Sieci neuronowe - wprowadzenie

Krzysztof Halawa

Wstęp

2024

Polecana literatura do samodzielnej nauki

[FChollet] **Francois Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras, Helion, 2019**

[AGeron] **Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Wyd. III, Helion, 2023**

[EStevens] Eli Stevens, Luca Antiga, Thomas Viehmann, Deep Learning with PyTorch, MANNING, 2020 - dobre do nauki PyTorcha, ale gorzej do zrozumienia zasad działania sieci

[Rotman] Denis Rothman Transformers for Natural Language Processing and Computer Vision - Third Edition: Explore Generative AI and Large Language Models with Hugging Face, ChatGPT, GPT-4V, and DALL-E 3 3rd ed., Pact Publishing Ltd. 2024 - transformery, berty, gpt, DALL-E itp.

[Transformer] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention Is All You Need v7, https://arxiv.org/abs/1706.03762

Literatura ułatwiająca

Zasady działania algorytmów gradientowych I i II rzędu są pokazane m.in. w: [SOsowski] Stanisław Osowski, Sieci neuronowe do przetwarzania informacji, OWPW, 2020, (uwaga MLP jest nazywany w książce siecią sigmoidalną, w wydaniu z 2020 pojawił się krótki rozdział o sieciach głębokich). Ta książka ułatwia zrozumienie podstaw działania alg. gradientowych I i II rzędu

[S.OsowskiMatlabPython] Matematyczne modele uczenia maszynowego w językach MATLAB i PYTHON, OWPW, 2023 - różne metody uczenia maszynowego i więcej architektur sieci głębokich ale bez alg. gradientowych II rzędu, które nie są stosowane w sieciach głębokich.

W [AGeron] są podane alg. I rzędu obecnie stosowane do głębokiego uczenia

Trochę historii

1943 - Model neuronu McCullocha-Pittsa
1958 - Perceptron - Frank Roseblatt
1960 - ADALINE - prof. Bernard Widrow i absolwent Ted Hoff
1969 - Książka Marvin Minsky, Seymour Papert, 'Perceptrons: an introduction to computational geometry' pokazała ograniczenia jednowarstwowych sieci (np. problem nieliniowo separowalny XOR) i rozpoczęła 'dark ages of neural networks'. W II połowie lat 80. po raz kolejny "odkryto" algorytm wstecznej propagacji błędów. Popularne stały się wielowarstwowe perceptrony.
Następnie maszyny wektorów nośnych SVM (ang. Support Vector Machines). W 1995 ukazał się artykuł V.Vapnik, C.Cortes, 'Support-Vector Networks', Machine Learning 20, Nr. 3, str. 273-297. (SVMy do problemów liniowych opublikowali już w 1964).
Drzewa decyzyjne i lasy losowe - znane od dawna. W 2010 lasy częściej wykorzystywane w konkursach Kaggle niż SVMy.
Obecnie do trudnych zadań często stosowane uczenie głębokie (ang. deep learning)
Od kilkunastu lat bardzo popularna biblioteka do lasów losowych XGBoost - (eXtreme Gradient Boosting)
2017 - pierwszy transformer (mechanizm uwagi)

Keras https://keras.io/

Bardzo popularna i dobrze zaprojektowana darmowa biblioteka umożliwiająca łatwe tworzenie różnorodnych architektur sieci neuronowych (dostępna od 2015r.). Może być użyta w projektach komercyjnych (liberalna licencja MiT). Umożliwia trening na CPU lub na niektórych GPU.
Trening dużych modeli na GPU jest znacznie szybszy niż na CPU.

Do wersji 2.3 Keras wspierał wiele "backendów" np.: TensorFlow (rozwijany przez google), Theano (MILA lab at Université de Montréal), Microsoft Cognitive Toolkit (CNTK) PlaidML.

Pod koniec 2016 Keras można było uruchamiać z Apple Core MX, Java Script, Type Script, Apple Core ML, Apache MXNwt, PlainMD - GPU nie tylko NVidia).

Od wersji 2.4 Keras działał tylko z Tensorflow.

Od wersji 3.0 działa z www.tensorflow.orgTensorFlow, JAX, and PyTorch. Keras stosuje wzorzec projektowy adapter zwany też opakowaniem (ang. wrapper). Obecnie moduł Tensorflow2 zawiera Keras.

Popularne narzędzia do deep learning w 2024

TensorFlow z Keras - Tensorflow utworzono w Google Brain Team

PyTorch - stworzony przez Facebook AI Research lab (FAIR)

2016 - pierwsza wersja Pytorch (licencja BSD)
2017 - pierwsza wersja TensorFlow 1.0.0 (Apache License 2.0)
2019 - Tensorflow 2.0
2021.04 - Pytorch 1.8.0 oraz Tensorflow 2.4.1
2024.03 Tensorflow 2.16 domyślnie z Keras 3.0 (ale jest też Keras 2)
2024 Keras 3.0
2024 Pytorch 2.3 `https://pytorch.org/`
Ultrlytics
Roboflow

Keras 3.0 i JAX

"JAX is NumPy on the CPU, GPU, and TPU, with great automatic differentiation for high-performance machine learning research."
https://jax.readthedocs.io/en/latest/notebooks/quickstart.html

fit w Keras vs training loop w PyTorch

Training loop w PyTorch:
https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

Przydatne dodatki

Scikitlearn - maszynowe uczenie (bez deep learning)
Google Colaboratory (Colab)
Jupyter notebook
XGBoost - (eXtreme Gradient Boosting)
XGBoostbardzo prosto się używa m.in. w pythonie
Oczywiście trzeba znać NumPy i Python3

Konfiguracja na GPU Nvidia

Sterowniki Nvidia, CUDA, CuDNN (PyTorch nie potrzebuje CuDNN), Pytorch lub Tensorflow, ewentualnie jeszcze JAX.
UWAGA na dobór kompaktybilnych wersji sterowników, CUDA i CuDNN.
PyTorch może działać na większej liczbie urządzeń niż Tensorflow

Do nauki niedużych sieci wystarczy CPU :)

Tensory

Tensor zerowymiarowy - skalar (w numpy ndim==0)

Tensor jednowymiarowy - wektor (w numpy ndim==1, jedna oś)

Tensor dwuwymiarowy - macierz (w numpy ndim==2, dwie osie)

Tensor trójwymiarowy - np. obraz kolorowy RGB

...

Utworzenie macierzy w numpy (dwa wiersze, trzy kolumny):

```
import numpy as np
x=np.array([[1,2,3],[4,5,6]])  #Macierz z listy list 1D
print (x.ndim) #Liczba osi (ranga)
print (x.shape) #Kształt tensora
print (x.dtype) #Typ elementów tensora (int64)
```

Numpy

```
>>> import numpy as np
>>> x=np.array([[1,2,3],[4,5,6]])
>>> y=np.array([[7,8,9],[10,11,12]])
>>> print(x)
[[1 2 3]
 [4 5 6]]
>>> x.ndim
2
>>> x.shape
(2, 3)
>>> x.dtype
dtype('int64')
```

Tensory można dodawać, odejmować, mnożyć, dzielić, obliczać iloczyn Hadamarda...

```
>>> print(x+y)
[[ 8 10 12]
 [14 16 18]]
```

Ważne różnice w składni Numpy i Matlab:

W Numpy pierwszy element ma indeks 0 zamiast 1.
Dostęp do elementów za pomocą nawiasów kwadratowych (a nie okrągłych).
Apostrof nie oznacza transpozycji.

```
>>> x[0,0]
1
>>> x[0,:]
array([1, 2, 3])
>>> x[:,2]
array([3, 6])
>>> x[:,0:2]
array([[1, 2],
       [4, 5]])
>>> x[1]
array([4, 5, 6])
>>> x*2
array([[ 2,  4,  6],
       [ 8, 10, 12]])
>>> np.sin(x)
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ]])
>>> x.T    #macierz transponowana
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Keras

W Kerasie można tworzyć sieci neuronowe na dwa sposoby:

Za pomocą klasy `Sequential` można konstruować sieci poprzez dodawanie kolejnych warstwy (stos warstw). Jest to najczęściej stosowany sposób tworzenia typowych architektur sieci (bardzo łatwy).

Za pomocą `functional API` formułuje się skierowane grafy warstw. W ten sposób można stworzyć nietypowe architektury sieci.

Keras

Typowe etapy tworzenia sieci w Kerasie (przykład na następnej str.):

- utwórz lub wczytaj tensory wejściowe i wyjściowe (zwykle z Numpy),
- zdefiniuj sieć (na ogół z `Sequential`)
- skonfiguruj uczenie (model.compile z argumentami loss=f.straty, optimizer=alg.uczenia, metrics=wskaźnik jakości),
- trening sieci (model.fit).

Na następnym slajdzie pokazano przykład tworzenia wielowarstwowego perceptronu.

Przykład

Zadanie regresji. Nauka funkcji dwóch zmiennych $f(x_1, x_2) = sin(x_1) + x_2^2$.
Wielowarstwowy perceptron: 2 wejścia, jedna warstwa ukryta z 10 neuronami z funkcją aktywacji tangens hiperboliczny, w warstwie wyjściowej jeden neuron z liniową f. aktywacji. Liczba neuronów w warstwie wyjściowej = liczbie wyjść sieci.

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense

#Tworzenie zbioru uczącego:
N = 200 #liczba elementów zbioru uczącego
X = np.random.rand(N,2)  #macierz liczb pseudolosowych 200 x 2 o rozkladzie jednostajnym na przedziale [0,1)
Y = np.sin (X[:,0]) + X[:,1]**2  #Y ma 200 elementow

#Utworzenie sieci neuronowej:
model = Sequential()
model.add(Dense(units=10, activation='tanh', input_dim=2))
model.add(Dense(units=1, activation='linear'))

model.compile(loss='mse', optimizer='sgd', metrics=['mse']) #funkcja celu (straty) MSE, algorytm uczenia rmsprop

#Trening sieci:
model.fit(X, Y, epochs=100, batch_size=40)   #100epok, mini batche 40 elementowe
```

W wyniku wykonania tego kodu otrzymuje się nauczoną sieć. Do obliczenia jej wyjścia wystarczy `y=model( NewX )`
gdzie `NewX` oznacza tensor z wartościami wejść sieci.

Jeżeli sieć była uczona dla x_1 i x_2 należących do przedziału [0,1), to tylko na tym przedziale będzie dobrze aproksymowała.

uczenie wsadowe, on-line i mini wsady (batche)

W przykładzie zbiór danych składał się z 200 elementów (par).

Uczenie odbywało się za pomocą mini-batchy (wsadów) o rozmiarze 40, tzn. w każdej iteracji gradientowego algorytmu uczenia wykorzystywanych było 40 elementów ze zbioru uczącego.

Epoka - jednokrotne wykorzystanie danych z całego zbioru uczącego.

W epoce było wykonywanych $\frac{200}{40} = 5$ iteracji alg. uczenia.

W uczeniu wsadowym (ang. batch) w każdej iteracji są wykorzystywane wszystkie dane ze zbioru uczącego.

W uczeniu on-line w iteracjach są używane tylko pojedyncze elementy ze zbioru uczącego, które są wybierane w losowej kolejności (bez losowania kolejności mogłyby tworzyć się cykle podczas nauki, w których sieć dopasowywałaby cyklicznie wagi do kolejnych części zbioru uczącego).

Mini-batche są podejściem pośrednim między uczeniem wsadowym a on-line.

Zwiększając wsad można skrócić epokę (mniejszy transfer danych między CPU i GPU)

Nauka z mini-batchami wymaga mniejszej pamięci niż w trybie wsadowym (ma to istotne znaczenie dla ogromnych zbiorów danych a także dla dużych sieci głębokich. Zbiory danych mogą zawierać miliony obrazów...)

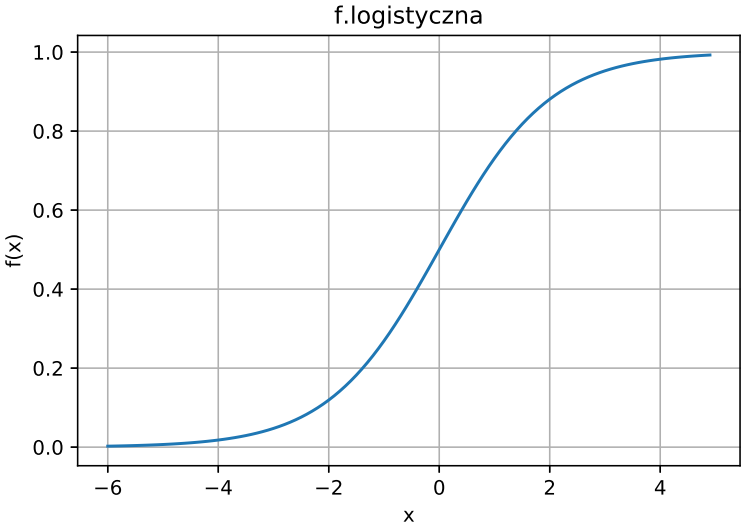
popularne funkcje aktywacji - funkcje sigmoidalne

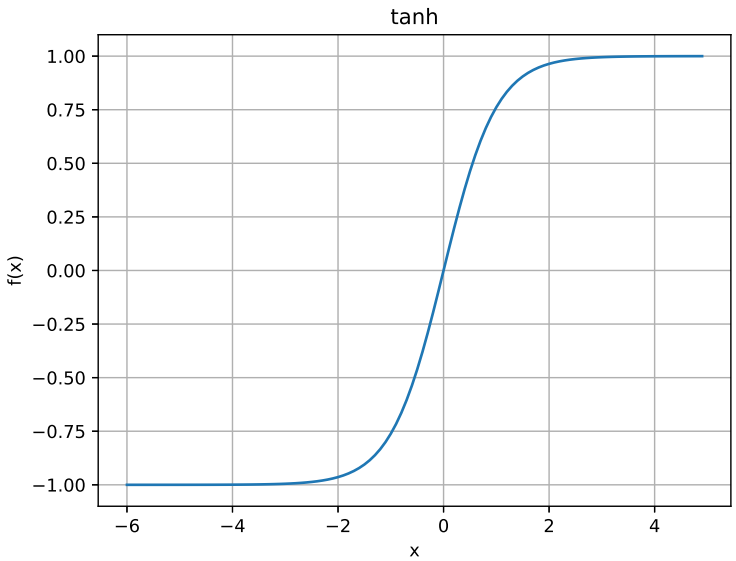
Funkcje sigmoidalne (mają charakterystyczny kształt przypominający literę s) np:
funkcja logistyczna - keras.activations.sigmoid(x):

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{1}$$

tangens hiperboliczny - keras.activations.tanh(x):

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{2}$$





Funkcje sigmoidalne są stosowane w warstwach ukrytych i w warstwie wyjściowej.

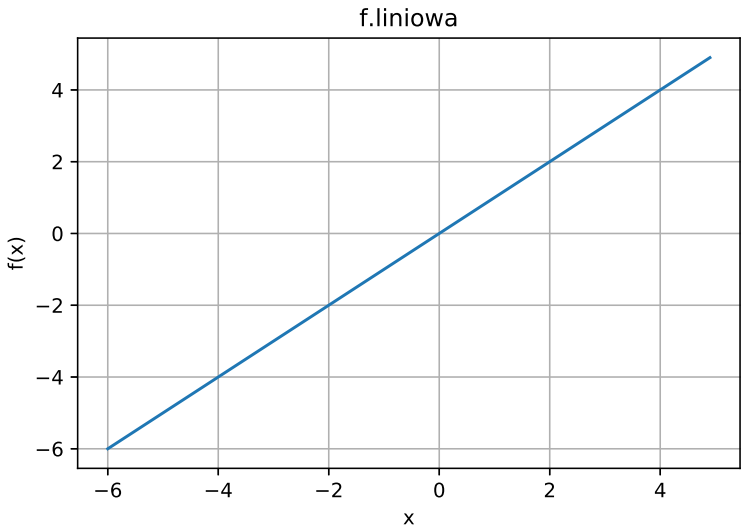
popularne funkcje aktywacji

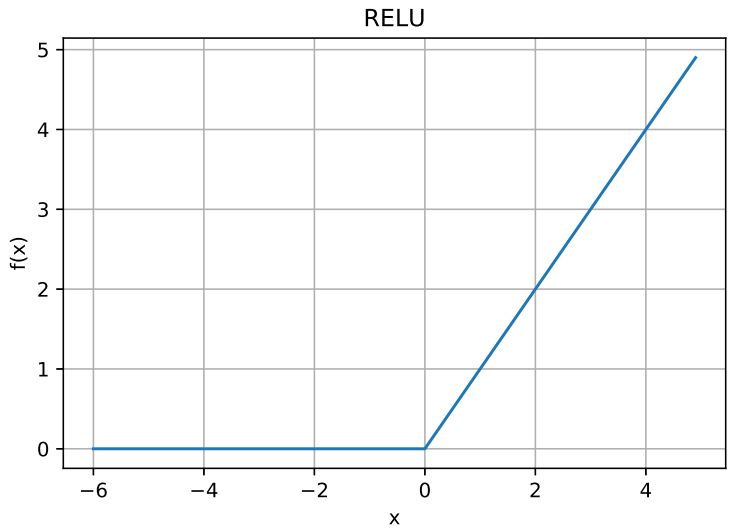
Liniowa funkcja = brak funkcji aktywacji - keras.activations.linear

$$f(x) = x, \tag{3}$$

Funkcja ReLU (ang. Rectified Linear Unit) keras.activations.relu

$$f(x) = \max(0, x), \tag{4}$$



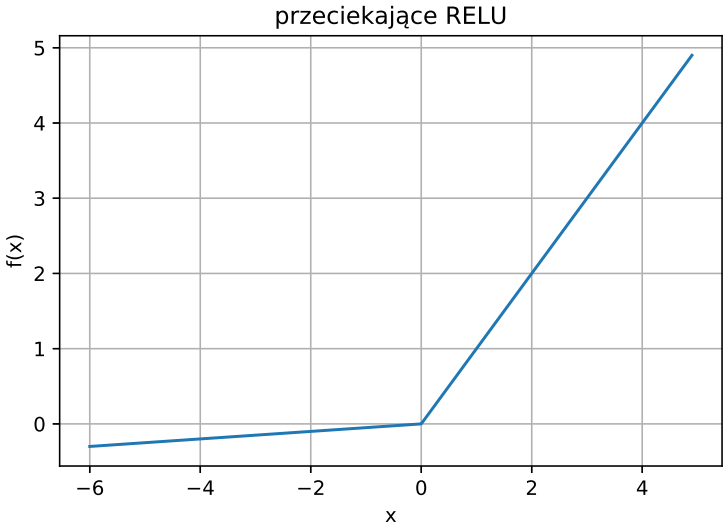


Funkcje ReLU są stosowane w warstwach ukrytych. Funkcja liniowa jest stosowana w warstwie wyjściowej w zadanich regresji. Więcej funkcji aktywacji jest podanych na stronie https://keras.io/activations/

popularne funkcje aktywacji

Liniowa funkcja = brak funkcji aktywacji - Funkcja przeciekające ReLU (ang. Leaky Rectified Linear Unit) keras.activations.relu z parametrem alpha=a, gdzie najczęściej a=0.01

$$f(x) = \begin{cases} x & \text{dla } x \geqslant 0 \\ \alpha x & \text{dla } x < 0 \end{cases} \tag{5}$$



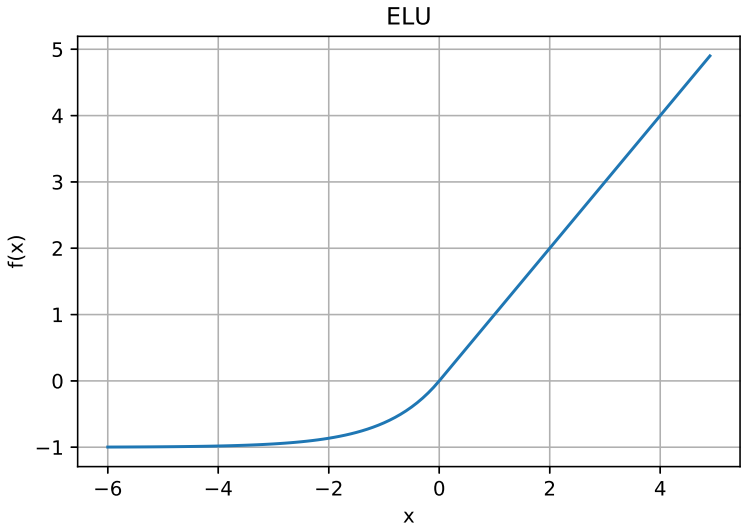
W przypadku parametrycznych ReLU (PReLU ang. parametric ReLU) wartość parametru *a* jest zmieniana podczas treningu jak pozostałe wagi. Więcej funkcji aktywacji jest podanych na stronie https://keras.io/activations/

Funkcje aktywacji

Istnieją różnorodne modyfikacje ReLU i funkcje podobne do ReLU, którą są znacznie rzadziej stosowane niż ReLU, np. ELU, SELu, GELU przeciekające ReLU (ang. leaky ReLU) itd. Funkcja ELU (ang. exponential linear unit)

$$f(x) = \begin{cases} x & \text{dla } x \geqslant 0, \\ c \cdot (e^x - 1) & \text{dla } x < 0, \end{cases} \tag{6}$$

gdzie c jest stałą.



