AI TECH - Akademia Innowacyjnych Zastosowań Technologii Cyfrowych. Programu Operacyjnego Polska Cyfrowa na lata 2014-2020

SKRYPT DO LABORATORIUM Uczenie głebokie LABORATORIUM 1: Uczenie sieci splotowych Natalia Głowacka

Twórcy tego cudeńka

Jakub Sachajko, 179976

Grzegorz pozorski, 180169

1. Opis ćwiczenia

Wymagania wstępne:

Wymagania w odniesieniu do studenta:

Właściwe przygotowanie się studenta do zajęć pozwoli na osiągnięcie celów ćwiczenia. Przed przystąpieniem do ćwiczenia student powinien:

- powtórzyć wiedzę nabytą w czasie wykładów,
- zapoznać się z instrukcją do ćwiczenia ilustrującą podstawowe zagadnienia z zakresu tematyki ćwiczenia.

Dodatkowo wymagana jest od studenta:

- podstawowa umiejętność programowania,
- podstawowa znajomość języka Python.

Wymagania w odniesieniu do stanowiska laboratoryjnego:

Stanowisko laboratoryjne powinno być wyposażone w komputer z dostępem do sieci komputerowej oraz z następującymi zasobami:

- przykładowe kody programów (załącznik),
- Python 3,
- pakiety NumPy, Matplotlib, TensorFlow, Keras, Seaborn,
- Jupyter Notebook,
- inne: przeglądarka WWW.

Cele ćwiczenia:

Celem ćwiczenia jest praktyczne przedstawienie wiedzy zdobytej podczas wykładów. Podczas realizacji ćwiczenia laboratoryjnego studenci zdobędą wiedzę i umiejętności w zakresie podstawowych zagadnień dotyczących sieci splotowych, operacji splotu, wykorzystania podstawowych warstw sieci splotowych oraz implementacji i trenowania prostej sieci splotowej.

Spodziewane efekty kształcenia - umiejętności i kompetencje:

Po zakończeniu ćwiczenia laboratoryjnego student będzie posiadał umiejętność w zakresie obliczania wyniku operacji splotu, a także obliczania rozmiaru modelu wyjściowego po zastosowaniu kilku warstw charakterystycznych dla sieci splotowych. Zdobędzie także podstawową wiedzę w zakresie implementacji i trenowania prostej sieci splotowej.

Metody dydaktyczne:

Na początku student realizuje zadania przykładowe, które prezentują poszczególne elementy obliczania wyniku operacji splotu dla przykładowych danych. W kolejnym kroku realizuje zadania według wytycznych, na podstawie zdobytych umiejętności i wiedzy. Kody źródłowe i wyniki wykonywanych programów student powinien umieszczać w dokumencie elektronicznym, który stanie się sprawozdaniem z ćwiczenia laboratoryjnego.

Materiały wprowadzające i pomocnicze:

- Numpy - dokumentacja biblioteki https://numpy.org/doc/,
- Matplotlib - dokumentacja biblioteki https://matplotlib.org/stable/contents.html,
- Tensorflow – dokumentacja biblioteki https://www.tensorflow.org/api_docs,
- Keras - dokumentacja biblioteki https://keras.io/api/,
- Seaborn - dokumentacja biblioteki https://seaborn.pydata.org,
- Python 3 - dokumentacja języka https://docs.python.org/3/.

Zasady oceniania/warunek zaliczenia ćwiczenia:

Każde z realizowanych podczas ćwiczenia laboratoryjnego zadań będzie podlegało ocenie. W trakcie realizacji ćwiczenia, mogą zostać przydzielone dodatkowe zadania do realizacji. Maksymalna liczba punktów do zdobycia wynosi 10.

Wykaz literatury podstawowej do ćwiczenia:

1. Treści wykładowe do przedmiotu "Uczenie głębokie"
2. Bengio Yoshua, Courville Aaron, Goodfellow Ian, Deep Learning, Systemy uczące się, PWN 2018
3. Andrew W. Trask, Zrozumieć głębokie uczenie, PWN, 2019

2. Przebieg ćwiczenia

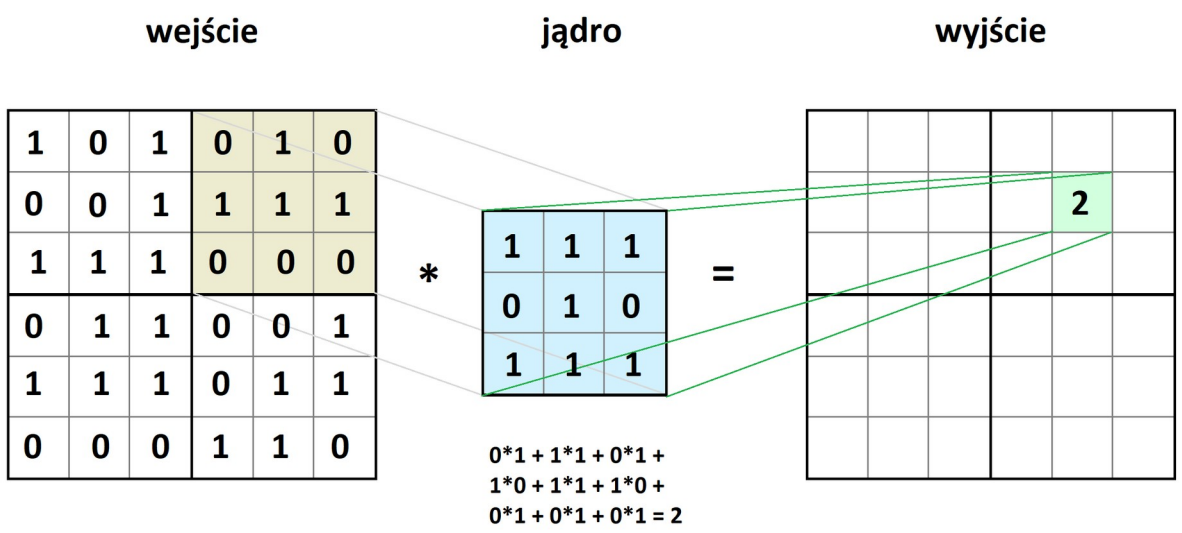
1. Zapoznanie się z instrukcją laboratoryjną (przed ćwiczeniem)
2. Zadanie 1 - Obliczenie splotu ręcznie (30 min.)
3. Zadanie 2 - Wykorzystanie splotu różnych jąder na obrazach (25 min.)
4. Zadanie 3 - Obliczanie rozmiaru wyjściowego modelu (30 min.)
5. Zadanie 4 - Implementacja i trenowanie prostej sieci splotowej (40 min.)
6. Przesłanie sprawozdania z zajęć (10 min.)

