

Практична робота № 13

Тема: Класифікація зображень засобами згорткових нейронних мереж

Мета роботи: набути практичних навичок роботи в Python; зрозуміти принцип роботи згорткових нейронних мереж.

Література

Документація: <https://keras.io/models/sequential/>

Розпізнавання об'єктів в базі даних зображень CIFAR - 10 <https://samyzaf.com/ML/cifar10/cifar10.html>

Зміст роботи

Завдання 1. Розробити модель згорткової нейронної мережі (CNN, Convolutional Neural Networks) і застосувати її до класифікації зображень на основі набору даних CIFAR-10.

airplane



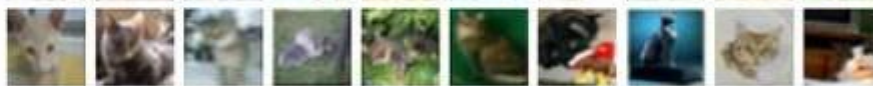
automobile



bird



cat



deer



dog



Набір даних CIFAR-10, який зазвичай використовується для дослідження машинного навчання містить 60 000 кольорових зображень 32x32 у 10 різних класах (літак, автомобіль, птах, кішка, олень, собака, жаба, кінь, корабель і вантажівка), 6 000 зображень кожного класу.

Згорткова нейронна мережа (ConvNet/CNN) - це одна з різних типів нейронних мереж глибокого навчання. CNN дуже потужний і широко використовується в класифікації зображень, розпізнаванні зображень, комп'ютерному зору тощо.

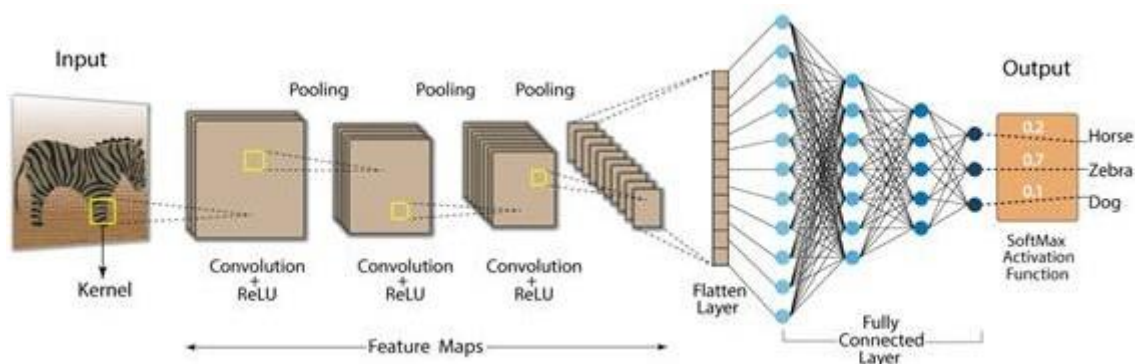
CNN складається з вхідного, вихідного та безлічі прихованих шарів між ними.

Ці шари виконують операції, які змінюють дані з метою вивчення властивостей, притаманних цим даним. Три найбільш поширені рівні - це згортка, активація або ReLU і об'єднання.

Convolution - це процес, що включає поєднання двох функцій, в результаті чого виходить інша функція. У CNN вхідне зображення піддається згортці з використанням фільтрів, які створюють карту ознак.

ReLU дозволяє проводити більш швидко та ефективно навчання, зрівнюючи негативні значення з нулем та зберігаючи позитивні значення. Іноді це називають активацією, тому що лише активовані функції переносяться на наступний рівень.

Pooling спрощує вивід, виконуючи нелінійну субдискретизацію, зменшуючи кількість параметрів, які мережа має вивчити.



Бібліотеки:

```
import numpy as np
```

Підпрограми для вибору набору даних CIFAR-10

```
from keras.datasets import cifar10
```

Базовий клас для навчання нейронної мережі

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Convolution2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.preprocessing import image
import matplotlib.pyplot as plt
```

Утиліти для одноразового кодування значень

```
from keras.utils import np_utils
якщо буде помилка використовуйте keras.utils.to_categorical
і можливо треба буде pip install np_utils
```

Зазвичай CNN використовує більше гіперпараметрів, ніж MLP. Задаємо наступні гіперпараметри:

– *batch_size* - кількість навчальних зразків, оброблюваних одночасно за одну ітерацію алгоритму градієнтного спуску;

- *num_epochs* - кількість ітерацій навчального алгоритму;
 - *kernel_size* - розмір ядра в згорткових шарах;
 - *pool_size* - розмір підвибірки в шарах підвибірки;
 - *conv_depth* - кількість ядер в згорткових шарах;
 - *drop_prob* (dropout probability) - ми будемо застосовувати dropout після кожного шару підвибірки, а також після повнозв'язного шару;
 - *hidden_size* - кількість нейронів у повнозв'язному шарі MLP
- ```