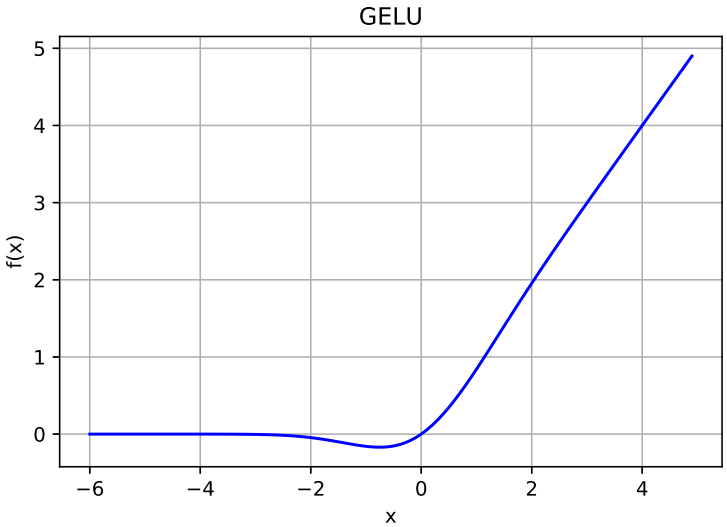
Funkcje podobne do ReLU są stosowane w warstwach ukrytych. Więcej funkcji aktywacji jest podanych na stronie https://keras.io/activations/

Funkcje aktywacji

Istnieją różnorodne modyfikacje ReLU i funkcje podobne do ReLU, którą są znacznie rzadziej stosowane niż ReLU, np. ELU, SELu, GELU przeciekające ReLU (ang. leaky ReLU) itd. Funkcja GELU (ang. Gaussian Error Linear Unit)

$$f(x) = x\Phi(x) \tag{7}$$

gdzie $\Phi(x)$ oznacza dystrybuantę standaryzowanego rozkładu Gaussowskiego.



Funkcje podobne do ReLU są stosowane w warstwach ukrytych. Więcej funkcji aktywacji jest podanych na stronie https://keras.io/activations/

Funkcja softmax jest stosowana tylko w sieciach do zadań klasyfikacji w gęstej warstwie wyjściowej (warstwy `Dense` w Kerasie).
Wartość wyjścia j-tego neuronu z f.aktywacji softmax jest określona wzorem

$$f(\mathrm{u}) = \frac{e^{\mathrm{u}^T \mathrm{w}_j}}{\sum_{k=1}^{K} e^{\mathrm{u}^T \mathrm{w}_k}}, \tag{8}$$

gdzie K oznacza liczbę neuronów w warstwie wyjściowej, v_j oznacza wektor wag j-tego neuronu, u oznacza wektor z wartościami wejść neuronów w ostatniej warstwie.
Dzięki mianownikowi wartości wyjść sieci należą do przedziału $[0, 1]$ oraz suma wartości wszystkich wyjść jest równa 1 (pewne podobieństwo do prawdopodobieństwa - prawdopodobieństwo $\in [0, 1]$ i suma prawdopodobieństw wszystkich możliwych zdarzeń równa 1.)

Więcej funkcji aktywacji jest podanych w https://keras.io/activations/

Tabela: Dobór funkcji straty (celu) oraz funkcji aktywacji do typowych zadań

Zadanie	Funkcja aktywacji w ostatniej warstwie	Funkcja straty (celu)	Funkcja celu w Keras
Regresja	liniowa	Błąd średniokwadratowy MSE	mse
Klasyfikacja binarna	sigmoid	Binarna entropia krzyżowa	binary_crossentropy
Wieloklasowa klasyfikacja (jednoetykietowa) (ang. multilabel classification)	softmax	Kategorialna entropia krzyżowa	categorical_crossentropy
Wieloklasowa klasyfikacja wielowyjściowa (ang. multioutput-multiclass classification lub multioutput classification)	sigmoid	Binarna entropia krzyżowa	binary_crossentropy

Klasyfikacja binarna - klasyfikacja do dwóch klas (są tylko dwie klasy)
Wieloklasowa klasyfikacja (binarna, jednoetykietowa) - (liczba klas$>$ 2) dana należą do pojedynczych klasy - mają przypisana pojedyńcze etykietę
Wieloklasowa klasyfikacja (wielowyjściowa, wieloetykietowa) - dana może należeć jednocześnie do kilku klas (mogą mieć przypisanych jednocześnie kilka etykiet).

Funkcja straty - f.celu (ang. oss function, objective function, loss function, cost function)

Dla nietypowych zadań istnieje możliwość tworzenia własnych f. straty w Kerasie. Funkcje te można zaprojektować uwzględniając specyfikę zagadnienia i pożądane cele optymalizacji. Funkcja powinna być zaprojektowana tak, aby dla lepszych sieci miała mniejszą wartość niż dla kiepsko działających sieci.

UWAGA

Podstawowe zasady uczenia algorytmami gradientowymi były przedstawione na II wykładzie i są pokazan w książkach [SOsowski]. W [AGeron] są alg. I rzędu obecnie stosowane do głębokiego uczenia. Na dzisiejszym wykładzie są wspomniane tylko w wielkim skrócie. Większość osób poznała je na studiach inżynierskich.

Oznaczenia

n - liczba parametrów sieci zmienianych podczas treningu (liczba wag)
E - funkcja straty (np. MSE, binarna entropia, kategoryczna entropia)
Wektor wag sieci:

$$\mathrm{w} = [w_1, w_2, \ldots, w_n]^T \tag{9}$$

Wektor gradientu:

$$\nabla E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right]^T \tag{10}$$

Macierz drugich pochodnych zwana macierzą Hessego lub hesjanem [SOsowski] dana jest wzorem

$$\mathrm{H} = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1 \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_n} \end{bmatrix}. \tag{11}$$

Hesjan jest macierzą symetryczną.

Sieci MLP i CNN są uczone iteracyjnie algorytmami gradientowymi.

Algorytmy gradientowe, które bazują jedynie na wartościach gradientu nie wykorzystując informacji o dokładnych lub przybliżonych wartościach Hesjanu bądź jego odwrotności, są nazywane algorytmami pierwszego rzędu.

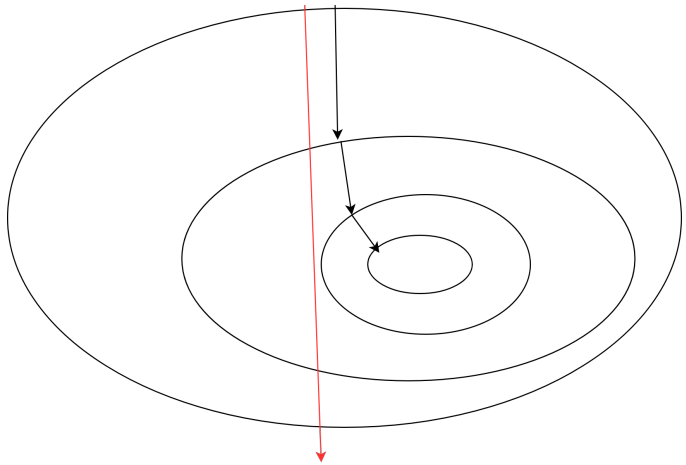
Zmiany wag w kolejnych iteracjach metody najszybszego spadku następują w kierunku antygradientu (najszybszego spadku f.celu):

$$\mathrm{w}_{i+1} = \mathrm{w}_i - \eta_i \nabla E, \tag{12}$$

gdzie η_i jest wielkością skalarną zwaną współczynnikiem uczenia, w_i oznacza wektor wag w i-tej iteracji. W metodzie najszybszego spadku współczynnik uczenia jest dobierany w każdej iteracji (optymalizacja kierunkowa - jak daleko można się posunąć w kierunku antygradientu zanim f.celu nie zacznie rosnąć).
Metoda gradientu prostego różni się od metody najszybszego spadku brakiem optymalizacji kierunkowej.

Metody gradientowe

Przypomnienie - w metodach gradientowych oprócz doboru kierunku istotne jest też określenie jak daleko można podążyć w danym kierunku. Na czerwono zaznaczono sytuację dla zbyt dużego współczynnika uczenia. Z kolei zbyt mała jego wartość powodowałaby dużą liczbę drobnych kroków. Optymalizacja kierunkowa służy do odpowiedniego doboru η.
W MLP początkowe wartości wag są liczbami pseudolosowymi, więc można zakończyć uczenie w różnorodnych minimach lokalnych funkcji celu (działanie dwóch sieci o identycznej budowie może się z tego powodu bardzo różnić.)
W metodzie najszybszego spadku kierunek zmian wag jest prostopadły do poziomic funkcji celu. Gdyby we wzorze (12) zmienić + na -, to zmiany następowałyby w kierunku najszybszego wzrostu funkcji celu.



Rysunek: Elipsy oznaczają poziomice funkcji celu (gdyby sieć miała tylko 2 wagi, to można by poziomice przedstawić na płaszczyźnie). W środku najmniejszej elipsy znajduje się minimum lokalne.

Dodanie momentu

Poprawę działania można często uzyskać poprzez dodanie tzw. momentu. Wówczas

$$w_{i+1} = w_i - \eta_i \nabla E + \alpha(w_i - w_{i-1}), \tag{13}$$

gdzie α jest skalarnym współczynnikiem momentu zwykle należącym do przedziału [0,1]. Sposoby doboru i zmian α podczas uczenia są opisane m.in. w [SOsowski].
Dzięki dodaniu momentu zmiany wektora wag nie zależą tylko od aktualnej wartości gradientu, ale również od poprzednich wartości wag. Zbliżanie się do minimum lokalnego f.celu jest mniej kręte (chaotyczne) oraz niekiedy umożliwia wyskoczenie z płytkiego minimum lokalnego.

```
opt = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
model.compile(loss='mse', optimizer=opt, metrics=['mse'])
```

Zmienny współczynnik uczenia

Ponadto bliżej minimum f.celu (straty) można precyzyjniej z mniejszym współczynnikiem uczenia

```
LearningRate = LearningRate * 1/(1 + decay * epoch)
```

```
epochs = 50
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8
sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

SGD - Keras https://keras.io/api/optimizers/sgd/

Domyślne argumenty:

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD", **kwargs
)
```

```
############  Gdy momentum==0:\\
w = w - learning_rate * g\\


############ Dla momentum>0 i nesterov==False:\\
velocity = momentum * velocity - learning_rate * g
w = w + velocity


########### Dla momentum>0 i nestorov==True: (gradient nie jest obliczany w bieżacym miejscu lecz w miejscu uwzględniającym przesunięc
velocity = momentum * velocity - learning_rate * g
w = w + momentum * velocity - learning_rate * g
```

Wyjaśnienie momentu z nestorov=True w:
https://dominikschmidt.xyz/nesterov-momentum/

http://proceedings.mlr.press/v28/sutskever13.pdf

Uczenie MLP - wspomnienie metod II rzędu, które były omawiane na II wykładzie

W metodzie newtonowskiej zmiany wag następują zgodnie z formułą

$$\mathrm{w}_{i+1} = \mathrm{w}_i - \eta_i [\mathrm{H}]^{-1} \nabla E. \tag{14}$$

Współczynnik uczenia η_i jest dobierany w każdej iteracji jak w metodzie najszybszego spadku. Wyprowadzenie (14) jest w [SOsowski]. W algorytmach zmiennej metryki zamiast dokładnie wyznaczać elementy hesjanu w zależności (14), stosuje się odpowiednio obliczone przybliżenie H lub $[\mathrm{H}]^{-1}$. Do aproksymowania wartości $[\mathrm{H}]^{-1}$ można np. wykorzystać algorytm Broydena-Fletchera-Goldfarba-Shannon lub formułę Davidona-Fletchera-Powella.

Algorytm Levenberga-Marquardta łączy część zalet metod drugiego rzędu z niektórymi atutami metody najszybszego spadku. Kierunek poszukiwań w tym algorytmie zależy od wartości skalarnej zwanej parametrem Levenberga-Marquardta, którego wartość zmienia się odpowiednio podczas uczenia i powoduje, że kierunek poszukiwań może być zbliżony do kierunku w metodzie najszybszego spadku lub bardziej przypominać sposób modyfikacji wag w metodzie Newtona. Algorytm Levenberga-Marquardta wymaga specyficznej postaci funkcji celu. Może być użyty z MSE.

Uwaga

W Kerasie nie ma alg. gradientowych II rzędu

Dlaczego proste algorytmy pierwszego rzędu są stosowane do uczenia sieci głębokich?

Istotne wady i zalety metod gradientowych pierwszego rzędu

Zalety:

Złożoność pamięciowa O(n), co jest kluczowe dla sieci głębokich! Głęboka sieć konwolucyjna VGG16 ma 138357544 parametrów (wag) zmienianych podczas nauki. Macierz hesjanu miałaby $138357544^2 = 19142809981711936 \approx 19 \cdot 10^{15}$ elementów czyli około 19 biliardów elementów (19 peta elementów)!

Prosta implementacja

Wady:
wolna zbieżność (liniowa).

Uczenie MLP i CNN - wytrenowane sieci głębokie dostępne w Keras

W lewej kolumnie są linki do dokumentacji sieci w Kerasie

Model	Rozmiar	Top-1 Dokładność (ang. Accuracy)	Top-5 Dokładność	Parametery	Głębokość
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88

Tabela: Dane sieci z `https://keras.io/applications/`

Dokładność Top 5 - prawidłowa klasa wśród pięciu najbardziej prawdopodobnych klas wskazywanych przez sieć

Dokładność Top 1 - prawidłowa klasa wskazana przez sieć jako najbardziej prawdopodobna

Najmniej parametrów ma MobileNetV2. Nawet dla tej sieci hesjan miałby aż $3538984^2 = 3538984 \approx 13 \cdot 10^{12}$ elementów (ponad 12 teta elementów)!

Ucząc duże sieci na małych zbiorach danych można łatwo przeuczyć (ang. overfitting) i utracić zdolność generalizacji. Im większy zbiór danych uczących tym lepiej.

Dostępne modele w Kerasie

https://keras.io/api/applications/

Dygresja Keras - wczytanie VGG16 i podgląd jej architektury

```
>>>from keras.applications.vgg16 import VGG16
>>>model = VGG16()
>>>print(model.summary())

Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
```

cd. z poprzedniego slajdu

```
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
None
```

Proszę zwrócić uwagę na rozmiary kolejnych warstw konwolucyjnych

Kolejne warstwy konwolucyjne mają więcej cech i mniejsze rozmiary tensorów wyjściowych, co jest typowym rozwiązaniem.

Można też uzyskać graficzną reprezentację sieci, która jest bardziej czytelna dla skomplikowanych struktur:

```
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```

Uczenie MLP i CNN

Algorytmy gradientowe drugiego rzędu są często stosowane do uczenia płytkich wielowarstwowych perceptronów.

Wielowarstwowe perceptrony

Wielowarstwowe perceptrony posiadają co najmniej jedna warstwę ukrytą (często jest tylko jedna warstwa ukryta). W warstwach są neurony McCullocha-Pittsa (w Kerasie warstwy Dense). W warstwach ukrytych są stosowane nieliniowe funkcje aktywacji np. tanh lub ReLU.

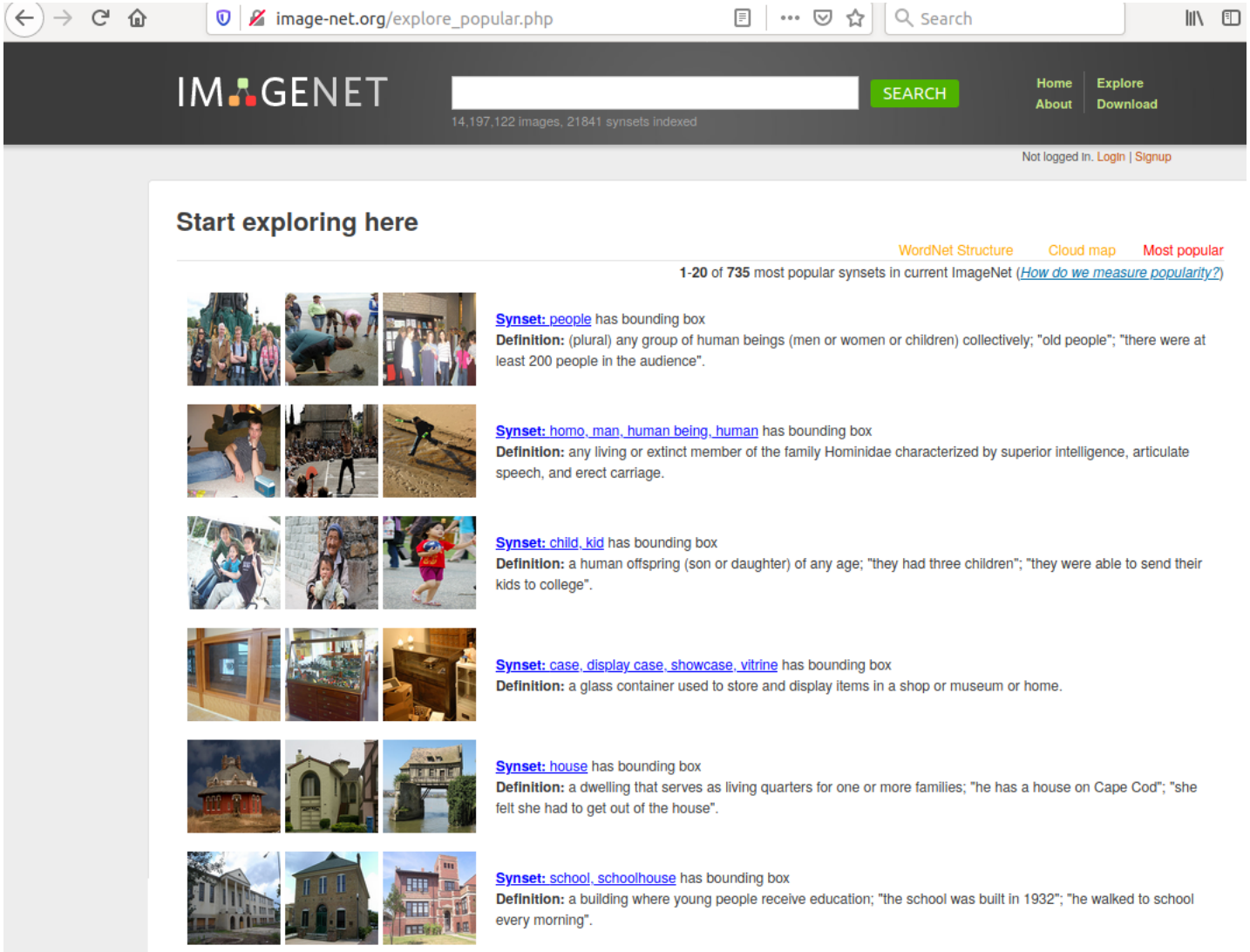
Tworzenie prostego MLP do klasyfikacji (5 wejść, 2 warstwy ukryte z 10 i 7 neuronów, 4 wyjścia (4 klasy)

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(10, input_shape=(5,), activation='relu'))
model.add(Dense(7, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

Algorytmy gradientowe II rzędu

- Złożoność pamięciowa $O(n^2)$. Ponadto złożoność obliczeniowa pojedynczej iteracji jest też $O(n^2)$
- Zbieżność kwadratowa
- Trudniejsze do samodzielnego zaprogramowania.

ImageNet - kilkanaście milionów obrazów podzielonych na 22tys. kategorii. Podział na kategorie i podkategorie według WordNet https://wordnet.princeton.edu/ (m.in. hiperonimy, hiponimy, synonimy...)
Przykłady zdjęć:



Po rozpakowaniu ponad 1 TB.

Zbiór obrazów ImageNet

Przeważnie nazwa ImageNet jest używana na podzbiór z konkursów ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
https://www.image-net.org/challenges/LSVRC/
1000 kategorii
100000 testowych obrazów
1,2 miliona treningowych obrazów.

```
model = VGG16(weights='imagenet')
model = VGG16(weights='imagenet', include_top=False)
```

Keras - VGG16 i VGG19

```
tf.keras.applications.VGG16(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

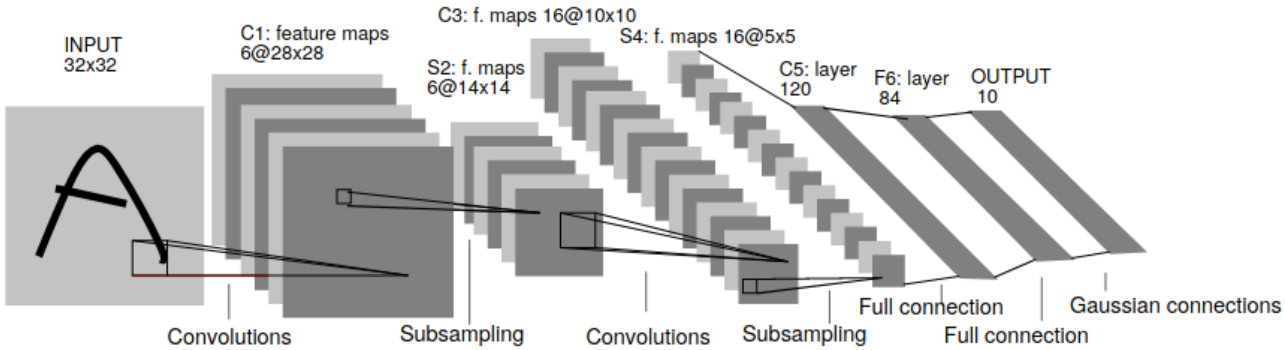
Tworzenie zbiorów danych

Jeżeli nie ma odpowiedniego zbioru danych uczących, to w celu jego utworzenia może się okazać niezbędne zatrudnienie grupy osób wykonujących żmudne przypisanie etykiet do danych (przyporządkowywanie do klas). Samo tworzenie sieci na ogół jest znacznie mniej pracochłonne i często wymaga jedynie napisania krótkiego kodu w Kerasie (po utworzeniu sieci nie należy zapomnieć ją przetestować na danych nieużytych do uczenia). Są narzędzia do zaznaczania ramkami obiektów na zdjęciach oraz do obrysowania obiektywów (do segmentacji).

