DEEP LEARNING

Zadania zaliczeniowe z Kerasa

Adrian Horzyk

horzyk@agh.edu.pl

AGH University of Science and Technology
Krakow, Poland

Zadania na zaliczenie z Kerasa

Zadanie 1. (minimum na ocenę dostateczną)

Zbuduj i wytrenuj konwolucyjną sieć neuronową CNN, która dokona lepszej klasyfikacji zbioru MNIST niż wynik podany na wykładzie.

Zmodyfikuj hiperparametry sieci, poeksperymentuj (np. zastosuj różne sposoby optymalizacji, różne parametry drop out, różną ilość warstw itp.)!

Im lepszy wynik, tym lepsza ocena z tego zadania na zaliczenie.

Zadanie 2. (obowiązkowe na ocenę wyższą niż dostateczną)

Zbuduj i wytrenuj konwolucyjną sieć neuronową CNN, która nauczy się klasyfikacji zbioru CIFAR-10 lub CIFAR-100 uzyska możliwie wysoki poziom uczenia (generalizacji/performance).

Zadanie 3. (Opcjonalne, dla ambitnych lub ew. zamiennie z Zadaniem 2.)

Zbuduj i wytrenuj konwolucyjną sieć neuronową CNN, która nauczy się klasyfikacji **dowolnego wybranego zbioru uczącego (np. ze zbiorów Kaggle.com)** uzyska możliwie wysoki poziom uczenia (generalizacji/performance).

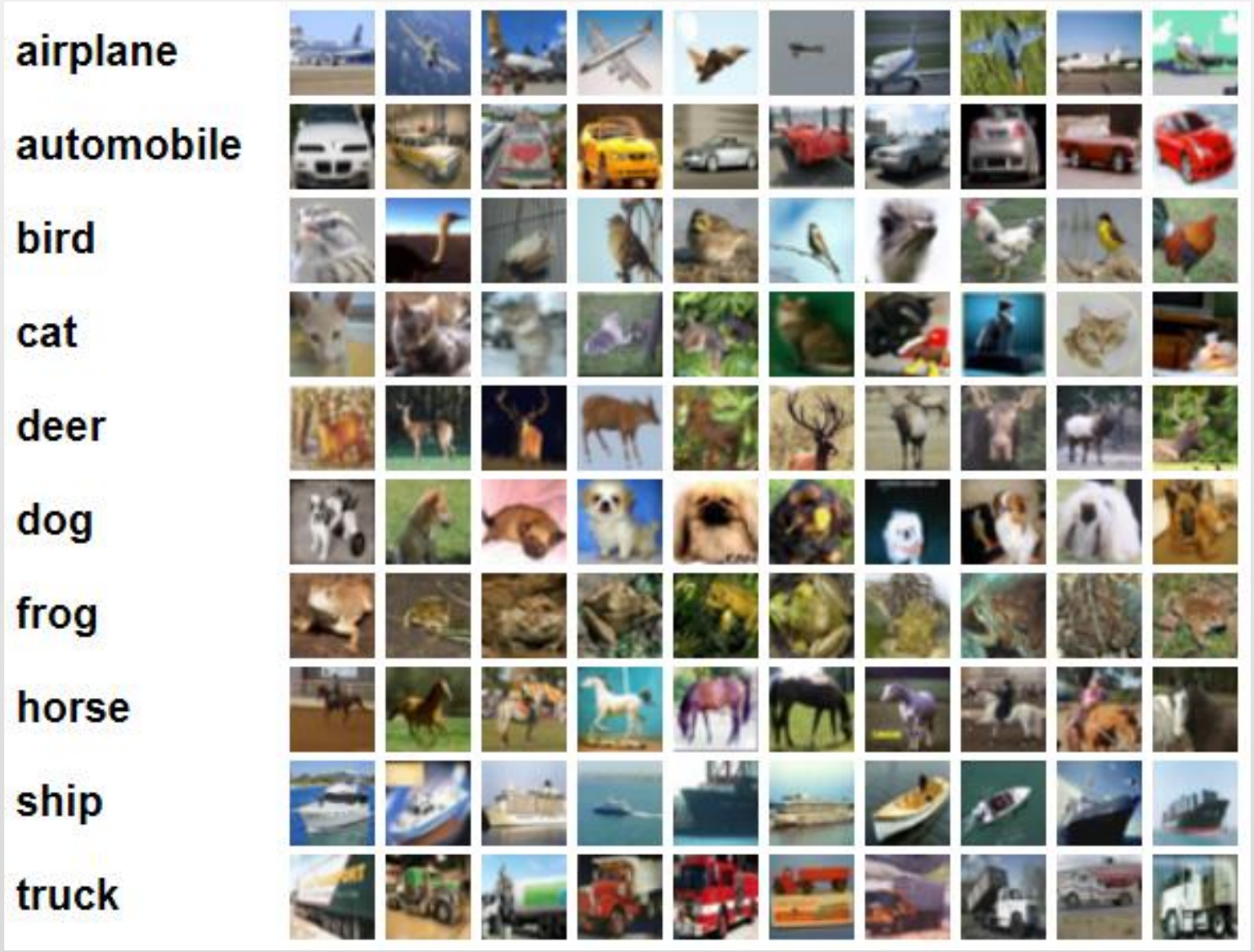
Implementacja CNN korzystająca z Kerasa

Na kolejnych slajdach podano kod Convolutional Neural Network (CNN) to dla zbioru uczącego MNIST, który należy odpowiednio zmodyfikować, rozwinąć w celu realizacji zadań 1, 2. i 3.:

Każdy obraz w zbiorze MNIST składa się z 28x28 pikseli z wycentrowaną cyfrą z zakresu 0 do 9 w skali szarości. Należy dokonać klasyfikacji z wykorzystaniem konwolucyjnej sieci neuronowej (sekwencyjnej) oraz frameworka Keras.

Implementacja CNN korzystająca z Kerasa

Zbiór uczący CIFAR-10 zawiera obrazy sklasyfikowane do 10 klas:



Implementacja CNN korzystająca z Kerasa

Zbiór uczący CIFAR-100 zawiera obrazy sklasyfikowane 100 klas:

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Kod prostej implementacji CNN dla MNIST

```
'''Trains a simple ConvNet on the MNIST dataset. It gets more than 99% test accuracy after 12 epochs
(but there is still a lot of margin for parameter tuning). Training can take a few minutes!'''

# Import libraries
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

# Define hyperparameters
batch_size = 128
num_classes = 10
epochs = 12

# Input image dimensions
img_rows, img_cols = 28, 28

# Split the data between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

Kod prostej implementacji CNN dla MNIST

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Define the sequential Keras model composed of a few layers
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu', input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model using optimizer
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])

# Train the model, validate, evaluate, and present scores
model.fit(x_train, y_train,batch_size=batch_size, epochs=epochs,
          verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Kod prostej implementacji CNN dla MNIST

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 54s 900us/step - loss: 0.2651 - accuracy: 0.9186 - val_loss: 0.0572 - val_acc
uracy: 0.9807
Epoch 2/12
60000/60000 [==============================] - 58s 961us/step - loss: 0.0884 - accuracy: 0.9734 - val_loss: 0.0397 - val_acc
uracy: 0.9867
Epoch 3/12
60000/60000 [==============================] - 61s 1ms/step - loss: 0.0669 - accuracy: 0.9799 - val_loss: 0.0342 - val_accur
acy: 0.9883
Epoch 4/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0547 - accuracy: 0.9834 - val_loss: 0.0303 - val_accur
acy: 0.9902
Epoch 5/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0453 - accuracy: 0.9862 - val_loss: 0.0283 - val_accur
acy: 0.9909
Epoch 6/12
60000/60000 [==============================] - 60s 992us/step - loss: 0.0406 - accuracy: 0.9878 - val_loss: 0.0287 - val_acc
uracy: 0.9901
Epoch 7/12
60000/60000 [==============================] - 60s 998us/step - loss: 0.0365 - accuracy: 0.9887 - val_loss: 0.0285 - val_acc
uracy: 0.9909
Epoch 8/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0346 - accuracy: 0.9897 - val_loss: 0.0278 - val_accur
acy: 0.9902
Epoch 9/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0310 - accuracy: 0.9903 - val_loss: 0.0382 - val_accur
acy: 0.9884
Epoch 10/12
60000/60000 [==============================] - 60s 995us/step - loss: 0.0308 - accuracy: 0.9906 - val_loss: 0.0277 - val_acc
uracy: 0.9913
Epoch 11/12
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0295 - accuracy: 0.9908 - val_loss: 0.0259 - val_accur
acy: 0.9919
Epoch 12/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0271 - accuracy: 0.9916 - val_loss: 0.0271 - val_accur
acy: 0.9919
Test loss: 0.027094655736458435
Test accuracy: 0.9919000267982483
```

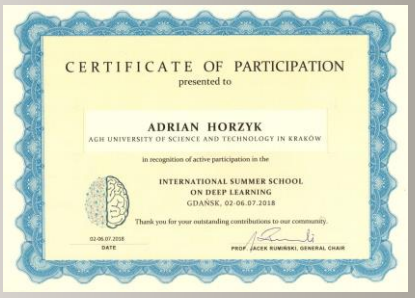
Bibliography and Literature

1. https://www.youtube.com/watch?v=XNKeayZW4dY
2. https://victorzhou.com/blog/keras-cnn-tutorial/
3. https://github.com/keras-team/keras/tree/master/examples
4. https://medium.com/@margaretmz/anaconda-jupyter-notebook-tensorflow-and-keras-b91f381405f8
5. https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html
6. http://coursera.org/specializations/tensorflow-in-practice
7. https://udacity.com/course/intro-to-tensorflow-for-deep-learning

Adrian Horzyk

horzyk@agh.edu.pl

Google: Horzyk



University of Science and Technology in Krakow, Poland