

# Deep Learning

Praca z językiem Python  
i biblioteką Keras

Wydanie III



François Chollet  
Matthew Watson

Helion 

Tytuł oryginału: Deep Learning with Python, 3<sup>rd</sup> Edition

Tłumaczenie: Anna Mizerska, z wykorzystaniem fragmentów poprzedniego wydania  
w przekładzie Konrada Matuka

ISBN: 978-83-289-3938-7

© Helion S.A. 2026.

Authorized translation of the English edition © 2026 Manning Publications.

This translation is published and sold by permission of Manning Publications,  
the owner of all rights to publish and sell the same.

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/delep3](https://helion.pl/user/opinie/delep3)

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](mailto:helion@helion.pl)

WWW: [helion.pl](https://helion.pl) (księgarnia internetowa, katalog książek)

Printed in Poland.

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

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

# Spis treści

|                                                                              |    |
|------------------------------------------------------------------------------|----|
| <i>Przedmowa</i> .....                                                       | 15 |
| <i>Podziękowania</i> .....                                                   | 17 |
| <i>O książce</i> .....                                                       | 19 |
| <i>O autorach</i> .....                                                      | 23 |
| <i>O okładce</i> .....                                                       | 24 |
| <b>1. Czym jest uczenie głębokie?</b> .....                                  | 25 |
| 1.1. Sztuczna inteligencja, uczenie maszynowe i uczenie głębokie .....       | 26 |
| 1.2. Sztuczna inteligencja .....                                             | 26 |
| 1.3. Uczenie maszynowe .....                                                 | 27 |
| 1.4. Formy danych umożliwiające uczenie .....                                | 29 |
| 1.5. „Głębia” uczenia głębokiego .....                                       | 32 |
| 1.6. Działanie uczenia głębokiego przedstawione na trzech rysunkach .....    | 34 |
| 1.7. Co wyróżnia uczenie głębokie? .....                                     | 36 |
| 1.8. Era generatywnej sztucznej inteligencji .....                           | 37 |
| 1.9. Co dotychczas osiągnięto za pomocą uczenia głębokiego? .....            | 38 |
| 1.10. Nie wierz w tymczasową popularność .....                               | 39 |
| 1.11. Lato może zamienić się w zimę .....                                    | 40 |
| 1.12. Nadzieje na powstanie sztucznej inteligencji .....                     | 42 |
| <b>2. Matematyczne podstawy sieci neuronowych</b> .....                      | 44 |
| 2.1. Pierwszy przykład sieci neuronowej .....                                | 45 |
| 2.2. Reprezentacja danych sieci neuronowych .....                            | 50 |
| 2.2.1. Skalary (tensory zerowymiarowe) .....                                 | 50 |
| 2.2.2. Wektory (tensory jednowymiarowe) .....                                | 50 |
| 2.2.3. Macierze (tensory dwuwymiarowe) .....                                 | 51 |
| 2.2.4. Trójuwymiarowe tensory i tensory<br>o większej liczbie wymiarów ..... | 51 |
| 2.2.5. Główne atrybuty .....                                                 | 52 |
| 2.2.6. Obsługa tensorów w bibliotece NumPy .....                             | 53 |
| 2.2.7. Wsad danych .....                                                     | 54 |
| 2.2.8. Prawdziwe przykłady tensorów danych .....                             | 54 |

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| 2.3. Koła zębate sieci neuronowych: operacje na tensorach .....            | 57  |
| 2.3.1. Operacje wykonywane element po elemencie .....                      | 58  |
| 2.3.2. Rzutowanie .....                                                    | 59  |
| 2.3.3. Iloczyn tensorowy .....                                             | 61  |
| 2.3.4. Zmiana kształtu tensora .....                                       | 63  |
| 2.3.5. Geometryczna interpretacja operacji tensorowych .....               | 64  |
| 2.3.6. Interpretacja geometryczna uczenia głębokiego .....                 | 68  |
| 2.4. Silnik sieci neuronowych: optymalizacja gradientowa .....             | 69  |
| 2.4.1. Czym jest pochodna? .....                                           | 70  |
| 2.4.2. Pochodna operacji tensorowej: gradient .....                        | 72  |
| 2.4.3. Stochastyczny spadek wzdłuż gradientu .....                         | 73  |
| 2.4.4. Łączenie pochodnych: algorytm propagacji wstecznej .....            | 77  |
| 2.5. Ponowna analiza pierwszego przykładu .....                            | 82  |
| 2.5.1. Ponowna implementacja pierwszego przykładu<br>od podstaw .....      | 84  |
| 2.5.2. Wykonanie jednego kroku trenowania .....                            | 86  |
| 2.5.3. Pełna pętla trenowania .....                                        | 88  |
| 2.5.4. Ocena modelu .....                                                  | 88  |
| 2.6. Podsumowanie .....                                                    | 89  |
| <br>3. Wprowadzenie do TensorFlow, PyTorch, JAX i Keras .....              | 90  |
| 3.1. Krótka historia frameworków uczenia głębokiego .....                  | 91  |
| 3.2. Wzajemne powiązania tych frameworków .....                            | 93  |
| 3.3. Wprowadzenie do TensorFlow .....                                      | 94  |
| 3.3.1. Pierwsze kroki w TensorFlow .....                                   | 95  |
| 3.3.2. Kompletny przykład: klasyfikacja liniowa w TensorFlow .....         | 100 |
| 3.3.3. Unikatowość podejścia TensorFlow .....                              | 104 |
| 3.4. Wprowadzenie do PyTorch .....                                         | 105 |
| 3.4.1. Pierwsze kroki w PyTorch .....                                      | 106 |
| 3.4.2. Kompletny przykład: klasyfikacja liniowa w PyTorch .....            | 108 |
| 3.4.3. Unikatowość podejścia PyTorch .....                                 | 111 |
| 3.5. Wprowadzenie do JAX .....                                             | 112 |
| 3.5.1. Pierwsze kroki w JAX .....                                          | 113 |
| 3.5.2. Tensory w JAX .....                                                 | 113 |
| 3.5.3. Generowanie liczb losowych w JAX .....                              | 114 |
| 3.5.4. Kompletny przykład: klasyfikacja liniowa w JAX .....                | 118 |
| 3.5.5. Unikatowość podejścia JAX .....                                     | 120 |
| 3.6. Wprowadzenie do Keras .....                                           | 120 |
| 3.6.1. Pierwsze kroki w Keras .....                                        | 121 |
| 3.6.2. Warstwy: podstawowe bloki konstrukcyjne<br>uczenia głębokiego ..... | 123 |
| 3.6.3. Od warstw po modele .....                                           | 126 |
| 3.6.4. Krok „kompilacji”: konfiguracja procesu uczenia .....               | 128 |
| 3.6.5. Wybór funkcji straty .....                                          | 130 |

|                                                                                                  |            |
|--------------------------------------------------------------------------------------------------|------------|
| 3.6.6. Metoda fit .....                                                                          | 130        |
| 3.6.7. Monitorowanie straty i wskaźników<br>dla danych walidacyjnych .....                       | 131        |
| 3.6.8. Wnioskowanie: używanie modelu po treningu .....                                           | 132        |
| Podsumowanie .....                                                                               | 133        |
| <b>4. Klasyfikacja i regresja .....</b>                                                          | <b>134</b> |
| 4.1. Przykład klasyfikacji binarnej: klasyfikacja recenzji filmów .....                          | 136        |
| 4.1.1. Zbiór danych IMDb .....                                                                   | 136        |
| 4.1.2. Przygotowywanie danych .....                                                              | 137        |
| 4.1.3. Budowa modelu .....                                                                       | 138        |
| 4.1.4. Walidacja modelu .....                                                                    | 142        |
| 4.1.5. Używanie wytrenowanej sieci do generowania<br>przewidywań dotyczących nowych danych ..... | 145        |
| 4.1.6. Dalsze eksperymenty .....                                                                 | 146        |
| 4.1.7. Wnioski .....                                                                             | 146        |
| 4.2. Przykład klasyfikacji wieloklasowej: klasyfikacja<br>krótkich artykułów prasowych .....     | 147        |
| 4.2.1. Zbiór danych Agencji Reutera .....                                                        | 147        |
| 4.2.2. Przygotowywanie danych .....                                                              | 148        |
| 4.2.3. Budowanie modelu .....                                                                    | 149        |
| 4.2.4. Walidacja modelu .....                                                                    | 150        |
| 4.2.5. Generowanie przewidywań dotyczących<br>nowych danych .....                                | 154        |
| 4.2.6. Inne sposoby obsługi etykiet i funkcji straty .....                                       | 154        |
| 4.2.7. Dlaczego warto tworzyć odpowiednio<br>duże warstwy pośrednie? .....                       | 155        |
| 4.2.8. Dalsze eksperymenty .....                                                                 | 156        |
| 4.2.9. Wnioski .....                                                                             | 156        |
| 4.3. Przykład regresji: przewidywanie cen mieszkań .....                                         | 156        |
| 4.3.1. Zbiór cen domów w Kalifornii .....                                                        | 157        |
| 4.3.2. Przygotowywanie danych .....                                                              | 158        |
| 4.3.3. Budowanie modelu .....                                                                    | 159        |
| 4.3.4. K-krotna walidacja krzyżowa .....                                                         | 160        |
| 4.3.5. Generowanie przewidywań dotyczących<br>nowych danych .....                                | 164        |
| 4.3.6. Wnioski .....                                                                             | 164        |
| <b>5. Podstawy uczenia maszynowego .....</b>                                                     | <b>166</b> |
| 5.1. Generalizacja: cel uczenia maszynowego .....                                                | 166        |
| 5.1.1. Zbyt słabe dopasowanie i nadmierne dopasowanie .....                                      | 167        |
| 5.1.2. Generalizacja w uczeniu głębokim .....                                                    | 173        |
| 5.2. Ocena modeli uczenia maszynowego .....                                                      | 179        |
| 5.2.1. Zbiory treningowe, walidacyjne i testowe .....                                            | 179        |
| 5.2.2. Proste punkty odniesienia oparte na zdrowym rozsądku ....                                 | 183        |
| 5.2.3. Rzeczy, o których warto pamiętać .....                                                    | 183        |

|                                                                                          |            |
|------------------------------------------------------------------------------------------|------------|
| 5.3. Poprawa dopasowania modelu .....                                                    | 184        |
| 5.3.1. Dostrajanie kluczowych parametrów spadku gradientu .....                          | 185        |
| 5.3.2. Lepsze założenia wstępne dla architektury .....                                   | 186        |
| 5.3.3. Zwiększanie pojemności modelu .....                                               | 187        |
| 5.4. Poprawa generalizacji .....                                                         | 190        |
| 5.4.1. Porządkowanie zbioru danych .....                                                 | 190        |
| 5.4.2. Przetwarzanie cech .....                                                          | 191        |
| 5.4.3. Wczesne zatrzymywanie trenowania .....                                            | 192        |
| 5.4.4. Regularyzacja modelu .....                                                        | 193        |
| Podsumowanie .....                                                                       | 201        |
| <b>6. Uniwersalny przepływ roboczy uczenia maszynowego ....</b>                          | <b>203</b> |
| 6.1. Definiowanie zadania .....                                                          | 204        |
| 6.1.1. Definiowanie problemu .....                                                       | 205        |
| 6.1.2. Zbieranie danych .....                                                            | 207        |
| 6.1.3. Zrozumienie danych .....                                                          | 211        |
| 6.1.4. Wybór miary sukcesu .....                                                         | 212        |
| 6.2. Budowa modelu .....                                                                 | 212        |
| 6.2.1. Przygotowywanie danych .....                                                      | 213        |
| 6.2.2. Określanie techniki oceny wydajności modelu .....                                 | 215        |
| 6.2.3. Tworzenie modeli lepszych od linii bazowej .....                                  | 215        |
| 6.2.4. Skalowanie w górę: tworzenie modelu,<br>który ulega nadmiernemu dopasowaniu ..... | 217        |
| 6.2.5. Regularyzacja modelu i jego dostrajanie .....                                     | 217        |
| 6.3. Wdrażanie modelu .....                                                              | 218        |
| 6.3.1. Prezentacja modelu interesariuszom<br>i ustalanie oczekiwań .....                 | 219        |
| 6.3.2. Dostarczanie modelu wnioskowania .....                                            | 219        |
| 6.3.3. Monitorowanie modelu działającego<br>w środowisku produkcyjnym .....              | 224        |
| 6.3.4. Utrzymywanie modelu .....                                                         | 224        |
| Podsumowanie .....                                                                       | 225        |
| <b>7. Szczegółowe omówienie biblioteki Keras .....</b>                                   | <b>227</b> |
| 7.1. Spektrum przepływów pracy .....                                                     | 228        |
| 7.2. Różne sposoby budowy modeli Keras .....                                             | 229        |
| 7.2.1. Model sekwencyjny .....                                                           | 229        |
| 7.2.2. Funkcjonalny interfejs API .....                                                  | 232        |
| 7.2.3. Tworzenie modelu przez dziedziczenie po klasie Model ...                          | 239        |
| 7.2.4. Mieszanie i dopasowywanie różnych komponentów .....                               | 241        |
| 7.2.5. Pamiętaj: używaj odpowiednich narzędzi<br>do swojego zadania .....                | 242        |
| 7.3. Wbudowane trenowanie i pętle ewaluacyjne .....                                      | 242        |
| 7.3.1. Pisanie własnych wskaźników .....                                                 | 244        |
| 7.3.2. Używanie wywołań zwrotnych .....                                                  | 245        |

|                                                                                         |            |
|-----------------------------------------------------------------------------------------|------------|
| 7.3.3. Pisanie własnych wywołań zwrotnych .....                                         | 247        |
| 7.3.4. Monitorowanie i wizualizacja z użyciem TensorBoard ...                           | 248        |
| 7.4. Pisanie własnych pętli trenowania i ewaluacji .....                                | 251        |
| 7.4.1. Trenowanie a wnioskowanie .....                                                  | 251        |
| 7.4.2. Pisanie niestandardowych funkcji kroku trenowania ....                           | 252        |
| 7.4.3. Niskopoziomowe użycie wskaźników .....                                           | 257        |
| 7.4.4. Użycie metody <code>fit()</code> z niestandardową pętlą trenowania ...           | 258        |
| 7.4.5. Obsługa wskaźników w niestandardowej<br>metodzie <code>train_step()</code> ..... | 262        |
| Podsumowanie .....                                                                      | 265        |
| <b>8. Klasyfikacja obrazów .....</b>                                                    | <b>266</b> |
| 8.1. Wprowadzenie do konwolucyjnych sieci neuronowych .....                             | 267        |
| 8.1.1. Działanie sieci konwolucyjnej .....                                              | 270        |
| 8.1.2. Operacja <code>max-pooling</code> .....                                          | 276        |
| 8.2. Trenowanie konwolucyjnej sieci neuronowej<br>na małym zbiorze danych .....         | 278        |
| 8.2.1. Stosowanie uczenia głębokiego w problemach<br>małych zbiorów danych .....        | 278        |
| 8.2.2. Pobieranie danych .....                                                          | 279        |
| 8.2.3. Budowa modelu .....                                                              | 282        |
| 8.2.4. Wstępna obróbka danych .....                                                     | 284        |
| 8.2.5. Stosowanie techniki augmentacji danych .....                                     | 288        |
| 8.3. Korzystanie ze wstępnie wytrenowanego modelu .....                                 | 293        |
| 8.3.1. Ekstrakcja cech ze wstępnie wytrenowanego modelu ....                            | 293        |
| 8.3.2. Dostrajanie wstępnie wytrenowanego modelu .....                                  | 301        |
| Podsumowanie .....                                                                      | 304        |
| <b>9. Wzorce architektury sieci konwolucyjnych .....</b>                                | <b>305</b> |
| 9.1. Modułowość, hierarchia i ponowne użycie .....                                      | 306        |
| 9.2. Połączenie rezydualne .....                                                        | 309        |
| 9.3. Normalizacja wsadu .....                                                           | 312        |
| 9.4. Separowalne konwolucje po kanałach .....                                           | 314        |
| 9.5. Zebranie informacji: mały model podobny do Xception .....                          | 317        |
| 9.6. Wykraczanie poza konwolucję: transformer wizyjny.....                              | 318        |
| Podsumowanie .....                                                                      | 319        |
| <b>10. Czego uczą się neuronowe sieci konwolucyjne? .....</b>                           | <b>321</b> |
| 10.1. Wizualizacja pośrednich aktywacji .....                                           | 322        |
| 10.2. Wizualizacja filtrów konwolucyjnych sieci neuronowych .....                       | 329        |
| 10.2.1. Wznoszenie gradientu w TensorFlow .....                                         | 331        |
| 10.2.2. Wznoszenie gradientu w PyTorch .....                                            | 332        |