Roboflow `https://www.superannotate.com/blog/best-data-labeling-tools`
`https://smartone.ai/blog/top-10-open-source-data-labeling-tools-for-computer-vision/`
`https://www.labellerr.com/blog/top-image-labeling-tools/`

Sieci Konwolucyjne (splotowe) ang. Convolutional Neural Networks

Sieci konwolucyjne są bardzo często stosowanymi sieciami. Powszechnie używane m.in. do przetwarzania obrazów (klasyfikacja, segmentacja).



Rysunek: Rysunek architektury pierwszej sieci LeCun-5 z 1989 z przełomowego artykułu: LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Na rysunku subsampling - to maxpooling. Na końcu znajdują się warstwy gęste. Do klasyfikacji liter we współczesnych sieciach w ostatniej warstwie byłaby f.aktywacji softmax.
Dodanie warstwy konwolucyjnej w Kerasie z 32 mapami cech (ang. feature maps) 3x3 - rozmiar jądra:

```
model.add(Conv2D(32, (3, 3), activation="relu"))
```

Sieci Konwolucyjne (splotowe) ang. Convolutional Neural Networks

Każda mapa cech (ang. feature map) wykrywa inną cechę. Pierwsza warstwa konwolucyjna może się nauczyć wykrywać stosunkowo proste cechy np.: kąty ostre, przecięcia się linii itp. Kolejna warstwa na podstawie prostych cech z poprzedniej warstwy może wykrywać bardziej skomplikowane przedmioty np. oczy, uszy, nos... Jeszcze dalsza warstwa konwolucyjna na podstawie wcześniej ekstrachowanych części twarzy może wykonywać detekcję całych twarzy.
Ekstrakcja cech - od prostych do coraz bardziej abstrakcyjnych. Można wizualizować wartości wag z niższych warstw sieci konwolucyjnych na dwuwymiarowych obrazach i w ten sposób przeglądać jakie cechy są wykrywane. Bardzo abstrakcyjne cechy (z dalszych warstw) są znacznie trudniejsze do interpretacji przez człowieka.

Proszę popatrzeć jeszcze raz na budowę nowszej sieci konwolucyjnej VGG16

```
>>>from keras.applications.vgg16 import VGG16
>>>model = VGG16()
>>>print(model.summary())
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
```

cd. z poprzedniego slajdu

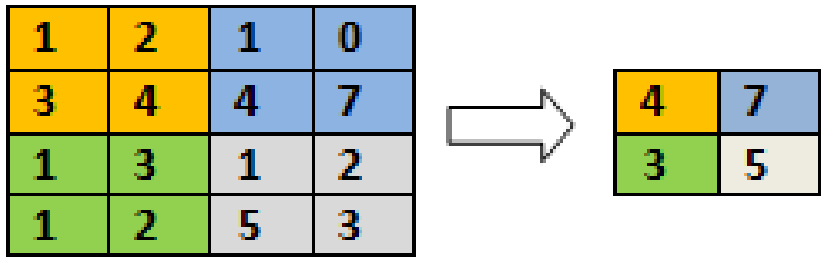
```
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
```

Proszę zwrócić uwagę na rozmiary kolejnych warstw konwolucyjnych

Kolejne warstwy mają coraz mniejsze rozmiary, co jest typowym podejściem przy tworzeniu sieci konwolucyjnych.

Maxpooling

Maxpooling powoduje zmniejszenie rozdzielczości. Sposób działania przedstawiono na rysunku (z fragmentów zaznaczonych różnymi kolorami są wybierane największe wartości).



Czasami bywają też stosowane inne metody poolingu np. z uśrednianiem (AveragePooling)

Warstwy konwolucyjne (przykład - splot2D)

Splot 2D z poniższym jądrem o rozmiarze 3x3:

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

Elementy jądra są przemnażane przez elementy zielonego obszaru po lewej.
$1 \cdot 1 + (-1) \cdot 2 + 0 \cdot 3 + 0 \cdot 0 + 2 \cdot 0 + 0 \cdot 1 + 1 \cdot 3 + 0 \cdot 2 + (-1) \cdot 1 = 1$
Po prawej stronie zielona kratka zawiera wynik konwolucji jądra z lewą stroną:

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

Warstwy konwolucyjne (przykład splot2D)

Splot z jądrem

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

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

Konwolucje (Splot2D)

Splot z jądrem

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

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

Wyznaczania wartości elementów drugiego wiersza.
Splot z jądrem

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

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

Warstwy konwolucyjne (Przykład - splot2D)

Wyznaczanie elementów trzeciego wiersza.
Splot z jądrem

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

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

1	2	3	2	3
0	0	1	0	0
3	2	1	1	1
1	0	0	1	0
1	1	1	0	1

1	5	1
2	0	2
1	3	2

F.aktywacji warstwy konwolucyjnej

Warstwy konwolucyjne - f.aktywacji

Wartość funkcja aktywacji jest wyznaczana dla każdego elementu macierzy otrzymanej w wyniku wyznaczenia splotu (macierzy po prawej stronie powyższych slajdów). W niektórych publikacjach funkcja aktywacji jest traktowana jako część warstwy konwolucyjnej, a w innych artykułach jako osobna warstwa. W wyniku zastosowania funkcji aktywacji f otrzymywana jest macierz:

$$\begin{bmatrix} f(1+b) & f(5+b) & f(1+b) \\ f(2+b) & f(0+b) & f(2+b) \\ f(1+b) & f(3+b) & f(2+b), \end{bmatrix}$$

gdzie b oznacza wartość progową (jedna z wag dostrajanych podczas uczenia)

Dodanie warstwy konwolucyjnej z funkcją aktywacji ReLU, 32 mapami cech i rozmiarem jądra 3x3:

```
model.add(Conv2D(32, (3, 3), activation="relu"))
```

Warstwy konwolucyjne

Dzięki przesuwaniu jądra cechy mogą być wykrywane w różnych miejscach.

Są też konwolucje 3D dla tensorów trójwymiarowych oraz 1D dla danych w postaci tensorów jednowymiarowych (dla 1D jądro jest wektorem, który jest w analogiczny sposób przesuwany na większym wektorze).

W wyniki działania warstwy konwolucyjnej zmniejsza się rozmiar danych.

Warstwy konwolucyjne - stride

Stride określa o ile pozycji przesuwane jest jądro - przeważnie jest równy 1 tak jak w przykładzie przedstawionym na poprzednich slajdach. Gdyby stride był 2 to wynikiem byłaby macierz:

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

(zielony kwadrat przesuwałby się o dwa elementy zamiast o jeden)

Ładne animacje przedstawiające działanie warstw konwolucyjnych są np. na stronie https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Warstwy konwolucyjne

Analogiczny mechanizm był już na studiach - cyfrowe przetwarzanie obrazów za pomocą dwuwymiarowych splotów masek z obrazami np. maski Sobela itp. W sieciach CNN parametry masek są dobierane automatycznie podczas treningu.

Warstwy konwolucyjne

Padding zerami - dodanie obwoluty wypełnionej zerami.
Przykład:

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	3	2	1	1	1	0
0	1	0	0	1	0	0
0	1	1	1	0	1	0
0	0	0	0	0	0	0

same - padding zerami - rozmiar tensora wyjściowego z warstwy taki jak wejściowego

valid - bez paddingu - warstwa wykorzystuje tylko valid input data

W same zera są dodawane równomiernie z dwóch stron. Jeżeli do wyrównania konieczne jest dodanie nieparzystej liczby kolumn, to wówczas z prawej dodatkowa kolumna.

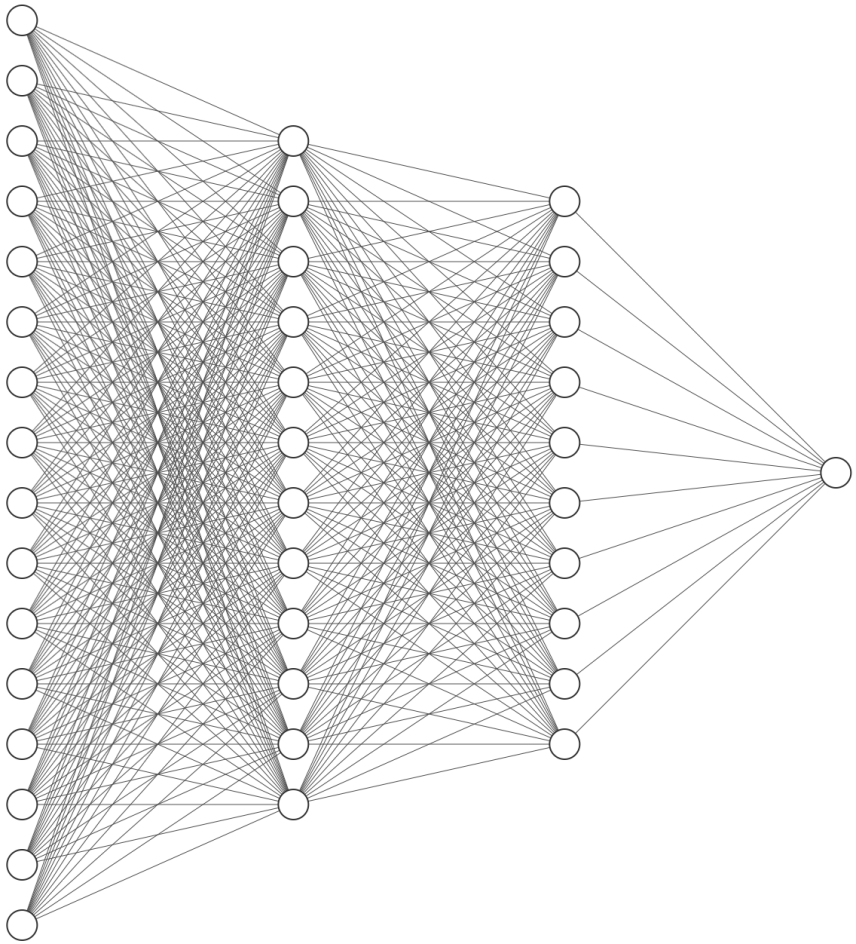
Keras klasa Conv2D

https://keras.io/api/layers/convolution_layers/convolution2d/

```
keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Warstwy gęste (ang. Dense, FC Full Connected)

Wejścia wszystkich neuronów McCulloch-Pittsa (takie same jak w MLP) są połączone ze wszystkimi wyjściami warstwy poprzedniej.
Poniższy rysunek pokazuje trafność nazwy:



Regularyzacja - dropout (porzucanie) - najbardziej popularny sposób regularyzacji!

Została zaproponowana przez E.G.Hintona w 2012r.

W poszczególnych iteracjach treningu, neurony z warstwy mają pewne prawdopodobodiebeństwo p, że mogą zostać czasowo "porzucone" tzn. pominiety w uczeniu. $p \in [0.1, 0.5]$ - współczynnik porzyucnaia (ang. dropout rate).

część wyjść warstwy jest zerowana podczas uczenia (zerowane wyjścia są wybierane losowo w każdej iteracji). ułamek określający część wyzerowanych wyjść - dropout rate

często tylko w trzech szczytowych warstwach z wyjątkiem warstwy wyjściowej, w której nie jest stosowany.

dla sieci CNN często 40-50%, dla sieci rekurencyjnych 20-30%.

po nauczeniu sieci dropout jest wyłączony, połaczenia są mnożone przez prwdopodobieństwo utrzymania (ang. keep propability) $1 - p$, gdyż trzeba zrekompensować większa liczbę aktywnych neuronów wysyłajacych sygnały do następnej warstwy.

Regularyzacja - dropout

Dodanie do dwóch warstw regularyzacji dropout z drop rate 0.4 i 0.3.:

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(10, input_shape=(5,), activation='relu'))
model.add(layers.dropout (0.4)
model.add(Dense(7, activation='relu'))
model.add(layers.dropout (0.3)
model.add(Dense(4, activation='softmax'))
```

Regularyzacja - dropout

można uzyskać lepszą zdolność generalizacji i dokładność np. o 2%. Na pierwszy rzut oka ktoś mógłby pomyśleć, że 2% to nie dużo, ale zauważmy, iż po zwiększeniu dokładności z 96% do 98% dwukrotnie zmniejsza się liczba nieprawidłow sklasyfikowanych :)

dłużej trwa uczenie (dłuższa zbieżność)

z sieciami samonormalizującymi z f.aktywacji SELU zamiast zwykłego porzucania stosuje się alfa porzucanie (ang. alpha dropout)

Regularyzacja z kara za duże wartości wag

.

Przykładowa f.celu:

$$E = \text{MSE} + \alpha \cdot ||\text{w}||, \tag{15}$$

gdzie α jest stałą (im większe α tym większy wpływ kary), w oznacza wektor wag, $|| \cdot ||$ oznacza normę L1 lub L2. Na ogół α ma niewielką wartość mniejszą niż 0.2 (wartość można dobierać eksperymentalnie).

W Kerasie regularyzację L1, L2 można zastosować indywidualnie do wag każdej warstwy (oddzielnie dla wartości progowych (ang. biases) i dla pozostałych wag z wybranej warstwy).

Regularyzacja z normą L1 lub L2

Norma

$$||\mathrm{w}|| = \sum_{k=1}^{n} (|w_k|^p)^{1/p}, \tag{16}$$

gdzie $\mathrm{w} = [w_1, w_2, ..., w_n]^T$ jest wektorem wag (parametrów modelu), $p > 0$.
Regularyzacja LASSO (ang. Least Absolute Shrinkage and Operator Regression) - z normą L1 ($p = 1$)

$$E = \mathrm{MSE} + \alpha \cdot ||\mathrm{w}||_1 = \mathrm{MSE} + \alpha \sum_{k=1}^{n} |w_k|. \tag{17}$$

Za pomocą normy L1 można uzyskać modele rzadkie - eliminacja mniej ważnych parametrów (wartości ≈ 0).
Regularyzacja grzbietowa, regularyzacja Tichonowa (ang. ridge regression, Tikhonov regularization) - z normą L2 do kwadratu

$$E = \mathrm{MSE} + \alpha \cdot ||\mathrm{w}||_2^2 = \mathrm{MSE} + \alpha \sum_{k=1}^{n} w_k^2. \tag{18}$$

Hiperparametr α dobierany jest eksperymentalnie do zadania.

Metoda elastycznej siatki (ang. elastic net) - połączenie regularyzacji grzbietowej i LASSO (dodawane obydwie kary).

Przypomnienie

Przypomnienie - Nie należy oceniać jakości sieci na danych użytych do nauki !!!

Przeważnie podział danych na dwa zbiory: uczący do treningu sieci i testujący do oceny działania sieci.

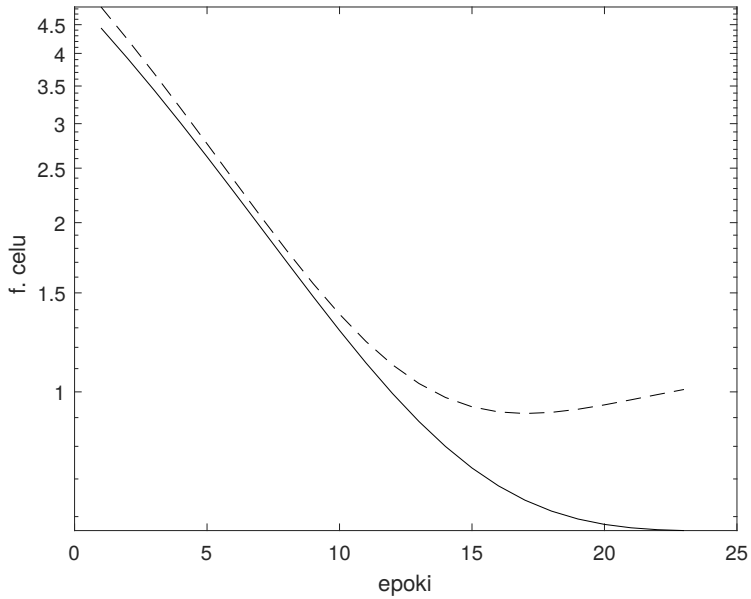
Regularyzacja - Wczesne zatrzymanie

Regularyzacja poprzez wczesne zatrzymanie (ang. early stopping)

Podział danych na trzy rozłączne zbiory: uczący, testujący i DODATKOWO walidujący.

W kolejnych epokach wartość f.celu wyznaczana jest dla zbioru uczącego oraz dla zbioru walidującego.

Gdy wartość f.celu zaczyna rosnąć na zbiorze walidującym to następuje zatrzymanie uczenia (dalsze uczenie powodowałoby przeuczenie - nadmierne dopasowanie do danych uczących i utratę zdolności generalizacji).



Rysunek: Wart. f.celu (linia ciągła - zbiór uczący, linia przerywana - zbiór walidujący).

Regularyzacja - wczesne zatrzymanie

Podczas uczenia mini-batchami lub on-line wartość f.celu prawdopodobnie nie będzie spadała tak ładnie monotonicznie jak na poprzednim slajdzie (ale trend będzie spadkowy). Dlatego decyzję o zatrzymaniu podejmuje się na ogół dopiero po kilku epokach wzrostu na zbiorze walidującym - wartości wag można zachowywać i przywrócić z epoki, w której wartość f. celu miała najmniejszą wartość.

Normalizacja

Wartości wejść z tego samego zakresu np. [-1, 1] lub [0, 1].
W przypadku 24bitowych obrazów RGB (każdy kolor pixela kodowany na 8 bitach uint8) img=img/255.
Można też odjąć od każdej cechy jej wartość średnią i podzielić przez odchylenie standardowe

$$\frac{c_i - \bar{c}_i}{\sigma_{c_i}} \sim N(0, 1)$$

,
gdzie c_i oznacza itą cechę na podawaną na $i$te wejście sieci.

Normalizacja wsadowa (ang. Batch normalization)

Przydaje się do zapobiegania zanikaniu/eksplozji gradientu.
Zaproponowana w 2015 w https://arxiv.org/abs/1502.03167
W warstwach jest stosowana tuż przed funkcją aktywacji (ale można też po f.aktywacji).

Algorytm

Niech m oznacza liczbę elementów mini-batcha. Podczas treningu w kolejnych mini-batchach dane są normalizowane a następnie przeskalowywane i przesuwane zgodnie ze wzorami:

$\bar{x} = \frac{1}{m}\sum_{k=1}^{m} x_k,$ średnia w mini-batchu z wektorów danych $x_1, ..., x_m$

$\sigma^2 = \frac{1}{m}\sum_{k=1}^{m} (x_k - \bar{x})^2,$ wariancja w mini-batchu

$\hat{x}_i = \frac{x_i - \bar{x}}{\sqrt{\sigma^2 + \epsilon}},$ normalizacja

$y_i = a \otimes \hat{x}_i + b.$ przesunięcie i przeskalowanie

Elementy wektorów skali a i przesunięcia b dobierane są alg. gradientowymi podczas nauki sieci (tak jak wagi). ϵ jest bardzo małą wartością - zabezpieczenie przed ewentualnym dzieleniem przez 0. $\otimes$ - oznacza mnożenie wektorów po elementach (Iloczyn Hadamarda)

Batch normalization (normalizacja wsadowa)

Normalizacja wsadowa

Podczas uczenia dane są standaryzowane a następnie przeskalowywane i przesuwane. W trakcie uczenia nie ma problemu z obliczaniem średniej i wariancji. Gorzej podczas testowania. Średnią i wariancję można obliczyć z wykorzystaniem wszystkich danych, ale najczęściej po zakończeniu treningu wykorzystywane są oszacowania tych statystyki za pomocą średniej kroczących z fazy uczenia.

Normalizacja wsadowa w Kerasie - najprościej po aktywacji - normalizacja danych wejściowych do kolejnej warstwy

```
model = Sequential()
model.add(Dense(200, input_dim=2, activation='elu')
model.add(BatchNormalization())
model.add(Dense(100, input_dim=2, activation='elu')
model.add(BatchNormalization())
model.add(Dense(10, activation='softmax'))
...
```

Batch normalization (normalizacja wsadowa)

Normalizacja wsadowa

Podczas uczenia dane są standaryzowane a następnie przeskalowywane i przesuwane. W trakcie uczenia nie ma problemu z obliczaniem średniej i wariancji. Gorzej podczas testowania. Średnią i wariancję można obliczyć z wykorzystaniem wszystkich danych, ale najczęściej po zakończeniu treningu wykorzystywane są oszacowania tych statystyki za pomocą średniej kroczących z fazy uczenia.

Normalizacja wsadowa w Kerasie - najprościej po aktywacji, ale łatwo też zastosować ją przed f. aktywacji - normalizacja wartości argumentów funkcji aktywacji

```
model = Sequential()
model.add(Dense(200, input_dim=2)
model.add(BatchNormalization())
model.add(Activation,'elu')
model.add(Dense(100, input_dim=2)
model.add(BatchNormalization())
model.add(Activation,'elu')
model.add(Dense(10, activation='softmax'))
...
```

Autorzy metody w https://arxiv.org/abs/1502.03167 zalecali stosowanie

Batch normalization (normalizacja wsadowa)

Normalizacja wsadowa

Pojedyncza epoka trwa dłużej, ale całkowity czas uczenia sieci na ogół jest krótsze (szybsze zbieganie f.celu do minimum). Różnica jest szczególnie duża dla głębokich sieci, natomiast dla małych sieci (np. 2 warstwy ukryte) nie będzie miała istotnego wpływu.

