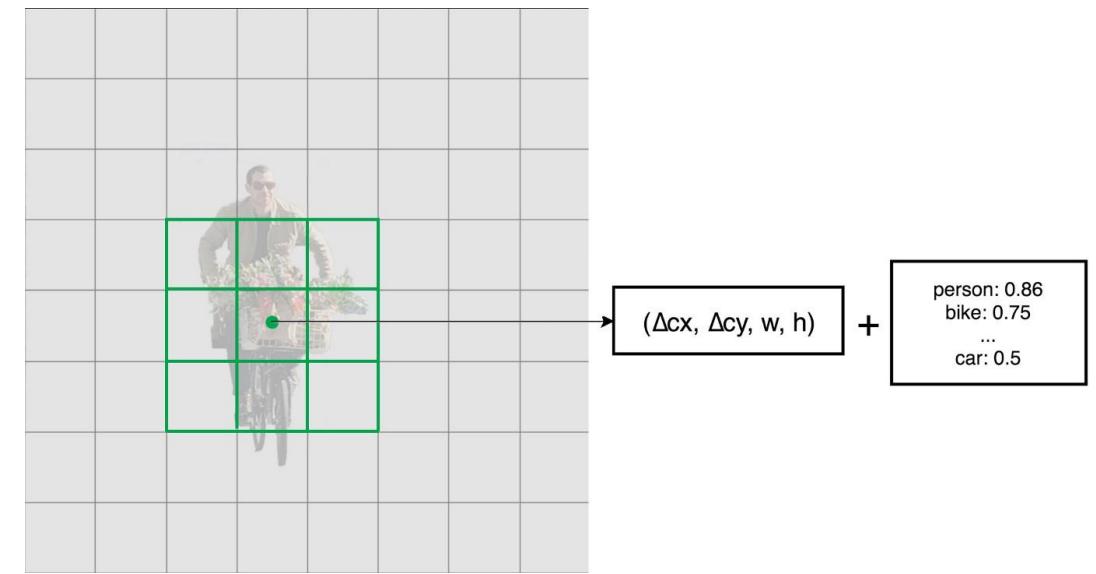
Single Shot MultiBox Detector jest, w odróżnieniu od poprzedniego omawianego algorytmu, stworzony do wykrywania obiektów w czasie rzeczywistym.

Przyśpiesza proces poprzez eliminację używania metod sztucznej inteligencji w procesie gromadzenia potencjalnych obszarów zawierających obiekt oraz jeszcze większe zmniejszenie rozdzielczości. Jest to oczywiście związane ze spadkiem jakości działania algorytmu.

# Single Shot MultiBox Detector (SSD)

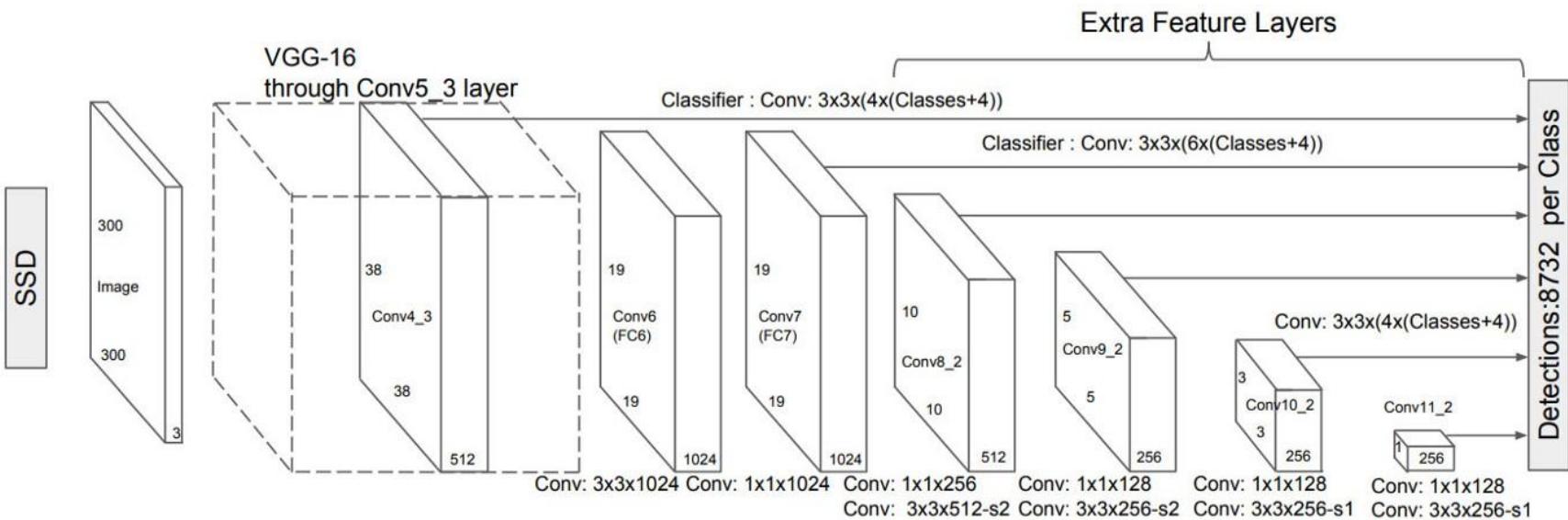
W celu określenia regionów kandydujących stosowany jest splot o kernelu 3x3 dla każdej z komórek (wydzielonej w początkowym podziale obrazu na 38x38 kwadratów), w celu dokonania predykcji dla każdej z nich.

Każdy z powstały w ten sposób filtrów wyznacza jedno pole ograniczające oraz prawdopodobieństwo wystąpienia danej klasy.



# Single Shot MultiBox Detector (SSD)

SSD używa wielu warstw (wieloskalowych map cech) do niezależnego wykrywania obiektów. Architektura VGG wraz z kolejnymi warstwami stopniowo zmniejsza wymiar przestrzenny, w związku z tym rozdzielcość tych map maleje wraz z każdą warstwą. Autorzy założyli, że z mniejszej skali łatwiej wydobywać duże obiekty, dlatego właśnie tam są poszukiwane, natomiast małe znajdowane są na pierwszych warstwach.



# Single Shot MultiBox Detector (ssd) - Podsumowanie

Można powiedzieć, że autorzy tej architektury założyli, że uproszczą działania sieci z R-CNN do minimum, a następnie postarają się poprawić jakość modelu wprowadzając operacje niewymagające obliczeniowo.

W tym celu wprowadzono:

- Małe warstwy konwolucyjne służące do klasyfikacji obiektów i przesunięć do domyślnych pól granicznych.
- Oddzielne filtry dla domyślnych bounding boxów wraz z obsługą różnych współczynników proporcji.
- Wieloskalowe feature maps do niezależnego wykrywania obiektów przy różnej deformacji obrazu.

# Yolo - Wersje

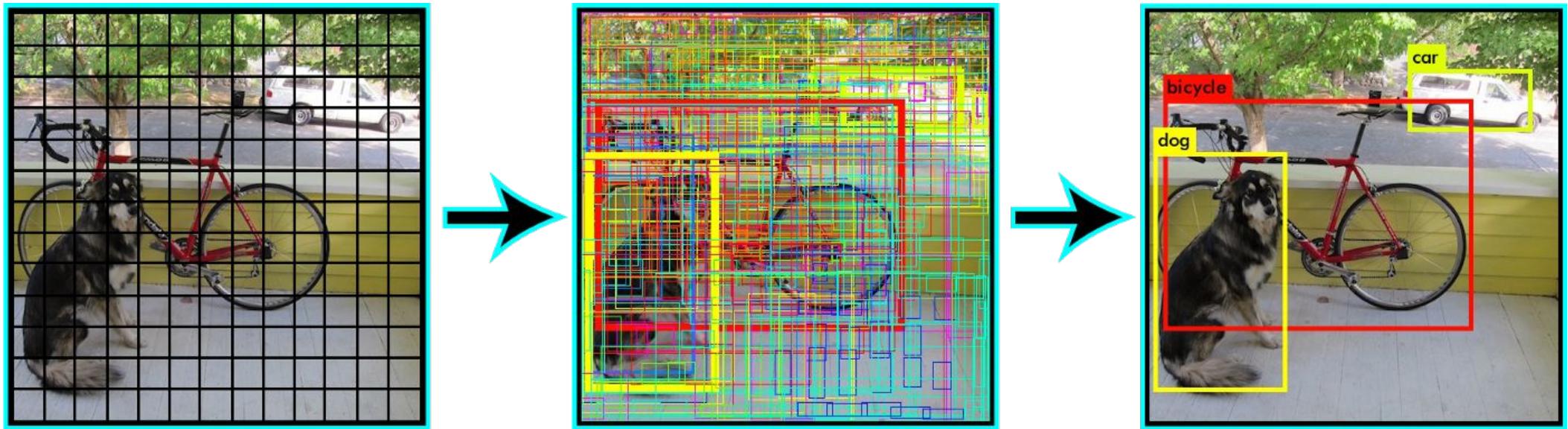
**You Only Look Once** (YOLO) to niezwykle ciekawy przypadek algorytmu przeznaczonego do detekcji obiektów w czasie rzeczywistym.