3. Wprowadzenie do ćwiczenia

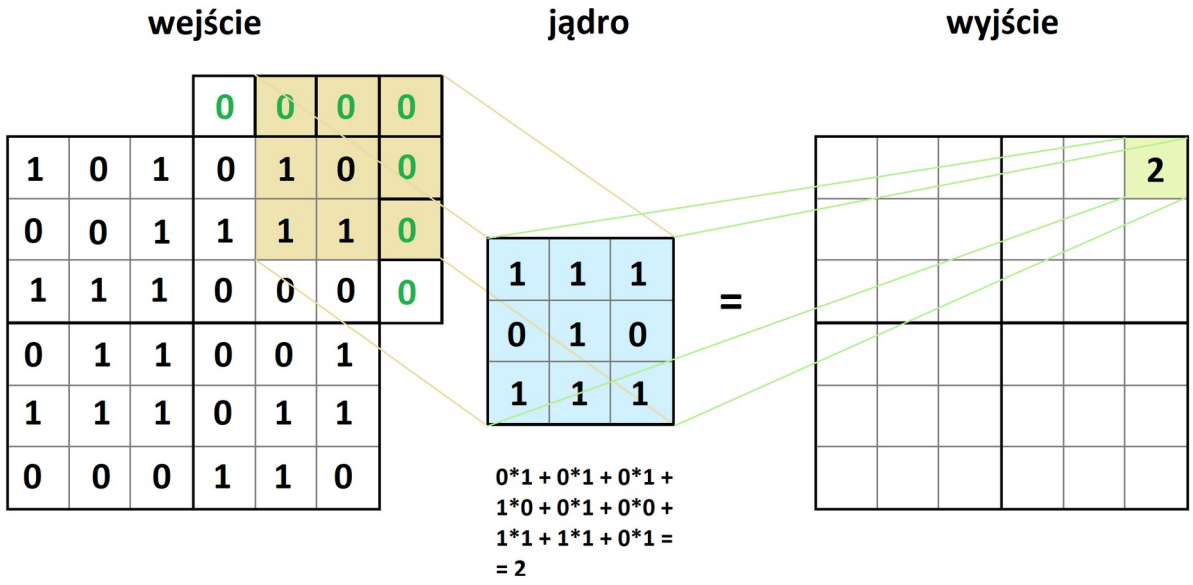
Zadanie 1. Obliczenie splotu ręcznie

Podstawą splotu jest wykorzystanie faktu, iż każde z wejść jest dodawane do swoich lokalnych sąsiadów (w ramach lokalnego pola recepcyjnego) z pewną wagą wyrażoną poprzez jądro.

Na rysunku poniżej przedstawiony został przykład obliczenia splotu dla przykładowych wartości wejścia oraz jądra.



Splot możemy obliczyć wtedy, kiedy środek jądra może zostać nałożony na pole w całości. Dla przypadku przedstawionego na powyższym rysunku (obraz 6x6), jądro o wymiarach 3x3 możemy nałożyć na 16 pól, dlatego wyjściowa mapa cech miałaby wymiar 4x4. Jeśli chcemy uzyskać mapę cech o takim samym wymiarze jak wejście, należy wykorzystać *technikę dopełniania*. W przypadku obliczenia wartości splotu na krawędziach, często dodaje się zera wokół danych wejściowych (*zero padding*). Na rysunku poniżej przedstawiony został przykład zastosowania uzupełniania zerami do obliczenia splotu dla wartości położonej na krawędzi obrazu wejściowego.



1a. Wykorzystując zdefiniowane poniżej jądra oraz obraz wejściowy, oblicz wartości wyjściowe dla każdej z wartości wejściowych (wyjście jest takiego samego rozmiaru jak wejście - zastosuj uzupełnianie zerami).

```
#wyświetlanie wszystkich wyników z komórki
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
kernels = {
    "k1":   np.array([
                            [2., 0., -2.],
                            [2., 0., -2.],
                            [2., 0., -2.],
                            ]),
    "k2":  np.array([
                            [0., 1., 0.],
                            [1., 3., 1.],
                            [0., 1., 0.],
                            ]),
    "k3":  np.array([
                            [-1., -1., -1.],
                            [ 0.,  0.,  0.],
                            [ 1.,  1.,  1.],
                            ])
}


image = np.array([
                  [1., 0., 3.],
                  [6., 2., 7.],
                  [0., 8., 4.]
])

def manual_convolve_2d(image, kernel):
    output = np.zeros_like(image)

    x_shape, y_shape = image.shape

    image_padded = np.zeros((x_shape + 2, y_shape + 2))
    image_padded[1:-1, 1:-1] = image

    for y in range(y_shape):
        for x in range(x_shape):
            output[x, y]=(kernel*image_padded[x:x+3, y:y+3]).sum()
    return output

results = {name: manual_convolve_2d(image, kernel) for name, kernel in
kernels.items()}

for k, v in results.items():
    print(f"Convolution result with {k}:")
    print(v)
```

```
Convolution result with k1:
[[ -4.  -6.   4.]
 [-20. -14.  20.]
 [-20. -10.  20.]]
Convolution result with k2:
[[ 9.  6. 16.]
 [21. 27. 30.]
 [14. 30. 27.]]
Convolution result with k3:
[[  8.  15.   9.]
 [  7.   8.   9.]
 [ -8. -15.  -9.]]
```

##**Zadanie 2.** Wykorzystanie splotu różnych jąder na obrazach

Warstwy splotowe same uczą się wyodrębniania istotnych cech z danych - lokalnych wzorców. Wzorce te są rozpoznawane przez sieć niezależnie od ich położenia. Wagi poszczególnych neuronów reprezentowane są przez jądra splotowe (filtry), których rozmiar jest taki jak rozmiar pola recepcyjnego. Wybór wartości danego jądra determinuje to, jakie wzorce będą wykrywane.

