НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи №4

з дисципліни «Системи глибинного навчання»

на тему

“Розпізнавання двовимірних кольорових об’єктів за допомогою згорткової

нейронної мережі”

|  |  |
| --- | --- |
| Виконав: | Викладач: |
| студент групи КМ-01 | Терейковський І. А. |
| Шолоп Л. О. |  |

Київ — 2023

Зміст

[Теоретичні відомості 3](#_Toc153373592)

[Основна частина 4](#_Toc153373593)

[Додаток А – Код програми 6](#_Toc153373594)

## Теоретичні відомості

Згорткові нейронні мережі (Convolutional Neural Networks або CNN) є потужним інструментом у сфері обробки зображень та визначення образів. Вони виникли спеціально для роботи з великою кількістю даних, такими як зображення, і здатні ефективно впоратися з просторовою структурою таких даних. Основні концепції та елементи згорткових мереж включають:

1. ***Згорткові шари (Convolutional Layers):***

* Основний будівельний блок CNN. Використовується для виявлення локальних шаблонів у вхідних даних.
* Згортка - операція, яка застосовує фільтр (ядро) до малих фрагментів вхідного зображення для виділення особливостей.

1. ***Пулінгові шари (Pooling Layers):***

* Використовуються для зменшення просторових розмірів представлення та зменшення обчислювальної складності.
* Зазвичай використовуються максимальне або середнє пулінги.

1. ***Повні з'єднувальні шари (Fully Connected Layers):***

* Один або кілька повністю з'єднаних шарів в кінці мережі.
* Використовуються для зведення просторової інформації в одновимірний вектор перед подачею на вихід.

1. ***Функції активації:***

* Розташовані після кожного згорткового та повністю з'єднувального шару.
* Зазвичай використовують ReLU (Rectified Linear Unit) або інші нелинійні функції для введення нелинійності в модель.

1. ***Фільтри та ядра:***

* Фільтри або ядра використовуються для виявлення різних властивостей в зображеннях, таких як границі, форми, текстури тощо.

1. ***Стеки згорткових шарів:***

* Згорткові мережі зазвичай мають декілька шарів, які утворюють стек для більш складної і абстрактної репрезентації.

1. ***Зануреність (Invariance):***

* Згорткові шари дозволяють нейронній мережі виявляти ознаки, які є інваріантними відносно зсувів та зміни масштабу, що робить їх ефективними для визначення образів.

1. ***Регуляризація:***

* Для запобігання перенавчанню використовують техніки регуляризації, такі як Dropout, які випадковим чином вимикають деякі нейрони під час тренування.

# Основна частина

Запускаємо таймер, щоб знати скільки часу витратить програма на навчання моделі та її тестування.

Будемо імпортувати такі бібліотеки:

* Numpy – для математичних розрахунків
* Pandas – для табличного відображення виводу
* Matplotlib – для демонстрації картинок у датасеті
* TensorFlow – для створення мережі CNN
* Keras – для створення шарів моделі
* Sklearn – для оцінки моделі

Завантажуємо датасет. Нормалізуємо дані в ньому шляхом поділу на 255. Перекодовуємо мітки в категорійні змінні. Розбиваємо тестову вибірку на 2 рівні частини. Одна частина лишиться тестовим набором (тест моделі в самому кінці), інша частина валідаційним набором (тест моделі після кожної епохи). Ініціалізуємо розмір навчальної вибірки аналогічно до прикладу в умові. Створюємо список із назвами класів зображень

Відображаємо картинки, для розуміння з чим будемо працювати



Ініціалізуємо структуру мережі аналогічно до прикладу в умові. Додаємо шари Dropout для запобіганню перенавчання моделі. В якості оптимізатора будемо використовувати SGD (градієнтний спуск) із функцією втрат – категорійною кроссентропією, в якості метрики на кожній епосі будемо відображати точність.

Якщо модель навчена, то імпортуємо її з файлу, якщо ні, то навчаємо її протягом 25 епох та зберігаємо готову модель у файл.

Тепер коли модель завершила навчання знайдемо метрики для розуміння наскільки добре модель навчилась:

Метрики моделі на тестових даних:

Метрика Значення

Accuracy 0.691400

Precision 0.700315

Recall 0.691400

F1-score 0.686122

Отже, точність моделі приблизно рівна 68-70 %

Для реального використання це замала точність. Покращити результати можна шляхом збільшення епох навчання, підбору кращого оптимізатора, фільтрації зображень, які є важко розпізнаваними.

У результаті програма навчила та протестувала модель за 24 хв, 55 сек

# Додаток А – Код програми

import time

start\_timer = time.time()

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import precision\_score, recall\_score, f1\_score

from keras import layers, models

from keras.optimizers import SGD

from keras.datasets import cifar10

from keras.utils import to\_categorical

# Завантажимо дані

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

# Нормалізуємо дані

X\_train = X\_train / 255.0

X\_test  = X\_test / 255.0

# Створимо мітки класів

y\_train = to\_categorical(y\_train, 10)

y\_test  = to\_categorical(y\_test,  10)

# Створимо валідаційну вибірку для тестування моделі на кожній епосі

X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=10)

# Ініціалізуємо розмір навчальної вибірки на кожній епосі

batch\_size = 32

# Інііалізуємо назви класів датасету cifar 10

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

print(f'Записів у тренувальній вибірці: {len(X\_train)}')

print(f'Записів у тестовій вибірці: {len(X\_test)}')

print(f'Записів у валідаційній вибірці: {len(X\_val)}')

selected\_images = []

selected\_labels = []

for class\_index in range(10):

    index = np.where(np.argmax(y\_train, axis=1) == class\_index)[0][42]

    selected\_images.append(X\_train[index])

    selected\_labels.append(y\_train[index])

plt.figure(figsize=(15, 5))

for i in range(10):

    plt.subplot(2, 5, i + 1)

    plt.imshow(selected\_images[i])

    plt.title(class\_names[np.argmax(selected\_labels[i])])

    plt.axis('off')

plt.show()

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Додавання повнозв'язних шарів

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dropout(0.5))

model.add(layers.Dense(10, activation='softmax'))  # 10 класів виводу

# Компіляція моделі

sgd = SGD(learning\_rate=0.01, momentum=0.9, nesterov=True)

model.compile(optimizer=sgd,

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

try:

    model = models.load\_model('my\_model.keras')

except:

    history = model.fit(X\_train, y\_train,

                        epochs=25,

                        batch\_size=batch\_size,

                        validation\_data=(X\_val, y\_val))

    model.save('my\_model.keras') # Зберігаємо модель

test\_results = model.evaluate(X\_test, y\_test, verbose=0)

test\_accuracy = test\_results[1]

predictions = np.argmax(model.predict(X\_test, verbose=0), axis=1)

test\_precision = precision\_score(np.argmax(y\_test, axis=1), predictions, average='weighted')

test\_recall = recall\_score(np.argmax(y\_test, axis=1), predictions, average='weighted')

test\_f1\_score = f1\_score(np.argmax(y\_test, axis=1), predictions, average='weighted')

metrics = pd.DataFrame({

    'Метрика': ['Accuracy', 'Precision', 'Recall', 'F1-score'],

    'Значення': [test\_accuracy, test\_precision, test\_recall, test\_f1\_score]

})

print("Метрики моделі на тестових даних:")

print(metrics.to\_string(index=False))

elapsed\_time = time.time() - start\_timer

minutes, seconds = divmod(elapsed\_time, 60)

if minutes > 0:

    print(f'Час роботи: {int(minutes)} хв, {int(seconds)} сек')

else:

    print(f'Час роботи: {int(seconds)} сек')