





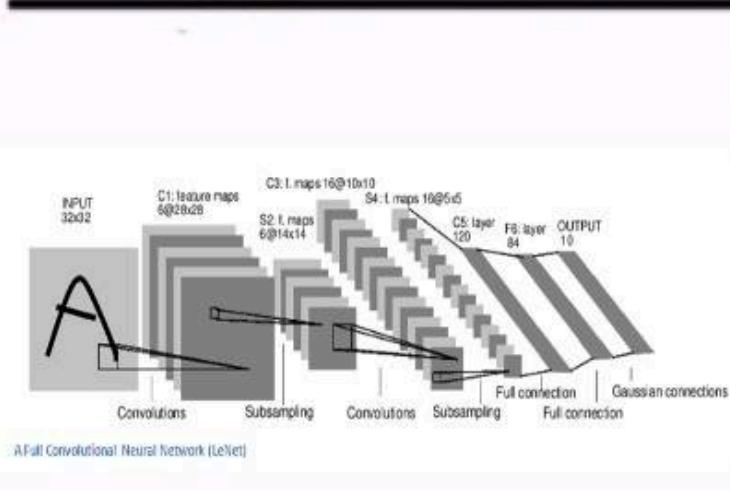
memegenerator.net



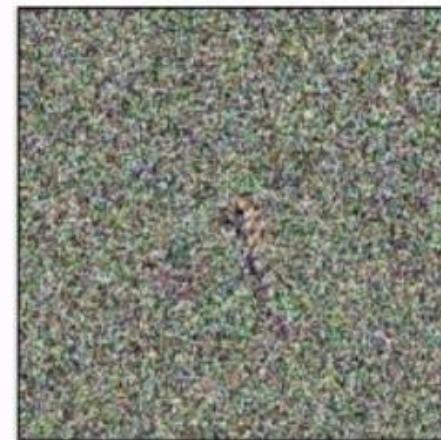
18:30 - turbogofry!



WHO WOULD WIN?



