НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

Звіт до комп’ютерного практикуму 4 з дисципліни:
“**Програмні засоби проектування та реалізації нейромережевих систем**”

Виконав
ІП-01 Черпак А.В.

Перевірив:
Шимкович В.М.

Київ – 2023

Комп’ютерний практикум 4

Тема: Згорткові нейронні мережі для розпізнавання зображень

Завдання:

Написати програму що реалізує згорткову нейронну мережу AlexNet для розпізнавання об’єктів з датасету ImageNet цифр.

Виконання:

Завантаження та обробка даних:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.python.data.ops.dataset_ops import DatasetV1
from numpy import ndarray


def get_data() -> tuple[DatasetV1, DatasetV1, DatasetV1]:
    (train_images, train_labels), (test_images, test_labels) =
keras.datasets.cifar10.load_data()

    validation_images, validation_labels = train_images[:5000],
train_labels[:5000]
    train_images, train_labels = train_images[5000:], train_labels[5000:]

    train_ds = tf.data.Dataset.from_tensor_slices((train_images, train_labels))
    test_ds = tf.data.Dataset.from_tensor_slices((test_images, test_labels))
    validation_ds = tf.data.Dataset.from_tensor_slices((validation_images,
validation_labels))
    return train_ds, test_ds, validation_ds


def visualize_data(train_ds: DatasetV1, classes: list[str]) -> None:
    plt.figure(figsize=(20, 20))
    for i, (image, label) in enumerate(train_ds.take(25)):
        ax = plt.subplot(5, 5, i + 1)
        plt.imshow(image)
        plt.title(classes[label.numpy()[0]])
        plt.axis('off')

    plt.show()


def process_images(image: ndarray, label: str) -> tuple[ndarray, str]:
    image = tf.image.per_image_standardization(image)
    image = tf.image.resize(image, (227, 227))
    return image, label


def get_ds_size(ds: DatasetV1) -> int:
    return tf.data.experimental.cardinality(ds).numpy()


def process_ds(ds: DatasetV1) -> DatasetV1:
    return
ds.map(process_images).shuffle(buffer_size=get_ds_size(ds)).batch(batch_size=32,
drop_remainder=True)
```

Побудова та компіляція моделі:

```
import tensorflow as tf
from keras.layers import Conv2D, BatchNormalization, MaxPool2D, Flatten, Dense,
Dropout
from keras.models import Sequential


def AlexNet(class_number) -> Sequential:
    model = Sequential([
        Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4),
               activation='relu', input_shape=(227, 227, 3)),
        BatchNormalization(),
        MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
        Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1),
               activation='relu', padding='same'),
        BatchNormalization(),
        MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
        Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1),
               activation='relu', padding='same'),
        BatchNormalization(),
        Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1),
               activation='relu', padding='same'),
        BatchNormalization(),
        Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1),
               activation='relu', padding='same'),
        BatchNormalization(),
        MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
        Flatten(),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(class_number, activation='softmax')
    ])