Dotychczas powstało już wiele rodzajów tej sieci, poniżej przedstawiono część tych, które warto wymienić:

- YOLO
- YOLO v2
- YOLO v3
- YOLO v4
- YOLO v5
- PP-YOLO

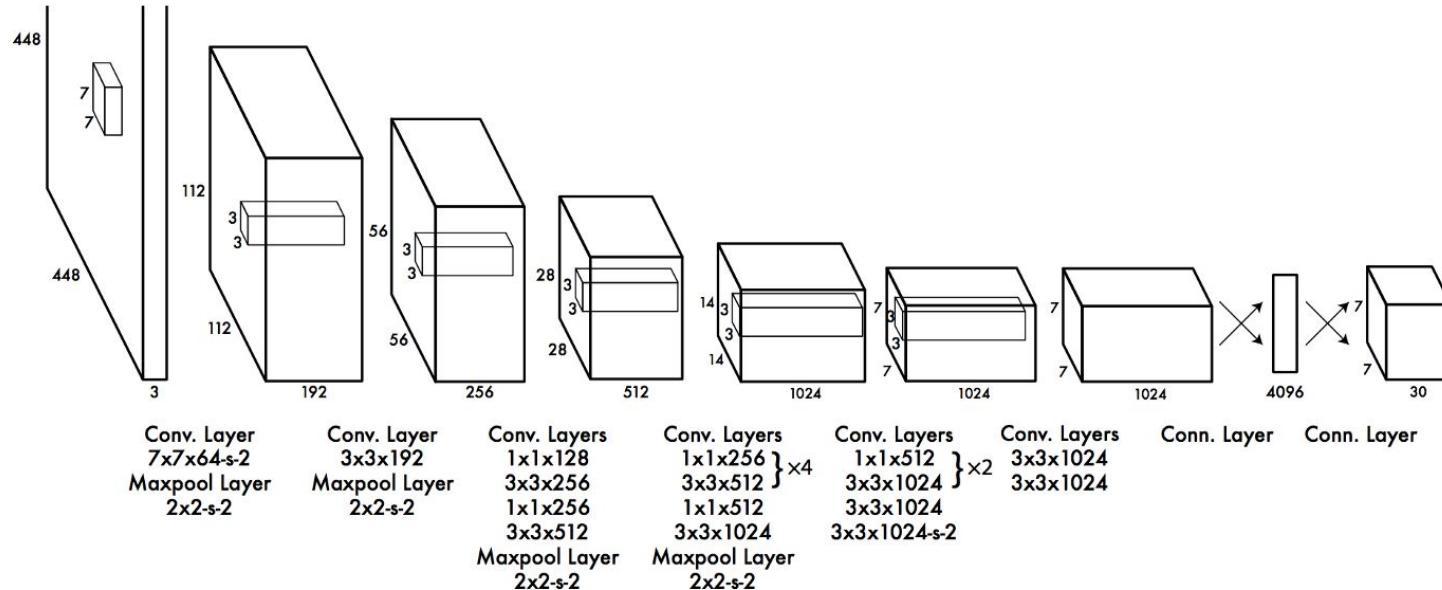
# Yolo - Działanie

YOLO dzieli obraz wejściowy na  $S \times S$  obszarów. Każdy z nich w zasadzie swojego działania ma przewidywać tylko jeden obiekt. Dodatkowo w każdej w tak powstałej komórce zapisywana jest określona ilość ramek ograniczających.



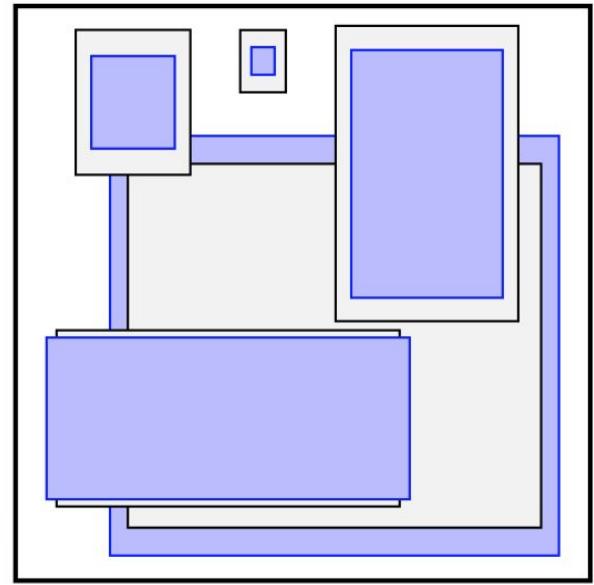
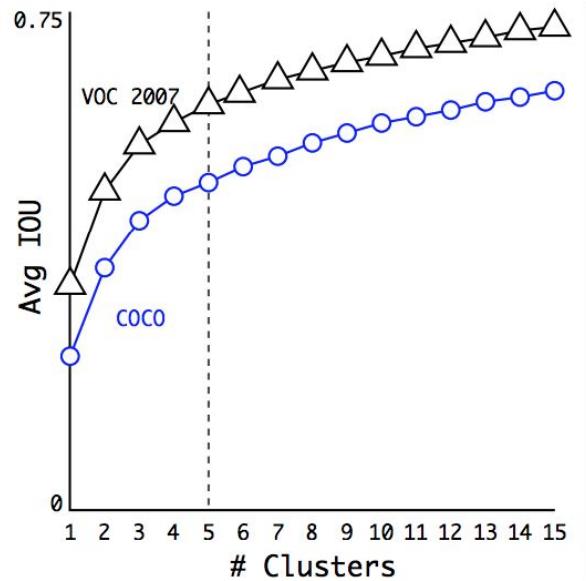
# Yolo - Działanie

Główną ideą będącą podstawą do stworzenia do tego algorytmu był pomysł, aby za pomocą zwykłej sieci konwolucyjnej i sieci w pełni połączonych wyprodukować predykcje o podanych powyżej wymiarach. W związku z tym YOLO posiada 24 warstwy konwolucyjne, po których występują 2 w pełni połączone (FC). Niektóre z warstw wykorzystują znane nam już warstwy reducyjne 1x1.



# Yolo v2

Kolejne wersje wprowadzały różnorodne usprawnienia. Np. Wersja v2 postawiła tezę, że dana klasa obiektów najczęściej posiada zazwyczaj określoną proporcję wymiarów. W związku z tym zastosowano klasteryzację do określenia najczęściej pojawiających się wymiarów:



# Yolo v3

Wersja 3 mocno  
zoptymalizowała działanie  
algorytmu kosztem niewielkiego  
spadku jakości predykcji.

Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-416	COCO trainval	test-dev	57.9	140.69 Bn	20

# Ciekawostka!

Ze względu na potęgę rozwiązań autor YOLO Joe Redmon zdecydował się w 2020 zawiesić swoją działalność naukową.



**Joe Redmon**  
@pjreddie

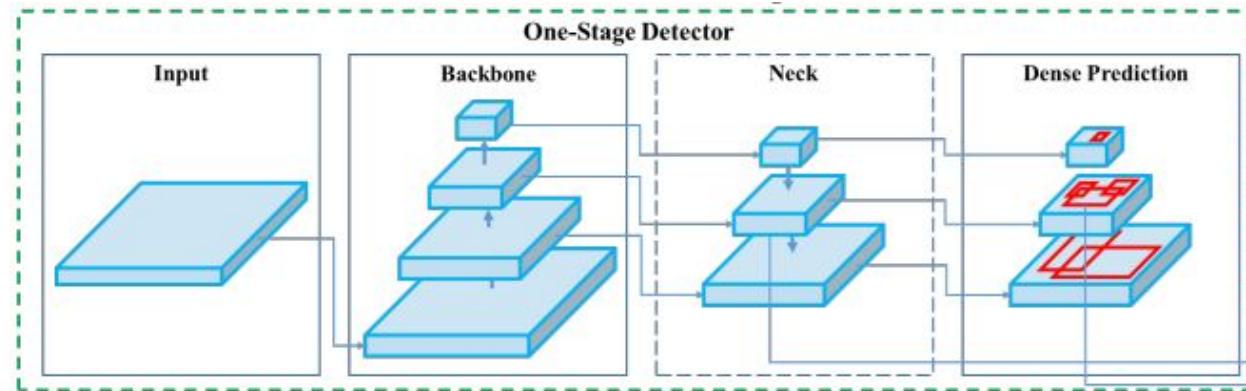
I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore.

# YOLO v4

Jednak kilka miesięcy później został opublikowany duchowy spadkobierca YOLO v3.

YOLO v4 wykorzystuje w swoim działaniu Feature Pyramid Network oraz listę usprawnień skumulowanych pod dwoma pojęciami:

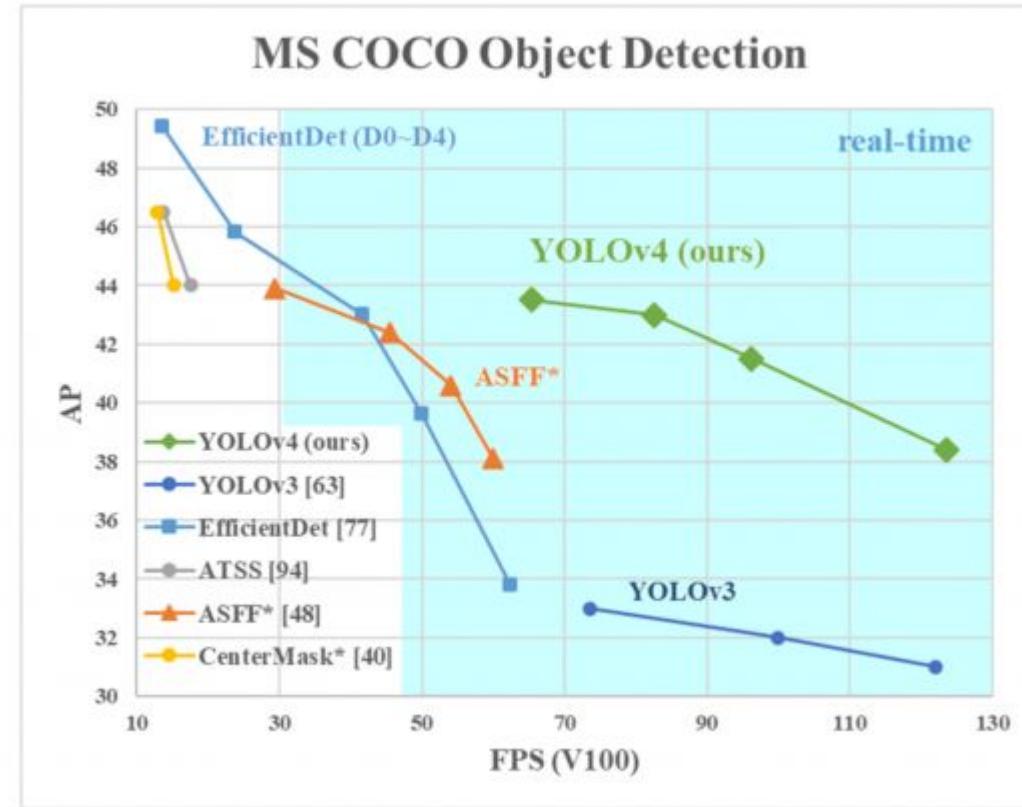
- BoF – metody poprawiające dokładność detektora bez zwiększenia czasu predykcji. Zwiększa się tylko czas szkolenia.
- BoS – metody nieznacznie zwiększające czas predykcji, przy znacznej poprawie dokładności wykrywania obiektów.