|                                                                           |            |
|---------------------------------------------------------------------------|------------|
| 10.2.3. Wznoszenie gradientu w JAX .....                                  | 332        |
| 10.2.4. Pętla wizualizacji filtra .....                                   | 333        |
| 10.3. Wizualizacja map ciepła aktywacji klas .....                        | 336        |
| 10.3.1. Pobieranie gradientu najwyższej klasy:<br>wersja TensorFlow ..... | 339        |
| 10.3.2. Pobieranie gradientu najwyższej klasy:<br>wersja PyTorch .....    | 339        |
| 10.3.3. Pobieranie gradientu najwyższej klasy: wersja JAX .....           | 340        |
| 10.3.4. Wyświetlanie mapy ciepła aktywacji klasy .....                    | 340        |
| 10.4. Wizualizacja przestrzeni ukrytej sieci konwolucyjnej .....          | 342        |
| Podsumowanie .....                                                        | 343        |
| <b>11. Segmentacja obrazu .....</b>                                       | <b>345</b> |
| 11.1. Zadania widzenia komputerowego .....                                | 345        |
| 11.1.1. Rodzaje segmentacji obrazu .....                                  | 347        |
| 11.2. Trenowanie modelu segmentacji od podstaw .....                      | 348        |
| 11.2.1. Pobieranie zbioru danych na potrzeby segmentacji .....            | 348        |
| 11.2.2. Budowa i trenowanie modelu segmentującego obraz .....             | 350        |
| 11.3. Użycie wstępnie wytrenowanego modelu segmentacji .....              | 354        |
| 11.3.1. Pobieranie modelu Segment Anything Model .....                    | 355        |
| 11.3.2. Jak działa model SAM? .....                                       | 355        |
| 11.3.3. Przygotowanie obrazu testowego .....                              | 357        |
| 11.3.4. Promptowanie modelu z podaniem punktu docelowego .....            | 359        |
| 11.3.5. Promptowanie modelu z podaniem ramki .....                        | 362        |
| Podsumowanie .....                                                        | 363        |
| <b>12. Wykrywanie obiektów .....</b>                                      | <b>364</b> |
| 12.1. Jednoetapowe i dwuetapowe detektory obiektów .....                  | 365        |
| 12.1.1. Dwuetapowe detektory R-CNN .....                                  | 366        |
| 12.1.2. Detektory jednoetapowe .....                                      | 367        |
| 12.2. Trenowanie modelu YOLO od podstaw .....                             | 367        |
| 12.2.1. Pobieranie zbioru danych COCO .....                               | 367        |
| 12.2.2. Budowanie modelu YOLO .....                                       | 371        |
| 12.2.3. Przygotowanie danych ze zbioru COCO<br>dla modelu YOLO .....      | 374        |
| 12.2.4. Trenowanie modelu YOLO .....                                      | 377        |
| 12.3. Używanie wstępnie wytrenowanego detektora RetinaNet .....           | 381        |
| Podsumowanie .....                                                        | 384        |
| <b>13. Prognozowanie szeregów czasowych .....</b>                         | <b>385</b> |
| 13.1. Różne rodzaje zadań z szeregami czasowymi .....                     | 385        |
| 13.2. Przykład prognozowania temperatury .....                            | 386        |
| 13.2.1. Przygotowanie danych .....                                        | 390        |

|         |                                                                                    |            |
|---------|------------------------------------------------------------------------------------|------------|
| 13.2.2. | Punkt odniesienia oparty na zdrowym rozsądku,<br>a nie na uczeniu maszynowym ..... | 393        |
| 13.2.3. | Próba wykorzystania podstawowego<br>modelu uczenia maszynowego .....               | 394        |
| 13.2.4. | Próba wykorzystania jednowymiarowego<br>modelu konwolucyjnego .....                | 396        |
| 13.3.   | Rekurencyjne sieci neuronowe .....                                                 | 398        |
| 13.3.1. | Działanie rekurencyjnych sieci neuronowych .....                                   | 399        |
| 13.3.2. | Warstwa rekurencyjna w Keras .....                                                 | 402        |
| 13.3.3. | Wykorzystanie pełnego potencjału rekurencyjnych<br>sieci neuronowych .....         | 407        |
| 13.3.4. | Rekurencyjna technika dropout jako odpowiedź<br>na nadmierne dopasowanie .....     | 407        |
| 13.3.5. | Układanie warstw rekurencyjnych w stos .....                                       | 410        |
| 13.3.6. | Użycie dwukierunkowych rekurencyjnych<br>sieci neuronowych .....                   | 412        |
| 13.4.   | Idąc jeszcze dalej .....                                                           | 414        |
|         | Podsumowanie .....                                                                 | 415        |
| 14.     | <b>Klasyfikacja tekstu .....</b>                                                   | <b>417</b> |
| 14.1.   | Krótką historią przetwarzania języka naturalnego .....                             | 417        |
| 14.2.   | Przygotowanie danych tekstowych .....                                              | 420        |
| 14.2.1. | Tokenizacja na znaki i na słowa .....                                              | 423        |
| 14.2.2. | Tokenizacja na podsłowa .....                                                      | 426        |
| 14.3.   | Zbiory a sekwencje .....                                                           | 431        |
| 14.3.1. | Wczytywanie zbioru danych IMDb .....                                               | 432        |
| 14.4.   | Modele oparte na zbiorach .....                                                    | 434        |
| 14.4.1. | Trenowanie modelu typu worek słów .....                                            | 434        |
| 14.4.2. | Trenowanie modelu bigramowego .....                                                | 438        |
| 14.5.   | Modele sekwencyjne .....                                                           | 440        |
| 14.5.1. | Trenowanie modelu rekurencyjnego .....                                             | 442        |
| 14.5.2. | Osadzenia słów .....                                                               | 444        |
| 14.5.3. | Używanie osadzeń słów .....                                                        | 446        |
| 14.5.4. | Wstępne trenowanie osadzeń tekstu .....                                            | 450        |
| 14.5.5. | Klasyfikacja z użyciem wstępnie wytrenowanych<br>osadzeń słów .....                | 454        |
|         | Podsumowanie .....                                                                 | 456        |
| 15.     | <b>Modele językowe i transformer .....</b>                                         | <b>457</b> |
| 15.1.   | Model językowy .....                                                               | 457        |
| 15.1.1. | Trenowanie szekspirowskiego modelu językowego .....                                | 458        |
| 15.1.2. | Generowanie tekstu w stylu Szekspira .....                                         | 462        |

|                                                                                               |            |
|-----------------------------------------------------------------------------------------------|------------|
| 15.2. Uczenie sekwencja do sekwencji .....                                                    | 464        |
| 15.2.1. Tłumaczenie z języka angielskiego na hiszpański .....                                 | 466        |
| 15.2.2. Trenowanie sekwencja do sekwencji z użyciem<br>rekurencyjnych sieci neuronowych ..... | 468        |
| 15.3. Architektura transformera .....                                                         | 473        |
| 15.3.1. Mechanizm uwagi oparty na iloczynie skalarnym .....                                   | 475        |
| 15.3.2. Blok enkodera w transformerze .....                                                   | 479        |
| 15.3.3. Blok dekodera w transformerze .....                                                   | 482        |
| 15.3.4. Uczenie sekwencja do sekwencji<br>za pomocą transformera .....                        | 483        |
| 15.3.5. Osadzanie informacji o pozycji .....                                                  | 486        |
| 15.4. Klasyfikacja z użyciem wstępnie wytrenowanego transformera .....                        | 489        |
| 15.4.1. Wstępne trenowanie enkodera transformera .....                                        | 490        |
| 15.4.2. Wczytywanie wstępnie wytrenowanego transformera ...                                   | 491        |
| 15.4.3. Wstępne przetwarzanie recenzji filmowych IMDb .....                                   | 493        |
| 15.4.4. Dostrajanie wstępnie wytrenowanego transformera .....                                 | 496        |
| 15.5. Dlaczego transformer jest tak wydajny? .....                                            | 497        |
| Podsumowanie .....                                                                            | 500        |
| <b>16. Generowanie tekstu .....</b>                                                           | <b>502</b> |
| 16.1. Krótka historia generowania sekwencji .....                                             | 504        |
| 16.2. Trenowanie miniGPT .....                                                                | 506        |
| 16.2.1. Budowanie modelu .....                                                                | 509        |
| 16.2.2. Wstępne trenowanie modelu .....                                                       | 512        |
| 16.2.3. Dekodowanie generatywne .....                                                         | 514        |
| 16.2.4. Strategie próbkowania .....                                                           | 517        |
| 16.3. Używanie wstępnie wytrenowanych dużych modeli językowych ...                            | 520        |
| 16.3.1. Generowanie tekstu z modelem Gemma .....                                              | 521        |
| 16.3.2. Dostrajanie na instrukcjach .....                                                     | 524        |
| 16.3.3. Adaptacja niskiego rzędu .....                                                        | 526        |
| 16.4. Dalsze kroki z modelami LLM .....                                                       | 531        |
| 16.4.1. Uczenie ze wzmocnieniem z informacją zwrotną<br>od człowieka (RLHF) .....             | 531        |
| 16.4.2. Multimodalne modele LLM .....                                                         | 534        |
| 16.4.3. Generowanie wspomagane wyszukiwaniem (RAG) ...                                        | 538        |
| 16.4.4. Modele „rozumujące” .....                                                             | 539        |
| 16.5. Dokąd zmierzają duże modele językowe? .....                                             | 541        |
| Podsumowanie .....                                                                            | 543        |
| <b>17. Generowanie obrazu .....</b>                                                           | <b>545</b> |
| 17.1. Uczenie głębokie do generowania obrazu .....                                            | 545        |
| 17.1.1. Próbkowanie z ukrytych przestrzeni obrazów .....                                      | 546        |
| 17.1.2. Autoenkodery wariacyjne .....                                                         | 547        |
| 17.1.3. Implementacja autoenkodera wariacyjnego z Keras ...                                   | 550        |

|                                                                                                                 |            |
|-----------------------------------------------------------------------------------------------------------------|------------|
| 17.2. Modele dyfuzyjne .....                                                                                    | 556        |
| 17.2.1. Zbiór danych Oxford Flowers .....                                                                       | 557        |
| 17.2.2. Autoenkoder odszumiający typu U-Net .....                                                               | 558        |
| 17.2.3. Pojęcia kroku dyfuzji i harmonogramu dyfuzji .....                                                      | 560        |
| 17.2.4. Proces trenowania .....                                                                                 | 562        |
| 17.2.5. Proces generowania .....                                                                                | 564        |
| 17.2.6. Wizualizacja rezultatów z niestandardowym<br>wywołaniem zwrotnym .....                                  | 564        |
| 17.2.7. Czas zacząć! .....                                                                                      | 565        |
| 17.3. Modele tekst-do-obrazu .....                                                                              | 567        |
| 17.3.1. Badanie przestrzeni ukrytej modelu tekst-do-obrazu .....                                                | 569        |
| Podsumowanie .....                                                                                              | 573        |
| <b>18. Sprawdzone praktyki .....</b>                                                                            | <b>574</b> |
| 18.1. Korzystanie z pełni możliwości modeli .....                                                               | 575        |
| 18.1.1. Optymalizacja hiperparametrów .....                                                                     | 575        |
| 18.1.2. Składanie modeli .....                                                                                  | 581        |
| 18.2. Trenowanie modeli na większą skalę z użyciem wielu urządzeń ....                                          | 583        |
| 18.2.1. Trenowanie na wielu procesorach GPU .....                                                               | 584        |
| 18.2.2. Trenowanie rozproszone w praktyce .....                                                                 | 587        |
| 18.2.3. Trenowanie na TPU .....                                                                                 | 591        |
| 18.3. Przyspieszanie trenowania i wnioskowania<br>za pomocą obliczeń z mniejszą precyzją .....                  | 593        |
| 18.3.1. Precyzja zmiennoprzecinkowa .....                                                                       | 593        |
| 18.3.2. Wnioskowanie w połowicznej precyzji .....                                                               | 594        |
| 18.3.3. Trenowanie z mieszaną precyzją .....                                                                    | 595        |
| 18.3.4. Skalowanie straty z mieszaną precyzją .....                                                             | 596        |
| 18.3.5. Trenowanie z float8 .....                                                                               | 596        |
| 18.3.6. Szybsze wnioskowanie z kwantyzacją .....                                                                | 597        |
| Podsumowanie .....                                                                                              | 599        |
| <b>19. Przyszłość sztucznej inteligencji .....</b>                                                              | <b>600</b> |
| 19.1. Ograniczenia uczenia głębokiego .....                                                                     | 600        |
| 19.1.1. Modele uczenia głębokiego słabo adaptują się<br>do nowości .....                                        | 601        |
| 19.1.2. Modele uczenia głębokiego są bardzo wrażliwe<br>na sposób formułowania i inne czynniki rozpraszające .. | 603        |
| 19.1.3. Modele uczenia głębokiego mają problem z nauką<br>programów podlegających generalizacji .....           | 605        |
| 19.1.4. Ryzyko antropomorfizacji modeli<br>uczenia maszynowego .....                                            | 606        |
| 19.2. Skala nie jest złotym środkiem .....                                                                      | 607        |
| 19.2.1. Automaty a inteligentne agenty .....                                                                    | 608        |
| 19.2.2. Generalizacja lokalna a generalizacja ekstremalna .....                                                 | 610        |

|                                                                                                                     |     |
|---------------------------------------------------------------------------------------------------------------------|-----|
| 19.2.3. Zastosowanie inteligencji .....                                                                             | 612 |
| 19.2.4. Spektrum generalizacji .....                                                                                | 613 |
| 19.3. Jak budować inteligencję? .....                                                                               | 614 |
| 19.3.1. Hipoteza kalejdoskopowa .....                                                                               | 614 |
| 19.3.2. Istota inteligencji: dostrzeganie abstrakcji<br>i jej rekombinacja .....                                    | 615 |
| 19.3.3. Ustalenie odpowiedniego celu jest ważne .....                                                               | 616 |
| 19.3.4. Nowy cel: przystosowywanie się w locie .....                                                                | 618 |
| 19.3.5. Fundacja ARC Prize .....                                                                                    | 620 |
| 19.3.6. Era przystosowywania się w czasie testowania .....                                                          | 620 |
| 19.3.7. ARC-AGI-2 .....                                                                                             | 621 |
| 19.4. Brakujące elementy: wyszukiwanie i symbole .....                                                              | 623 |
| 19.4.1. Dwa bieguny abstrakcji .....                                                                                | 624 |
| 19.4.2. Poznanie jako połączenie dwóch rodzajów abstrakcji ...                                                      | 626 |
| 19.4.3. Dlaczego uczenie głębokie nie jest pełną odpowiedzią<br>na dostrzeganie abstrakcji? .....                   | 627 |
| 19.4.4. Alternatywne podejście do sztucznej inteligencji:<br>synteza programów .....                                | 628 |
| 19.4.5. Łączenie uczenia głębokiego i syntezy programów .....                                                       | 629 |
| 19.4.6. Wielokrotne używanie modułowych procedur<br>składowych i nieustanne uczenie się .....                       | 632 |
| 19.4.7. Przewidywania dotyczące dalekiej przyszłości .....                                                          | 634 |
| 20. Wnioski .....                                                                                                   | 636 |
| 20.1. Przypomnienie najważniejszych koncepcji .....                                                                 | 636 |
| 20.1.1. Różne podejścia do sztucznej inteligencji .....                                                             | 636 |
| 20.1.2. Co sprawia, że uczenie głębokie to wyjątkowa<br>dziedzina uczenia maszynowego? .....                        | 638 |
| 20.1.3. Jak należy traktować uczenie głębokie? .....                                                                | 638 |
| 20.1.4. Najważniejsze technologie .....                                                                             | 640 |
| 20.1.5. Uniwersalny przepływ roboczy<br>uczenia maszynowego .....                                                   | 641 |
| 20.1.6. Najważniejsze architektury sieci .....                                                                      | 642 |
| 20.2. Ograniczenia uczenia głębokiego .....                                                                         | 647 |
| 20.3. Przyszłość uczenia głębokiego .....                                                                           | 649 |
| 20.4. Bycie na bieżąco z nowościami związanymi<br>z szybko rozwijającą się dziedziną .....                          | 650 |
| 20.4.1. Zdobywaj wiedzę praktyczną, pracując<br>z prawdziwymi problemami przedstawianymi<br>w serwisie Kaggle ..... | 650 |
| 20.4.2. Czytaj o nowych rozwiązaniach w serwisie arXiv .....                                                        | 650 |
| 20.4.3. Eksploruj ekosystem związany z pakietem Keras .....                                                         | 651 |
| 20.5. Ostatnie słowa .....                                                                                          | 651 |
| Skorowidz .....                                                                                                     | 652 |



# 12 Wykrywanie obiektów

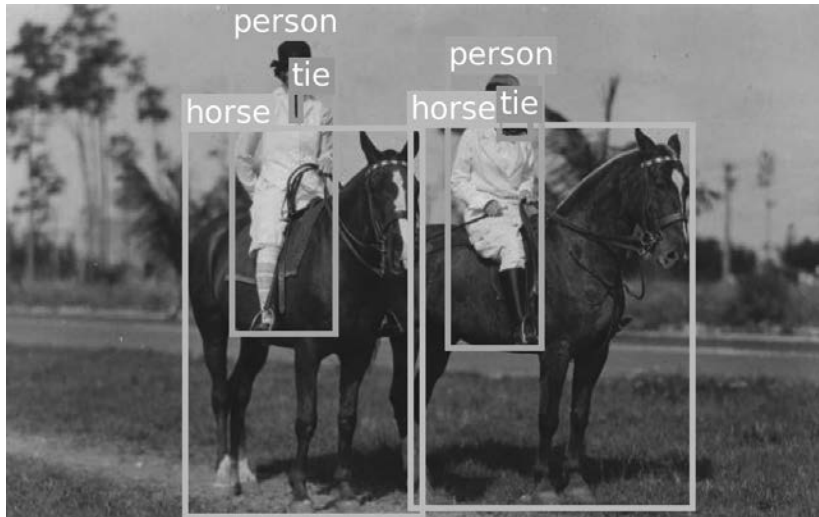
## W tym rozdziale opisaliśmy:

- Problem wykrywania obiektów.
- Dwu- i jednoetapowe wykrywanie obiektów.
- Trenowanie od podstaw prostego jednoetapowego detektora obiektu.
- Używanie wstępnie wytrenowanego modelu wykrywającego obiekty.

