

OKIEM EKSPERTA

Python

Uczenie maszynowe w przykładach

Najlepsze praktyki w realnych zastosowaniach

Wydanie IV

Yuxi (Hayden) Liu



Helion 

«packt»

Tytuł oryginału: Python Machine Learning By Example: Unlock machine learning best practices with real-world use cases, 4th Edition

Tłumaczenie: Piotr Rajca, z wykorzystaniem fragmentów poprzedniego wydania w przekładzie Andrzeja Watraka

ISBN: 978-83-289-3171-8

Copyright © Packt Publishing 2024. First published in the English language under the title 'Python Machine Learning By Example - Fourth Edition – (9781835085622)'

Polish edition copyright © 2026 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

helion.pl/user/opinie/pytuc4

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: helion.pl (księgarnia internetowa, katalog książek)

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści |

O autorze	13
O korektorach merytorycznych	14
Przedmowa	17
ROZDZIAŁ 1	
Pierwsze kroki z uczeniem maszynowym i Pythonem	21
Wprowadzenie do uczenia maszynowego	21
Dlaczego uczenie maszynowe jest potrzebne?	22
Różnice między uczeniem maszynowym a automatyką	24
Zastosowania uczenia maszynowego	25
Wstępne wymagania	26
Trzy rodzaje uczenia maszynowego	27
Krótka historia rozwoju algorytmów uczenia maszynowego	29
Istota uczenia maszynowego	30
Uogólnianie danych	31
Nadmierne i niedostateczne dopasowanie modelu oraz kompromis między obciążeniem a wariancją	32
Zapobieganie nadmiernemu dopasowaniu poprzez weryfikację krzyżową	36
Zapobieganie nadmiernemu dopasowaniu za pomocą regularyzacji	38
Zapobieganie nadmiernemu dopasowaniu poprzez selekcję cech i redukcję wymiarowości	40
Wstępne przetwarzanie danych i inżynieria cech	41
Wstępne przetwarzanie i eksploracja danych	42
Inżynieria cech	46
Łączenie modeli	47
Głosowanie i uśrednianie	47
Agregacja bootstrap	47
Wzmacnianie	49
Składowanie	49

Instalacja i konfiguracja oprogramowania	51
Przygotowanie Pythona i środowiska pracy	51
Instalacja najważniejszych pakietów Pythona	52
Podsumowanie	57
Ćwiczenia	57

ROZDZIAŁ 2

Tworzenie systemu rekomendacji filmów na bazie naiwnego

klasyfikatora Bayesa	58
Pierwsze kroki z klasyfikacją	58
Klasyfikacja binarna	59
Klasyfikacja wieloklasowa	60
Klasyfikacja wieloetykietowa	61
Naiwny klasyfikator Bayesa	62
Twierdzenie Bayesa w przykładach	62
Mechanizm działania naiwnego klasyfikatora Bayesa	64
Implementacja naiwnego klasyfikatora Bayesa	68
Implementacja od podstaw	68
Implementacja z wykorzystaniem pakietu scikit-learn	71
Budowanie systemu rekomendacyjnego na bazie klasyfikatora Bayesa	72
Wstępne przygotowanie danych	72
Trenowanie naiwnego klasyfikatora Bayesa	76
Ocena jakości klasyfikacji	77
Strojenie modeli poprzez weryfikację krzyżową	81
Podsumowanie	83
Ćwiczenia	84
Bibliografia	84

ROZDZIAŁ 3

Prognozowanie kliknięć reklam internetowych przy użyciu

algorytmów drzewiastych	85
Wprowadzenie do prognozowania kliknięć reklam	85
Wprowadzenie do dwóch typów danych: liczbowych i kategorialnych	87
Badanie drzewa decyzyjnego od korzeni do liści	87
Budowanie drzewa decyzyjnego	89
Wskaźniki jakości podziału zbioru	91
Implementacja drzewa decyzyjnego od podstaw	96
Implementacja drzewa decyzyjnego za pomocą biblioteki scikit-learn	102
Prognozowanie kliknięć reklam za pomocą drzewa decyzyjnego	103
Gromadzenie drzew decyzyjnych: las losowy	108

Gromadzenie drzew decyzyjnych: drzewa ze wzmocnieniem gradientowym	110
Podsumowanie	113
Ćwiczenia	113

ROZDZIAŁ 4

Prognozowanie kliknięć reklam internetowych

przy użyciu regresji logistycznej 114

Przekształcanie cech kategoryalnych w liczbowe: kodowanie porządkowe i „1 z n”	114
Klasyfikowanie danych z wykorzystaniem regresji logistycznej	117
Wprowadzenie do funkcji logistycznej	118
Przejście od funkcji logistycznej do regresji logistycznej	119
Trening modelu opartego na regresji logistycznej	122
Trening modelu opartego na regresji logistycznej z gradientem prostym	122
Prognozowanie kliknięć reklam z wykorzystaniem regresji logistycznej z gradientem prostym	127
Trening modelu opartego na regresji logistycznej ze stochastycznym gradientem prostym (SGD)	128
Trening modelu opartego na regresji logistycznej z regularyzacją	130
Selekcja cech w regularyzacji L1	132
Selekcja cech z wykorzystaniem lasu losowego	133
Trening modelu na dużym zbiorze danych z uczeniem online	134
Klasyfikacja wieloklasowa	136
Implementacja regresji logistycznej za pomocą pakietu TensorFlow	138
Podsumowanie	141
Ćwiczenia	142

ROZDZIAŁ 5

Prognozowanie cen akcji za pomocą algorytmów regresji 143

Co to jest regresja?	143
Pozyskiwanie cen akcji	145
Krótkie wprowadzenie do giełdy i cen akcji	145
Pierwsze kroki z inżynierią cech	147
Pozyskiwanie danych i generowanie cech	150
Szacowanie za pomocą regresji liniowej	154
Jak działa regresja liniowa?	154
Implementacja regresji liniowej od podstaw	155
Implementacja regresji liniowej z wykorzystaniem pakietu scikit-learn	158
Implementacja regresji liniowej z wykorzystaniem pakietu TensorFlow ...	158

Prognozowanie za pomocą regresyjnego drzewa decyzyjnego	160
Przejście od drzewa klasyfikacyjnego do regresyjnego	160
Implementacja regresyjnego drzewa decyzyjnego	162
Implementacja lasu regresyjnego	166
Ocena jakości regresji	166
Prognozowanie cen akcji za pomocą trzech algorytmów regresji	168
Podsumowanie	172
Ćwiczenia	172

ROZDZIAŁ 6

Prognozowanie cen akcji za pomocą sztucznych sieci neuronowych 173

Demistyfikacja sieci neuronowych	173
Pierwsze kroki z jednowarstwową siecią neuronową	174
Funkcje aktywacji	175
Propagacja wstecz	178
Wprowadzanie kolejnych warstw do sieci neuronowej i uczenie głębokie	179
Tworzenie sieci neuronowej	181
Implementacja sieci neuronowej od podstaw	181
Implementacja sieci neuronowej przy użyciu pakietu scikit-learn	184
Implementacja sieci neuronowej przy użyciu pakietu TensorFlow	184
Tworzenie sieci neuronowych z wykorzystaniem PyTorch	186
Dobór właściwej funkcji aktywacji	187
Zapobieganie nadmiernemu dopasowaniu sieci	188
Dropout	189
Wczesne zakończenie treningu	191
Prognozowanie cen akcji za pomocą sieci neuronowej	193
Trening prostej sieci neuronowej	193
Dostrojenie parametrów sieci neuronowej	194
Podsumowanie	201
Ćwiczenie	201

ROZDZIAŁ 7

Badanie 20 grup dyskusyjnych przy użyciu technik analizy tekstu 202

Jak komputery rozumieją ludzi, czyli przetwarzanie języka naturalnego	202
Czym jest przetwarzanie języka naturalnego?	203
Historia przetwarzania języka naturalnego	204
Zastosowania przetwarzania języka naturalnego	204
Przegląd bibliotek Pythona i podstawy przetwarzania języka naturalnego ...	206
Instalacja najważniejszych bibliotek	207
Korpusy	208

Tokenizacja	210
Oznaczanie części mowy	211
Rozpoznawanie jednostek nazwanych	212
Stemming i lematyzacja	212
Modelowanie semantyczne i tematyczne	213
Pozyskiwanie danych z grup dyskusyjnych	214
Badanie danych z grup dyskusyjnych	216
Przetwarzanie cech danych tekstowych	219
Zliczanie wystąpień wszystkich tokenów	219
Wstępne przetwarzanie tekstu	222
Usuwanie stop-słów	222
Upraszczenie odmian	222
Wizualizacja danych tekstowych z wykorzystaniem techniki t-SNE	223
Co to jest redukcja wymiarowości?	224
Redukcja wymiarowości przy użyciu techniki t-SNE	224
Reprezentowanie słów w formie gęstych wektorów — osadzenia słów ...	227
Podsumowanie	231
Ćwiczenia	232

ROZDZIAŁ 8

Wyszukiwanie ukrytych tematów w grupach dyskusyjnych

poprzez ich klastrowanie i modelowanie tematyczne 233

Nauka bez wskazówek, czyli uczenie nienadzorowane	233
Klastrowanie grup dyskusyjnych metodą k-średnich	235
Jak działa klastrowanie metodą k-średnich?	235
Implementacja klastrowania metodą k-średnich od podstaw	236
Implementacja klastrowania metodą k-średnich z wykorzystaniem pakietu scikit-learn	241
Dobór wartości k	243
Klastrowanie danych z grup dyskusyjnych	245
Klastrowanie danych z grup dyskusyjnych metodą k-średnich	245
Opisywanie klastrów przy użyciu narzędzia ChatGPT	248
Odkrywanie ukrytych tematów grup dyskusyjnych	251
Modelowanie tematyczne z wykorzystaniem nieujemnej faktoryzacji macierzy	252
Modelowanie tematyczne z wykorzystaniem ukrytej alokacji Dirichleta ...	255
Podsumowanie	258
Ćwiczenia	259

ROZDZIAŁ 9**Rozpoznawanie twarzy przy użyciu maszyny wektorów nośnych 260**

Określanie granic klas za pomocą maszyny wektorów nośnych	260
Scenariusz 1. Określenie hiperpłaszczyzny rozdzielającej	261
Scenariusz 2. Określenie optymalnej hiperpłaszczyzny rozdzielającej	262
Scenariusz 3. Przetwarzanie punktów odstających	265
Implementacja maszyny wektorów nośnych	266
Scenariusz 4. Więcej niż dwie klasy	268
Scenariusz 5. Rozwiązywanie nierozdzielnego liniowo problemu za pomocą jądra	272
Wybór między jądrem liniowym a radialną funkcją bazową	275
Klasyfikowanie zdjęć twarzy za pomocą maszyny wektorów nośnych	278
Badanie zbioru zdjęć twarzy	278
Tworzenie klasyfikatora obrazów opartego na maszynie wektorów nośnych	280
Zwiększanie skuteczności klasyfikatora obrazów za pomocą analizy głównych składowych	281
Szacowanie z użyciem regresji wektorów nośnych	284
Rozwiązanie regresji wektorów nośnych	285
Podsumowanie	287
Ćwiczenia	287

ROZDZIAŁ 10**Dobre praktyki uczenia maszynowego 288**

Proces rozwiązywania problemów uczenia maszynowego	288
Najlepsze praktyki przygotowywania danych	289
Dobra praktyka nr 1. Dokładne poznanie celu projektu	289
Dobra praktyka nr 2. Zbieranie wszystkich istotnych pól	290
Dobra praktyka nr 3. Ujednolicenie danych	290
Dobra praktyka nr 4. Opracowanie niekompletnych danych	291
Dobra praktyka nr 5. Przechowywanie dużych ilości danych	294
Dobre praktyki tworzenia zbioru treningowego	295
Dobra praktyka nr 6. Oznaczanie cech kategoryalnych liczbami	295
Dobra praktyka nr 7. Rozważenie kodowania cech kategoryalnych	295
Dobra praktyka nr 8. Rozważenie selekcji cech i wybór odpowiedniej metody	296
Dobra praktyka nr 9. Rozważenie redukcji wymiarowości i wybór odpowiedniej metody	297
Dobra praktyka nr 10. Rozważenie normalizacji cech	298
Dobra praktyka nr 11. Inżynieria cech na bazie wiedzy eksperckiej	298

Dobra praktyka nr 12. Inżynieria cech bez wiedzy eksperckiej	299
Dobra praktyka nr 13. Dokumentowanie procesu tworzenia cech	300
Dobra praktyka nr 14. Wyodrębnianie cech z danych tekstowych	300
Najlepsze praktyki trenowania, oceniania i wybierania modelu	304
Dobra praktyka nr 15. Wybór odpowiedniego algorytmu początkowego	304
Dobra praktyka nr 16. Zapobieganie nadmiernemu dopasowaniu	307
Dobra praktyka nr 17. Diagnostowanie nadmiernego i niedostatecznego dopasowania	308
Dobra praktyka nr 18. Modelowanie dużych zbiorów danych	310
Dobre praktyki wdrażania i monitorowania modelu	311
Dobra praktyka nr 19. Zapisywanie, ładowanie i wielokrotne stosowanie modelu	311
Dobra praktyka nr 20. Monitorowanie skuteczności modelu	315
Dobra praktyka nr 21. Regularne aktualizowanie modelu	316
Podsumowanie	316
Ćwiczenia	317

ROZDZIAŁ 11

Kategoryzacja zdjęć odzieży przy użyciu konwolucyjnej

sieci neuronowej 318

Prezentacja bloków konstrukcyjnych konwolucyjnej sieci neuronowej	318
Warstwa konwolucyjna	319
Warstwa nieliniowa	321
Warstwa redukująca	321
Budowanie konwolucyjnej sieci neuronowej na potrzeby klasyfikacji	323
Badanie zbioru zdjęć odzieży	324
Klasyfikowanie zdjęć odzieży za pomocą konwolucyjnej sieci neuronowej ...	328
Tworzenie sieci	328
Trening sieci	331
Wizualizacja filtrów konwolucyjnych	334
Wzmacnianie konwolucyjnej sieci neuronowej poprzez uzupełnianie danych	335
Obracanie obrazów w poziomie i pionie	335
Obracanie obrazów	336
Przycinanie obrazów	337
Usprawnianie klasyfikatora obrazów poprzez uzupełnianie danych	338

Rozwijanie klasyfikatora z wykorzystaniem uczenia transferowego	339
Rozwój architektur sieci konwolucyjnych oraz wstępnie wytrenowanych modeli	340
Poprawa klasyfikatora obrazów odzieży poprzez dostrajanie sieci ResNet	342
Podsumowanie	343
Ćwiczenia	343