a complex model designed
by elite researchers



cheetah

one noisy boi

Transformacja obrazu w NN



zwiększenie
podatności na
błędy

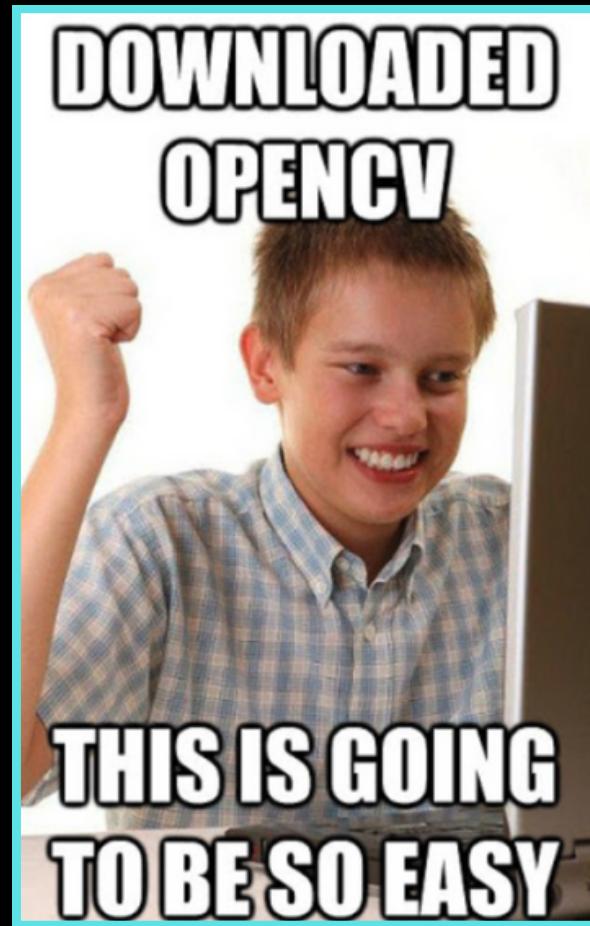


sprawdzenie
precyzji
przewidywania



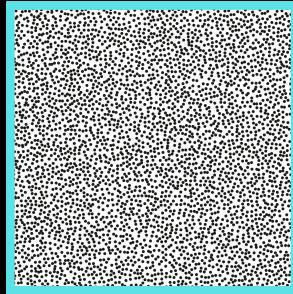
pozbycie się
niepożądanych
artefaktów

Biblioteka OpenCV



pip install opencv-python

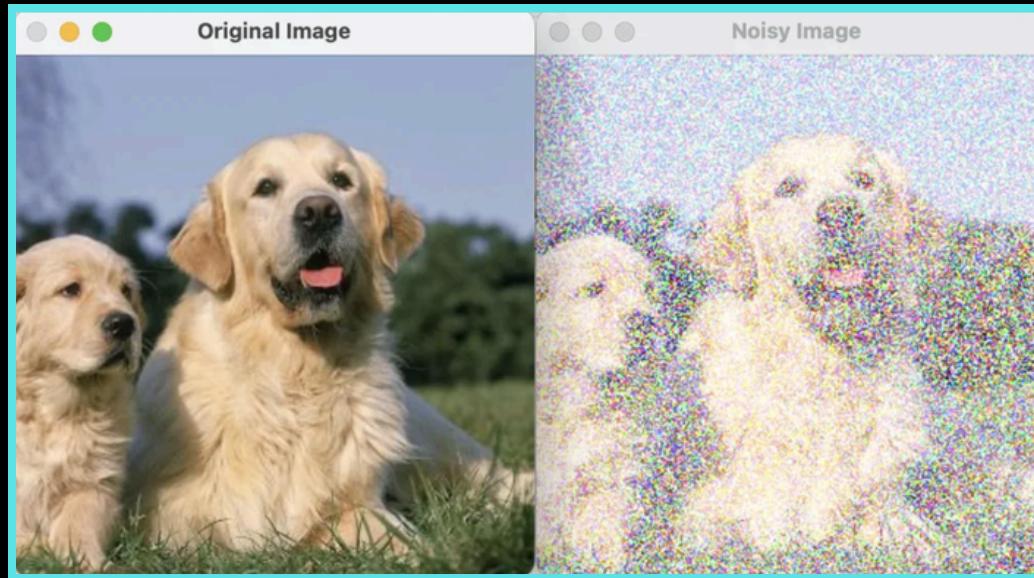
Transformacje - Szum



Na obrazach w datasetie może znajdować się szum, który zniekształca identyfikowany obiekt, uniemożliwiając poprawne trenowanie sieci.

Transformacje - Szum Gaussowski

przejawia się kolorowymi artefaktami i/lub przejaśnieniem obrazu



<https://www.askpython.com/python/examples/adding-noise-images-opencv>

Transformacje - Sztum Gaussowski

```
import cv2
import numpy as np

def add_gaussian_noise(image, mean=0, std=25):
    noise = np.random.normal(mean, std, image.shape).astype(np.uint8)
    noisy_image = cv2.add(image, noise)
    return noisy_image

original_image = cv2.imread('testimg.jpeg')

if original_image is None:
    raise Exception("Image not loaded properly. Check the file path.")

noisy_image = add_gaussian_noise(original_image, mean=0, std=25)
```

<https://www.askpython.com/python/examples/adding-noise-images-opencv>

Transformacje - Szum Binarny

znany też jako “sól i pieprz” przejawia się czarnymi i białymi punktami na obrazie



