



TensorFlow

# Plan

1. Wstęp
2. Tensory
  1. `tf.Tensor`
  2. Operacje na tensorach
  3. Szczególne rodzaje tensorów
  4. `tf.Variable`
3. Dodatki



Wstep

# TensorFlow

TensorFlow to stworzona w 2011 roku przez firmę Google biblioteka dedykowana do generowania oraz wdrażania modeli uczenia maszynowego. W 2015 roku kod biblioteki został przeniesiony do domeny publicznej na licencji Apache2.0 i jest utrzymywany na githubie pod adresem <https://github.com/tensorflow/tensorflow>. Biblioteka napisana jest w języku C++, posiada pełny wrapper w języku Python oraz częściowe wrappery w językach: Java, JavaScript, Go i Swift. W roku 2017 została wypuszczona pierwsza stabilna wersja biblioteki - TensorFlow 1.0. W 2019 roku Google wypuściła kolejną wersję biblioteki - TensorFlow 2.0. W nowa wersji położono szczególny nacisk na łatwość używania biblioteki oraz wyposażenie biblioteki w wysokopoziomowe api.

Dokumentacja biblioteki znajduje się pod adresem <https://tensorflow.org>.

## Opanuj swoją ścieżkę

Aby zostać ekspertem w dziedzinie uczenia maszynowego, potrzebujesz najpierw solidnych podstaw w czterech obszarach uczenia się : kodowania, matematyki, teorii uczenia maszynowego i tworzenia własnego projektu uczenia maszynowego od początku do końca.

Rozpocznij od wyselekcjonowanych programów nauczania TensorFlow, aby poprawić te cztery umiejętności, lub wybierz własną ścieżkę nauki, przeglądając naszą bibliotekę zasobów poniżej.



# Wprowadzenie do TensorFlow

TensorFlow ułatwia początkującym i ekspertom tworzenie modeli uczenia maszynowego dla komputerów stacjonarnych, urządzeń mobilnych, sieci i chmury. Aby rozpocząć, zapoznaj się z poniższymi sekcjami.



## Przepływ Tensora

Poznaj podstawy TensorFlow dzięki samouczkom dla początkujących i ekspertów, które pomogą Ci stworzyć kolejny projekt uczenia maszynowego.



## Dla JavaScript

Użyj TensorFlow.js do tworzenia nowych modeli uczenia maszynowego i wdrażania istniejących modeli za pomocą JavaScript.



## Do urządzeń mobilnych i IoT

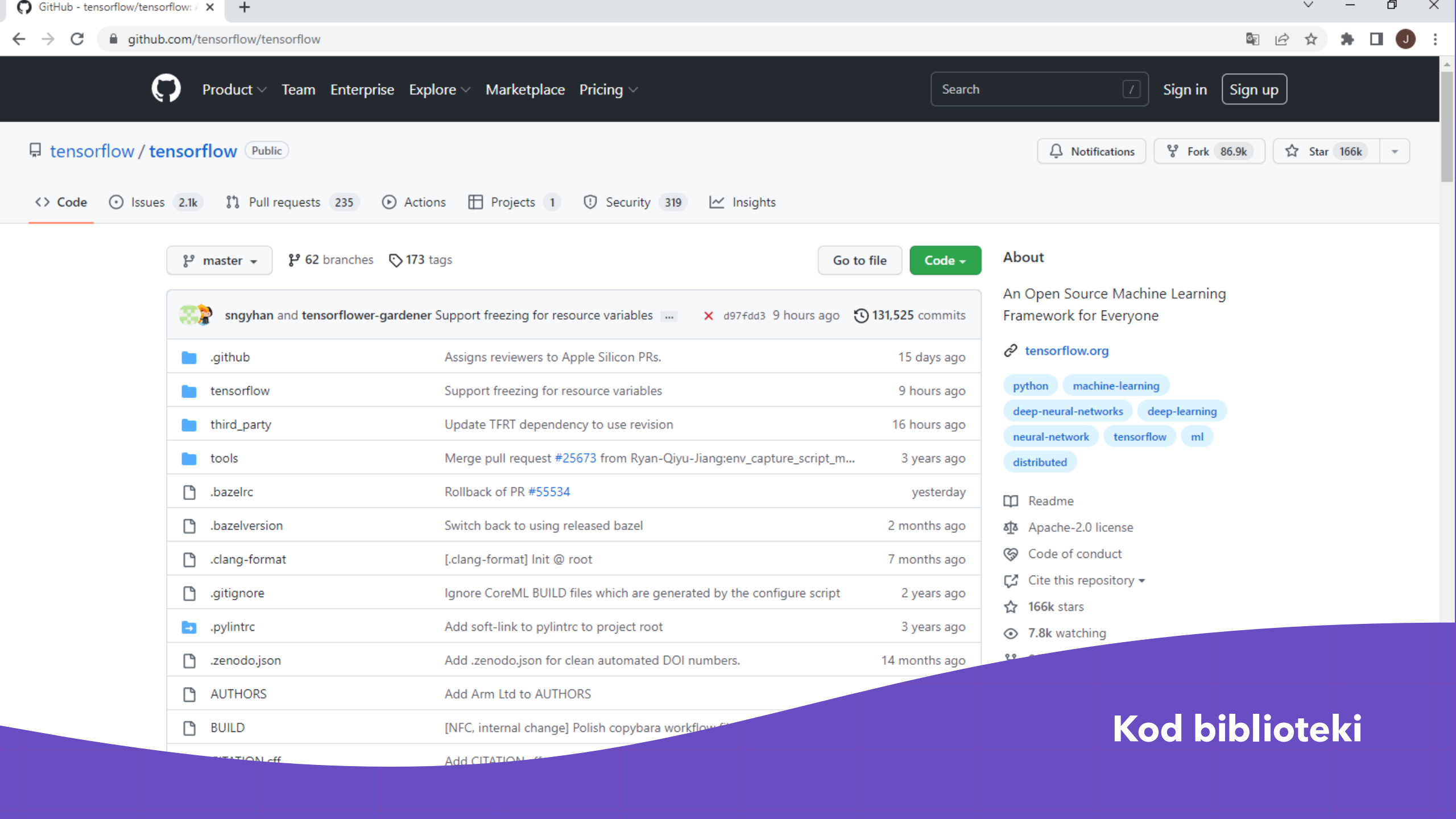
Uruchom wnioskowanie za pomocą TensorFlow Lite na urządzeniach mobilnych i wbudowanych.



## Do produkcji

Wdróż gotowy do produkcji potok ML do uczenia i wnioskowania przy użyciu TensorFlow Serving.

Materiały edukacyjne



About

An Open Source Machine Learning Framework for Everyone

tensorflow.org

- python machine-learning
- deep-neural-networks deep-learning
- neural-network tensorflow ml
- distributed

Readme

Apache-2.0 license

Code of conduct

Cite this repository

166k stars

7.8k watching

sngyhan and tensorflower-gardener Support freezing for resource variables d97fdd3 9 hours ago 131,525 commits		
.github	Assigns reviewers to Apple Silicon PRs.	15 days ago
tensorflow	Support freezing for resource variables	9 hours ago
third_party	Update TFRT dependency to use revision	16 hours ago
tools	Merge pull request #25673 from Ryan-Qiyu-Jiang:env_capture_script_m...	3 years ago
.bazelrc	Rollback of PR #55534	yesterday
.bazelversion	Switch back to using released bazel	2 months ago
.clang-format	[.clang-format] Init @ root	7 months ago
.gitignore	Ignore CoreML BUILD files which are generated by the configure script	2 years ago
.pylintrc	Add soft-link to pylintrc to project root	3 years ago
.zenodo.json	Add .zenodo.json for clean automated DOI numbers.	14 months ago
AUTHORS	Add Arm Ltd to AUTHORS	
BUILD	[NFC, internal change] Polish copybara workflow file	
CITATION.cff	Add CITATION.cff	

Kod biblioteki

# TensorFlow 1.0 vs TensorFlow 2.0

Początkowo biblioteka TensorFlow zniechęcała użytkowników skomplikowanym schematem użycia. Prowadziło to do zwiększenia popularności młodszych, wyżej poziomowych bibliotek takich jak PyTorch czy Keras. Żeby zapobiec utracie użytkowników twórcy TensorFlow zdecydowali się zintegrować TensorFlow z mniejszą biblioteką Keras. Jednak nie zmieniło to znacząco popularności biblioteki. Pisanie kodu w TensorFlow wciąż było skomplikowane. PyTorch dzięki prostocie użycia w 2019 roku zaczęła popularnością dorównywać bibliotece TensorFlow.

## Search Results





# TensorFlow 1.0 vs TensorFlow 2.0

W związku z rosnącą popularnością PyTorch twórcy TensorFlow zdecydowali się na wprowadzenie zmian, które uprościły sposób korzystania z biblioteki. Do najważniejszych zmian towarzyszących nowszej wersji należą:

1. zrezygnowanie z mechanizmu sesji,
2. odejście od ręcznego tworzenia grafów obliczeniowych na rzecz "eager execution",
3. uspoźnienie interfejsu programistycznego biblioteki,
4. wprowadzenie wysokopoziomego api na wzór biblioteki Keras.

# Eager Execution vs graph execution and session

W wersji 1.x biblioteki Tensor instrukcje nie były wykonywane kolejno, tak jak w większości bibliotek. Instrukcje były tylko deklaracjami, a ich wykonanie oddelegowywane było do sesji. Po zainicjalizowaniu sesji biblioteka tworzyła z przekazanych do sesji, zadeklarowanych obiektów graf obliczeniowy i wszystkie obliczenia wykonywała na tym grafie (*graph execution*). Taki sposób pisania kodu jest trudniejszy od *eager execution* (wykonywanie kodu linijka po linijce), ale w zamian kod może być znacząco szybszy. Wynika to z faktu, że tworzone w sesji grafy są zoptymalizowane pod względem obliczeniowym. W wersji 2.x wciąż można używać *graph execution*, ale domyślnym sposobem wykonywania kodu jest *eager execution*.

Więcej o *graph execution* znaleźć można w dodatkach do tej prezentacji.