Wykrywanie obiektów polega na rysowaniu ramek (zwanymi **ramkami ograniczającymi**, ang. *bounding boxes*) wokół interesujących nas obiektów na zdjęciu (patrz rysunek 12.1). Dzięki temu nie tylko wiesz, jakie obiekty są na zdjęciu, ale również gdzie się znajdują. Główne zastosowania wykrywania obrazów to:

- **Liczenie.** Model zwraca informację o liczbie określonych obiektów na zdjęciu.
- **Śledzenie.** Model śledzi, jak obiekty poruszają się w czasie, przez wykrywanie obiektów w każdej klatce filmu.
- **Przycinanie.** Model znajduje obszar zawierający interesujący nas obiekt, przycina go i wysyła wersję tego fragmentu obrazu w wyższej rozdzielczości do modelu klasyfikacji lub do modelu OCR (optyczne rozpoznawanie znaków).

Możesz pomyśleć, że jeśli masz maskę segmentacji dla instancji obiektu, to możesz obliczyć współrzędne najmniejszej ramki zawierającej maskę. Więc dlaczego nie można cały czas stosować segmentacji obrazu? Czy w ogóle potrzebujemy modeli wykrywania obrazu?



**Rysunek 12.1.** Detektory obiektów rysują ramki wokół obiektów i oznaczają je

Rzeczywiście, segmentacja jest nadzbiorem wykrywania. Zwraca wszystkie informacje, które mógłby zwracać model wykrywania obrazu, i o wiele więcej. Jednak tyle informacji nie jest za darmo, wiąże się to z większymi zasobami obliczeniowymi. Zwykle dobry model wykrywający obiekty będzie działał dużo szybciej niż model segmentacji. Z modelem segmentacji wiąże się również koszt etykietowania. Aby wytrenować taki model, potrzebujesz masek z dokładnością co do piksela, a żeby je uzyskać, potrzeba dużo więcej czasu niż w przypadku ramek ograniczających wymaganych przez modele wykrywające obiekty.

Dlatego zawsze będziesz wybierać model wykrywający obiekty, gdy nie będziesz potrzebować informacji na poziomie piksela, np. jeśli chcesz jedynie policzyć obiekty na obrazie.

## 12.1. Jednoetapowe i dwuetapowe detektory obiektów

Są dwie szerokie kategorie architektur wykrywania obiektów:

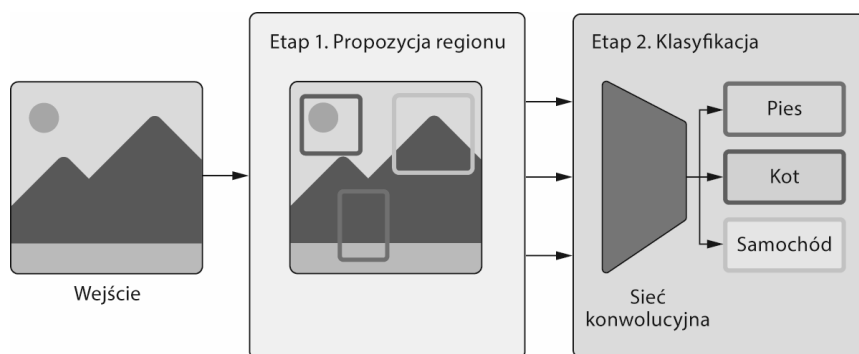
- Detektory dwuetapowe, znane jako modele neuronowych sieci konwolucyjnych opartych na regionach (R-CNN), które najpierw wyodrębniają propozycje regionów, a następnie klasyfikują je i doprecyzowują ich położenie.
- Detektory jednoetapowe, takie jak RetinaNet lub rodzina modeli You Only Look Once.

Przyjrzyjmy się, jak działają.

### 12.1.1. Dwuetapowe detektory R-CNN

Neuronowa sieć konwolucyjna oparta na regionach, inaczej: model R-CNN, to dwuetapowy model. Pierwszy etap polega na pobraniu obrazu i wytworzeniu kilku tysięcy częściowo nakładających się ramek ograniczających wokół obszarów przypominających obiekty. Ramki te są określane mianem **propozycji regionów**. Na tym etapie model nie jest zbyt inteligentny, więc nie możemy być pewni, czy zaproponowane regiony naprawdę zawierają obiekty, a jeśli zawierają, to nie wiemy, jakie to są obiekty.

Te problemy są rozwiązywane na drugim etapie. Neuronowa sieć konwolucyjna sprawdza każdy zaproponowany region i przyporządkowuje go do wcześniej zdefiniowanych klas, tak jak modele z rozdziału 9. (patrz rysunek 12.2). Zaproponowany region, który ma niski wynik dla wszystkich klas, jest odrzućany. W rezultacie zostaje dużo mniejsza liczba ramek z wysokim wynikiem dla jednej określonej klasy. Na koniec ramki ograniczające wokół każdego obiektu są udoskonalane, aby pozbyć się duplikatów i aby każda ramka ograniczająca była jak najdokładniejsza.



**Rysunek 12.2.** Model R-CNN najpierw wyodrębnia propozycje regionów, a następnie je klasyfikuje za pomocą sieci konwolucyjnej

W pierwszych wersjach modeli R-CNN pierwszym etapem był **model heurystyczny**, zwany selektywnym przeszukiwaniem, który używał definicji spójności przestrzennej do znajdowania obszarów przypominających obiekty. Z pojęciem *heurystyczny* będziesz się dość często spotykać przy okazji uczenia maszynowego, a oznacza ono po prostu „zestaw reguł zakodowanych na sztywno”. Zwykle jest używany w kontrze do wytrenowanych modeli (w których reguły są wyznaczane automatycznie) lub modeli ręcznie zaprojektowanych na podstawie wiedzy teoretycznej. W późniejszych wersjach modeli R-CNN, m.in. w modelu Faster-R-CNN, etap generowania ramek zaczął być realizowany za pomocą modelu uczenia głębokiego — sieci propozycji regionów.

Dwuetapowe podejście modeli R-CNN działa w praktyce bardzo dobrze, ale wymaga sporej ilości zasobów obliczeniowych, głównie z powodu konieczności

klasyfikacji tysiąca łat dla każdego przetwarzanego obrazu. Dlatego takie modele nie nadają się do działania w czasie rzeczywistym i dla systemów wbudowanych. Z doświadczenia wiemy, że w rzeczywistości zwykle nie ma potrzeby sięgania po wymagające pod względem zasobów obliczeniowych systemy wykrywające obiekty, takie jak modele R-CNN, bo jeśli wnioskowanie wykonywane jest po stronie serwera z potężnym procesorem graficznym, to prawdopodobnie lepiej użyć modelu segmentacji, takiego jak model SAM, który poznałeś w poprzednim rozdziale. A jeśli masz ograniczone zasoby, wtedy lepiej użyć wydajniejszej pod względem obliczeniowym architektury wykrywania obiektu — detektora jednoetapowego.

### 12.1.2. Detektory jednoetapowe

Około 2015 r. badacze i praktycy zaczęli eksperymentować z jednym modelem uczenia głębokiego, który przewidywał zarówno współrzędne ramek, jak i ich etykiety. Taka architektura nazywana jest **detektorem jednoetapowym**. Najpopularniejszymi detektorami jednoetapowymi są RetinaNet, Single Shot MultiBox Detectors (SSD) oraz rodzina modeli You Only Look Once (w skrócie YOLO, tak jak w memach — to celowy zabieg).

Detektory jednoetapowe, szczególnie ostatnie wersje YOLO, są dużo szybsze i wydajniejsze niż detektory dwuetapowe, aczkolwiek nieco mniej dokładne. Obecnie model YOLO jest prawdopodobnie najpopularniejszym detektorem obiektów, szczególnie jeśli chodzi o działanie w czasie rzeczywistym. Zwykle co roku pojawia się nowa wersja tego modelu, a co ciekawe, każda kolejna wersja jest tworzona przez inną organizację.

W następnym podrozdziale zbudujemy uproszczony model YOLO od podstaw.

## 12.2. Trenowanie modelu YOLO od podstaw

Ogólnie rzecz biorąc, budowanie detektora obiektów może być sporym przedsięwzięciem, ale nie dlatego, że to skomplikowane zadanie. Po prostu trzeba napisać bardzo dużo kodu do obsługi ramek ograniczających i przewidzianych przez model danych wyjściowych. My, dla uproszczenia, odtworzymy pierwszy model YOLO z 2015 r. Gdy piszemy tę książkę, jest 12 wersji tego modelu, ale z tą pierwszą pracuje się najłatwiej.

### 12.2.1. Pobieranie zbioru danych COCO

Do wytrenowania naszego modelu potrzebujemy zbioru danych. *Common Objects in Context*<sup>1</sup> (w skrócie COCO) to najlepiej znany i najczęściej używany zbiór danych do trenowania modeli wykrywania obiektów. Składa się ze zdjęć pochodzących z różnych źródeł i z etykiet nadanych przez ludzi. Zawiera etykiety

---

<sup>1</sup> Zbiór COCO 2017 znajduje się na stronie <https://cocodataset.org/>. Większość obrazów w tym rozdziale pochodzi z tego zbioru.

obiektów, ramki ograniczające, adnotacje i pełne maski segmentacji. Nie będziemy używać masek segmentacji i skupimy się wyłącznie na ramkach ograniczających.

Pobierzmy zbiór danych COCO, a dokładnie jego wersję z 2017 r. (patrz listing 12.1). Choć na dzisiejsze standardy nie jest on duży, ten 18-gigabajtowy zbiór jest największym, z jakiego do tej pory korzystaliśmy w tej książce. Jeśli uruchamiasz kod w miarę czytania tej książki, teraz jest dobra okazja, by nieco odpocząć, gdyż pobieranie danych chwilę potrwa.

#### Listing 12.1. Pobieranie zbioru danych COCO 2017

```
import keras
import keras_hub

images_path = keras.utils.get_file(
    "coco",
    "http://images.cocodataset.org/zips/train2017.zip",
    extract=True,
)
annotations_path = keras.utils.get_file(
    "annotations",
    "http://images.cocodataset.org/annotations/annotations_trainval2017.zip",
    extract=True,
)
```

Musimy nieco przetworzyć dane wejściowe, zanim będziemy mogli ich użyć. Pierwsze pobranie daje nam wszystkie obrazy ze zbioru COCO bez etykiet. Drugie pobranie zawiera metadane wszystkich obrazów w pliku JSON. W zbiorze COCO każdy plik z obrazem ma identyfikator, a każda ramka ograniczająca jest sparowana z jednym z tych identyfikatorów. Musimy zestawić wszystkie obrazy i ramki razem.

Każda ramka ograniczająca ma współrzędne lewego górnego rogu ( $x$ ,  $y$ ) oraz wymiary  $width$  i  $height$ , a początkiem układu współrzędnych jest lewy górny róg obrazu. Podczas wczytywania danych możemy przeskalować współrzędne wszystkich ramek ograniczających, tak by były punktami w kwadracie o bokach w przedziale  $[0, 1]$  (patrz listing 12.2). Dzięki temu łatwiej będzie manipulować tymi ramkami bez konieczności sprawdzania rozmiaru każdego obrazu wejściowego.

#### Listing 12.2. Przygotowanie danych COCO

```
import json

with open(f"{annotations_path}/annotations/instances_train2017.json", "r") as f:
    annotations = json.load(f)

images = {image["id"]: image for image in annotations["images"]} ← Sortuje metadane obrazów po identyfikatorze.

def scale_box(box, width, height): ← Konwertuje ramkę ograniczającą na współrzędne w kwadracie o bokach w przedziale [0, 1].
    scale = 1.0 / max(width, height)
    x, y, w, h = [v * scale for v in box]
    x += (height - width) * scale / 2 if height > width else 0
```

```

y += (width - height) * scale / 2 if width > height else 0
return [x, y, w, h]

```

**Zbiera wszystkie adnotacje ramek ograniczających przy użyciu identyfikatora obrazu.**

```

metadata = {}
for annotation in annotations["annotations"]:
    id = annotation["image_id"]
    if id not in metadata:
        metadata[id] = {"boxes": [], "labels": []}
    image = images[id]
    box = scale_box(annotation["bbox"], image["width"], image["height"])
    metadata[id]["boxes"].append(box)
    metadata[id]["labels"].append(annotation["category_id"])
    metadata[id]["path"] = images_path + "/train2017/" + image["file_name"]
metadata = list(metadata.values())

```

Przyjrzyjmy się danym, które właśnie wczytaliśmy (listing 12.3).

#### Listing 12.3. Sprawdzanie danych ze zbioru COCO

```

>>> len(metadata)
117266
>>> min([len(x["boxes"]) for x in metadata])
1
>>> max([len(x["boxes"]) for x in metadata])
63
>>> max(max(x["labels"]) for x in metadata) + 1
91
>>> metadata[435]
{"boxes": [[0.12, 0.27, 0.57, 0.33],
           [0.0, 0.15, 0.79, 0.69],
           [0.0, 0.12, 1.0, 0.75]],
 "labels": [17, 15, 2],
 "path": "/root/.keras/datasets/coco/train2017/000000171809.jpg"}
>>> [keras_hub.utils.coco_id_to_name(x) for x in metadata[435]["labels"]]
["cat", "bench", "bicycle"]

```

Mamy 117 266 unikatowych obrazów, a każdy z nich może zawierać od 1 do 63 obiektów z powiązaną ramką ograniczającą. Jest tylko 91 możliwych etykiet, wybranych przez twórców zbioru danych COCO.

Aby zmapować etykiety w postaci liczb całkowitych na czytelne dla człowieka nazwy, użyjemy narzędzia `keras_hub.utils.coco_id_to_name(id)` z KerasHub. Podobnie robiliśmy w rozdziale 8., odkodowując przewidywania ImageNet w celu uzyskania etykiet tekstowych.

Zwizualizujemy przykładowy obraz, aby zobaczyć, jak to działa. Z użyciem biblioteki Matplotlib możemy zdefiniować funkcję rysującą obraz i funkcję rysującą ramkę ograniczającą z etykietą na obrazie (patrz listing 12.4). Obu tych funkcji będziemy używać w całym rozdziale. Możemy użyć przestrzeni kolorów HSV, by zastosować prostą sztuczkę polegającą na generowaniu nowych kolorów dla każdej nowej wyświetlanej etykiety. Określimy stałe nasycenie i jasność koloru, a będziemy zmieniać tylko jego odcień — w ten sposób możemy generować nowe kolory, które będą się wyróżniać na obrazie.

