МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

ЗВІТ

про виконання лабораторної роботи №3

з дисципліни: «Обробка зображень методами штучного інтелекту»

на тему: «Класифікація зображень. Застосування нейромереж для пошуку подібних зображень»

Виконав:

студент групи КН-410

Шиманський П.С.

Прийняв:

Пелешко Д.Д.

Львів – 2021

Мета: набути практичних навиків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації.

Теоретичні відомості

Зго́рткові нейро́нні мере́жі (**ЗНМ**, англ. *convolutional neural network, CNN, ConvNet*) в машинному навчанні — це клас глибинних штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу візуальних зображень.

ЗНМ використовують різновид багатошарових перцептронів, розроблений так, щоби вимагати використання мінімального обсягу попередньої обробки. Вони відомі також як **інваріа́нтні віднóсно зсу́ву** (англ. *shift invariant*) або **просторóво інваріа́нтні шту́чні нейро́нні мере́жі** (англ. *space invariant artificial neural networks, SIANN*), виходячи з їхньої архітектури спільних ваг та характеристик інваріантності відносно паралельного перенесення.

Код програми

```
from google.colab import drive
drive.mount('/content/drive')
!pip install keras-tuner
from tensorflow.keras import layers
from tensorflow import keras
import tensorflow as tf
import kerastuner as kt
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from typing import Tuple
from sklearn.manifold import TSNE
label_to_description = {
    0: 'T-Shirt',
    1: 'Trouser',
    2: 'Pullover',
    3: 'Press',
    4: 'Coat',
```

```
    5: 'Sandal',
    6: 'Shirt',
    7: 'Sneaker',
    8: 'Bag',
    9: 'Ankle boot'
}
def plot_image(img: np.ndarray, title: str = '', figsize: Tuple[int, int] = None, ax: 'AxesSubplot' = None, **params) -> None:
    if ax is None:
        plt.figure(figsize=figsize)
        ax = plt.axes()

    if title:
        ax.set_title(title)

    ax.imshow(img, **params)
    ax.axis('off')
def lenet(activation):
    model = keras.Sequential()

    model.add(layers.Conv2D(6, (5, 5), activation=activation, input_shape=(32, 32, 1)))
    model.add(layers.MaxPool2D((2, 2), 2))

    model.add(layers.Conv2D(16, (5, 5), activation=activation))
    model.add(layers.MaxPool2D((2, 2), 2))

    model.add(layers.Flatten())

    model.add(layers.Dense(120, activation='relu'))
    model.add(layers.Dense(84, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))

    return model

model = lenet('relu')
model.summary()
def one_hot(arr, max_=None):
    if not max_:
        max_ = arr.max() + 1

    size = arr.size
    encoding = np.zeros((size, max_))
    encoding[np.arange(size), arr] = 1

    return encoding
def get_Fashion_MNIST():
    (X_train, y_train), (X_test, y_test) = keras.datasets.fashion_mnist.load_data()

    X_train = X_train.astype('float32') / 255
    X_test = X_test.astype('float32') / 255
```

```
    X_train = np.pad(X_train, ((0, 0), (2, 2), (2, 2)), 'constant')
    X_test = np.pad(X_test, ((0, 0), (2, 2), (2, 2)), 'constant')

    X_train = X_train.reshape((-1, 32, 32, 1))
    X_test = X_test.reshape((-1, 32, 32, 1))

    y_train = one_hot(y_train)
    y_test = one_hot(y_test)

    return (X_train, y_train), (X_test, y_test)
def lenet_model(hp):

    hp_activation = hp.Choice('activation', ['relu', 'elu', 'tanh', 'swish'])

    model = lenet(hp_activation)



    hp_learning_rate = hp.Float('learning_rate', min_value=3e-4, max_value=3e-3)

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                  loss=keras.losses.CategoricalCrossentropy(),
                  metrics=['accuracy'])

    return model
(X_train, y_train), (X_test, y_test) = get_Fashion_MNIST()
idx = np.random.randint(0, len(X_train))
plot_image(X_train[idx].squeeze(-1), cmap='gray')
print(f'This is {label_to_description[y_train[idx].argmax()]}')
tuner = kt.Hyperband(lenet_model,
                     objective='val_accuracy',
                     max_epochs=15,
                     project_name='lab3',
                     directory='drive/MyDrive')
stop_early = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
tuner.search(X_train, y_train, epochs=15,
             validation_data=(X_test, y_test), callbacks=[stop_early])
best_hp = tuner.get_best_hyperparameters()[0]
best_hp.get('activation'), best_hp.get('learning_rate')
model = lenet('relu')
model.compile(optimizer=keras.optimizers.Adam(learning_rate=best_hp.get('learning_rate')),
                  loss=keras.losses.CategoricalCrossentropy(),
                  metrics=['accuracy'])
history_log = model.fit(X_train, y_train, epochs=15,
             validation_data=(X_test, y_test), batch_size=8)
model.save('drive/MyDrive/')
history = history_log.history
history.keys()
f, ax = plt.subplots(2, figsize=(10, 10))
```

```
ax[0].plot(history['loss'], label='Train loss')
ax[0].plot(history['val_loss'], label='Val loss')
ax[0].set_title('Loss')
ax[0].legend()

ax[1].plot(history['accuracy'], label='Train accuracy')
ax[1].plot(history['val_accuracy'], label='Val accuracy')
ax[1].set_title('Accuracy')
ax[1].legend()
def create_feature_extractor(input_shape):
    def conv_block(input_, i):
        input_ = layers.Conv2D(8 * 2**i, kernel_size=(3, 3), activation='linear')(input_)
        input_ = layers.BatchNormalization()(input_)