2a. Wykorzystując funkcję *apply_conv2d*, która pozwala na obliczenie splotu jądra z obrazem wejściowym, przeprowadź eksperymenty wykorzystując zdefiniowane wcześniej jądra oraz obraz *lab1_1.jpg*. Sprawdź i wyświetl jakie cechy podkreślają dla obrazu w skali szarości poszczególne filtry - wyświetl obraz przed i po zastosowaniu splotu. Dodatkowo, stwórz jeden własny filtr 5x5 i również wyświetl efekt splotu dla tego jądra.

```
import numpy as np
import tensorflow as tf

def apply_conv2d(input, kernel, strides=(1,1), padding='SAME'):

  #zmieniamy kształt danych na (batch, no_of_rows, no_of_columns,
no_of_channels)
  input = input.astype(np.float64)
  input = input[np.newaxis, ...]

  #wymiary jądra
  k_width, k_height = kernel.shape
  #liczba wyjść
  no_of_filters = 1

  input_depth = 1

  #kształt jądra powinnien być: (kernel_width, kernel_height,
input_depth, no_of_filters)
  k_shape=(k_width, k_height, input_depth, no_of_filters)

  kernel_c = kernel.reshape(k_shape)

  #tworzymy warstwę splotową 2D
```

```
  layer_2D_1 = tf.keras.layers.Conv2D(no_of_filters,
                                      kernel_size=(k_width, k_height),
                                      strides = strides,
                                      padding=padding,
                                      activation='linear',
                                      use_bias=False,
                                      weights=[kernel_c])

  return layer_2D_1(input).numpy()

import keras
# funkcja pozwalająca na wczytanie obrazu i zmianę jego rozmiaru oraz
trybu koloru
def read_image(filepath, h, w, color_mode='grayscale'):
  img = keras.preprocessing.image.load_img(filepath, target_size=(h,
w), color_mode=color_mode)
  img_array = keras.preprocessing.image.img_to_array(img)

  # TF > 2.9.1
  # img = keras.utils.load_img(filepath, target_size=(h, w),
color_mode=color_mode)
  # img_array = keras.utils.img_to_array(img)

  return img_array

kernels = {
    "k1":   np.array([
                            [2., 0., -2.],
                            [2., 0., -2.],
                            [2., 0., -2.],
                            ]),
    "k2":  np.array([
                            [0., 1., 0.],
                            [1., 3., 1.],
                            [0., 1., 0.],
                            ]),
    "k3":  np.array([
                            [-1., -1., -1.],
                            [ 0.,  0.,  0.],
                            [ 1.,  1.,  1.],
                            ]),
    "none":  np.array([
                            [1., 1., 1.],
                            [ 1.,  1.,  1.],
                            [ 1.,  1.,  1.],
                            ]),
    "custom_filter": np.array([
                            [2., 2, 2., 2, 2.],
                            [2., 1, 1., 1, 2.],
                            [2., 1, 0., 1, 2.],
```

```
                              [2., 1, 1., 1, 2.],
                              [2., 2, 2., 2, 2.],
                              ])
}

from matplotlib import pyplot as plt

img = read_image('lab1_1.jpg', 160, 120)

fig = plt.figure(figsize=(10, 7))
rows = 1
cols = 5

cnt = 1
for name, kernel in kernels.items():
    convolution = apply_conv2d(img, kernel)
    fig.add_subplot(rows, cols, cnt)
    plt.imshow(convolution[0][:][:], cmap="gray")
    cnt += 1
```

```
<Axes: >
```

```
<matplotlib.image.AxesImage at 0x7b3799088910>
```

```
<Axes: >
```

```
<matplotlib.image.AxesImage at 0x7b37985c5cf0>
```

```
<Axes: >
```

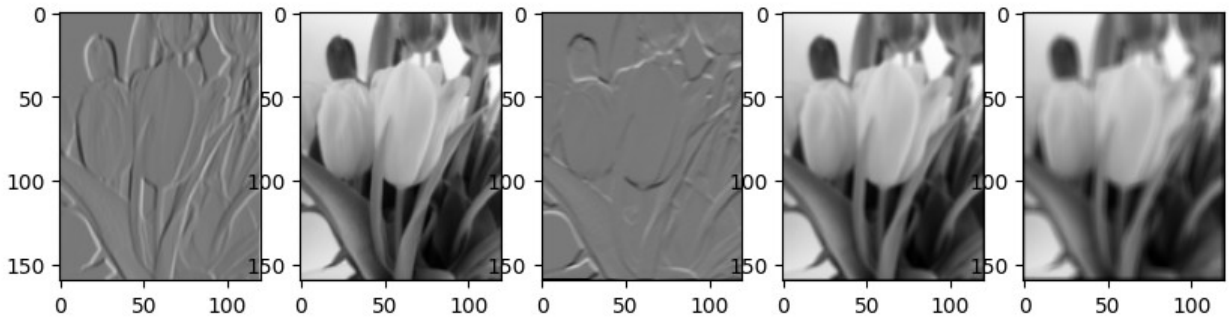
```
<matplotlib.image.AxesImage at 0x7b379841e9e0>
```

```
<Axes: >
```

```
<matplotlib.image.AxesImage at 0x7b37984a6110>
```

```
<Axes: >
```

```
<matplotlib.image.AxesImage at 0x7b37984479a0>
```



2b. Funkcja *apply_con2d* pozwala również na zdefiniowanie kroku oraz tego, czy wykorzystujemy dopełnienie zerami dla wartości położonych na krawędziach. Sprawdź, jak zmieniają się wymiary wyjścia po zastosowaniu splotu obrazu wejściowego z jądrem, stosując (lub nie) dopełnianie zerami oraz zmianę wartości kroku. Zapisz swoje wnioski.

```
kernel = next(iter(kernels.values()))
convolution = apply_conv2d(img, kernel)
convolution.shape
```

```
(1, 160, 120, 1)
```

```
kernel = next(iter(kernels.values()))
convolution = apply_conv2d(img, kernel, padding='valid')
shape = convolution.shape
print(shape)
```

```
(1, 158, 118, 1)
```

```
kernel = next(iter(kernels.values()))
convolution = apply_conv2d(img, kernel, strides=(2,2))
shape = convolution.shape
print(shape)
```

```
(1, 80, 60, 1)
```

```
kernel = next(iter(kernels.values()))
convolution = apply_conv2d(img, kernel, strides=(2,1))
shape = convolution.shape
print(shape)
```

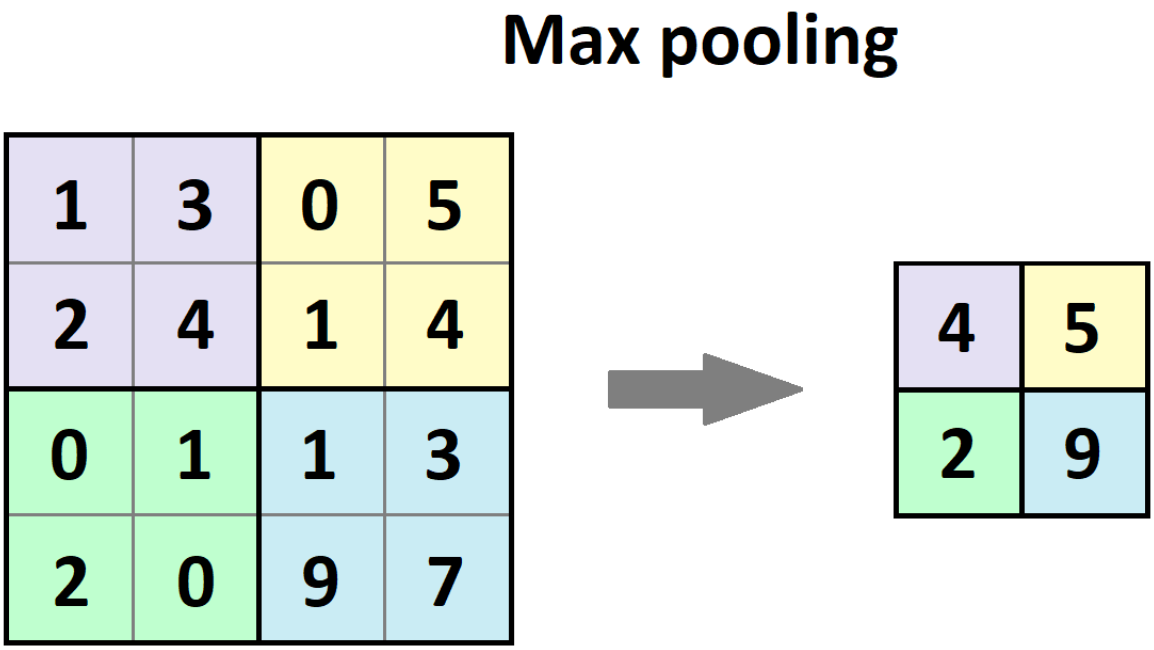
```
(1, 80, 120, 1)
```

Ustawienie padding="VALID" nie dodaje paddingu z zerami, przez co wynikowy obraz jest zmniejszony o 2 pionowo i poziomo. Ustawienie kroku na x zmniejsza dwukrotnie wymiar obrazu x razy

Zadanie 3. Obliczanie rozmiaru wyjściowego modelu

Warstwy *pooling* są kolejny elementem budującym sieć splotową. Zmniejszają wymiar przestrzenny, co pomaga ograniczyć ryzyko przetrenowania (zmniejsza się liczba parametrów), działają na każdym z kanałów (każdej mapie cech), więc głębokość wyjściowa jest taka sama (zmienia się tylko wymiar przestrzenny). Warstwa ta nie ma wag.