<https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>

# Object Detection - podsumowanie

Obecnie modele oparte o strukturę YOLO wyprzedziły konkurencję:



# Segmentacja

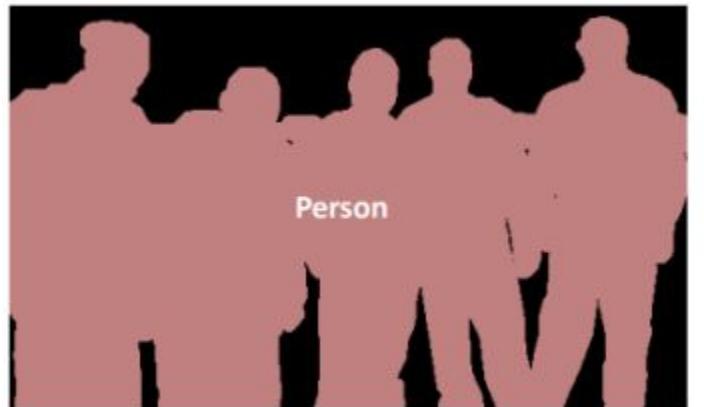
Segmentacja obrazu to proces dzielenia obrazu cyfrowego na wiele segmentów o podobnych atrybutach.



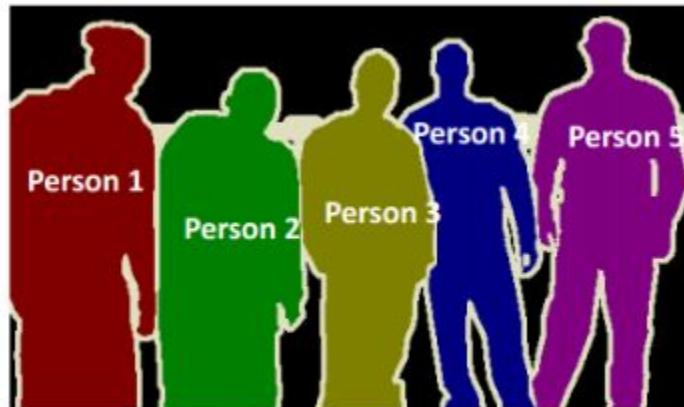
# Semantic vs Instance

Na obrazku 1 każdy piksel należy do określonej klasy (tła lub osoby). Ponadto wszystkie piksele należące do określonej klasy są reprezentowane przez ten sam kolor (czarne tło, a osoba różowy).

Obraz 2 również przypisał określoną klasę każdemu pikselowi obrazu. Jednak różne obiekty tej samej klasy mają różne kolory (Osoba 1 jako czerwona, Osoba 2 jako zielona, tło jako czarne itd.).



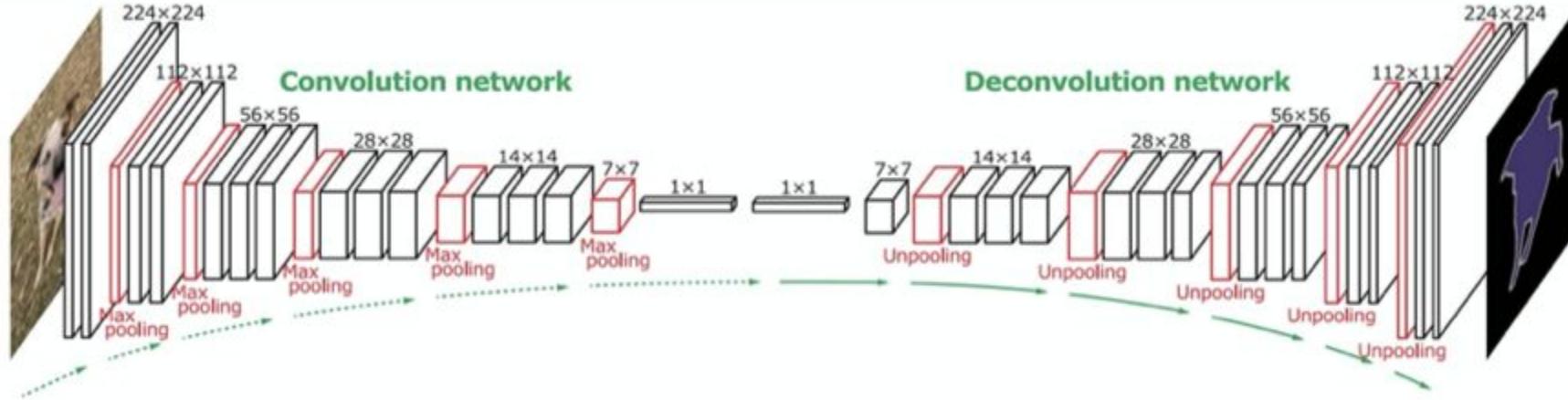
Semantic Segmentation



Instance Segmentation

# Encoder-Decoder Based Models

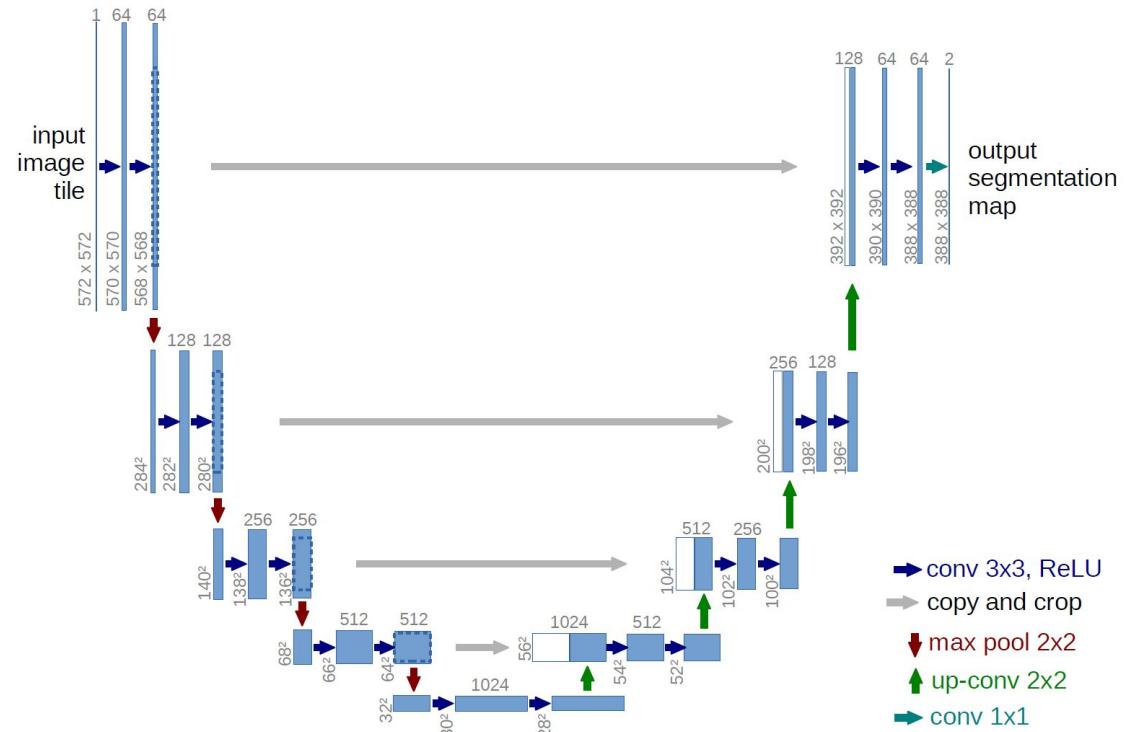
Składa się z dwóch części: kodera i dekodera. Koder wykorzystuje warstwy splotowe, podczas gdy dekoder wykorzystuje sieć dekonwolucyjną, która generuje mapę prawdopodobieństw klas pikselowych na podstawie wejściowego wektora cech.



# U-Net

U-Net to konwolucyjna architektura sieciowa do szybkiej i precyzyjnej segmentacji obrazów. Stworzony z myślą o obrazach medycznych.

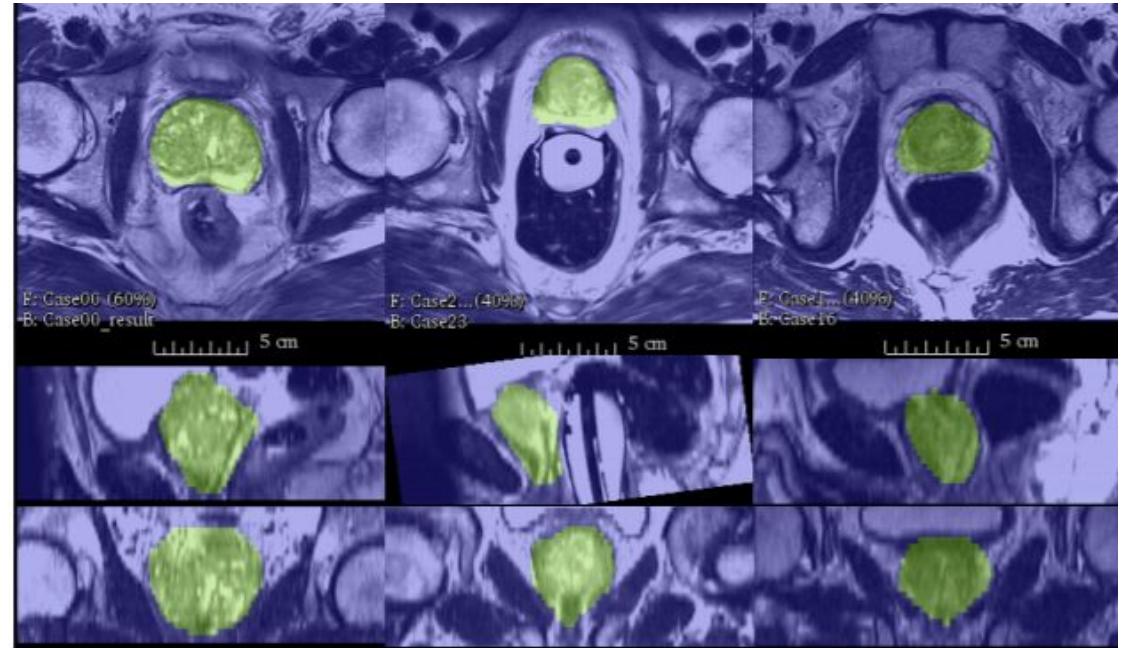
Szare pola reprezentują skopiowane feature mapy. Każde niebieskie pole odpowiada wielokanałowej mapie cech.