<https://www.askpython.com/python/examples/adding-noise-images-opencv>

Transformacje - Szum Binarny

```
import cv2
import numpy as np


def add_salt_and_pepper_noise(image, noise_ratio=0.02):
    noisy_image = image.copy()
    h, w, c = noisy_image.shape
    noisy_pixels = int(h * w * noise_ratio)

    for _ in range(noisy_pixels):
        row, col = np.random.randint(0, h), np.random.randint(0, w)
        if np.random.rand() < 0.5:
            noisy_image[row, col] = [0, 0, 0]
        else:
            noisy_image[row, col] = [255, 255, 255]

    return noisy_image

original_image = cv2.imread('testimg.jpeg')

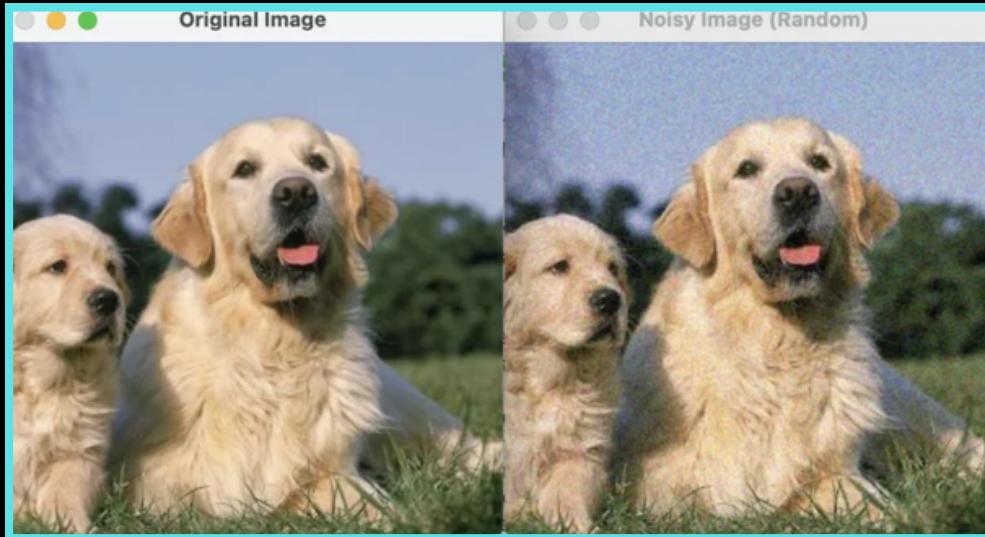
if original_image is None:
    raise Exception("Image not loaded properly. Check the file path.")

noisy_image = add_salt_and_pepper_noise(original_image, noise_ratio=0.02)
```

<https://www.askpython.com/python/examples/adding-noise-images-opencv>

Transformacje - Szum Losowy

przejawia się przyciemnieniami i rozjaśnieniami fragmentów obrazu



<https://www.askpython.com/python/examples/adding-noise-images-opencv>

Transformacje - Sztum Losowy

```
import cv2
import numpy as np

def add_random_noise(image, intensity=25):
    noisy_image = image.copy()
    noise = np.random.randint(-intensity, intensity + 1, noisy_image.shape)
    noisy_image = np.clip(noisy_image + noise, 0, 255).astype(np.uint8)
    return noisy_image

original_image = cv2.imread('testimg.jpeg')

if original_image is None:
    raise Exception("Image not loaded properly. Check the file path.")

noisy_image = add_random_noise(original_image, intensity=25)
```

<https://www.askpython.com/python/examples/adding-noise-images-opencv>

Transformacje - Blur (rozmycie)



Zdarzają się sytuacje, że chcemy pozbyć się szumów w obrazach, do czego rozmycia są bardzo użyteczne przy minimalnych stratach jakości obrazów.

Transformacje - rozmycie Gaussa

Rozmycie Gaussa to operacja konwolucji na obrazie z tzw. "jądrem Gaussowskim".

$$\frac{1}{16} \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$


Transformacje - rozmycie Gaussa

Syntax: cv2.GaussianBlur(obraz, (kernel_w, kernel_h), sigma)

```
# Gaussian Blur
Gaussian = cv2.GaussianBlur(image, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)
cv2.waitKey(0)
```

Transformacje - rozmycie medianą

- ➡ nieliniowa operacja arytmetyczna na każdym pikselu obrazu
- ➡ najczęściej używana w usuwaniu szumu typu "sól i pieprz"
- ➡ polega na przejściu NxN wymiarowego jądra przez obraz po każdym pikselu i zastępowania piksela w centrum jądra medianą ze wszystkich wartości w jądrze

Transformacje - rozmycie medianą

Syntax: cv2.medianBlur(obraz, rozmiar_jądra)

```
# Median Blur  
median = cv2.medianBlur(image, 5)  
cv2.imshow('Median Blurring', median)  
cv2.waitKey(0)
```