Na poniższym rysunku przedstawiona została operacja *max pooling* z krokiem 2 oraz wymiarem jądra 2, czyli wybór maksymalnej wartości z pola recepcyjnego - pozostałe wartości pozostają odrzucone. W takim podejściu odrzucamy wiele wartości, ale pozostawiamy tylko ekstremalne, najbardziej znaczące cechy.



Stosując warstwę *average pooling* uzyskamy taki sam efekt redukcji wymiarów, jednak zamiast wartości maksymalnej z pola recepcyjnego, obliczmy wartość średnią w tym polu. Przy tym podejściu możemy uzyskać zbyt uogólnione wyjście.

3a. Analizując poniżej przedstawiony przykład sekwencji operacji na podanym wejściu, zapisz jaki jest kształt wyjściowy dla każdego z poszczególnych kroków. Konkretny przykład zostanie podany na laboratorium przez prowadzącego ćwiczenie.

input = np.ones((24, 24, 1))

1. Splot z 16 filtrami o rozmiarze 3x3, krok (1,1), padding SAME
2. Pooling - jądro 5x5, krok (3,3), padding VALID
3. Splot z 32 filtrami o rozmiarze 3x3, krok (1,1), padding SAME
4. Pooling - jądro 3x3, krok (2,2), padding VALID

```
input = np.ones((24, 24, 1))
keras_model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, kernel_size=3, activation='relu',
input_shape=input.shape, padding='SAME'),
    tf.keras.layers.MaxPooling2D(pool_size=(5,5), strides=(3, 3),
padding="VALID"),
    tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu',
padding='SAME'),
    tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2),
padding="VALID"),
  ])

keras_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 24, 24, 16)        160

 max_pooling2d_8 (MaxPoolin  (None, 7, 7, 16)          0
 g2D)

 conv2d_11 (Conv2D)          (None, 7, 7, 32)          4640

 max_pooling2d_9 (MaxPoolin  (None, 3, 3, 32)          0
 g2D)

=================================================================
Total params: 4800 (18.75 KB)
Trainable params: 4800 (18.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Zadanie 4. Implementacja i trenowanie prostej sieci splotowej

Początkowym etapem jest przygotowanie zbioru danych. W tym zadaniu wykorzystany zostanie zbiór danych *CIFAR10*.

4a. Wczytaj dane ze zbioru *CIFAR10* i podziel je na dane treningowe i testowe. Przygotuj listę zawierającą etykiety poszczególnych klas. Wyświetl przykładowy obraz należący do jednej z klas.

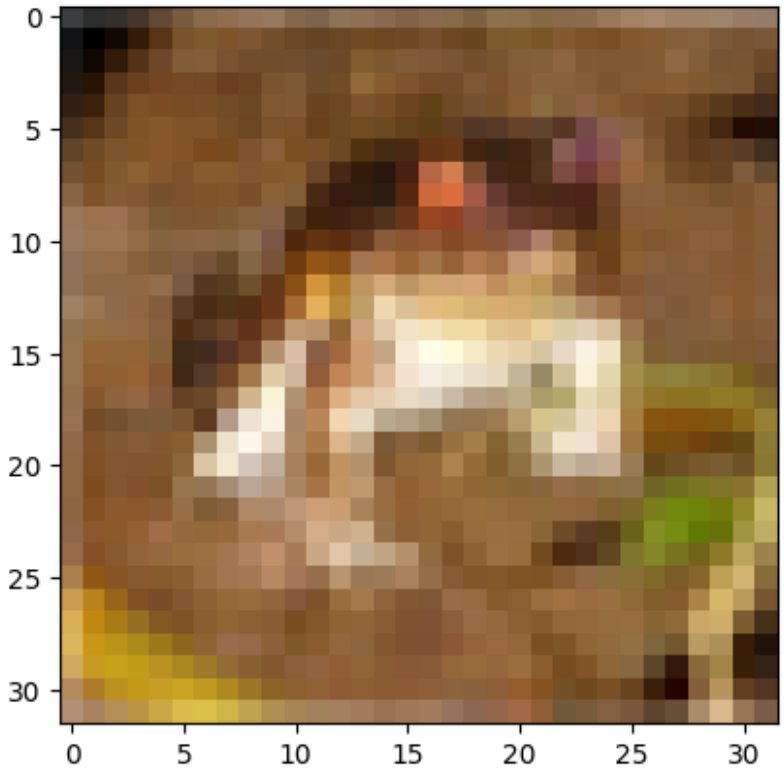
Dokumentacja - zbiór CIFAR10

```
(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.cifar10.load_data()
print(x_train.shape)
print(y_train.shape)
```

```
(50000, 32, 32, 3)
(50000, 1)
```

```
from matplotlib import pyplot as plt
plt.imshow(x_train[0][:][:])
```

```
<matplotlib.image.AxesImage at 0x7d28f7ff3190>
```



Kolejnym etapem jest sprawdzenie i przygotowanie danych do treningu sieci.

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# normalizacja danych 0-1
x_train = x_train / 255.0
x_test = x_test / 255.0

print("Kształt danych treningowych: ", x_train.shape)
```

```
Kształt danych treningowych:  (50000, 32, 32, 3)
```

W kolejnym kroku zdefiniujmy podstawowe parametry treningu.

```
# wybór parametrów uczenia
batch_size = 64
epochs = 15
no_of_classes = 10
learning_rate = 0.01
```

Po przygotowaniu danych, wykorzystując API Keras, zdefiniujmy prosty model oraz wyświetlmy architekturę stworzonej sieci.

```
# definicja modelu
keras_model = tf.keras.Sequential([

    tf.keras.layers.Conv2D(2, kernel_size=4, activation='relu',
                           input_shape=x_train.shape[1:],
padding='SAME'),
    tf.keras.layers.Flatten(),  # spłaszczenie danych do wykorzystania
warstwy gęstej
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(no_of_classes, activation='softmax')

  ])

# wyświetlenie architektury
keras_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 32, 32, 2)         98

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 50)                102450

 dense_1 (Dense)             (None, 10)                510

=================================================================
Total params: 103058 (402.57 KB)
Trainable params: 103058 (402.57 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Dla utworzonego powyżej modelu, kształt wyjściowy z pierwszej warstwy konwolucyjnej jest taki sam jak kształt wejściowy (zastosowane zostało dopełnienie zerami), a głębokość wyjściowa jest równa liczbie ustawionych filtrów - 2.

W kolejnym etapie ustawmy parametry treningu i rozpocznijmy uczenie modelu na danych treningowych.