# V-Net

Podobny do U-Net, ale opracowany do segmentacji obrazów MRI.

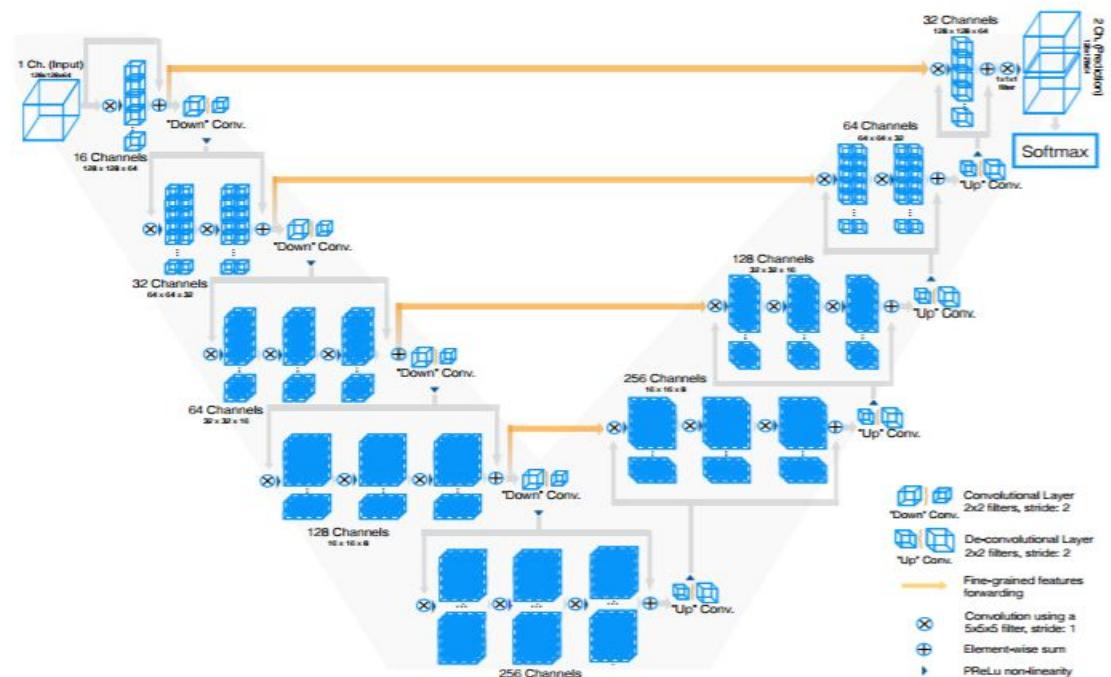
- Pooling nie jest używany.
- Na każdą warstwę nakłada się od 1 do 3 filtrów konwolucyjnych.
- Każda warstwa ma residualną funkcję.



# V-Net

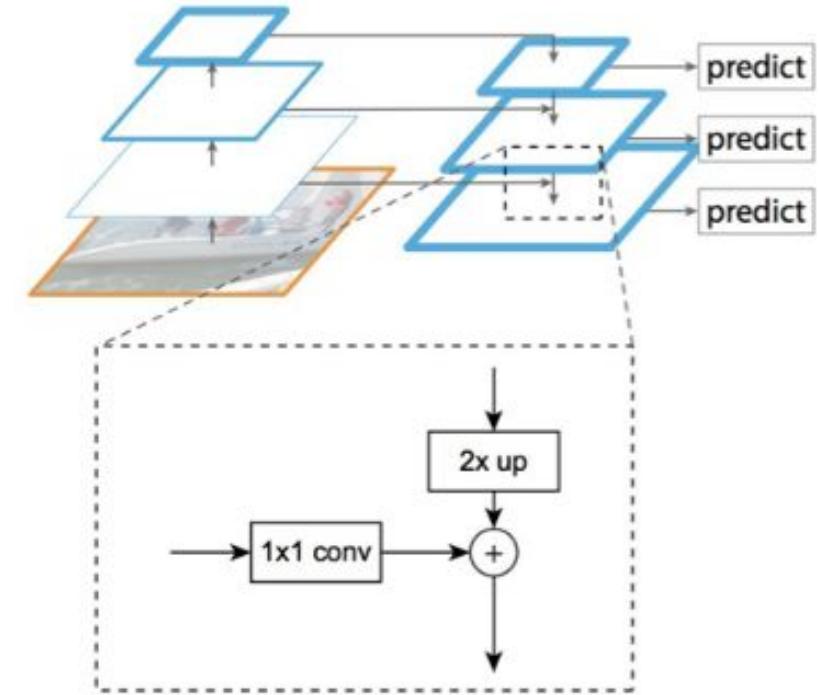
Dane wejściowe dla danego poziomu sq:

- przetwarzane przez pierwszą konwolucję na danym poziomie
- cechy z lewej części sq przesyłane do prawej części



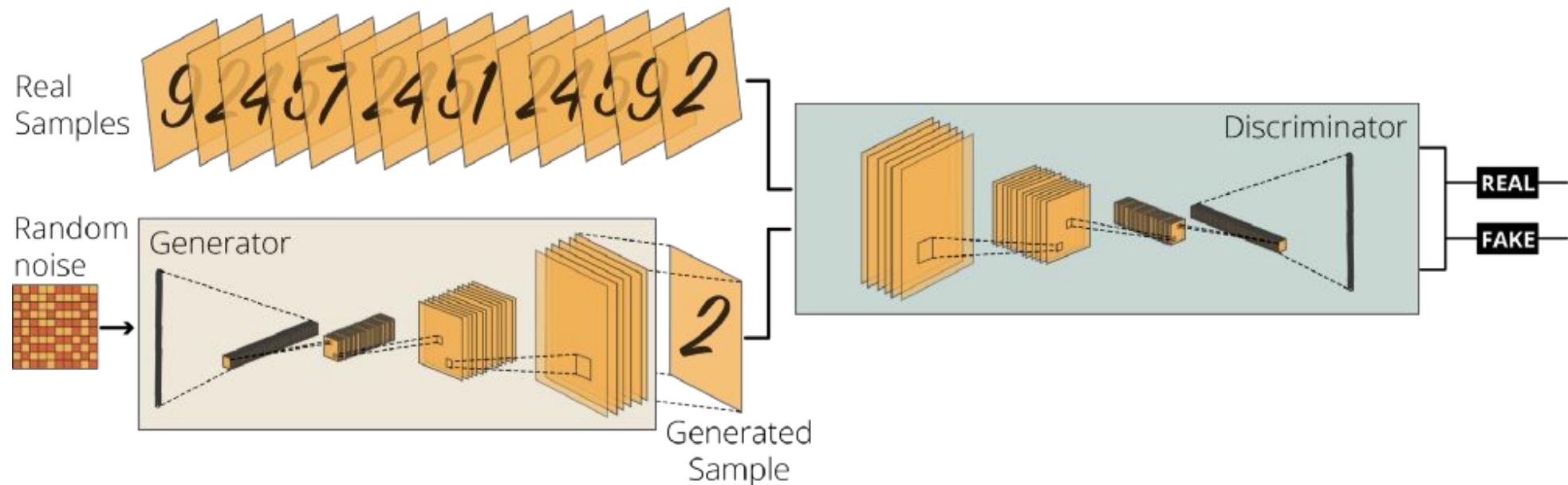
# Multi-Scale and Pyramid Network Based Models

FPN wspomniany wcześniej w kontekście YOLO został użyty również do segmentacji obrazu. Tworzy piramidę funkcji splotowych i wykorzystuje ścieżkę oddolną, ścieżkę odgórną i połączenia boczne, aby połączyć cechy o niskiej i wysokiej rozdzielcości. Następnie używa splotu  $3 \times 3$  na połączonych mapach cech, aby wygenerować dane wyjściowe każdego etapu. Wreszcie, każdy etap ścieżki odgórznej generuje prognozę wykrycia obiektu.



# Image generation

W zasadzie, aby wygenerować cokolwiek z uczeniem maszynowym, musimy użyć algorytmu generatywnego i przynajmniej na razie jednym z najlepiej działających algorytmów generujących do generowania obrazu są Generative Adversarial Networks (lub GAN).