**Listing 12.4. Wyświetlanie obrazu ze zbioru danych COCO wraz z ramką ograniczającą i z etykietą**

```
import matplotlib.pyplot as plt
from matplotlib.colors import hsv_to_rgb
from matplotlib.patches import Rectangle

color_map = {0: "gray"}

def label_to_color(label):
    if label not in color_map:
        h, s, v = (len(color_map) * 0.618) % 1, 0.5, 0.9
        color_map[label] = hsv_to_rgb((h, s, v))
    return color_map[label]

def draw_box(ax, box, text, color):
    x, y, w, h = box
    ax.add_patch(Rectangle((x, y), w, h, lw=2, ec=color, fc="none"))
    textbox = dict(fc=color, pad=1, ec="none")
    ax.text(x, y, text, c="white", size=10, va="bottom", bbox=textbox)

def draw_image(ax, image):
    ax.set(xlim=(0, 1), ylim=(1, 0), xticks=[], yticks=[], aspect="equal")
    image = plt.imread(image)
    height, width = image.shape[:2]
    hpad = (1 - height / width) / 2 if width > height else 0
    wpad = (1 - width / height) / 2 if height > width else 0
    extent = [wpad, 1 - wpad, 1 - hpad, hpad]
    ax.imshow(image, extent=extent)
```

Używa złotego podziału do generowania nowych odcieni jasnego koloru z przestrzeni HSV.

Rysuje obraz w kwadracie o bokach w przedziale [0, 1], którego lewy górny bok ma współrzędne (0,0).

Uzupełnia obraz, by wpasował się w kwadrat o bokach w przedziale [0, 1].

Użyjmy naszych nowych funkcji, by wyświetlić sprawdzany przez nas wcześniej przykładowy obraz<sup>2</sup> (patrz rysunek 12.3):

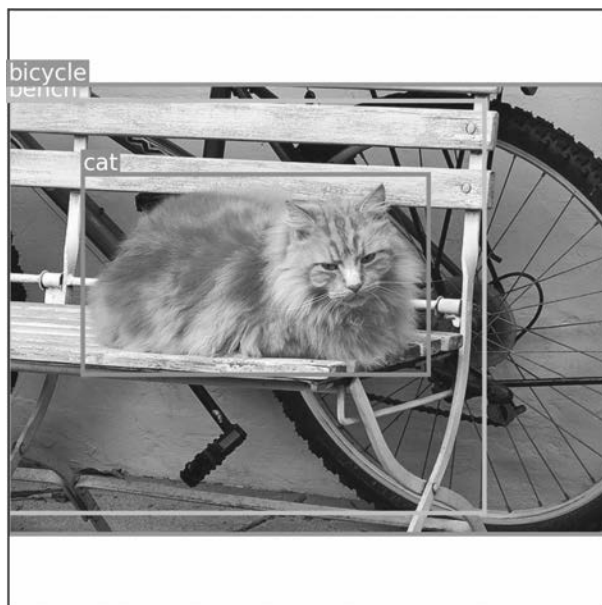
```
sample = metadata[435]
ig, ax = plt.subplots(dpi=300)
draw_image(ax, sample["path"])
for box, label in zip(sample["boxes"], sample["labels"]):
    label_name = keras_hub.utils.coco_id_to_name(label)
    draw_box(ax, box, label_name, label_to_color(label))
plt.show()
```

Choć trenowanie modelu na całym 18-gigabajtowym zbiorze danych byłoby ciekawe, chcemy, by przykłady w tych książkach można było łatwo uruchomić na skromnym sprzęcie. Jeśli ograniczymy się tylko do obrazów z co najwyżej czterema ramkami ograniczającymi, trenowanie będzie łatwiejsze i zmniejszymy zbiór danych o połowę. Zróbmy to więc i potasujmy nasze dane. Obrazy są zgrupowane według typu obiektu, a to negatywnie wpłynęłoby na trenowanie:

```
import random

metadata = list(filter(lambda x: len(x["boxes"]) <= 4, metadata))
random.shuffle(metadata)
```

<sup>2</sup> Obraz ze zbioru COCO 2017: <https://cocodataset.org/>. Obraz z serwisu Flickr: [https://farm8.staticflickr.com/7250/7520201840\\_3e01349e3f\\_z.jpg](https://farm8.staticflickr.com/7250/7520201840_3e01349e3f_z.jpg), CC BY 2.0, <https://creativecommons.org/licenses/by/2.0/>.



**Rysunek 12.3.** YOLO zwraca przewidzianą ramkę ograniczającą i etykietę przewidzianej klasy dla każdego regionu na obrazie

Po wczytaniu danych możemy rozpocząć budowę naszego modelu YOLO.

### 12.2.2. Budowanie modelu YOLO

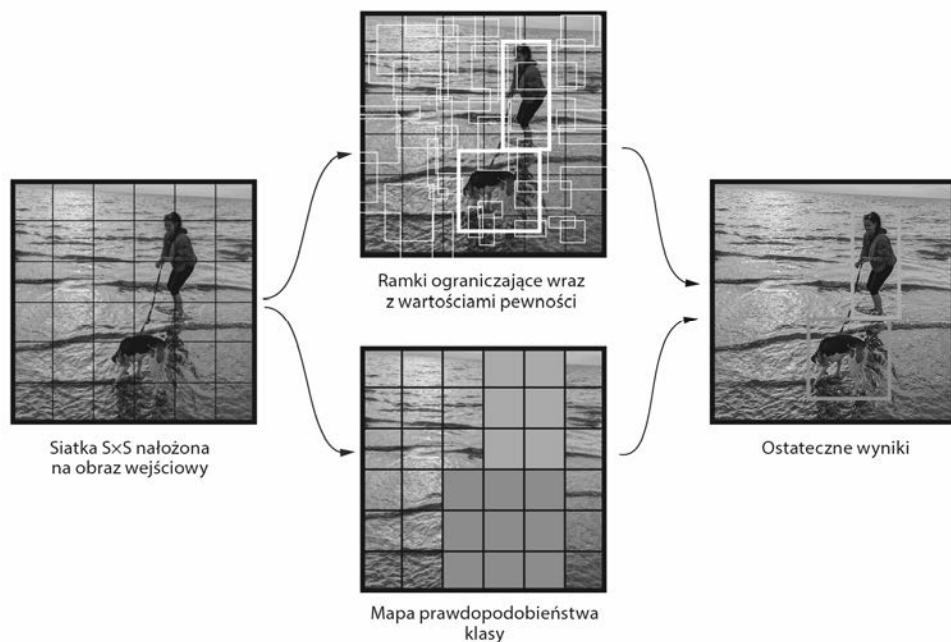
Jak wspomnieliśmy wcześniej, model YOLO jest jednoetapowym detektorem. Zamiast najpierw próbować znaleźć szukane obiekty, a następnie klasyfikować regiony tych obiektów, YOLO za jednym razem zaproponuje ramki ograniczające i etykiety obiektów.

Nasz model podzieli obraz na siatkę i przewidzi oddzielnie ramkę ograniczającą i etykietę klasy dla każdej lokalizacji na siatce. W artykule naukowym Redmona i innych<sup>3</sup> można przeczytać, że w pierwotnej wersji model tak naprawdę przewidywał kilka ramek dla jednej lokalizacji, ale dla uproszczenia będziemy przewidywać jedną ramkę dla każdej kratki na siatce.

Na większości obrazów obiekty nie są równomiernie rozmieszczone na siatce. Model będzie brał to pod uwagę, dlatego wraz z każdą ramką będzie również zwracał **wartość pewności**, jak widać na rysunku 12.4. Ta wartość ma być wysoka, gdy obiekt został wykryty w danej lokalizacji, a równa zero, gdy nie ma obiektu. Większość kratek w siatce nie będzie zawierała obiektów i model powinien zwracać wartość pewności bliską zero.

---

<sup>3</sup> Redmon i inni, *You Only Look Once: Unified, Real-Time Object Detection*, CoRR, 2015, <https://arxiv.org/abs/1506.02640>.



**Rysunek 12.4.** Zwizualizowane wyniki zwracane przez model YOLO, zgodnie z pierwszym artykułem naukowym na temat tego modelu

Jak wiele modeli widzenia komputerowego, YOLO używa *rdzenia* sieci konwolucyjnej, by dla obrazu wejściowego uzyskać interesujące wysokopoziomowe cechy (mówiliśmy o tym w rozdziale 8.). Autorzy wspomnianego wcześniej artykułu naukowego stworzyli własny model rdzenia i wytrenowali go na zbiorze ImageNet (przeznaczonym do trenowania klasyfikacji). My nie będziemy sami trenować takiego modelu i za pomocą KerasHub wczytamy wstępnie wytrenowany rdzeń.

Do tej pory w tej książce używaliśmy Xception jako rdzenia, ale teraz użyjemy ResNet — to rodzina modeli, o której wspomnieliśmy w rozdziale 9. Jej struktura jest podobna do Xception, ale ResNet używa kroków zamiast warstw łączących do zmniejszania rozdzielczości obrazu. Jak już wiesz z rozdziału 11., konwolucje z krokami są lepszym wyborem, gdy zależy nam na informacji o *położeniu w przestrzeni* danych wejściowych.

Wczytajmy nasz wstępnie wytrenowany model i przeprowadźmy wstępne przetwarzanie dopasowań (by przeskalować obraz; patrz listing 12.5). Zmienimy rozmiary naszych obrazów, tak by wszystkie miały wymiary 448×448. Rozmiar obrazu wejściowego jest dość ważny w zadaniach polegających na wykrywaniu obiektów.

## Listing 12.5. Wczytywanie modelu ResNet

```
image_size = 448

backbone = keras_hub.models.Backbone.from_preset(
    "resnet_50_imagenet",
)
preprocessor = keras_hub.layers.ImageConverter.from_preset(
    "resnet_50_imagenet",
    image_size=(image_size, image_size),
)
```

Teraz możemy zmienić nasz rdzeń na model wykrywania obiektów przez dodanie nowych warstw na potrzeby zwracania ramek i przewidzianych klas. Konfiguracja zaproponowana w dokumentacji modelu YOLO jest dość prosta. Należy wziąć dane wyjściowe z rdzenia sieci konwolucyjnej i przepuścić je przez dwie gęsto połączone warstwy z aktywacją pośrodku. Następnie należy podzielić uzyskany rezultat. Pierwsze pięć liczb jest związane z ramką ograniczającą (cztery to współrzędne ramki, a jedna — wartość pewności dla ramki). Pozostałe liczby są używane na potrzeby **mapy prawdopodobieństwa klasy** pokazanej na rysunku 12.4, czyli predykcji klasyfikacji dla każdej kartki w siatce dla wszystkich możliwych 91 etykiet.

Wypróbujmy to (patrz listing 12.6).

## Listing 12.6. Dołączanie wyników ostatnich warstw modelu YOLO (główicy predykcyjnej)

```
from keras import layers
```

```
grid_size = 6
num_labels = 91
```

```
inputs = keras.Input(shape=(image_size, image_size, 3))
x = backbone(inputs)
x = layers.Conv2D(512, (3, 3), strides=(2, 2))(x)
x = keras.layers.Flatten()(x)
x = layers.Dense(2048, activation="relu", kernel_initializer="glorot_normal")(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(grid_size * grid_size * (num_labels + 5))(x)
x = layers.Reshape((grid_size, grid_size, num_labels + 5))(x)
box_predictions = x[..., :5]
class_predictions = layers.Activation("softmax")(x[..., 5:])
outputs = {"box": box_predictions, "class": class_predictions}
model = keras.Model(inputs, outputs)
```

Przepuszcza nasze spłaszczone mapy cech przez dwie gęsto połączone warstwy.

Zmniejsza dane wyjściowe z rdzenia, a następnie spłaszcza cechy wyjściowe.

Zmienia kształt danych wyjściowych na siatkę o wymiarach 6×6.

Rozdziela ramkę i przewidywania klas.

Podsumowanie modelu pomoże nam go lepiej zrozumieć:

```
>>> model.summary()
Model: "functional_3"
```

| Layer (type)               | Output Shape        | Param # | Connected to |
|----------------------------|---------------------|---------|--------------|
| input_layer_7 (InputLayer) | (None, 448, 448, 3) | 0       | -            |

|                                          |                      |            |                       |
|------------------------------------------|----------------------|------------|-----------------------|
| res_net_backbone_12<br>(ResNetBackbone)  | (None, 14, 14, 2048) | 23,580,512 | input_layer_7[0][...] |
| conv2d_3 (Conv2D)                        | (None, 6, 6, 512)    | 9,437,696  | rest_net_bckbone_...  |
| flatten_3(Flatten)                       | (None, 18432)        | 0          | conv2d_3[0][0]        |
| dense_6 (Dense)                          | (None, 2048)         | 37,750,784 | flatten_3[0][0]       |
| dropout_3 (Dropout)                      | (None, 2048)         | 0          | dense_6[0][0]         |
| dense_7 (Dense)                          | (None, 3456)         | 7,081,344  | dropout_3[0][0]       |
| reshape_3 (Reshape)                      | (None, 6, 6, 96)     | 0          | dense_7[0][0]         |
| get_item_7(GetItem)                      | (None, 6, 6, 91)     | 0          | reshape_3[0][0]       |
| get_item_6(GetItem)                      | (None, 6, 6, 5)      | 0          | reshape_3[0][0]       |
| activation_33 (Activation)               | (None, 6, 6, 91)     | 0          | get_item_7[0][0]      |
| =====                                    |                      |            |                       |
| Total params: 77,850,336 (296.98 MB)     |                      |            |                       |
| Trainable params: 77,797,088 (296.77 MB) |                      |            |                       |
| Non-trainable params: 53,248 (208.00 KB) |                      |            |                       |

Nasze dane wyjściowe z rdzenia mają kształt (batch\_size, 14, 14, 2048). Jest to 401 408 liczb zmiennoprzecinkowych na obraz, czyli trochę za dużo dla naszych gęstych warstw. Zmniejszamy rozmiar map cech za pomocą warstwy konwolucyjnej z krokami, tak by miały kształt (batch\_size, 6, 6, 512) i 18 432 wartości zmiennoprzecinkowe na obraz.

Następnie dodajemy nasze dwie gęsto połączone warstwy. Spłaszczamy całą mapę cech, przepuszczamy ją przez warstwę Dense z aktywacją relu, a potem przepuszczamy ją przez ostatnią warstwę Dense z dokładną liczbą przewidywań wyjściowych — 5 dla ramki ograniczającej i wartości pewności oraz 91 wartości dla każdej klasy obiektu w każdej kratce siatki.

Na koniec zmieniamy kształt danych wyjściowych, tak by miały wymiary siatki 6×6, i rozdzielamy ramkę i przewidywania klas. Jak zwykle dla klasyfikacji stosujemy aktywację softmax. Dane wyjściowe dla ramki będą wymagać szczególnej uwagi, o czym powiemy później.

Ponieważ spłaszczamy całą mapę cech za pomocą warstwy klasyfikacji, każdy detektor siatki może używać cech całego obrazu — nie ma ograniczeń związanych z położeniem. To jest celowe. Bardzo duże obiekty będą zajmować więcej niż jedną komórkę siatki.

### 12.2.3. Przygotowanie danych ze zbioru COCO dla modelu YOLO

Nasz model jest stosunkowo prosty, ale nadal musimy wstępnie przetworzyć dane wejściowe, by dopasować je do siatki predykcji. Każdy detektor siatki będzie odpowiedzialny za wykrywanie ramek, których środek leży w środku kratki

siatki. Nasz model będzie zwracał pięć wartości zmiennoprzecinkowych dla ramki ( $x$ ,  $y$ ,  $w$ ,  $h$ , pewność). Wartości  $x$  i  $y$  to środek obiektu względem granic kratki siatki (mieszczą się w zakresie od 0 do 1). Wartości  $w$  i  $h$  to rozmiar obiektu względem rozmiaru obrazu.

W naszych danych treningowych mamy już wartości  $w$  i  $h$ . Musimy jednak przetłumaczyć wartości  $x$  i  $y$  na siatkę i z powrotem. Zdefiniujmy dwie funkcje pomocnicze:

```
def to_grid(box):
    x, y, w, h = box
    cx, cy = (x + w / 2) * grid_size, (y + h / 2) * grid_size
    ix, iy = int(cx), int(cy)
    return (ix, iy), (cx - ix, cy - iy, w, h)

def from_grid(loc, box):
    (xi, yi), (x, y, w, h) = loc, box
    x = (xi + x) / grid_size - w / 2
    y = (yi + y) / grid_size - h / 2
    return (x, y, w, h)
```

Przygotujmy dane treningowe, by pasowały do tej nowej struktury siatki. Możemy utworzyć dwie tablice o długości naszego zbioru danych z siatką (patrz listing 12.7):

- Pierwsza tablica będzie zawierać naszą mapę prawdopodobieństw klas. Odpowiednią etykietą oznaczmy wszystkie komórki siatki przecinające się z ramką ograniczającą. Aby nie komplikować kodu, nie będziemy się przejmować nakładającymi się ramkami.
- Druga tablica będzie zawierała ramki. Przełożymy wszystkie ramki na siatkę i oznaczmy odpowiednią komórkę siatki współrzędnymi ramki. Wartość pewności dla rzeczywistej ramki w naszych danych etykietowych zawsze będzie wynosić jeden, a pewność dla pozostałych lokalizacji będzie wynosić zero.