# Session

VS

# Eager execution

```
▶ import tensorflow as tf

a = tf.constant(5)
b = tf.constant(4)
c = a + b

with tf.Session() as sess:
    print(sess.run(c))
```

9

```
▶ import tensorflow as tf

a = tf.constant(5)
b = tf.constant(4)
c = a + b

print(c.numpy())
```

9

# Zalety biblioteki TensorFlow

- zoptymalizowana pod kątem tworzenia i trenowania modeli uczenia maszynowego,
- jest open source
- posiada zbiór przydatnych narzędzi: **TensorBoard**, TensorFlow Playground, **TensorFlow Hub**,
- kompatybilna z popularnymi bibliotekami Pythona: NumPy, Matplotlib, pandas,
- posiada gotowe zbiory danych,
- posiada bogaty zbiór wytrenowanych modeli w TF Hub,
- posiada narzędzia do trenowania modeli na urządzeniach iOS, Android, Windows, MacOS, web
- posiada świetną dokumentację oraz wiele samouczków
- posiada ogromną, aktywną społeczność

## TensorBoard

Informacje ogólne Przewodnik

translated by Google Ta strona została przetłumaczona przez Cloud Translation API. [Switch to English](#)

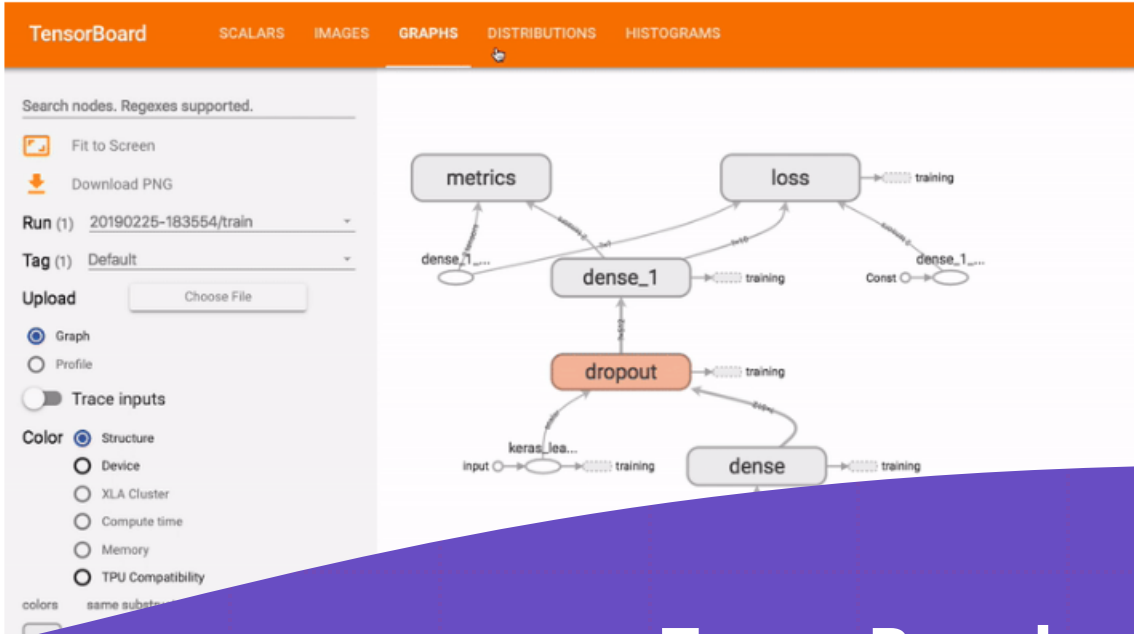
# TensorBoard: zestaw narzędzi do wizualizacji TensorFlow

TensorBoard zapewnia wizualizację i narzędzia potrzebne do eksperymentowania z uczeniem maszynowym:

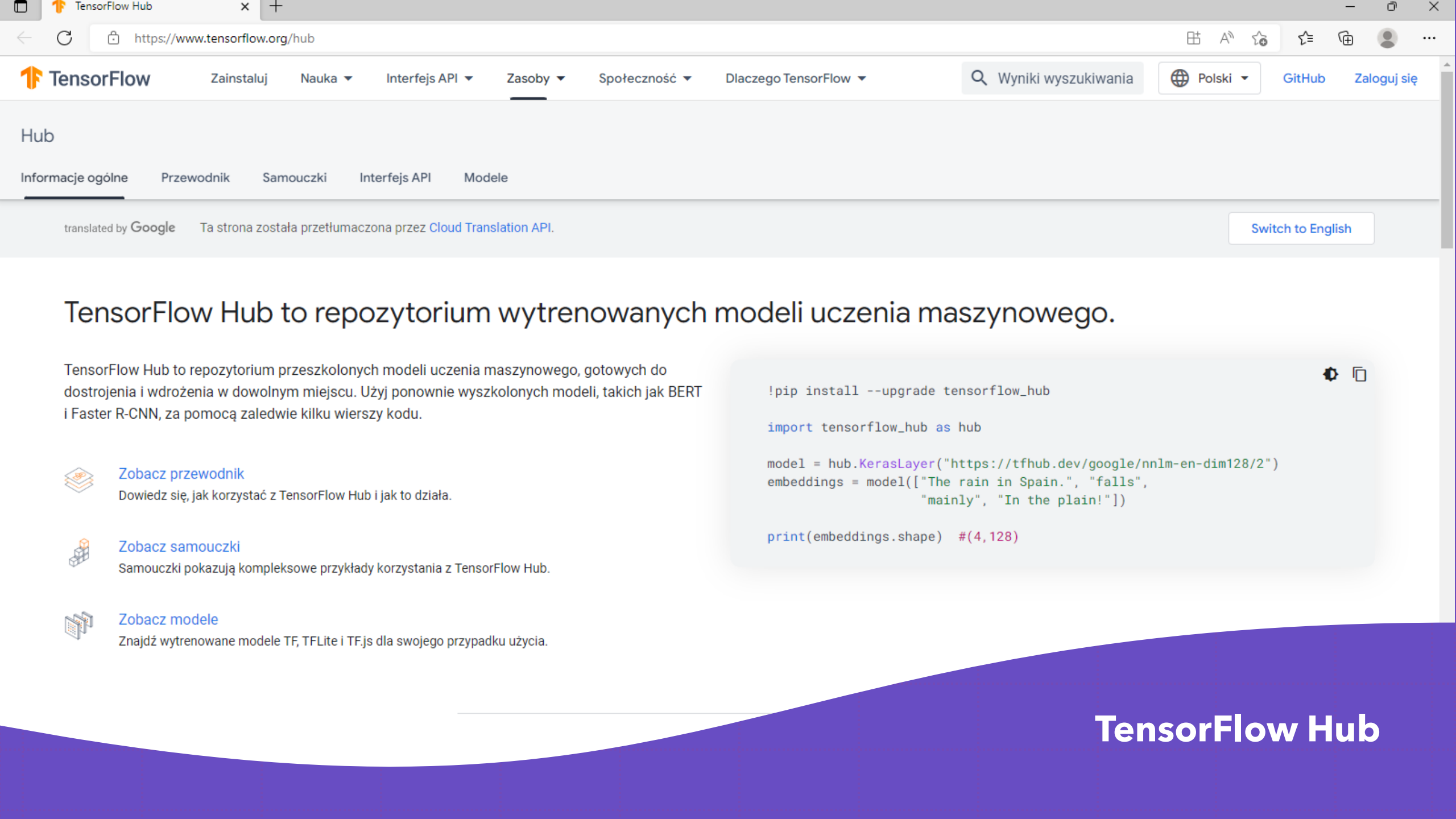
- Śledzenie i wizualizacja wskaźników, takich jak strata i dokładność
- Wizualizacja wykresu modelu (operacje i warstwy)
- Przeglądanie histogramów wag, błędów systematycznych lub innych tensorów zmieniających się w czasie
- Projekcja osadzeń do przestrzeni o niższych wymiarach
- Wyświetlanie obrazów, tekstu i danych dźwiękowych
- Profilowanie programów TensorFlow
- I wiele więcej

[TensorBoard.dev](#) umożliwia łatwe hostowanie, śledzenie i udostępnianie wyników eksperymentów.

[Rozpocznij](#) [Zacznij korzystać z TensorBoard.dev](#)



TensorBoard



## Hub

[Informacje ogólne](#)[Przewodnik](#)[Samouczki](#)[Interfejs API](#)[Modele](#)

translated by Google

Ta strona została przetłumaczona przez [Cloud Translation API](#).[Switch to English](#)

# TensorFlow Hub to repozytorium wytrenowanych modeli uczenia maszynowego.

TensorFlow Hub to repozytorium przeszkolonych modeli uczenia maszynowego, gotowych do dostrojenia i wdrożenia w dowolnym miejscu. Użyj ponownie wyszkolonych modeli, takich jak BERT i Faster R-CNN, za pomocą zaledwie kilku wierszy kodu.

[Zobacz przewodnik](#)

Dowiedz się, jak korzystać z TensorFlow Hub i jak to działa.

[Zobacz samouczki](#)

Samouczki pokazują kompleksowe przykłady korzystania z TensorFlow Hub.

[Zobacz modele](#)

Znajdź wytrenowane modele TF, TFLite i TF.js dla swojego przypadku użycia.

```
!pip install --upgrade tensorflow_hub
```

```
import tensorflow_hub as hub
```




```
model = hub.KerasLayer("https://tfhub.dev/google/nnlm-en-dim128/2")  
embeddings = model(["The rain in Spain.", "falls",  
                    "mainly", "In the plain!"])
```

```
print(embeddings.shape)  #(4,128)
```

# TensorFlow Hub

# Tinker With a **Neural Network** Right Here in Your Browser.

Don't Worry, You Can't Break It. We Promise.



Epoch  
000,000

Learning rate  
0.03

Activation  
Tanh

Regularization  
None

Regularization rate  
0

Problem type  
Classification

## DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

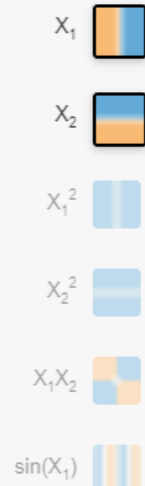
Noise: 0

Batch size: 10

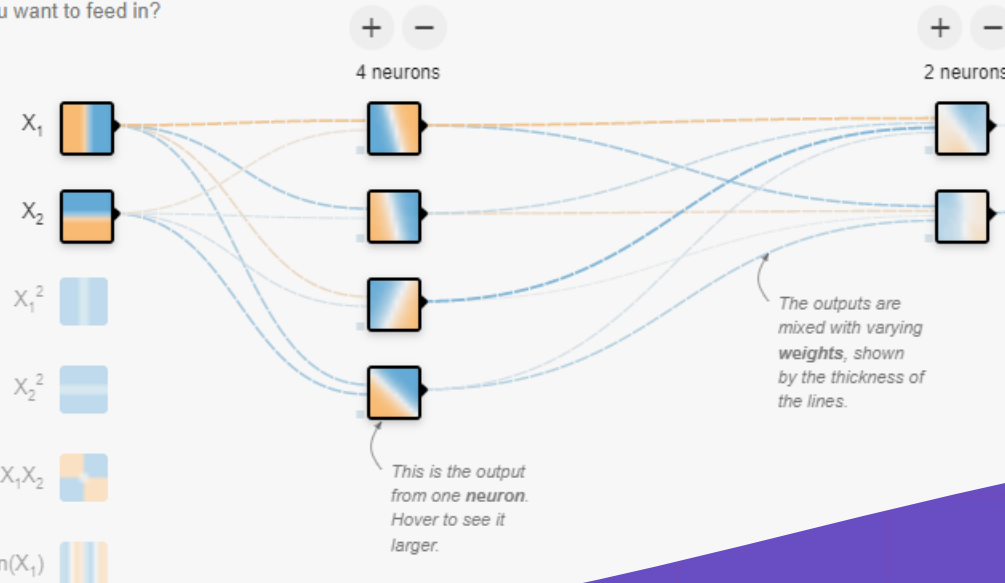
REGENERATE

## FEATURES

Which properties do you want to feed in?