# GAN

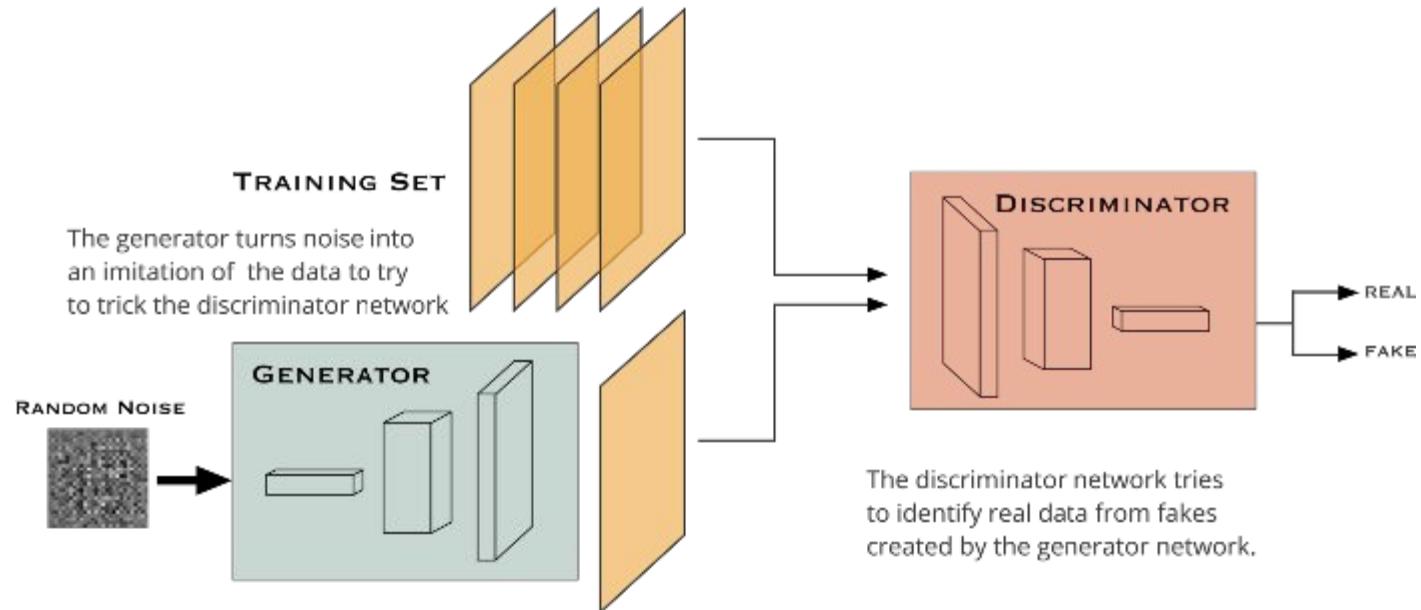
W zwykłej strukturze GAN konkurują ze sobą dwa modele:

- Generator
- Dyskryminator

Mogą być zaprojektowane przy użyciu różnych sieci (np. konwolucyjnych sieci neuronowych (CNN), rekurencyjnych sieci neuronowych (RNN) lub po prostu zwykłych sieci neuronowych (ANN lub zwykłych sieci)). Ponieważ generowane są obrazy, CNN lepiej nadają się do tego zadania.

# GAN - Działanie

Generator ma za zadanie wygenerowanie odręcznych cyfr bez podawania dodatkowych danych. Jednocześnie pobierane są istniejące odrekcne cyfry do dyskryminatora i podejmowana jest decyzja, czy obrazy generowane przez Generator są autentyczne, czy nie. Na początku Generator wygeneruje kiepskie obrazy, które zostaną natychmiast oznaczone jako fałszywe przez Dyskryminatora. Po uzyskaniu wystarczającej ilości informacji zwrotnych od Dyskryminatora, Generator nauczy się oszukiwać Dyskryminatora w wyniku zmniejszenia odchylenia od oryginalnych obrazów.



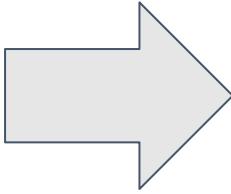
# Data Augmentation

Rozszerzanie danych w analizie danych to techniki służące do zwiększania ilości danych poprzez dodawanie nieznacznie zmodyfikowanych kopii już istniejących danych lub nowo utworzonych danych syntetycznych z istniejących danych.

Data Augmentation w szczególności znajduje swoje zastosowanie w przetwarzaniu obrazów. Python oferuje wiele bibliotek do tego zadania. Wartościową listę i zestawienie można odnaleźć pod poniższym linkiem:

<https://github.com/AgaMiko/data-augmentation-review>

# Data Augmentation



# Data Augmentation

Jeżeli chodzi o przetwarzanie obrazów to istnieje co najmniej kilka bibliotek, które oferują taką funkcjonalność:

1. Augmentor
2. Albumentations
3. ImgAug
4. AutoAugment (DeepAugment)
5. OpenCV
6. Keras, TF, MXNet, Pytorch

Różnice można znaleźć w pełnym zestawieniu:

<https://neptune.ai/blog/data-augmentation-in-python>

# Keras - ImageDataGeneration

Najprostrzym rozwiążaniem jest skorzystanie z wbudowanej klasy ImageDataGeneration, jednak:

- Nie oferuje zbyt wielu transformacji
- Działa stosunkowo wolno
- Wspiera jedynie przetwarzanie obrazów do zagadnień z zakresu Klasyfikacji
- Zamknięte na modyfikację

# ImgAug (imgaug)

ImgAug pozwala na przeprowadzanie Data Augmentation dla różnych zagadnień z zakresu Przetwarzania obrazu:

	Image	Heatmaps	Seg. Maps	Keypoints	Bounding Boxes, Polygons
Original Input					
Gauss. Noise + Contrast + Sharpen					
Affine					
Crop + Pad					
Flplr + Perspective					

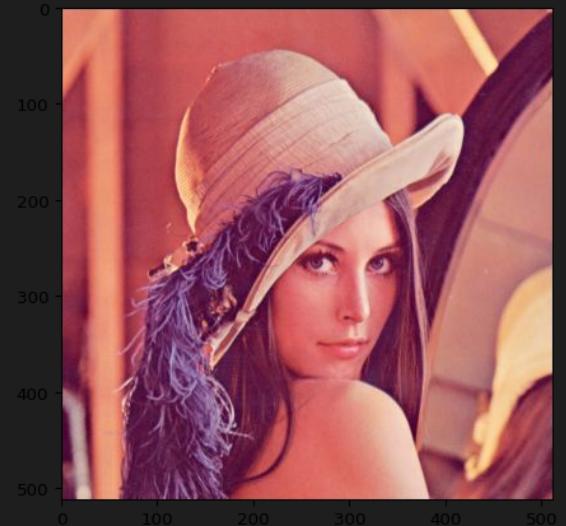
# ImgAug (imgaug)

Oferuje bardzo dużo możliwych transformacji obrazu ->



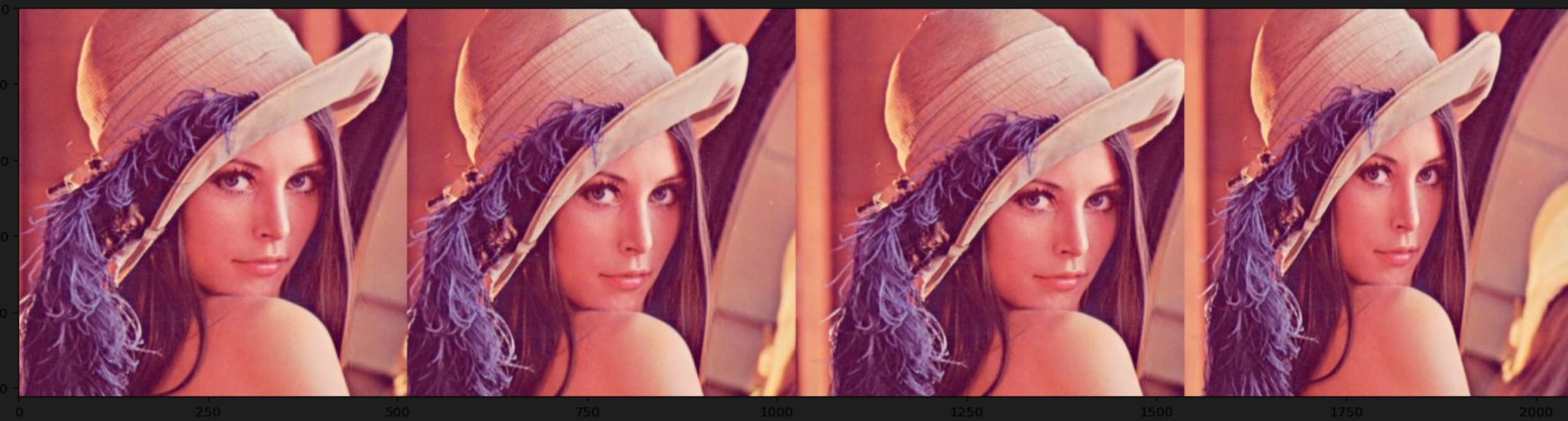
# Wyświetlanie obrazów za pomocą imgaug

```
from imgaug import augmenters as iaa  
ia.seed(4)  
  
images = [image, image, image, image]  
  
rotate = iaa.Affine(rotate=(-25, 25))  
image_aug = rotate(images=images)  
  
print("Augmented:")  
ia.imshow(np.hstack(image_aug))
```



```
from imgaug import augmenters as iaa  
ia.seed(4)  
  
images = [image, image, image, image]  
  
rotate = iaa.Crop(percent=(0, 0.2))  
image_aug = rotate(images=images)  
  
print("Augmented:")  
ia.imshow(np.hstack(image_aug))
```

# Aplikowanie operacji na obrazach



```
# Sekwencje
images = [image, image, image]

seq = iaa.Sequential([
    iaa.Affine(rotate=(-25, 25)),
    iaa.AdditiveGaussianNoise(scale=(10, 60)),
    iaa.Crop(percent=(0, 0.2))
])

images_aug = seq(images=images)

print("Augmented:")
ia.imshow(np.hstack(images_aug))
```

# Sekwencje



```
# Sometimes
images = [image, image, image, image]
def sometimes(aug):
    return iaa.Sometimes(0.5, aug)

seq = iaa.Sequential([sometimes(iaa.Affine(rotate=(-25, 25)))]))

images_aug = seq(images=images)

print("Augmented:")
ia.imshow(np.hstack(images_aug))
```