    return model


def compile_model(model: Sequential):
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=tf.optimizers.SGD(learning_rate=0.001),
        metrics=['accuracy']
    )
    model.summary()
    return model
```

Тренування та тестування моделі:

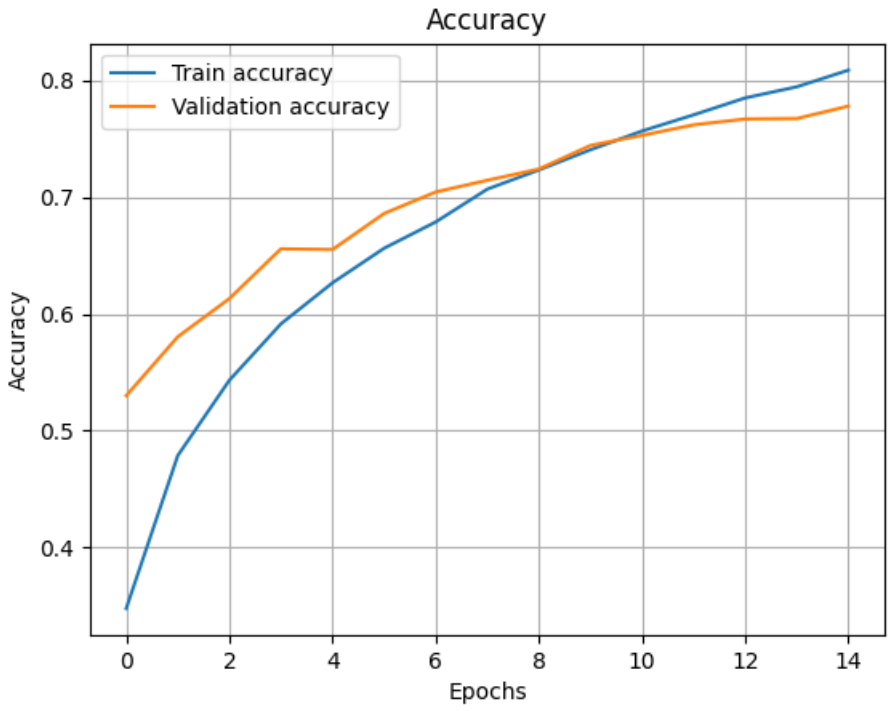
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from neural_network_types import FeedForwardBackprop, CascadeForwardBackprop,
ElmanBackprop, NeuralNetworkModel
from training_data_generation import data_split, generate_data


def get_learning_rate(epochs, batch_size):
    initial_learning_rate = 10 ** (-3)
    final_learning_rate = 10 ** (-7)
    learning_rate_decay_factor = (final_learning_rate / initial_learning_rate) **
(1 / epochs)
    steps_per_epoch = int(len(train) / batch_size)
    return tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=initial_learning_rate,
        decay_steps=steps_per_epoch,
        decay_rate=learning_rate_decay_factor
    )
```

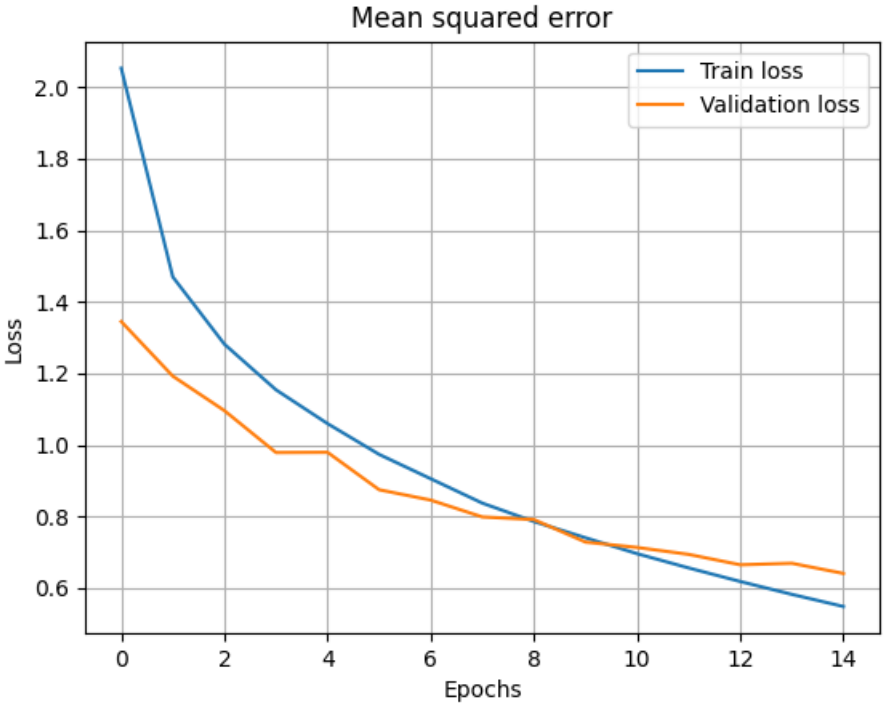
```
def train_model(model_type: type(NeuralNetworkModel), hidden_neurons, train_data,
test_data, epochs_num, batch_sz, learning_rate):
    model_t = model_type(hidden_neurons)
    model = model_t.model
    model.compile(loss='mean_squared_error',
optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate))
    model.summary()
    return model.fit(np.reshape(train_data[:, :2], (-1, 2)), train_data[:, 2],
epochs=epochs_num, batch_size=batch_sz,
                     validation_data=(np.reshape(test_data[:, :2], (-1, 2)),
test_data[:, 2]), verbose=1).history, model_t.nn_model_name


def graph(train_loss, val_loss, name):
    plt.title(name+'`s mean_squared_error')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(train_loss, label='Train loss')
    plt.plot(val_loss, label='Validation loss')
    plt.grid()
    plt.legend()
    plt.show()


if __name__ == '__main__':
    data = generate_data(1000000)
    train, test = data_split(data, 0.7)
    epochs = 500
    batch_size = 10000
    lr = get_learning_rate(epochs, batch_size)
    cases = [# (FeedForwardBackprop, [10]), (FeedForwardBackprop, [20]),
             # (CascadeForwardBackprop, [20]), (CascadeForwardBackprop, [10, 10]),
             # (ElmanBackprop, [15]),
        (ElmanBackprop, [10, 10, 10])]

    for (nn_model, hidden_neurons_number) in cases:
        log, name = train_model(nn_model, hidden_neurons_number, train, test,
epochs, batch_size, lr)
        graph(log['loss'], log['val_loss'], name)
```