ROZDZIAŁ 12

Prognozowanie sekwencji danych przy użyciu rekurencyjnej

sieci neuronowej	344
Wprowadzenie do uczenia sekwencyjnego	344
Architektura rekurencyjnej sieci neuronowej na przykładzie	345
Mechanizm rekurencyjny	345
Sieć typu „wiele do jednego”	347
Sieć typu „jedno do wielu”	348
Sieć synchroniczna typu „wiele do wielu”	348
Sieć niesynchroniczna typu „wiele do wielu”	349
Trening rekurencyjnej sieci neuronowej	350
Długoterminowe zależności i sieć LSTM	351
Analiza recenzji filmowych za pomocą rekurencyjnej sieci neuronowej	354
Analiza i wstępne przetworzenie recenzji	354
Zbudowanie prostej sieci LSTM	358
Poprawa skuteczności poprzez wprowadzenie dodatkowych warstw ...	361
Prognozowanie cen akcji przy użyciu sieci LSTM	362
Pisanie nowej powieści <i>Wojna i pokój</i> za pomocą rekurencyjnej sieci neuronowej	365
Pozyskanie i analiza danych treningowych	365
Utworzenie zbioru treningowego dla generatora tekstu	366
Utworzenie generatora tekstu	368
Podsumowanie	372
Ćwiczenia	372

ROZDZIAŁ 13

Ulepszanie rozumienia i generowania tekstów

z wykorzystaniem modeli transformerów	373
Mechanizm samouwagi	373
Reprezentacje klucza, wartości i zapytania	374
Uwaga wielogłowa	379

Prezentacja architektury Transformera	380
Struktura koder-dekoder	381
Kodowanie pozycyjne	382
Normalizacja warstwy	384
Udoskonalanie analizy sentymentu przy użyciu modelu BERT i transformerów	384
Wstępne trenowanie modelu BERT	384
Dostrajanie modelu BERT	386
Dostrajanie wstępnie wytrenowanego modelu BERT do analizy sentymentu	387
Wykorzystanie API Trainer do szkolenia modeli Transformer	391
Generowanie tekstu przy użyciu modelu GPT	394
Wstępne trenowanie GPT i generowanie autoregresywne	394
Tworzenie własnej wersji <i>Wojny i pokoju</i> przy użyciu GPT	395
Podsumowanie	399
Ćwiczenia	399

ROZDZIAŁ 14

Budowanie wyszukiwarki obrazów z wykorzystaniem

modelu CLIP — podejście multimodalne 400

Przedstawiamy model CLIP	400
Zrozumienie mechanizmu działania modelu CLIP	401
Badanie zastosowań modelu CLIP	404
Rozpoczęcie pracy z zestawem danych	406
Pozyskiwanie zbioru danych Flickr8k	407
Wczytywanie zbioru danych Flickr8k	407
Projektowanie modelu CLIP	410
Mechanizm projekcji w uczeniu kontrastowym	412
Wyszukiwanie obrazów za pomocą słów	415
Trenowanie modelu CLIP	415
Generowanie osadzeń dla obrazów i tekstu w celu identyfikacji dopasowań	416
Wyszukiwanie obrazów przy użyciu wstępnie wytrenowanego modelu CLIP	419
Klasyfikacja metodą zero-shot	421
Podsumowanie	425
Ćwiczenia	425
Bibliografia	425

ROZDZIAŁ 15**Podjmowanie decyzji w skomplikowanych warunkach****z wykorzystaniem uczenia przez wzmacnianie 426**

Przygotowanie środowiska do uczenia przez wzmacnianie	426
Wprowadzenie do pakietów narzędziowych Gym i Gymnasium	427
Instalowanie Gymnasium	428
Wprowadzenie do uczenia przez wzmacnianie z przykładami	429
Komponenty uczenia przez wzmacnianie	429
Sumaryczna nagroda	430
Algorytmy uczenia przez wzmacnianie	431
Problem FrozenLake i programowanie dynamiczne	432
Utworzenie środowiska FrozenLake	432
Rozwiązanie problemu przy użyciu algorytmu iteracji wartości	436
Rozwiązanie problemu przy użyciu algorytmu iteracji polityki	440
Metoda Monte Carlo uczenia przez wzmacnianie	443
Utworzenie środowiska Blackjack	443
Ocenianie polityki w metodzie Monte Carlo	445
Sterowanie Monte Carlo z polityką	447
Rozwiązywanie problemu Blackjack przy użyciu algorytmu Q-uczenia	450
Wprowadzenie do algorytmu Q-uczenia	450
Implementacja algorytmu Q-uczenia	451
Podsumowanie	453
Ćwiczenia	453

Skorowidz 454

Kategoryzacja zdjęć odzieży przy użyciu konwolucyjnej sieci neuronowej

W poprzednim rozdziale zostały opisane dobre praktyki ogólnego, tradycyjnego uczenia maszynowego. Począwszy od tego rozdziału zajmiemy się bardziej zaawansowanymi zagadnieniami: uczeniem głębokim i uczeniem przez wzmacnianie.

Klasyfikacja obrazów zwykle polega na ich spłaszczaniu, generowaniu wektorów pikseli i przesyłaniu ich do sieci neuronowej lub innego modelu. W ten sposób można wprawdzie rozwiązać problem, ale traci się cenne informacje przestrzenne. W tym rozdziale będziemy wykorzystywać konwolucyjną sieć neuronową do wyodrębniania z obrazów bogatych, różnorodnych informacji. Dowiesz się, jak sieć rozpoznaje, że obraz cyfry 9 to cyfra 9, 4 to 4, kot to kot, a pies to pies.

Zacniemy od przedstawienia bloków konstrukcyjnych sieci konwolucyjnej. Następnie zbudujemy przy użyciu pakietu TensorFlow sieć klasyfikującą zdjęcia odzieży. W ten sposób zdemistyfikujemy mechanizm działania sieci konwolucyjnej. Na koniec zajmiemy się uzupełnianiem danych i zwiększaniem skuteczności modeli sieci.

W tym rozdziale są opisane następujące zagadnienia:

- prezentacja bloków konstrukcyjnych konwolucyjnej sieci neuronowej,
- budowanie sieci konwolucyjnej do celów klasyfikacji,
- poznawanie zbioru danych ze zdjęciami odzieży,
- klasyfikacja zdjęć odzieży przy użyciu sieci konwolucyjnej,
- wzmacnianie klasyfikatora przy wykorzystaniu uzupełniania danych,
- rozwijanie klasyfikatora z wykorzystaniem uczenia transferowego.

Prezentacja bloków konstrukcyjnych konwolucyjnej sieci neuronowej

Zwykle, w pełni połączone warstwy ukryte, którymi zajmowaliśmy się do tej pory, do pewnego stopnia radzą sobie z wyodrębnianiem cech danych. Jednak uzyskane wyniki nie zawsze pozwalają rozróżniać klasy obrazów. Natomiast za pomocą **konwolucyjnej**

sieci neuronowej (ang. *Convolutional Neural Network*, CNN) można wydobywać bogate i różnorodne informacje, na przykład rozpoznawać samochody, samoloty, ręcznie napisane litery. Sieć konwolucyjna to rodzaj sieci neuronowej, o budowie inspirowanej ludzką korą wzrokową. Aby zdemistyfikować tego rodzaju sieć, zacznijmy od zapoznania się z jej komponentami — warstwami: konwolucyjną, nieliniową i redukującą.

Warstwa konwolucyjna

Pierwszą lub kilka pierwszych warstw sieci konwolucyjnej stanowią **warstwy konwolucyjne**. Danymi wejściowymi są obrazy lub macierze.

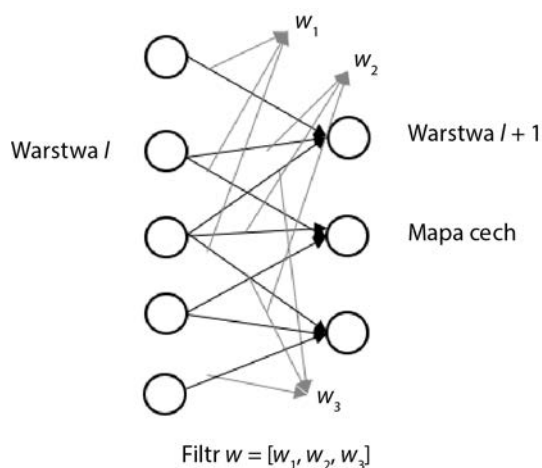
W praktyce konwolucyjna sieć neuronowa, a konkretnie jej warstwy konwolucyjne, funkcjonują podobnie jak komórki wzrokowe:

- Kora wzrokowa człowieka składa się z komórek nerwowych, posiadających wrażliwe podobszary pola widzenia, nazywane **polami recepcyjnymi**. Komórki reagują silniej lub słabiej na linie znajdujące się w polu widzenia w określonym położeniu. Na przykład niektóre komórki reagują tylko na pionowe linie proste, a inne na poziome. Wszystkie komórki razem wzięte tworzą system wzrokowy, przy czym każda komórka jest wyspecjalizowanym komponentem. Warstwa konwolucyjna w sieci zawiera zbiór filtrów funkcjonujących podobnie jak komórki w korze wzrokowej.
- Prosta komórka reaguje tylko wtedy, gdy w jej podobszarze recepcyjnym znajdzie się określona linia. Bardziej złożona komórka jest wrażliwa na większe podobszary i reaguje na linie proste znajdujące się w całym polu widzenia. Stos warstw konwolucyjnych jest zbiorem złożonych komórek, które wspólnie mogą wykrywać wzorce w większym zakresie.

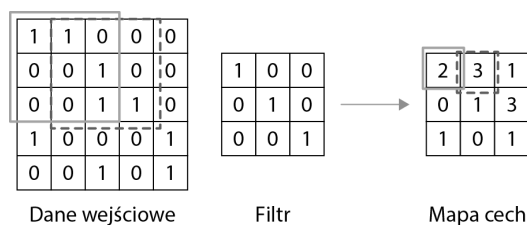
Warstwa konwolucyjna przetwarza przesyłane obrazy lub macierze i funkcjonuje podobnie jak komórki nerwowe, reagujące na sygnały odbierane za pomocą pól recepcyjnych. Przetwarza dane wejściowe, wykonując na nich operacje konwolucyjne, czyli mówiąc językiem matematycznym, wylicza iloczyn skalarny wartości zawartych w swoich węzłach i w poszczególnych małych obszarach warstwy wejściowej. Mały obszar jest polem recepcyjnym, natomiast węzeł można uznać za część filtru. W miarę przemieszczania się filtru przez warstwę wejściową wyliczany jest iloczyn skalarny wartości zawartych w tym filtrze i w bieżącym polu recepcyjnym (podobszarze). Gdy filtr przejdzie przez wszystkie podobszary, tworzona jest nowa warstwa, tzw. **mapa cech**. Rysunek 11.1 ilustruje prosty przykład.

W tym przykładzie pierwsza warstwa składa się z pięciu, a filtr z trzech węzłów [w_1 , w_2 , w_3]. Najpierw jest wyliczany iloczyn skalarny wartości zawartych w filtrze i w pierwszych trzech węzłach pierwszej warstwy. W ten sposób jest tworzony pierwszy węzeł w wyjściowej mapie cech. Następnie jest wyliczany iloczyn skalarny wartości zawartych w filtrze i w trzech środkowych węzłach. Tak powstaje drugi węzeł w mapie cech. Na koniec, na bazie trzech ostatnich węzłów pierwszej warstwy, jest tworzony trzeci węzeł warstwy l — mapy cech.

Teraz przyjrzyjmy się dokładniej procesowi konwolucji, pokazanemu na rysunku 11.2.



Rysunek 11.1. Proces tworzenia mapy cech



Rysunek 11.2. Proces konwolucji

W tym przykładzie filtr o wymiarach 3×3 przemieszcza się po macierzy o wymiarach 5×5 poczynając od lewego górnego podobszaru, w kierunku prawego dolnego. Dla każdego podobszaru jest przy użyciu filtru wyliczany iloczyn skalarny. Na przykład w lewym górnym obszarze (pomarańczowy prostokąt — linia ciągła) jest wykonywane działanie $1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 2$. W efekcie lewy górny węzeł w mapie cech uzyskuje wartość 2. Dla kolejnego podobszaru (niebieski prostokąt — linia przerywana) iloczyn jest równy $1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 3$ i taką wartość uzyskuje następny węzeł (wewnątrz tego prostokąta) w wynikowej mapie cech. Ostatecznie powstaje mapa cech o wymiarach 3×3.

Do czego są wykorzystywane warstwy konwolucyjne? Do wyodrębniania takich cech, jak linie proste i krzywe. Jeżeli pole recepcyjne zawiera linię prostą lub krzywą, rozpoznaną przez filtr, wówczas piksel w wynikowej mapie cech uzyskuje wysoką wartość. W powyższym przykładzie filtr definiuje odwrotny ukośnik ($\backslash$), a w polu recepcyjnym wewnątrz niebieskiego prostokąta znajduje się podobna linia. W efekcie tworzony jest piksel zawierający najwyższą wartość 3. Natomiast pole recepcyjne w prawym górnym rogu nie zawiera ukośnika, więc piksel w wynikowej mapie cech ma wartość 0. Zatem filtr konwolucyjny działa jak detektor krzywych lub kształtów.

Warstwa konwolucyjna zazwyczaj zawiera wiele filtrów wykrywających różne linie krzywe i kształty. W powyższym prostym przykładzie został użyty jeden filtr i utworzona została jedna mapa cech określająca, jak wiernie obraz wejściowy oddaje krzywą

zapisaną w filtrze. Aby wykrywać w danych wejściowych więcej wzorców, stosuje się dodatkowe filtry, definiujące na przykład linie poziomą, pionową, nachyloną pod kątem 30° i prostopadłe do siebie.

Aby uzyskać reprezentacje wyższego poziomu, na przykład podstawowe kształty lub kontury, stosuje się kilka warstw konwolucyjnych. Łącząc większe liczby warstw, uzyskuje się większe pola recepcyjne, umożliwiające rozpoznawanie kolejnych ogólnych kształtów.

Często za każdą warstwą konwolucyjną znajduje się warstwa nieliniowa.