## 2 HIDDEN LAYERS



## OUTPUT

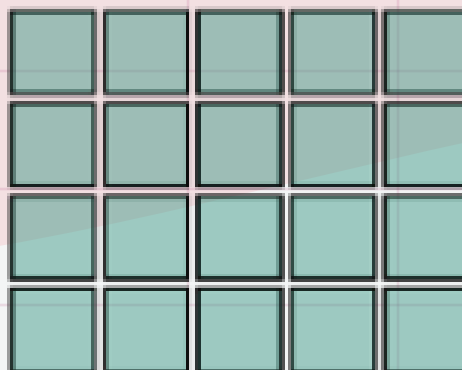
Test loss 0.517  
Training loss 0.498



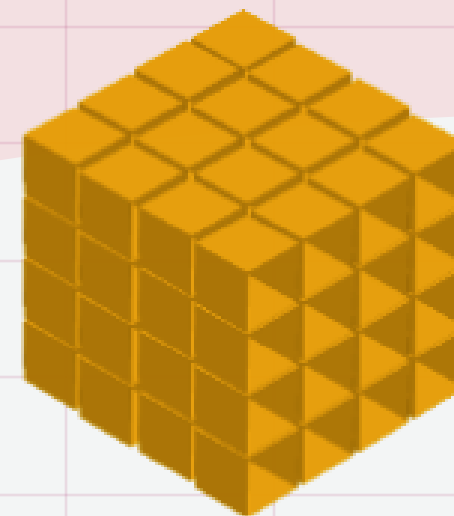
# TensorFlow Playground



RANK-1 TENSOR



RANK-2 TENSOR



RANK-3 TENSOR

# Tensory



# Tensor - definicja

W zależności od dziedziny słowo tensor może znaczyć coś innego. Spośród podstawowych definicji słowa tensor wyróżnić można definicję:

- **matematyczną**

Obiekt matematyczny, którego współrzędne zmieniają się w określony sposób przy transformacji układu współrzędnych (tzw. bazy). Na przykład: tensor metryczny.

- **fizyczną**

Obiekt fizyczny, który posiada określone niezmienniki przy afinicznej transformacji układu odniesienia. Na przykład: tensor naprężeń.

- **programistyczną**

Uogólnienie tablicy, na którym wybrane operacje (iloczyny) wykonywane są w specyficzny sposób. W bibliotece tensorflow tensor reprezentowany jest za pomocą klasy ***tf.Tensor*** (*tensor niemodyfikowalny*) oraz ***tf.Variable*** (*tensor modyfikowalny*).



# Tf.Tensor

<https://www.tensorflow.org/guide/tensor>

# tf.Tensor - tworzenie

Z perspektywy programisty tensory są bardzo podobne do tablic numpy. W bibliotece tensorflow tensory reprezentowane są za pomocą klas `tf.Tensor` oraz `tf.Variable`. W pierwszej kolejności popatrzymy na klasę **tf.Tensor**. W bibliotece tensorflow istnieje kilka funkcji do tworzenia obiektów tej klasy.

<b>constant()</b>	<b>ones_like()</b>	<b>zeros_like()</b>	<b>fill()</b>
<b>ones()</b>	<b>zeros()</b>	<b>range()</b>	<b>random.normal()</b>

Tworzenie obiektów klasy `tf.Tensor` poprzez ich bezpośrednią inicjalizację nie jest zalecane przez twórców tensorflow. Obiekty klasy `tf.Tensor` są **niemodyfikowalne** (niemutowalne), tzn. po utworzeniu nie można zmieniać ich wartości (jak napisy i krotki w pythonie).

# tf.Tensor – charakterystyki

Do podstawowych charakterystyk tensora należą:

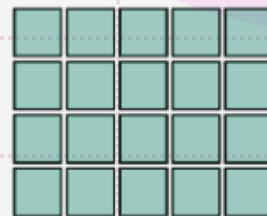
1. **rzęd** (*ang. rank*) - liczba wymiarów (osi) tensora (skalar jest tensorem rzędu 0, wektor jest tensorem rzędu 1, macierz jest tensorem rzędu 2)
2. **kształt** (*ang. shape*) - lista zawierająca długość (liczbę elementów) każdego z wymiarów (osi) tensora.
3. **rozmiar** (*ang. size*) - całkowita liczba elementów tensora
4. **typ** (*ang. type*) - wszystkie elementy tensora muszą być jednakowego typu.

Poszczególne charakterystyki zostaną szczegółowo omówione na kolejnych slajdach.

# tf.Tensor - rząd



RANK-1 TENSOR



RANK-2 TENSOR



RANK-3 TENSOR

Tensory są klasyfikowane na podstawie liczby posiadanych wymiarów:

- tensor rzędu 0 to tensor posiadający jedną wartość i żadnych osi (skalar)
- tensor rzędu 1 to tensor posiadający listę wartości wzdłuż jednej osi (wektor)
- tensor rzędu 2 to tensor posiadający dwie osie (macierz)
- tensor rzędu N to tensor posiadający N osi

Rząd tensora przechowywany jest w atrybucie ***ndim***.

# tf.Tensor - kształt

't'
'e'
'n'
's'
'o'
'r'

(6,)

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

(6, 4)

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

(4, 4, 2)

Zgodnie z konwencją przyjętą w tensorflow:

- dla tensora rzędu 2 pierwszy indeks atrybutu shape dotyczy liczby wierszy, a drugi liczby kolumn
- jeżeli tensor trzeciego rzędu wyobrazimy sobie jako pudełko, to pierwszy indeks atrybutu shape będzie dotyczył wysokości pudełka (liczby wierszy), drugi indeks długości pudełka (liczby kolumn), a trzeci indeks głębokości pudełka

Kształt tensora przechowywany jest w atrybucie **shape**.

# tf.Tensor – rozmiar

Rozmiar tensora to liczba jego elementów. W celu sprawdzenia rozmiaru tensora należy użyć funkcji ***tf.size()***.

# tf.Tensor - typ

Tensory poza standardowymi typami numerycznymi takim jak int czy float, mogą składać się również z elementów typu liczby zespolone czy napisy. Wszystkie elementy tensora muszą mieć ten sam typ.

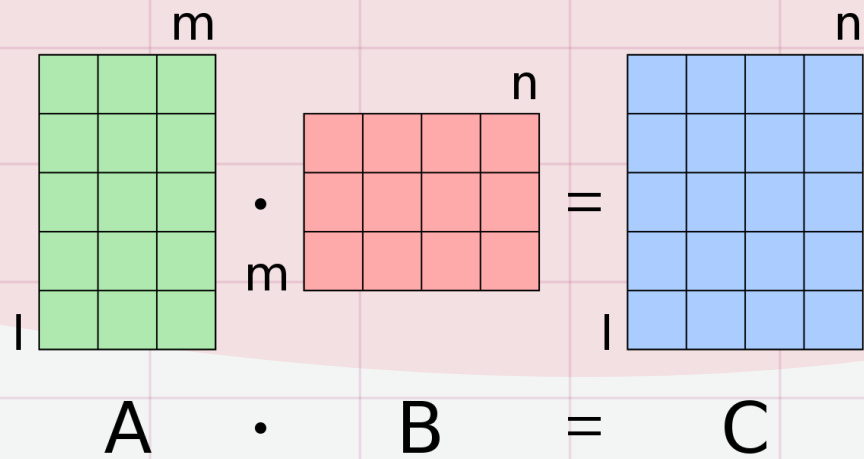
Typ elementów tensora przechowywany jest w atrybucie dtype.



# tf.Tensor - typ

Typ	Obiekt tensorflow
floating point	tf.float32, tf.float64
integer	tf.int8, tf.int16, tf.int32, tf.int64
unsigned integer	tf.uint8, tf.uint16
string	tf.string
boolean	tf.bool
complex number	tf.complex64, tf.complex128

Dostępne w tensorflow typy



| + | - | \* | / | @ | % |

Podstawowe operacje  
wykonywane na tensorach

# Podstawowe operacje na tensorach

Do podstawowych operacji można zaliczyć:

1. Indeksowanie
2. Operacje "po współrzędnych" (element-wise operations):
  - A. Dodawanie (funkcja *tf.add*, operator "+")
  - B. Odejmowanie (funkcja *tf.subtract*, operator "-")
  - C. Mnożenie po współrzędnych (funkcja *tf.multiply*, operator "\*")
  - D. Dzielenie (funkcja *tf.divide*, operator "/")
  - E. Modulo (funkcja *tf.math.floormod*, operator "%")
3. Mnożenie macierzowe (funkcja *tf.matmul*, operator "@")
4. Znajdowanie maksimum, minimum oraz sumy wszystkich elementów (funkcje *tf.reduce\_max*, *tf.reduce\_min*, *tf.reduce\_sum*)
5. Znajdowanie indeksu maksymalnego, minimalnego elementu (funkcje *tf.argmax*, *tf.argmin*)

# Indeksowanie

Do podstawowych reguł indeksowania należą:

- indeksowanie od 0
- obsługa ujemnych indeksów
- obsługa wycinków (operator ":")
- obsługa zagnieżdżonych indeksów (operator ",",")

