Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота № 6

з дисципліни
«Штучний інтелект в задачах обробки зображень»

Виконав:
студент групи ІМ-11
Царик Микола Миколайович

Перевірив:
Нікітін Валерій Андрійович

Київ 2024

Мета: Отримати навички реалізації архітектури AlexNet CNN з використанням бібліотек TensorFlow та Keras

Завдання:

1. Реалізувати засобами TensorFlow та Keras AlexNet;
2. Отримати оцінку точності навченої мережі.

Хід виконання:

1) Скрипт починається з імпорту TensorFlow, TensorFlow Datasets для завантаження набору даних, моделей і шарів Keras для побудови нейронної мережі, Matplotlib для візуалізації, а також NumPy для операцій з масивами.

```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
```

2) В цьому кроці визначається клас ImageClassifier, який інкапсулює всі основні методи, необхідні для роботи з нейронною мережею, включаючи завантаження даних, передобробку зображень, створення моделі, тренування моделі та відображення результатів.

```
class ImageClassifier:
    def __init__(self, dataset_name='imagenette/160px-v2',
data_dir='D:/tensorflow_datasets'):
        try:
            self.data, self.info = tfds.load(dataset_name, with_info=True,
as_supervised=True, data_dir=data_dir)
            self.train_set, self.validation_set = self.data['train'],
self.data['validation']
            self.CLASS_NAMES = ['tench', 'English springer', 'cassette player',
'chainsaw',
                                'church', 'horn', 'garbage truck', 'gas pump',
'golf ball', 'parachute']
        except Exception as e:
            print(f"Помилка завантаження даних: {e}")
            raise SystemExit

        self.model = None
```

3) Методи process_image та preprocess_dataset відповідають за стандартизацію і масштабування зображень, а також підготовку пакетів даних для тренування та валідації.

```
def process_image(self, image):
        image = tf.image.per_image_standardization(image)
        image = tf.image.resize(image, (64, 64))
        return image

    def preprocess_dataset(self):
        train_dataset = self.train_set.map(lambda image, label:
(self.process_image(image), label)).batch(32)
        validation_dataset = self.validation_set.map(lambda image, label:
(self.process_image(image), label)).batch(32)
        return train_dataset, validation_dataset
```

4) create_AlexNet створює модель AlexNet з використанням Keras, а train_model тренує модель, використовуючи підготовлені набори даних.

```
def create_AlexNet(self):
        self.model = models.Sequential([
            layers.Conv2D(filters=128, kernel_size=(11, 11), strides=(4, 4),
activation='relu', input_shape=(64, 64, 3)),
            layers.BatchNormalization(),
            layers.MaxPool2D(pool_size=(2, 2)),
            layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1),
activation='relu', padding="same"),
            layers.BatchNormalization(),
            layers.MaxPool2D(pool_size=(3, 3)),
            layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding="same"),
            layers.BatchNormalization(),
            layers.Conv2D(filters=256, kernel_size=(1, 1), strides=(1, 1),
activation='relu', padding="same"),
            layers.BatchNormalization(),
            layers.Conv2D(filters=256, kernel_size=(1, 1), strides=(1, 1),
activation='relu', padding="same"),
            layers.BatchNormalization(),
            layers.MaxPool2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dense(1024, activation='relu'),
            layers.Dropout(0.5),
            layers.Dense(1024, activation='relu'),
            layers.Dropout(0.5),
```

```
            layers.Dense(10, activation='softmax')
        ])
        self.model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
        self.model.summary()

    def train_model(self, epochs):
        train_dataset, validation_dataset = self.preprocess_dataset()
        history = self.model.fit(train_dataset, epochs=epochs,
validation_data=validation_dataset, validation_freq=1)
        self.model.save('class_recognizer_imagenet1.h5')
        return history
```

5) display_image і show_accuracy відображають зображення і графіки точності відповідно, тоді як load_model завантажує готову модель для подальших передбачень.

```
def load_model(self, path):
        self.model = models.load_model(path)

    def display_image(self, image, prediction):
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.xlabel(f'{prediction}')
        plt.imshow(image)
        plt.show()

    def show_accuracy(self, history):
        plt.plot(history.history['accuracy'], label='accuracy')
        plt.plot(history.history['val_accuracy'], label='val_accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.ylim([0, 1])
        plt.legend(loc='lower right')
        plt.show()
```

6) Метод make_predictions використовується для генерації передбачень з використанням моделі, показуючи результати на вхідних зображеннях.

```
 def make_predictions(self, num_predictions=10):
        self.validation_set =
self.validation_set.shuffle(buffer_size=len(self.validation_set))
        for _ in range(num_predictions):
            image, label = next(iter(self.validation_set))
            processed_image = self.process_image(image)
            processed_image = tf.expand_dims(processed_image, axis=0)
            prediction = np.argmax(self.model.predict(processed_image)[0])
            self.display_image(image.numpy(), self.CLASS_NAMES[prediction])
```