- batch_size=32
- num_epochs=5
- kernel_size=3
- pool_size=1
- conv_depth_1=32
- conv_depth_2=64
- drop_prob_1=0.25
- drop_prob_2=0.5
- hidden_size=512

```

200 епох може зайняти занадто багато часу, якщо у вашому розпорядженні немає графічного процесора. Якщо ви збираєтеся навчати мережу на CPU, варто скоротити кількість епох і / або ядер.

Завантаження та первинну обробку CIFAR-10 *Keras* виконує все автоматично.

```

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
num_train, depth, height, width = X_train.shape
num_test = X_test.shape[0]
num_classes = np.unique(y_train).shape[0]
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

```

Нормалізувати дані до [0, 1] діапазону

```

X_train/=np.max(X_train)
X_test/=np.max(X_train)
Y_train=np_utils.to_categorical(y_train,num_classes)
Y_test=np_utils.to_categorical(y_test,num_classes)

```

Мережа буде складатися з чотирьох шарів *Convolution\_2D* і шарів *MaxPooling2D* після другої і четвертої згортки. Подвоюємо кількість ядер після першого шару підвибірки, після чого вихідне зображення шару підвибірки трансформується у одновимірний вектор (шаром *Flatten*) і проходить два повнозв'язних шари (*Dense*). На всіх шарах, крім вихідного, використовується функція активації *ReLU*, останній же шар використовує *softmax*.

Для регуляризації отриманої моделі після кожного шару підвибірки і першого повнозв'язного шару застосовується шар *Dropout*. Тут Keras також виділяється на тлі інших фреймворків: в ньому є внутрішній прапор, який автоматично включає і вимикає dropout, в залежності від того, знаходиться модель в фазі навчання або тестування.

#### Налаштування:

- використовуємо перехресну ентропію як функції втрат;
- використовуємо оптимізатор Адама для градієнтного спуску;
- вимірюємо точність моделі (так як вихідні дані розподілені по класам рівномірно);
- залишаємо 10% даних для подальшої валідації.

```
inp=Input(shape=(depth,height,width))
conv_1=Convolution2D(conv_depth_1,kernel_size,kernel_size,padding='same',activation='relu')(inp)
conv_2=Convolution2D(conv_depth_1,kernel_size,kernel_size,padding='same',activation='relu')(conv_1)
pool_1=MaxPooling2D(pool_size=(pool_size,pool_size))(conv_2)
drop_1=Dropout(drop_prob_1)(pool_1)
conv_3=Convolution2D(conv_depth_2,kernel_size,kernel_size,padding='same',activation='relu')(drop_1)
conv_4=Convolution2D(conv_depth_2,kernel_size,kernel_size,padding='same',activation='relu')(conv_3)
pool_2=MaxPooling2D(pool_size=(pool_size,pool_size))(conv_4)
drop_2=Dropout(drop_prob_1)(pool_2)
flat=Flatten()(drop_2)
hidden=Dense(hidden_size,activation='relu')(flat)
drop_3=Dropout(drop_prob_2)(hidden)
out=Dense(num_classes,activation='softmax')(drop_3)

model=Model(inp,out)
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X_train,Y_train,batch_size=batch_size,nb_epoch=num_epochs,verbose=1,validation_split=0.1)
model.evaluate(X_test,Y_test,verbose=1)
```

#### Результат:

```
WARNING:tensorflow:The `nb_epoch` argument in `fit` has been renamed
`epochs`.
Train on 45000 samples, validate on 5000 samples
Epoch 1/10
45000/45000 [=====] - 16s 349us/sample - loss:
1.8304 - acc: 0.3226 - val_loss: 1.5397 - val_acc: 0.4452
Epoch 2/10
45000/45000 [=====] - 16s 351us/sample - loss:
1.5712 - acc: 0.4235 - val_loss: 1.4027 - val_acc: 0.4900
Epoch 3/10
45000/45000 [=====] - 15s 344us/sample - loss:
1.4817 - acc: 0.4580 - val_loss: 1.3176 - val_acc: 0.5222
```

```
Epoch 4/10
45000/45000 [=====] - 17s 371us/sample - loss:
1.4226 - acc: 0.4833 - val_loss: 1.3143 - val_acc: 0.5298
Epoch 5/10
```

**Завдання 2.** Проведіть експерименти з моделлю, що отримали у завданні 1: Спробуйте досягти більшої точності.

**Збереження навченої нейронної мережі.** Після навчання мережі в Colaboratory потрібно її зберегти і завантажити з хмарної платформи на свій комп'ютер. Для збереження скористайтеся стандартними засобами *Keras*:

```
model_json=model.to_json()
json_file=open("cifar10_model.json","w")
json_file.write(model_json)
json_file.close()
model.save_weights("cifar10_model.h5")
```

Архітектуру нейронної мережі і ваги зберігаємо у файли `cifar10_model.json` і `cifar10_model.h5`. Але ці файли знаходяться на віртуальній машині. Перевірити це можна командою `ls`.