Listing 12.7. Tworzenie wartości docelowych modelu YOLO

```
import numpy as np
import math

class_array = np.zeros((len(metadata), grid_size, grid_size))
box_array = np.zeros((len(metadata), grid_size, grid_size, 5))

for index, sample in enumerate(metadata):
    boxes, labels = sample["boxes"], sample["labels"]
    for box, label in zip(boxes, labels):
        (x, y, w, h) = box
        left, right = math.floor(x * grid_size), math.ceil((x + w) * grid_size)
        bottom, top = math.floor(y * grid_size), math.ceil((y + h) * grid_size)
        class_array[index, bottom:top, left:right] = label

for index, sample in enumerate(metadata):
    boxes, labels = sample["boxes"], sample["labels"]
```

Znajduje wszystkie komórki siatki,  
których środek leży w środku ramki.

```

for box, label in zip(boxes, labels):
    (xi, yi), (grid_box) = to_grid(box) ← Przekłada ramkę na system współrzędnych siatki.
    box_array[index, yi, xi] = [*grid_box, 1.0]
    class_array[index, yi, xi] = label ← Upewnia się, że etykieta klasy dla położenia środka
                                         ramki pasuje do ramki.

```

Zwizualizujemy nasze dane treningowe dla modelu YOLO za pomocą funkcji pomocniczych rysujących ramki (patrz listing 12.8 i rysunek 12.5). Narysujemy całą mapę aktywacji klasy na naszym pierwszym obrazie wejściowym<sup>4</sup> i dodamy wartość pewności dla ramki wraz z jej etykietą.

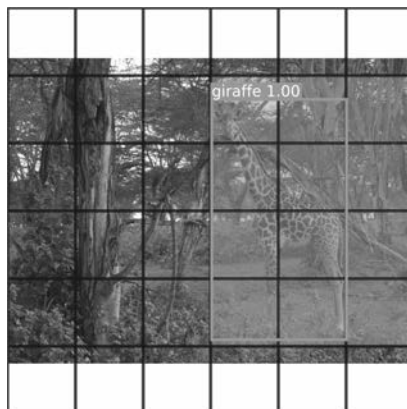
#### Listing 12.8. Wizualizacja danych docelowych dla modelu YOLO

```

def draw_prediction(image, boxes, classes, cutoff=None):
    fig, ax = plt.subplots(dpi=300)
    draw_image(ax, image)
    for yi, row in enumerate(classes): ← Rysuje siatkę wyjściową i mapę
        for xi, label in enumerate(row): ← prawdopodobieństwa klasy modelu YOLO.
            color = label_to_color(label) if label else "none"
            x, y, w, h = (v / grid_size for v in (xi, yi, 1.0, 1.0))
            r = Rectangle((x, y), w, h, lw=2, ec="black", fc=color, alpha=0.5)
            ax.add_patch(r)
    for yi, row in enumerate(boxes): ← Rysuje wszystkie ramki w każdej komórce
        for xi, box in enumerate(row): ← siatki powyżej naszej wartości odcięcia.
            box, confidence = box[:4], box[4]
            if not cutoff or confidence >= cutoff:
                box = from_grid((xi, yi), box)
                label = classes[yi, xi]
                color = label_to_color(label)
                name = keras_hub.utils.coco_id_to_name(label)
                draw_box(ax, box, f"{name} {max(confidence, 0):.2f}", color)
    plt.show()

draw_prediction(metadata[0]["path"], box_array[0], class_array[0], cutoff=1.0)

```



**Rysunek 12.5.** Model YOLO zwraca ramkę ograniczającą i etykiety klasy dla każdego regionu obrazu

<sup>4</sup> Obraz ze zbioru COCO 2017: <https://cocodataset.org/>. Obraz z serwisu Flickr: [https://farm9.staticflickr.com/8081/8387882360\\_5b97a233c4\\_z.jpg](https://farm9.staticflickr.com/8081/8387882360_5b97a233c4_z.jpg), CC BY 2.0, <https://creativecommons.org/licenses/by/2.0/>.

Na koniec wczytajmy dane naszego obrazu za pomocą `tf.data`. Wczytamy nasze obrazy z dysku, wstępnie je przetworzymy i podzielimy na partie wsadowe (patrz listing 12.9). Powinniśmy również wydzielić zbiór walidacyjny w celu monitorowania treningu.

**Listing 12.9.** Tworzenie zbioru danych do trenowania modelu

```
import tensorflow as tf

def load_image(path):
    x = tf.io.read_file(path)
    x = tf.image.decode_jpeg(x, channels=3)
    return preprocessor(x)

images = tf.data.Dataset.from_tensor_slices([x["path"] for x in metadata])
images = images.map(load_image, num_parallel_calls=8)
labels = {"box": box_array, "class": class_array}
labels = tf.data.Dataset.from_tensor_slices(labels)

dataset = tf.data.Dataset.zip(images, labels).batch(16).prefetch(2)
val_dataset, train_dataset = dataset.take(500), dataset.skip(500)
```

Wczytuje dane i zmienia ich rozmiar za pomocą `tf.data`.

Tworzy scalony zbiór danych i dzieli je na partie.

Wydziela dane walidacyjne.

Teraz nasze dane są gotowe do trenowania.

**UWAGA** Ten przykład trenowania jasno pokazuje, dlaczego biblioteka taka jak `tf.data` jest pomocna. Wczytywanie wszystkich obrazów z tego ogromnego zbioru danych naraz przeciążyłoby pamięć (pamiętaj, że tensor z danymi obrazu jest dużo większy niż skompresowany plik JPEG). Dzięki `tf.data` możemy wczytać dane naszych obrazów partia po partii i zwalniać pamięć, gdy jakaś część danych zostanie wczytana. Mapujemy konkretne części zbioru danych, których potrzebujemy w określonym momencie. Wywołanie `prefetch(2)` spowoduje, że `tf.data` będzie buforować tylko dwie partie, które będą gotowe do użycia, więc nie trzeba przerywać trenowania, aby wczytać więcej obrazów i zmienić ich rozmiar.

## 12.2.4. Trenowanie modelu YOLO

Mamy już model i dane treningowe, ale brakuje nam jeszcze jednego elementu, by uruchomić metodę `fit()`, mianowicie funkcji straty. Nasz model przewiduje ramki i etykiety siatki. W rozdziale 7. widzieliśmy, jak można zdefiniować wiele strat dla każdego wyjścia. Keras po prostu będzie sumował straty w trakcie treningu. Stratę klasyfikacji możemy obsługiwać jak zawsze z użyciem `sparse_categorical_crossentropy`.

Jednak strata ramki wymaga szczególnego traktowania. Podstawowa strata zaproponowana przez autorów modelu YOLO jest dość prosta. Używają błędu sumy kwadratów różnicy między docelowymi parametrami ramki a przewidywanymi. Błąd ten będziemy obliczać tylko dla komórek siatki z rzeczywistymi ramkami w oznaczonych danych.

Jeśli chodzi o stratę, wyzwanie stanowi wyjściowa pewność ramki. Autorzy chcieli, by wartość pewności odzwierciedlała nie tylko obecność obiektu, ale

również to, jak dobrze model przewidział ramkę. Aby określić, jak dobrze model przewidział ramki, autorzy proponują użycie *współczynnika nakładania*, czyli wskaźnika IoU, który poznaliśmy w poprzednim rozdziale. Jeśli komórka siatki jest pusta, wartość pewności powinna być równa 0. Lecz jeśli komórka siatki zawiera obiekt, to wartość pewności może być równa współczynnikowi nakładania bieżącej przewidzianej ramki i rzeczywistej ramki. W ten sposób, gdy model coraz lepiej przewiduje położenia ramek, wskaźnik IoU i wartości pewności rosną.

Wymaga to zastosowania niestandardowej funkcji straty. Możemy zacząć od zdefiniowania funkcji pomocniczej do obliczania wartości IoU dla ramek docelowych i przewidywanych (listing 12.10).

#### Listing 12.10. Obliczanie wskaźnika IoU dla dwóch ramek

```
from keras import ops

def unpack(box): <————— Rozpakowuje tensor z ramkami.
    return box[..., 0], box[..., 1], box[..., 2], box[..., 3]

def intersection(box1, box2): <————— Oblicza obszar nakładania się dwóch tensorów ramek.
    cx1, cy1, w1, h1 = unpack(box1)
    cx2, cy2, w2, h2 = unpack(box2)
    left = ops.maximum(cx1 - w1 / 2, cx2 - w2 / 2)
    bottom = ops.maximum(cy1 - h1 / 2, cy2 - h2 / 2)
    right = ops.minimum(cx1 + w1 / 2, cx2 + w2 / 2)
    top = ops.minimum(cy1 + h1 / 2, cy2 + h2 / 2)
    return ops.maximum(0.0, right - left) * ops.maximum(0.0, top - bottom)

def intersection_over_union(box1, box2): <————— Oblicza wskaźnik IoU dla dwóch
    cx1, cy1, w1, h1 = unpack(box1)                                tensorów ramek.
    cx2, cy2, w2, h2 = unpack(box2)
    intersection_area = intersection(box1, box2)
    a1 = ops.maximum(w1, 0.0) * ops.maximum(h1, 0.0)
    a2 = ops.maximum(w2, 0.0) * ops.maximum(h2, 0.0)
    union_area = a1 + a2 - intersection_area
    return ops.divide_no_nan(intersection_area, union_area)
```

Użyjmy tych funkcji pomocniczych, by zdefiniować naszą niestandardową stratę. Redmon i inni proponują kilka sztuczek w kwestii skalowania, aby poprawić jakość trenowania:

- Skalują stratę umiejscowienia ramki za pomocą współczynnika równego pięć, aby stała się ważniejszą częścią całego treningu.
- Ponieważ większość komórek siatki jest pusta, skalują w dół o współczynnik równy dwa stratę pewności w pustych miejscach. Dzięki temu wartości pewności równe zero nie przytłaczają straty.
- Przed obliczeniem straty wyciągają pierwiastek z szerokości i wysokości. Dzięki temu bardzo duże ramki mają takie samo znaczenie jak te mniejsze. My użyjemy funkcji `sqrt`, która zachowuje znak, gdyż na początku treningu nasz model może przewidywać ujemne szerokości i wysokości.

Napiszmy kod z wykorzystaniem tych sztuczek (listing 12.11).

**Listing 12.11. Definicja straty ramek ograniczających w modelu YOLO**

```
def signed_sqrt(x):
    return ops.sign(x) * ops.sqrt(ops.absolute(x) + keras.config.epsilon())

def box_loss(true, pred):
    xy_true, wh_true, conf_true = true[..., :2], true[..., 2:4], true[..., 4:]
    xy_pred, wh_pred, conf_pred = pred[..., :2], pred[..., 2:4], pred[..., 4:]
    no_object = conf_true == 0.0
    xy_error = ops.square(xy_true - xy_pred)
    wh_error = ops.square(signed_sqrt(wh_true) - signed_sqrt(wh_pred))
    iou = intersection_over_union(true, pred)
    conf_target = ops.where(no_object, 0.0, ops.expand_dims(iou, -1))
    conf_error = ops.square(conf_target - conf_pred)
    error = ops.concatenate(
        (
            ops.where(no_object, 0.0, xy_error * 5.0),
            ops.where(no_object, 0.0, wh_error * 5.0),
            ops.where(no_object, conf_error * 0.5, conf_error),
        ),
        axis=-1,
    )
    return ops.sum(error, axis=(1, 2, 3))
```

**Rozpakowuje wartości.**

**Oblicza błędy umiejscowienia ramki.**

**Oblicza błąd pewności.**

**Jeśli confidence\_true jest równe 0,0, to znaczy, że nie ma obiektu w tej komórce siatki.**

**Łączy błędy ze sztuczkami skalowania.**

**Zwraca jedną wartość straty na próbkę. Keras zsumuje wartości z wszystkich próbek.**

Teraz jesteśmy już gotowi i możemy rozpocząć trenowanie naszego modelu YOLO (patrz listing 12.12). Dla uproszczenia pominiemy wskaźniki. W rzeczywistości będziesz potrzebować wskaźników, m.in. dokładności modelu przy różnych progach pewności.

**Listing 12.12. Trenowanie modelu YOLO**

```
model.compile(
    optimizer=keras.optimizers.Adam(2e-4),
    loss={"box": box_loss, "class": "sparse_categorical_crossentropy"},
)
model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=4,
)
```

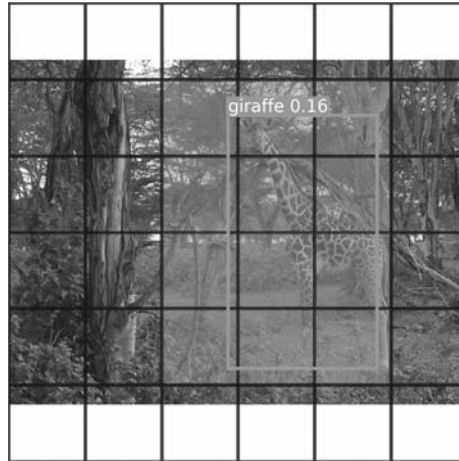
W darmowym środowisku wykonawczym Colab trenowanie trwa ponad godzinę (patrz listing 12.13), a nasz model dalej nie jest wytrenowany (strata walidacji wciąż spada!). Spróbujmy zwizualizować wyjście modelu (patrz rysunek 12.6). Zastosujemy niski próg pewności, gdyż model jeszcze niezbyt dobrze wykrywa obiekty.

## Listing 12.13. Trenowanie modelu YOLO

```

x, y = next(iter(val_dataset.rebatch(1))) ← Ponownie dzieli na partie wsadowe nasz zbiór danych,
preds = model.predict(x)                  ← by zamiast 16 próbek uzyskać jedną.
boxes = preds["box"][0]                   Znajduje najbardziej prawdopodobną etykietę
classes = np.argmax(preds["class"][0], axis=-1) ← w każdej komórce siatki za pomocą funkcji argmax.
path = metadata[0]["path"]                ← Wczytuje obraz z dysku, aby wyświetlić
draw_prediction(path, boxes, classes, cutoff=0.1) go w pełnym rozmiarze.

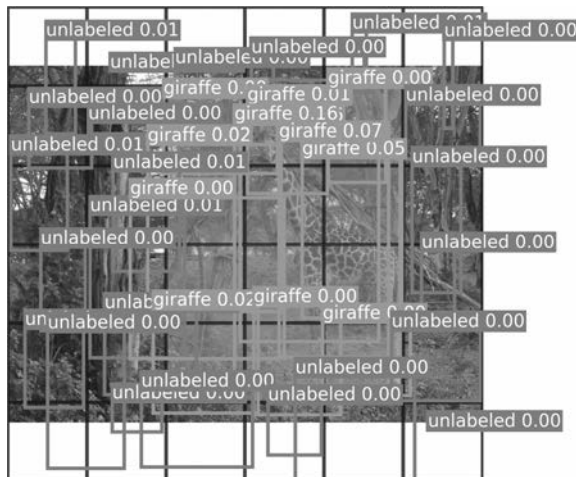
```



Rysunek 12.6. Przewidywania dla naszego przykładowego obrazu

Jak widać, nasz model zaczyna rozumieć położenia ramek i etykiety klas, ale wciąż nie jest bardzo dokładny. Zobaczmy każdą przewidzianą przez model ramkę (rysunek 12.7), nawet te, dla których wartość pewności jest równa zero:

```
draw_prediction(path, boxes, classes, cutoff=None)
```



Rysunek 12.7. Wszystkie ramki ograniczające przewidziane przez model YOLO

Nasz model zwraca bardzo niskie wartości pewności, gdyż jeszcze nie nauczył się lokalizować obiektów na obrazie w sposób spójny. Aby go ulepszyć, musimy spróbować kilku rzeczy:

- trenowanie modelu przez większą liczbę epok,
- użycie całego zbioru danych COCO,
- augmentacji danych (np. przesuwania i obracania obrazów wejściowych i ramek),
- ulepszenia naszej mapy prawdopodobieństw klas dla nakładających się ramek,
- przewidywania kilku ramek dla danej lokalizacji na siatce przy użyciu większej siatki wyjściowej.

Wszystko to pozytywnie wpłynie na działanie modelu i przybliży nas do oryginalnego modelu YOLO. Ale ten przykład ma jedynie pokazać, jak wygląda trenowanie modelu wykrywającego obiekty. Trenowanie pełnowartościowego modelu od podstaw wymagałoby ogromnej ilości zasobów i czasu. Zamiast tego, aby uzyskać wgląd w lepiej działający model wykrywania obiektów, spróbujmy użyć wstępnie wytrenowanego modelu o nazwie RetinaNet.