7) Виклик основних функцій

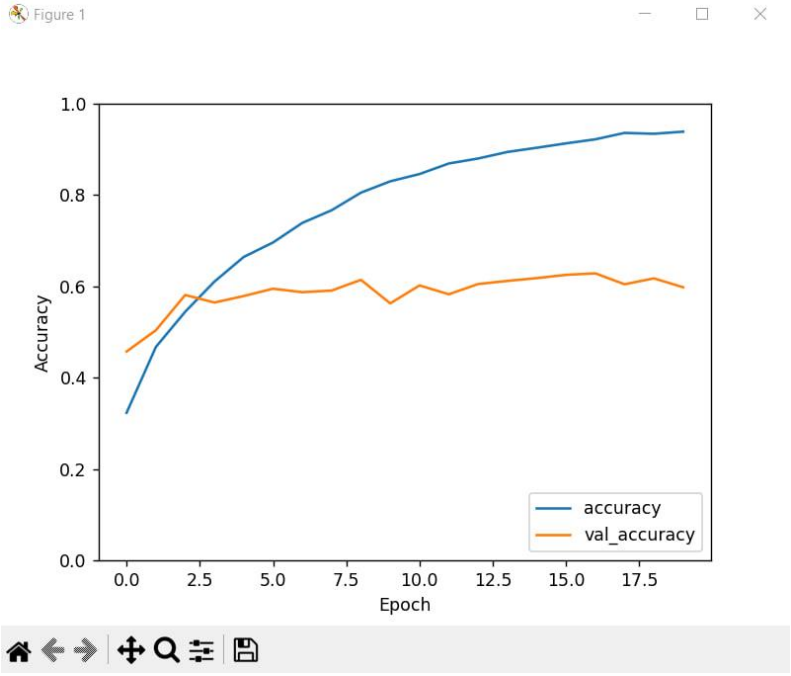
```
def main():
    classifier = ImageClassifier()
    # classifier.create_AlexNet()
    # history = classifier.train_model(20)
    # classifier.show_accuracy(history)
    classifier.load_model('class_recognizer_imagenet1.h5')
    classifier.make_predictions(10)

main()
```

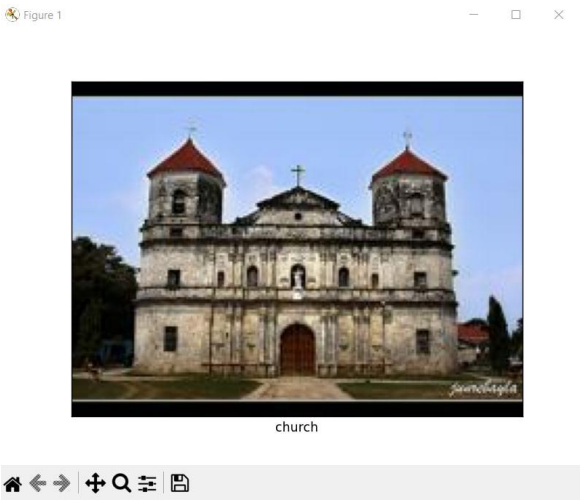
Інформація про модель:

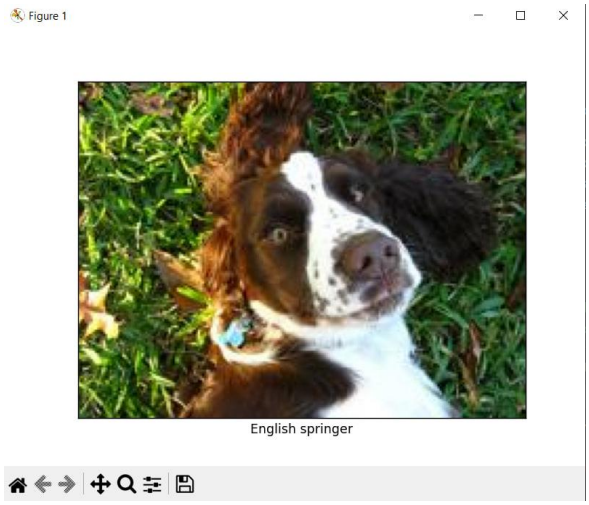
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 128)	46,592
batch_normalization (BatchNormalization)	(None, 14, 14, 128)	512
max_pooling2d (MaxPooling2D)	(None, 7, 7, 128)	0
conv2d_1 (Conv2D)	(None, 7, 7, 256)	819,456
batch_normalization_1 (BatchNormalization)	(None, 7, 7, 256)	1,024
max_pooling2d_1 (MaxPooling2D)	(None, 2, 2, 256)	0
conv2d_2 (Conv2D)	(None, 2, 2, 256)	590,080
batch_normalization_2 (BatchNormalization)	(None, 2, 2, 256)	1,024
conv2d_3 (Conv2D)	(None, 2, 2, 256)	65,792
batch_normalization_3 (BatchNormalization)	(None, 2, 2, 256)	1,024
conv2d_4 (Conv2D)	(None, 2, 2, 256)	65,792
batch_normalization_4 (BatchNormalization)	(None, 2, 2, 256)	1,024