Warstwa nieliniowa

Warstwa nieliniowa jest w zasadzie warstwą aktywacji, którą poznałeś w rozdziale 6., „Prognozowanie cen akcji za pomocą sieci neuronowych”. Oczywiście jej zadaniem jest wprowadzanie nieliniowości. Jak już wiesz, w warstwie konwolucyjnej są wykonywane tylko operacje liniowe, na przykład mnożenie lub dodawanie. Sieć neuronowa zawierająca wyłącznie liniowe warstwy ukryte niezależnie od ich liczby funkcjonuje jak perceptron jednowarstwowy. Dlatego zaraz po warstwie konwolucyjnej niezbędne jest wprowadzanie nieliniowych aktywacji. Jak zawsze najlepszą potencjalną funkcją nieliniową w głębokiej sieci neuronowej jest ReLU.

Warstwa redukująca

Zazwyczaj jedna lub kilka warstw konwolucyjnych (wraz z aktywacją nieliniową) generuje cechy, które można bezpośrednio wykorzystywać w klasyfikacji. Na przykład na potrzeby klasyfikacji wieloklasowej można zastosować funkcję softmax. Wykonajmy jednak najpierw kilka obliczeń matematycznych.

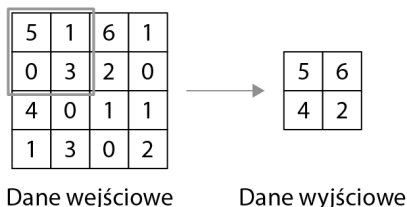
Żałómy, że obraz wejściowy ma wymiary 28×28 , a pierwsza warstwa konwolucyjna zawiera 20 filtrów, każdy o wymiarach 5×5 . Uzyskujemy wtedy 20 map cech, każda o wielkości $(28 - 5 + 1) \cdot (28 - 5 + 1) = 24 \cdot 24 = 576$. Oznacza to, że liczba cech stanowiących dane wejściowe dla następnej warstwy wzrasta z $28 \cdot 28 = 784$ do $20 \cdot 576 = 11\,520$. Jeżeli w drugiej warstwie konwolucyjnej będzie 20 filtrów, każdy o wymiarach 5×5 , to liczba cech wyjściowych wzrasta do $50 \cdot 20 \cdot (24 - 5 + 1) \cdot (24 - 5 + 1) = 400\,000$, czyli znacznie więcej od początkowej liczby 784. Wymiarowość danych gwałtownie rośnie wraz z każdą warstwą konwolucyjną aż do końcowej warstwy softmax. Jest to niekorzystny efekt, ponieważ model może łatwo nadmiernie się dopasować do danych, nie mówiąc już o kosztach jego treningu przy tak dużej liczbie wag.

Aby rozwiązać ten problem, często po warstwach konwolucyjnej i nieliniowej wprowadza się **warstwę redukującą** (ang. *pooling layer*), nazywaną również **warstwą próbkującą w dół** (ang. *downsampling layer*). Jak sugeruje nazwa, warstwa ta zmniejsza wymiarowość mapy cech. Agreguje w tym celu informacje o cechach znajdujących się w poszczególnych podobszarach. Typowe techniki redukcji są następujące:

- maksymalizacja, polegająca na wybieraniu maksymalnych wartości zawartych w nienakładających się podobszarach;

- uśrednianie, polegające na uśrednieniu wartości zawartych w nienakładających się podobszarach.

W przykładzie pokazanym na rysunku 11.3 zastosowany jest filtr maksymalizujący o wymiarach 2×2, przekształcający mapę o wymiarach 4×4 na wynikową mapę 2×2.



Dane wejściowe

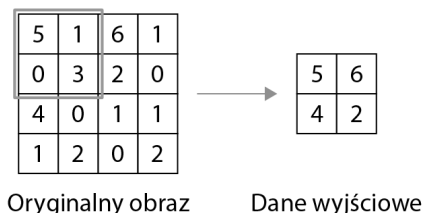
Dane wyjściowe

Rysunek 11.3. Zasada funkcjonowania warstwy redukującej

Dodatkową zaletą warstwy redukującej, oprócz redukcji wymiarowości, jest niezmienność translacji. Oznacza to, że wyjście nie zmienia się nawet wtedy, gdy macierz wejściowa ulega niewielkim przekształceniom. Na przykład jeśli obraz wejściowy zostanie przesunięty o kilka pikseli w lewo lub w prawo, dane wyjściowe maksymalizującej warstwy redukującej pozostaną niezmiennie, o ile wartości pikseli w poszczególnych podobszarach będą takie same. Innymi słowy, prognozy są wtedy mniej wrażliwe na położenie obrazów.

Rysunek 11.4 ilustruje niezmienność translacji w warstwie redukującej. Przedstawia obraz o wymiarach 4×4 i wyjście warstwy redukującej o wymiarach 2×2.

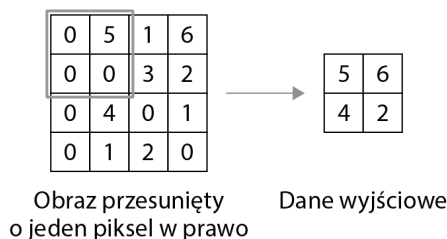
Rysunek 11.5 pokazuje wynik uzyskany po przesunięciu obrazu o jeden piksel w prawo.



Oryginalny obraz

Dane wyjściowe

Rysunek 11.4. Przesunięty oryginalny obraz i wyjście maksymalizującej warstwy redukującej

Obraz przesunięty
o jeden piksel w prawo

Dane wyjściowe

Rysunek 11.5. Przesunięty obraz i uzyskany wynik

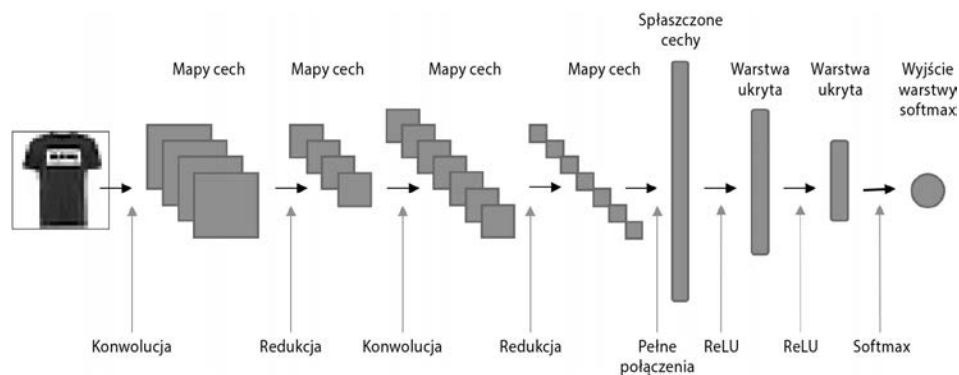
Wynik jest taki sam, mimo że obraz został przesunięty w poziomie. W ten sposób warstwa redukująca poprawia stabilność translacji obrazów.

Poznałeś bloki konstrukcyjne konwulucyjnej sieci neuronowej. Są prostsze, niż przypuszczałeś, prawda? Teraz dowiedz się, jak buduje się z nich sieć konwulucyjną.

Budowanie konwulucyjnej sieci neuronowej na potrzeby klasyfikacji

Wykorzystując trzy opisane wcześniej typy warstw oraz warstwy w pełni połączone, można zbudować strukturę modelu klasyfikacyjnego, opartego na konwulucyjnej sieci neuronowej. Ilustruje to rysunek 11.6.

W tym przykładzie obrazy wejściowe są wprowadzane do złożonej z kilku filtrów warstwy konwulucyjnej, wykorzystującej aktywację ReLU. Współczynniki filtrów mogą być modyfikowane podczas treningu. Dobrze wytrenowana pierwsza warstwa konwulucyjna dostarcza dobrych niskopoziomowych reprezentacji obrazów wejściowych, co ma krytyczne znaczenie dla kolejnych warstw konwulucyjnych (jeżeli są), jak również dalszych zadań klasyfikacyjnych. Każda wynikowa mapa cech jest następnie próbkowana w dół przez warstwę redukującą.



Rysunek 11.6. Architektura konwulucyjnej sieci neuronowej

Zagregowane mapy cech są wprowadzane do drugiej warstwy konwulucyjnej. Podobnie jak poprzednio, druga warstwa redukująca zmniejsza rozmiar wyjściowej mapy cech. Można tworzyć dowolnie długi łańcuch par zawierających warstwy konwulucyjną i redukującą. Druga warstwa konwulucyjna (i następne, jeżeli są) próbuje generować na podstawie serii niskopoziomowych reprezentacji uzyskanych z poprzednich warstw wysoko poziomowe reprezentacje, takie jak kształty lub kontury.

Mapy cech są macierzami. Dlatego przed klasyfikacją trzeba je spłaszczyć do wektorów. Spłaszczone cechy są danymi wejściowymi dla jednej lub kilku w pełni połączonych warstw ukrytych. Konwulucyjną sieć neuronową można traktować jako hierarchicznego ekstraktora cech, opartego na zwykłej sieci neuronowej. Sieć konwulucyjna jest dobrze przystosowana do identyfikowania silnych, unikatowych cech wyróżniających obrazy.

Na końcu sieci znajduje się funkcja logistyczna (w przypadku klasyfikacji binarnej), funkcja softmax (w przypadku klasyfikacji wieloklasowej) lub zbiór funkcji logistycznych (w przypadku wielu etykiet).

W tym momencie powinieneś już dobrze rozumieć funkcjonowanie konwolucyjnej sieci neuronowej i być gotowy do rozwiązania problemu klasyfikacji zdjęć odzieży. Zaczniemy od zbadania zbioru danych.

Badanie zbioru zdjęć odzieży

Zbiór Fashion-MNIST (<https://github.com/zalando-research/fashion-mnist>) zawiera zdjęcia produktów marki Zalando — największego w Europie internetowego sklepu odzieżowego. Zbiór składa się z 60 000 próbek treningowych i 10 000 testowych. Każda próbka jest obrazem o wymiarach 28×28 pikseli w odcieniach szarości. Obrazowi jest przypisana jedna z następujących 10 klas odzieży:

- 0: t-shirt/top,
- 1: spodnie,
- 2: sweter,
- 3: sukienka,
- 4: płaszcz,
- 5: sandały,
- 6: koszula,
- 7: tenisówki,
- 8: torebka,
- 9: buty.

Firma Zalando nazwała swój zbiór Fashion-MNIST, aby stał się równie popularny jak wykorzystywany do analiz porównawczych zbiór MNIST odręcznie napisanych cyfr.

Zbiór ten możesz łatwo pobrać, klikając odnośnik *Download* na stronie serwisu GitHub lub korzystając z modułu PyTorch, który oprócz zbioru zawiera odpowiedni interfejs API. W poniższym przykładzie wykorzystana jest druga możliwość.

```
>>> import torch, torchvision
>>> from torchvision import transforms
>>> image_path = './'
>>> transform = transforms.Compose([transforms.ToTensor()])
>>> train_dataset = torchvision.datasets.FashionMNIST(root=image_path,
...                                                  train=True,
...                                                  transform=transform,
...                                                  download=True)
>>> test_dataset = torchvision.datasets.FashionMNIST(root=image_path,
...                                                  train=False,
...                                                  transform=transform,
...                                                  download=False)
```

W powyższym przykładzie zaimportowałeś pakiet torchvision, część biblioteki PyTorch zapewniającą dostęp do zbiorów danych, architektur modeli oraz przeróżnych narzędzi

do przekształcania obrazów służących do wykonywania zadań związanych z wizją komputerową.

Uwaga

Biblioteka `torchvision` zawiera następujące kluczowe elementy:

- **Zbiory danych i narzędzia do ładowania danych:** Pakiet `torchvision.datasets` umożliwia dostęp do standardowych zbiorów danych wykorzystywanych w zadaniach takich jak klasyfikacja obrazów, detekcja obiektów, segmentacja semantyczna. Przykładowe zbiory danych to: MNIST, CIFAR-10, ImageNet, FashionMNIST. Klasa `torch.utils.data.DataLoader` ułatwia tworzenie obiektów wczytujących dane, które wydajnie ładują i wstępnie przetwarzają partie danych ze zbiorów danych.
- **Przekształcenia:** Moduł `torchvision.transforms` oferuje szeroki wybór narzędzi do przekształcania obrazów, służących do wzbogacania danych, normalizacji i wstępnego przetwarzania. Do najczęściej używanych przekształceń należą zmiana rozmiaru, przycinanie i normalizacja.
- **Architektury modeli:** Moduł `torchvision.models` udostępnia wstępnie wytrenowane architektury modeli przeznaczone do różnych zadań z zakresu komputerowego rozpoznawania obrazów.
- **Narzędzia pomocnicze:** Moduł `torchvision.utils` zawiera funkcje pomocnicze, które umożliwiają wizualizację obrazów, konwersję obrazów na różne formaty i wykonywanie innych operacji.

Zbiór danych Fashion-MNIST, który właśnie wczytałeś, zawiera gotowy sposób podziału danych na zbiory treningowy i testowy. Zbiór treningowy jest przechowywany w miejscu określonym przez zmienną `image_path`. Kolejnym krokiem będzie przekształcanie tych zbiorów do formatu Tensor. Wyniki zwracane przez te dwa obiekty zbiorów danych zawierają dodatkowe informacje na ich temat:

```
>>> print(train_dataset)
Dataset FashionMNIST
  Number of datapoints: 60000
  Root location: ./
  Split: Train
  StandardTransform
Transform: Compose(
  ToTensor()
)
>>> print(test_dataset)
Dataset FashionMNIST
  Number of datapoints: 10000
  Root location: ./
  Split: Test
  StandardTransform
Transform: Compose(
  ToTensor()
)
```

Jak widać, składają się one z 60 000 próbek treningowych i 10 000 próbek testowych.

Teraz wczytaj zbiór testowy we fragmentach zawierających po 64 próbki:

```
>>> from torch.utils.data import DataLoader
>>> batch_size = 64
>>> torch.manual_seed(42)
>>> train_dl = DataLoader(train_dataset, batch_size, shuffle=True)
```

W bibliotece PyTorch `DataLoader` to klasa narzędziowa, która umożliwia efektywne ładowanie i wstępne przetwarzanie danych z zestawu danych podczas treningu lub ewaluacji modeli uczenia maszynowego. W praktyce `DataLoader` stanowi opakowanie dla zestawu danych i udostępnia metody pozwalające na iterowanie po partiach danych. Jest to szczególnie przydatne podczas pracy z dużymi zbiorami danych, które nie mieszczą się w całości w pamięci operacyjnej.

