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:
студент групи КМ-01
Іваник Ю. П.

Викладач:
Професор кафедри ПМА
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості

Згорткові нейронні мережі (ЗНМ) є ключовим елементом глибокого навчання, використовуючи їх для обробки вхідних даних з великою кількістю параметрів, таких як зображення. Основна ідея полягає в тому, щоб автоматично вивчати корисні фільтри або характеристики з вхідних даних, що дозволяє ЗНМ знаходити високорівневі абстракції та шаблони у вхідних сигналах.

Ключовим елементом ЗНМ є згортка, яка представляє собою локальний взяття з вхідного сигналу, що дозволяє нейронам виявляти локальні характеристики, такі як риси на зображенні. Згортки використовуються для створення карти ознак, яка виокремлює важливі деталі та структури з вхідного сигналу.

Після згорток застосовуються шари підвищення важливості (пулінг), щоб зменшити просторові розміри отриманої карти ознак та зберегти найважливіші характеристики. Це допомагає зменшити обчислювальні витрати та вчить модель фокусуватися на ключових аспектах даних.

В результаті кілька шарів згорток та пулінгу можуть вивчати високорівневі абстракції, що робить ЗНМ дуже ефективними для завдань обробки зображень, таких як класифікація об'єктів чи виявлення облич.

Згорткові нейронні мережі використовуються не лише для обробки зображень, а також для різних сфер, таких як обробка природної мови та робота з часовими рядами, де вони можуть автоматично вивчати ієрархічні рівні ознаки.

Основна частина

Для початку імпортуємо потрібні бібліотеки такі як: Numpy, Tensorflow, Keras, Sklearn

Завантажимо дані та розподілимо їх на навчальні, тестові та валідаційні набори

```
Кількість записів у навчальному наборі:   50000
Кількість записів у тестовому наборі:     5000
Кількість записів у валідаційному наборі: 5000
```

Нормалізуємо дані шляхом поділу на 255.0

Створимо структуру моделі CNN. Оптимізатор – SGD із параметрами:

learning_rate=0.01, momentum=0.9, nesterov=True

Навчимо модель. Критерієм зупинки навчання є виконання всіх епох або досягнення точності 70% на валідаційному наборі.

Якщо модель уже створена, то вона буде завантажена з файлу, якщо ще немає навченої моделі, то програми почне процес тренування та збереже результат у файл. Максимальна кількість епох = 25