        input_ = layers.Activation('relu')(input_)

        input_ = layers.Conv2D(16 * 2**i, kernel_size=(3, 3), activation='linear')(input_)
        input_ = layers.BatchNormalization()(input_)
        input_ = layers.Activation('relu')(input_)
        input_ = layers.MaxPool2D((2, 2))(input_)

        return input_

    img_in = layers.Input(shape=input_shape, name='FeatureNet_ImageInput')
    input_ = img_in

    for i in range(2):
        input_ = conv_block(input_, i)

    input_ = layers.Flatten()(input_)
    input_ = layers.Dense(32, activation='linear')(input_)
    input_ = layers.Dropout(0.5)(input_)
    input_ = layers.BatchNormalization()(input_)
    input_ = layers.Activation('relu')(input_)

    model = keras.Model(inputs=[img_in], outputs=[input_], name='FeatureGenerationModel')

    return model

feature_extractor_model = create_feature_extractor((32, 32, 1))
feature_extractor_model.summary()
def create_siamese_model(feature_extractor, input_shape=(32, 32, 1)):
    img_a_in = layers.Input(shape=input_shape, name='ImageA_Input')
    img_b_in = layers.Input(shape=input_shape, name='ImageB_Input')

    img_a_features = feature_extractor(img_a_in)
    img_b_features = feature_extractor(img_b_in)

    combined_features = layers.concatenate([img_a_features, img_b_features], name='merge_features')
```

```
    combined_features = layers.Dense(16, activation='linear')(combined_features)
    combined_features = layers.BatchNormalization()(combined_features)
    combined_features = layers.Activation('relu')(combined_features)

    combined_features = layers.Dense(4, activation='linear')(combined_features)
    combined_features = layers.BatchNormalization()(combined_features)
    combined_features = layers.Activation('relu')(combined_features)

    combined_features = layers.Dense(1, activation='sigmoid')(combined_features)

    model = keras.Model(inputs=[img_a_in, img_b_in], outputs=[combined_features], name='Similarity_model')

    return model

similarity_model = create_siamese_model(feature_extractor_model, (32, 32, 1))
similarity_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['mae'])
similarity_model.summary()
class SiameseGenerator(keras.utils.Sequence):
    def __init__(self, data, batch_size=16):
        self.X, self.y = data
        self.N = self.X.shape[0]

        self.batch_size = batch_size if batch_size > 0 else self.N

        self.indxs_a = np.arange(0, self.N)
        self.indxs_b = np.arange(0, self.N)

        self.on_epoch_end()

    def __len__(self):
        return int(np.floor(self.N / self.batch_size))

    def __getitem__(self, i):
        batch_start = i*self.batch_size
        batch_end = batch_start + self.batch_size

        idxs_a = self.indxs_a[batch_start:batch_end]
        idxs_b = self.indxs_b[batch_start:batch_end]

        labels_a = self.y[idxs_a].argmax(axis=1)
        labels_b = self.y[idxs_b].argmax(axis=1)

        res_labels = (labels_a == labels_b).astype('int')

        return (self.X[idxs_a], self.X[idxs_b]), res_labels


    def on_epoch_end(self):
```

```
        np.random.shuffle(self.indxs_a)
        np.random.shuffle(self.indxs_b)
class SiameseGenerator2(keras.utils.Sequence):
    def __init__(self, data, batch_size=32):
        self.batch_size = batch_size

        self.X, self.y = data

        self.labels = self.y.argmax(axis=1)
        self.N = self.X.shape[0]
        self.classes = np.unique(self.labels).size

        self.classes_indxs = [np.where(self.labels == category) for category in range(self.classes)]

        self.on_epoch_end()

    def __len__(self):
        return int(np.floor(self.N / self.batch_size))

    def __getitem__(self, i):
        pass

    def on_epoch_end(self):
        pass

    @staticmethod
    def __create_all_possible_pairs(labels):
        pass
train_generator = SiameseGenerator((X_train, y_train))
test_generator = SiameseGenerator((X_test, y_test), batch_size=-1)
loss_history = similarity_model.fit(train_generator, validation_data=test_generator,
                                    epochs=25)
history = loss_history.history
history.keys()
f, ax = plt.subplots(2, figsize=(10, 10))

ax[0].plot(history['loss'], label='Train loss')
ax[0].plot(history['val_loss'], label='Val loss')
ax[0].set_title('Loss')
ax[0].legend()

ax[1].plot(history['mae'], label='Train mae')
ax[1].plot(history['val_mae'], label='Val mae')
ax[1].set_title('MAE')
ax[1].legend()
test_features = feature_extractor_model.predict(X_test)
tsne_obj = TSNE(n_components=2, init='pca', random_state=101,
            method='barnes_hut', n_iter=500, verbose=2)
tsne_features = tsne_obj.fit_transform(test_features)
```

```
plt.figure(figsize=(10, 10))
test_labels = y_test.argmax(axis=1)
for label, name in label_to_description.items():
    indxs = np.where(test_labels == label)
    plt.scatter(tsne_features[indxs, 0], tsne_features[indxs, 1],
                label=name, alpha=0.8)
plt.legend()
Y_pred = model.predict(X_test)
confusion_matrix = tf.math.confusion_matrix(y_test.argmax(axis=1), Y_pred.argmax(axis=1)).numpy()
plt.figure(figsize=(18, 10))
sns.heatmap(confusion_matrix, annot=True, cmap=plt.cm.Blues,
            xticklabels=label_to_description.values(), yticklabels=label_to_description.values());
```