Uwaga

Kluczowe cechy klasy `DataLoader`:

- **Podział na partie:** Potrafi automatycznie dzielić zestaw danych na partie o określonej wielkości, pozwalając na ich stosowanie w procesie trenowania modeli.
- **Losowa zmiana kolejności:** Przypisanie parametrowi `shuffle` wartości `True` zapewnia możliwość losowej zmiany kolejności danych przed rozpoczęciem każdego treningu, co pozwala zredukować obciążenie i poprawić konwergencję

Jeśli chcesz, sprawdź próbki obrazów oraz ich etykiety z pierwszej partii danych:

```
>>> data_iter = iter(train_dl)
>>> images, labels = next(data_iter)
>>> print(labels)
tensor([5, 7, 4, 7, 3, 8, 9, 5, 3, 1, 2, 3, 2, 3, 3, 7, 9, 9, 3, 2, 4, 6, 3, 5, 5,
        3, 2, 0, 0, 8, 4, 2, 8, 5, 9, 2, 4, 9, 4, 4, 3, 4, 9, 7, 2, 0, 4, 5, 4, 8, 2, 6,
        7, 0, 2, 0, 6, 3, 3, 5, 6, 0, 0, 8])
```

Etykiety nie są nazwami klas. Zdefiniuj je więc jak niżej. Wykorzystasz je później do wyświetlenia wyników:

```
>>> class_names = ['T-shirt/top', 'Spodnie', 'Sweter', 'Sukienka', 'Płaszcz',
                   'Sandały', 'Koszula', 'Tenisówki', 'Torebka', 'Buty']
```

Sprawdź format obrazów:

```
>>> print(images[0].shape)
torch.Size([1, 28, 28])
>>> print(torch.max(images), torch.min(images))
tensor(1.) tensor(0.)
```

Każda próbka jest obrazem o wymiarach 28×28 pikseli.

Wyświetl próbkę treningową za pomocą poniższego kodu:

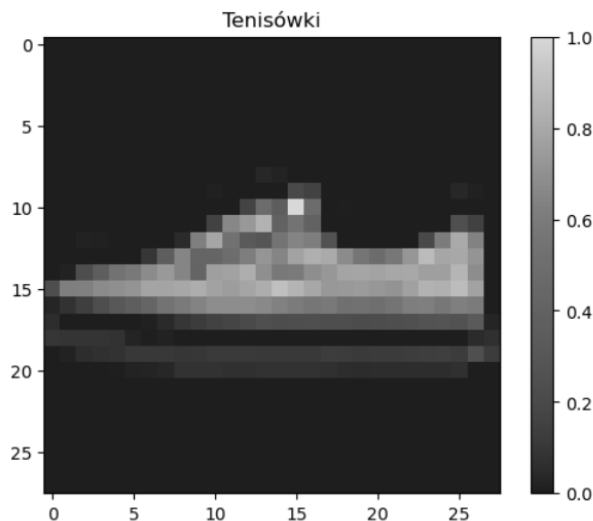
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> npimg = images[1].numpy()
>>> plt.imshow(np.transpose(npimg, (1, 2, 0)))
```

```
>>> plt.colorbar()
>>> plt.title(class_names[labels[1]])
>>> plt.show()
```

Uwaga

W bibliotece PyTorch podczas wyświetlania obrazów przy użyciu matplotlib należy zastosować wywołanie: `np.transpose(npimg, (1, 2, 0))`. Krotka `(1, 2, 0)` określa nową kolejność wymiarów. W PyTorch obrazy są reprezentowane w formacie: kanały, wysokość, szerokość. Wywołanie `np.transpose(npimg, (1, 2, 0))` powoduje zmianę kolejności wymiarów tablicy obrazu, tak by odpowiadały one wymaganiom biblioteki matplotlib.

Rysunek 11.7 przedstawia uzyskany wynik.



Rysunek 11.7. Próbką treningowa ze zbioru Fashion-MNIST

W podobny sposób możesz wyświetlić pierwszych 16 próbek treningowych:

```
>>> for i in range(16):
...     plt.subplot(4, 4, i + 1)
...     plt.subplots_adjust(hspace=.3)
...     plt.xticks([])
...     plt.yticks([])
...     npimg = images[i].numpy()
...     plt.imshow(np.transpose(npimg, (1, 2, 0)), cmap="Greys")
...     plt.title(class_names[labels[i]])
>>> plt.show()
```

Rysunek 11.8 przedstawia uzyskany wynik.

W następnym podrozdziale zbudujesz model sieci konwolucyjnej klasyfikującej zdjęcia odzieży.



Rysunek 11.8. Wynik wstępnego przetworzenia zbioru treningowego

Klasyfikowanie zdjęć odzieży za pomocą konwolucyjnej sieci neuronowej

Jak wspominałem wcześniej, model konwolucyjnej sieci neuronowej składa się z dwóch komponentów: ekstraktora cech zbudowanego z warstw konwolucyjnych i redukujących oraz klasyfikatora podobnego do zwykłej sieci neuronowej.

Zacznijmy ten projekt od przygotowania modelu konwolucyjnej sieci neuronowej.

Tworzenie sieci

Zaimportuj niezbędny moduł i zainicjuj model Sequential:

```
>>> import torch.nn as nn
>>> model = nn.Sequential()
```

Ekstraktor cech będzie się składał z trzech warstw konwolucyjnych. Pierwsza niech zawiera 32 niewielkie filtry, każdy o wymiarach 3×3. Zaimplementuj ją za pomocą poniższego kodu:

```
>>> model.add_module('conv1',
...                     nn.Conv2d(in_channels=1,
...                               out_channels=32,
...                               kernel_size=3)
...                     )
>>> model.add_module('relu1', nn.ReLU())
```

Zwróć uwagę na użytą funkcję aktywacji ReLU.

Po warstwie konwolucyjnej będzie się znajdować maksymalizująca warstwa redukująca, zawierająca filtr o wymiarach 2×2:

```
>>> model.add_module('pool1', nn.MaxPool2d(kernel_size=2))
```

Teraz dodaj drugą warstwę konwolucyjną, zawierającą 64 filtry, każdy o wymiarach 3×3 oraz — jak poprzednio — funkcję aktywacji ReLU:

```
>>> model.add_module('conv2',
...                     nn.Conv2d(in_channels=32,
...                               out_channels=64,
...                               kernel_size=3)
...                     )
>>> model.add_module('relu2', nn.ReLU())
```

Po drugiej warstwie konwolucyjnej umieść następną maksymalizującą warstwę redukującą, zawierającą filtr o wymiarach 2×2:

```
>>> model.add_module('pool2', nn.MaxPool2d(kernel_size=2))
```

Następnie dodaj trzecią warstwę konwolucyjną, tym razem zawierającą 128 filtrów, każdy o wymiarach 3×3:

```
>>> model.add_module('conv3',
...                     nn.Conv2d(in_channels=64,
...                               out_channels=128,
...                               kernel_size=3)
...                     )
>>> model.add_module('relu3', nn.ReLU())
```

Zróbmy sobie w tym miejscu krótką przerwę, by sprawdzić, jak wyglądają uzyskane mapy filtrów. W tym celu wczytaj do modelu grupę 64 losowo wybranych próbek:

```
>>> x = torch.rand((64, 1, 28, 28))
>>> print(model(x).shape)
torch.Size([64, 128, 3, 3])
```

W tym przypadku przekazałeś dane wejściowe o kształcie określonym jako (64, 1, 28, 28), czyli jedną grupę 64 obrazów o wymiarach 28×28, i uzyskałeś wyniki o kształcie (64, 128, 3, 3), czyli mapy cech zawierające 128 kanałów o wymiarach 3×3.

Następnie musisz spłaszczyć uzyskaną mapę cech, aby wprowadzić je do klasyfikatora, który znajdzie się w dalszej części sieci:

```
>>> model.add_module('flatten', nn.Flatten())
```

W efekcie spłaszczyłeś do wymiarów (64, 1152), jak widać poniżej:

```
>>> print(model(x).shape)
torch.Size([64, 1152])
```

Jako klasyfikatora użyj ukrytej warstwy składającej się z 64 węzłów:

```
>>> model.add_module('fc1', nn.Linear(1152, 64))
>>> model.add_module('relu4', nn.ReLU())
```

Warstwa ukryta jest zwykłą, w pełni połączoną warstwą zagęszczoną, wykorzystującą funkcję aktywacji ReLU.

Ostatnia warstwa, generująca wyniki, niech składa się z 10 węzłów reprezentujących 10 klas. Funkcją aktywacji niech będzie softmax:

```
>>> model.add_module('fc2', nn.Linear(64, 10))
>>> model.add_module('output', nn.Softmax(dim = 1))
```

Przjrzyjmy się architekturze utworzonego modelu:

```
>>> print(model)
Sequential(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (relu1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (relu2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (relu3): ReLU()
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=1152, out_features=64, bias=True)
  (relu4): ReLU()
  (fc2): Linear(in_features=64, out_features=10, bias=True)
  (output): Softmax(dim=1)
)
```

Jeśli będziesz chciał wyświetlić szczegółowe informacje na temat każdej z warstw tego modelu, w tym kształt danych wyjściowych każdej z warstw i liczbę ich parametrów, które można trenować, możesz to zrobić za pomocą biblioteki torchsummary. Zainstaluj ją, używając pip, jak pokazano poniżej:

```
>>> pip install torchsummary
>>> from torchsummary import summary
>>> summary(model, input_size=(1, 28, 28), batch_size=-1, device="cpu")
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 26, 26]	320
ReLU-2	[-1, 32, 26, 26]	0
MaxPool2d-3	[-1, 32, 13, 13]	0
Conv2d-4	[-1, 64, 11, 11]	18,496
ReLU-5	[-1, 64, 11, 11]	0
MaxPool2d-6	[-1, 64, 5, 5]	0
Conv2d-7	[-1, 128, 3, 3]	73,856

ReLU-8	[-1, 128, 3, 3]	0
Flatten-9	[-1, 1152]	0
Linear-10	[-1, 64]	73,792
ReLU-11	[-1, 64]	0
Linear-12	[-1, 10]	650
Softmax-13	[-1, 10]	0

```

=====
Total params: 167,114
Trainable params: 167,114
Non-trainable params: 0
-----
Input size (MB): 0.00
Forward/backward pass size (MB): 0.53
Params size (MB): 0.64
Estimated Total Size (MB): 1.17
-----

```

Powyższy wynik zawiera informacje o poszczególnych warstwach modelu, kształtach danych wyjściowych każdej z nich oraz o liczbach trenowanych parametrów. Z pewnością zauważyłeś, że wyjście warstwy konwulucyjnej ma trzy wymiary. Pierwsze dwa są wymiarami mapy cech, a trzecim jest liczba użytych filtrów. Wielkość wyjścia (pierwsze dwa wymiary) maksymalizującej warstwy redukującej jest równa połowie wielkości wejściowej mapy cech. Mapy cech są próbkowane w dół za pomocą warstwy redukującej. Zapewne chciałbyś wiedzieć, ile parametrów musiałbyś trenować, gdyby nie było warstw redukujących. Liczba ta wyniosłaby 4 058 314! Jak widać, korzyści ze stosowania warstw redukujących są oczywiste: zapobiega się nadmiernemu dopasowaniu modelu i zmniejsza koszt treningu.

Być może zastanawiasz się, dlaczego kolejne warstwy konwulucyjne zawierają coraz więcej filtrów. Przypomnę, że każda warstwa usiłuje identyfikować hierarchiczne wzorce danych. W pierwszej są to linie proste, kropki i linie krzywe. Każda kolejna warstwa łączy wzorce uzyskane z warstwy poprzedniej i tworzy wzorce wyższego poziomu, na przykład kształty i kontury. W większości przypadków w kolejnych warstwach kombinacji wzorców jest coraz więcej. Dlatego liczba filtrów też musi rosnąć (lub w ostateczności nie może się zmniejszać).

Trening sieci

Czas wytrenować zbudowany model sieci.

Najpierw skompiluj model, używając optymalizatora Adam, entropii krzyżowej jako funkcji straty i dokładności klasyfikacji jako metryki:

```

>>> device = torch.device("cuda:0")
# device = torch.device("cpu")
>>> model = model.to(device)
>>> loss_fn = nn.CrossEntropyLoss()
>>> optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

```

W tym przypadku do trenowania zastosowaliśmy procesor graficzny (GPU) — z tego względu użyte zostało wywołanie `torch.device("cuda:0")` (które odwołuje się do pierwszego urządzenia, o indeksie 0) — i na nim zapisaliśmy tensor. Trenowanie

modelu na procesorze głównym (CPU) jest alternatywnym rozwiązaniem — zadziała, lecz będzie wolniejsze.

Następnie wytrenuj model, używając do tego celu następującej funkcji:

```
>>> def train(model, optimizer, num_epochs, train_dl):
...     for epoch in range(num_epochs):
...         loss_train = 0
...         accuracy_train = 0
...         for x_batch, y_batch in train_dl:
...             x_batch = x_batch.to(device)
...             y_batch = y_batch.to(device)
...             pred = model(x_batch)
...             loss = loss_fn(pred, y_batch)
...             loss.backward()
...             optimizer.step()
...             optimizer.zero_grad()
...             loss_train += loss.item() * y_batch.size(0)
...             is_correct = (torch.argmax(pred, dim=1) == y_batch).float()
...             accuracy_train += is_correct.sum().cpu()
...
...         loss_train /= len(train_dl.dataset)
...         accuracy_train /= len(train_dl.dataset)
...
...         print(f'Epoka {epoch+1} - strata: {loss_train:.4f} - dokładność:
{accuracy_train:.4f}')
```

Wytrenuj model w 30 iteracjach, monitorując przy tym postępy procesu uczenia

```
>>> num_epochs = 30
>>> train(model, optimizer, num_epochs, train_dl)
Epoka 1 - strata: 1.7253 - dokładność: 0.7385
Epoka 2 - strata: 1.6333 - dokładność: 0.8287
...
Epoka 10 - strata: 1.5572 - dokładność: 0.9041
...
Epoka 20 - strata: 1.5344 - dokładność: 0.9270
...
Epoka 29 - strata: 1.5249 - dokładność: 0.9362
Epoka 30 - strata: 1.5249 - dokładność: 0.9363
```

Dokładność na zbiorze treningowym wyniosła 94%. Kolejny przykład pokazuje, jak można sprawdzić dokładność uzyskiwaną na zbiorze testowym:

```
>>> test_dl = DataLoader(test_dataset, batch_size, shuffle=False)
>>> def evaluate_model(model, test_dl):
...     accuracy_test = 0
...     with torch.no_grad():
...         for x_batch, y_batch in test_dl:
...             pred = model.cpu()(x_batch)
...             is_correct = torch.argmax(pred, dim=1) == y_batch
...             accuracy_test += is_correct.float().sum().item()
...
...     print(f'Dokładność na zbiorze testowym: {100 * accuracy_test / 10000} %')
>>> evaluate_model(model, test_dl)
Dokładność na zbiorze testowym: 90.25 %
```

Na zbiorze testowym model osiągnął dokładność 90%. Zwróć uwagę, że ten wynik może się zmieniać w zależności od takich czynników jak różnice w inicjalizacji warstw ukrytych lub niedeterministyczny sposób działania procesora graficznego.