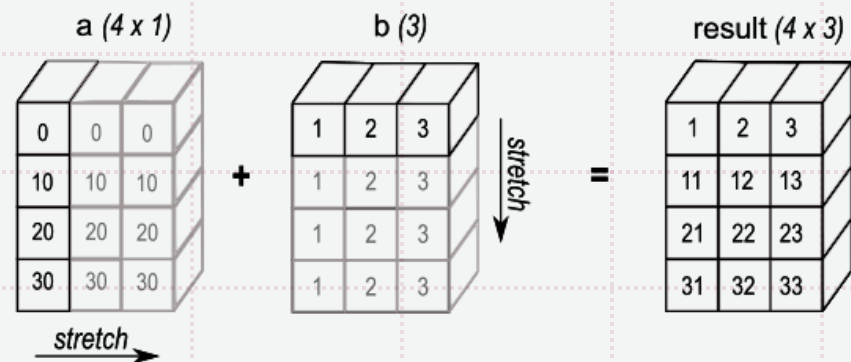
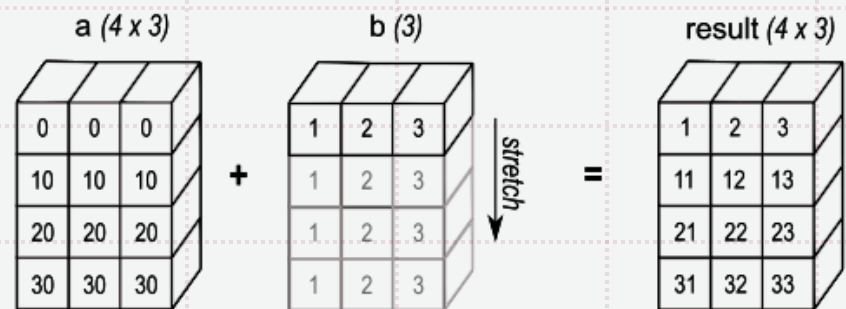
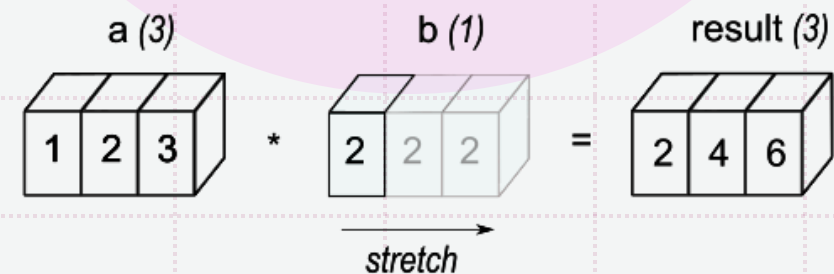
# Zaawansowane operacje

Do zaawansowanych operacji można zaliczyć m.in.:

1. Zmiana kształtu tensora (funkcja *tf.reshape*)
2. Broadcasting
3. Liczenie gradientu (funkcja *tf.gradient*)

# Broadcasting

Podobnie jak w bibliotece numpy, kiedy kształty tensorów nie są zgodne pod kątem wykonania danej operacji tensorflow na ile to jest możliwe powiela wartości w taki sposób, żeby dopasować do siebie wymiary obu tensorów.





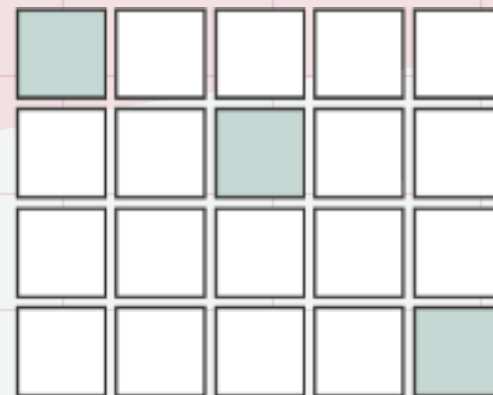
RAGGED TENSOR

"This Tensor"

"is an example of"

"String Tensor"

STRING TENSOR



SPARSE TENSOR

# Szczególne rodzaje tensorów

# Szczególne tensory

Do szczególnych rodzajów tensorów można zaliczyć:

1. nierówny tensor (ragged tensor) - tensor z różną liczbą elementów wzdłuż jednej osi
2. tensor napisów - tensor składający się z elementami typu string
3. rzadki tensor (sparse tensor) - tensor z wieloma zerowymi elementami





**tf.Variable**

# tf.Variable

Klasa `tf.Variable` podobnie jak klasa `tf.Tensor` reprezentuje tensor w bibliotece tensorflow. Obiekty klasy `tf.Variable` w odróżnieniu od obiektów klasy `tf.Tensor` są **modyfikowalne** (analogicznie jak listy vs krotki w pythonie). Ma to znaczenie w trakcie trenowania modelu, kiedy wagi modelu są nieustannie modyfikowane. Wagi przechowujemy w obiektach klasy `tf.Variable`.

Klasa `tf.Variable` jest podklasą klasy `tf.Tensor` w związku z tym dziedziczy wszystkie atrybuty i umiejętności klasy `tf.Tensor`.

Obiekty klasy `tf.Variable` modyfikujemy za pomocą funkcji `assign()`.

# tf.Variable - tworzenie

Obiekt klasy `tf.Variable` tworzymy standardowo, za pomocą inicjalizatora klasy. Parametrem inicjalizatora może być `integer`, `float`, `string`, lista, a nawet obiekt klasy `tf.Tensor`.

# tf.Variable – charakterystyki i operacje

Wszystkie poznane atrybuty i metody klasy `tf.Tensor` oraz funkcje działające na obiektach tej klasy zachowują się w identyczny sposób na obiektach klasy `tf.Variable`.

Klasa `tf.Variable` posiada dodatkowo metodę ***assign()***. Metody *assign()* używamy do przypisania nowej wartości obiektowi klasy `tf.Variable`. Klasa `tf.Variable` nie obsługuje operatora przypisania.