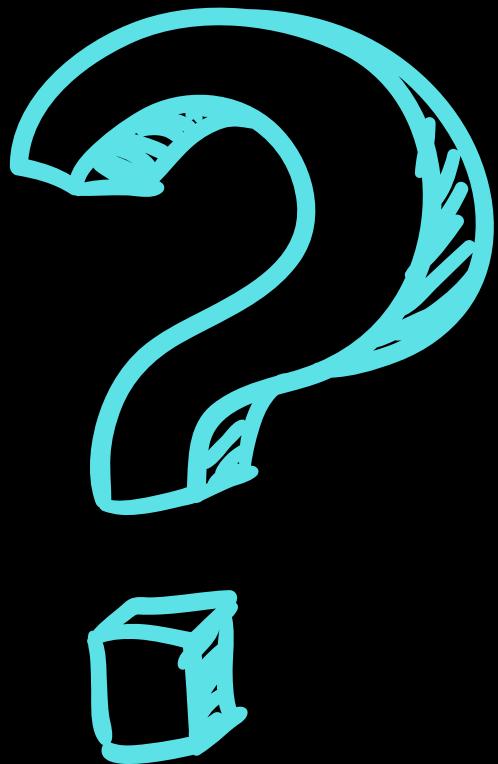
Słynny obrazek testowy



Słynny obrazek testowy



Duży zbiór do trenowania sieci



ImageNet

14 milionów
obrazów

różna rodzinieczkość

1000 klas



ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Modele pretrenowane

Torch posiada zestaw pretrenowanych modeli do różnych zastosowań w module `torchvision.models`. Są to gotowe do użycia lub dotrenowania sieci wyszkolone na zestawie ImageNet w przypadku klasyfikatorów, ale znajdziemy też sieci identyfikujące punkty kluczowe, warstwy segmentacji i generalnej detekcji obiektów.

Modele pretrenowane i ich różnice

Głównymi różnicami w modelach pretrenowanych jest ich generalna architektura, wydajność i precyzja przy zastosowaniu dołączonych do nich wag oraz charakterystyka zastosowań modelu, o których więcej można dowiedzieć się z dokumentacji.

Modele pretrenowane i ich zastosowanie

Inception

rozpoznawanie
obiektów o
różnych
rozmiarach

MobileNet

aplikacje mobilne
szybkie i lekkie modele
klasyfikacji obrazów

ResNet

głęboka
klasyfikacja

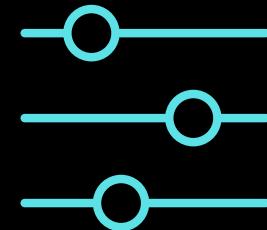
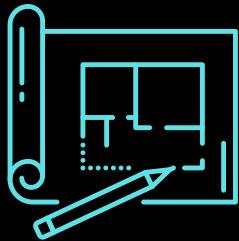
detekcja

segmentacja

VGG

prosta
klasyfikacja
punkt
wyjściowy

VGG: VGG16, VGG19



prosta i jednorodna

wolny w trenowaniu

138/144 mln parametrów

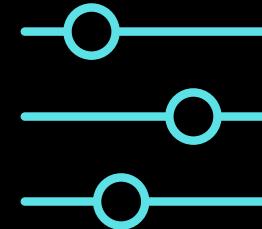
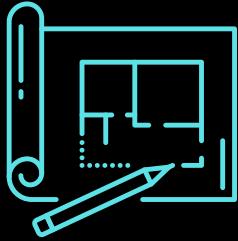
13/16 warstw konwolucyjnych

brak optymalizacji

3 warstwy w pełni połączone

filtr 3x3

ResNet: 18, 34, 50, 101, 152



zastosowanie *residual connections*

średni, ale szybszy w trenowaniu niż VGG

12-58 mln parametrów

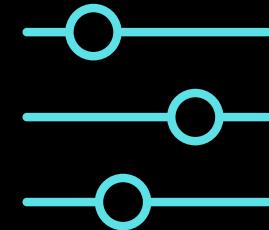
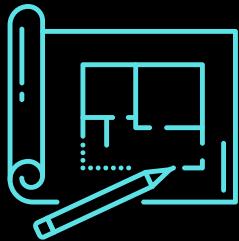
How Neural Network works?