Najlepsze praktyki

Operacje, które lepiej nadają się do wykonywania na procesorze graficznym niż na procesorze głównym, to zazwyczaj zadania, które można realizować równolegle i które korzystają z ogromnej równoległości oraz mocy obliczeniowej oferowanej przez architekturę GPU. Oto kilka przykładów:

- Operacje macierzowe oraz konwolucyjne.
- Przetwarzanie dużych partii danych jednocześnie. Zadania obejmujące przetwarzanie wsadowe, takie jak uczenie i wnioskowanie na fragmentach danych w modelach uczenia maszynowego, korzystają z równoległego przetwarzania, które można realizować na procesorach graficznych.

Propagacja w przód i wstecz w sieciach neuronowych, która zwykle, dzięki akceleracji sprzętowej, przebiega szybciej na procesorach graficznych.

Najlepsze praktyki

Operacje, które lepiej wykonywać na procesorze głównym niż na procesorze graficznym, to zazwyczaj zadania trudne do zrównoleglenia, wymagające przetwarzania sekwencyjnego lub dotyczące niewielkich ilości danych. Oto kilka przykładów:

- Wstępne przetwarzanie, takie jak ładowanie danych, ekstrakcja cech oraz uzupełnianie danych.
 - Wnioskowanie na małych modelach. W przypadku małych modeli lub zadań inferencyjnych o niskich wymaganiach obliczeniowych wykonywanie operacji na procesorze głównym może być bardziej opłacalne.
 - Operacje związane z przepływem sterowania. Operacje obejmujące instrukcje warunkowe lub pętle są zazwyczaj bardziej efektywne na procesorze głównym ze względu na jego sekwencyjny charakter przetwarzania.
-

Wiesz już, jak działa wytrenowany model, i zapewne zastanawiasz się, jak wyglądają jego filtry konwolucyjne. Dowiesz się tego w następnym punkcie.

Wizualizacja filtrów konwolucyjnych

Aby wyodrębnić filtry konwolucyjne z wytrenowanego modelu i zwizualizować je, wykonaj poniższe kroki:

1. Na podstawie podsumowania modelu wiemy, że warstwami konwolucyjnymi naszego modelu są warstwy: conv1, conv2 i conv3. Wyodrębnij filtry na przykład z trzeciej warstwy konwolucyjnej:

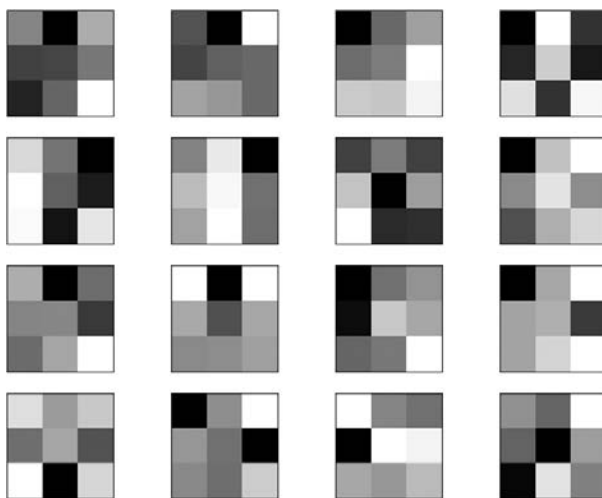
```
>>> conv3_weight = model.conv3.weight.data
>>> print(conv3_weight.shape)
torch.Size([128, 64, 3, 3])
```

Wyraźnie widać, że istnieje 128 filtrów, a każdy z nich ma wymiary 3×3 i zawiera 64 kanały.

2. Dla uproszczenia zwizualizuj jedynie pierwszy kanał pierwszych 16 filtrów, przedstawiając je w układzie czterech wierszy i czterech kolumn:

```
>>> n_filters = 16
>>> for i in range(n_filters):
...     weight = conv3_weight[i].cpu().numpy()
...     plt.subplot(4, 4, i+1)
...     plt.xticks([])
...     plt.yticks([])
...     plt.imshow(weight[0], cmap='gray')
... plt.show()
```

Rysunek 11.9 przedstawia uzyskany wynik.



Rysunek 11.9. Wytrenowane filtry konwolucyjne

Ciemniejsze i jaśniejsze odcienie reprezentują, odpowiednio, mniejsze i większe wagi. Można się domyślić, że filtr w drugim wierszu i drugiej kolumnie identyfikuje pionowe linie znajdujące się w polu recepcyjnym, natomiast trzeci filtr w pierwszym wierszu identyfikuje przejścia koloru z jasnego w prawym dolnym rogu do ciemnego w lewym górnym.

Ten przykład polegał na wytrenowaniu klasyfikatora zdjęć odzieży na 60 000 próbek opatrzonych etykietami. W rzeczywistości uzyskanie tak dużego, oznaczonego zbioru nie jest łatwe. Szczególnie opatrywanie próbek etykietami jest kosztowne i czasochłonne. Jak można trenować klasyfikator na ograniczonej liczbie próbek? Jednym z rozwiązań jest uzupełnianie danych.

Wzmacnianie konwolucyjnej sieci neuronowej poprzez uzupełnianie danych

Uzupełnianie danych (ang. *data augmentation*) oznacza powiększanie istniejącego treningowego zbioru danych w celu zwiększenia skuteczności modelu. Unika się w ten sposób kosztów gromadzenia dodatkowych danych i opatrywania ich etykietami. W bibliotece PyTorch uzupełnianie obrazów w czasie rzeczywistym można implementować przy użyciu modułu `torchvision.transforms`.

Obracanie obrazów w poziomie i pionie

Istnieje wiele sposobów uzupełniania danych obrazów. Najprostszy polega na obracaniu ich w poziomie lub pionie. Na przykład uzyskamy nowy obrazek poprzez utworzenie odbicia istniejącego obrazka w poziomie. Aby stworzyć takie odbicie obrazka w poziomie, wystarczy użyć funkcji `transforms.functional.hflip`, jak pokazano poniżej:

```
>>> image = images[1]
>>> img_flipped = transforms.functional.hflip(image)
```

A oto jak można wyświetlić ten nowy obraz:

```
>>> def display_image_greys(image):
...     npimg = image.numpy()
...     plt.imshow(np.transpose(npimg, (1, 2, 0)), cmap="Greys")
...     plt.xticks([])
...     plt.yticks([])
...
>>> plt.figure(figsize=(8, 8))
>>> plt.subplot(1, 2, 1)
>>> display_image_greys(image)
>>> plt.subplot(1, 2, 2)
>>> display_image_greys(img_flipped)
>>> plt.show()
```

Rysunek 11.10 przedstawia uzyskany wynik.

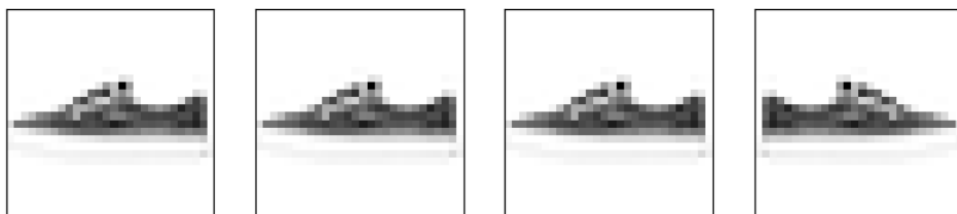


Rysunek 11.10. Uzupełnianie obrazów poprzez obrócenie w poziomie

Podczas trenowania modeli z wykorzystaniem uzupełniania danych będziemy tworzyć zmodyfikowane obrazy, używając do tego celu generatora losowego. W przypadku obrotów w poziomie skorzystamy z funkcji `transforms.RandomHorizontalFlip`, która losowo obraca obrazek w poziomie z prawdopodobieństwem 50%, co pozwala na efektywne uzupełnienie zbioru danych. Przyjrzyjmy się trzem przykładom:

```
>>> torch.manual_seed(42)
>>> flip_transform = transforms.Compose([transforms.RandomHorizontalFlip()])
>>> plt.figure(figsize=(10, 10))
>>> plt.subplot(1, 4, 1)
>>> display_image_greys(image)
>>> for i in range(3):
...     plt.subplot(1, 4, i+2)
...     img_flip = flip_transform(image)
...     display_image_greys(img_flip)
```

Wyniki powyższego kodu zostały przedstawione na rysunku 11.11.



Rysunek 11.11. Losowe obracanie obrazka w poziomie w celu uzupełniania danych

Jak widać, wygenerowane obrazy są odwrócone lub nie są.

Zazwyczaj obrazy odwrócone w poziomie zawierają te same informacje, co obrazy oryginalne. Odwracanie w pionie stosuje się rzadko, choć można to robić za pomocą klasy `transforms.RandomVerticalFlip`. Należy pamiętać, że odwracanie jest pomocne tylko w modelach niewrażliwych na orientację, na przykład klasyfikujących zdjęcia kotów i psów lub rozpoznających części samochodów. Natomiast jest bardzo niebezpieczne w sytuacjach, gdy orientacja jest ważna, na przykład w klasyfikowaniu znaków skrętu w lewo lub w prawo.

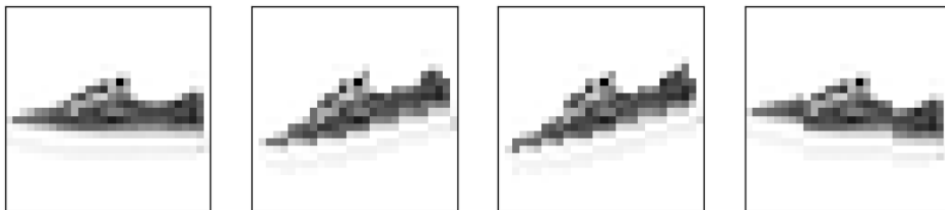
Obracanie obrazów

Obrazy można uzupełniać, nie tylko odwracając je w poziomie i pionie, ale również obracając o niewielki kąt. Wypróbuj to na przykładzie losowego obracania o losowy kąt przy użyciu modułu `transforms`:

```
>>> torch.manual_seed(42)
>>> rotate_transform =
...     transforms.Compose([transforms.RandomRotation(20)])
>>> plt.figure(figsize=(10, 10))
>>> plt.subplot(1, 4, 1)
>>> display_image_greys(image)
```

```
>>> for i in range(3):
...     plt.subplot(1, 4, i+2)
...     img_flip = rotate_transform(image)
...     display_image_greys(img_flip)
```

Rysunek 11.12 przedstawia uzyskany wynik.



Rysunek 11.12. Uzupełnianie obrazów poprzez obracanie

W tym przykładzie oryginalny obraz był obracany o kąty z przedziału od -20° do 20° (odpowiednio, przeciwnie i zgodnie z ruchem wskazówek zegara).

Przycinanie obrazów

Przycinanie obrazów jest kolejną często stosowaną techniką uzupełniania danych. Pozwala ona generować nowe obrazy poprzez wybieranie fragmentów oryginalnego obrazu. Zazwyczaj proces ten jest uzupełniany o przeskalowanie przyciętego obrazka do z góry określonych wymiarów wyjściowych w celu zagwarantowania jednolitych wymiarów obrazów w całym zbiorze danych.

Zobaczmy teraz, w jaki sposób można zastosować tę technikę, korzystając z klasy `transforms.RandomResizedCrop`, aby losowo wybierać proporcje wycinanego fragmentu obrazka, a następnie przeskalować go ponownie do wymiarów oryginału:

```
>>> torch.manual_seed(42)
>>> crop_transform = transforms.Compose([
...     transforms.RandomResizedCrop(size=(28, 28), scale=(0.7, 1))]
>>> plt.figure(figsize=(10, 10))
>>> plt.subplot(1, 4, 1)
>>> display_image_greys(image)
>>> for i in range(3):
...     plt.subplot(1, 4, i+2)
...     img_crop = crop_transform(image)
...     display_image_greys(img_crop)
```

W tym przypadku parametr `size` określa wielkość wynikowego obrazka po przycięciu i przeskalowaniu, a parametr `scale` definiuje zakres skalowania stosowanego do przycinania. W razie użycia wartości (`min_scale`, `max_scale`) wielkość obszaru przycinania będzie wybierana losowo i będzie mieścić się w zakresie od `min_scale` do `max_scale` wymiarów początkowego obrazka.

Wyniki generowane przez powyższy kod zostały przedstawione na rysunku 11.13.



Rysunek 11.13. Obrazki przycięte na potrzeby uzupełniania danych

Jak widać w tym przykładzie, parametr `scale=(0.7, 1.0)` oznacza, że wielkość obszaru przycinania będzie się wahać w zakresie od 70% do 100% wielkości obrazka wyjściowego.

Usprawnianie klasyfikatora obrazów poprzez uzupełnianie danych

Wykorzystaj poznane techniki uzupełniania obrazów do wytrenowania klasyfikatora na małym zbiorze. W tym celu wykonaj poniższe kroki:

1. Zaczynij od przygotowania funkcji przekształcającej, która wykorzysta wszystkie techniki uzupełniania danych opisane we wcześniejszych punktach rozdziału:

```
>>> torch.manual_seed(42)
>>> transform_train = transforms.Compose([
...     transforms.RandomHorizontalFlip(),
...     transforms.RandomRotation(10),
...     transforms.RandomResizedCrop(size=(28, 28), scale=(0.9, 1)),
...     transforms.ToTensor(),
... ])
```

Zastosowaliśmy tu odwracanie w poziomie, obrót o kąt 10° oraz przycinanie w zakresie od 90% do 100% wielkości wyjściowego obrazka.

2. Ponownie wczytaj zbiór treningowy, używając powyższej funkcji przekształcającej, przy czym do uczenia użyj jedynie 500 próbek:

```
>>> train_dataset_aug =
...     torchvision.datasets.FashionMNIST(root=image_path,
...                                     train=True,
...                                     transform=transform_train,
...                                     download=False)
>>> from torch.utils.data import Subset
>>> train_dataset_aug_small = Subset(train_dataset_aug,
...                                 torch.arange(500))
```

Teraz przekonajmy się jak uzupełnianie danych poprawi zdolność uogólniania oraz wydajność modelu, nawet jeśli dysponujemy jedynie bardzo małym zbiorem danych.