Графік точності:



Результати:





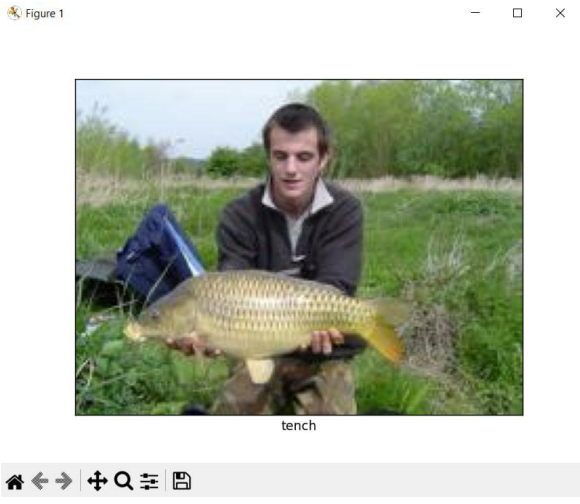


Figure 1

tench

Figure 1

golf ball

Figure 1

parachute

Figure 1

parachute

Figure 1

tench

Figure 1

cassette player

Висновок:

У цій лабораторній роботі було успішно реалізовано згорткову нейронну мережу AlexNet для класифікації зображень з частини набору даних ImageNet. Спочатку було завантажено і підготовано навчальні та валідаційні дані, оброблено зображення до розміру 64x64 пікселів. Після створення моделі проведено тренування на навчальних даних та оцінено ефективність на валідаційному наборі протягом 20 епох. Завдяки цій роботі мною було набуто практичних навичок у створенні та тренуванні згорткових нейронних мереж, досягнувши точності розпізнавання в 58%

Контрольні запитання:

1. Що таке AlexNet?

AlexNet – це глибока нейронна мережа, створена Алексом Крижевським, Іллею Сутскевером та Джеффрі Гінтоном у 2012 році. Вона здобула велику популярність після перемоги у конкурсі ImageNet Large Scale Visual Recognition Challenge (ILSVRC) у 2012 році, значно покращивши точність розпізнавання зображень порівняно з попередніми моделями.

2. Яка архітектура AlexNet?

Архітектура AlexNet складається з 8 шарів з навчуваними параметрами. Перші 5 шарів – це згорткові шари (Convolutional Layers), деякі з яких слідують за шаром підвибірки (max-pooling). Останні 3 шари – це повнозв'язні (fully connected) шари. Мережа використовує функцію активації ReLU (Rectified Linear Unit). Використовується дропаут (dropout) для запобігання перенавчанню. Лінійна нормалізація (Local Response Normalization) для поліпшення узгодженості виходів згорткових шарів.

3. Які переваги та недоліки даної мережі?

Переваги:

- AlexNet значно покращила точність класифікації зображень у порівнянні з попередніми моделями.
- Функція активації ReLU пришвидшила процес навчання завдяки нелінійності та відсутності проблеми затухання градієнта.
- Зменшення перенавчання завдяки випадковому вимкненню нейронів під час навчання.
- Мережа була розподілена на два графічні процесори для прискорення навчання.

Недоліки:

- Для тренування потрібні потужні графічні процесори та значні ресурси пам'яті.
- Потреба в значній кількості даних для уникнення перенавчання.
- Вимагає значних зусиль для налаштування та оптимізації.

4. Навіщо потрібна компіляція мережі?

Компіляція нейронної мережі означає налаштування моделі перед її навчанням. Це включає: Визначення функції втрат (loss function), яка буде використовуватися для оцінки помилки моделі. Вибір оптимізатора (optimizer), який буде використовуватися для мінімізації функції втрат. Визначення метрик (metrics), які будуть використовуватися для оцінки продуктивності моделі під час навчання та тестування.

5. Як відбувається навчання?

Навчання нейронної мережі, складається з таких кроків:

1. **Ініціалізація:** Випадкове встановлення ваг та зміщень (biases).
2. **Прямий прохід (Forward Pass):** Обчислення передбачуваного виходу моделі для кожного вхідного зображення, проходячи через усі шари мережі.
3. **Обчислення втрат:** Використання функції втрат для оцінки різниці між передбачуваними виходами та фактичними мітками.
4. **Зворотний прохід (Backward Pass):** Обчислення градієнтів функції втрат відносно ваг мережі за допомогою методу зворотного розповсюдження помилки (backpropagation).
5. **Оновлення ваг:** Використання оптимізатора для коригування ваг та зміщень, зменшуючи функцію втрат.
6. **Ітерація:** Повторення кроків 2-5 для багатьох епох (epochs) до досягнення задовільної точності.