## 12.3. Używanie wstępnie wytrenowanego detektora RetinaNet

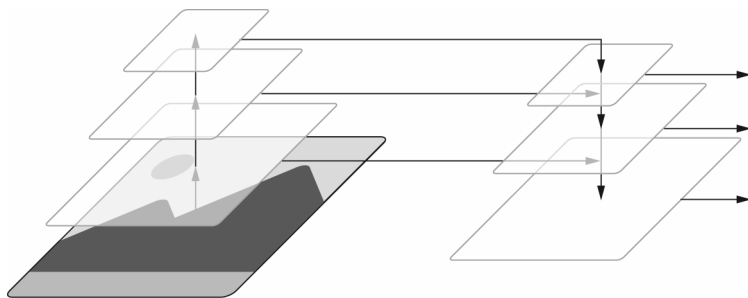
RetinaNet, tak jak model YOLO, również jest jednoetapowym detektorem i działa na tych samych zasadach. Model RetinaNet różni się od naszego modelu YOLO pod względem conceptualnym, gdyż w inny sposób wykorzystuje leżącą u jego podstaw sieć konwolucyjną, by lepiej obsługiwać jednocześnie małe i bardzo duże obiekty.

W modelu YOLO po prostu braliśmy ostateczne wyniki naszej sieci konwolucyjnej i używaliśmy ich do budowy detektora obiektów. Te wyjściowe cechy mapowały się na bardzo duże obszary na naszym obrazie wejściowym i w rezultacie nie były wydajne w znajdowaniu małych obiektów na obrazie.

Jeden sposób na rozwiązanie tego problemu ze skalowaniem mógłby polegać na bezpośrednim użyciu danych wyjściowych z wcześniejszych warstw w naszej sieci konwolucyjnej. Wyodrębniłoby to wysokiej rozdzielczości cechy mapujące się na małe obszary na naszym obrazie wejściowym. Jednak dane wyjściowe z tych warstw nie są *semantycznie istotne*. Mogą się mapować na różne rodzaje prostych cech, takich jak krawędzie i krzywe, a dopiero w późniejszych warstwach konwolucyjnych zaczynamy budować ukryte reprezentacje całych obiektów.

Rozwiązanie zastosowanie przez model RetinaNet nosi nazwę sieci piramidy cech. Za pomocą warstwy Conv2DTranspose rozdzielczość ostatecznych cech z bazowego modelu sieci konwolucyjnej jest zwiększana, tak jak robiliśmy

to w poprzednim rozdziale. Co ważne, uwzględniamy również *ukryte połączenia*, w których te mapy cech o zwiększonej rozdzielczości sumujemy z mapami cech o tym samym rozmiarze z pierwotnej sieci konwolucyjnej. Łączymy w ten sposób cechy istotne semantycznie o niskiej rozdzielczości z końca sieci konwolucyjnej z drobnoskalowymi cechami o wysokiej rozdzielczości z początku sieci konwolucyjnej. Na rysunku 12.8 przedstawiono ogólny zarys tej architektury.



**Rysunek 12.8.** Sieć piramidy cech tworzy mapy cech istotnych semantycznie w różnych skalach

Sieci piramid cech mogą znacząco ulepszyć działanie modelu przez budowanie wydajnych cech zarówno dla małych, jak i bardzo dużych obiektów pod względem liczby zajmowanych pikseli. Najnowsze wersje modelu YOLO również stosują taką konfigurację.

Wypróbujmy zatem model RetinaNet, który został wytrenowany na zbiorze danych COCO. Aby było ciekawiej, użyjemy obrazu spoza zbioru — obrazu namalowanego techniką puentylizmu, zatytułowanego *Niedzielne popołudnie na wyspie Grande Jatte*.

Najpierw pobierzmy obraz i przekształćmy go w tablicę NumPy:

```
url = (
    "https://upload.wikimedia.org/wikipedia/commons/thumb/7/7d/"
    "A_Sunday_on_La_Grande_Jatte%2C_Georges_Seurat%2C_1884.jpg/"
    "1280px-A_Sunday_on_La_Grande_Jatte%2C_Georges_Seurat%2C_1884.jpg"
)
path = keras.utils.get_file(origin=url)
image = np.array([keras.utils.load_img(path)])
```

Następnie pobierzmy model i użyjmy go (patrz listing 12.14). Tak jak w poprzednim rozdziale, możemy użyć interfejsu API wysokopoziomowego zadania ObjectDetector — przetwarzanie wstępne jest już w nim uwzględnione.

#### Listing 12.14. Tworzenie modelu ResNet

```
detector = keras_hub.models.ObjectDetector.from_preset(
    "retinanet_resnet50_fpn_v2_coco",
    bounding_box_format="rel_xywh",
)
predictions = detector.predict(image)
```

Jak pewnie zauważyłeś, przekazujemy dodatkowy argument, by określić format ramki ograniczającej. W Keras można to zrobić dla większości modeli i warstw obsługujących ramki ograniczające. Tutaj przekazujemy "rel\_xywh", by format był taki sam jak w przypadku modelu YOLO, dzięki czemu możemy używać tych samych funkcji pomocniczych do rysowania co wcześniej. Człon rel oznacza „w odniesieniu do rozmiaru obrazu” (np. [0, 1]). Sprawdźmy, co przewidział model:

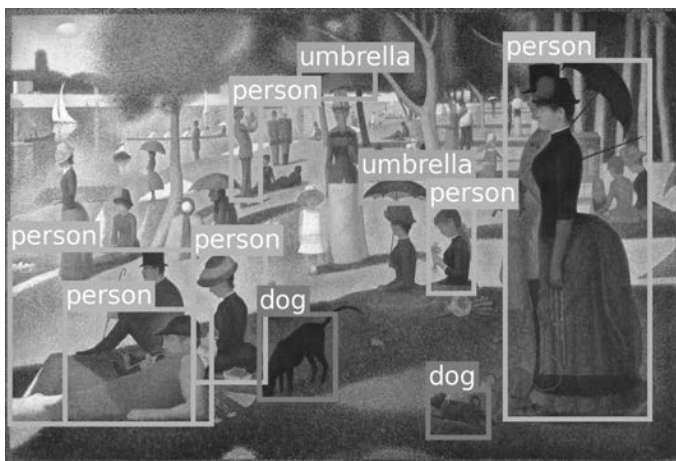
```
>>> [(k, v.shape) for k, v in predictions.items()]
[("boxes", (1, 100, 4)),
 ("confidence", (1, 100)),
 ("labels", (1, 100)),
 ("num_detections", (1,))]
>>> predictions["boxes"][0][0]
array([0.53, 0.00, 0.81, 0.29], dtype=float32)
```

Model zwrócił cztery różne elementy: ramki ograniczające, wartości pewności, etykiety i całkowitą liczbę wykrytych obiektów. Nasz model YOLO zwraca podobne informacje. Potrafi wykryć do 100 obiektów na jednym obrazie wejściowym.

Wyświetlmy to, co zwrócił model, za pomocą naszych funkcji pomocniczych do rysowania (patrz listing 12.15 i rysunek 12.9).

#### Listing 12.15. Wnioskowanie z RetinaNet

```
fig, ax = plt.subplots(dpi=300)
draw_image(ax, path)
num_detections = predictions["num_detections"][0]
for i in range(num_detections):
    box = predictions["boxes"][0][i]
    label = predictions["labels"][0][i]
    label_name = keras_hub.utils.coco_id_to_name(label)
    draw_box(ax, box, label_name, label_to_color(label))
plt.show()
```



**Rysunek 12.9.** Obiekty znalezione przez model RetinaNet na obrazie testowym

Model RetinaNet potrafi z łatwością generalizować obraz namalowany techniką puentylizmu, mimo że nie był trenowany na tego typu obrazach! To tak naprawdę jedna z zalet jednoetapowych detektorów obiektów. Obrazy i zdjęcia bardzo się różnią na poziomie pikseli, ale na wyższym poziomie mają podobną strukturę. Z kolei dwuetapowe detektory, takie jak R-CNN, muszą osobno klasyfikować małe obszary obrazu wejściowego, co stanowi dodatkową trudność, gdy małe obszary pikseli różnią się od tego, co model widział podczas trenowania. Detektory jednostopniowe mogą czerpać z cech całego wejścia i są bardziej odporne na nowe dane wejściowe podczas testów.

W ten sposób dotarliśmy do końca części tej książki poświęconej widzeniu komputerowemu! Wytrenowaliśmy od podstaw modele klasyfikujące obrazy, segmentujące obrazy i wykrywające obiekty. Na tym etapie powinieneś już intuicyjnie wyczuwać, jak działają sieci konwolucyjne, które były pierwszym wielkim sukcesem uczenia głębokiego. Ale jeszcze nie skończyliśmy pracować z obrazami i wrócimy do nich ponownie w rozdziale 17., w którym będziemy generować obrazy.

## Podsumowanie

- Proces wykrywania obiektów znajduje obiekty i wskazuje ich położenie na obrazie za pomocą ramek ograniczających. Jest to najsłabsza odmiana segmentacji obrazów, ale z reguły jest o wiele wydajniejsza.
- Są dwa główne podejścia do wykrywania obrazów:
  - Neuronowe sieci konwolucyjne oparte na regionach (R-CNN), które są dwuetapowymi modelami. Na pierwszym etapie modele te proponują regiony, a na drugim klasyfikują je za pomocą sieci konwolucyjnej.
  - Detektory jednoetapowe (takie jak RetinaNet i YOLO) wykonują te dwa wyżej wymienione zadania za jednym razem. Takie detektory są zwykle szybsze i wydajniejsze, dzięki czemu lepiej nadają się do pracy w czasie rzeczywistym (np. w pojazdach autonomicznych).
- Model YOLO podczas trenowania zwraca jednocześnie dwa różne wyjścia: możliwe ramki ograniczające i mapę prawdopodobieństwa klas:
  - Dla każdej zaproponowanej ramki ograniczającej model podaje wartość pewności, opartą na wskaźniku IoU dla przewidzianej ramki i rzeczywistej ramki ograniczającej.
  - Mapa prawdopodobieństwa klas klasyfikuje różne obszary obrazu jako należące do różnych obiektów.
- RetinaNet bazuje na tej koncepcji, lecz używa sieci piramidy cech (FPN, ang. *feature pyramid network*), która łączy cechy z kilku sieci konwolucyjnych, by tworzyć mapy cech o różnych skalach, co pozwala dokładniej wykrywać obiekty różnych wielkości.



# Skorowidz

## A

- abstrakcja, 615, 624
  - skoncentrowana na programach, 625, 626
  - skoncentrowana na wartościach, 624, 626
- adnotacje, 251
- AGI, Artificial General Intelligence, 39, 607
- AI, Artificial Intelligence, 26, 637
- aktywacja, 322
- algorytm
  - Adagrad, 75
  - propagacji wstecznej, 35, 77
  - regresji logistycznej, 157
  - RLHF, 532
  - RMSProp, 75
  - SGD, 75, 76
  - spadku gradientowego, 329
  - spadku gradientu mini-batch, 89
  - stochastycznego spadku wzdłuż gradientu, 75
  - stochastycznego wznoszenia wzdłuż gradientu, 332
  - walidacji k-krotnej, 161
  - wzrostu gradientu, 603
- AlphaGo, 630, 634
- antropomorfizacja, 606
- architektura
  - autoenkodera, 559
  - sieci konwolucyjnych
    - hierarchia, 306
    - modułowość, 306
    - normalizacja wsadu, 312
    - połączenie rezydualne, 309
    - ponowne użycie, 306
    - separowalne konwolucje po kanałach, 314
  - transformera, 127, 473
  - w stylu Xception, 318

- architektury
  - modeli, 643
  - uczenia głębokiego, 308
- augmentacja danych, 278, 288–292
  - tekstowych, 450
- autoenkodery
  - architektura, 559
  - odszumiające, 556, 557
    - typu U-Net, 558
  - wariacyjne, VAE, 545, 547
    - implementacja z Keras, 550
- automaty, 608
  - behawioralne, 612
- automatyczna optymalizacja
  - hiperparametrów, 580
- automatyczne
  - różniczkowanie, 46
  - wnioskowanie kształtu, 124, 126
- automatyzacja, 610
  - kognitywna, 40

## B

- baza konwolucyjna Xception, 329
- biblioteka
  - Keras, 45, 120
  - KerasHub, 295, 491, 494
  - Matplotlib, 101, 143
  - NumPy, 46, 99
  - SentencePiece, 508
  - TensorFlow Serving, 220
- bigramy, 438
- BLAS, Basic Linear Algebra Subprograms, 59
- błąd
  - przewidywań, 135
  - średniokwadratowy, 216

budowanie modelu  
   dostrajanie, 217  
   konfiguracja trenowania, 215  
   miniGPT, 509  
   ocena wydajności, 215  
   przez dziedziczenie, 229, 239–241  
   przez ponowne użycie wyjść, 238  
   przygotowywanie danych, 213  
   regularyzacja, 217  
   ResNet, 382  
   segmentującego obraz, 350  
   sekwencyjnego opartego na LSTM, 443, 447, 454  
   selekcja cech, 215  
   skalowanie w górę, 217  
   tekst-do-obrazu, 567  
   transformera, 484, 487  
   typu ciągły worek słów, 453  
   wybór  
     architektury, 215  
     funkcji aktywacji, 216  
     funkcji straty, 216  
     wskaźników, 216  
 YOLO, 371  
 za pomocą  
   interfejsu API, 229, 232–237, 241  
   klasy Sequential, 229–231

## C

CBOW, Continuous Bag of Words, 451  
 ConvNet, 267, 643

## D

dane, 130  
   docelowe, 135  
   oznaczanie, 208  
   przygotowywanie, 213  
     normalizacja, 213  
     obsługa brakujących wartości, 214  
     wektoryzacja, 213  
   reprezentatywność, 183, 209  
   sekwencyjne, 54, 55  
   szeregu czasowego, 54, 55  
   treningowe, 178  
   treningowe zaszumione, 168  
   w postaci obrazów, 56  
   walidacyjne, 131  
   wyjściowe, 29

  wektorowe, 54, 55  
   wideo, 57  
   wizualizowanie, 211  
   wyjściowe, 135  
 dekodery, 500, 546  
   sekwencja do sekwencji, 470  
   VAE, 552  
 dekorator  
   @jax.jit, 118  
   @tf.function, 118  
 destylacja  
   danych, 47  
   informacji, 33  
 detektor obrazów RetinaNet, 381  
 detektory obiektów  
   dwuetapowe, 365  
   dwuetapowe R-CNN, 366  
   jednoetapowe, 365, 367  
 dokładność, 48, 49, 212  
   trenowania i walidacji, 144, 152, 153  
 dopasowanie zbyt słabe, 167  
 dostrajanie, 180, 301, 304  
   na instrukcjach, 524, 544  
   nadzorowane, 531  
   parametrów spadku gradientu, 185  
   RLHF, 531  
   za pomocą KerasTuner, 576  
 dropout, 407  
 dryf koncepcji, concept drift, 210, 224  
 duże modele językowe, LLM, 504  
 dyfuzja, 557  
   odwrócona, 557

## E

ekstrakcja cech, 278, 288, 292, 293, 298, 301, 304  
   bez augmentacji danych, 296  
   z augmentacją danych, 299  
 enkoder, 479, 500  
   sekwencja do sekwencji, 469  
 entropia krzyżowa, 141, 156  
   binarna, 216  
   kategoryczna, 216  
 epoka, 84, 88, 130  
 etyka, 207  
 etykieta, 46, 101, 135, 208, 251  
 ewaluacja modelu  
   LSTM, 455  
   regresyjnego, 440

**F**

filtr danych, 47  
 FPN, feature pyramid network, 384  
 framework  
   JAX, 112  
   PyTorch, 105  
   TensorFlow, 94  
 frameworki  
   dla backendu, 121  
   uczenia głębokiego, 91, 93  
 funkcje  
   aktywacji  
     relu, 140  
     sigmoid, 140, 216  
     softmax, 216  
   celu, 35  
   ciągłość, 71  
   gładkie, 71  
   kosztu, 35  
   krzywizna, 73  
   pochodna, 70  
   różniczkowalność, 70, 71  
   straty, 35, 47, 48, 89, 128–130, 165, 637  
     binarna entropia krzyżowa, 216  
     błąd średniokwadratowy, 102, 119, 159, 216  
     kategoryczna entropia krzyżowa, 216  
 funkcjonalny interfejs API, 229, 232, 234–237, 240–242