```
!ls
```

Результат:

```
📁 B_1.jpg B_3.jpg cifar10_model.h5 cifar10_model.json sample_data
```

Для завантаження файлів з віртуальної машини на власний комп'ютер можна скористатися вбудованими в Colaboratory засобами роботи з файлами. Підключаємо відповідний модуль:

```
from google.colab import files
```

Завантажуємо файли командами:

```
files.download("cifar10_model.json")
files.download("cifar10_model.h5")
```

**Завдання 3.** Протестуйте роботу нейронної мережі на зображеннях.

Для цього потрібно завантажити власні зображення з локального комп'ютера на віртуальну машину *Colaboratory*. Скористаємося модулем *files*, у якого є метод *upload*. При виклику цього методу відкриється вікно вибору файлів браузера. Файли, які ви виберете в цьому вікні, будуть відправлені на віртуальну машину.

```
from google.colab import files
files.upload()
```

Результат:

```
Обзор... dog3.jpg
dog3.jpg(image/jpeg) - 7407 bytes, last modified: n/a
Saving dog3.jpg to dog3.jpg
{'dog3.jpg': b'\xff\xd8\xff\xe0\x00\x10JFIF
\x00\x00\xff\xdb\x00\x01\x00=\x03\x00\x10'
```

Зображення для перевірки роботи мережі:

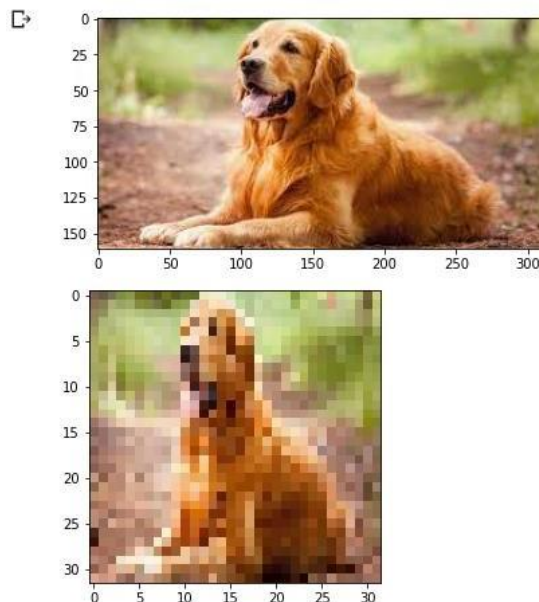


Перевіряємо загрузку файлу.

Перетворюємо картинку у масив для розпізнавання .

```
img_path='dog3.jpg'
#img=image.load_img(img_path)
img=tf.keras.utils.load_img(img_path)
plt.imshow(img)
plt.show()
#img=image.load_img(img_path,target_size=(32,32))
img=tf.keras.utils.load_img(img_path,target_size=(32,32))
plt.imshow(img)
plt.show()
```

Результат:



Запускаємо розпізнавання:

```
x=tf.keras.utils.img_to_array(img)

x/=255
x=np.expand_dims(x,axis=0)
```



```
prediction=model.predict(x)
classes=['літак', 'автомобіль', 'птах', 'кіт', 'олень', 'собака', 'жаба', 'кінь',
'корабель', 'вантажівка']
print(classes[np.argmax(prediction)])
```

Результат:

---

👉 собака

**Завдання 2.3.** Протестуйте роботу нейронної мережі на 10 різних зображеннях (зображення краще вибирати не зовсім однозначні, щоб отримати різні результати). Проаналізуйте отримані результати. Дайте оцінку роботи нейронній мережі.

***Звіт оформити у вигляді скріншотів та пояснень + відповіді на контрольні запитання.***

### **Контрольні запитання**

1. Назвіть сфери застосування нейронних мереж.
2. Які шари використовуються в CNN?
3. Назвіть та охарактеризуйте етапи побудови нейромережевої моделі.
4. Яким чином перевіряється правильність роботи навченої мережі?
5. Назвіть недоліки алгоритму зворотного розповсюдження.
6. Охарактеризуйте набір даних CIFAR-10.