3. Załaduj mały, ale uzupełniony zbiór danych w partiach po 64 próbki, tak jak zrobiłeś to wcześniej:

```
>>> train_dl_aug_small = DataLoader(train_dataset_aug_small,
...                                batch_size,
...                                shuffle=True)
```

Zwróć uwagę, że w przypadku zastosowania tego sposobu ładowania danych dla jednego obrazka do trenowania modelu użytych zostanie kilka obrazków uzupełnionych, które mogą powstać poprzez odwrócenie, obrót lub przycięcie.

4. Zainicjuj model konwolucyjny, używając do tego celu takiej samej architektury i optymalizatora jak wcześniej:

```
>>> model = nn.Sequential()
>>> # tu pomijamy powtarzający się kod
>>> model = model.to(device)
>>> optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

5. Wytrenuj model na uzupełnionych danych w 1000 iteracji:

```
>>> train(model, optimizer, 1000, train_dl_aug_small)
Epoka 1 - strata: 2.3013 - dokładność: 0.1400
...
Epoka 301 - strata: 1.6817 - dokładność: 0.7760
...
Epoka 601 - strata: 1.5006 - dokładność: 0.9620
...
Epoka 1000 - strata: 1.4904 - dokładność: 0.9720
```

6. Sprawdź działanie modelu na zbiorze testowym:

```
>>> evaluate_model(model, test_dl)
Accuracy on test set: 79.24%
```

Model wytrenowany na uzupełnionych danych uzyskał na zbiorze testowym dokładność 79,24%. Zwróć uwagę, że ten wynik może się zmieniać.

Przeprowadziliśmy również eksperymenty z trenowaniem modelu bez uzupełniania danych, co zapewniło dokładność na zbiorze testowym na poziomie około 76%. Po zastosowaniu uzupełniania dokładność wzrosła do 79%. Jak zawsze możesz swobodnie dostrajać hiperparametry, tak jak robiliśmy to w rozdziale 6., „Prognozowanie cen akcji za pomocą sieci neuronowych”, i sprawdzić, czy uzyskasz jeszcze większą dokładność klasyfikacji.

Kolejnym sposobem poprawiania wydajności klasyfikatorów jest technika określana jako uczenie transferowe. Przyjrzymy się jej bliżej w następnym podrozdziale.

Rozwijanie klasyfikatora z wykorzystaniem uczenia transferowego

Uczenie transferowe (ang. *transfer learning*) to technika uczenia maszynowego, w której model wytrenowany na jednym zadaniu jest adaptowany lub dostrajany do drugiego, powiązanego zadania. W uczeniu transferowym wiedza zdobyta podczas trenowania pierwszego zadania (tak zwanego zadania źródłowego) jest wykorzystywana do poprawy procesu uczenia drugiego zadania (zadania docelowego). Rozwiązanie to jest szczególnie przydatne, gdy dysponujemy ograniczoną ilością danych dla zadania docelowego, ponieważ umożliwia przeniesienie wiedzy z większego lub bardziej zróżnicowanego zbioru danych.

Typowy przebieg pracy z uczeniem transferowym obejmuje:

1. **Model wstępnie wytrenowany:** Rozpoczynamy od modelu wstępnie wytrenowanego, który został już nauczony na dużym i odpowiednim zbiorze danych dla innego, lecz powiązanego zadania. Taki model często jest głęboką siecią neuronową, na przykład siecią konwolucyjną wykorzystywaną do zadań związanych z obrazami.
2. **Ekstrakcja cech:** Wykorzystujemy model wstępnie wytrenowany jako ekstraktor cech. W takim przypadku należy z niego usunąć ostatnie warstwy klasyfikacyjne (jeśli takie są) i wykorzystać wyjście jednej z warstw pośrednich jako reprezentację cech danych. Takie cechy potrafią uchwycić wzorce wysokiego poziomu oraz informacje z zadania źródłowego.
3. **Dostrajanie:** Do ekstraktora cech dodajemy nowe warstwy. Te nowe warstwy są specyficzne dla zadania docelowego i zazwyczaj są inicjalizowane losowo. Następnie trenujemy cały model, łącznie z ekstraktorem cech i nowymi warstwami, na docelowym zbiorze danych. Dostrajanie umożliwia modelowi dostosowanie się do specyfiki zadania docelowego.

Zanim przejdziemy do implementacji uczenia transferowego w naszym zadaniu klasyfikacji obrazów odzieży, przyjrzyjmy się ewolucji architektur sieci konwolucyjnych oraz modeli wstępnie wytrenowanych. Nawet najwcześniejsze architektury sieci konwolucyjnych wciąż są aktywnie używane! Kluczową kwestią jest to, że wszystkie te architektury cały czas stanowią wartościowe narzędzia we współczesnym zestawie narzędzi do pracy z uczeniem głębokim, zwłaszcza gdy są stosowane w zadaniach uczenia transferowego.

Rozwój architektur sieci konwolucyjnych oraz wstępnie wytrenowanych modeli

Koncepcja wykorzystania sieci konwolucyjnych do przetwarzania obrazów sięga lat 90. XX wieku. Wczesne architektury, takie jak **LeNet-5** (z 1998 r.), pokazały potencjał głębokich sieci neuronowych w zadaniach klasyfikacji obrazów. LeNet-5 składa się z dwóch zestawów warstw konwolucyjnych, po których umieszczone są dwie warstwy w pełni połączone oraz jedna warstwa wyjściowa. Każda warstwa konwolucyjna wykorzystuje jądra o rozmiarze 5×5 . Architektura LeNet-5 odegrała istotną rolę w udowodnieniu skuteczności uczenia głębokiego w zadaniach związanych z klasyfikacją obrazów. Model ten osiągnął wysoką dokładność na zbiorze danych MNIST, który jest powszechnie używanym zestawem testowym do rozpoznawania ręcznie pisanych cyfr.

Osiągnięcia architektury LeNet-5 utorowały drogę do powstania bardziej złożonych architektur, takich jak **AlexNet** (z 2012 r.). Architektura ta składa się z ośmiu warstw — pięciu zestawów warstw konwolucyjnych, po których następują trzy warstwy w pełni połączone. W architekturze AlexNet po raz pierwszy w głębokiej sieci konwolucyjnej zastosowano funkcję aktywacji ReLU oraz technikę dropout w warstwach w pełni połączonych w celu zapobiegania przeuczeniu. Do poprawy zdolności uogólniania modelu wykorzystano techniki uzupełniania danych, takie jak losowe przycinanie oraz odwracanie w poziomie. Sukces architektury AlexNet przyczynił się do ponownego

wzrostu zainteresowania sieciami neuronowymi i zapoczątkował rozwój jeszcze głębszych i bardziej złożonych architektur, takich jak VGGNet, GoogLeNet i ResNet, które stały się podstawą współczesnej wizji komputerowej.

Architektura **VGGNet** została zaproponowana przez Visual Geometry Group z Uniwersytetu Oksfordzkiego w 2014 r. Charakteryzuje się prostotą i jednolitością. Składa się z szeregu warstw konwolucyjnych, po których następują warstwy maksymalizujące, a na końcu znajduje się stos warstw w pełni połączonych. W tej architekturze wykorzystywane są głównie filtry konwolucyjne o rozmiarze 3×3 , co pozwala sieci na wychwytywanie szczegółowych informacji przestrzennych. Najczęściej wykorzystywanymi wersjami tej architektury są VGG16 i VGG19, zawierające, odpowiednio, 16 i 19 warstw. Modele o tej architekturze są często używane jako punkt wyjścia do uczenia transferowego w różnych zadaniach z zakresu wizji komputerowej.

W tym samym roku firma Google opracowała architekturę **GoogLeNet**, znaną również jako **Inception**. Jej cechą charakterystyczną jest moduł *incepcji* (ang. *inception*). Zamiast stosowania pojedynczej warstwy konwolucyjnej o stałym rozmiarze filtra moduły incepcji wykorzystują równoległe filtry o różnych rozmiarach (1×1 , 3×3 , 5×5). Te równoległe operacje pozwalają wychwytywać cechy w różnych skalach i zapewniają bogatszą reprezentację danych. Podobnie jak VGGNet, wstępnie wytrenowane modele GoogLeNet występują w różnych wariantach, takich jak InceptionV1, InceptionV2, InceptionV3 i InceptionV4, z których każdy charakteryzuje się pewnymi modyfikacjami i ulepszeniami architektury.

Architektura **ResNet**, czyli **sieć rezydualna** (ang. *residual network*), została wprowadzona przez Kaiminga He i współpracowników w 2015 r. w celu rozwiązania problemu zanikających gradientów – zjawiska, w którym gradienty funkcji straty w konwolucyjnych sieciach neuronowych stają się bardzo małe. Kluczową innowacją tej architektury jest zastosowanie połączeń rezydualnych. Bloki rezydualne pozwalają sieci pomijać wybrane warstwy podczas procesu uczenia. Zamiast bezpośrednio uczyć się odwzorowania wejścia na wyjście, bloki rezydualne uczą się tzw. odwzorowania rezydualnego, które następnie dodawane jest do początkowego wejścia. Dzięki temu możliwe stało się budowanie głębszych sieci. Wstępnie wytrenowane modele ResNet dostępne są w różnych wersjach, takich jak ResNet-18, ResNet-34, ResNet-50, ResNet-101 czy niezwykle głęboki ResNet-152. Liczby te oznaczają głębokość sieci, czyli liczbę warstw.

Rozwój architektur konwolucyjnych sieci neuronowych oraz wstępnie wytrenowanych modeli trwa, czego przykładem są innowacje takie jak EfficientNet, MobileNet czy architektury projektowane do specyficznych zastosowań. Na przykład modele **MobileNet** zostały zaprojektowane z myślą o wysokiej wydajności pod względem zużycia zasobów obliczeniowych i pamięci. Są one przeznaczone do wdrożeń na urządzeniach o ograniczonych możliwościach sprzętowych, takich jak smartfony, urządzenia IoT oraz urządzenia brzegowe.

Uwaga

Lista wszystkich wstępnie wytrenowanych modeli dostępnych w bibliotece PyTorch znajduje się na stronie <https://pytorch.org/vision/stable/models.html#classification>.

Ewolucja architektur sieci konwolucyjnych oraz dostępność wstępnie wytrenowanych modeli zrewolucjonizowały zadania związane z komputerowym rozpoznawaniem obrazów. Znacząco poprawiły one wyniki osiągane w klasyfikacji obrazów, detekcji obiektów, segmentacji oraz wielu innych zastosowaniach.

Przyjrzyjmy się teraz, jak wykorzystać model wstępnie wytrenowany, aby ulepszyć nasz klasyfikator obrazów odzieży.

Poprawa klasyfikatora obrazów odzieży poprzez dostrajanie sieci ResNet

W tym przykładzie do przedstawienia wykorzystania uczenia transferowego użyjemy modelu o architekturze ResNet, a konkretnie modelu ResNet-18:

1. Zaczniij od zaimportowania wstępnie wytrenowanego modelu ResNet-18 z pakietu torchvision:

```
>>> from torchvision.models import resnet18
>>> my_resnet = resnet18(weights='IMAGENET1K_V1')
```

Wartość IMAGENET1K odwołuje się do wstępnie wytrenowanego modelu, który był trenowany na zbiorze danych ImageNet-1K (<https://www.image-net.org/download.php>), natomiast V1 określa pierwszą wersję tego wstępnie wytrenowanego modelu.

W ten sposób dysponujesz już wstępnie wytrenowanym modelem.

2. Zbiór danych ImageNet-1K zawiera obrazy RGB, dlatego też pierwsza warstwa konwolucyjna pierwotnego modelu ResNet została zaprojektowana pod kątem trójwymiarowych danych wejściowych. Jednak używany we wcześniejszych przykładach zbiór danych FashionMNIST zawiera obrazy w skali szarości, zatem musisz zmodyfikować model tak, by akceptował dane jednowymiarowe:

```
>>> my_resnet.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2,
...                               padding=3, bias=False)
```

W tym przykładzie w wywołaniu tworzącym pierwszą warstwę konwolucyjną zmieniliśmy pierwszy argument, wymiar danych wejściowych, z 3 na 1. Jej pierwotna definicja ma następującą postać:

```
self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,
bias=False)
```

3. Zmień warstwę wynikową, tak by zwracała 100 z 1000 klas:

```
>>> num_fts = my_resnet.fc.in_features
>>> my_resnet.fc = nn.Linear(num_fts, 10)
```

Zmieniliśmy jedynie wielkość wyników warstwy wyjściowej.

Kroki 2. i 3. stanowią przygotowanie do procesu **dostrajania** modelu.

4. Dostrój zaadaptowany wstępnie wytrenowany model na pełnym treningowym zbiorze danych:

```
>>> my_resnet = my_resnet.to(device)
>>> optimizer = torch.optim.Adam(my_resnet.parameters(), lr=0.001)
>>> train(my_resnet, optimizer, 10, train_d1)
```

Epoka 1 - strata: 0.4845 - dokładność: 0.8257
 Epoka 2 - strata: 0.3305 - dokładność: 0.8795
 Epoka 3 - strata: 0.2865 - dokładność: 0.8968
 Epoka 4 - strata: 0.2679 - dokładność: 0.9037
 Epoka 5 - strata: 0.2361 - dokładność: 0.9140
 Epoka 6 - strata: 0.2183 - dokładność: 0.9198
 Epoka 7 - strata: 0.1916 - dokładność: 0.9301
 Epoka 8 - strata: 0.1884 - dokładność: 0.9306
 Epoka 9 - strata: 0.1736 - dokładność: 0.9363
 Epoka 10 - strata: 0.1426 - dokładność: 0.9476

Jak widać, w przypadku użycia dostrojonego modelu ResNet po 10 epokach uzyskaliśmy dokładność wynoszącą 94%.

5. Sprawdź wyniki uzyskiwane na zbiorze testowym:

```
>>> evaluate_model(my_resnet, test_d1)
```

Dokładność na zbiorze testowym: 91.27 %

Dokładność na zbiorze testowym udało się nam poprawić z 90% na 91% i to przy zastosowaniu jedynie 10 iteracji treningu.

Uczenie transferowe z wykorzystaniem konwolucyjnych sieci neuronowych jest potężną techniką pozwalającą na korzystanie ze wstępnie wytrenowanych modeli oraz adaptowanie ich do specyficznych zadań związanych z klasyfikowaniem obrazów.

Podsumowanie

W tym rozdziale została opisana klasyfikacja zdjęć odzieży, oparta na konwolucyjnej sieci neuronowej. Najpierw szczegółowo zapoznałeś się z elementami sieci i jej działaniem inspirowanym komórkami układu wzrokowego człowieka. Następnie zbudowałeś sieć konwolucyjną klasyfikującą zdjęcia odzieży marki Zalando, zawarte w zbiorze Fashion-MNIST. Dowiedziałeś się także, czym jest uzupełnianie danych, i poznałeś kilka popularnych metod uzupełniania obrazów. I w końcu, po opisaniu ewolucji konwolucyjnych sieci neuronowych, poznałeś przykład uczenia transferowego z wykorzystaniem modelu o architekturze ResNet.

W następnym rozdziale poznasz inny typ sieci uczenia głębokiego: rekurencyjną sieć neuronową. Sieci konwolucyjna i rekurencyjna to dwie najpotężniejsze odmiany sieci neuronowych, dzięki którym uczenie głębokie jest dzisiaj tak popularne.

Ćwiczenia

1. Jak wspomniałem wcześniej, poeksperymentuj z parametrami konwolucyjnej sieci neuronowej i sprawdź, czy uzyskasz jeszcze lepszy wynik.
2. Spróbuj zastosować techniki dropout w celu poprawy wyników modelu konwolucyjnej sieci neuronowej.
3. Spróbuj poeksperymentować z wykorzystaniem wstępnie wytrenowanego modelu Vision Transformer: <https://huggingface.co/google/vit-base-patch16-224>.

| Skorowidz

A

- agregacja bootstrap, 47, 108
- AI, 21
- akcje, 145
 - kluczowe cechy, 146
- algebra liniowa, 26
- algorytm
 - AlphaGo, 23
 - C4.5, 90
 - CART, 90, 96, 112
 - CBOW, 227
 - CHAID, 90
 - DBSCAN, 251
 - GBT, 110, 112
 - ID3, 90
 - iteracji polityki, 440, 443
 - iteracji wartości, 436, 443
 - k-średnich, 235
 - lasu regresyjnego, 166
 - Q-uczenia, 450
 - implementacja, 451
 - regresji liniowej, 155
 - regresji logistycznej, 124
 - regresyjnego drzewa decyzyjnego, 162
 - skip-gram, 228
 - word2vec, 227
- algorytmy
 - drzewiaste, 85
 - regresji, 143
 - uczenia maszynowego, 30
 - uczenia przez wzmacnianie, 431
- Anaconda, 51
- analiza
 - głównych składowych, PCA, 281
 - sentymentu
 - dostrajanie modelu BERT, 387
 - użycie modelu BERT, 384
 - tekstu
 - pakiet scikit-learn, 219, 222
 - pakiet seaborn, 217

API

- narzędzia ChatGPT, 249
- Trainer, 391
- aproksymacja, 178
- atrybut, 62
- automatyka, 24

B

- badanie
 - danych tekstowych
 - przetwarzanie cech, 219
 - rozkład klas, 218
 - tworzenie słownika, 216
 - z grup dyskusyjnych, 216
 - zbioru zdjęć odzieży, 324
 - zbioru zdjęć twarzy, 278
- BERT, bidirectional encoder representations from transformers, 384
- biblioteka
 - Gensim, 207, 213, 230
 - Gymnasium, 427, 435
 - Hugging Face, 391
 - Matplotlib, 57, 217, 327
 - NLTK, 57, 207, 211, 223
 - PyTorch, 55, 57
 - SentenceTransformers, 419
 - spaCy, 207, 212
 - TensorFlow, 54
 - TextBlob, 208
 - torchvision, 325
- biblioteki
 - do przetwarzania języka naturalnego, 207
 - do uczenia maszynowego, 52
- binaryzacja, 299
- Blackjack
 - utworzenie środowiska, 443
 - użycie algorytmu Q-uczenia, 450
- błąd
 - klasyfikacji, 265
 - średniokwadratowy MSE, 35, 119, 166, 178

C

cecha, feature, 40
cechy
 danych tekstowych, 219
 dokumentowanie, 300
 ilościowe, 87
 jakościowe, 87
 kategorialne, 87, 108, 295
 przekształcanie, 114
 liczbowe, 108, 114
 normalizacja, 298
 porządkowe, 87
 selekcja, 131–133, 296
 wyodrębnianie, 300
charakterystyka operacyjna odbiornika, ROC, 79
ChatGPT
 opisywanie klastrów, 248
ciągłe wartości docelowe, 143
CLIP, Contrastive Language-Image
 Pre-Training, 401
czystość klasy, class purity, 89

D

dane
 kategorialne, 87
 liczbowe, 87
 przechowywanie, 294
 przygotowywanie, 289–295
 tekstowe
 pozyskiwanie, 214
 przetwarzanie cech, 219
 wizualizacja, 223
 treningowe, 40
długoterminowe zależności, 351
dostrajanie
 hiperparametrów, 194
 modelu BERT, 386
 wstępnie wytrenowanego, 387
 z użyciem API Trainer, 391
 sieci ResNet, 342
dowód, evidence, 65
dropout, odrzucanie, 189
drzewa ze wzmocnieniem gradientowym,
 GBT, 100–112
drzewo decyzyjne, 87, 112, 306
 algorytmy, 90
 budowanie, 89
 gromadzenie, 108
 implementacja, 96, 102
 klasyfikacyjne, 160

 korzeń i liście, 87
 prognozowanie kliknięć reklam, 103
 regresyjne, 160
 wyszukiwanie algorytmu, 88
duże obciążenie, 34
dyskretyzacja, 299
działanie lasu losowego, 110
dzielenie zbioru próbek, 90

E

eksploracja danych, 42
entropia, 94
etapy
 prognozowania, 59
 treningu, 59
etykiety, klasy, 58
kodowanie, 43

F

filtry konwolucyjne, 334
FrozenLake, 432
 utworzenie środowiska, 432
funkcja
 aktywacji, activation fuction, 175
 dobór, 187
 GELU, 413
 Leaky ReLU, 188
 liniowa, 188
 ReLU, 175, 177, 188
 sigmoida, 118, 175, 176, 187
 softmax, 187
 tangens hiperboliczny, 175, 176, 188
 wagi, 175
 jądra, kernel function, 273
 kosztu, 120, 122, 130, 154
 logistyczna, 118
 radialna bazowa, RBF, 274, 275
 straty kontrastowej, 403
funkcje niewypukłe, 120
funkcje wypukłe, 120

G

generowanie
 autoregresywne, 394
 obrazów, 406
 osadzeń, 416
 tekstu, 366, 368, 394, 406
 wektorów osadzenia, 376

Gensim, 207, 213, 230
 gęste osadzanie, dense embedding, 45
 giełda, 145
 analiza fundamentalna, 145
 analiza techniczna, 145
 indeks NASDAQ Composite, 146
 wyliczanie nowych cech, 147
 głosowanie, 47
 GPT, Generative Pre-training Transformer, 394
 gradient, 55
 prosty, 122, 134
 prosty stochastyczny, SGD, 128, 134
 graf
 obliczeniowy, 55
 skierowany acykliczny, DAG, 56
 grupowanie danych, 46
 Gym, 427
 Gymnasium, 427, 435
 instalowanie, 428

H

hiperparametry, 109
 wybór wartości początkowych, 283
 hiperpłaszczyzna, 260
 decyzyjna, 285
 optymalna, 262, 263
 rozdzielająca, 261
 Hugging Face, 391

I

implementacja
 algorytmu Q-uczenia, 451
 drzewa decyzyjnego, 96, 102
 klastrowania metodą k-średnich, 236, 241
 lasu regresyjnego, 166
 maszyny wektorów nośnych, 266
 modelu CLIP, 414
 naiwnego klasyfikatora Bayesa, 68
 regresji liniowej, 155, 158
 regresji logistycznej, 138
 regresyjnego drzewa decyzyjnego, 162
 sieci neuronowej, 181, 184, 186
 imputowanie, 43
 interakcja, 299
 inżynieria cech, 46, 147, 298, 299

J

jądro
 gaussowskie, 274
 liniowe, 275
 sigmoidalne, 275
 wielomianowe, 275

K

klasa
 CountVectorizer, 220, 222, 245
 DataLoader, 326
 KMeans, 242
 LatentDirichletAllocation, 256
 MLPRegressor, 184
 SGDRegressor, 158
 SimpleImputer, 291
 StratifiedKfold, 82
 TSNE, 225
 klastrowanie, clustering, 234
 metodą k-średnich, 235
 danych z grup dyskusyjnych, 245
 dobór wartości k, 243
 implementacja, 236
 implementacja z użyciem scikit-learn, 241
 wyniki, 239–242
 klasy, etykiety, 58
 klasyfikacja, 29, 58
 binarna, 59, 61, 261
 metodą zero-shot, 421, 424
 obrazów, 404
 tekstu, 405
 ocena jakości, 77
 trójklasowa, 268
 wieloetykietowa, 61
 wieloklasowa, 60, 61, 136
 wielomianowa, 60
 klasyfikacyjne drzewo decyzyjne, 160
 klasyfikator
 obrazów, 280, 404
 analiza głównych składowych, 281
 odzieży, 328, 338, 342
 twarzy, 278
 probabilistyczny, 119
 uzupełnianie danych, 338
 wykorzystanie uczenia transferowego, 339, 342
 klasyfikowanie
 danych
 użycie regresji logistycznej, 117

- zdjęć odzieży
 - użycie konwolucyjnej sieci neuronowej, 328
- zdjęć twarzy
 - użycie maszyny wektorów nośnych, 278
- kodowanie
 - „1 z n”, 44, 114
 - cech kategorialnych, 108
 - etykiet, 43
 - porządkowe, 114, 116
 - pozycyjne, 382
- kojarzenie, association, 234
- komórka sieci LSTM, 353
- kompromis między obciążeniem a wariancją, 35
- korpusy, 208
- korzeń, root, 88
- krzywa
 - ROC, 80
 - uczenia, 308

L

- las
 - losowy, 108, 110, 112, 306
 - selekcja cech, 133
 - regresyjny
 - implementacja, 166
- lematyzacja, 212, 222
- LSTM, Long Short-Term Memory, 351

Ł

- łączenie modeli, 47

M

- macierz, 40
 - nieujemna faktoryzacja, 252
 - pomyłek, confusion matrix, 77
- maszyna wektorów nośnych, SVM, 260, 305
 - błąd klasyfikacji, 265
 - hiperpłaszczyzna, 261
 - hiperpłaszczyzna optymalna, 262
 - implementacja, 266
 - klasyfikator obrazów, 278, 280
 - określanie granic klas, 260
 - podejście „jeden na jednego”, 269
 - podejście „jeden przeciw reszcie”, 268
 - z jądrem, 272
- Matplotlib, 57, 217, 327

- mechanizm
 - projekcji, 412
 - samouwagi, self-attention, 373
 - wektor klucza, K, 375
 - wektor wartości, V, 375
 - wektor zapytania, Q, 375
 - uwagi, 373
 - uwagi jednogłowej, 379
- metoda
 - Monte Carlo, 443
 - ocenianie polityki, 445
 - sterowanie z polityką, 447
 - wstrzymywanie, holdout method, 38
- miara podobieństwa
 - cosinusowego, 244
 - Jaccarda, 244
- model
 - BERT, 384
 - dostrajanie, 386, 387
 - dostrajanie z użyciem API Trainer, 391
 - wstępne trenowanie, 384
 - CLIP, 400
 - architektura, 401
 - generowanie osadzeń, 416
 - identyfikacja dopasowań, 416
 - implementacja, 414
 - klasyfikacja metodą zero-shot, 421, 424
 - koder tekstu, 402, 411
 - koder wizyjny, 402, 410
 - mechanizm działania, 401
 - moduł projekcji, 412
 - trenowanie, 415
 - uczenie kontrastowe, 403
 - wyszukiwanie obrazów za pomocą słów, 415, 419
 - zastosowania, 404
 - GPT
 - generowanie tekstu, 394
 - pisanie powieści, 395
 - wstępne trenowanie, 394
 - regresji logistycznej, 119, 122, 123, 127–132
 - tematyczny, 251
- modele
 - agregacja bootstrap, 47
 - aktualizowanie, 316
 - generatywne, generative models, 30
 - głosowanie i uśrednianie, 47
 - łączenie, 47
 - monitorowanie, 311–316
 - nadmierne dopasowanie, 32
 - niedostateczne dopasowanie, 34

modele

- składowanie, 49
- strojenie, 81
- wzmacnianie, 49

modelowanie

- addytywne, 111
- dużych zbiorów danych, 310
- języka maskowanego, MLM, 385
- semantyczne, 213
- tematyczne, topic modeling, 213, 251
 - użycie nieujemnej faktoryzacji macierzy, 252
 - użycie ukrytej alokacji Dirichleta, 255

moduł

- BernoulliNB, 71
- DecisionTreeClassifier, 102
- DictVectorizer, 115
- LinearSVC, 264
- OneHotEncoder, 106, 115
- pickle, 311
- RandomForestClassifier, 133
- SVC, 264
- TensorBoard, 195
- TensorFlow, 138
- torchtext, 354

N

nadmierne dopasowanie, 32

- diagnozowanie, 308
- zapobieganie, 307
 - redukcja wymiarowości, 40
 - regularyzacja, 38
 - selekcja cech, 40
 - weryfikacja krzyżowa, 36

naiwny klasyfikator Bayesa, 62, 305

- etapy treningu i prognozowania, 66
- implementacja, 68
- mechanizm działania, 64
- prognozowanie kliknięć, 87
- system rekomendacyjny, 72
- trenowanie, 76

niedostateczne dopasowanie, 34

nieujemna faktoryzacja macierzy, NMF, 252

NLP, natural language processing, 180, 203

NLTK, 57, 207, 223

- części mowy, 211
- korpusy, 208

normalizacja warstw, 384

NumPy, 53

O

obciążenie, bias, 34, 35, 119, 154

obserwacje, 143

obsługa brakujących wartości, 43

ocena f1, f1 score, 78

odległość

- Czebyszewa, 236
- euklidesowa, 236, 244
- Manhattan, 236, 244

OpenAI Gym, 57, 427

osadzanie słów, word embedding, 214, 227, 301

- tworzenie modeli osadzeń, 227
- wstępnie wytrenowane modele, 229

osadzenia Word2Vec, 302

P

pakiet

- gensim, 230
- Gym, 57, 427
