МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
Національний технічний університет   
«Харківський політехнічний інститут»  
Кафедра ГМКГ

**Лабораторна работа №6**

З дисципліни «Інтелектуальний аналіз даних»

Виконав:

Студент групи ІКМ-220 г.

Ульянов Кирило Юрійович

Перевірив:

Доц. Дашкевич А.О.

Харків 2023

**Мета роботи:** вивчення алгоритмiв iнтелектуального аналiзу даних

для обробки та аналiзу цифрових зображень.

**Завдання на роботу:** колоризацiя зображень у вiдтiнках сiрого через

автокодувальник. Навчання ознак для розв’язання задачi семантичної роз мiтки зображень.

**Завдання 1**

1. Обрані картинки для колоризації:

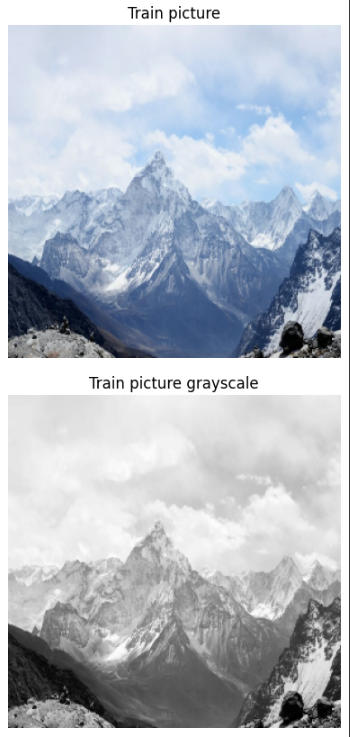
Для тренування



Для тесту



Переводимо у відтінки сірого:

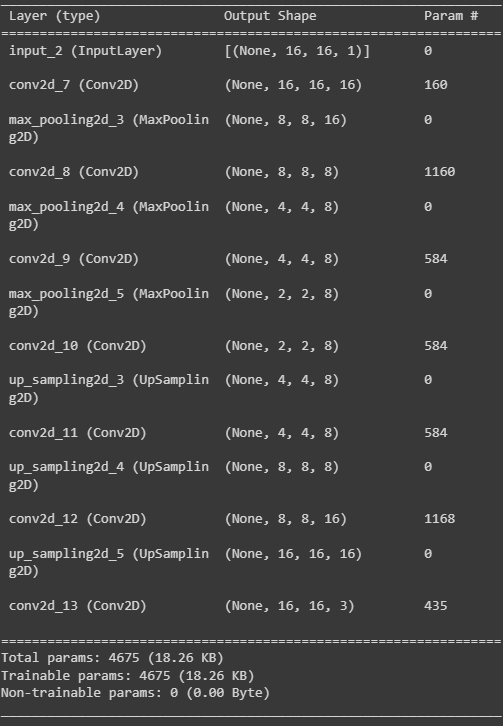


Переводимо у відтінки сірого:



1. Налаштування моделі автоенкодеру:

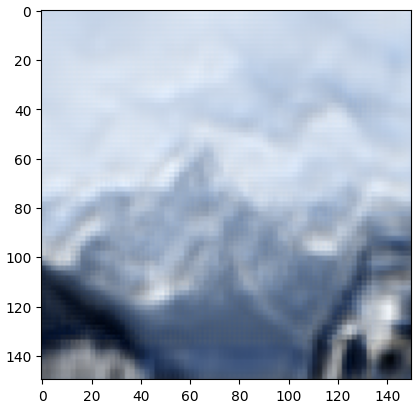
**Кількість епох: 7**



1. Колоризовані тестове та тренувальне зображення

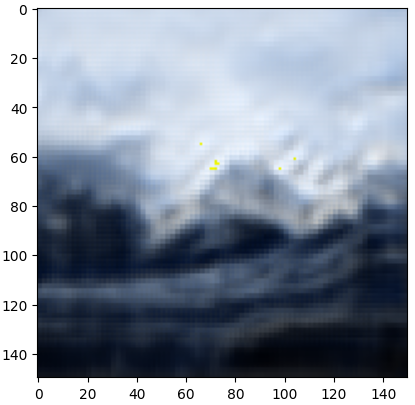
Тренувальне

Похибка: 34153.51

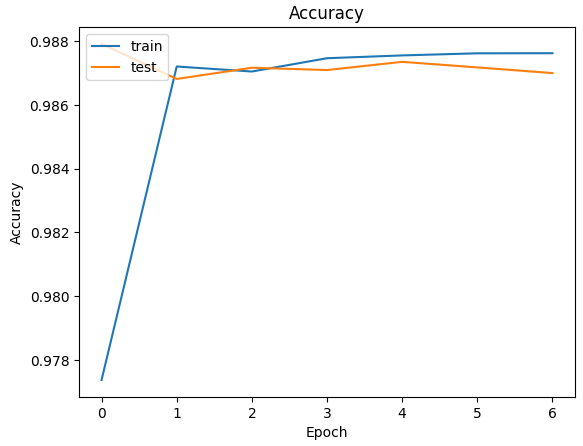


Тестове

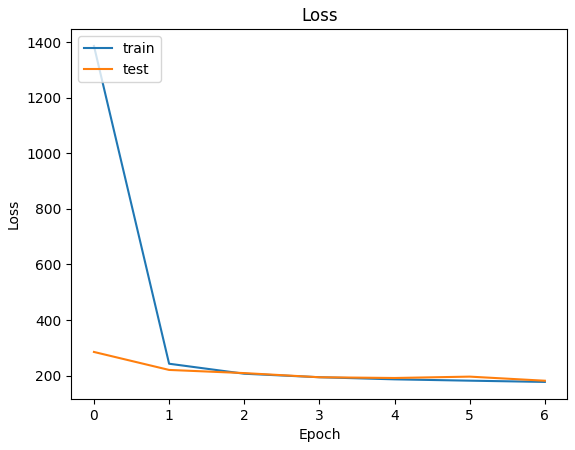
Похибка: 24795.953



4) Графік залежності точності від кількості епох



5) Графік залежності похибки від кількості епох

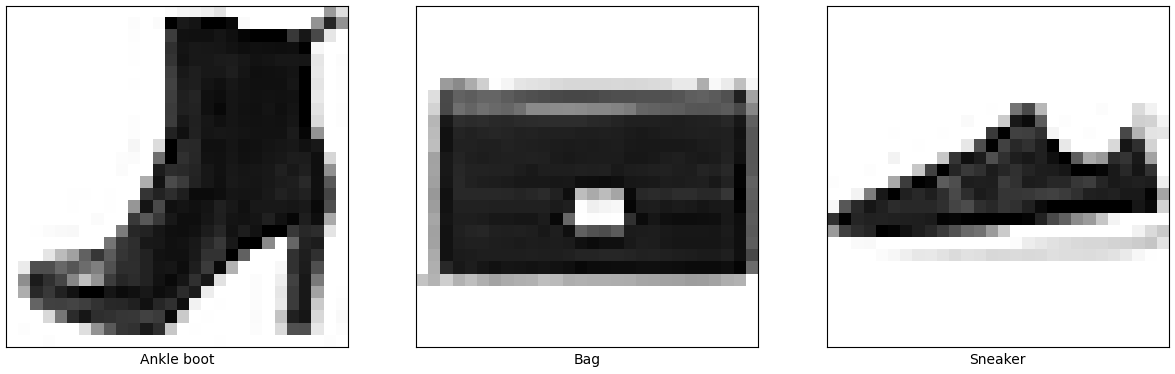


**Завдання 2**

1. Завантажую датасет **Fashion MNIST** та виводжу Зображення для побудови векторів ознак.

Беру **900** зображень

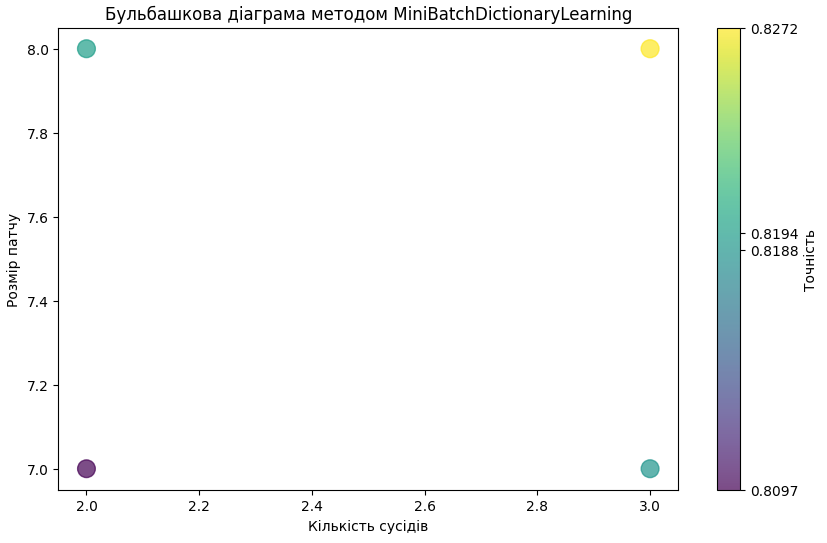
Обрав такі мітки: **Ankle boot, Bag, Sneaker**



Словникове навчання через **MiniBatchDictionaryLearning**

|  |  |  |
| --- | --- | --- |
| Розмір патчів | Кількість сусідів | Точність |
| (7, 7) | 2 | 0.809711 |
| (7, 7) | 3 | 0.818767 |
| (8, 8) | 2 | 0.819413 |
| (8, 8) | 3 | 0.827160 |