# Sometimes



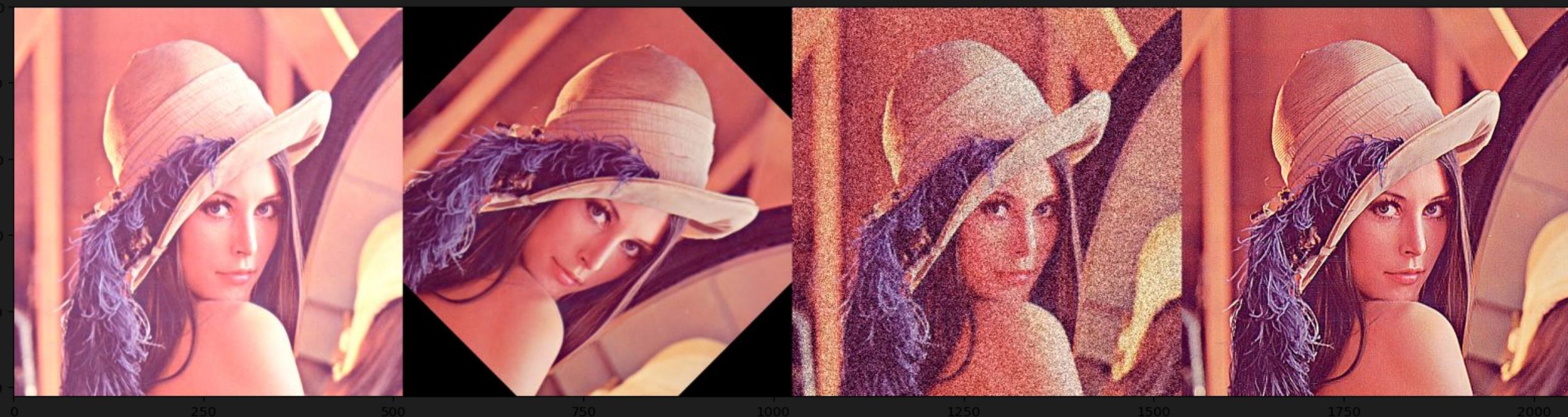
```
# iaa.OneOf
images = [image, image, image, image]

seq = iaa.OneOf([
    iaa.Affine(rotate=45),
    iaa.AdditiveGaussianNoise(scale=0.2*255),
    iaa.Add(50, per_channel=True),
    iaa.Sharpen(alpha=0.5)
])

images_aug = seq(images=images)

print("Augmented:")
ia.imshow(np.hstack(images_aug))
```

# OneOf



# Podejścia w Data Augmentation

- Generowanie obrazów w trakcie trenowania modelu
- Wygenerowanie obrazów przed trenowaniem modelu

# Jakie operacje warto przeprowadzać w kontekście CNN?

- Warto zwrócić uwagę na czas wykonywania poszczególnych operacji, wykonywanie Data Augmentation w trakcie trwania programu może znacznie wydłużyć czas jego wykonywania.

<https://imgaug.readthedocs.io/en/latest/source/performance.html?highlight=performance>

- Warto skupić się na operacjach niemożliwych do osiągnięcia poprzez filtry konwolucyjne (crop, rotate itp.).

# Implementacja

W celu stworzenia wydajnego i sprężonego z Kerasem narzędzia do Augmentacji warto stworzyć własną klasę generującą dane, dziedziczącą po keras.utils.Sequence. Klasa ta powinna implementować metody:

- `__init__`
- `__len__`
- `on_epoch_end`
- `__get_item__`

```
import keras
import numpy as np
import imgaug.augmenters as iaa

class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, images, labels, batch_size=64, shuffle=False, augment=False):
        self.labels = labels                      # array of labels
        self.images = images                       # array of images
        self.batch_size = batch_size                # batch size
        self.shuffle = shuffle                     # shuffle bool
        self.augment = augment                    # augment data bool
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.images) / self.batch_size))

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.images))
        if self.shuffle:
            np.random.shuffle(self.indexes)

    def __getitem__(self, index):
        pass

    def augmentor(self, images):
        pass
```

# Klasa Generatora

# Augmentor

```
def augmentor(self, images):
    'Apply data augmentation'
    def sometimes(aug):
        return iaa.Sometimes(0.5, aug)
    seq = iaa.Sequential([sometimes(iaa.Crop(px=(1, 16), keep_size=True)),
                          sometimes(iaa.Fliplr(0.5)),
                          sometimes(iaa.GaussianBlur(sigma=(0, 3.0)))])
    return seq.augment_images(images)
```

```
def __getitem__(self, index):
    'Generate one batch of data'
    # selects indices of data for next batch
    indexes = self.indexes[index * self.batch_size :
                           (index + 1) * self.batch_size]

    # select data and load images
    labels = np.array([self.labels[k] for k in indexes])
    images = np.array([self.images[k] for k in indexes])

    # preprocess and augment data
    if self.augment == True:
        images = self.augmentor(images)

    images=images/ 255

    return images, labels
```

## \_\_get\_item\_\_

# Porównanie:

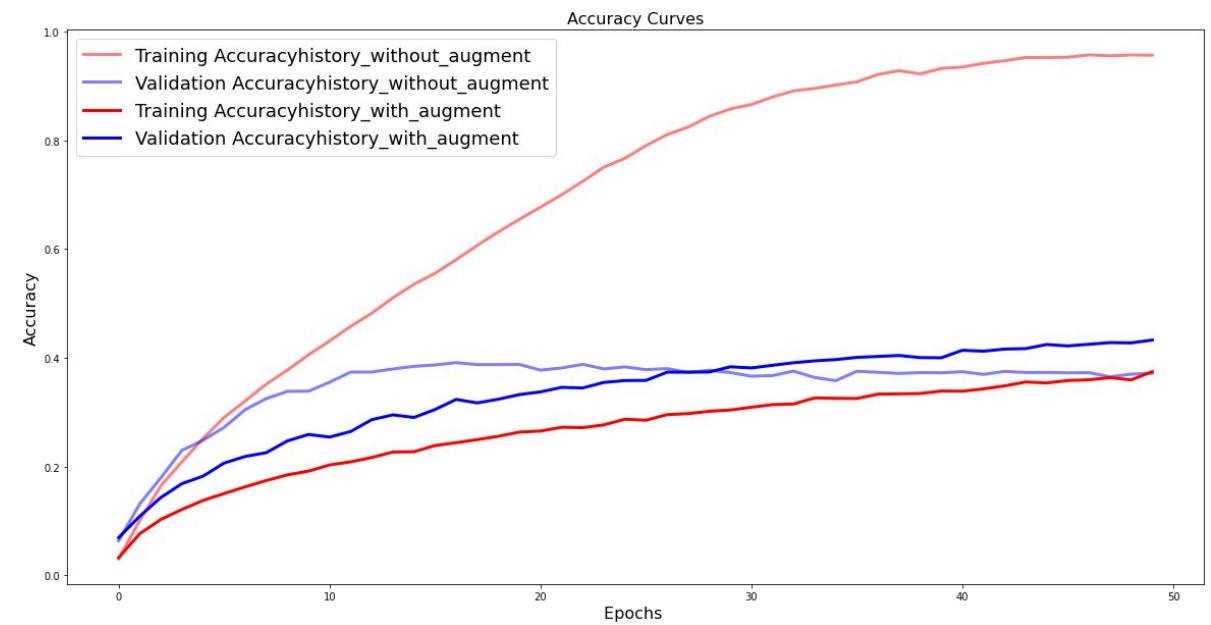
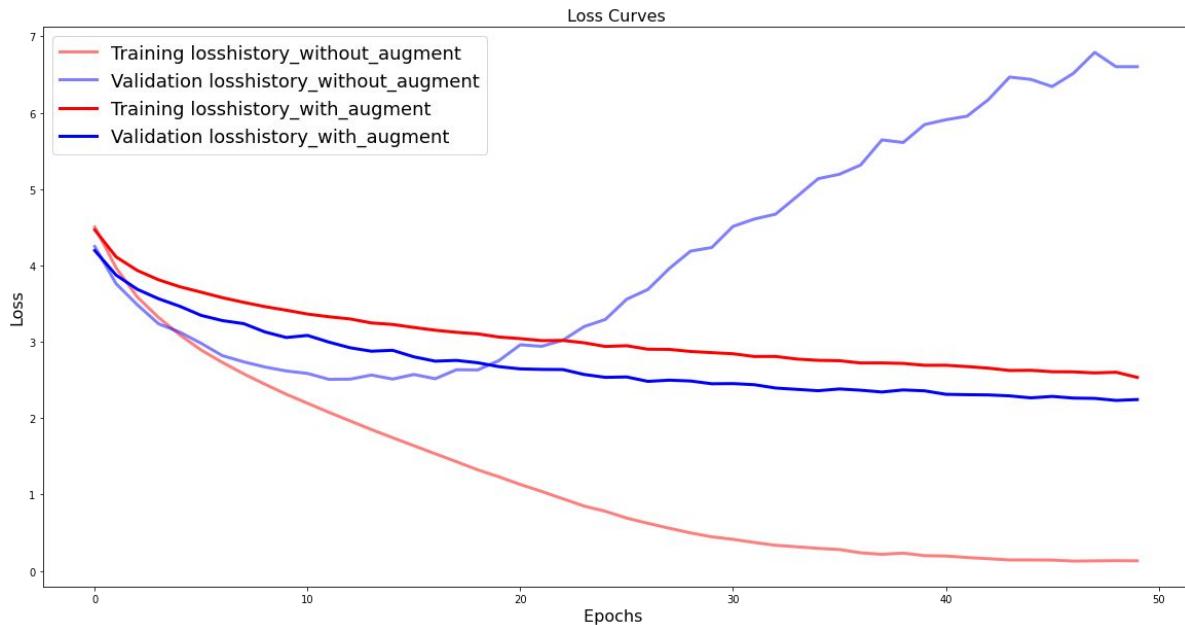
```
generator = DataGenerator(images=train_x, labels=train_y_one_hot,
                           batch_size=64, shuffle=True, augment=False)           generator = DataGenerator(images=train_x, labels=train_y_one_hot,
                           batch_size=64, shuffle=True, augment=True)

#es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min',
verbose=1, patience=2)                         #es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1,
                                                patience=2)

history_without_augment = model.fit_generator(generator, epochs=50,
                                               verbose=True,                                     history_with_augment = model.fit_generator(generator, epochs=50, verbose=True,
validation_data=(test_x, test_y_one_hot))      validation_data=(test_x, test_y_one_hot))

#callbacks = [es])                                callbacks = [es])
```

# Porównanie:



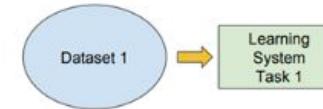
Udało się poprawić wynik o kilka procent!

# Transfer Learning

W Transfer Learning wiedza o już wyszkolonym modelu uczenia maszynowego jest stosowana do innego, ale pokrewnego problemu. Na przykład, jeśli mamy klasyfikator kotków, to możemy wykorzystać część warstw do klasyfikacji np. psów. Dzięki temu, w zasadzie próbujemy wykorzystać to, czego nauczyliśmy się w jednym zadaniu, aby poprawić generalizację w innym. Przenosimy wiedzę, które sieć nauczyła się w zadaniu A, do nowego zadania B.