Усі епохи навчання:

```
Epoch 1/25

1563/1563 [==============================] - 61s 37ms/step - loss: 1.8561 -
accuracy: 0.3091 - val_loss: 1.5019 - val_accuracy: 0.4470

Epoch 2/25

1563/1563 [==============================] - 57s 37ms/step - loss: 1.5387 -
accuracy: 0.4395 - val_loss: 1.3889 - val_accuracy: 0.5162

Epoch 3/25

1563/1563 [==============================] - 55s 35ms/step - loss: 1.4094 -
accuracy: 0.4912 - val_loss: 1.3174 - val_accuracy: 0.5264

Epoch 4/25

1563/1563 [==============================] - 55s 35ms/step - loss: 1.3311 -
accuracy: 0.5239 - val_loss: 1.2609 - val_accuracy: 0.5534

Epoch 5/25
```

```
1563/1563 [==============================] - 56s 36ms/step - loss: 1.2753 -
accuracy: 0.5482 - val_loss: 1.1930 - val_accuracy: 0.5766

Epoch 6/25

1563/1563 [==============================] - 58s 37ms/step - loss: 1.2253 -
accuracy: 0.5695 - val_loss: 1.1991 - val_accuracy: 0.5770

Epoch 7/25

1563/1563 [==============================] - 61s 39ms/step - loss: 1.1899 -
accuracy: 0.5839 - val_loss: 1.1125 - val_accuracy: 0.6062

Epoch 8/25

1563/1563 [==============================] - 57s 36ms/step - loss: 1.1596 -
accuracy: 0.5968 - val_loss: 1.2204 - val_accuracy: 0.5854

Epoch 9/25

1563/1563 [==============================] - 56s 36ms/step - loss: 1.1343 -
accuracy: 0.6062 - val_loss: 1.0969 - val_accuracy: 0.6128

Epoch 10/25

1563/1563 [==============================] - 56s 36ms/step - loss: 1.1150 -
accuracy: 0.6119 - val_loss: 1.0995 - val_accuracy: 0.6262

Epoch 11/25

1563/1563 [==============================] - 57s 37ms/step - loss: 1.0995 -
accuracy: 0.6213 - val_loss: 1.1404 - val_accuracy: 0.6216

Epoch 12/25

1563/1563 [==============================] - 58s 37ms/step - loss: 1.0792 -
accuracy: 0.6249 - val_loss: 0.9526 - val_accuracy: 0.6756

Epoch 13/25

1563/1563 [==============================] - 56s 36ms/step - loss: 1.0629 -
accuracy: 0.6333 - val_loss: 1.1411 - val_accuracy: 0.6128

Epoch 14/25

1563/1563 [==============================] - 57s 36ms/step - loss: 1.0575 -
accuracy: 0.6370 - val_loss: 0.9376 - val_accuracy: 0.6768

Epoch 15/25

1563/1563 [==============================] - 55s 36ms/step - loss: 1.0508 -
accuracy: 0.6379 - val_loss: 1.0970 - val_accuracy: 0.6252

Epoch 16/25

1563/1563 [==============================] - 57s 36ms/step - loss: 1.0336 -
accuracy: 0.6443 - val_loss: 1.0606 - val_accuracy: 0.6516

Epoch 17/25

1563/1563 [==============================] - 60s 38ms/step - loss: 1.0275 -
accuracy: 0.6477 - val_loss: 0.9057 - val_accuracy: 0.6918
```

```
Epoch 18/25

1563/1563 [==============================] - 55s 35ms/step - loss: 1.0171 -
accuracy: 0.6487 - val_loss: 1.0018 - val_accuracy: 0.6658

Epoch 19/25

1563/1563 [==============================] - 56s 36ms/step - loss: 1.0095 -
accuracy: 0.6536 - val_loss: 0.9814 - val_accuracy: 0.6628

Epoch 20/25

1563/1563 [==============================] - 55s 35ms/step - loss: 1.0048 -
accuracy: 0.6556 - val_loss: 0.9963 - val_accuracy: 0.6702

Epoch 21/25

1563/1563 [==============================] - 55s 35ms/step - loss: 1.0052 -
accuracy: 0.6567 - val_loss: 0.9297 - val_accuracy: 0.6844

Epoch 22/25

1563/1563 [==============================] - 55s 35ms/step - loss: 0.9895 -
accuracy: 0.6608 - val_loss: 0.9396 - val_accuracy: 0.6872

Epoch 23/25

1563/1563 [==============================] - 55s 35ms/step - loss: 0.9911 -
accuracy: 0.6598 - val_loss: 0.9398 - val_accuracy: 0.6874

Epoch 24/25

1563/1563 [==============================] - 55s 35ms/step - loss: 0.9861 -
accuracy: 0.6617 - val_loss: 1.0566 - val_accuracy: 0.6540

Epoch 25/25

1563/1563 [==============================] - 54s 35ms/step - loss: 0.9730 -
accuracy: 0.6677 - val_loss: 0.9695 - val_accuracy: 0.6776
```

Модель завершила своє навчання після 25 епох досягнувши точності 0.6776

Протестуємо модель та знайдемо метрики для тестового набору:

```
Accuracy  моделі на тестових даних: 67.74 %
Precision моделі на тестових даних: 69.33 %
Recall    моделі на тестових даних: 67.74 %
F1-score  моделі на тестових даних: 67.42 %
```

Додаток А – Код програми

```
# КМ-01, Іваник Юрій, Лаб 4
### Імпортуємо бібліотеки
import numpy as np
import tensorflow as tf
from keras import layers, models
from keras.optimizers import SGD
from keras.datasets import cifar10
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
### Завантажимо дані
batch_size = 32

(X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train / 255.0
X_test  = X_test / 255.0

y_train = to_categorical(y_train, 10)
y_test  = to_categorical(y_test,  10)

X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, random_state=42)

print(f'Кількість записів у навчальному наборі:   {len(X_train)}')
print(f'Кількість записів у тестовому наборі:     {len(X_test)}')
```

```
print(f'Кількість записів у валідаційному наборі: {len(X_val)}')
### Створимо структуру моделі CNN
model = models.Sequential()

# Додавання згорткових шарів та шарів пулінгу
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))  # Додаємо Dropout

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))  # Додаємо Dropout

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))  # Додаємо Dropout
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'])
### Навчаємо модель
class myCallback(tf.keras.callbacks.Callback):
```

```
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('val_accuracy') is not None and logs.get('val_accuracy') > 0.7:
            self.model.stop_training = True
            print(f'\n\nAccuracy моделі на валідаційних даних > 70 %, зупинка навчання після епохи № {epoch+1}')


callbacks = myCallback()
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),
                        callbacks=[callbacks])
    model.save('my_model.keras') # Зберігаємо модель
### Тестуємо модель та знаходимо метрики
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')

print(f'Accuracy  моделі на тестових даних: {test_accuracy * 100:.2f} %')
print(f'Precision моделі на тестових даних: {test_precision * 100:.2f} %')
print(f'Recall    моделі на тестових даних: {test_recall * 100:.2f} %')
print(f'F1-score  моделі на тестових даних: {test_f1_score * 100:.2f} %')
```