Normalizacja wsadowa

Często po zastosowaniu normalizacji wsadowej na początku sieci neuronowej nie trzeba stosować standaryzować zestawu danych uczących

Augmentacja danych

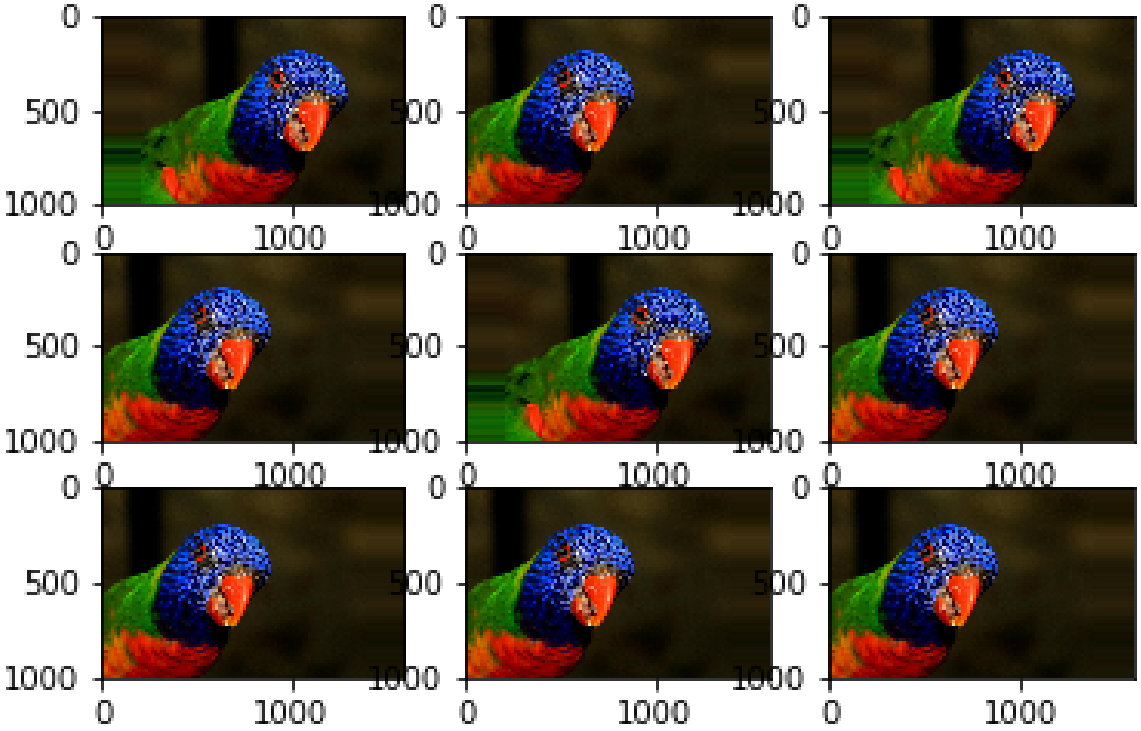
Technika szczególnie przydatna przy małych zbiorach danych.
Gdy jest niewielka liczba danych, to sieć neuronowa raczej nie stworzy dobrych uogólnień, które sprawdzą się na nowych danych.
Augmentacja danych polega na generowaniu nowych danych poprzez wykonywanie odpowiednich losowych przekształceń na dostępnych danych. Bardzo łatwo można zastosować tą technikę do obrazów.

Augmentacja zbioru obrazów

Zwiększanie zbioru uczącego poprzez wykonywanie różnorodnych przekształceń obrazów np. obrotów, przesunięć, powiększeń lub oddaleń, przycinań, operacji na kolorach...
Za każdym razie parametry przekształceń są dobierane losowo (losowy kąt obrotu, losowe powiększenie itp.)

W Kerasie do augmentacji obrazów udostępniono `ImageDataGenerator`

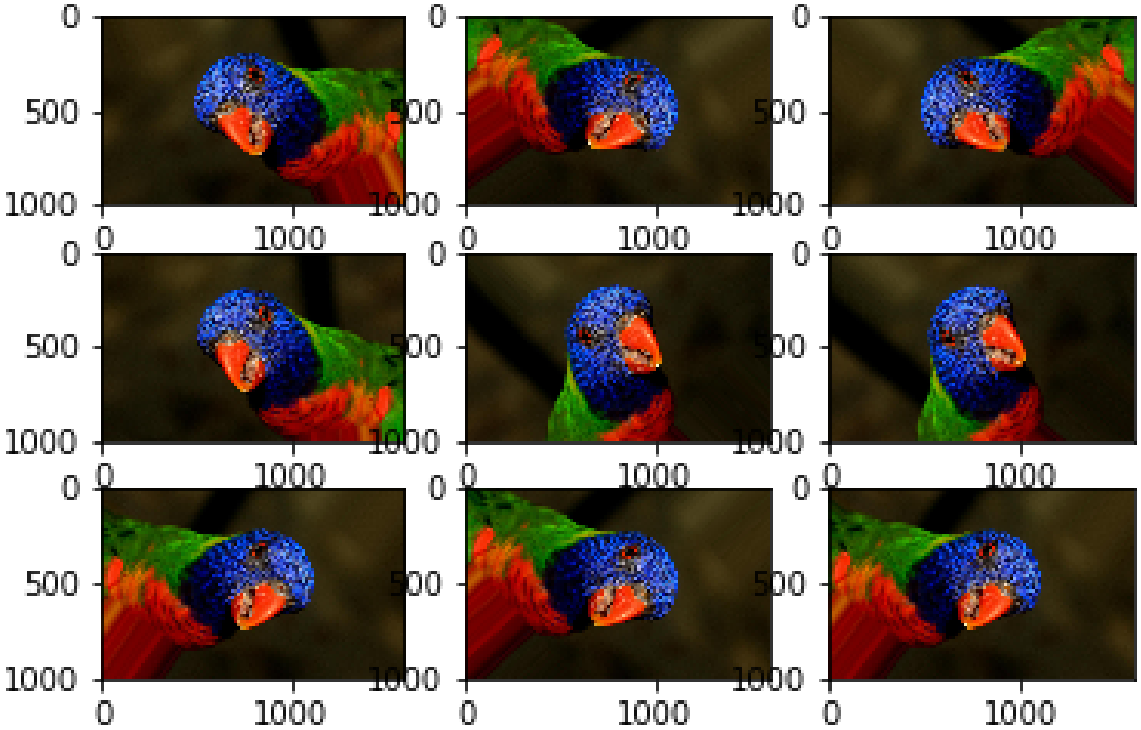
Augmentacja danych



Rys. https://www.flickr.com/photos/thenovys/3854468621/Feathered Friend by AndYaDontStop

```
generator = ImageDataGenerator(width_shift_range=[-200,200])
it = generator.flow(images, batch_size=1)
```

Augmentacja danych




```
generator = ImageDataGenerator(horizontal_flip=True, rotation_range=60)
it = generator.flow(images, batch_size=1)
```

Augmentacja danych https://keras.io/api/preprocessing/image/

Olbrzymia liczba kombinacji !!!

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False, #Set input mean to 0 over the dataset, feature-wise.
    samplewise_center=False,  #Set each sample mean to 0
    featurewise_std_normalization=False,  #Divide inputs by std of the dataset, feat
    samplewise_std_normalization=False,   #Divide each input by its std.
    zca_whitening=False,  #epsilon for ZCA whitening. Default is 1e-6
    zca_epsilon=1e-06,   #Apply ZCA whitening
    rotation_range=0, #Degree range for random rotations.
    width_shift_range=0.0,  #Range for picking a brightness shift value from.
    height_shift_range=0.0,
    brightness_range=None,  #Range for picking a brightness shift value from
    zoom_range=0.0,   #Range for random zoom
    shear_range=0.0,  #Shear Intensity
    channel_shift_range=0.0, #Range for random channel shifts
    fill_mode="nearest", #Points outside the boundaries of the input are filled acc
    cval=0.0,
    horizontal_flip=False, #Randomly flip inputs horizontally
    vertical_flip=False,  #Randomly flip inputs vertically
    rescale=None, #rescaling factor
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    dtype=None,
)
```

Augmentacja danych https://keras.io/examples/vision/image_classification_from_scratch/

Można augmentację dodać w Kerasie tak samo jak warstwy sieci:

```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(0.6),
    ]
)
```

Augmentacja danych

Więcej informacji i przykładów:
https://keras.io/api/preprocessing/image/
https://machinelearningmastery.com/
how-to-configure-image-data-augmentation-when-training-deep-learning-neura
https://keras.io/examples/vision/image_classification_from_scratch/

Inicjalizacja wag

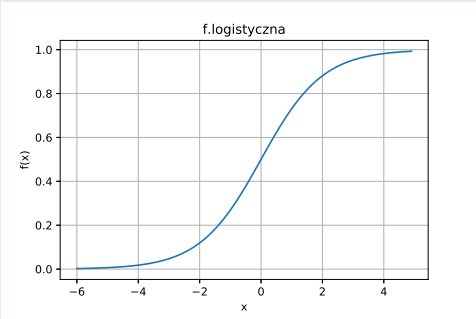
Początkowe wartości wag mają istotny wpływ na problem niestabilnego gradientu (zanikającego/eksplodującego)

Wagi w poszczególnych warstwach mogą być generowane za pomocą generatora liczb pseudolosowych o rozkładzie normalnym $N(0, \sigma)$ lub jednostajnym na przedziale $[r, r]$.
Dla każdej warstwy σ i r są na ogół różne i zależą od liczby wejść i wyjść warstwy.
Inicjalizacja Glorota określa sposód doboru σ i r dla funkcji aktywacji tanh, logistycznej, softmax i liniowej (brak f. aktywacji).
Inicjalizacja He podaje sposób doboru σ i r dla funkcji aktywacji ReLU i jej odmian.
https://mmuratarat.github.io/2019-02-25/xavier-glorot-he-weight-init

Inicjalizacja wag

Początkowe wartości wag mają istotny wpływ na problem niestabilnego gradientu (zanikającego/eksplodującego)

https://mmuratarat.github.io/2019-02-25/xavier-glorot-he-weight-init W artykule Glorot, Xavier, Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010 autorzy pokazali, że dla sigmoidalnych funkcji aktywacji stosowana wówczas często incjalizacja wag liczbami pseudolosowymi o rozkładzie normalnym $N(0, 1)$ powoduje, że na wyjściu każdej warstwy wariancja jest większa niż na jej wejściu aż do wystąpienia nasycenie sigmoidalnych f. aktywacji w wyższych warstwach.



Dla argumentów f.aktywacji spoza [-4,4] bardzo małe zmiany wartości i mała pochodna - tzw. nasycenie neuronu.

Dobrze, żeby wariancja wyjść w kolejnych warstwach była równa wariancji wejść tych warstw i gradienty miały taką samą wartość, podczas propagacji w odwrotnym warunku. Nie da się spełnić tych dwóch warunki jednocześnie jeżeli liczba wejść i wyjść warstwy jest różna, ale można parametr σ lub wartość r dobrać na podstawie średniej liczbę wejść i wyjść warstwy. Wagi mogą mieć rozkład normalny $N(0, \sigma)$ lub jednostajny na przedziale $[-r, r]$.

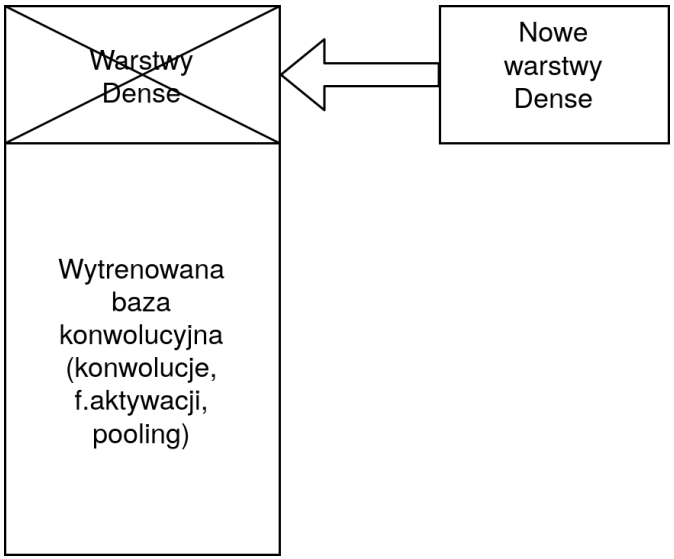
Kodowanie i normalizacja danych

Zalecana jest normalizacja lub standaryzacja danych numerycznych, które są podawane na wejścia sieci!
(Dygresja - podobny zabieg można stosować nie tylko do danych wejściowych, ale również między warstwami).

W przypadku obrazów w odcieniach szarości można wykonać skalowanie (dzielenie przez największą możliwą wartość piksela), tak aby otrzymać macierz o wartościach odcieni pikseli z przedziału $[0, 1]$.
Kodowanie danych za pomocą gorącej jedynki jest stosowane, gdy dana cecha może należeć do jednej z kilku kategorii. Załóżmy, że są 4 klasy i dana należy do III klasy - wówczas kodowanie za pomocą wektora $[0, 0, 1, 0]^T$.

Uczenie transferowe (ang. Transfer learning)

Transfer learning polega na wykorzystaniu uprzednio wytrenowanej sieci do nowego zagadnienia.



Usuwane są dotychczasowe warstwy gęste, które są zastępowane nowymi. Te nowe warstwy są trenowane do rozwiązywania nowego zagadnienia. Najczęściej zamraża się wagi dotychczasowych warstw konwolucyjnych (zamrażane - nie są trenowane).
W celu polepszenia działania warstwy te mogą być później dostrajane.
Do nowego zagadnienia są wykorzystywane cechy ekstrachowane przez uprzednio wytrenowane warstwy konwolucyjne.

Uczenie transferowe (ang. Transfer learning)

Zalety transfer learning:

- Można wykorzystać bazę ogromnej sieci konwolucyjnej wytrenowanej na olbrzymich zbiorach danych (np. ImageNet - kilkanaście milionów obrazów podzielonych na 22tys. kategorii).
- Przyspiesza uczenie i zmniejsza zapotrzebowania na moc obliczeniową potrzebną do treningu.
- Umożliwia trening na mniejszych zbiorach danych (uczenie od zera dużej sieci na małym zbiorze danych prawdopodobnie skończyłoby się tragicznym przeuczeniem).

Ekstrakcja cech - reprezentacje wyuczone przez bazę konwolucyjną są na ogół dość ogólne i często dobrze mogą być wykorzystane do nowych zastosowań.
Baza konwolucyjna - część sieci z warstwami konwolucyjnymi bez ostatnich warstw gęstych.
W Kerasie można od razu załadować bez warstw gęstych gotowe sieci ze slajdu 24.

```
from Keras.applications import VGG16
conv_base = VGG16 (weights='imagenet', include_top=False, input_shape=(150,150,3))
```

Zamrożenie wag:

```
conv_base.trainable=False
```

Uczenie transferowe (ang. Transfer learning)

Na ImageNet zostały wytrenowane m.in. następujące sieci:

- VGG16 oraz VGG19,
- Xception V3,
- ResNet50,
- MobileNet.

Można próbować wykorzystać ich bazy do różnorodnych zastosowań związanych z przetwarzaniem obrazów.

UWAGA

Jeżeli nowy zbiór danych na którym ma pracować nowa sieć różni się znacznie od zbioru uczącego, na którym została wytrenowana baza konwolucyjna, to z bazy konwolucyjnej można pobrać tylko pierwsze warstwy (kolejne zawierającą coraz bardziej abstrakcyjne cechy przydatne do poprzedniego zastosowania, a niekoniecznie odpowiednie do aktualnego).

DOSTRAJANIE

Dostrajanie polega na odmrożeniu i dostrojeniu kilku górnych warstw zamrożonej bazy (później można niższe warstwy) i trenowaniu ich wraz z dodanymi nowymi warstwami gęstymi (po ich uprzednim wstępnym wytrenowaniu).
Nie można od razu odmrozić wszystkich warstw, ponieważ końcowe nowe warstwy są kompletnie niewyuczone jeszcze zadania - propagacja ogromnych gradientów z nich błyskawicznie zniszczyłaby wagi uprzednio wytrenowanej bazy !!!

Procedura transfer learning z dostrajaniem:

- Dodanie nowych warstw gęstych do bazy konwolucyjnej z uprzednio wytrenowanej starej sieci
- Zamrożenie bazy
- Wytrenowanie nowych warstw gęstych
- Odmrożenie wybranych (na ogół końcowych) warstw bazy
- Trenowanie wszystkich odmrożonych warstw (konwolucyjnych i gęstych)
- Ewntualne odmrożenie i trening kolejnych warstw (można odmrozić całą bazę)

Uczenie transferowe (ang. Transfer learning)

Transfer learning jest bardzo przydatny w przypadku małego zbioru danych, jeżeli dysponuje się siecią wytrenowaną do podobnego zadania (uczenie dużej sieci na małym zbiorze danych prawdopodobnie doprowadziłoby do tragicznego przeuczenia i sieć ta nie miałaby zdolności generalizacji).

```
model_wczesniejszy = keras.models.load_model ("model_do_czegos_podobnego.h5")
nowy_model = keras.models.Sequential (model_wczesniejszy.layers[:-1])  #!!!!!!!!!
nowy_model.add (keras.layer.Dense(10, activation="softmax")
```

Podczas uczenia będą zmieniane wagi w obydwu modelach (kopiowanie obiektów w języku Python) - można tego uniknąć poprzez klonowanie

```
nowy_model = keras.models.clone_model(model_wczesniejszy)
nowy_model.set_weights (model_wczesniejszy.get_weights())
```

Uczenie transferowe (ang. Transfer learning)

Włączanie i wyłączanie uczenia wybranych warstw jest trywialne:

```
for layer in nasz_model.layers[:-1]:
  layer.trainable=False
```

Uczenie transferowe (ang. Transfer learning)

Alternatywne podejście programistyczne, które może ułatwić pracę z niektórymi prostymi bibliotekami:

Zamiast tworzyć jedną sieć można ewentualnie utworzyć dwie osobne sieci. Jedna z nich będzie po prostu bazą konwolucyjną, a druga będzie zawierała same warstwy gęste (druga może być wielowarstwowym perceptronem). Na wejścia drugiej sieci będą podawane wyjścia ostatniej warstwy pierwszej sieci czyli dane przepuszczone przez bazę konwolucyjną.

Uczenie transferowe i dostępne modele w Kerasie

https://keras.io/api/applications/

Połączenia oddalonych warstw w sieciach konwolucyjnych

W sieciach konwolucyjnych mogą istnieć połączenia między odległymi warstwami, a nie tylko między kolejnymi warstwami. Takie połączenia zmniejszają problem zanikającego gradientu i dostarczają więcej informacji do dalszych warstw. Połączenia tego typu występują np. w sieci ResNet50. Więcej informacji o tworzeniu takich połączeń zostanie przedstawionych na innym wykładzie.