```
# Ustawiamy parametry treningu

keras_model.compile(
    # wybór optymalizatora
    optimizer='adam',
    loss='categorical_crossentropy',
```

```
    # miara, którą chcemy monitorować
    metrics=['accuracy']
)

# Rozpoczynamy trening z zdefiniowanymi wcześniej hiperparametrami
history = keras_model.fit(x_train, y_train, epochs=epochs,
batch_size=batch_size)

Epoch 1/15
782/782 [==============================] - 6s 4ms/step - loss: 1.9228
- accuracy: 0.3143
Epoch 2/15
782/782 [==============================] - 3s 4ms/step - loss: 1.7836
- accuracy: 0.3733
Epoch 3/15
782/782 [==============================] - 3s 4ms/step - loss: 1.7427
- accuracy: 0.3877
Epoch 4/15
782/782 [==============================] - 3s 4ms/step - loss: 1.7000
- accuracy: 0.4006
Epoch 5/15
782/782 [==============================] - 3s 4ms/step - loss: 1.6700
- accuracy: 0.4100
Epoch 6/15
782/782 [==============================] - 3s 4ms/step - loss: 1.6500
- accuracy: 0.4166
Epoch 7/15
782/782 [==============================] - 3s 4ms/step - loss: 1.6303
- accuracy: 0.4225
Epoch 8/15
782/782 [==============================] - 3s 4ms/step - loss: 1.6195
- accuracy: 0.4250
Epoch 9/15
782/782 [==============================] - 3s 4ms/step - loss: 1.6103
- accuracy: 0.4279
Epoch 10/15
782/782 [==============================] - 4s 5ms/step - loss: 1.6009
- accuracy: 0.4333
Epoch 11/15
782/782 [==============================] - 6s 8ms/step - loss: 1.5908
- accuracy: 0.4348
Epoch 12/15
782/782 [==============================] - 4s 5ms/step - loss: 1.5871
- accuracy: 0.4366
Epoch 13/15
782/782 [==============================] - 3s 4ms/step - loss: 1.5758
- accuracy: 0.4421
Epoch 14/15
782/782 [==============================] - 3s 4ms/step - loss: 1.5718
- accuracy: 0.4425
```

```
Epoch 15/15
782/782 [==============================] - 3s 4ms/step - loss: 1.5665
- accuracy: 0.4427
```

Uzyskany wynik dokładności nie jest zadowalający, jednak stworzony model jest bardzo prosty i trenowany był przez 15 epok. Wyświetlmy wykres z historii, aby zobaczyć jak zmieniała się dokładność modelu.

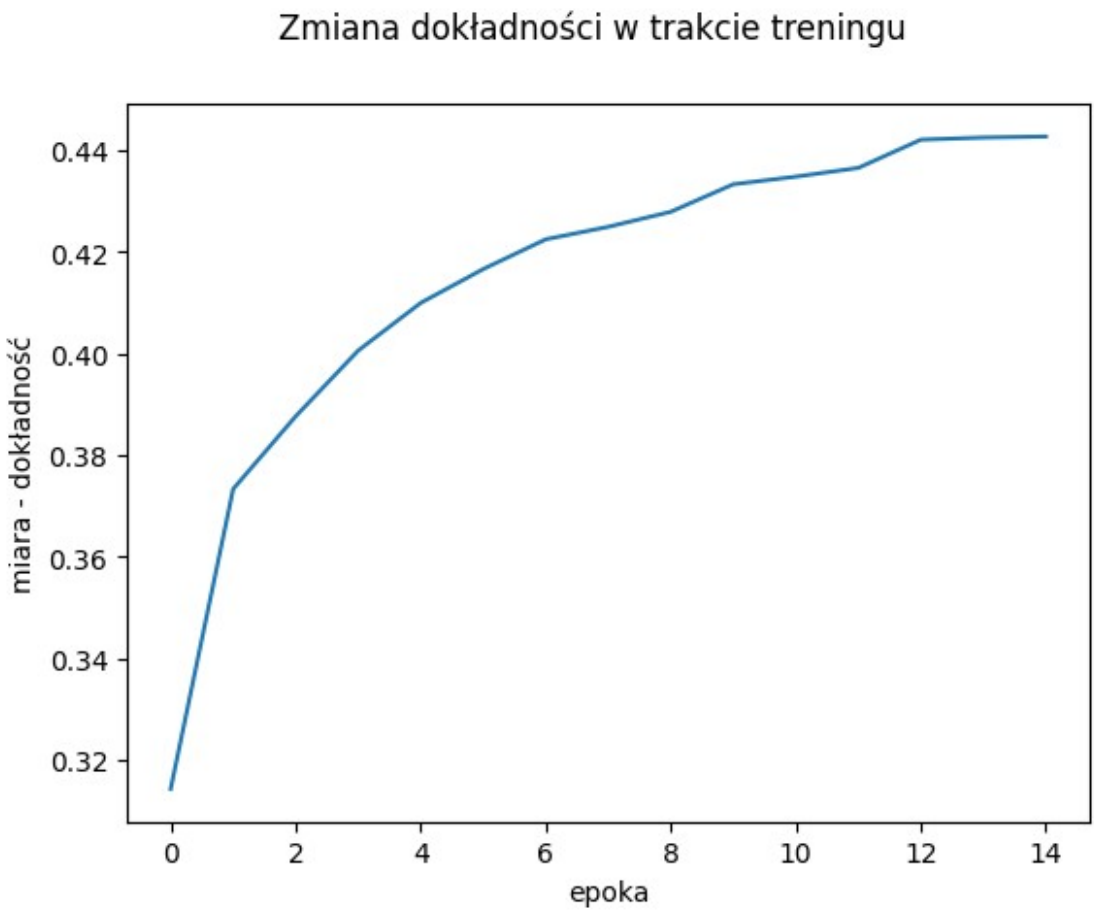
```
figure = plt.figure()
figure.suptitle("Zmiana dokładności w trakcie treningu")
figure_acc = figure.add_subplot(1,1,1)
figure_acc.set_xlabel('epoka')
figure_acc.set_ylabel('miara - dokładność')
plt.plot(history.history['accuracy'])

plt.show()
```

```
Text(0.5, 0.98, 'Zmiana dokładności w trakcie treningu')
```

```
Text(0.5, 0, 'epoka')
```

```
Text(0, 0.5, 'miara - dokładność')
```

```
[<matplotlib.lines.Line2D at 0x7d28906e8790>]
```



Sprawdźmy, jakie wartości metryk ewaluacyjnych osiąga wytrenowany model dla danych testowych.

