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-03
Шаповалов Г. Г.

Викладач:
Професор кафедри ПМА
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості

Згорткові нейронні мережі (ЗНМ) є потужним інструментом у галузі глибокого навчання, особливо в обробці зображень та відео. Основним компонентом ЗНМ є згорткові шари, які дозволяють ефективно взаємодіяти з просторовою структурою вхідних даних.

Кожен згортковий шар включає набір фільтрів (ядро), які рухаються по вхідному зображенню та виконують операцію згортки. Ці фільтри виявляють різні особливості, такі як границі, форми або текстури, що дозволяє ЗНМ ефективно витягувати важливі ознаки з вхідних даних.

Після згортки використовуються шари пулінгу, які зменшують просторовий розмір отриманих карт ознак, зберігаючи при цьому важливі інформаційні особливості. Це допомагає зменшити кількість параметрів та обчислювальний обсяг, що поліпшує ефективність навчання та прискорює роботу мережі.

Після згортки та пулінгу використовують повнозв'язані шари, які об'єднують отримані ознаки для прийняття рішення або класифікації. Згорткові нейронні мережі успішно використовуються в різних задачах, таких як розпізнавання об'єктів, визначення обличчя, аналіз зображень та багато інших, завдяки їх здатності автоматично вивчати ієрархічні функції та властивості вхідних даних.

Основна частина

Для початку імпортуємо потрібні бібліотеки:

Numpy – для математичних розрахунків

Tensorflow – для створення НМ

Keras – для завантаження датасету, перекодування міток

Sklearn – для розбиття наборів даних та знаходження метрик у кінці

Валідаційні дані – дані для тестування після кожної епохи. Походження даних – розбиття тестового набору на 2 рівні частини.

В якості оптимізатора моделі вибрано SGD з параметрами: ***learning_rate=0.01, momentum=0.9, nesterov=True***

Датасет має 10 класів зображень, тому використовувалась ***categorical_crossentropy*** в якості ***loss функції***

Модель навчається 25 епох.