- Gymnasium, 427
- Matplotlib, 57, 217
- NLTK, 57
- NumPy, 53
- OpenAI Gym, 57, 427
- pandas, 54, 105
- scikit-learn, 54
- SciPy, 54
- Seaborn, 57, 217
- TensorFlow, 54, 138, 158, 184, 312
- transformers, 57

pandas, 54, 105

perceptron wielowarstwowy, 184

pierwiastek błędu średniokwadratowego, RMSE, 166

pole pod krzywą, AUC, 79

polityka, policy, 431

pozyskiwanie danych

- Flickr8k, 407
- z grup dyskusyjnych, 214
- z serwisu Project Gutenberg, 365
- z serwisu Yahoo Finance, 150

prawdopodobieństwo

- a posteriori, 65
- a priori, 65

prawo Moore'a, 30

precyzja, precision, 78

proces

- konwolucji, 320
- regresji, 143
- uczenia przez wzmacnianie, 430

prognozowanie, 59, 154
 cen akcji, 143
 użycie algorytmów regresji, 168
 użycie sieci LSTM, 362
 użycie sieci neuronowych, 173, 193
 kliknięć reklam, 85, 103, 114
 użycie regresji logistycznej, 127
 użycie regresyjnego drzewa decyzyjnego, 160
programowanie dynamiczne, 432
projekcja, 412
 cech, 41
propagacja
 wstecz, 178
 wstecz w czasie, BPTT, 351
próbki
 testowe, 31
 treningowe, 31
przechwycenie, intercept, 119, 154
przetwarzanie języka naturalnego, NLP, 180, 203
 biblioteka
 Gensim, 207, 213
 NLTK, 207, 223
 spaCy, 207, 212
 TextBlob, 208
 lematyzacja, 212, 222
 model BERT, 384
 model GPT, 394
 modelowanie semantyczne i tematyczne, 213
 nieujemna faktoryzacja macierzy, 252
 osadzenia słów, 214, 227
 oznaczanie części mowy, 211
 pakiet gensim, 230
 redukcja wymiarowości, 224
 rozpoznawanie jednostek nazwanych, 212
 rozproszone przetwarzanie danych, 214
 stemming, 212, 222
 tokenizacja, 210
 usuwanie stop-słów, 222
 wstępne przetwarzanie tekstu, 222
 wyszukiwanie według podobieństwa, 214
 zastosowania, 204
 zliczanie tokenów, 219
przewidywanie następnego zdania, NSP, 385
przyrost informacji, information gain, 91, 94
Python
 instalacja i konfiguracja, 51
 instalacja pakietów, 52
PyTorch, 55, 57
 dostrajanie hiperparametrów, 194
 implementacja sieci neuronowej, 186

 uzupełnianie obrazów, 335
 wstępnie wytrenowane modele, 341
 zapisywanie i ładowanie modelu, 314

R

redukcja wymiarowości, 40, 224, 229, 252, 297
 technika t-SNE, 224
regresja, 29, 143
 liniowa, 144, 154
 implementacja, 155
 implementacja z użyciem scikit-learn, 158
 implementacja z użyciem TensorFlow, 158
 logistyczna, 119, 305
 algorytm, 124, 129
 implementacja z użyciem TensorFlow, 138
 klasyfikowanie danych, 117
 prognozowanie kliknięć, 127
 trenowanie modelu, 127, 128
 wielomianowa, 136
 z gradientem prostym, 122, 127
 z regularyzacją, 130
 ze stochastycznym gradientem prostym, 128
 ocena jakości, 166
 softmax, 136
 wektorów nośnych, SVR, 284, 285
regresyjne drzewo decyzyjne, 160
 implementacja, 162
 prognozowanie, 160
regularyzacja, 38, 130
 L1, 130, 132
 selekcja cech, 132
 L2, 130
ReLU, Rectified Linear Unit, 175, 177, 188
rozkład t-Studenta, 224
rozrzut, recall, 78
rzutowanie, projection, 234

S

samouwaga, 373
 maskowana, masked self-attention, 382
scikit-learn, 54
 implementacja
 drzewa decyzyjnego, 102
 regresji liniowej, 158
 sieci neuronowej, 184
 klastrowanie metodą k-średnich, 241

- scikit-learn
 - moduł BernoulliNB, 71
 - moduł DecisionTreeClassifier, 102
 - obsługa klasyfikacji wieloklasowych, 270
 - usuwanie stop-słów, 222
 - zastosowanie techniki t-SNE, 224
 - zliczanie tokenów, 219
- SciPy, 54
- Seaborn, 57, 217
- sekwencja, 345
- selekcja cech, 40, 131
 - las losowy, 133
 - regularyzacja L1, 132
- SentenceTransformers, 419
- sieć
 - LSTM, 351
 - budowanie, 358
 - komórka, 352
 - prognozowanie cen akcji, 362
 - neuronowa, 306
 - dostrajanie parametrów, 194
 - funkcje aktywacji, 175, 187
 - głęboka, 180
 - implementacja, 181
 - implementacja z użyciem PyTorch, 186
 - implementacja z użyciem scikit-learn, 184
 - implementacja z użyciem TensorFlow, 184
 - jednokierunkowa, 175
 - jednowarstwowa, 174
 - krawędzie, 175
 - nadmierne dopasowanie, 188
 - ograniczanie dopasowania, 189, 191
 - prognozowanie cen akcji, 173, 193
 - trening, 193
 - tworzenie modeli osadzeń, 227
 - warstwy, 174
 - warstwy osadzeń, 303
 - węzły, 174
 - neuronowa jednokierunkowa, 346
 - neuronowa konwolucyjna, CNN, 319
 - architektura, 323, 340
 - budowanie, 323
 - filtry konwolucyjne, 334
 - klasyfikowanie zdjęć odzieży, 328
 - trening sieci, 331
 - tworzenie sieci, 328
 - uzupełnianie danych, 335
 - uzupełnianie obrazów poprzez obrót, 337
 - uzupełnianie obrazów poprzez odbicie, 335
 - uzupełnianie obrazów poprzez
 - przycięcie, 338
 - warstwa konwolucyjna, 319
 - warstwa nieliniowa, 321
 - warstwa redukująca, 321
 - neuronowa rekurencyjna, RNN, 344
 - analiza recenzji filmowych, 354
 - architektura, 345
 - budowanie, 358
 - generator tekstu, 368
 - GRU, 351
 - komórka, 352
 - LSTM, 351
 - niesynchroniczna typu „wiele do wielu”, 349
 - pisanie powieści, 365
 - synchroniczna typu „wiele do wielu”, 348
 - trening sieci, 350, 367
 - typu „jedno do wielu”, 348
 - typu „wiele do jednego”, 347
 - wprowadzenie dodatkowych warstw, 361
 - neuronowa Transformer
 - architektura, 380
 - koder-dekoder, 381
 - kodowanie pozycyjne, 382
 - mechanizm samouwagi, 373
 - normalizacja warstw, 384
 - samouwaga maskowana, 382
 - uwaga wielogłowa, 379
- sigmoida, 118, 175, 176, 187
- skalowanie, 45
 - cech liczbowych, 108
 - w pionie, scale-up, 294
 - w poziomie, scale-out, 294
- składowanie, stacking, 49
- softmax, 136, 187
- spaCy, 207, 212
- stemming, 212, 222
- sterowanie Monte Carlo, 447
- stochastyczny gradient prosty, SGD, 128, 134
- stopień dopasowania modelu regresyjnego, R^2 , 166
- strata logarytmiczna, 121
- strojenie modeli, 81
- struktura koder-dekoder, 381
- sumaryczna nagroda, cumulative reward, 430
- sygnał, 62
- system rekomendacyjny, 72
 - budowanie, 72
 - przygotowanie danych, 72
 - trenowanie, 76

szansa, likelihood, 65
sztuczna inteligencja, AI, 21
szybkość uczenia, learning rate, 122

Ś

średni błąd bezwzględny, MAE, 166

T

tangens hiperboliczny, 175, 176, 188
TensorBoard
wartości hiperparametrów, 199
widok panelu, 198
TensorFlow, 54
implementacja
regresji liniowej, 158
regresji logistycznej, 138
sieci neuronowej, 184
zapisywanie i ładowanie, 312
teoria prawdopodobieństwa, 26
TextBlob, 208
TF, term frequency, 300
TF-IDF, inverse document frequency, 300
tokeny, 210, 356
zliczanie wystąpień, 219
torchvision, 325
transformacja wielomianowa, 46, 299
transformers, 57
transformery, 384
trening modelu
BERT, 384
CLIP, 415
etapy, 59
na dużym zbiorze danych, 134
najlepsze praktyki, 304–310
regresji liniowej, 155
regresji logistycznej, 122, 128, 130
trenowanie sieci neuronowej, 193
t-SNE, 223, 224
redukcja wymiarowości, 224
wizualizacja danych tekstowych, 223
twierdzenie
Bayesa, 62
o uniwersalnej aproksymacji, 178

U

uczenie maszynowe, 21
dobre praktyki
przygotowywanie danych, 289–295
trenowanie, ocenianie i wybieranie
modelu, 304–310

tworzenie zbioru treningowego, 295–304
wdrażanie i monitorowania modelu,
311–316
głębokie, deep learning, 174, 179
inżynieria cech, 147
kontrastowe, 403, 412
nadzorowane, supervised learning, 28
częściowo, semi-supervised learning, 29
klasyfikacja, 29
regresja, 29
nienadzorowane, unsupervised learning,
27, 233
klastrowanie, 234
kojarzenie, 234
rzutowanie, 234
przez wzmacnianie, reinforcement
learning, 28, 426
algorytmy, 431
komponenty, 429
metoda Monte Carlo, 443
sumaryczna nagroda, 430
online, online learning, 134, 135
redukcja wymiarowości, 224
rozwiązywanie problemów, 288
rozwój algorytmów, 29
sekwencyjne, sequence learning, 111, 345
transferowe, transfer learning, 30, 339
dostrajanie, 340
ekstrakcja cech, 340
wstępne wytrenowanie, 340
zastosowania, 25
ukryta alokacja Dirichleta, LDA, 255
unigram, 210
uogólnianie danych, 31
uśrednianie, 47
utrata zawiasu, hinge loss, 265
uwaga
koder-dekoder, encoder-decoder
attention, 382
wielogłowa, 379
uzupełnianie danych, data augmentation
poprzez obrót obrazu, 336
poprzez odbicie obrazu, 335
poprzez przycięcie obrazu, 337

W

wagi, 175
uwagi, 376
wariancja, 33, 35
warstwa wielogłowa uwagi, 381
warstwy
normalizacja, 384

wczesne zatrzymanie, early stopping, 40
wektory
 nośne, support vectors, 260
 osadzeń, 228
 wag, 119
wektoryzacja dokumentów, 214, 220–223, 227–230
weryfikacja
 k-krotna, 37
 krzyżowa, 36, 37, 83
 k-krotna, 81
 wewnętrzna, 38
 zagnieżdżona, 38
 zewnętrzna, 38
 LOOCV, 37
widzenie komputerowe, computer vision, 180
wielkość kroku, step size, 122
wielomian wyższego rzędu, 39
wizualizacja
 danych tekstowych, 223
 filtrów konwolucyjnych, 334
 osadzeń słów
 grupowanie, klastrowanie, 229
 redukcja wymiarowości, 224, 229
worek
 słów, BoW, 219
 słów ciągły, CBOW, 227
wskaźnik
 TF, 300
 TF-IDF, 246, 300
wskaźniki
 jakości podziału zbioru, 91
 skuteczności klasyfikatora, 77

współczynnik, 119
 dyskontowy, 439
 klikalności, CTR, 86
 uczenia, learning rate, 111
wstępne przetwarzanie, 41, 42
wygładzanie Laplace’a, 67
wykres zanieczyszczenia Giniego, 92
wykrywanie ukrytych tematów
 klastrowanie, 245
 modelowanie tematyczne, 252, 255
wysoka wariancja, 33
wzmacnianie, boosting, 49, 110

Z

zanieczyszczenie Giniego, Gini impurity, 91, 92
zbiór, 31
 Fashion-MNIST, 324
 Flickr8k, 406
 pozyskiwanie, 407
 wczytywanie, 407
 Labeled Faces in the Wild, 278
 MNIST, 60
 testowy, 38, 127
 treningowy, 38, 127, 328, 366
 dobre praktyki, 295–304
zestawy, 31
 weryfikacyjne, 32
zmienne predykcyjne, 58
zwrot, return, 430

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Najlepsze praktyki uczenia maszynowego? Tylko z Pythonem!

Python wraz ze swoimi bibliotekami umożliwia tworzenie coraz bardziej wyrafinowanych implementacji algorytmów uczenia maszynowego. Dzięki temu systemy przetwarzania języka naturalnego i obrazów wkraczają do naszego życia na szeroką skalę. Aby jednak uzyskiwać najlepsze wyniki w tej dziedzinie, potrzebna jest znajomość dobrych praktyk.

Oto czwarte wydanie popularnego podręcznika, z którym nauczysz się stosować zaawansowane techniki uczenia maszynowego. Zawiera dwa nowe rozdziały poświęcone architekturze Transformer oraz modelom takim jak BERT i GPT, jak również multimodalnym modelom komputerowego rozpoznawania obrazów implementowanym z wykorzystaniem PyTorch i Hugging Face. Znajdziesz tu solidną dawkę teorii połączonej z przykładami jej praktycznego zastosowania. Dzięki lekturze poszerzysz wiedzę z zakresu uczenia głębokiego, odkryjesz pełny potencjał zaawansowanych technik uczenia maszynowego i łatwiej sprostasz codziennym wyzwaniom.

W książce między innymi:

- najlepsze praktyki uczenia maszynowego
- budowa i ulepszanie klasyfikatorów obrazów
- tworzenie i strojenie sieci neuronowych z wykorzystaniem TensorFlow i PyTorch
- rekurencyjne sieci neuronowe, transformery i model CLIP
- maszyna wektorów nośnych i poprawa ich wydajności
- regularyzacja, wybór cech i wiele innych przydatnych technik

Yuxi (Hayden) Liu jest autorem cenionych książek poświęconych uczeniu maszynowemu. Wcześniej pracował w Google i stosował techniki uczenia maszynowego w takich dziedzinach jak analiza danych i cyberbezpieczeństwo. Entuzjasta edukacji.

	KOD KORZYŚCI Sięgnij po więcej! ➤ 
 helion.pl  HELION S.A. ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	ISBN 978-83-289-3171-8  9 788328 931718
Cena: 129,00 zł	

<packt>