```
# Ewaluacja modelu na danych testowych
loss, accuracy = keras_model.evaluate(x_test, y_test)

print("Wynik ewaluacji - loss: ", loss)
print("Wynik ewaluacji - dokładność (accuracy): ", accuracy)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.5925
- accuracy: 0.4378
Wynik ewaluacji - loss:  1.5925005674362183
Wynik ewaluacji - dokładność (accuracy):  0.43779999017715454
```

4b. Stwórz nowy model, poprzez rozszerzenie modelu *keras_model* o dodatkową warstwę konwolucyjną (liczba filtrów: 32, kernel_size=3, dopełnianie zerami) oraz dwie warstwy max pooling (2,2). W pierwszej warstwie splotowej zwiększ liczbę filtrów do 8. Przeprowadź trening modelu (z parametrami takim jak dla pierwszej architektury) i porównaj uzyskane wyniki z prostym modelem, zawierającym tylko jedną warstwę splotową - zamieść komentarz.

```python
# definicja modelu
keras_model_2 = tf.keras.Sequential([

    tf.keras.layers.Conv2D(8, kernel_size=4, activation='relu',
                           input_shape=x_train.shape[1:],
padding='SAME'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu',
                           input_shape=x_train.shape[1:],
padding='SAME'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),  # spłaszczenie danych do wykorzystania
warstwy gęstej
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(no_of_classes, activation='softmax')
  ])

# wyświetlenie architektury
keras_model_2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 8)         392

 max_pooling2d_4 (MaxPoolin  (None, 16, 16, 8)         0
 g2D)

 conv2d_7 (Conv2D)           (None, 16, 16, 32)        2336

 max_pooling2d_5 (MaxPoolin  (None, 8, 8, 32)          0
 g2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 50)                102450

 dense_3 (Dense)             (None, 10)                510

=================================================================
Total params: 105688 (412.84 KB)
Trainable params: 105688 (412.84 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

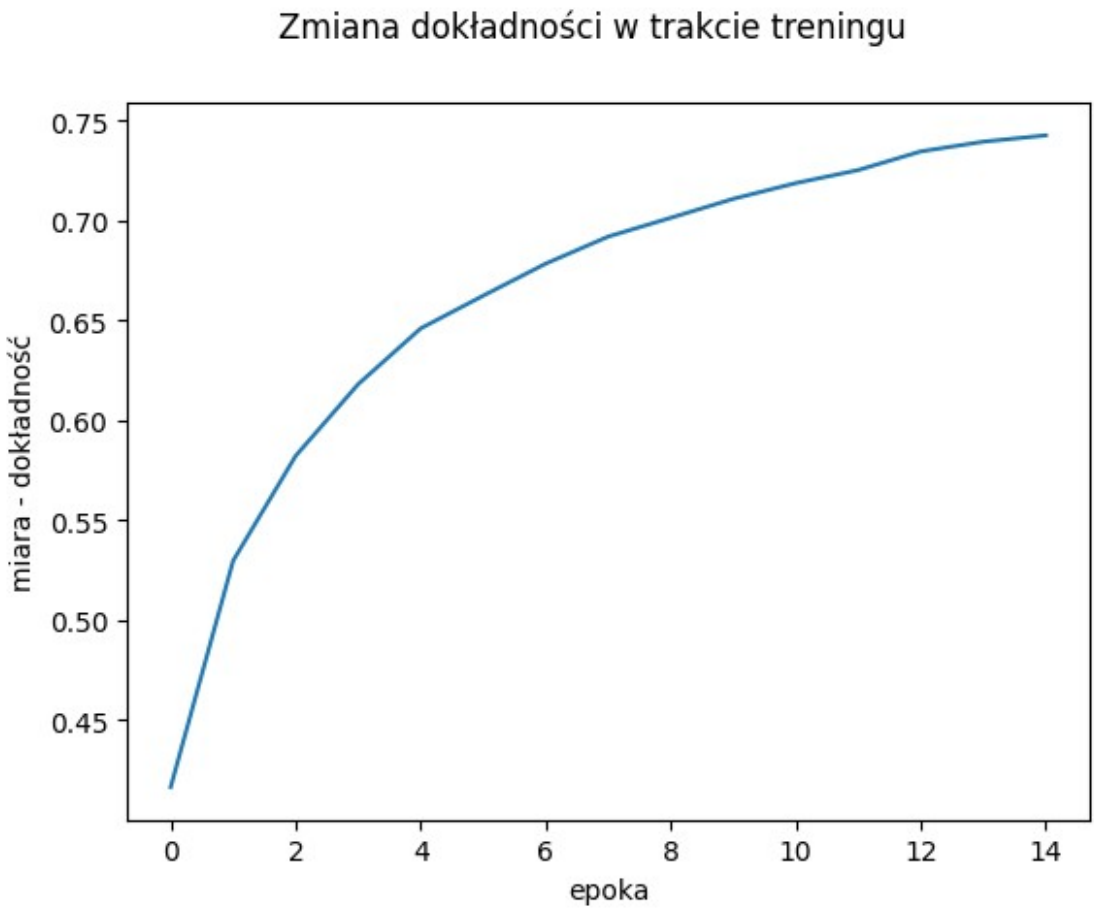
```python
# Ustawiamy parametry treningu

keras_model_2.compile(
    # wybór optymalizatora
```

```
    optimizer='adam',
    loss='categorical_crossentropy',
    # miara, którą chcemy monitorować
    metrics=['accuracy']
)

# Rozpoczynamy trening z zdefiniowanymi wcześniej hiperparametrami
history = keras_model_2.fit(x_train, y_train, epochs=epochs,
batch_size=batch_size)

Epoch 1/15
782/782 [==============================] - 5s 4ms/step - loss: 1.6286
- accuracy: 0.4165
Epoch 2/15
782/782 [==============================] - 4s 5ms/step - loss: 1.3181
- accuracy: 0.5298
Epoch 3/15
782/782 [==============================] - 3s 4ms/step - loss: 1.1864
- accuracy: 0.5822
Epoch 4/15
782/782 [==============================] - 3s 4ms/step - loss: 1.0923
- accuracy: 0.6180
Epoch 5/15
782/782 [==============================] - 3s 4ms/step - loss: 1.0234
- accuracy: 0.6459
Epoch 6/15
782/782 [==============================] - 4s 5ms/step - loss: 0.9686
- accuracy: 0.6623
Epoch 7/15
782/782 [==============================] - 3s 4ms/step - loss: 0.9250
- accuracy: 0.6783
Epoch 8/15
782/782 [==============================] - 3s 4ms/step - loss: 0.8896
- accuracy: 0.6919
Epoch 9/15
782/782 [==============================] - 4s 5ms/step - loss: 0.8599
- accuracy: 0.7012
Epoch 10/15
782/782 [==============================] - 3s 4ms/step - loss: 0.8352
- accuracy: 0.7107
Epoch 11/15
782/782 [==============================] - 3s 4ms/step - loss: 0.8114
- accuracy: 0.7186
Epoch 12/15
782/782 [==============================] - 3s 4ms/step - loss: 0.7900
- accuracy: 0.7251
Epoch 13/15
782/782 [==============================] - 4s 5ms/step - loss: 0.7686
- accuracy: 0.7345
Epoch 14/15
```

```
782/782 [==============================] - 3s 4ms/step - loss: 0.7523
- accuracy: 0.7393
Epoch 15/15
782/782 [==============================] - 3s 4ms/step - loss: 0.7375
- accuracy: 0.7425
```

```
figure = plt.figure()
figure.suptitle("Zmiana dokładności w trakcie treningu")
figure_acc = figure.add_subplot(1,1,1)
figure_acc.set_xlabel('epoka')
figure_acc.set_ylabel('miara - dokładność')
plt.plot(history.history['accuracy'])

plt.show()
```

```
Text(0.5, 0.98, 'Zmiana dokładności w trakcie treningu')

Text(0.5, 0, 'epoka')

Text(0, 0.5, 'miara - dokładność')

[<matplotlib.lines.Line2D at 0x7d289032c910>]
```



```
# Ewaluacja modelu na danych testowych
loss, accuracy = keras_model_2.evaluate(x_test, y_test)

print("Wynik ewaluacji - loss: ", loss)
print("Wynik ewaluacji - dokładność (accuracy): ", accuracy)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.0006
- accuracy: 0.6644
Wynik ewaluacji - loss:  1.0005995035171509
Wynik ewaluacji - dokładność (accuracy):  0.6643999814987183
```