# Moduły TensorFlow

- Na tej stronie
- Modules**
- Classes
- Functions
- Other Members

## Moduły TensorFlow API

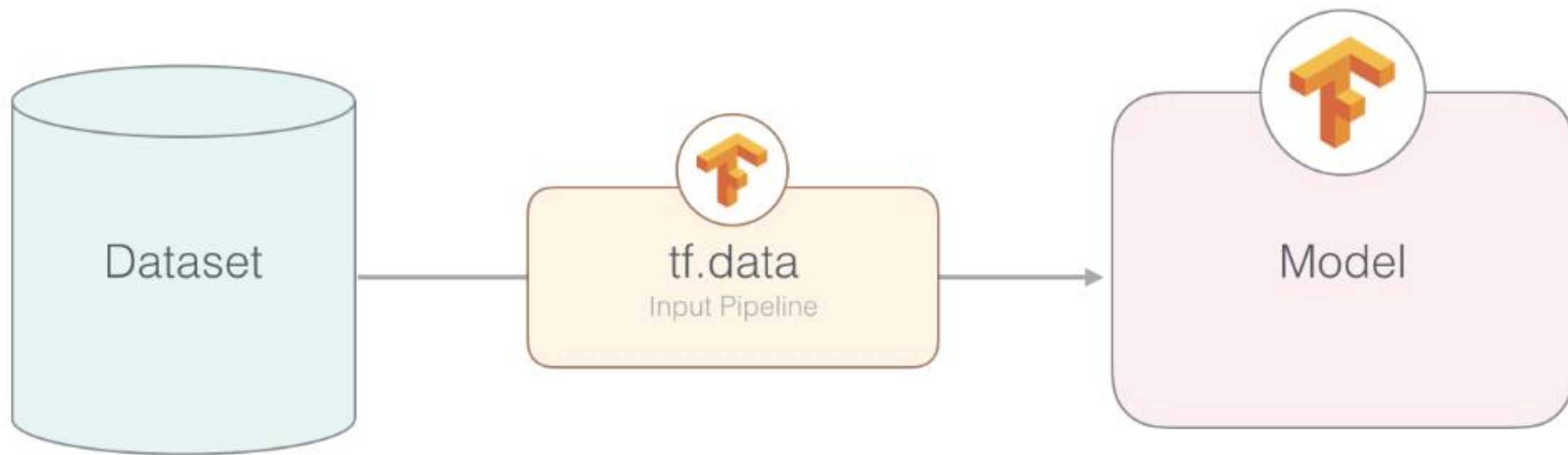


# TensorFlow

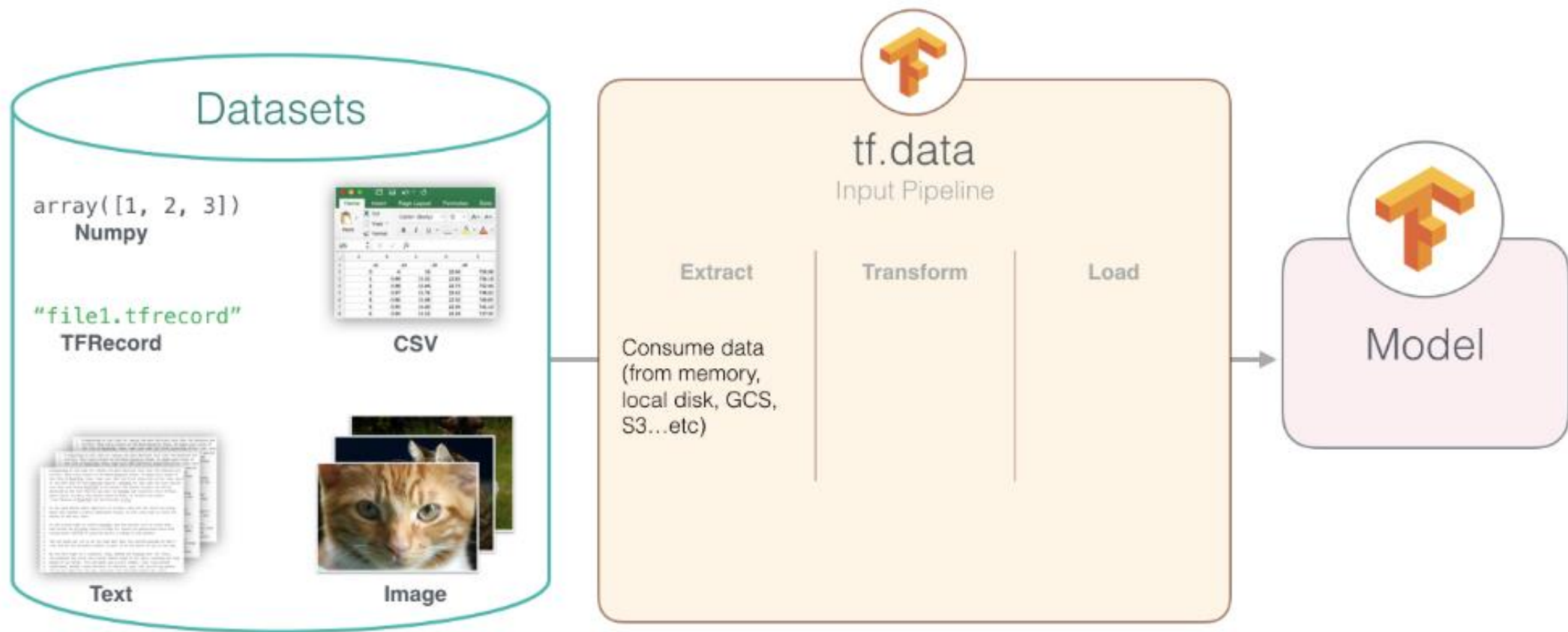
2.0

# tf.data

tf.data





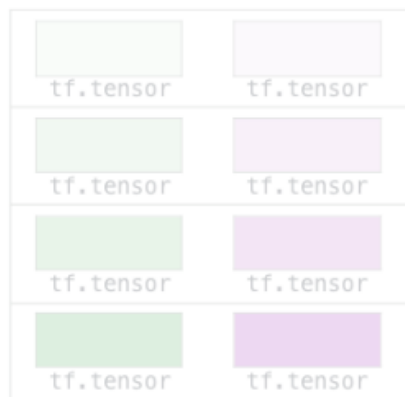




## tf.data Input Pipeline

### Extract

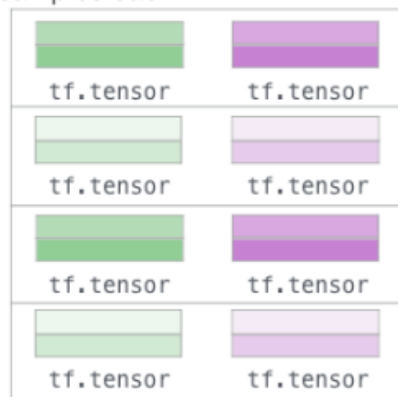
- 1) Consume data (from "dataset.csv")



tf.data.Dataset

### Transform

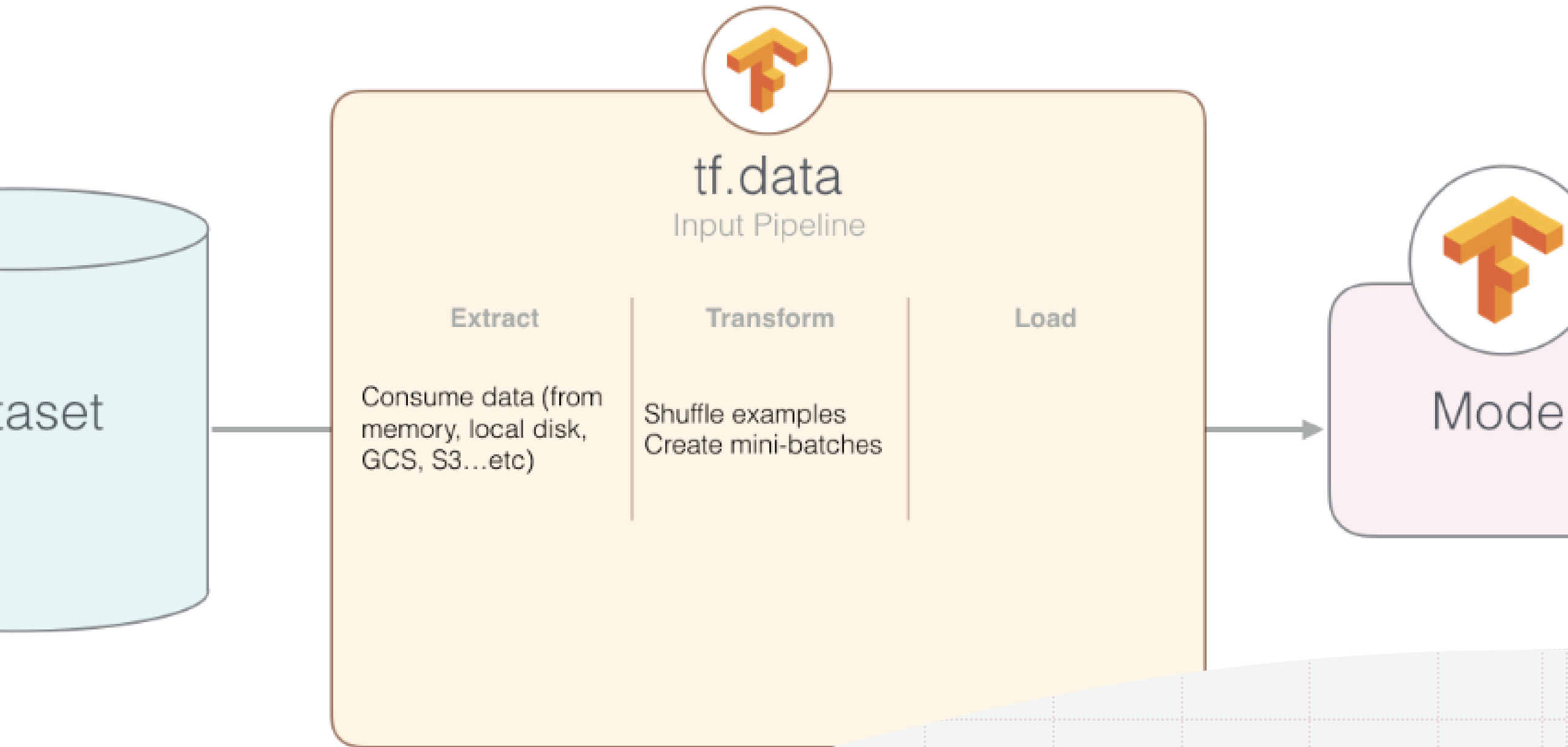
- 2) Shuffle examples
- 3) Repeat for 2 epochs
- 4) Create mini-batches of 2 samples each



tf.data.Dataset

### Load





**ETL - Extract Transform Load**

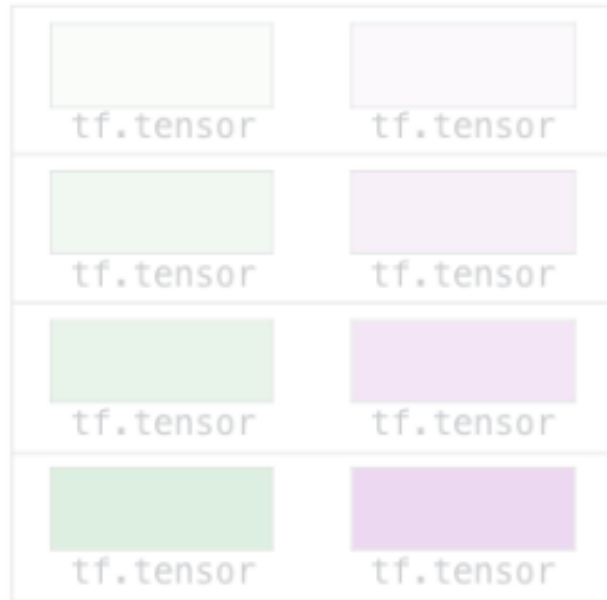


# tf.data

Input Pipeline

## Extract

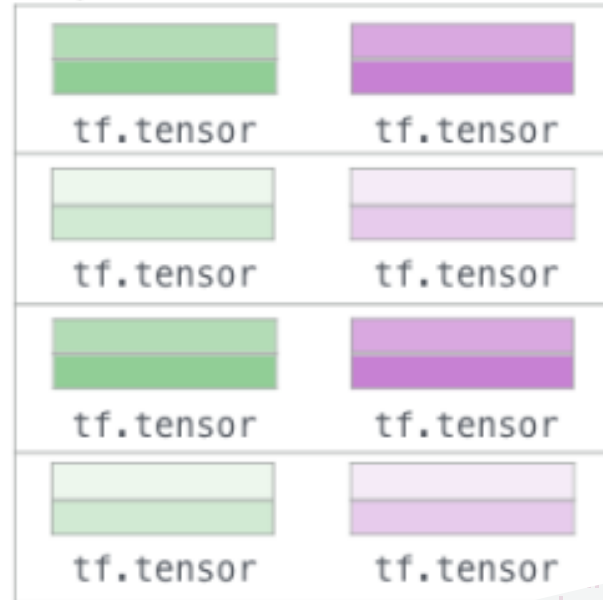
- 1) Consume data (from "dataset.csv")



tf.data.Dataset

## Transform

- 2) Shuffle examples
- 3) Repeat for 2 epochs
- 4) Create mini-batches of 2 samples each



tf.data.Dataset

## Load

**ETL - Extract Transform Load**

E

```
files = tf.data.Dataset.list_files(file_pattern)
dataset = tf.data.TFRecordDataset(files)
```

T

```
dataset = dataset.shuffle(10000)
dataset = dataset.repeat(NUM_EPOCHS)
dataset = dataset.map(lambda x: tf.parse_single_example(x, features))
dataset = dataset.batch(BATCH_SIZE)
```

L

```
iterator = dataset.make_one_shot_iterator()
features = iterator.get_next()
```





Simple. Flexible. Powerful.

Get started

API docs

Guides

Examples

```
from tensorflow import keras
from tensorflow.keras import layers

# Instantiate a trained vision model
vision_model = keras.applications.ResNet50()

# This is our video encoding branch using the trained vision_model
video_input = keras.Input(shape=(100, None, None, 3))
encoded_frame_sequence = layers.TimeDistributed(vision_model)(video_input)
encoded_video = layers.LSTM(256)(encoded_frame_sequence)

# This is our text-processing branch for the question input
question_input = keras.Input(shape=(100,), dtype='int32')
embedded_question = layers.Embedding(10000, 256)(question_input)
encoded_question = layers.LSTM(256)(embedded_question)

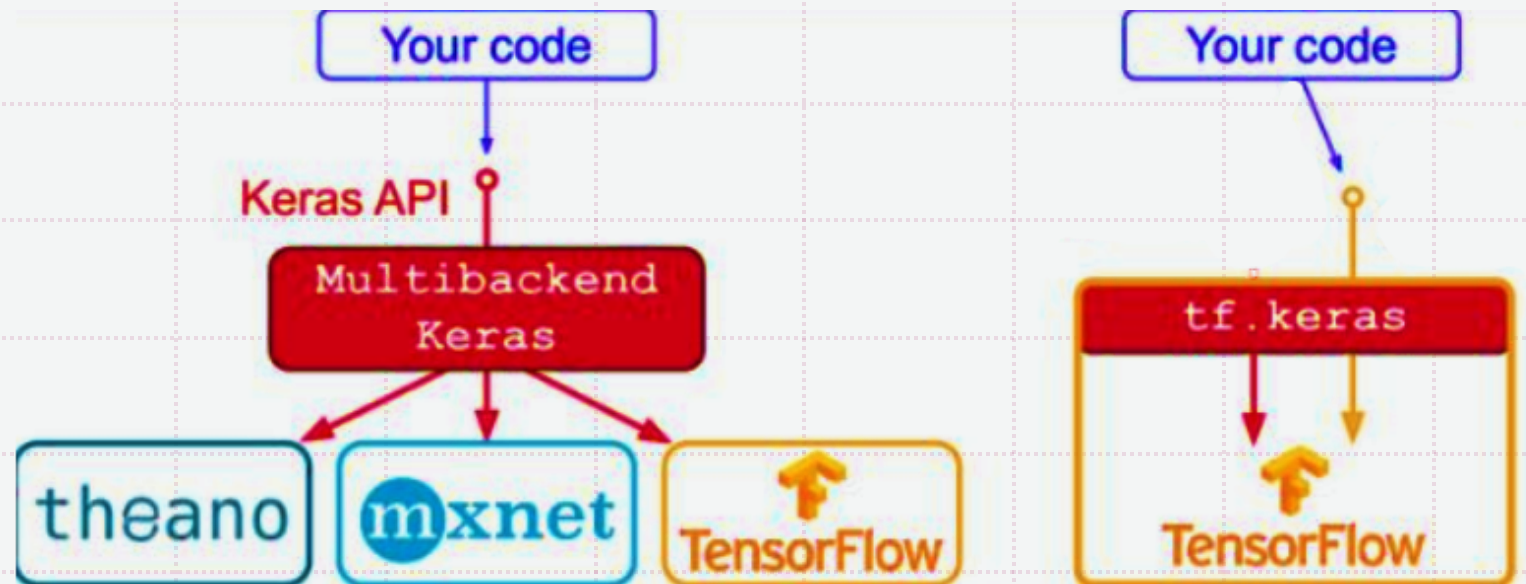
# Question answering model:
def question_answering_model(encoded_video, encoded_question):
```