```
from keras.applications.resnet50 import ResNet50
from keras.utils import plot_model
model = ResNet50()
plot_model (ResNet50(),to_file='ResNet50.png')
```

Sieć ResNet50 jest bardzo rozbudowana - rysunek jej architektury nie zmieści się na slajdach.

Połączenia warstw w sieciach konwolucyjnych

Najczęściej połączone są tylko kolejne warstwy

Rozgałęzienia i ich zalety przedstawiono na kolejnych slajdach

Od kilku lat coraz częściej są stosowane rozgałęzienia

Rozgałęzienia są przydatne zwłaszcza w sieciach mających wiele warstw

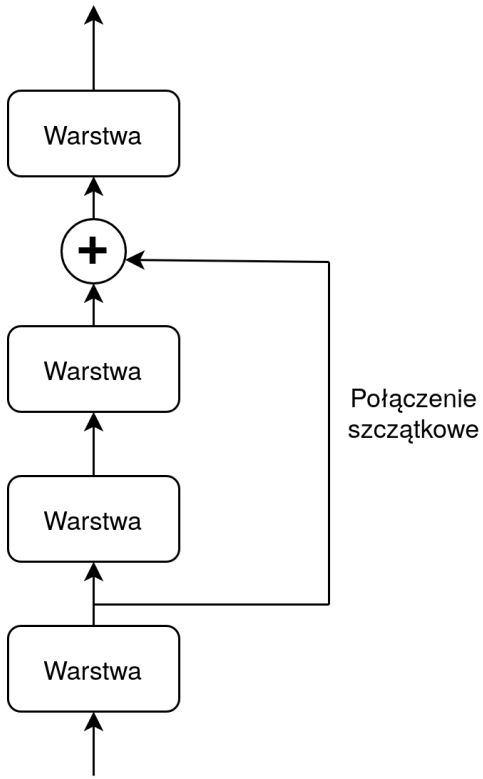
Połączenia szczątkowe (ang. residual connections)

Połączenia szczątkowe

Zostały zastosowane m.in. w sieci Xception w 2015r. i stają się coraz popularniejsze

Zmniejszają problem zaniku gradientu (ułatwiają uczenie sieci mających dużo warstw - dodanie połączeń do sieci mających kilkanaście lub więcej warstw prawdopodobnie polepszy działanie sieci)

Pozwalają dostarczyć więcej szczegółów do wyższych warstw (zapobiegają zanikaniu detali)

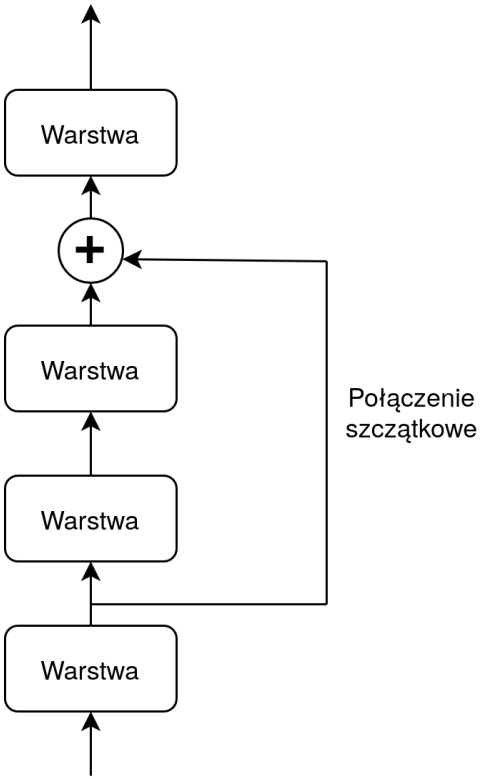


Połączenia szczątkowe (ang. residual connections)

Sumowanie wyjść warstw

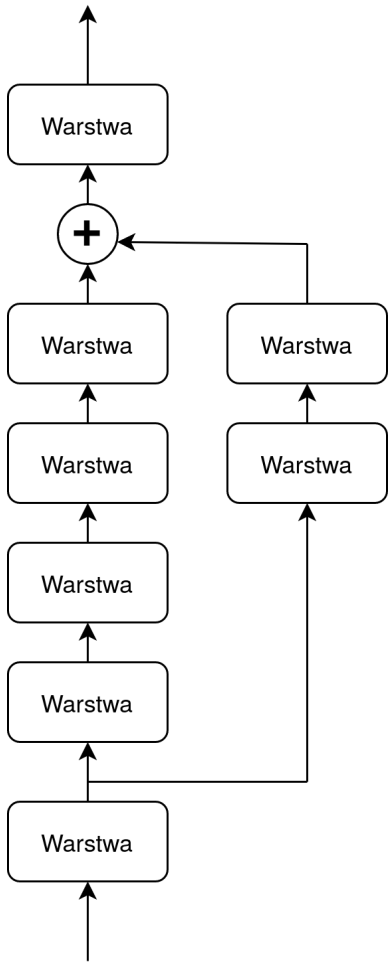
Bez problemu można dodawać tensory o tych samych wymiarach. Sumowane wyjścia warstw dołączonych do sumatora muszą mieć takie same wymiary. Padding zerami zapobiega zmniejszeniu rozmiaru danych przepuszczanych przez warstwę konwolucyjną. Można go zastosować w warstwach między sumatorem a rozgałęzieniem.

W Kerasie połączenia szczątkowych można zaimplementować za pomocą `functionalAPI`.



Połączenia między dalszymi warstwami

Można również tworzyć bardziej skomplikowane struktury z połączeniami między dalszymi warstwami. Dodanie po prawej stronie warstw ułatwia uzyskanie takich samych rozmiarów wyjść (sumowane tensory muszą mieć takie same rozmiary). Po lewej i prawej stronie można stosować np. MaxPooling. Uwaga - pooling może pomniejszać znacznie więcej razy niż na przykładzie w poprzedniej prezentacji.

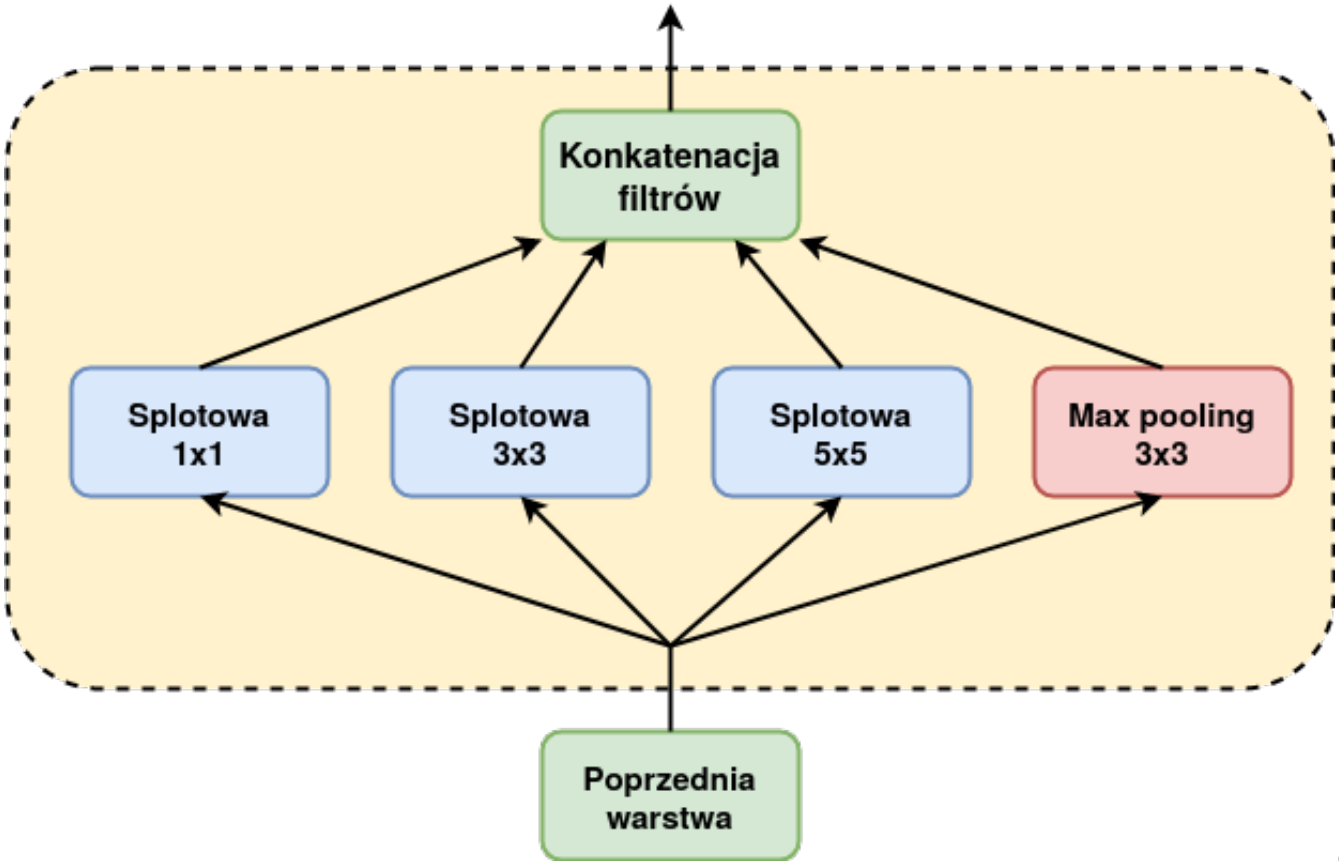


.

Rozgałęzienia

Oczywiście można utworzyć znacznie więcej odnóg. Na przykład rozbudowane rozgałęzienia występują w sieciach Inception v1, Inception v2, Inception v3 i Inception v4. O sieciach tych można przeczytać w artykule
https://arxiv.org/pdf/1409.4842v1.pdf

Naiwny Inception

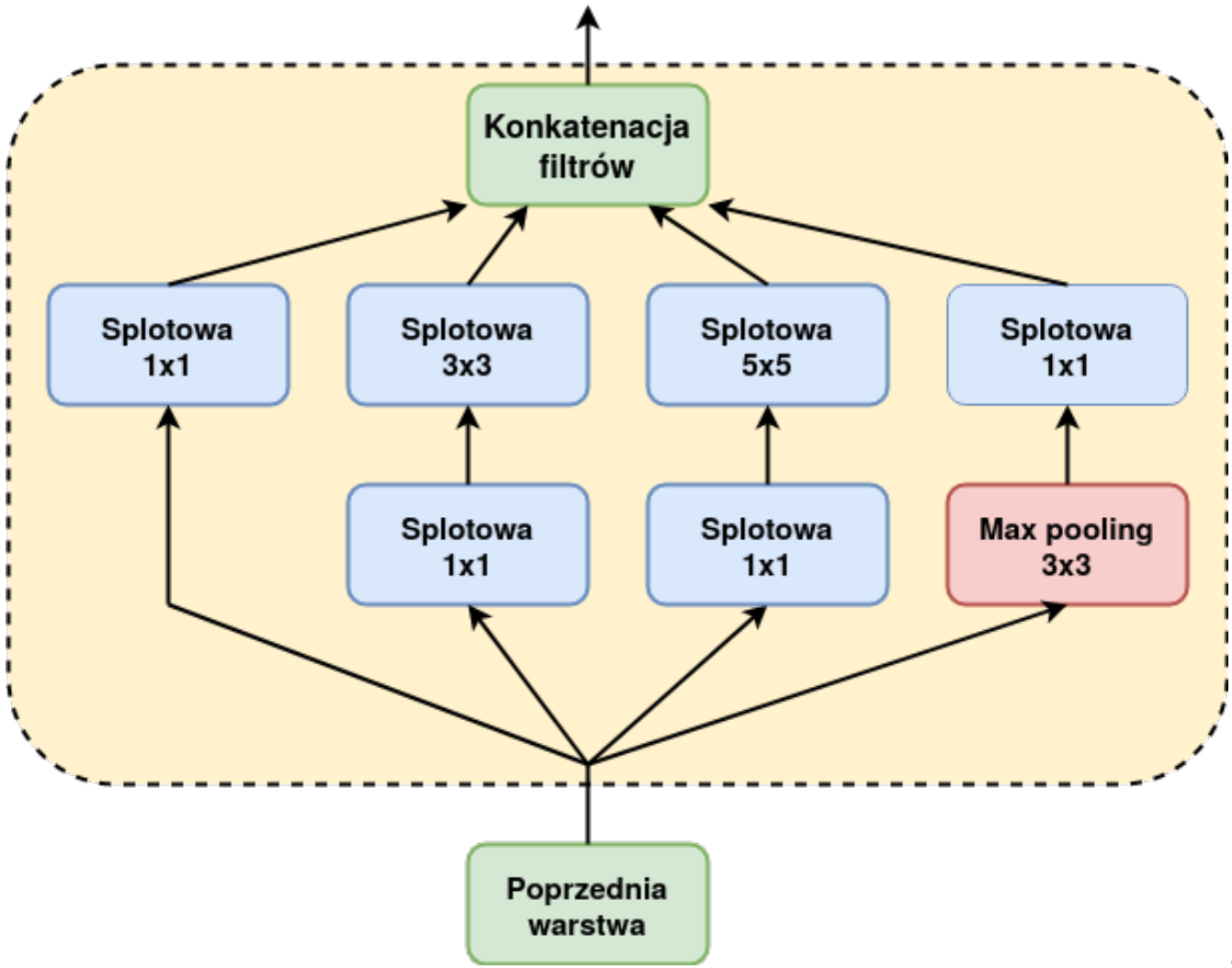


.

Goingdeeperwithconvolutionshttps://arxiv.org/pdf/1409.4842v1.pdf

Inception

Występuja np. w GoogLeNet (rok 2014)



.

Goingdeeperwithconvolutionshttps://arxiv.org/pdf/1409.4842v1.pdf

EfficientNet (rok 2019) - łatwe skalowanie sieci (głębokość, szerokość, rozdzielczość

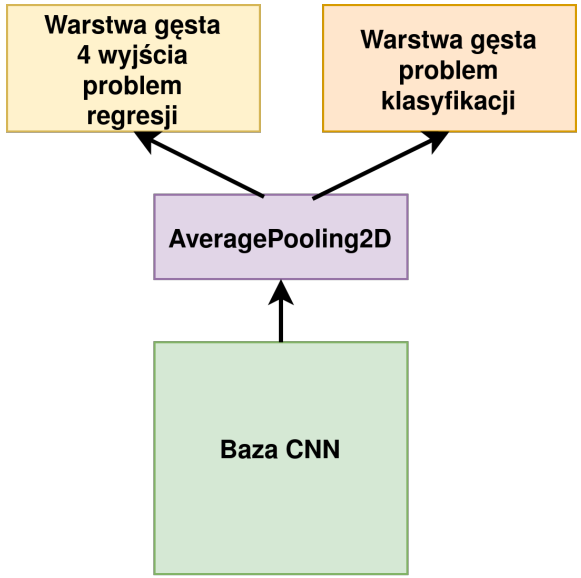
https://arxiv.org/pdf/1905.11946

Lokalizacja przedmiotów na obrazie

Jak można lokalizować przedmioty na obrazie - wyznaczać ramki otaczające przedmioty ?

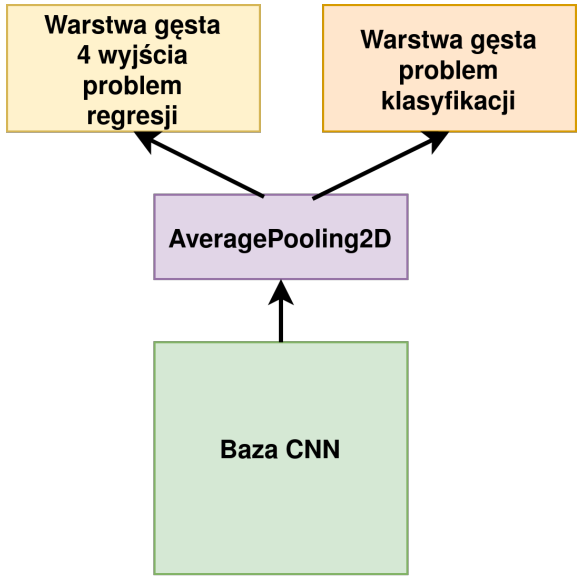
Załóżmy, że tylko jedna ramka na jeden obraz

Można dodać do sieci do klasyfikacji cztery wyjścia:
dwa wyjścia wskazują współrzędne środka przedmiotu, dwa pozostałe wskazują szerokość i wysokość ramki
Te cztery wartości przeważnie znormalizowane - przedział [0,1]
Zadanie regresji



Lokalizacja przedmiotów na obrazie - pojedyncza ramka

Różne funkcje celu - problem regresji oraz klasyfikacji
Oczywiście nie muszą być pojedyncze warstwy gęste - może być ich więcej



Musi być odpowiedni zbiór danych zawierający współrzędne i rozmiary ramek !!!

Lokalizacja przedmiotów na obrazie - pojedyncza ramka

Musi być odpowiedni zbiór danych zawierający współrzędne i rozmiary ramek !!!

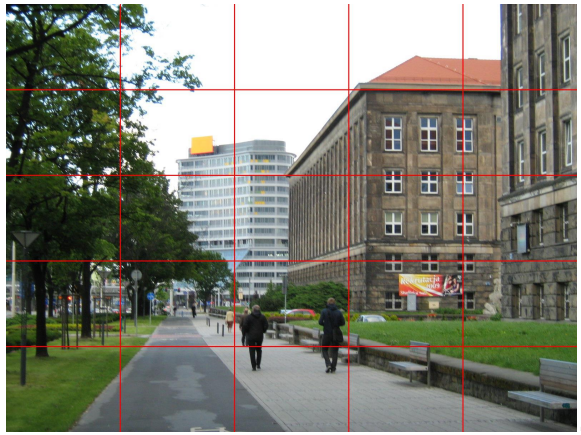
```
from tensorflow.keras.applications.xception import Xception
#Extreme Inception w 2016 Francois Chollet
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras import Model