Mając wytrenowany model możemy sprawdzić, jakich wzorców zdołał się nauczyć, szczególnie dla warstw splotowych.

Przyglądając się architekturze modelu, spróbujmy wyświetlić wagi oraz dane wyjściowe dla pierwszej warstwy konwolucyjnej modelu drugiego. Stwórzmy model, który po podaniu na jego wejście obrazu, zwróci wartości aktywacji dla modelu oryginalnego. Model będzie przyjmował jeden obiekt wejściowy i generował jeden obiekt wyjściowy dla wybranej warstwy.

```
# należy uzyskać instnację klasy tf.keras.layers.Layer - wyjście
konkretnej warstwy oraz jej wagi
# dostęp można uzyskać poprzez odwołanie się do jej nazwy lub indeksu

layer_output = keras_model_2.get_layer("conv2d_6").output # należy
sprawdzić nazwę modelu
layer_weights = keras_model_2.get_layer("conv2d_6").get_weights() #
należy sprawdzić nazwę modelu

# stwórzmy instancję modelu na podstawie tensora wejściowego oraz
wyjściowego
activation_model = tf.keras.Model(inputs=keras_model_2.input,
outputs=layer_output)
```

Sprawdźmy, jak będzie wyglądało wyjście dla przykładowego obrazu ze zbioru testowego.

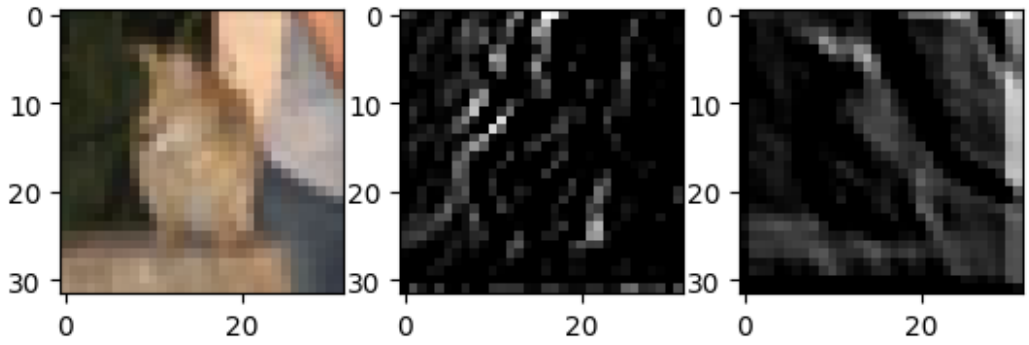
```
# wczytanie i przygotowanie obrazu
test_img = x_test[46]
test_img = test_img[np.newaxis, ...] # (batch, img_height, img_width,
no_of_channels)

activations = activation_model.predict(test_img)

# wyświetlmy kształt wyjścia
print(activations.shape)
```

```
1/1 [==============================] - 0s 154ms/step
(1, 32, 32, 8)
```

Na wyjściu otrzymujemy mapę cech o wymiarach 32x32, składającą się z 8 kanałów. Wyświetlmy w formie graficznej 4 i 8 kanał aktywacji tej warstwy oraz obraz wejściowy.

```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(1, 3)
axs[0].imshow(test_img[0])
axs[0].grid(False)
axs[1].imshow(activations[0, :, :, 3],cmap=plt.cm.gray)
axs[1].grid(False)
axs[2].imshow(activations[0, :, :, 7],cmap=plt.cm.gray)
axs[2].grid(False)
plt.show()
```

```
<matplotlib.image.AxesImage at 0x7d287b833e20>

<matplotlib.image.AxesImage at 0x7d287b495960>

<matplotlib.image.AxesImage at 0x7d287b495d80>
```



4c. Wyświetl wagi dla pierwszej warstwy splotowej, dla każdego z kanałów (komponentów) filtra 5 z wykorzystaniem map ciepła.

Dokumentacja - wyświetlanie danych w postaci mapy ciepła

```
import seaborn as sns

layer_weights = keras_model_2.get_layer("conv2d_6").get_weights()

weights = np.array(layer_weights[0])
weights = weights[:,:,:,4]

sns.heatmap(weights[:, :, 0])
plt.show()
sns.heatmap(weights[:, :, 1])
plt.show()
sns.heatmap(weights[:, :, 2])
plt.show()

# sns.heatmap(layer_weights)
```