Deep learning for humans.

Keras is an API designed to  
follow the

tf.keras

# Keras kiedyś i dziś



# Tworzenie modeli w tf.keras

W tf.keras istnieją dwie podstawowe metody tworzenia modeli sztucznych sieci neuronowych:

- Model sekwencyjny
- Funkcjonalne API



# Tworzenie modeli w tf.keras

## Model sekwencyjny

```
▶ # 3 warstwy
layer1 = layers.Dense(2, activation="relu", name="layer1")
layer2 = layers.Dense(3, activation="relu", name="layer2")
layer3 = layers.Dense(4, name="layer3")

# Wywołanie poszczególnych warstw na danych wejściowych x
x = tf.ones((3, 3))
y = layer3(layer2(layer1(x)))
```

# Tworzenie modeli w tf.keras

## Funkcjonalne API

```
▶ # 3-wartwowy model sekwencyjny
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Wywołanie modelu na danych wejściowych x
x = tf.ones((3, 3))
y = model(x)
```



# Dodatki



# Zbiory danych w TensorFlow

# Zbiory danych w tensorflow

## tensorflow-datasets

```
In [ ]: !pip install tensorflow-datasets
```

```
In [9]: import tensorflow_datasets as tfds
```

## Datasets

Overview Catalog Community Catalog Guide API

Filter

### Overview

- Dataset Collections
- 3d
- Abstractive text summarization
- Anomaly detection
- Audio
- Biology
- Categorical
- Common sense reasoning
- Conditional image generation
- Coreference resolution
- D4rl
- Density estimation
- Dependency parsing
- Document summarization
- Fine grained image classification
- Graph
- Graphs
- Image
- Image classification
- Image clustering
- Image compression
- Image generation
- Image super resolution
- Image to image translation
- Image classification
- Instance segmentation

Kategorie

TensorFlow > Resources > Datasets > Catalog

Was this helpful?

## Datasets

★ **Note:** The datasets documented here are from **HEAD** and so not all are available in the current **tensorflow-datasets** package. They are all accessible in our nightly package **tfds-nightly**.

### Usage

See our [getting-started guide](#) for a quick introduction.

```
for ex in tfds.load('cifar10', split='train'):  
    ...
```

### All Datasets

#### Dataset Collections

- longt5
- xtreme

3d

### On this page

- [Usage](#)
- All Datasets
- Dataset Collections
- 3d
- Abstractive text summarization
- Anomaly detection
- Audio
- Biology
- Categorical
- Common sense reasoning
- Conditional image generation
- Coreference resolution
- D4rl
- Density estimation
- Dependency parsing
- Document summarization
- Fine grained image classification
- Graph
- Graphs
- Image
- Image classification
- Image clustering
- Image compression
- Image generation
- Image super resolution
- Image to image translation
- Image classification
- Instance segmentation
- Language modelling
- Machine translation

# Zbiory danych w tensorflow

## tensorflow.keras.datasets

```
In [1]: import tensorflow as tf  
print(f"TensorFlow version: {tf.__version__}")
```

TensorFlow version: 2.12.0-rc1

```
In [2]: for dataset in dir(tf.keras.datasets):  
        if not dataset.startswith("_"):  
            print(dataset)
```

boston\_housing  
cifar10  
cifar100  
fashion\_mnist  
imdb  
mnist  
reuters

[About Keras](#)[Getting started](#)[Developer guides](#)[Keras API reference](#)[Models API](#)[Layers API](#)[Callbacks API](#)[Optimizers](#)[Metrics](#)[Losses](#)[Data loading](#)[Built-in small datasets](#)[MNIST digits classification dataset](#)[CIFAR10 small images classification dataset](#)[CIFAR100 small images classification dataset](#)[IMDB movie review sentiment classification dataset](#)[Reuters newswire classification dataset](#)[Fashion MNIST dataset, an alternative to MNIST](#)[Boston Housing price regression dataset](#)

# Datasets

The `tf.keras.datasets` module provide a few toy datasets (already-vectorized, in Numpy format) that can be used for debugging a model or creating simple code examples.

If you are looking for larger & more useful ready-to-use datasets, take a look at [TensorFlow Datasets](#).

## Available datasets

### MNIST digits classification dataset

- `load_data` function

### CIFAR10 small images classification dataset

- `load_data` function

### CIFAR100 small images classification dataset

- `load_data` function

### IMDB movie review sentiment classification dataset

- `load_data` function
- `get_word_index` function

### Reuters newswire classification dataset

- `load_data` function
- `get_word_index` function

### Fashion MNIST dataset, an alternative to MNIST

- `load_data` function

### Boston Housing price regression dataset

- `load_data` function

## Datasets

### ◆ Available datasets

[MNIST digits classification dataset](#)[CIFAR10 small images classification dataset](#)[CIFAR100 small images classification dataset](#)[IMDB movie review sentiment classification dataset](#)[Reuters newswire classification dataset](#)[Fashion MNIST dataset, an alternative to MNIST](#)[Boston Housing price regression dataset](#)





VS



# Scikit-learn vs TensorFlow

## Podobieństwa

Otwarte biblioteki pythonowe udostępniające zestaw narzędzi do uczenia maszynowego.



# Scikit-learn vs TensorFlow

## Różnice

**Scikit-learn** biblioteka do uczenia maszynowego ogólnego zastosowania.

**TensorFlow** biblioteka do uczenia maszynowego wyposażona w wyspecjalizowane narzędzia do tworzeniu sztucznych sieci neuronowych oraz algorytmów uczenia głębokiego.

**EAGER  
EXECUTION**

**EX**

**GRAPH  
EXECUTION**

# Grafy obliczeniowe

TensorFlow daje użytkownikowi możliwość wyboru jednego z dwóch trybów wykonywania kodu:

- eager execution (linijka po linijce)
- graph execution (kod wykonywany na podstawie tzw. grafu obliczeniowego)



# Grafy obliczeniowe

W starszej wersji biblioteki dostępny był tylko ten tryb grafowy – graph execution (wzorzec lazy evaluation), co miało duży wpływ na spadek popularności biblioteki. Pisanie kodu w starszych wersjach biblioteki polegało na ręcznym tworzeniu tzw. **grafu obliczeniowego**, w którym wierzchołkami były tensory, a krawędziami operacje wykonywane na tych tensorach. Wykonanie kodu polegało na uruchomieniu stworzonego grafu w ramach tzw. **sesji**.

W wersji TF2.0 twórcy ustawili tryb eager execution jako domyślny i obecnie można pisać w tensorflow programy nie zdając sobie nawet sprawy, że taki graf obliczeniowy kiedyś był niezbędny. Eager execution wykonuje kod natychmiast, instrukcja po instrukcji, co jest bardzo wygodne, ale w przypadku bardziej złożonych modeli może być wolniejsze (graf obliczeniowy pozwala na optymalizację obliczeniową algorytmu poprzez realizację kosztownych operacji tensorowych za pomocą szybszych operacji na grafach za pomocą tzw. **Grapplera**).

# Grafy obliczeniowe – dekorator `tf.function`

Ze względu na możliwości optymalizacyjne jakie dają grafy obliczeniowe twórcy tensorflow nie wyrzucili grafów w nowszych wersjach. Eager execution pomija grafy obliczeniowe wykonując kod linijka po linijce. Wciąż można jednak w nowszych wersjach wykonać napisany kod na grafie. Nie jest to już jednak tak żmudne jak było w starszych wersjach, w których cały graf trzeba było stworzyć ręcznie. W tensorflow 2.x do budowania grafu służy dekorator `tf.function`. Udekorowanie funkcji dekoratorem `tf.function` powoduje, że tensorflow traktuje ją jako pojedynczy obiekt grafu.