## Traditional ML

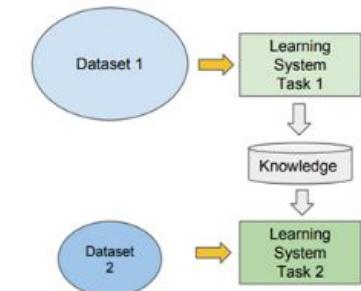
- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

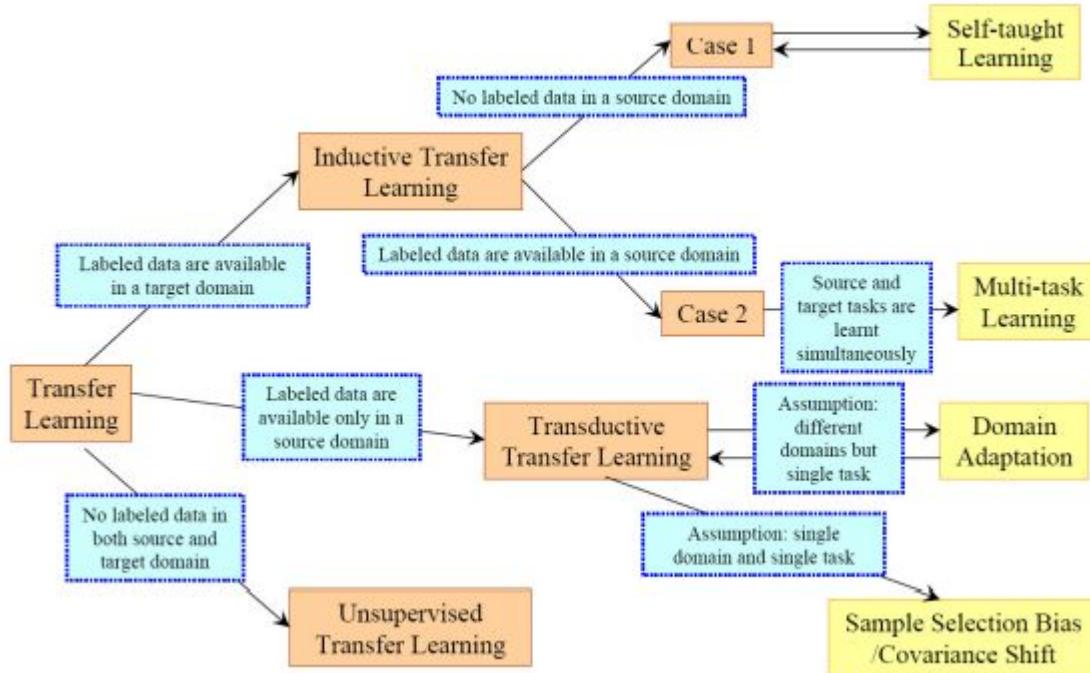
## Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



# Transfer Learning - strategie

Istnieją różne strategie i techniki transfer learningu, które można zastosować w zależności od domeny, wykonywanego zadania i dostępności danych.



[https://www.cse.ust.hk/~qyang/Docs/2009/tkde\\_transfer\\_learning.pdf](https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf)

# Domena vs Zadanie

**To samo zadanie, różna domena**

0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9



**Ta sama domena, różne zadanie**



<https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a>

# Inductive Transfer Learning

W tym scenariuszu domena źródłowa i docelowa są takie same, ale zadania źródłowe i docelowe różnią się od siebie. Posiadamy natomiast dane ze zbioru docelowego.

W zależności od tego, czy domena źródłowa zawiera oznaczone dane, czy nie, można uwzględnić je na etapie retrenowania modelu.

# Unsupervised Transfer Learning

- To ustawienie jest podobne do Inductive Transfer Learning, z naciskiem na uczenie nienadzorowane.
- Domeny źródłowa i docelowa są podobne, ale zadania są różne.
- W tym scenariuszu dane oznaczone etykietami są niedostępne zarówno dla źródła, jak i docelowego zbioru danych.

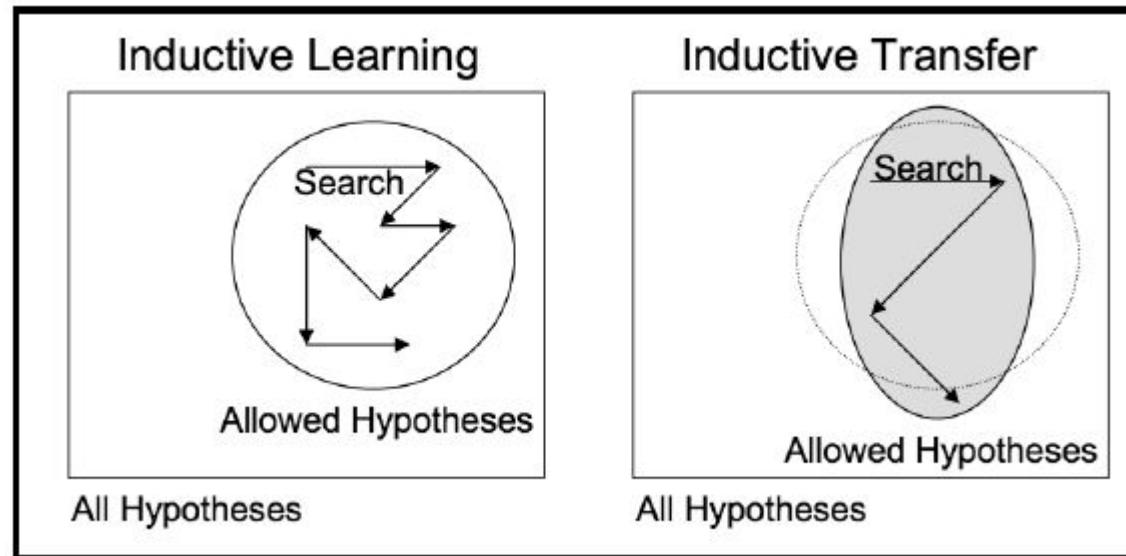
# Transductive Transfer Learning

W tym scenariuszu istnieją podobieństwa między zadaniami źródłowymi i docelowymi, ale odpowiadające im domeny są różne.

W tym ustawieniu domena źródłowa zawiera wiele oznaczonych danych, a domena docelowa żadnych.

# Deep Transfer Learning Strategies

W przypadku Deep Learningu najczęściej zastosowań ma podejście Inductive Transfer Learning.



# Transfer Learning - ogólny pipeline

1. Wybierz model źródłowy. Wstępnie wytrenowany model źródłowy jest wybierany spośród dostępnych. Wiele instytucji badawczych publikuje wagi wytrenowane na dużych i trudnych zbiorach danych.
2. Użyj modelu ponownie. Model wstępnie wytrenowany może być następnie użyty jako punkt wyjścia dla modelu drugiego. Może to obejmować użycie całości lub części modelu.
3. Dostrój model. Dostosowania na podstawie nowych danych dostępnych dla nowego zadania.

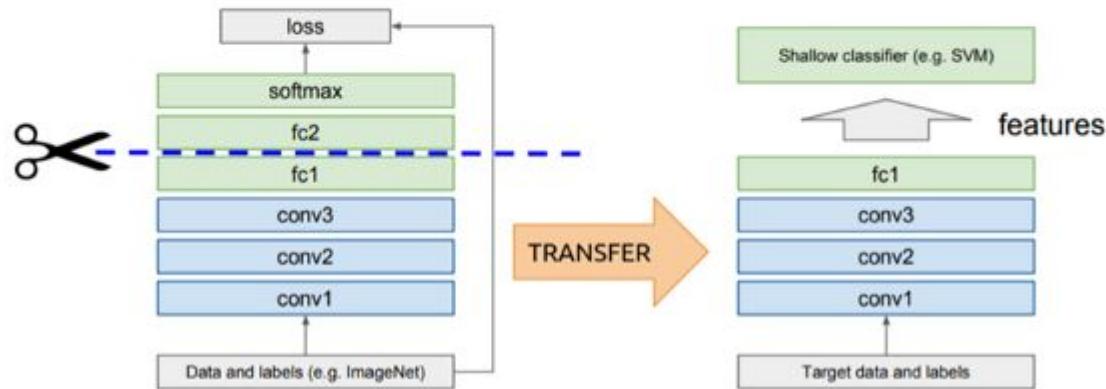
# Transfer Learning - Rodzaje

- Off-the-shelf Pre-trained Models as Feature Extractors
- Fine Tuning Off-the-shelf Pre-trained Models

