Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №4**

з дисципліни «Програмні засоби проектування та реалізаціїї нейромережевих систем»

Тема: " Згорткові нейронні мережі"

Виконав: Перевірив:

студент групи ІП-93 Шимкович Володимир

Домінський Валентин Миколайович

Олексійович

Київ 2022

Зміст:

[Мета: 3](#_Toc103086148)

[Вихідний код 3](#_Toc103086149)

[Результат роботи: 5](#_Toc103086150)

[Висновки: 5](#_Toc103086151)

## Мета:

Написати програму що реалізує згорткову нейронну мережу AlexNet для розпізнавання об’єктів з датасету CIFAR-10

## Вихідний код

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

from os import path

model\_name = 'Lab4.h5'

model\_weights = 'Lab4\_weights.h5'

normalize\_type = 'float32'

normalize\_divider = 256

images\_to\_take\_num = 3500

training\_text = "Training -"

test\_text = "Testing -"

validation\_text = "Validation -"

batch\_size = 128

loss\_type = 'sparse\_categorical\_crossentropy'

# A type of activation function that transforms the

# value results of a neuron. The transformation imposed

# by ReLU on values from a neuron is represented by

# the formula y=max(0,x). The ReLU activation function

# clamps down any negative values from the neuron to 0,

# and positive values remain unchanged. The result of this

# mathematical transformation is utilized as the output

# of the current layer and used as input to

# a consecutive layer within a neural network

relu\_activation = 'relu'

padding\_type = "same"

# A type of activation function that is utilized to

# derive the probability distribution of a set of numbers

# within an input vector. The output of a softmax

# activation function is a vector in which its set of

# values represents the probability of an occurrence of

# a class or event. The values within the vector all add up to 1

softmax\_activation = 'softmax'

input\_image\_size = 128

learning\_rate = 0.001

accuracy\_metrics = 'accuracy'

epochs = 10

validation\_freq = 1

airplane\_image\_type = 'Airplane'

automobile\_image\_type = 'Automobile'

bird\_image\_type = 'Bird'

cat\_image\_type = 'Cat'

deer\_image\_type = 'Deer'

dog\_image\_type = 'Dog'

frog\_image\_type = 'Frog'

horse\_image\_type = 'Horse'

ship\_image\_type = 'Ship'

truck\_image\_type = 'Truck'

image\_types = [airplane\_image\_type, automobile\_image\_type, bird\_image\_type, cat\_image\_type, deer\_image\_type, dog\_image\_type, frog\_image\_type, horse\_image\_type, ship\_image\_type, truck\_image\_type]

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.cifar10.load\_data()

# Normalizes images: `uint8` -> `float32`

# TFDS provide images of type tf.uint8, while the model expects

# tf.float32. Therefore, you need to normalize images

x\_train = x\_train.astype(normalize\_type) / normalize\_divider

x\_test = x\_test.astype(normalize\_type) / normalize\_divider

x\_validation, y\_validation = x\_train[:images\_to\_take\_num], y\_train[:images\_to\_take\_num]

x\_train, y\_train = x\_train[images\_to\_take\_num:], y\_train[images\_to\_take\_num:]

print(y\_train[0])

print(y\_validation[0])

xy\_train\_origin = tf.data.Dataset.from\_tensor\_slices((x\_train, y\_train))

xy\_test\_origin = tf.data.Dataset.from\_tensor\_slices((x\_test, y\_test))

xy\_validation\_origin = tf.data.Dataset.from\_tensor\_slices((x\_validation, y\_validation))

plt.figure(figsize=(15,15))

for i, (img, name) in enumerate(xy\_train\_origin.take(9)):

ax = plt.subplot(3,3,i+1)

plt.imshow(img)

plt.title(image\_types[name.numpy()[0]])

def get\_image\_and\_name(img, name):