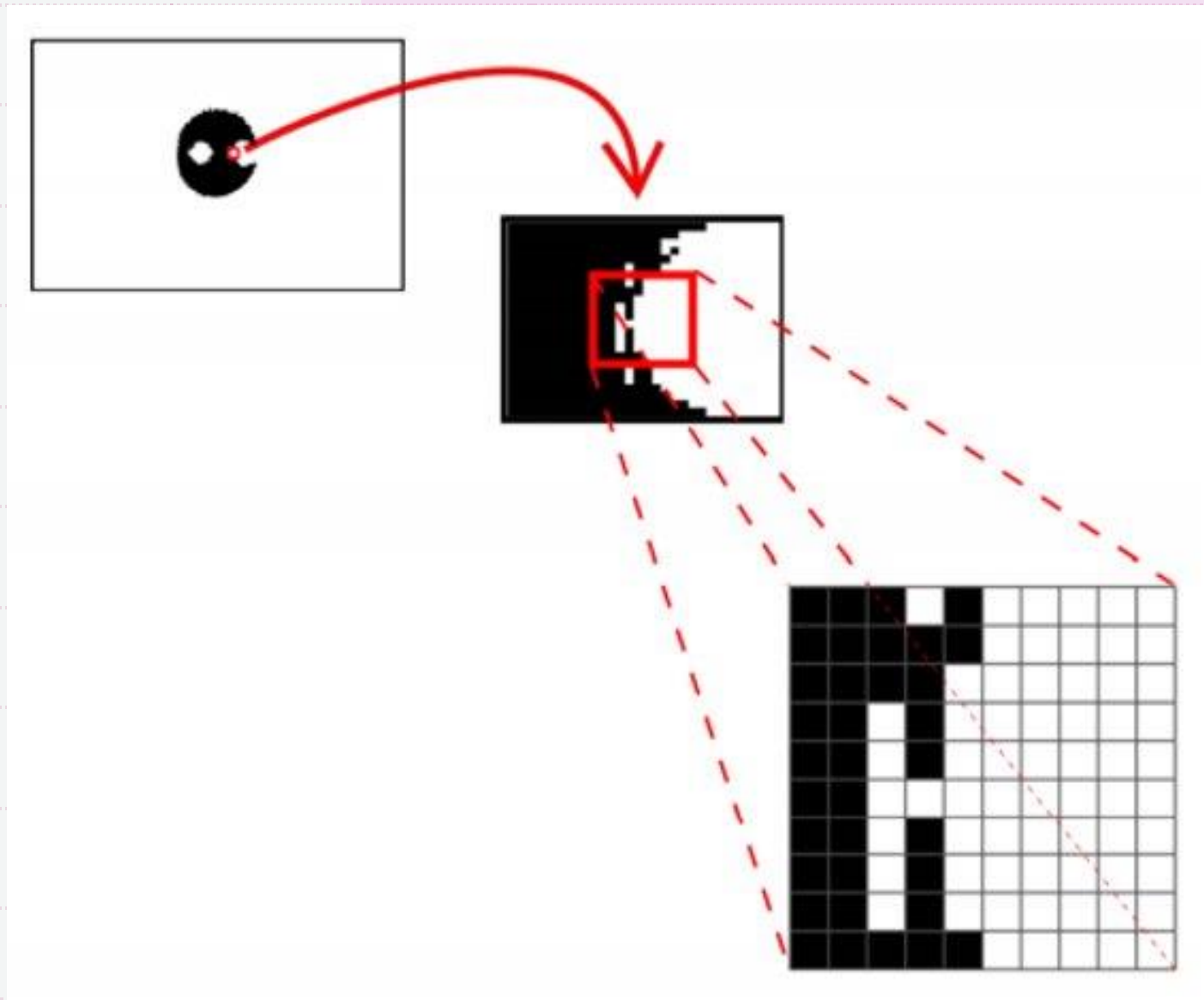
Przykładowy kod można znaleźć w notatniku w sekcji Dodatki.



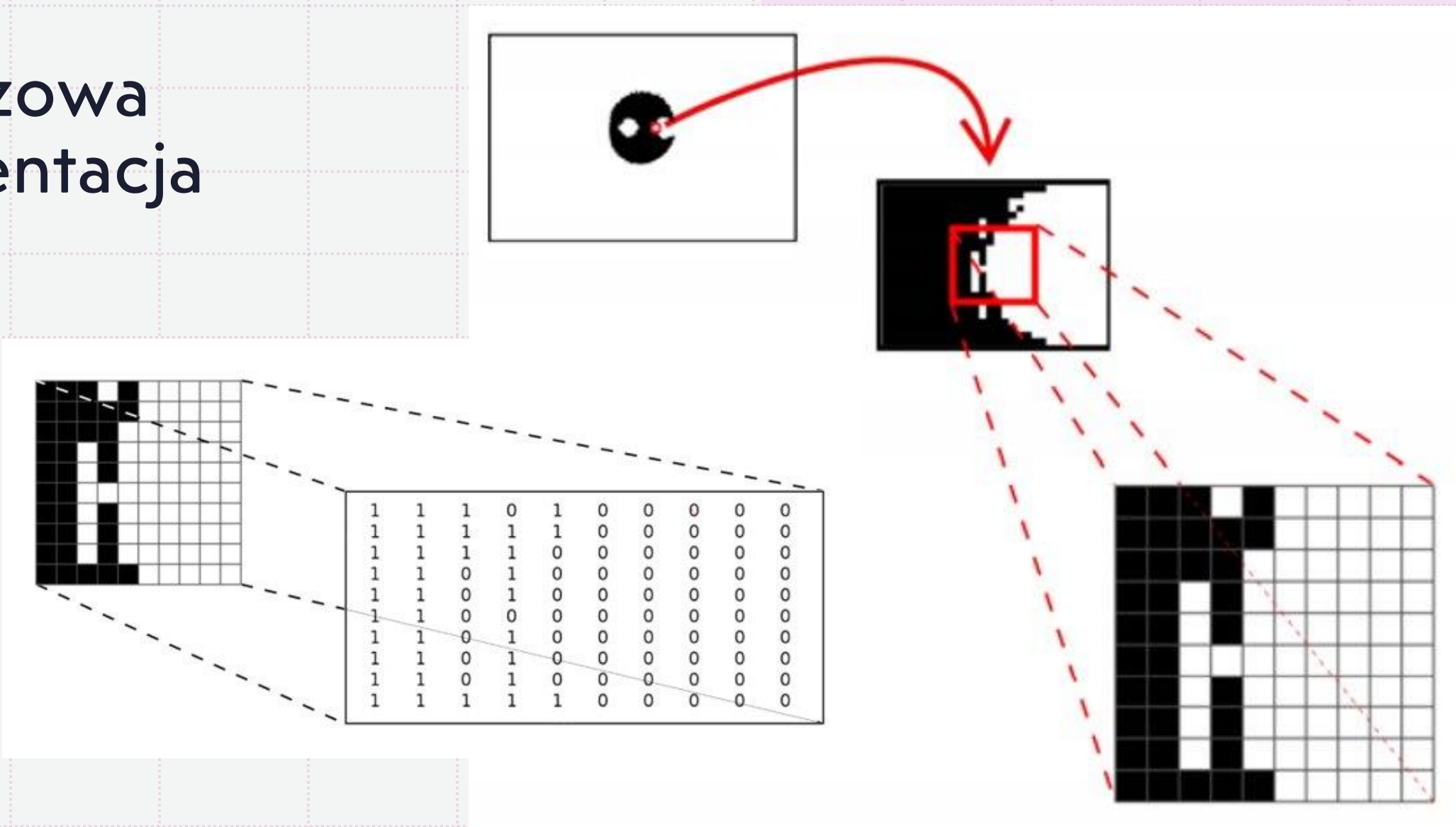
# Obrazy w uczeniu maszynowym



# Struktura czarno-białego (binarnego) obrazu

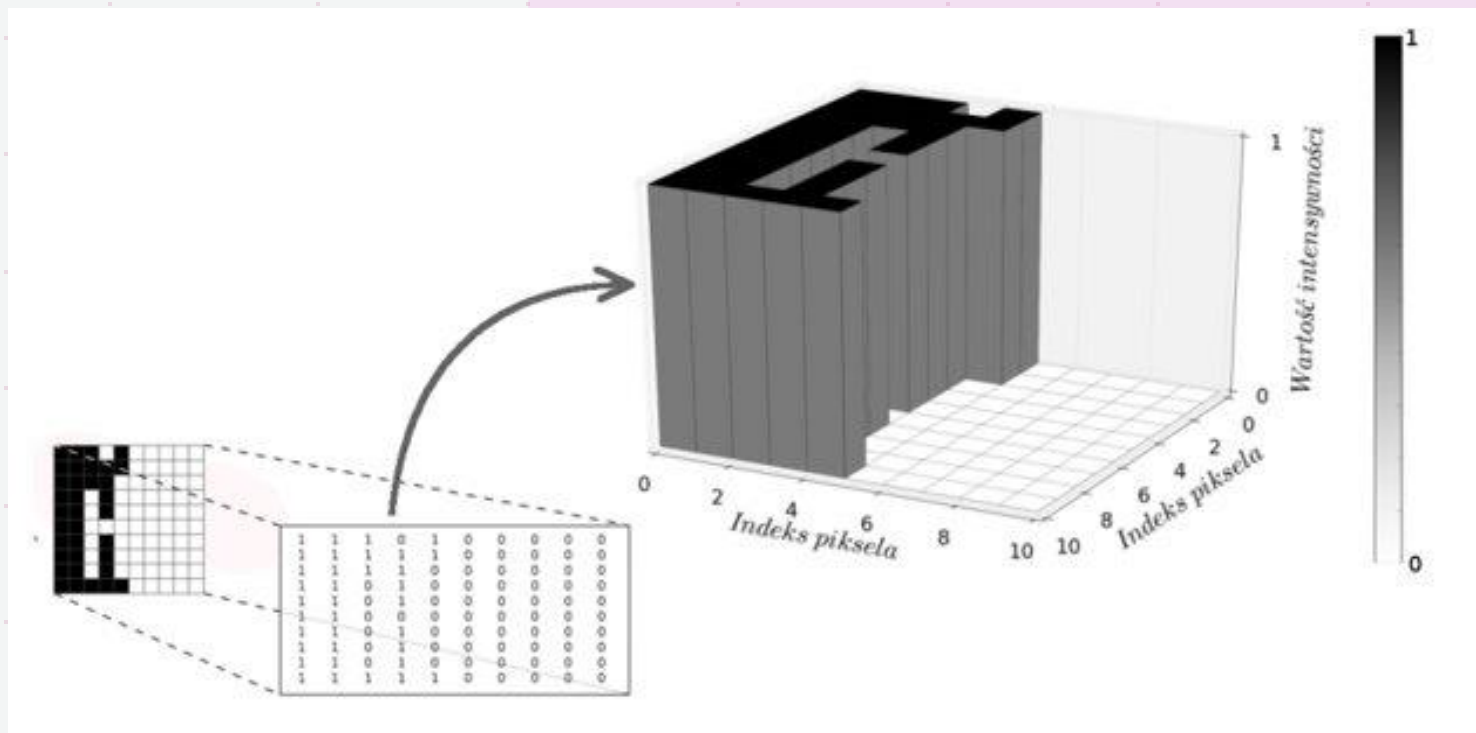


# Macierzowa reprezentacja obrazu

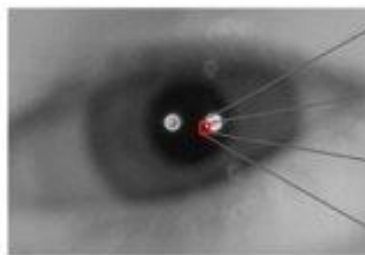


# Funkcyjna reprezentacja obrazu

W ujęciu algebraicznym macierz może reprezentować odwzorowanie (funkcję). Tym samym o obrazie możemy myśleć jako o funkcji (odwzorowaniu).