Візуалізація даних:

Результати тренування:

```
Epoch 1/10
1406/1406 [==============================] - 1582s 1s/step - loss: 2.0493 - accuracy: 0.3516 - val_loss: 1.3424 - val_accuracy: 0.5274
Epoch 2/10
1406/1406 [==============================] - 1533s 1s/step - loss: 1.4775 - accuracy: 0.4755 - val_loss: 1.2074 - val_accuracy: 0.5749
Epoch 3/10
1406/1406 [==============================] - 1541s 1s/step - loss: 1.2941 - accuracy: 0.5396 - val_loss: 1.0796 - val_accuracy: 0.6262
Epoch 4/10
1406/1406 [==============================] - 1583s 1s/step - loss: 1.1652 - accuracy: 0.5880 - val_loss: 0.9930 - val_accuracy: 0.6536
Epoch 5/10
1406/1406 [==============================] - 1882s 1s/step - loss: 1.0627 - accuracy: 0.6232 - val_loss: 0.9402 - val_accuracy: 0.6715
Epoch 6/10
1406/1406 [==============================] - 2340s 2s/step - loss: 0.9765 - accuracy: 0.6569 - val_loss: 0.8566 - val_accuracy: 0.7063
Epoch 7/10
1406/1406 [==============================] - 2486s 2s/step - loss: 0.8991 - accuracy: 0.6815 - val_loss: 0.8074 - val_accuracy: 0.7204
Epoch 8/10
1406/1406 [==============================] - 2866s 2s/step - loss: 0.8430 - accuracy: 0.7031 - val_loss: 0.7865 - val_accuracy: 0.7274
Epoch 9/10
1406/1406 [==============================] - 3149s 2s/step - loss: 0.7866 - accuracy: 0.7224 - val_loss: 0.7525 - val_accuracy: 0.7420
Epoch 10/10
1406/1406 [==============================] - 3426s 2s/step - loss: 0.7358 - accuracy: 0.7410 - val_loss: 0.7402 - val_accuracy: 0.7442
```





(Насправді було здійснено 15 епох, але останні 5, на жаль, заскрінити не вдалося.)

Результати тестування:

```
Training data size: 45000
Test data size: 10000
Validation data size: 5000
312/312 [==============================] - 64s 199ms/step - loss: 0.6693 - accuracy: 0.7725
```

Висновок:

Під час виконання комп'ютерного практикуму ми реалізували згорткову нейромережу AlexNet для розпізнавання зображень з дата сету ImageNet. Після 15 епох тренувань ми отримали точність 77,25%, що насправді дуже навіть непогано. Щоправда, на її тренування було затрачено не менше доби.