**G**

generalizacja, 167, 201, 605  
   ekstremalna, 610, 611  
   lokalna, 176, 610, 611  
   porządkowanie zbioru danych, 190  
   przetwarzanie cech, 191  
   regularyzacja modelu, 193  
   szeroka, 613  
   w uczeniu głębokim, 173  
   wczesne zatrzymanie trenowania, 192  
 generator, 546  
   partii danych, 85  
 generatywna sztuczna inteligencja, 37, 504, 637  
 generowanie  
   obrazu, 545–573  
     autoenkodery wariacyjne, 547  
     modele dyfuzyjne, 556  
     sterowane tekstem, 546

sekwencji, 504  
 tekstu, 502–544  
 tekstu z modelem Gemma, 521  
 wspomagane wyszukiwaniem, RAG, 538  
 wyjść, 573  
 geometryczna interpretacja  
   operacje tensorowe, 64  
   uczenie głębokie, 68  
 gęste próbkowanie, 178, 190, 611  
 giełda, 415  
 Google Colaboratory, 45, 587  
 GPT, Generative Pretrained Transformer, 505  
 gradienty, 70, 72  
   drugiego rzędu, 98  
   obliczanie, 339, 340  
   w JAX, 116, 117  
   w PyTorch, 108  
   w TensorFlow, 97  
   zanikające, 307  
 graf obliczeniowy, 78, 79

**H**

harmonogram dyfuzji, 560–562  
 hierarchia cech, 307  
 hiperparametry, 575  
   automatyczna optymalizacja, 580  
   modelu, 180  
   optymalizacja, 575  
 hipoteza  
   kalejdoskopowa, 614  
   rozmaitości, 174

**I**

iloczyn  
   skalarny, 475  
   macierzy, 62  
   tensorów, 63  
   wektorów, 61  
   tensorowy, 61  
 indeksowanie, 421, 456  
 inteligencja, 614  
   dostrzeganie abstrakcji, 615, 623  
   rekombinacja abstrakcji, 615, 623  
 inteligentne agenty, 608, 612  
 interfejs  
   API Keras, 228, 229, 232, 234–237, 240–242  
   API LayoutMap, 589  
   REST API, 220

interpolacja, 175  
     między dwoma promptami, 573  
     sferyczna, 572  
 interpolacyjna baza danych, 499  
 inżynieria  
     cech, 36, 191, 215  
     promptów, 604, 605  
 IoU, Intersection over Union, 352  
 izomorfizm podgrafów, 625

## J

JAX, 112–120  
     gradienty, 116, 117  
     obliczanie gradientów, 340  
     obliczenia matematyczne, 116  
     przypisywanie wartości do tensora, 115  
     równoległość danych, 588  
     równoległość modelu, 588  
     tensory, 113  
     tensory losowe, 114  
     unikatowość, 120  
     wektoryzacja kodu, 115  
     wznoszenie gradientu, 332  
 jądro konwolucji, 273  
 Jupyter, 45

## K

Keras, 120–133, 227–265, 651  
     automatyczne wnioskowanie kształtu, 124, 126  
     backendy, 121  
     budowa modeli, 229  
     dostęp  
         do ONNX, 220  
         do TensorFlow Serving, 220  
     funkcja straty, 129, 130  
     implementacja  
         autoenkodera wariacyjnego, 550  
         warstw rekurencyjnych, 647  
     klasa  
         Layer, 123  
         Model, 126  
     kompilacja, 128  
     metoda fit, 130  
     optymalizatory, 129  
     pętla  
         ewaluacyjna, 242  
         trenowania, 130

pisanie pętli  
     ewaluacji, 251  
     trenowania, 251  
 przepływy pracy, 228  
 warstwy  
     augmentacji danych, 289  
     gęste, 124  
     rekurencyjne, 402  
     wbudowane trenowanie, 242  
     wskaźniki, 129  
 KerasHub, 528  
 KerasTuner, 576  
 k-krotna walidacja  
     iteracyjna, 215  
     krzyżowa, 181, 215  
     z losowaniem, 182  
 klasa, 46, 135  
     Callback, 248  
     Dataset, 285–287  
     Layer, 123, 227  
     Metric, 244  
     Model, 126, 239  
     Parameter, 107  
     Sequential, 47, 83, 85, 227, 229  
     TensorBoard, 248  
     tf.GradientTape, 98  
     tf.Variable, 96  
     Tokenizer, 492  
     torch.nn.Module, 110  
 klasyfikacja  
     artykułów prasowych  
         budowanie modelu, 149  
         funkcja straty, 154  
         obsługa etykiet, 154  
         przewidywanie tematów, 154  
         przygotowywanie danych, 148  
         walidacja modelu, 150  
         warstwy pośrednie, 155  
         zbiór danych Agencji Reutera, 147  
     binarna, 135, 136, 165, 216, 643  
     kategorialna  
         jednoetykietowa, 644  
         wieloetykietowa, 644  
     liniowa w PyTorch, 108  
     obrazów, 45–49, 82–88, 266–304, 345  
         architektura, 47  
         dane wejściowe, 83  
         dopasowywanie modelu, 48  
         implementacja, 84

- klasyfikacja
    - obrazów
      - kompilacja, 48, 83
      - krok aktualizowania wag, 86
      - krok trenowania, 86
      - obliczenie gradientu, 87
      - ocena działania, 49, 88
      - pętla trenowania, 88
      - przygotowywanie danych, 48
      - schemat zależności, 82
      - użycie modelu, 48
      - zbiór danych MNIST, 46
    - recenzji filmów, 134, 136
      - budowa modelu, 138
      - przewidywanie recenzji, 145
      - przygotowywanie danych, 137
      - trenowanie modelu, 142
      - uczenie modelu, 145
      - walidacja modelu, 142
      - zbiór danych IMDb, 136
    - tekstu, 417–456, 574–599
      - modele oparte na zbiorach, 434
      - modele sekwencyjne, 440
      - przygotowanie danych tekstowych, 420
      - sekwencje, 431
      - zbiory, 431
    - wielokategorialna, 135
    - wieloklasowa, 134, 135, 147, 165
      - jednoetykietowa, 147, 216
      - kodowanie etykiet, 156
      - wieloetykietowa, 135, 147, 216
    - z użyciem transformera, 489
  - klasyfikator liniowy, 101
  - klucze, keys, 114
  - koder VAE, 548, 549, 573
    - straty regularyzacji, 549
    - straty rekonstrukcji, 549
  - kodowanie
    - etykiet, 148
      - za pomocą kodowania
        - kategorycznego, 156
      - za pomocą wartości
        - całkowitoliczbowych, 156
  - kategorialne, 148
  - liczb zmiennoprzecinkowych, 594
  - one-hot, 148
  - oparte na bigramach, 439
  - par bajtów, 427
  - pozycyjne, 645
  - typu worek słów, 435
  - kompilacja
    - PyTorch, 111
    - TensorFlow, 99
  - konwolucje
    - kroczące, 275
    - separowalne po kanałach, 315
  - konwolucyjna baza modelu, 294
  - krok, stride, 275
  - dyfuzji, 560
- L**
- lematyzacja, 423
  - LLM, large language model, 504
  - logit, 514
  - LoRA, 527, 544
  - losowa inicjalizacja, 69
  - LSTM, Long Short-Term Memory, 398
- M**
- macierz, 51
    - kolumny, 51
    - transpozycja, 63
    - wiersze, 51
  - mapy
    - cech, 271
      - istotnych semantycznie, 382
      - technika dopełniania, 274
      - zmniejszanie rozdzielczości, 276
    - ciepła aktywacji klasy, 336, 341, 342
    - odpowiedzi, 272
    - prawdopodobieństw klasy, 373
  - maski segmentacji, 348
  - materiały wideo, 55
  - mechanizm
    - samouwagi, 480
    - uwagi, 473, 498, 501
      - iloczyn skalarny, 475
      - założenia, 474
      - zmiany, 478
  - metaprogramowanie, 116
  - metauczenie, 633
  - metoda
    - compile(), 243, 262, 263
    - evaluate(), 240, 243, 265
    - fit(), 228, 234, 240, 243–246, 249, 251, 258–262, 265, 286

- model(), 331
- predict(), 240, 243, 330
- metody wyszukiwania, 621
- MHR, modularity-hierarchy-reuse, 306
- mini-batch SGD, 74
- miniGPT, 506
  - budowanie modelu, 509
- minimum
  - globalne, 76
  - lokalne, 76
- mnożenie macierzy, 61
- moc
  - reprezentacyjna, 188
  - statystyczna modelu, 215
- model
  - BERT, 490
  - Gemma, 521, 534
    - adaptacja niskiego rzędu, 526
    - dostęp do wag, 521
    - dostrajanie, 529
    - dostrojony na instrukcjach, 524, 533
    - generowanie tekstu, 521
    - obraz testowy, 536
    - wczytanie modelu, 533
  - GPT-3, 506
  - LLM, 531
    - dostrajanie RLHF, 531
    - kluczowe elementy, 543
    - liczba parametrów, 542
    - multimodalny, 534
    - wady, 538
  - miniGPT
    - budowanie, 512
    - dekodowanie generatywne, 514
    - strategie próbkowania, 517
    - trenowanie, 506, 513
    - wstępne trenowanie, 512
  - R-CNN, 366
  - ResNet, 373
    - tworzenie, 382
    - wnioskowanie, 383
  - RetinaNet, 384
  - RoBERTa, 491–494
    - dostrajanie, 496
    - trenowanie, 497
    - warstwy, 493
    - wstępne przetwarzanie recenzji, 494
    - wstępnie wytrenowany, 492
  - SAM, 355
    - architektura, 357
    - działanie, 355
    - obraz testowy, 357
    - pobieranie modelu, 355
    - promptowanie z podaniem punktu docelowego, 359
    - promptowanie z podaniem ramki, 362
    - segmentacja obrazu, 354
  - Stable Diffusion, 568
  - Stable Diffusion 3, 573
  - VAE, 552
    - metoda compute\_loss(), 553
    - próbkowanie siatki punktów, 554
    - trenowanie, 554
  - Word2Vec, 497
  - Xception
    - architektura, 293, 308, 315
    - baza konwolucyjna, 295, 329
    - dostrajenie bazy konwolucyjnej, 302
    - filtry, 329
    - nazwy warstw konwolucyjnych, 329
    - wstępna obróbka obrazu, 337
    - wstępne przetwarzanie, 296
  - YOLO, 367, 384
    - budowanie, 371
    - obliczanie wskaźnika IoU, 378
    - pobieranie danych COCO, 368
    - przygotowanie danych COCO, 368, 374
    - ramki ograniczające, 379
    - sprawdzanie danych COCO, 369
    - trenowanie, 367, 377, 379
    - tworzenie wartości docelowych, 375
    - wizualizacja danych, 376
    - wyświetlanie obrazu, 370
- modele
  - „rozumujące”, 539
  - bazowe, 537, 544
  - bigramowe, 438
  - dyfuzyjne, 537, 556, 557, 643
    - generowanie obrazów, 564
    - trenowania, 562
    - trenowanie, 565
    - wizualizacja rezultatów, 564
  - dzielenie między procesory GPU, 585

## modele

- językowe, 457
  - autoregresyjne, 462
  - prompty, 462
  - przewidywanie jednego tokena, 463
  - przyczynowe, 490
  - sekwencja do sekwencji, 500
  - trenowanie, 458
  - uczenie sekwencja do sekwencji, 464
- maskowanego języka, 500
- odpowiednia przestrzeń przeszukiwań, 581
- oparte na zbiorach, 456
- optymalizacja hiperparametru, 575
- przyspieszanie trenowania i wnioskowania, 593
- rekurencyjne, 442
- sekwencyjne, 456
- skalowanie w górę, 607
- składanie, 581
- tekst-do-obrazu, 567
  - badanie przestrzeni ukrytej, 569
- typu worek słów, 434
  - ciągłe, 451, 452
  - regresyjne, 435
- uczenia głębokiego, 647
- modelowanie języka maskowanego, 490
- monitorowanie, 248
- multimodalne modele LLM, 534

**N**

- naddatek danych, 184
- nadmierne dopasowanie, 49, 145, 166–168, 180, 193, 197, 407, 580
- narzędzie KerasTuner, 576
- NLP, natural language processing, 418
- normalizacja
  - cech, 158, 213, 252
  - danych, 158, 390
  - wsadu, 312
- notacja sumowania Einsteina, 474
- NumPy
  - implementacja sieci RNN, 401
  - mnożenie macierzy, 61
  - obsługa tensorów, 53

**O**

- obliczanie
  - gradientów
    - z JAX, 340
    - z PyTorch, 339
    - z TensorFlow, 339
  - wskaźnika IoU, 378
- obrazy, 55
  - poprawa rozdzielczości, 556
  - usuwanie szumu, 556
- ONNX, Open Neural Network Exchange, 220
- operacje
  - na tensorach, 57, 64, 89
  - wykonywane element po elemencie, 58
- opisy próbek, 135
- optymalizacja, 167, 201
- optymalizator, 35, 48, 75, 89, 128, 129
- osadzanie, 456
- osadzenia słów, 444, 446
- oś, 50
  - cech, 55
  - czasu, 184
  - próbek, 54, 55
  - rzutowania, 60
  - wsadu, 54
- oznaczanie danych, 208

**P**

- parametry, 34, 69, 180, 575
  - w PyTorch, 106, 107
- pętla
  - ewaluacyjna, 251, 393
  - trenowania, 36, 69, 103, 130, 251
  - wizualizacji filtra, 333
- pochodna
  - funkcji, 71
  - operacji tensorowej, 72
- podział, 456
- pojemność modelu, 188
- pole pod wykresem krzywej ROC, 212
- porzucanie, dropout, 199
- potok oczyszczający informację, 328
- prawda, 135
- problem zanikających gradientów, 403
- procedury numeryczne BLAS, 59
- procesor
  - CPU, 121
  - GPU, 121, 587
  - TPU, 121, 592

- prognozowanie, 386
    - szeregów czasowych, 385–416
    - temperatury, 386
      - przygotowanie danych, 390
      - punkt odniesienia, 393
      - wykorzystanie jednowymiarowego modelu konwolucyjnego, 396
      - wykorzystanie podstawowego modelu, 394
  - program Midjourney, 503
  - programowanie genetyczne, 628
  - promptowanie, 37
  - prompty, 462
    - z podaniem punktu, 359
    - z podaniem ramki, 359
  - propagacja wsteczna, 89
  - próbka, 46
    - danych wejściowych, 135
  - próbkowanie
    - losowe, 519, 544
    - top-K, 519, 544
  - przechodzenie
    - w przód, 69, 74, 79, 102
    - w tył, 74, 80
  - przekształcenia afiniczne, 66, 101
  - przekształcenie liniowe, 66
  - przepływ pracy
    - budowa modelu, 204, 212
    - budowa modelu Keras, 229
    - definicja zadania, 204
    - publikowanie modelu, 204
    - w Keras, 228
    - wdrażanie modelu, 218
    - własne wskaźniki, 244
    - wywołania zwrotne, 245
  - przestrzeń
    - hipotez, 32, 126, 141, 305
    - możliwości, 126
    - osadzeń, 498
    - poszukiwań, 576
    - przeszukiwań, 581
  - przesunięcie, 65
  - przetrenowanie, 177
  - przetwarzanie języka naturalnego, NLP, 418
  - przewidywanie, 135, 252
    - cen mieszkań, 156, 164
    - prawdopodobieństw klas, 48
    - recenzji, 145
    - tematów prasowych, 154
  - PyTorch, 105–112
    - gradienty, 108
    - klasa Parameter, 107
    - klasa torch.nn.Module, 110
    - klasyfikacja liniowa, 108
    - kompilacja, 111
    - obliczanie gradientów, 339
    - obliczenia matematyczne, 107
    - parametry, 106
    - przypisywanie wartości do tensora, 107
    - tensory losowe, 106
    - tensory stałe, 106
    - tworzenie parametru, 107
    - unikatowość, 111
    - wznoszenie gradientu, 332
- ## R
- RAG, retrieval-augmented generation, 538
  - ramki ograniczające, bounding boxes, 346, 364
  - rdzeń modelu, backbone, 492
  - regresja, 134, 156, 216, 644
    - budowanie modelu, 159
    - k-krotna walidacja krzyżowa, 160
    - logistyczna, 157, 187
    - przewidywanie cen mieszkań, 156, 164
    - przygotowywanie danych, 158
    - skalarna, 135, 165
    - wektorowa, 135
    - zbiór cen domów, 157
  - regularyzacja, 179
    - modelu, 193, 202
      - porzucanie, 199
      - zmniejszenie rozmiaru modelu, 193
    - wag, 197
  - reguła łańcuchowa, 77, 81
  - rekurencyjna technika dropout, 407
  - replika, 585
  - reprezentacja danych, 30, 33, 50
    - macierze, 51
    - skalary, 50
    - tensory trójwymiarowe, 51
    - wektory, 50
  - reprezentacje koncepcji graficznych, 321
  - reprezentatywność danych, 183
  - REST API, 220, 221
  - RLHF, Reinforcement Learning with Human Feedback, 531
  - rozkład prawdopodobieństwa, 149

rozmaitości, 343  
rozwiązywanie zadania  
  definiowanie problemu, 205  
  wybór miary sukcesu, 212  
  zbieranie danych, 207  
  zrozumienie danych, 211  
równoległość modelu, 585  
różniczkowalność, 70  
różniczkowanie  
  automatyczne, 46, 78  
  symboliczne, 82  
rzutowanie, broadcasting, 60