# Uogólnienie obrazu na skalę szarości (reprezentacja macierzowa)



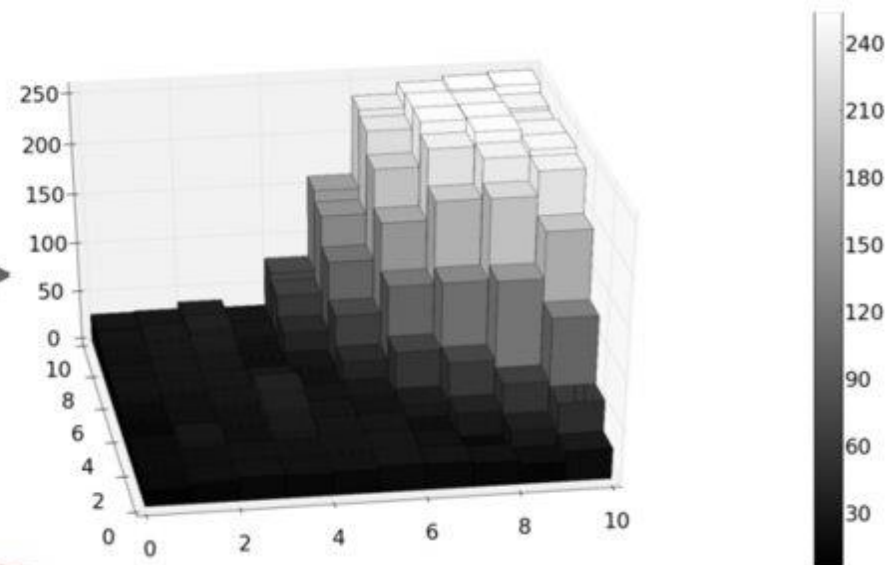
28	29	36	28	77	159	237	248	252	254
26	31	38	28	69	148	230	251	250	245
23	29	34	22	47	117	207	251	252	242
24	28	29	20	27	78	169	239	253	247
25	27	29	30	23	47	121	203	242	248
17	23	27	39	26	29	71	138	218	242
13	22	23	36	27	24	34	74	153	199
18	28	21	31	27	28	21	40	67	130
18	21	24	24	26	28	24	16	37	62
16	21	24	23	24	27	26	24	20	31

obraz (2, 1)  $\Rightarrow$  31

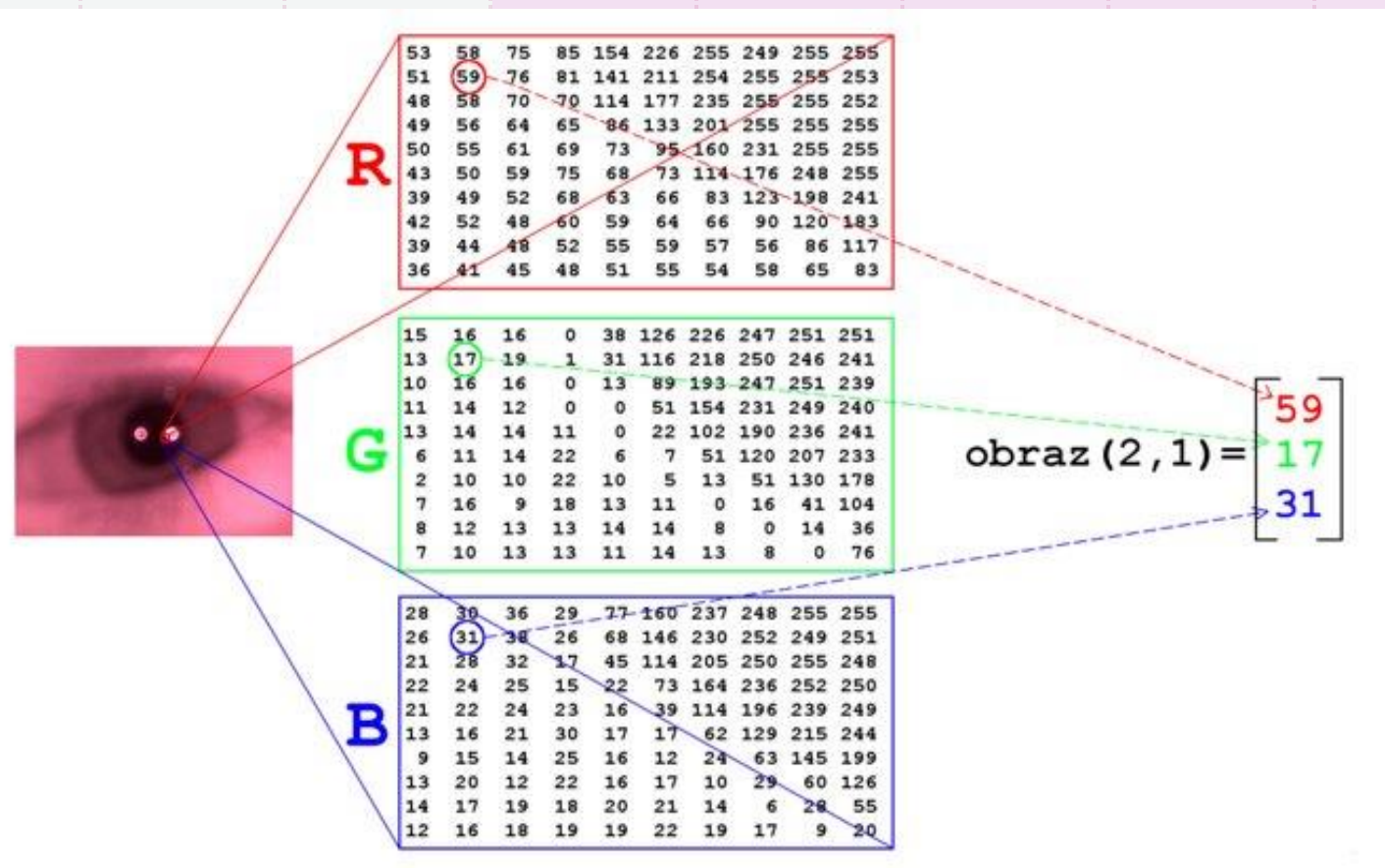
# Uogólnienie obrazu na skalę szarości (reprezentacja funkcyjna)



28	29	36	28	77	159	237	248	252	254
26	31	38	28	69	148	230	251	250	245
23	29	34	22	47	117	207	251	252	242
24	28	29	20	27	78	169	239	253	247
25	27	29	30	23	47	121	203	242	248
17	23	27	39	26	29	71	138	218	242
13	22	23	36	27	24	34	74	153	199
18	28	21	31	27	28	21	40	67	130
18	21	24	24	26	28	24	16	37	62
16	21	24	23	24	27	26	24	20	31



# Uogólnienie obrazu na kolor (reprezentacja macierzowa)





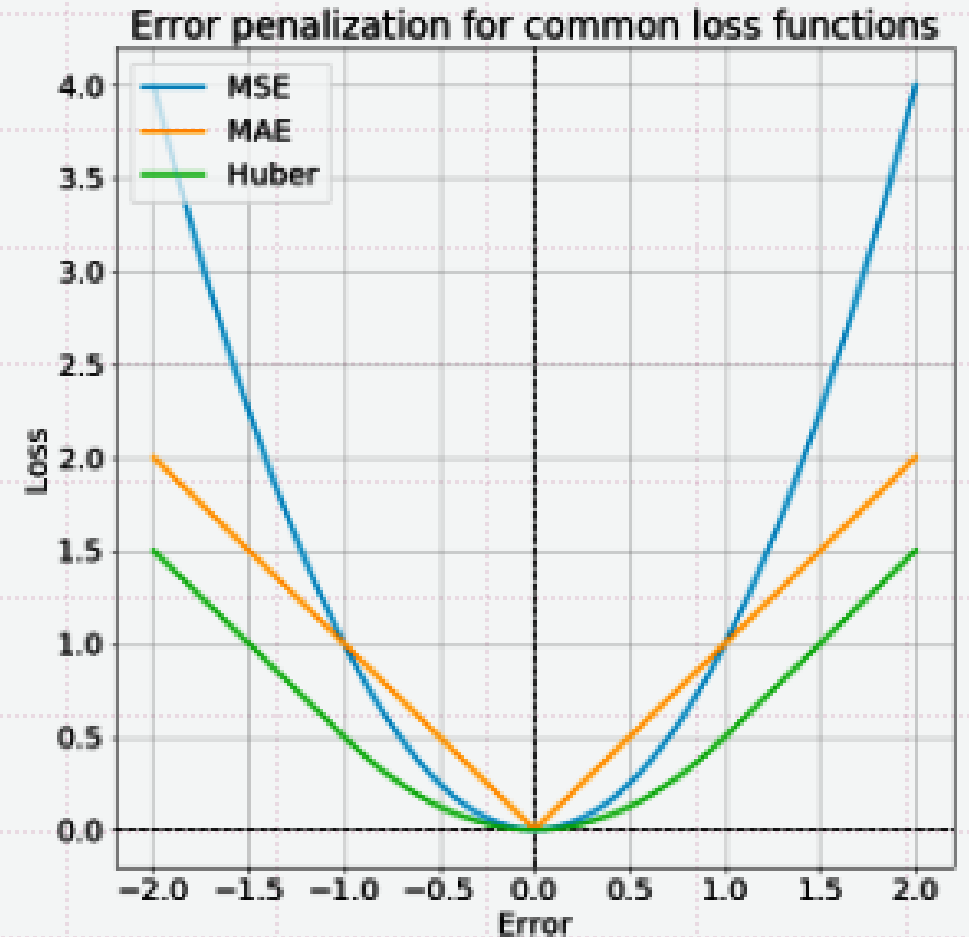
# Loss functions

# Popularne funkcje kosztów dla regresji liniowej

MSE  $L(a) = a^2$

MAE  $L(a) = |a|$

Huber  $L_{\delta}(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta \cdot (|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$





# Popularne funkcje kosztów dla regresji liniowej

## 1. Średni błąd kwadratowy (*ang. mean square error, mse*)

- mocno każe outlier-y
- duża czułość w pobliżu minimum

## 2. Średni błąd absolutny (*ang. mean absolute error, mae*)

- liniowy względem wartości błędu
- mała czułość w pobliżu minimum

## 3. Błąd Hubera (*ang. Huber error*)

- zachowuje się podobnie do mse w pobliżu minimum (zwiększenie czułości w pobliżu min)
- zachowuje się podobnie do mae z dala od minimum (zmniejszenie wpływu outlier-ów)