liczba_klas=200          #np. rozpoznajemy 200 obiektów
baza = Xception(include_top=False, weights="imagenet")
sred = GlobalAveragePooling2D ()(baza.output)
lokalizacja = Dense (4) (sred) #dwie wsporzedne srodka ramki + wysokosc i szerokosc
klasyfikacja = Dense( liczba_klas, activation="softmax")(sred)
siec_lokalizacja_klasyfikacja = Model (inputs=baza.input,
     outputs=[lokalizacja, klasyfikacja])
model.compile (loss=["mse","sparse_categorical_crossentripy"],
     loss_weights=[0.25, 0.75])
```

Detekcja (wykrywanie) przedmiotów na obrazie

detekcja = lokalizacja + klasyfikacja
Na początku "przesuwano" po całym obrazie sieć CNN wytrenowaną do klasyfikacji i lokalizacji pojedynczych obiektów
Na wejścia sieci podawane były kolejne fragmenty obrazu.

Wady takiego podejścia:

- duża liczba obliczeń,
- obiekty mają różne rozmiary i mogą nie mieścić się w pojedynczym obszarze - konieczne jest "przesuwanie" kilku sieci CNN o różnych rozmiarach
- ten sam obiekt może być wykrywany wielokrotnie, więc potrzebny jest jeszcze dodatkowy algorytm złączania



Detekcja (wykrywanie) przedmiotów na obrazie

Algorytm złączania non-max supression

- załóżmy, że wykrywamy samochody - do sieci dodawane jest wyjście określające obiektowość ramki - wartość z przedziału [0,1] (f. aktywacji logistyczna)
- usuwane są ramki nie obejmujące samochodu, dla których wartość obiektowości jest mała (mniejsza od założonego progu)
- znajdowana jest ramka o największej obiektowości i usuwane są wszystkie ramki nakładające się na nią w dużym stopniu

Dzisiaj nie trzeba już tak robić !

Powyższy algorytm z sieciami CNN lokalizującymi pojedyncze obiekty wymagał dużej liczby obliczeń wynikających z przesuwania sieci splotowych !!!

Detekcja (wykrywanie) przedmiotów na obrazie - specjalne sieci znacznie wydajniej

W 2014 zaproponowano R-CNN https://arxiv.org/pdf/1311.2524.pdf - specjalny algorytm proponuje 2000 regionów, które po przeskalowaniu sa podawane na sieci CNN działajace jako ekstraktory cech. Cechy te są podawane na końcu na maszyny wektorów nośnych (SVMy - ang. support vector machines). Później autorzy zaproponowali Fast R-CNN a następnie Faster R-CNN. R-CNN należy do grupy sieci mających angielską nazwę region proposal networks.
W 2015 zaproponowano w pełni połączone sieci splotowe FCNN (ang. Fully Convolutional Neural Network) https://arxiv.org/abs/1411.4038 , w których zastąpiono warstwy gęste warstwami splotowymi. Na wejścia FCNN podaje się cały obraz bez konieczności przesuwania sieci po kolejnych fragmentach obrazu. Sieć automatycznie oblicza wyniki dla kolejnych rozłącznych fragmentów. Bardziej szczegółowe wyjaśnienie działania jest w [AGeron].
W 2015 stworzono również znacznie szybsza sieć YOLO v1 (ang. You Only Look Once).
W kolejnych latach powstały udoskonalone YOLO v2, YOLO v3,...,YOLOv9.
W OpenCV jest Deep Neural Network module (DNN), który zawiera YOLO v3.
W 2017 Facebook AI Research (FAIR) stworzył RetinaNet, która działała znacznie szybciej od Faster R-CNN.
YOLO v3 jest szybszy od RetinaNet, ale mniej dokładny.
Gotowe YOLO v8 wygodnie z Ultralytics.

Bardzo stare klasyczne podejścia do wykrywania obiektów

HOG (Histogram of Oriented Gradients)
https://learnopencv.com/histogram-of-oriented-gradients/

SIFT (Scale-invariant feature transform)
https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
HOG+SVM

Podejscią z propozycja regionów (ang. Region Proposal)

R-CNN https://arxiv.org/pdf/1311.2524.pdf 2014 Ross Girshick et. all. UC Berkeley

Fast R-CNN https://arxiv.org/pdf/1504.08083.pdf 2015 Ross Girshick Microsoft Research

Faster R-CNN S. Ren, K. He, Ross Girshick i J. Sun https://arxiv.org/abs/1506.01497 2016

Są też **inne** mniej słynne i populrane RPNs (Region Proposal Networks) np. cascade R-CNN.

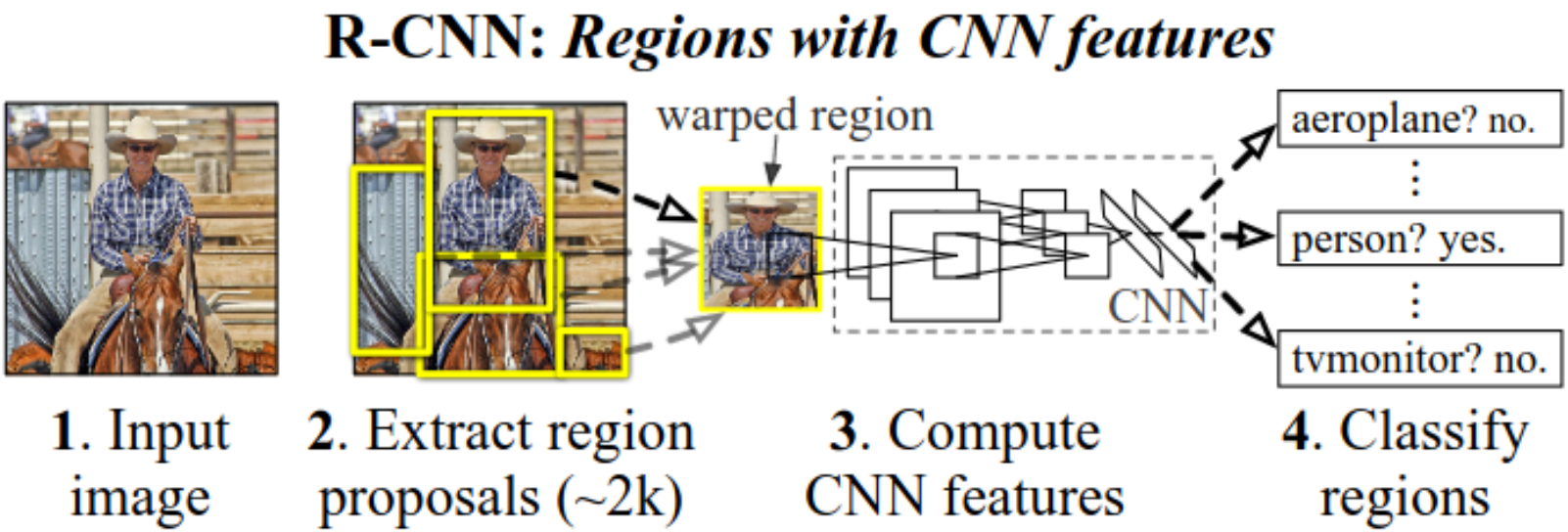
Detekcja (wykrywanie) przedmiotów na obrazie - specjalne sieci znacznie wydajniej

W 2014 zaproponowano R-CNN https://arxiv.org/pdf/1311.2524.pdf - specjalny algorytm o nazwie selective search proponuje 2000 regionów, które po przeskalowaniu sa podawane na sieci CNN działajace jako ekstraktory cech. Cechy te są podawane na końcu na maszyny wektorów nośnych (SVMy - ang. support vector machines). Z abstraktu: "Our approach combines two key insights: (1) one can apply high-capacity convolutional neural net- works (CNNs) to bottom-up region proposals in order to localize and segment objects and (2) when labeled training data is scarce, supervised pre-training for an auxiliary task, followed by domain-specific fine-tuning, yields a significant performance boost. Since we combine region proposals with CNNs, we call our method R-CNN: Regions with CNN features. We also compare R-CNN to OverFeat, a recently proposed sliding-window detector based on a similar CNN architecture. We find that R-CNN outperforms OverFeat by a large margin on the 200-class ILSVRC2013 detection dataset"

Selective search w R-CNN

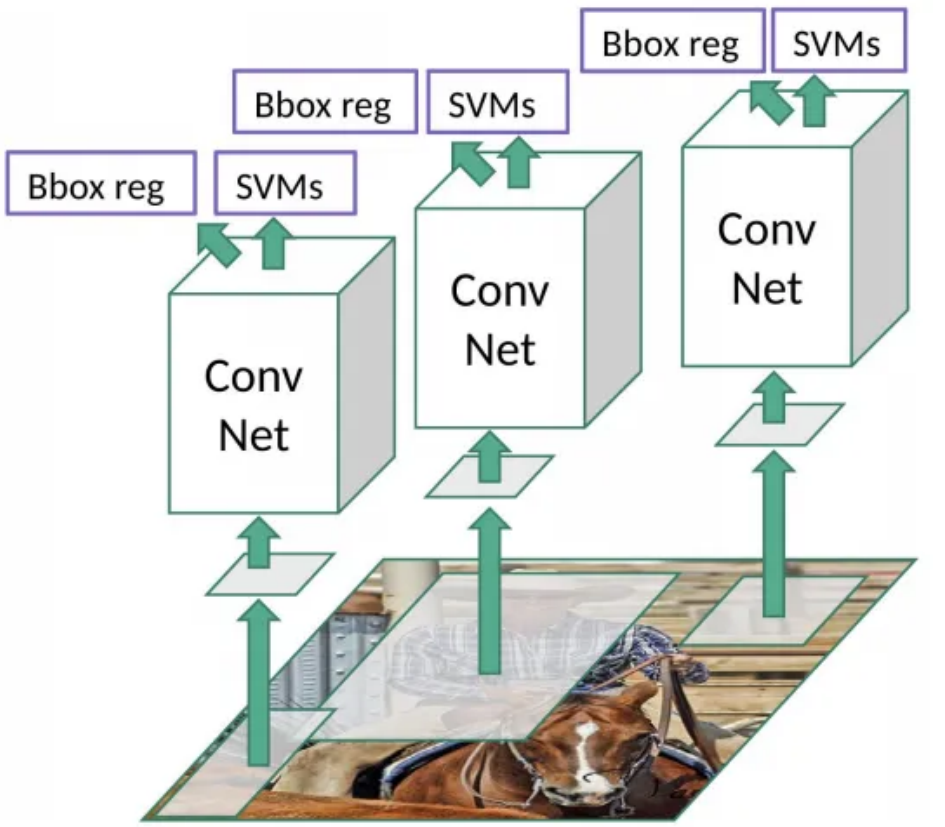
Selective Search: 1. Wykonaj wstępną segmentację "sub-segmentation", generowanie wielu kandydatów na region 2. Użycie zachłannego algorytmu do rekurencyjnego złącza podobnych regionów w większe 3. Następnie te regiony są wykorzystywane do tworzenia ostatecznych propozycji regionów (final candidate region proposals)

R-CNN https://arxiv.org/pdf/1311.2524.pdf



"Our system
(1) takes an input image,
(2) extracts around 2000 bottom-up region proposals,
(3) computes features for each proposal using a large convolutional neural network (CNN),
and then (4) classifies each region using class-specific linear SVMs."
Pretrenowana na dużym zbiorze (ILSVRC2012 classification) sieć CNN generuje 4096-elementowy wektor cech dla każdego proponowanego regionu (ang. feature) Jako CNN (backbone) można zastosować np. VGG, ResNet, Inception itp.

Detekcja obiektów - R-CNN



"**Domain-specific fine-tuning.** To adapt our CNN to the new task (detection) and the new domain (warped proposal windows), we continue stochastic gradient descent (SGD) training of the CNN parameters using only warped region proposals."

"Based on the error analysis, we implemented a sim- ple method to reduce localization errors. Inspired by the bounding-box regression employed in DPM [17], we train a linear regression model to predict a new detection window given the pool5 features for a selective search region pro- posal. Full details are given in Appendix C"

Istotne wady R-CNN

duża liczba obliczeń - 2000 regionów dla jednego obrazka
brak możliwości szkolenia na etapie selective search

Fast R-CNN https://arxiv.org/pdf/1504.08083.pdf 2015 Ross Girshick Microsoft Research

Fragment streszczenia (w abstract): "This paper proposes a Fast Region-based Convolutional Network method (Fast R-CNN) for object detection. Fast R-CNN builds on previous work to efficiently classify ob- ject proposals using deep convolutional networks. Com- pared to previous work, Fast R-CNN employs several in- novations to improve training and testing speed while also increasing detection accuracy. Fast R-CNN trains the very deep VGG16 network 9× faster than R-CNN, is 213× faster at test-time, and achieves a higher mAP on PASCAL VOC 2012."

Fast R-CNN https://arxiv.org/pdf/1504.08083.pdf

UWAGA - dla każdego RoI szary fragment

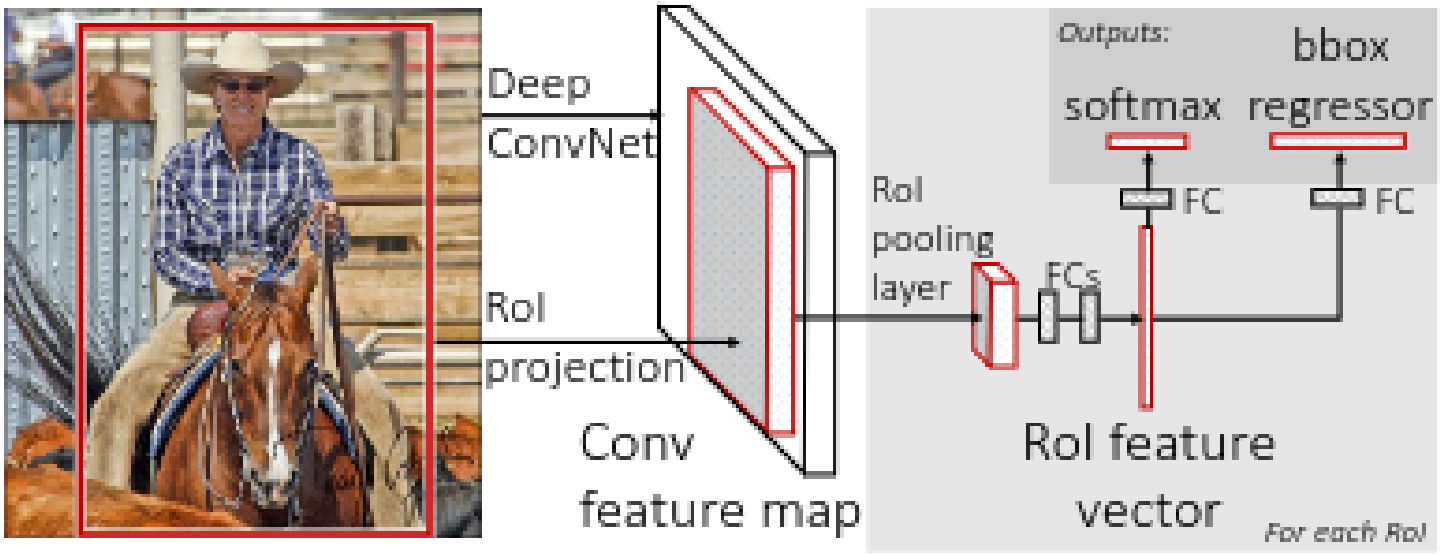


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

Fast R-CNN https://arxiv.org/pdf/1504.08083.pdf

Zamiast dostarczać propozycje regionów do CNN, przekazujemy obraz wejściowy do CNN, aby wygenerować mapę cech. Na podstawie mapy cech identyfikujemy regiony propozycji (też za pomocą selective search) i wtłaczamy je w kwadraty, a za pomocą warstwy RoI pulling przekształcamy je w stały rozmiar, aby można je było wprowadzić do w pełni połączonej warstwy. Z wektora cech RoI używamy warstwy softmax do przewidywania klasy proponowanego regionu, a także wartości przesunięcia dla ramki ograniczającej.

Zalety Fast R-CNN https://arxiv.org/pdf/1504.08083.pdf

1. Znacznie szybsze niż R-CNN - podajemy jeden cały obraz a nie aż 2000 proponowanych regionów

2 Lepsza jakość detekcji (mAP) niż R-CNN

3. Trening składa się z jednego etapu, w którym wykorzystywana jest specjalna funkcja straty (multi-task loss)

4. W wyniku treningu dostrajane sa wszystkie warstwy sieci

Fast R-CNN https://arxiv.org/pdf/1504.08083.pdf

Wąskim gardłem jest znajdowanie ROI z mapy cech

W Fast R-CNN do ekstrakcji cech wykorzystywana jest wstępnie trenowana sieć na dużym zbiorze np. VGG16

Faster R-CNN - podobnie jak w przypadku Fast R-CNN, obraz jest dostarczany jako dane wejściowe do sieci konwolucyjnej, która robi ekstrakcję - generuje mapę cech. Zamiast stosowania algorytmu wyszukiwania selektywnego (ang. selective search) na mapie cech do, do przewidywania propozycji regionów wykorzystywana jest oddzielna sieć. Propozycje przewidywanego obszaru są następnie przekształcane przy użyciu warstwy puli RoI, która jest następnie wykorzystywana do klasyfikowania obrazu w proponowanym regionie i przewidywania wartości przesunięcia dla ramek ograniczających.

YOLO 2016 - You Look Only Once

https://arxiv.org/pdf/1506.02640v5.pdf

YOLO jest o rząd szybsze, ale może mieć problemy z małymi obiektami np. stadem ptaków.
Zasada działania YOLO znacznie różni się od podejść opartych na propozycjach regionach, które widzieliśmy wcześniej. W YOLO pojedyncza sieć splotowa przewiduje ramki ograniczające (ang. bounding boxs) i prawdopodobieństwa klas dla tych ramek. W punkcie 2. https://arxiv.org/pdf/1506.02640v5.pdf wyjąsniona zasada działania.
Różnice między poszczególnymi wersjami YOLO są wyjasnione np. w
https://www.mdpi.com/2504-4990/5/4/83 i
https://www.v7labs.com/blog/yolo-object-detection
W dniu 8.06.204 najnowsze jest YOLOv10.

POSENET

https://www.tensorflow.org/lite/examples/pose_estimation/overview

Object detection vs semantic segmentation vs instance segmentation

https://blog.roboflow.com/
difference-semantic-segmentation-instance-segmentation/

UNET - segmentacja

https://towardsdatascience.com/
understanding-semantic-segmentation-with-unet-6be4f42d4b47

Dobór funkcji celu (przypomnienie)

Tabela: Dobór funkcji celu (straty) oraz funkcji aktywacji do typowych zadań

Zadanie	Funkcja aktywacji w ostatniej warstwie	Funkcja celu (straty)	Funkcja celu w Keras
Regresja	liniowa	Błąd średniokwadratowy MSE	mse
Klasyfikacja binarna	logistyczna (sigmoid)	Binarna entropia krzyżowa	binary_crossentropy
Wieloklasowa klasyfikacja (binarna)	softmax	Kategorialna entropia krzyżowa	categorical_crossentropy
Wieloklasowa klasyfikacja (wieloetykietowa, wielo-wyjściowa)	logistyczna (sigmoid)	Binarna entropia krzyżowa	binary_crossentropy

Klasyfikacja binarna - są tylko dwie klasy
Wieloklasowa klasyfikacja (jednoetykietowa, binarna) - klasyfikacja danych do wielu klas (dana jednocześnie może należeć tylko do jednej klasy - mieć jedną etykietę)
Wieloklasowa klasyfikacja (wielowyjściowa, wieloetykietowa) - dana może należeć jednocześnie do kilku klas (mieć przypisanych kilka etykiet).

Funkcja straty zwana też f.celu (ang. loss function, objective function, cost function)

Dla nietypowych zadań istnieje możliwość tworzenia własnych f. celu w Kerasie. Funkcje te można zaprojektować uwzględniając specyfikę zagadnienia i pożądane cele optymalizacji. Funkcja powinna być zaprojektowana tak, aby dla lepszych sieci miała mniejszą wartość niż dla kiepsko działających sieci. Funkcja musi być różniczkowalna dla alg. gradientowych.

Funkcje celu (straty)

Informacje podane w tab. ze slajdu 121 wykorzystuje się do tworzenia wielowarstwowych perceptronów, sieci konwolucyjnych, części sieci rekurencyjnych oraz wielu innych architektur sieci neuronowych.

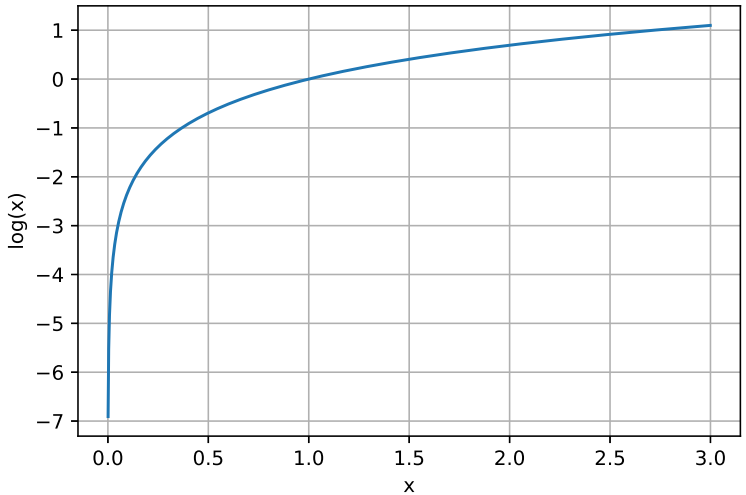
Funkcje celu (binarna entropia krzyżowa oraz kategorialna entropia krzyżowa)

Logarytm - przypomnienie ze szkoły średniej

Dwie właściwości niezbędne do zrozumienia binarnej entropii krzyżowej oraz kategorialnej entropii:

$$\log(1) = 0,$$

$$\lim_{x \to 0^+} \log(x) = -\infty.$$



Rysunek: Wykres logarytmu naturalnego

Funkcje celu - binarna entropia krzyżowa

Binarna entropia krzyżowa dla uczenia on-line (każdorazowo jedna próbka ze zbioru uczącego):

$$E_1 = -\left[y \log p + (1-y) \log(1-p)\right], \tag{19}$$

gdzie
y - oznacza prawdziwą klasę i może być równe 0 lub 1,
p - wartość wyjścia sieci, $p \in (0,1)$, gdyż f.aktywacji w ostatniej warstwie jest logistyczna zgodnie z tab. ze slajdu 121,
log - oznacza logarytm naturalny.

Wartości wyjścia bliskie 1 lub 0

Im wartość wyjścia bliższa 1, tym sieć z większą pewnością wskazuje na klasę jeden. Podobnie wartości wyjścia zbliżone do 0 oznaczają, że sieć wskazuje z dużą pewnością klasę 0.

Binarna entropia krzyżowa

Jeżeli $y = p$, to $E_1 = 0$. Proszę przeanalizować sytuację, gdy:
a) $y = 0$ i $p \approx 1$,
b) $y = 1$ i $p \approx 0$.
Binarna entropia krzyżowa, karze obydwa typy błędów, a zwłaszcza wyjątkowo silnie karane są nieprawidłowe odpowiedzi wskazywane z dużą pewnością !!!

Funkcje celu - binarna entropia krzyżowa

Prosty kod funkcji zwracającej wartość E_1:

```
def BinarnaEntropiaKrzyzowa (y,p):
    if y == 1:
      return -log(p)
    else:
      return -log(1 - p)
```

Można przeanalizować wartości zwracane przez funkcję dla $y = 0$ i $y = 1$ dla p zbliżonego do 1 oraz 0.

Binarna entropia krzyżowa

Wartość binarnej entropii krzyżowej przy uczeniu wsadowym lub mini-batchami:

$$E_2 = -\sum_{i=1}^{n} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right], \tag{20}$$

gdzie
n - liczba elementów mini-batchy,
p_i - wartość wyjścia sieci dla $i$tego elementu mini-batcha,
y_i - oznacza prawdziwą klasę.
Jeżeli n jest równe liczbie wszystkich elementów zbioru uczącego, to uczenie jest wsadowe.

Funkcje celu - Kategorialna entropia krzyżowa

C oznacza liczbę klas (np. C=4 dla 4 klas: rower, motorower, skuter, motocykl).
Kategorialna entropia krzyżowa przy uczniu on-line:

$$E_3 = -\sum_{k=1}^{C} y_k \log p_k. \tag{21}$$

gdzie
p_k - wartość k-tego wyjścia sieci,
y_k - pożądana wartość k-tego wyjścia. Jedno wyjście ma wartość 1, a pozostałe 0 zgodnie z kodowaniem gorącą jedynką (ang. hot one)
Przypomnienie kodowania z gorącą jedynką: jeżeli liczba klas $C = 4$, to wektor wskazujący III klasę ma postać $[0, 0, 1, 0]^T$.
Proszę zwrócić uwagę, że wszystkie wyjścia sieci sumują się do 1 ($p_1 + p_2 + \cdots + p_C = 1$), gdyż w ostatniej warstwie są stosowane funkcje aktywacji softmax zgodnie z tab. ze slajdu 121.

Przy uczeniu wsadowym lub mini-batchami:

$$E_4 = -\sum_{i=1}^{n}\sum_{k=1}^{C} y_{i,k} \log p_{i,k} \tag{22}$$

Funkcje celu - Kategorialna entropia krzyżowa

Keras: categorial_crossentropy i sparse_categorial_crossentropy

categorial_crossentropy - dla klas zakodowanych za pomocą gorącej jedynki np. [0,1,0].
sparse_categorial_crossentropy - dla klas zakodowanych za pomocą numerów.
Taki sam wzór określający wartość f. straty.
W przypadku wielu klas sparse_categorial_crossentropy pozwala trochę zaoszczędzić pamięć zastępując kodowanie gorącą jedynką [1,0,0,...,0], [0,1,0,....,0], [0,0,1,....,0] odpowiednimi numerami klas 1, 2, 3.

Funkcje celu - błąd średniokwadratowy MSE (ang. Mean Squared Error)

MSE jest stosowany w zadaniach regresji. W warstwie wyjściowej są liniowe f. aktywacji (gdyby w warstwie wyjściowej były funkcje aktywacji tanh lub logistyczne, to wartość wyjścia mogłaby się zmieniać tylko w przedziale $(-1, 1)$ lub $(0, 1)$. W przypadku funkcji ReLU w warstwie wyjściowej wartość wyjścia nie mogła by być ujemna).
Przy uczeniu wsadowym lub mini-batchami sieci z jednym wyjściem:

$$E_5 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{23}$$

gdzie
$\hat{y}_i \in \mathcal{R}$ - wartość wyjścia sieci
$y_i \in \mathcal{R}$ - pożądana wartość wyjścia
Uwaga - w toolboxie NNSYSID jest stosowana funkcja celu (5) przemnożona przez 0.5.
Jeżeli sieć ma więcej niż jedno wyjście (zadania regresji wektorowej), to:

$$E_6 = c\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{m}(y_{i,k} - \hat{y}_{i,k})^2, \tag{24}$$

gdzie
m oznacza liczbę wyjść,
c w różnych publikacjach ma jedną z następujących wartości 1, $\frac{1}{m}$, 0.5 lub $\frac{1}{2m}$.

Funkcje celu - błąd średniokwadratowy MSE (ang. Mean Squared Error)

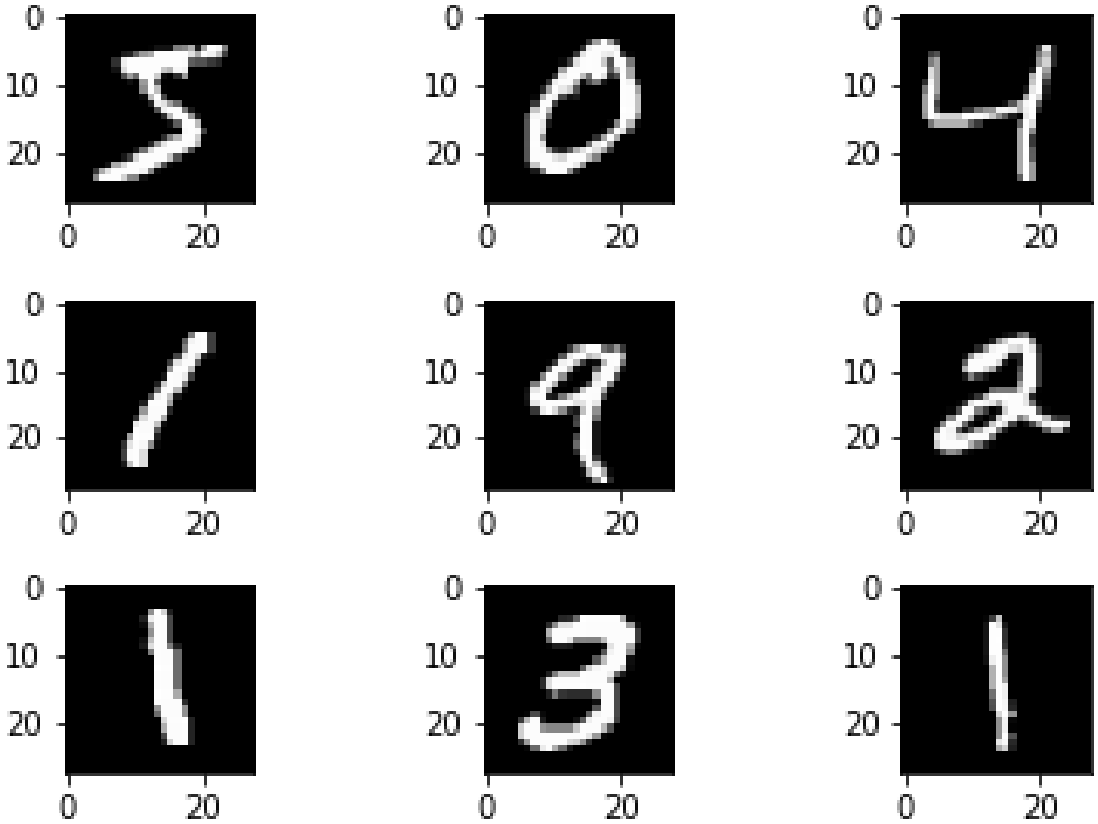
MSE jednakowo karze błędy o wartościach dodatnich i ujemnych
Obecność nawet pojedyńczych błędnych danych (błędów grubych) w zbiorze uczącym, które mają duży wpływ na wartość MSE, istotnie wpłynie na rezultaty uczenia. Jeden element odstający może mieć większy wpływ na wartość MSE, niż wszystkie pozostałe dane!

Jeżeli występują błędy grube, to można MSE zastąpić średnim błędem bezwzględnym - MAE (ang. mean absolute error) lub średnią z
https://www.tensorflow.org/api_docs/python/tf/keras/losses/Huber

$$E_7(e) = \begin{cases} \frac{1}{2}e^2 & \text{dla } |e| \leqslant \delta, \\ \delta\left(|e| - \delta/2\right) & \text{w przeciwnym przypadku,} \end{cases} \tag{25}$$

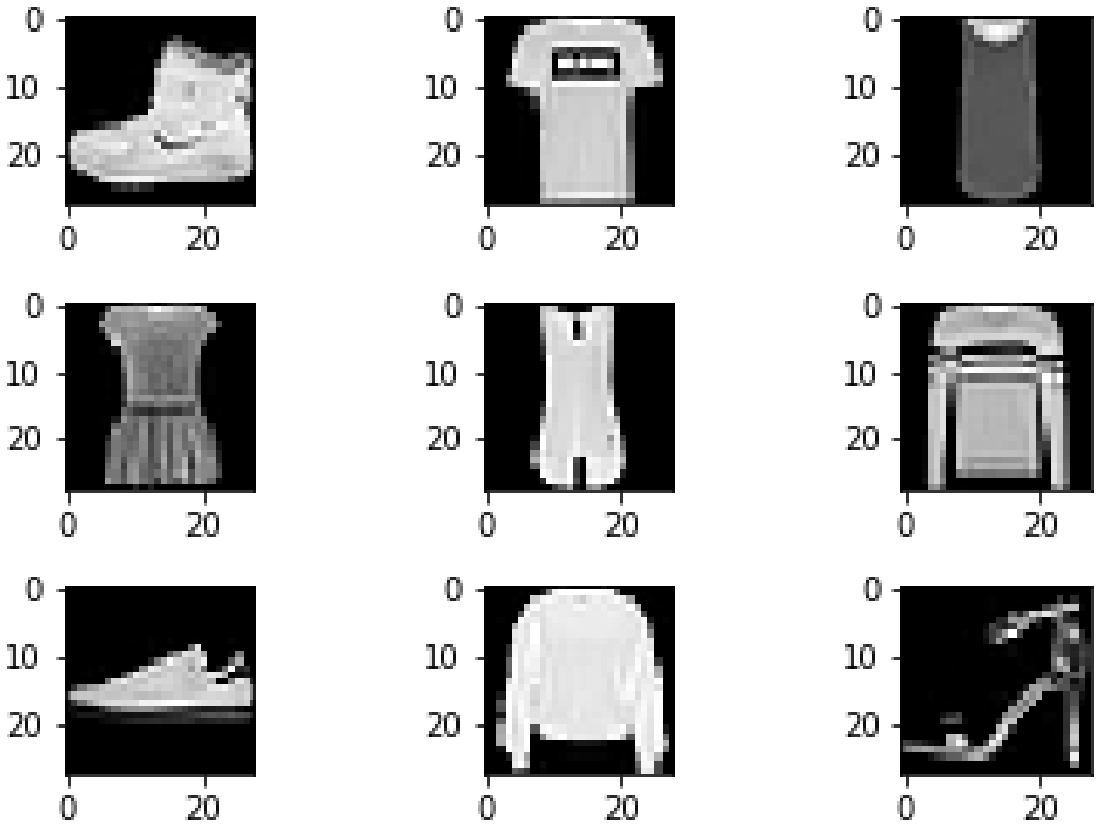
gdzie e oznacza błąd. Dla $|e| \leqslant \delta$ zależność kwadratowa (jak MSE), a dla większych liniowa (jak MAE).

Wybrane zbiory danych - MNIST
Odpowiednik "Hello world" w maszynowym uczeniu



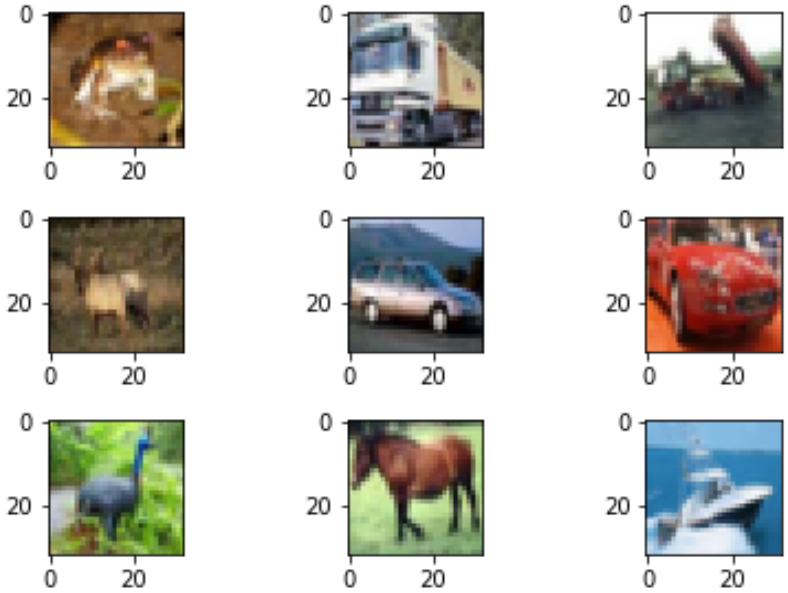
Obrazy 28x28
10 klas
Zbiór uczący - 60000 obrazów
Zbiór testowy - 10000 obrazów
http://yann.lecun.com/exdb/mnist/

Wybrane zbiory danych - Fashion MNIST



Obrazy 28x28
10 klas
Zbiór uczący - 60000 obrazów
Zbiór testowy - 10000 obrazów

Wybrane zbiory danych - CIFAR10 i CIFAR100



Obrazy 32x32
W CIFAR10 jest 10 klas: samoloty, samochody, ptaki, koty, jelenie, psy, żaby, konie, statki, ciężarówki
W CIFAR100 jest 20 zgrubnych kategorii i 100 szczegółowych
Zbiór uczący - 50000 obrazów
Zbiór testowy - 10000 obrazów
https://www.cs.toronto.edu/~kriz/cifar.html

Zbiory danych w Kerasie https://keras.io/api/datasets/ (W scikit i tensorflow jest więcej)

MNIST - klasyfikacja ręcznie pisanych cyfr
CIFAR10 - klasyfikacja obrazów
CIFAR100 - - klasyfikacja obrazów
IMDB - klasyfikacja recenzji filmów
Reuters newswire - klasyfikacja wiadomości
Fashion MNIST dataset - alternatywa do MNISTa
Boston Housing price - zadanie regresji (szacowanie cen mieszkań na podstawie 13 cech)

```
from keras.datasets import mnist
(trainX, trainy), (testX, testy) = mnist.load_data()
```

W scikit-learn jest dostępnych więcej zbiorów danych
https://scikit-learn.org/stable/datasets.html

Microsoft COCO - Common Objects in Context)

COCO - bardzo popularny

Pascal VOC - PASCAL Visual Object Classes

PASCAL VOC - też popularny

Zestawienie popularnych zbiorów z obrazami

https://en.wikipedia.org/wiki/List_of_datasets_in_computer_vision_and_image_processing

Niektóre inne zbiory danych w Internecie

kaggle.com
https://archive.ics.uci.edu/ml/index.php UC Irvine Machine Learning Repository
https://ibug.doc.ic.ac.uk/resources Intelligent Behaviour Understanding Group (iBUG), Department of Computing, Imperial College London - m.in. bazy danych z ludzkimi twarzami
https://ibug.doc.ic.ac.uk/resources/agedb/ Zbiór AgeDB - rozpoznawanie wieku osób z fotografii
https://cocodataset.org/#home COCO stworzona przez Microsoft - do detekcji obiektów lub segmentacji semantycznej (ang. semantic segmentation - przypisywanie pikseli zgodnie z klasą obiektu, do którego należy np. samochód, człowiek itp.)
https://scikit-learn.org/stable/datasets.html - zbiory w scikit-learn
W Internecie jest znacznie więcej zbiorów danych...

Keras - funkcja celu i metryka

```
#compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

dokładność (ang. accuracy)

dokładność - stosunek liczby poprawnie przewidzianych do wszystkich

Metryki - Keras

Metryka - funkcja użyta do oceny jakości modelu
Metryki są podobne do funkcji celu, ale nie są używane do uczenia sieci - nie są stosowane do obliczania wektora gradientu.
Warto zauważyć, że w Kerasie można użyć funkcję straty jako metrykę.

Metryki - dlaczego nie sama dokładność ?

dokładność (ang. accuracy) - stosunek liczby poprawnie przewidzianych do wszystkich

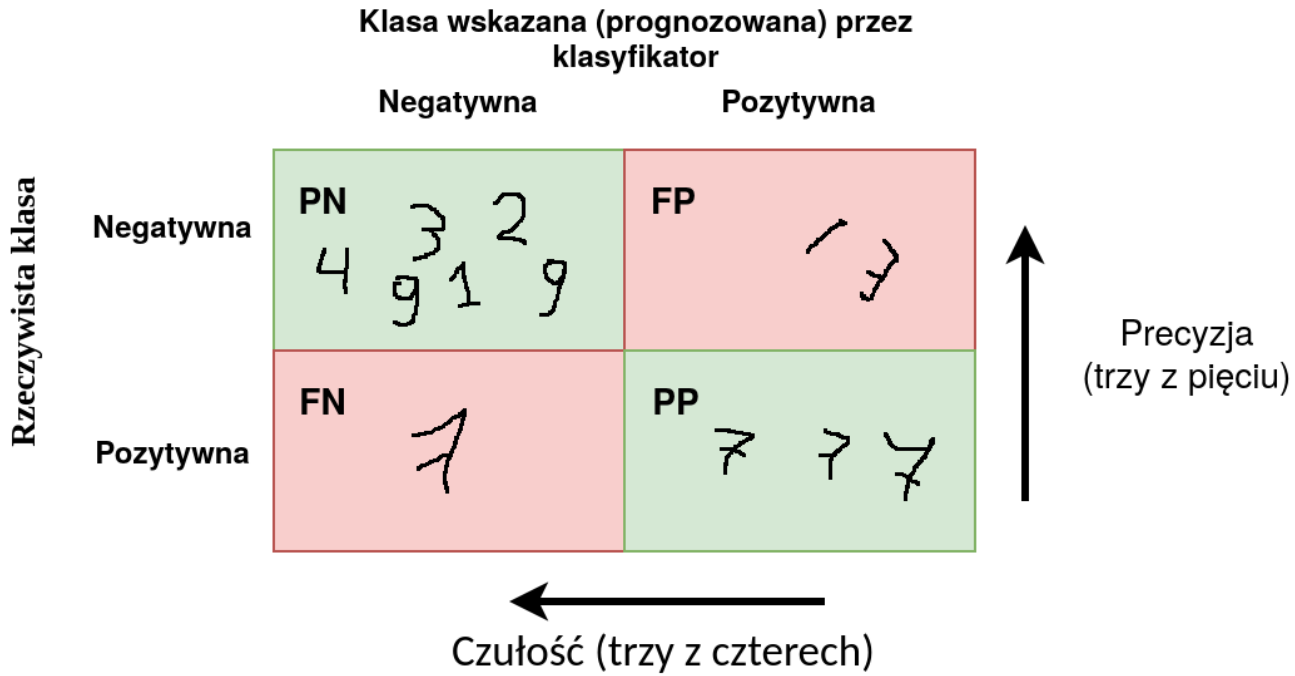
niezrównoważenie klas

Załóżmy, że są dwie klasy (klasyfikacja binarna) i w zbiorze danych 95% przykładów należy do klasy pierwszej.
Jeżeli klasyfikator będzie zawsze zwracał klasę pierwszą, to jaka będzie jego dokładność?

Dokładność będzie 0.95, co dla takiego zbioru jest bardzo kiepskim wynikiem !

Klasyfikator - siódemki vs reszta

Rozpoznawanie cyfr z MNIST. Jeżeli klasyfikator będzie zawsze wskazywał, że nie jest siódemka, to jego dokładność będzie wynosiła 90%. Potrzebne są jeszcze inne miary np. precyzja i czułość zwana też pełnością (odsetek prawdziwie pozytywnych).

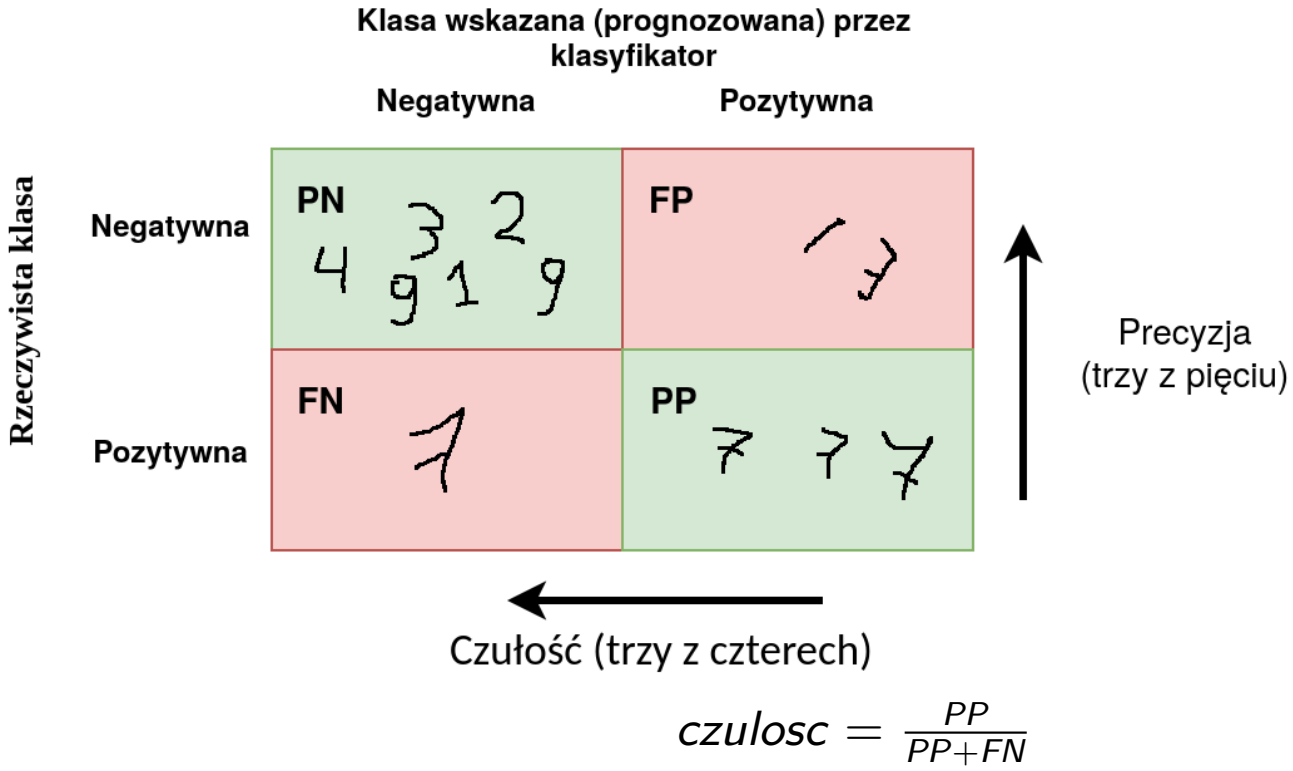


$$czulosc = \frac{PP}{PP+FN}$$

Synonimy: czułość, pełność (ang. sensitivity)

$$precyzja = \frac{PP}{PP+FP}$$

Metryki



$$precyzja = \frac{PP}{PP+FP}$$

$$czulosc = \frac{PP}{PP+FN}$$

Metryki

W niektórych sytuacjach istotniejsza jest większa precyzja, a w innych czułość.
Rozpoznawanie chorób - ważniejsza większa czułość (fałszywe alarmy zostaną zweryfikowane przez lekarzy)
Rozpoznawanie filmów odpowiednich dla dzieci - ważniejsza większa precyzja (lepiej odrzucić kilka dobrych filmów niż przepuścić nieodpowiednie)

Kompromis między precyzją a czułością

Zwiększenie precyzji zmniejsza czułość i vice versa.

Wskaźnik F_1

Przy porównywaniu klasyfikatorów czasami wygodnie jest połączyć precyzję z czułością w jednym wskaźniku.
F_1 - średnia harmoniczna z precyzji i czułości - mała wartość któregokolwiek składnika powoduje większe znaczne zmniejszenie średniej harmonicznej niż średniej arytmetycznej.
Wskaźnik F_1 - premiuje klasyfikatory mające dużą precyzję i czułość.

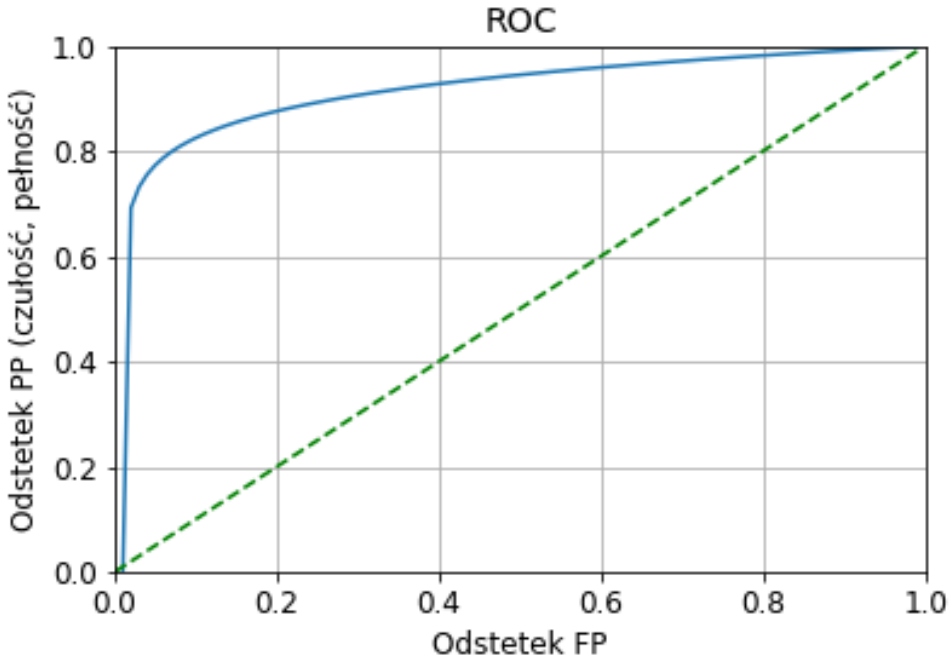
$$F_1 = \frac{2}{\frac{1}{precyzja} + \frac{1}{pelnosc}} = 2\frac{precyzja \cdot pelnosc}{precyzja + pelnosc}$$

Metryki

W niektórych sytuacjach istotniejsza jest większa precyzja, a w innych czułość.
Rozpoznawanie chorób - ważniejsza większa czułość (fałszywe alarmy zostaną zweryfikowane przez lekarzy)
Rozpoznawanie filmów odpowiednich dla dzieci - ważniejsza większa precyzja (lepiej odrzucić kilka dobrych filmów niż przepuścić nieodpowiednie)

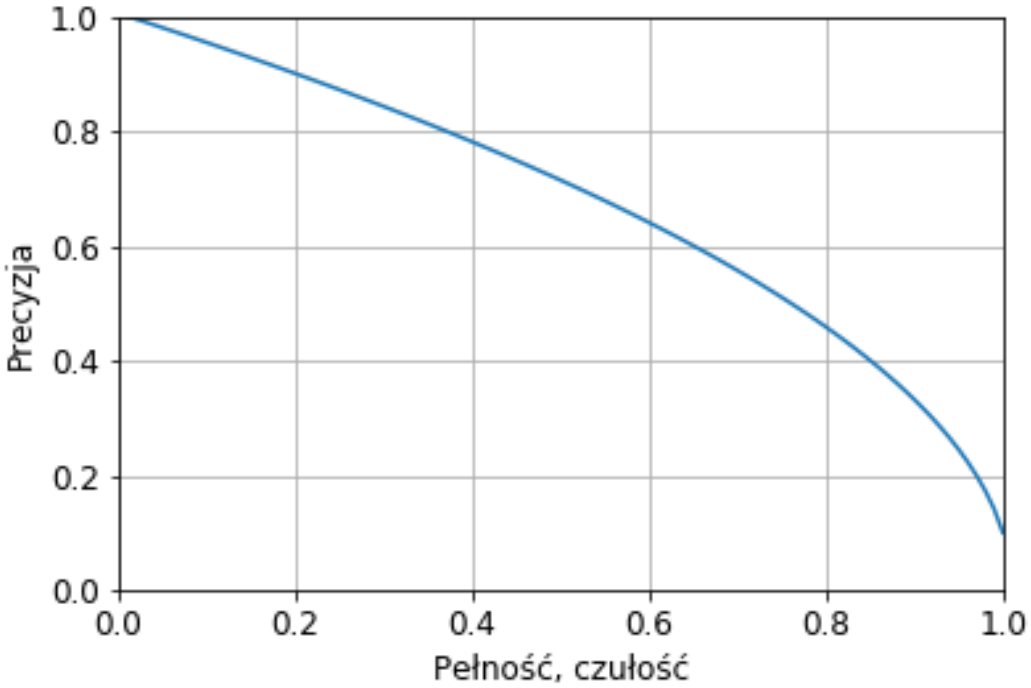
Krzywa ROC

Na niebiebsko -charakterystyka robocza odbiornika (ang. receiver operating characteristic) - popularne narzędzie do porównywania klasyfikatorów binarnych



Odsetek prawdziwie negatywnych - specyficzność
Krzywa ROC jest wykresem czułości w funkcji $(1 - \texttt{specyficzność})$
Kompromis - im większy odsetek prawdziwie pozytywnych OPP, tym większy odsetek fałszywie pozytywnych OFP.
Obszar pod krzywą ROC w przypadku doskonałego klasyfikatora byłby równy 1, a dla całkowicie losowego byłby równy 0.5.
Dobry klasyfikator odpowiednio nad linią przerywanej.