# Off-the-shelf Pre-trained Models as Feature Extractors

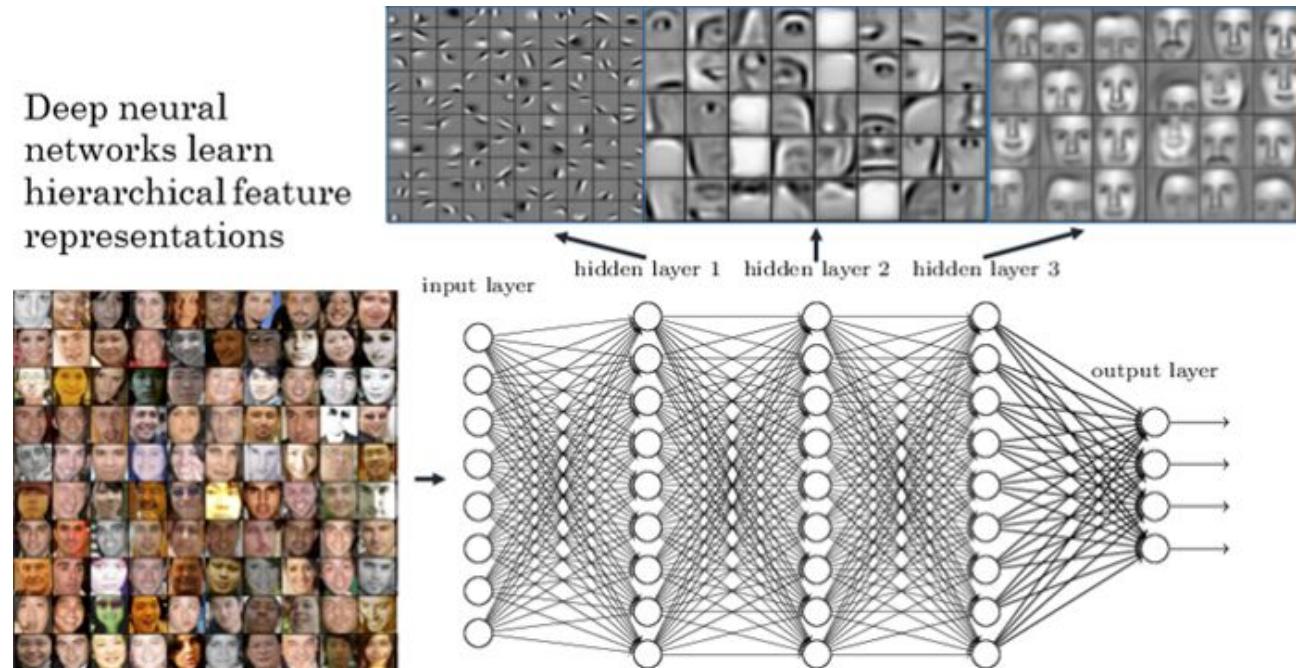
Systemy i modele uczenia głębokiego to architektury warstwowe, które uczą się hierarchicznej reprezentacji cech obrazu. Warstwy te są następnie ostatecznie łączone z ostatnią warstwą (np. Fully-Connected), aby uzyskać ostateczny wynik.

Ta wielowarstwowa architektura pozwala nam wykorzystać wstępnie wytrenowaną sieć bez jej ostatniej warstwy jako ekstraktora cech.



# Fine Tuning Off-the-shelf Pre-trained Models

Jest to bardziej skomplikowana technika, w której nie tylko zastępujemy ostatnią warstwę (do klasyfikacji / regresji), ale także selektywnie ponownie uczymy niektóre z poprzednich warstw. Początkowe warstwy skupiają się na wydobywaniu "jakichkolwiek" cech, podczas gdy późniejsze skupiały się bardziej na konkretnym zadaniu.



# Fine Tuning Off-the-shelf Pre-trained Models

Korzystając z tego, możemy zamrozić (zablokować wagi) określone warstwy podczas ponownego szkolenia i przeprowadzić fine-tuning wybranych warstw.

## Fine-tuning: supervised domain adaptation

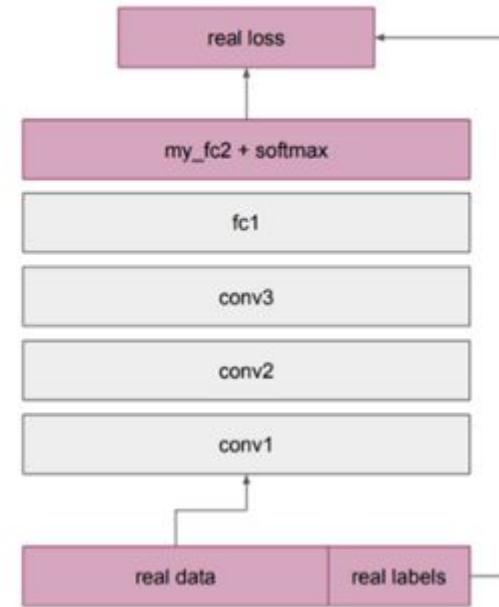
Train deep net on "nearby" task for which it is easy to get labels using standard backprop

- E.g. ImageNet classification
- Pseudo classes from augmented data
- Slow feature learning, ego-motion

Cut off top layer(s) of network and replace with supervised objective for target domain

**Fine-tune** network using backprop with labels for target domain until validation loss starts to increase

Aligns  $D_S$  with  $D_T$



# Freezing czy Fine-tuning?

Ille warstw zamrażać, a na ilu przeprowadzać fine-tuning?

## Freeze or fine-tune?

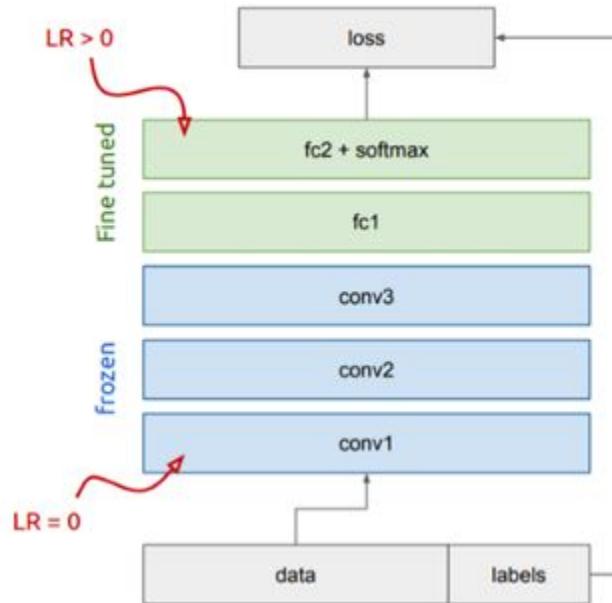
Bottom  $n$  layers can be frozen or fine tuned.

- **Frozen:** not updated during backprop
- **Fine-tuned:** updated during backprop

Which to do depends on target task:

- **Freeze:** target task labels are scarce, and we want to avoid overfitting
- **Fine-tune:** target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



```
import keras
from tensorflow.keras.applications import EfficientNetB0
from keras.datasets import cifar100
from keras.models import Sequential
from keras.layers import UpSampling2D, GlobalAveragePooling2D, Dense,
Dropout, BatchNormalization

efficient_model = EfficientNetB0(weights='imagenet', include_top=False,
input_shape=(32, 32, 3))

(x_train, y_train), (x_test, y_test) = cifar100.load_data()

x_test = x_test/255

print(x_test.shape)

# Change the labels from integer to categorical data
train_y_one_hot = keras.utils.to_categorical(y_train)
test_y_one_hot = keras.utils.to_categorical(y_test)
```

# Transfer Learning!

```
model = Sequential()

model.add(efficient_model)

model.add(GlobalAveragePooling2D())
model.add(Dense(256, activation='relu'))
model.add(Dropout(.25))
model.add(BatchNormalization())

num_classes = 100
model.add(Dense(num_classes, activation='softmax'))

for layer in efficient_model.layers:
    if isinstance(layer, BatchNormalization):
        layer.trainable = True
    else:
        layer.trainable = False
```

# Transfer Learning!

```
generator = DataGenerator(images=x_train, labels=train_y_one_hot,  
batch_size=64,  
shuffle=True, augment=True)
```

```
es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min',  
verbose=1, patience=2)
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy',  
metrics=['accuracy'])
```

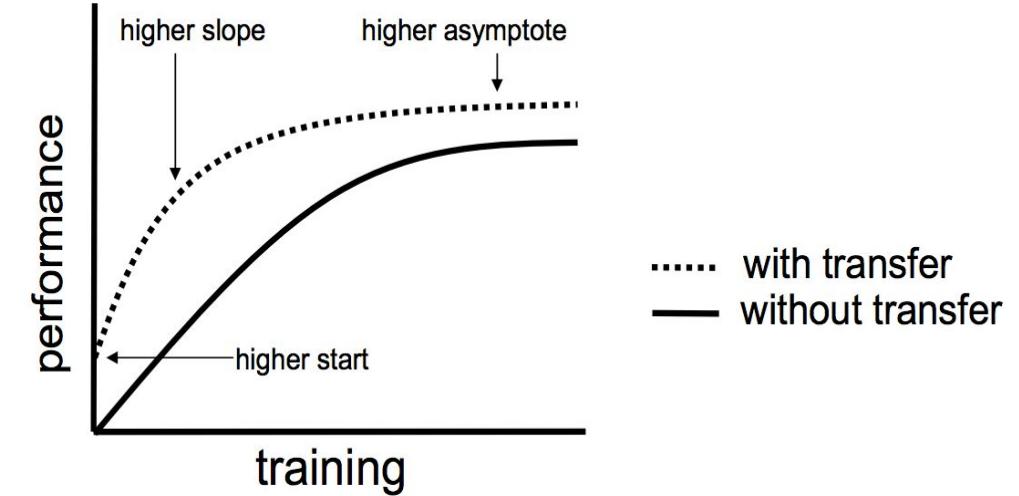
```
history = model.fit_generator(generator, epochs=50, verbose=True,  
validation_data=(x_test, test_y_one_hot),  
callbacks=[es])
```

# Transfer Learning!

# Transfer Learning - podsumowanie

Transfer Learning to czysta optymalizacja, droga do oszczędzania czasu lub uzyskania lepszej wydajności. Szczególnie przydatne kiedy posiadamy małą ilość danych. Pozwala na:

- "Wysoki" start. Początkowe accuracy podczas uczenia nowego modelu jest wyższe.
- Większe nachylenie. Tempo poprawy jakości modelu podczas trenowania jest większe.
- Wyższa asymptota. Maksymalna jakość osiągana przez wytrenowany model jest wyższa.



# Przetwarzanie obrazów - podsumowanie

- Rozwój metod przetwarzania obrazów jest ogromny. Każdego roku ten “świat” się zmienia.
- Część algorytmów/schematów się jednak powtarza, dlatego warto znać możliwie dużo algorytmów.
- Technologie CV stają się coraz bardziej dostępne i przystępne w użyciu.
- Jeżeli chcemy wydobyć pełnię możliwości z danego algorytmu należy go dogłębnie zrozumieć.

# Przydatne linki:

- [http://deeplearning.csail.mit.edu/instance\\_ross.pdf](http://deeplearning.csail.mit.edu/instance_ross.pdf)
- [http://vision.stanford.edu/teaching/cs231b\\_spring1213/slides/HOG\\_2011\\_Stanford.pdf](http://vision.stanford.edu/teaching/cs231b_spring1213/slides/HOG_2011_Stanford.pdf)
- <https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>
- <https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109>
- [https://www.cse.ust.hk/~qyang/Docs/2009/tkde\\_transfer\\_learning.pdf](https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf)
- <https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a>