4d. Zmieniając parametry w warstwach splotowych oraz pooling (w modelu utworzonym w punkcie 4b) lub dodając dodatkowe warstwy splotowe/pooling, spróbuj utworzyć model, który uzyska większą dokładność na danych testowych.

```
# definicja modelu
keras_model_3 = tf.keras.Sequential([

    tf.keras.layers.Conv2D(32, kernel_size=4, activation='relu',
                           input_shape=x_train.shape[1:],
padding='SAME'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, kernel_size=3, activation='relu',
                           input_shape=x_train.shape[1:],
padding='SAME'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
        tf.keras.layers.Conv2D(128, kernel_size=3, activation='relu',
                           input_shape=x_train.shape[1:],
padding='SAME'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),  # spłaszczenie danych do wykorzystania
warstwy gęstej
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(no_of_classes, activation='softmax')
  ])

# wyświetlenie architektury
keras_model_3.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 32, 32, 32)        1568

 max_pooling2d_14 (MaxPooli  (None, 16, 16, 32)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 16, 16, 64)        18496

 max_pooling2d_15 (MaxPooli  (None, 8, 8, 64)          0
 ng2D)

 conv2d_18 (Conv2D)          (None, 8, 8, 128)         73856

 max_pooling2d_16 (MaxPooli  (None, 4, 4, 128)         0
 ng2D)

 flatten_4 (Flatten)         (None, 2048)              0

 dense_8 (Dense)             (None, 50)                102450
```

```
 dense_9 (Dense)             (None, 10)                510

=================================================================
Total params: 196880 (769.06 KB)
Trainable params: 196880 (769.06 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

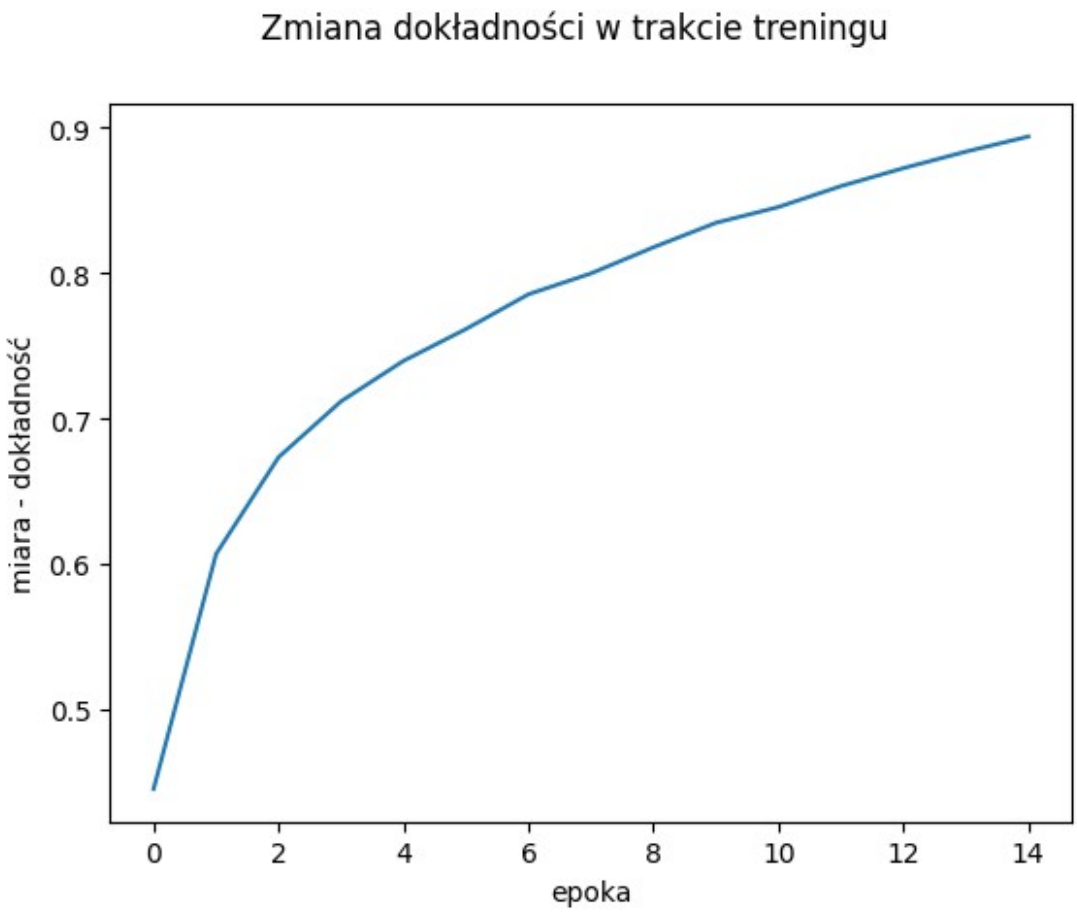
```
# Ustawiamy parametry treningu

keras_model_3.compile(
    # wybór optymalizatora
    optimizer='adam',
    loss='categorical_crossentropy',
    # miara, którą chcemy monitorować
    metrics=['accuracy']
)

# Rozpoczynamy trening z zdefiniowanymi wcześniej hiperparametrami
history = keras_model_3.fit(x_train, y_train, epochs=epochs,
batch_size=batch_size)
```

```
Epoch 1/15
782/782 [==============================] - 6s 6ms/step - loss: 1.5268
- accuracy: 0.4454
Epoch 2/15
782/782 [==============================] - 4s 5ms/step - loss: 1.1100
- accuracy: 0.6069
Epoch 3/15
782/782 [==============================] - 4s 5ms/step - loss: 0.9348
- accuracy: 0.6734
Epoch 4/15
782/782 [==============================] - 4s 5ms/step - loss: 0.8262
- accuracy: 0.7118
Epoch 5/15
782/782 [==============================] - 4s 5ms/step - loss: 0.7489
- accuracy: 0.7396
Epoch 6/15
782/782 [==============================] - 4s 5ms/step - loss: 0.6825
- accuracy: 0.7615
Epoch 7/15
782/782 [==============================] - 4s 5ms/step - loss: 0.6190
- accuracy: 0.7854
Epoch 8/15
782/782 [==============================] - 4s 5ms/step - loss: 0.5698
- accuracy: 0.7997
Epoch 9/15
782/782 [==============================] - 4s 5ms/step - loss: 0.5177
- accuracy: 0.8177
```

```
Epoch 10/15
782/782 [==============================] - 4s 6ms/step - loss: 0.4687
- accuracy: 0.8345
Epoch 11/15
782/782 [==============================] - 4s 5ms/step - loss: 0.4333
- accuracy: 0.8453
Epoch 12/15
782/782 [==============================] - 4s 5ms/step - loss: 0.3932
- accuracy: 0.8597
Epoch 13/15
782/782 [==============================] - 4s 6ms/step - loss: 0.3602
- accuracy: 0.8720
Epoch 14/15
782/782 [==============================] - 4s 5ms/step - loss: 0.3248
- accuracy: 0.8834
Epoch 15/15
782/782 [==============================] - 4s 5ms/step - loss: 0.2977
- accuracy: 0.8938
```

```
figure = plt.figure()
figure.suptitle("Zmiana dokładności w trakcie treningu")
figure_acc = figure.add_subplot(1,1,1)
figure_acc.set_xlabel('epoka')
figure_acc.set_ylabel('miara - dokładność')
plt.plot(history.history['accuracy'])

plt.show()
```

```
Text(0.5, 0.98, 'Zmiana dokładności w trakcie treningu')

Text(0.5, 0, 'epoka')

Text(0, 0.5, 'miara - dokładność')

[<matplotlib.lines.Line2D at 0x7d287ae64b80>]
```




```
# Ewaluacja modelu na danych testowych
loss, accuracy = keras_model_3.evaluate(x_test, y_test)

print("Wynik ewaluacji - loss: ", loss)
print("Wynik ewaluacji - dokładność (accuracy): ", accuracy)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.0584
- accuracy: 0.7279
Wynik ewaluacji - loss:  1.0583518743515015
Wynik ewaluacji - dokładność (accuracy):  0.7279000282287598
```

4. Forma i zawartość sprawozdania

Sprawozdanie powinno zawierać kopie ekranu stworzonych kodów i wyników ich działania dla zadań, które w instrukcji zostały oznaczone kolorem zielonym oraz stosowne komentarze, jeśli zadanie tego wymaga. Dokument powinien zostać przesłany na serwer wskazany przez prowadzącego ćwiczenie w formacie PDF.

Dodatki

Załącznik 1: Instrukcja wraz z wykonywalnymi kodami programów w formie pliku Jupyter Notebook

Załącznik 2: Obraz **lab1_1.jpg**

Fundusze Europejskie Polska Cyfrowa
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Rozwoju Regionalnego