Precyzja w funkcji czułości

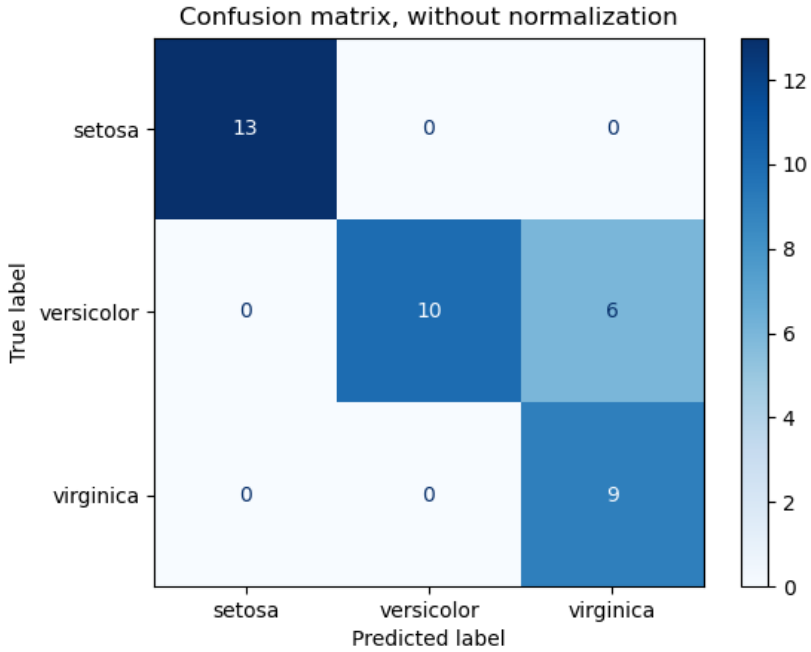


Ten wykres jest używany, gdy klasa pozytywna występuje rzadko lub gdy odsetek fałszywie pozytywnych jest istotniejszy niż odsetek fałszywie negatywnych. W pozostałych przypadkach przeważnie stosuje się wykres ROC.

Macierz pomyłek (ang. confusion matrix)

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

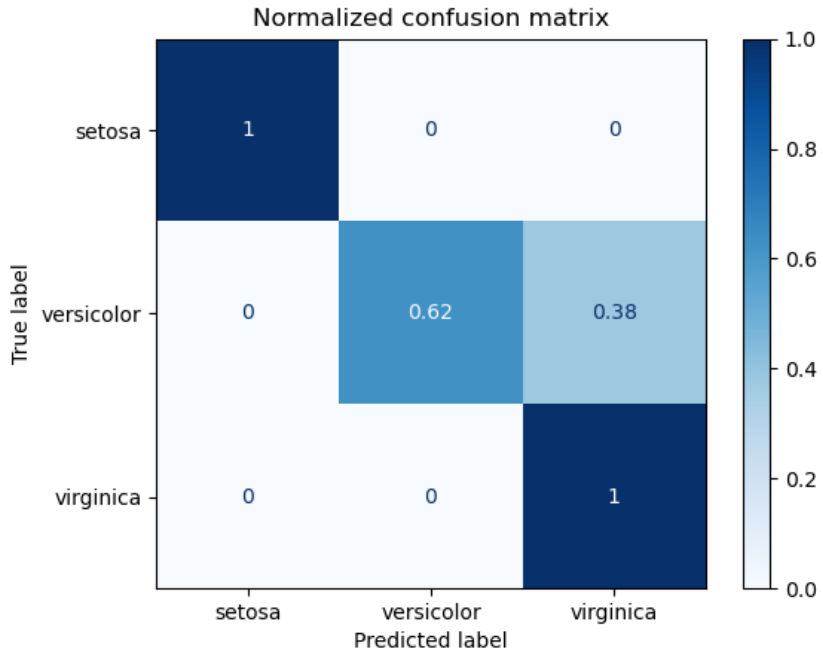
Rozpoznawanie trzech gatunków irysów.



predicted label - gatunek wskazany przez sieć
true label - rzeczywisty gatunek

Znormalizowana macierz pomyłek (ang. confusion matrix)

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html



predicted label - gatunek wskazany przez sieć
true label - rzeczywisty gatunek

Metryki

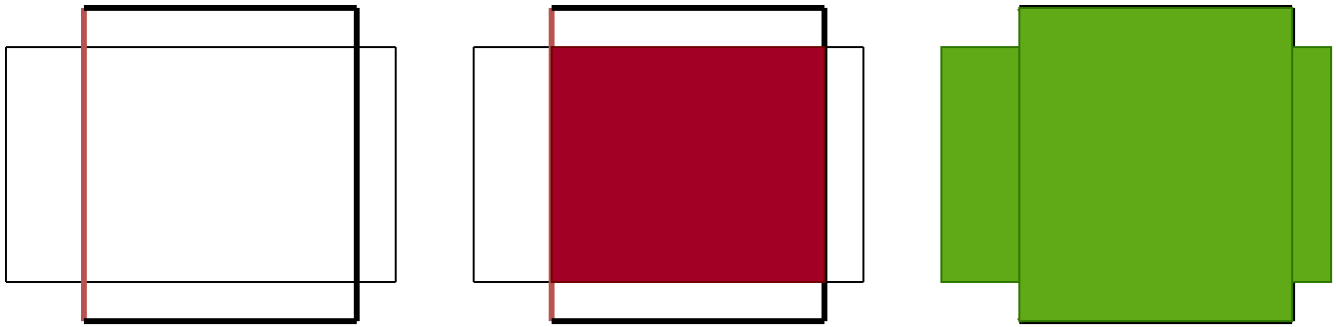
```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
             precision    recall  f1-score   support

    class 0       0.50      1.00      0.67         1
    class 1       0.00      0.00      0.00         1
    class 2       1.00      0.67      0.80         3

avg / total       0.70      0.60      0.61         5
```

https://scikit-learn.org/0.15/modules/generated/sklearn.metrics.classification_report.html

`keras.metrics.MeanIoU`.

Indeks Jaccarda - Intersection over Union (IoU) - przydatny do oceny lokalizacji. Część wspólna obszaru ramki przewidzianej przez sieć i pożądanego obszaru ramki (podanego w zbiorze uczącym) podzielona przez pole obszaru należącego do jednej lub drugiej ramki.

$$\texttt{IoT} = \frac{\texttt{czerwone pole}}{\texttt{zielone pole}}$$



Metryki do oceny np. tłumaczeń - pojęcia wstępne - N-gramy

Dla zdania "the cat sat on the mat," N-gramamami (ang. N-grams) są:

1-gram (unigram): "the," "cat," "sat", "on", "the", "mat"

2-gram (bigram): "the cat," "cat sat," "sat on", "on the", "the mat",

3-gram (trigram): "the cat sat" "cat sat on", "sta on the", "on the mat"

4-gram: "the cat sat on", "cat sat on the", "sat on the mat"

UWAGA:
W NLP N-gramy mogą być też na poziomie liter lub sylab, ale w metryce BLEU korzystamy z N-gramów złożonych ze słów (zwykle do 4 słów).

Metryki do oceny np. tłumaczeń - próba utworzenia metryki

Załóżmy, że mamy sieć do tłumaczenia tekstu lub generacji opisów zawartości zdjęć (ang. image captioning) i chcemy ocenać jakość wygenerowanego tekstu.

Przykładowy wynik wygenerowany przez sieć:
the the the the the the the

Referencje (tworzone przez np. bardzo dobrych tłumaczy):
Referencja 1 the cat is on the mat
Referencja 2 there is a cat on the mat

Pierwszy pomysł - użycie "zmodyfikowanej" precyzji

$$p = \frac{m}{w},$$

gdzie m oznacza liczbę słów wygenerowanego tekstu znalezionych w referencji, w jest liczbą słów referencji.
Łatwo zauważyć na powyższym przykładzie, że ten pomysł jest wyjątkowo kiepski, ponieważ dla referencji .2. $w = 7$ i $m = 7$ (siedem wygenerowanych słow "the" jest w referencji), więc $p = 7/7 = 1$.

Metryki do oceny np. tłumaczeń - próba utworzenia metryki - spróbujmy sprytniej

Wynik wygenerowany przez sieć:
the the the the the the the

Referencje:
Referencja 1 the cat is on the mat
Referencja 2 there is a cat on the mat

Uwzględnijnmy maksymalną liczbę wystąpień wystąpień wygenerowanego słowa w tłumaczeniach referencyjnych. W powyższym przykładzie, wyraz "the" pojawia się dwukrotnie w ref. 1 i raz w ref. 2. Maksymalną liczba wystąpień jest $m_{max} = 2$.
Dla wszytskich wygenerowanych słów liczymy maksymalna liczbę wystąpień. Następnie sumujemy maksymalne liczby wystąpień wszystkich słów i sumę tą dzielimy przez maksymalna liczbę unigramów (1-gramów) w wygenerowanym tekście. Dla podanego przykładu otrzymujemy tą metodą zmodyfikowanąprecyzję 2/7.

Metryki do oceny np. tłumaczeń - próba utworzenia metryki - spróbujmy sprytniej

A co byłoby, gdyby siec wygenerowała tekst "the cat" ?
Dla "the cat" zmodyfikowana precyzja:

$$\frac{1}{d} + \frac{1}{d} = \frac{1}{2} + \frac{1}{2} = \frac{2}{2} = 1,$$

($d = 2$ jest liczbą słów wygenerowanej odpowiedzi - liczbą unigramów).

Wartość zmodyfikowanej precyzji wynosi aż 1 (czyli 100%) - więc trzeba jeszcze sprytniejszej metryki. Poza tym możemy mieć niegramatyczne odpowiedzi - np. słowa w złej kolejności.

Metryki do oceny np. tłumaczeń - próba utworzenia metryki - spróbujmy sprytniej

Wynik wygenerowany przez sieć:
the cat

Referencje:
Referencja 1 the cat is on the mat
Referencja 2 there is a cat on the mat

Dla "the cat" modyfikowana precyzja:

$$\frac{1}{d} + \frac{1}{d} = \frac{2}{2} = 1,$$

Zauważamy, że zbyt krótkie teksty generowane przez sieć mogą mieć bardzo dużą wartość zmodyfikowanej precyzji (100%). Przydałaby się kara BP (ang. breive penalty) za zbyt krótkie odpowiedzi:

$$\text{BP} = \begin{cases} 1 & \text{jeśli } d > r, \\ e^{(1-\frac{r}{d})} & \text{jeśli } d \leqslant r, \end{cases} \tag{26}$$

gdzie r oznacza długość najdłuższego tekstu referencyjnego, d jest długością wygenerowanego tekstu.
Kara BP zapobiega bardzo wysokim ocenom zbyt krótkich tłumaczeń!

Metryki do oceny np. tłumaczeń - próba utworzenia metryki - spróbujmy sprytniej

Kolejny problem:
Możemy otrzymać niegramatyczną odpowiedź np. słowa w złej kolejności.

W praktyce jednak używanie pojedynczych słów do porównania tekstów nie jest dobre. Zamiast tego metryka BLEU oblicza zmodyfikowaną metrykę precyzji, używając N-gramów. Stwierdzono, że długość, która ma „najwyższą korelację z jednojęzycznymi ocenami ludzi", wynosi 4. Stwierdzono, że wyniki unigramów odpowiadają za adekwatność tłumaczenia i ilość zachowanych informacji. Dłuższe oceny N-gramowe odpowiadają za płynność tłumaczenia lub w jakim stopniu brzmi ono jak „dobry angielski".

Metryka BLEU (ang. BIlingual Evaluation Understudy) - metryka opracowana przez IBM w 2001

Popularna metryka, która dobrze koreluje z ludzka oceną generowanego tekstu. Mozna ją wykorzystać np. w tłumaczanie maszynowym do oceny jakości wygenerowanego tekstu w stosunku do jednego lub większej ilości tekstów referencyjnych (np. tłumaczeń). BLEU przyjmuje wartości z zakresu [0,1].

BLEU

BLEU wykorzystuje precyzję i karę za zwiezłość.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{i=1}^{n} w_i \ln p_i\right), \tag{27}$$

gdzie p_i oznacza zmodyfikowaną precyzję dla i-gramów, n maksymalna długość N-gramów (zwykle przyjmuje sie 4), w_i jest wagą, która okresla znaczenia i-gramów - (mamy całą rodzinę metryk BLEU rózniących sie doborem wartości w_i i n. W najprostszym przypadku $w_i = 1$. BLEU nie jest idealną metryką np. zależy od ilości dostępnych tłumaczeń - referencyjnych tekstów.

Metryka BLEU (ang. BIlingual Evaluation Understudy) - metryka opracowana przez IBM w 2001

Im wyższa wartość BLEU tym lepsza jakość generowanego tłumaczenia (lub opisu zdjęcia).

BLEU $<$ 0,1 - raczej bezużyteczny wynik
BLEU $<$ 0,2 - trudno zrozumieć istotę
BLEU $<$ 0,4 - dużo błedów gramatycznych
BLEU $>$ 0,5 - przyzwoite lub dobre tłumaczenia.
Inni zawodowi tłumacze często też nie przetłumaczą identycznie, jak któreś z tłumaczeń referencyjnych, więc nie należy oczekiwać, że BLEU=1.

Metryka BLEU (ang. BIlingual Evaluation Understudy) - metryka opracowana przez IBM w 2001

Inne metryki do NLP i tłumaczeń:
ROUGE (ang. Recall-Oriented Understudy for Gisting Evaluation)
METEOR (ang. Metric for Evaluation of Translation with Explicit Ordering)
BERT Score (ang. Bidirectional Encoder Representations from Transformers) Score

Te metryki są ładnie opisane na stronie https://medium.com/@kbdhunga/nlp-model-evaluation-understanding-bleu-rouge-meteor-and-bertscore-9bad7db
zatytułowanej "NLP Model Evaluation: Understanding BLEU, ROUGE, METEOR, and BERTScore".

„gisting" oznacza przygotowanie wstępnego tłumaczenia, które pozwala zapoznać się z tematem i treścią tłumaczonego materiału.

Keras - Sieci rekurencyjne

Keras umożliwia również łatwe tworzenie sieci rekurencyjnych - wystarczy w modelu `Sequential()` dodać odpowiednią warstwę rekurencyjną (może być nawet kilka warstw rekurencyjnych dla skomplikowanych zadań)

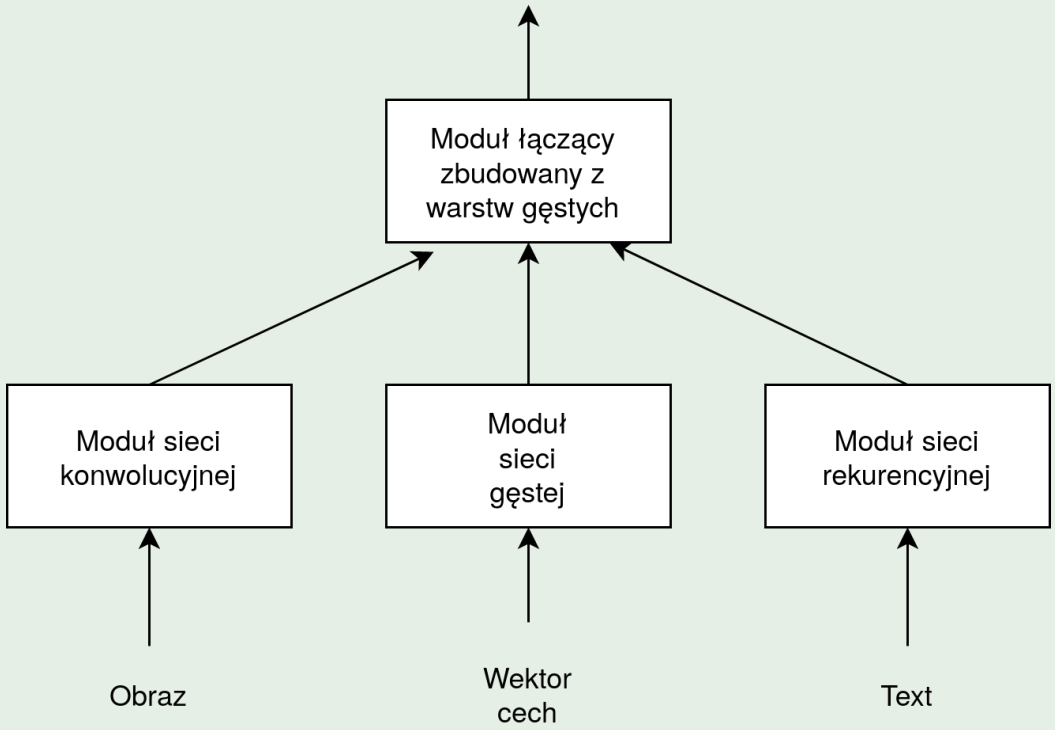
Przykład tworzenia sieci LSTM (ang. Long short-term memory)

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

model = Sequential()
model.add(LSTM(20, input_shape=(lahead, 1), batch_size=1)
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
```

Sieci rekurencyjne będą krótko omówione na innym wykładzie.

Sieć przetwarzająca równocześnie różnorodne dane wejściowe (np. obrazy, tekst i dane numeryczne)

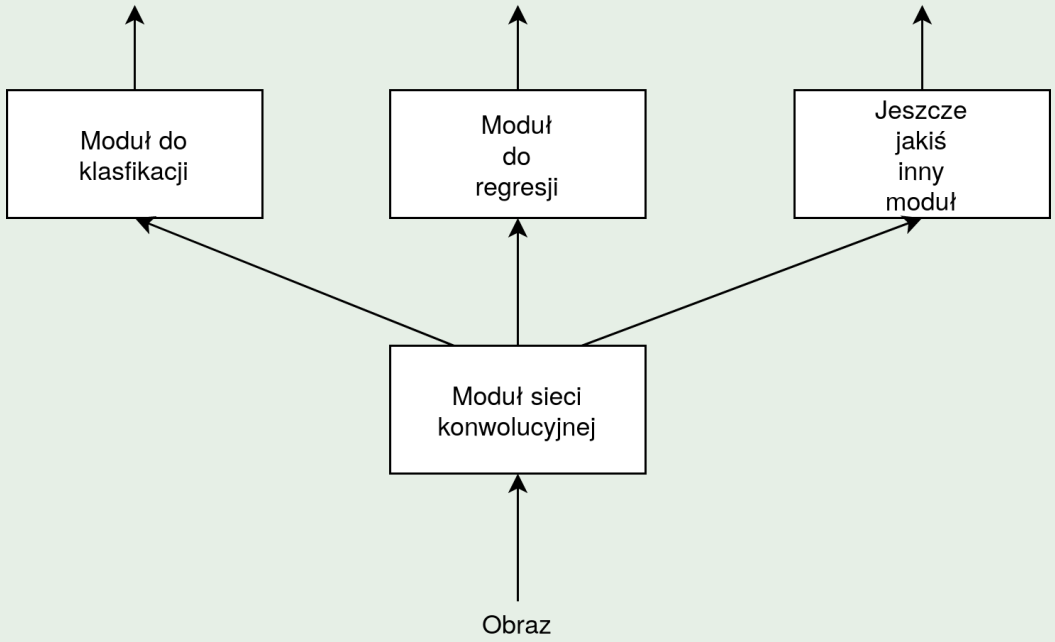
Można utworzyć sieci przetwarzające bardzo zróżnicowane dane



Rysunek: Przykład sieci do klasyfikacji i regresji na podstawie obrazów, tekstu i danych numerycznych

Taką sieć można stworzyć w Kerasie z `functional API`.

Można też utworzyć sieć wykonującą równocześnie różne zadania



Rysunek: Przykład sieci wykonującej równocześnie różne zadania (z różnorodnymi wyjściami)

Taką sieć można stworzyć w Kerasie z `functional API`.

Literatura do tego wykładu

[FChollet] Francois Chollet, Deep Learning Praca z językiem Python i biblioteką Keras, Helion, 2019 (dostępna online dla studentów PWr) - książka ta jest praktycznym kursem uczenia głębokiego napisanym przez autora biblioteki Keras

[AGeron] Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Wyd. III, Helion, 2023 - (w wyd.I był archaiczny tensorflow 1)

[S.OsowskiMatlabPython] Matematyczne modele uczenia maszynowego w językach MATLAB i PYTHON, OWPW, 2023 - różne metody uczenia maszynowego i więcej architektur sieci głębokich ale bez alg. gradientowych II rzędu, które nie są stosowane w sieciach głębokich.

Dziękuję za uwagę