Скріншоти роботи програми:

```
КМ-03 | Шаповалов Г. Г. | Лаб 4


[LOG]  Завантажуємо та нормалізуємо дані
Розмірність тренувального набору: 50000
Розмірність тестового набору:     5000
Розмірність валідаційного набору: 5000


[LOG]  Створюємо модель
2023-12-02 18:48:57.033971: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorF
low binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other oper
ations, rebuild TensorFlow with the appropriate compiler flags.


[LOG]  Тренуємо модель
Epoch 1/25
782/782 [==============================] - 53s 64ms/step - loss: 1.9225 - accuracy: 0.2792 -
val_loss: 1.5654 - val_accuracy: 0.4172
Epoch 2/25
782/782 [==============================] - 49s 63ms/step - loss: 1.5710 - accuracy: 0.4297 -
val_loss: 1.3946 - val_accuracy: 0.5076
Epoch 3/25
782/782 [==============================] - 49s 63ms/step - loss: 1.4290 - accuracy: 0.4839 -
val_loss: 1.4130 - val_accuracy: 0.5124
Epoch 4/25
782/782 [==============================] - 49s 62ms/step - loss: 1.3262 - accuracy: 0.5290 -
val_loss: 1.1913 - val_accuracy: 0.5804
Epoch 5/25
782/782 [==============================] - 49s 63ms/step - loss: 1.2553 - accuracy: 0.5547 -
val_loss: 1.1490 - val_accuracy: 0.6026
Epoch 6/25
782/782 [==============================] - 51s 65ms/step - loss: 1.1979 - accuracy: 0.5782 -
val_loss: 1.1549 - val_accuracy: 0.5870
Epoch 7/25
782/782 [==============================] - 51s 65ms/step - loss: 1.1595 - accuracy: 0.5926 -
val_loss: 1.0873 - val_accuracy: 0.6232
Epoch 8/25
782/782 [==============================] - 50s 64ms/step - loss: 1.1215 - accuracy: 0.6054 -
val_loss: 0.9982 - val_accuracy: 0.6486
Epoch 9/25
782/782 [==============================] - 50s 64ms/step - loss: 1.0905 - accuracy: 0.6166 -
val_loss: 1.1936 - val_accuracy: 0.5892
Epoch 10/25
782/782 [==============================] - 50s 64ms/step - loss: 1.0601 - accuracy: 0.6301 -
val_loss: 1.0166 - val_accuracy: 0.6500
Epoch 11/25
782/782 [==============================] - 50s 64ms/step - loss: 1.0330 - accuracy: 0.6389 -
val_loss: 0.9627 - val_accuracy: 0.6608
Epoch 12/25
782/782 [==============================] - 50s 64ms/step - loss: 1.0077 - accuracy: 0.6489 -
val_loss: 0.9581 - val_accuracy: 0.6796
```

```
Epoch 13/25
782/782 [==============================] - 50s 64ms/step - loss: 0.9916 - accuracy: 0.6522 -
val_loss: 0.9137 - val_accuracy: 0.6816
Epoch 14/25
782/782 [==============================] - 50s 64ms/step - loss: 0.9706 - accuracy: 0.6613 -
val_loss: 0.9929 - val_accuracy: 0.6602
Epoch 15/25
782/782 [==============================] - 50s 64ms/step - loss: 0.9602 - accuracy: 0.6657 -
val_loss: 0.9335 - val_accuracy: 0.6818
Epoch 16/25
782/782 [==============================] - 50s 64ms/step - loss: 0.9380 - accuracy: 0.6750 -
val_loss: 0.9453 - val_accuracy: 0.6644
Epoch 17/25
782/782 [==============================] - 50s 64ms/step - loss: 0.9280 - accuracy: 0.6777 -
val_loss: 0.8185 - val_accuracy: 0.7194
Epoch 18/25
782/782 [==============================] - 50s 64ms/step - loss: 0.9141 - accuracy: 0.6833 -
val_loss: 0.8613 - val_accuracy: 0.7056
Epoch 19/25
782/782 [==============================] - 50s 64ms/step - loss: 0.8966 - accuracy: 0.6892 -
val_loss: 0.9404 - val_accuracy: 0.6722
Epoch 20/25
782/782 [==============================] - 50s 64ms/step - loss: 0.8872 - accuracy: 0.6923 -
val_loss: 1.2731 - val_accuracy: 0.5972
Epoch 21/25
782/782 [==============================] - 50s 64ms/step - loss: 0.8826 - accuracy: 0.6938 -
val_loss: 0.8871 - val_accuracy: 0.7004
Epoch 22/25
782/782 [==============================] - 50s 64ms/step - loss: 0.8722 - accuracy: 0.6962 -
val_loss: 0.7919 - val_accuracy: 0.7290
Epoch 23/25
782/782 [==============================] - 50s 64ms/step - loss: 0.8565 - accuracy: 0.7008 -
val_loss: 0.7770 - val_accuracy: 0.7304
Epoch 24/25
782/782 [==============================] - 50s 64ms/step - loss: 0.8510 - accuracy: 0.7049 -
val_loss: 0.8149 - val_accuracy: 0.7210
Epoch 25/25
782/782 [==============================] - 51s 65ms/step - loss: 0.8447 - accuracy: 0.7062 -
val_loss: 0.7737 - val_accuracy: 0.7374


[LOG]  Тестуємо модель
Accuracy  = 73.06 %
Precision = 73.34 %
Recall    = 73.06 %
F1-score  = 72.66 %
```

Модель натренувалась на 25-ти епохах. Досягла точності в 0.7374 на валідаційному наборі на останній епосі. В результаті, метрики показали непогані результати, отже модель змогла більш-менш навчитись.

Додаток А – Код програми

```
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


print('\n\nКМ-03 | Шаповалов Г. Г. | Лаб 4')


print(f'\n\n[LOG]  Завантажуємо та нормалізуємо дані')
batch_size = 64

(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'Розмірність тренувального набору: {len(X_train)}')
print(f'Розмірність тестового набору:     {len(X_test)}')
print(f'Розмірність валідаційного набору: {len(X_val)}')


print(f'\n\n[LOG]  Створюємо модель')
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'))

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

sgd = SGD(learning_rate=0.01, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd,
              loss='categorical_crossentropy',
              metrics=['accuracy'])


print(f'\n\n[LOG]  Тренуємо модель')
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')


print(f'\n\n[LOG]  Тестуємо 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')

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