Neurons:

I know a guy who knows a guy

residual connections - umożliwiają tworzenie bardzo głębokich architektur przy zmniejszonym problemie zanikania gradientu

MobileNet: MobileNetV1, MobileNetV2



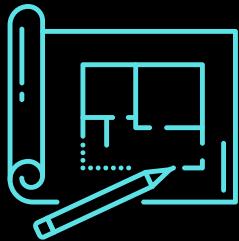
zastosowanie *depthwise
separable convolutions*

bardzo szybki w
trenowaniu

3-4 mln parametrów

depthwise separable convolutions – każda operacja konwolucyjna jest rozdzielona na dwie części: filtracja kanałów i filtrację przestrzenną

Inception: InceptionV1(GoogLeNet), V2, V3

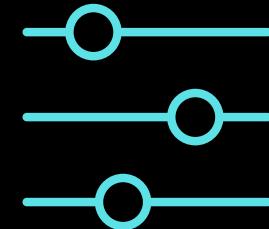


inception modules

różne rozmiary filtrów
(np. 1x1, 3x3, 5x5) w
jednej warstwie



może być wolny przez
równoległe filtry
konwolucyjne



7-23 mln parametrów



To który mam wybrać?



To który mam wybrać?

Czy zależy mi na czasie trenowania?

Czy mam dużo pamięci do wykorzystania?

Czy potrzebuję modelu dla urządzeń mobilnych?

Czy model ma być bardzo dokładny?

Jak duży jest zbiór danych?

Czy sieć ma być bardzo głęboka?

Czy zależy mi na prostocie architektury?

Czy chcę model odporny na zanikający gradient?

Czy potrzebuję wykrywać obiekty o różnych rozmiarach?

Czy zależy mi na szybkości działania modelu?

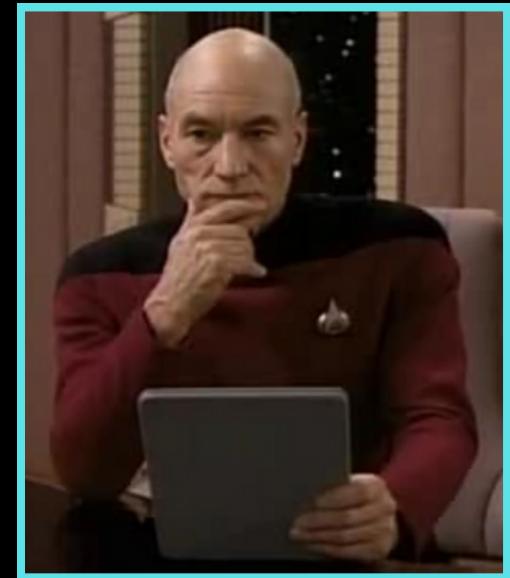
Czy potrzebuję modelu do pracy w czasie rzeczywistym?

Czy muszę zredukować liczbę parametrów?

Czy model musi działać na wielu urządzeniach?

Czy mam dużo danych, ale ograniczone zasoby obliczeniowe?

Czy chcę uzyskać najlepsze wyniki w klasyfikacji obrazów?



Ładowanie modeli

```
import torch
import torchvision.models as models

model = models.resnet18(pretrained=True) #PRZESTARZAŁE!

model.eval()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

print(model)

do fine-tuning: model.train()
```

Ładowanie modeli

```
from torchvision.io import decode_image
from torchvision.models import resnet50, ResNet50_Weights

img = decode_image("test/assets/encode_jpeg/grace_hopper_517x606.jpg")

# Step 1: Initialize model with the best available weights
weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = preprocess(img).unsqueeze(0)

# Step 4: Use the model and print the predicted category
prediction = model(batch).squeeze(0).softmax(0)
class_id = prediction.argmax().item()
score = prediction[class_id].item()
category_name = weights.meta["categories"][class_id]
print(f"{category_name}: {100 * score:.1f}%")
```

<https://pytorch.org/vision/stable/models.html>