## S

SAM, Segment Anything Model, 354  
segmentacja obrazu, 345–363  
  instancji, 347  
  panoptyczna, 348  
  semantyczna, 347  
selekcja cech, 172  
serwis  
  arXiv, 650  
  Kaggle, 650  
sferyczna interpolacja liniowa, 571  
siatka urządzeń, 588  
sieci neuronowe  
  gęsto połączone, 643  
  operacje na tensorach, 57  
  optymalizacja gradientowa, 69  
  piramidy cech, FPN, 382, 384  
  reprezentacja danych, 50  
sieć neuronowa, 32, 640  
  jednokierunkowa, 399  
  konwolucyjna, CNN, 266, 645  
  dostrajanie, 301, 304  
  dostrajanie częściowe, 302  
  działanie, 270  
  ekstrakcja cech, 293, 297, 304  
  hierarchia wzorców, 271  
  mapy cech, 271  
  mapy odpowiedzi filtra, 272  
  o separowalnej głębokości, 645  
  ocena modelu, 268, 288  
  oparta na regionach, R-CNN, 366  
  operacja max-pooling, 276  
  podsumowanie modelu, 276  
  połączenia rezydualne, 645  
  technika augmentacji danych, 288  
  trenowanie, 269, 278

tworzenie, 267, 282  
warstwy normalizujące partie  
  wsadowe, 645  
wizualizacja, 322, 329, 336, 342  
wskaźniki trenowania i walidacji, 288  
wstępna obróbka danych, 284  
wykorzystanie modelu, 293  
wzorce, 305  
z regularyzacją, 291  
z warstwą odrzucania, 291  
zamrażanie warstwy, 299  
rekurencyjna, RNN, 398, 400, 643, 646  
  dwukierunkowa, 412  
  działanie, 399  
  generowanie tłumaczeń, 471  
  trenowanie sekwencja do sekwencji, 468  
  wydajność, 409  
skalar, 50  
skalowanie, 36, 66  
  max-pooling, 275  
  modeli, 607  
  straty z mieszaną precyzją, 596  
  w górę, 608  
składanie modeli, 581  
słownik, 421, 429, 460  
  tworzenie, 467  
spektrum generalizacji, 613  
standaryzacja, 456  
  tekstu, 422  
stemming, 423  
stochastyczny spadek wzdłuż gradientu, 70, 73, 637  
  mini-batch, 74, 89  
strata, 89  
  treningowa, 132, 143, 151  
  walidacyjna, 132, 143, 151  
stronniczość próbkowania, 210  
synteza programów, 628  
szeregi czasowe, 385  
  klasyfikacja, 386  
  prognozowanie, 386  
  wykrywanie zdarzeń, 386  
sztuczna inteligencja, AI, 26, 600–635, 637  
  generatywna, 37, 504, 637  
  ogólna, AGI, 39, 635

## Ś

- średni błąd
  - bezwzględny, 160, 163, 395, 409, 411
  - kwadratowy, 159

## T

- technika
  - LoRA, 527, 544
  - uśredniania Polyak, 565
- tensor, 50, 89
  - czterowymiarowy, 55, 56
  - danych, 54
  - dodawanie, 64
  - dwuwymiarowy, 51, 54
  - jednowymiarowy, 50
  - konwertowanie na obraz, 333
  - krojenie, 53
  - kształt, 52
  - losowy, 96, 106, 114
  - obrót, 65
  - operacje wykonywane element
    - po elemencie, 58
  - osie, 50
  - oś próbek, 54
  - oś wsadu, 54
  - pięciowymiarowy, 55
  - przypisywanie wartości, 96, 107, 115
  - rząd, 50–52
  - rzutowanie, 59
  - skalarny, 50
  - skalowanie, 66
  - stały, 95, 106
  - symboliczny, 233
  - trójwymiarowy, 51, 54, 123, 268
  - tworzenie, 95
  - typ danych, 52
  - w JAX, 113
  - w Numpy, 53
  - w PyTorch, 106
  - w TensorFlow, 95
  - zerowymiarowy, 50
  - zmiana kształtu, 63
- TensorBoard, 248–250
- TensorFlow, 94–105
  - gradienty, 97
  - klasa
    - Dataset, 285
    - tf.GradientTape, 98
    - tf.Variable, 96
  - kompilacja
    - tryb domyślny, 99
    - z XLA, 99
  - obliczanie gradientów, 339
  - operacje na tensorach, 97
  - pętla trenowania, 103
  - przypisywanie wartości do tensora, 96
  - Serving, 220
  - tensory losowe, 96
  - tensory stałe, 95
  - unikatowość, 104
  - wznoszenie gradientu, 331
- test Turinga, 28
- tokenizacja, 456
  - na podsłowa, 426, 456
  - na słowa, 423, 456
  - na znaki, 423, 456
  - wybór techniki, 430
- tokenizer, 423
- tokeny, 420
- TPU, Tensor Processing Unit, 592
- transformer, 495, 501, 643, 645
  - architektura, 127, 473
  - blok dekodera, 482
  - blok enkodera, 479, 480
  - budowanie modelu, 484
  - generowanie tłumaczeń, 488
  - osadzanie informacji o pozycji, 486
  - uczenie sekwencja do sekwencji, 483
  - wizyjny, 318
  - wstępne trenowanie enkodera, 490
  - wstępnie wytrenowany, 491
  - wydajność, 497
- translacja, 65
- transpozycja macierzy, 63
- trenowanie, 28, 36, 48, 69, 102
  - całego wsadu, 102
  - dwukierunkowej warstwy LSTM, 414
  - gęsto połączonego modelu, 394
  - kodera VAE, 554
  - konwolucyjnej sieci neuronowej, 278
  - krok trenowania
    - PyTorch, 253
    - w JAX, 254
    - w TensorFlow, 253
  - małymi partiami, 102
  - miniGPT, 506

## trenowanie

## modelu

- bigramowego, 438
- dyfuzyjnego, 562, 565
- językowego, 458
- miniGPT, 513
- opartego na LSTM, 444, 449, 455
- regresyjnego, 436, 440
- rekurencyjnego, 442
- RoBERTa, 497
- segmentującego obraz, 348, 350
- sekwencyjnego, 444
- typu ciągly worek słów, 434, 454
- YOLO, 367, 377, 379

- na TPU, 591

- na wielu procesorach GPU, 584

## przyspieszanie

- precyzja mieszana, 595, 596
- precyzja połowiczna, 594
- precyzja zmiennoprzecinkowa, 593
- użycie float8, 596

- rozproszone, 587

- sekwencja do sekwencji, 465, 468, 646

- sieci konwolucyjnej, 269, 291

- średni błąd bezwzględny, 395, 397, 399, 409, 411

- techniką LoRA, 528

- w czasie testowania, 621

- w Keras, 242

- wstępne, 450, 456

- enkodera transformera, 490

- osadzeń tekstu, 450

- wykres dokładności, 144, 152, 153, 288, 292, 298, 301, 303, 318

- wykres straty, 143, 151, 288, 292, 298, 301, 303, 318, 353

- z użyciem wielu urządzeń, 583

## U

- uczenie, 29, 31, 89

- generatywne, 251

- głębokie, 25–43, 637, 638–640

- architektury, 308

- biblioteka Keras, 120

- charakterystyki, 36

- do generowania obrazu, 545

- dostrzeganie abstrakcji, 627

- działanie, 34

- działanie modelu, 176

- format danych obrazów, 269

- framework, 91, 93

- framework JAX, 112

- framework PyTorch, 105

- framework TensorFlow, 94

- generalizacja, 173

- głębokość modelu, 32

- interpretacja geometryczna, 68

- małe zbiory danych, 278

- mechanizm uwagi, 473

- modele hybrydowe, 649

- ograniczenia, 600, 647

- rozmaitości, 342

- sterowanie wyszukiwaniem

- programów, 631

- synteza programów, 629

- systemy hybrydowe, 629

- wagi, 123

- warstwy, 123

- wspomaganie wyszukiwania

- programów, 649

- zastosowania, 38

- maszynowe, 27, 637

- falszywe korelacje, 170

- generalizacja, 166

- niejednoznaczne cechy, 169

- ocena modeli, 179

- poprawa dopasowania, 184

- poprawa generalizacji, 190

- uniwersalny przepływ pracy, 204–226

- uniwersalny przepływ roboczy, 641

- zaszumione dane, 168

- zbyt słabe dopasowanie, 167

- nadzorowane, 251

- samonadzorowane, 37, 251

- sekwencja do sekwencji, 464, 483

- ze wzmocnieniem, 251

- ze wzmocnieniem RLHF, 531

- ukryta rozmaitość danych, 202

- unigramy, 439

- usuwanie szumu, 556

- uwaga neuronowa, 645

## V

- VAE, variational autoencoders, 545

## W

- wagi, 34, 69, 89, 123, 180, 637
  - kwantyzacja, 223
  - modelu Gemma, 521
  - niepodlegające uczeniu, 252
  - podlegające uczeniu, 252
  - prycinanie, 223
  - regularyzacja, 197
- walidacja
  - k-krotna iteracyjna, 181, 182, 215
  - k-krotna krzyżowa, 160, 181, 182, 215
  - k-krotna z losowaniem, 182
  - na odłożonych danych, 181
  - średni błąd bezwzględny, 395, 397, 399, 409, 411
  - wykras dokładności, 144, 152, 153, 288, 292, 298, 301, 303, 318
  - wykras straty, 143, 151, 188, 198, 288, 292, 298, 301, 303, 318, 353
- wariancja, 160
- warstwa, layer, 47, 123, 637
  - długiej pamięci krótkotrwałej, LSTM, 398, 405
  - normalizacji obrazu, 562
  - rekurencyjna w Keras, 402
- warstwy
  - gęsto połączone, 123, 643
  - konwolucyjne, 644
  - rekurencyjne, 123
    - dwukierunkowe, 407, 412
    - układane w stos, 407, 410
- wartość straty, 135
- wdrażanie modelu
  - bezpośrednio na urządzeniu, 221
  - etap wnioskowania, 219
  - jako interfejs REST API, 220
  - monitorowanie, 224
  - optymalizacja wnioskowania, 223
  - prezentacja, 219
  - utrzymywanie, 224
  - w aplikacji przeglądarkowej, 222
- wektor, 50, 639
- wektorowa baza danych, 538
- wektoryzacja danych, 213
- widzenie komputerowe, 345
- wiedza, 637
  - sieci, 123
- wizualizacja, 248
- filtrów, 329
- map ciepła, 336
- pośrednich aktywacji, 322
- przestrzeni ukrytej, 342
- wnioskowanie, 132, 220, 251, 585
  - przyspieszanie
    - kwantyzacja do formatu int8, 597
    - precyzja float16, 597
    - precyzja połowiczna, 594
  - z RetinaNet, 383
- worek słów, 434, 435, 451, 453
- wsad, batch, 135
- wskaźnik
  - dokładność binarna, 216
  - dokładność kategoriowa, 216
  - IoU, 352, 378
  - kategoriowa dokładność k-krotna, 216
  - pole pod wykresem krzywej ROC, 216
  - średni błąd bezwzględny, 216
- wskaźniki, 128, 129
  - interfejs API, 257
  - monitorowanie, 250
  - określenie kierunku, 578
  - trenowania i walidacji ekstrakcji cech, 298, 301
  - trenowania i walidacji, 288, 292, 298, 301, 303
    - modelu typu worek słów, 437
    - w stylu Xception, 318
  - w Keras, 244
  - walidacji, 187, 215
  - wbudowane, 578
  - własne, 244
- współczynnik
  - porzucania, dropout rate, 199
  - skalowania straty, 596
  - skuteczności, 618
  - uczenia, 74, 185
- wstępne
  - przetwarzanie tekstu
    - indeksowanie, 423
    - podział, 423
    - standaryzacja, 423
  - trenowanie, 450, 456
  - trenowanie enkodera transformera, 490
- wyciek
  - informacji, 180
  - tymczasowy, 184
  - wartości docelowych, 212

wydajność sieci RNN, 409

wykres

modelu z informacją o kształcie, 237

połączeń między warstwami, 236

porównania dokładności, 172

walidacji, 162

wywołania zwrotnego, 249

wykrywanie

anomalii, 386

obiektów, 346, 364–384

wymiar

próbek, 54

wsadu, 54

wymiarowość, 50

wyszukiwanie zachłanne, 544

wywołania

zwrotne, 245, 248

zwrotne własne, 247

wznoszenie gradientu

w JAX, 332

w PyTorch, 332

w TensorFlow, 331

wzorce filtrów, 335

## X

Xception, 317

## Z

zachłanne przeszukiwanie, greedy search,  
517, 519

zakrzywienie wielowymiarowej przestrzeni,  
72

zamrażanie warstwy, 299

zanik współczynnika uczenia, 565

zasada skrótu, 616

zbiory

testowe, 46, 179

treningowe, 46, 179

walidacyjne, 179, 215

zbiór danych

„Dogs vs. Cats”, 279

Agencji Reutera, 147

ARC-AGI, 619

ARC-AGI-2, 621

BookCorpus, 506

C4, 507

ceny domów, 157

COCO, 367

ImageNet, 279, 293

IMDb, 136, 432

Kaggle, 279

kursy akcji, 56

MNIST, 45, 46, 56, 168, 269

Oxford Flowers, 557

Oxford-IIIT Pets, 348

posty w serwisie, 56

# 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** 

# Naucz się uczenia głębokiego od twórcy Keras — warstwa po warstwie

Uczenie głębokie to jedna z najdynamiczniej rozwijających się dziedzin informatyki — w ciągu ostatniej dekady zrewolucjonizowało branżę IT, medycynę, transport i kulturę. Dziś, w erze ChatGPT, Midjourney i autonomicznych pojazdów, znajomość sieci neuronowych i generatywnej sztucznej inteligencji stała się kluczową kompetencją, którą powinien posiadać każdy programista i data scientist. **Deep Learning. Praca z językiem Python i biblioteką Keras** to kompleksowy, praktyczny przewodnik po uczeniu głębokim — od podstaw matematycznych po najnowsze architektury transformerów i modele dyfuzyjne. Napisany przez twórcę biblioteki Keras, François Cholleta, jest uznawany za jedno z najlepszych wprowadzeń do tej dziedziny na rynku.

Książka prowadzi czytelnika przez cały ekosystem uczenia głębokiego: od tensorów i propagacji wstecznej, przez sieci konwolucyjne do klasyfikacji obrazów i segmentacji, rekurencyjne sieci neuronowe do prognozowania szeregów czasowych, aż po zaawansowane modele językowe i generowanie obrazów. Trzecie wydanie zostało niemal całkowicie zmienione — zawiera nowe rozdziały poświęcone architekturze transformera, budowie własnego miniaturowego modelu GPT, dostrajaniu dużych modeli językowych techniką LoRA, a także generowaniu obrazów z użyciem modeli dyfuzyjnych i Stable Diffusion. Każdy rozdział uzupełniają gotowe do uruchomienia przykłady kodu w Pythonie z użyciem bibliotek: Keras, PyTorch, TensorFlow i JAX.

**François Chollet** — twórca biblioteki Keras i współzałożyciel laboratorium badawczego NdeA. Od 2012 roku pracuje nad uczeniem głębokim, przez wiele lat jako badacz w Google. Autor wyzwania ARC-AGI do pomiaru inteligencji maszyn i założyciel ARC Prize Foundation.

**Matthew Watson** — inżynier w Google, pracuje nad modelem Gemini i ekosystemem uczenia głębokiego open source. Główny opiekun biblioteki Keras, specjalizuje się w narzędziach NLP. Absolwent Uniwersytetu Stanforda.

”

Doskonała pozycja dla wszystkich, którzy lubią uczyć się przez praktykę — i chcą się uczyć od najlepszych w branży.

**Anthony Goldbloom**  
założyciel Kaggle

Inteligentny, praktyczny przewodnik, uczący, jak myśleć — od pierwszej zasady do budowania modeli, które naprawdę działają.

**Santiago Valdarrama**  
założyciel ml.school

Obecnie nie ma bardziej aktualnego i wyczerpującego przewodnika.

**Aran Komatsuzaki**  
EleutherAI

**Helion**  
helion.pl  
HELION S.A.  
ul. Kościuszki 1c  
44-100 Gliwice  
tel.: 32 230 98 63  
helion@helion.pl

KOD KORZYŚCI  
Sięgnij po więcej!



ISBN 978-83-289-3938-7



9 788328 939387

Cena: 149,00 zł

**MANNING**