# Preparing the data (normzlization and scaling)

img = tf.img.per\_image\_standardization(img)

img = tf.image.resize(img, (input\_image\_size, input\_image\_size))

return img, name

# Get the size of sets

xy\_train\_size = tf.data.experimental.cardinality(xy\_train\_origin).numpy()

print(training\_text, xy\_train\_size)

xy\_test\_size = tf.data.experimental.cardinality(xy\_test\_origin).numpy()

print(test\_text, xy\_test\_size)

xy\_validation\_size = tf.data.experimental.cardinality(xy\_validation\_origin).numpy()

print(validation\_text, xy\_validation\_size)

def preprocess\_dataset(origin\_dataset):

new\_dataset = (origin\_dataset

.map(get\_image\_and\_name)

.shuffle(buffer\_size = xy\_train\_size)

.batch(batch\_size = batch\_size, drop\_remainder = True))

return new\_dataset

xy\_train = preprocess\_dataset(xy\_train\_origin)

xy\_test = preprocess\_dataset(xy\_test\_origin)

xy\_validation = preprocess\_dataset(xy\_validation\_origin)

model = keras.models.Sequential([

# A convolution is a mathematical term that describes a dot

# product multiplication between two sets of elements.

# Within deep learning the convolution operation acts

# on the filters/kernels and image data array within

# the convolutional layer. Therefore a convolutional layer

# is simply a layer the houses the convolution operation

# that occurs between the filters and the images

# passed through a convolutional neural network

keras.layers.Conv2D(filters=96, kernel\_size=(11,11), strides=(4,4), activation = relu\_activation, input\_shape=(input\_image\_size, input\_image\_size,3)),

# Batch Normalization is a technique that mitigates

# the effect of unstable gradients within a neural

# network through the introduction of an additional

# layer that performs operations on the inputs from

# the previous layer. The operations standardize and

# normalize the input values, after that the input

# values are transformed through scaling and shifting operations

keras.layers.BatchNormalization(),

# Max pooling is a variant of sub-sampling where the maximum

# pixel value of pixels that fall within the receptive

# field of a unit within a sub-sampling layer is

# taken as the output. The max-pooling operation

# below has a window of 2x2 and slides across the

# input data, outputting an average of the

# pixels within the receptive field of the kernel.

keras.layers.MaxPool2D(pool\_size=(3,3), strides=(2,2)),

keras.layers.Conv2D(filters=256, kernel\_size=(5,5), strides=(1,1), activation = relu\_activation, padding = padding\_type),

keras.layers.BatchNormalization(),

keras.layers.MaxPool2D(pool\_size=(3,3), strides=(2,2)),

keras.layers.Conv2D(filters=384, kernel\_size=(3,3), strides=(1,1), activation = relu\_activation, padding = padding\_type),

keras.layers.BatchNormalization(),

keras.layers.MaxPool2D(pool\_size=(3,3), strides=(2,2)),

# Takes an input shape and flattens the input image data into a one-dimensional array

keras.layers.Flatten(),

# A dense layer has an embedded number of arbitrary

# units/neurons within. Each neuron is a perceptron

keras.layers.Dense(4096, activation = relu\_activation),

# Dropout technique works by randomly reducing the

# number of interconnecting neurons within

# a neural network. At every training step, each

# neuron has a chance of being left out, or

# rather, dropped out of the collated

# contributions from connected neurons

keras.layers.Dropout(0.5),

keras.layers.Dense(4096, activation = relu\_activation),

keras.layers.Dropout(0.5),

keras.layers.Dense(10, activation = softmax\_activation)

])

model.compile(loss = loss\_type, optimizer=tf.optimizers.Adam(learning\_rate = learning\_rate), metrics=[accuracy\_metrics])

model.summary()

if (path.exists(model\_name)):

model = tf.keras.models.load\_model(model\_name)

else:

model.fit(xy\_train,

epochs = epochs,

validation\_data = xy\_validation,

validation\_freq = validation\_freq)

model.save(model\_name)

model.save\_weights(model\_weights)

model.evaluate(xy\_test)

## Результат роботи:



## Висновки:

Я дізнався більше інформації про Convolutional neural network (CNN) мережі – використовують згортковий шар, який шукає патерни