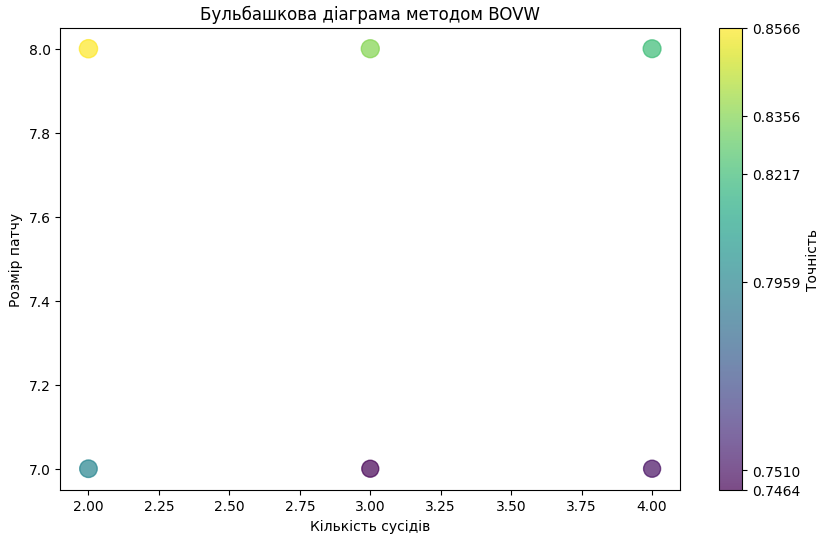
Побудування бульбашкової діаграми



Словникове навчання через **Bag of Visual Words**

|  |  |  |
| --- | --- | --- |
| Розмір патчів | Кількість сусідів | Точність |
| (7, 7) | 2 | 0.795897 |
| (7, 7) | 3 | 0.746362 |
| (7, 7) | 4 | 0.750962 |
| (8, 8) | 2 | 0.856584 |
| (8, 8) | 3 | 0.835619 |
| (8, 8) | 4 | 0.821696 |

Побудування бульбашкової діаграми



**Висновок:** можна зробити висновок що найкращу точність дає саме метод вилучення ознак Bag of Visual Words, але він працює в рази довше, ніж метод Mini Batch Dictionary Learning.

Найкраще значення для *MiniBatchDictionaryLearning*:

розмір патчів: **(8, 8)**

кількість сусідів: **3**

точність: **0.827160**

Найкраще значення для *Bag of Visual Words*:

розмір патчів: **(8, 8)**

кількість сусідів: **2**

точність: **0.856584**

.

**Код програми:**

Завдання 1

**from** skimage.io **import** imread, imsave

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

**import** tensorflow **as** tf

**from** tensorflow **import** keras

**from** sklearn.feature\_extraction.image **import** extract\_patches\_2d, reconstruct\_from\_patches\_2d

**from** sklearn.model\_selection **import** train\_test\_split

**from** tensorflow.keras **import** layers

**from** tensorflow.keras.datasets **import** fashion\_mnist

**from** sklearn.decomposition **import** MiniBatchDictionaryLearning

**from** skimage.feature **import** ORB

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.neighbors **import** KNeighborsClassifier

**import** cv2

train\_picture\_path = '/content/drive/MyDrive/Intelligent\_Analysys\_Labs/Laboratorna6/images/train\_photo.jpg'

test\_picture\_path = '/content/drive/MyDrive/Intelligent\_Analysys\_Labs/Laboratorna6/images/test\_photo.jpg'

image = imread(train\_picture\_path)

image = cv2.resize(image, (300, 300))

image\_gray = imread(train\_picture\_path, as\_gray=True)

image\_gray = cv2.resize(image\_gray, (300, 300))

image\_test\_gray = imread(test\_picture\_path, as\_gray=True)

image\_test\_color = imread(test\_picture\_path)

plt.imshow(image)

plt.axis('off')

plt.title("Train picture")

plt.show()

plt.imshow(image\_gray, cmap=plt.cm.gray)

plt.axis('off')

plt.title("Train picture grayscale")

plt.show()

plt.imshow(image\_test\_color)

plt.axis('off')

plt.title("Test picture")

plt.show()

plt.imshow(image\_test\_gray, cmap=plt.cm.gray)

plt.axis('off')

plt.title("Test picture grayscale")

plt.show()

patch\_size = (16, 16)

color\_patches = extract\_patches\_2d(image, patch\_size)

gray\_patches = extract\_patches\_2d(image\_gray, patch\_size)

color\_patches.shape, gray\_patches.shape

image\_test\_gray = cv2.resize(image\_test\_gray, (150, 150))

patch\_size = (16, 16)

step = 2

image\_height, image\_width = image\_test\_gray.shape

X\_train, X\_test, y\_train, y\_test = train\_test\_split(gray\_patches, color\_patches, test\_size=0.2, random\_state=0)

input\_img = keras.Input(shape=(16, 16, 1))

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input\_img)

x = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)

x = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)

encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)

x = layers.UpSampling2D((2, 2))(x)

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)

x = layers.UpSampling2D((2, 2))(x)

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)

x = layers.UpSampling2D((2, 2))(x)

decoded = layers.Conv2D(3, (3, 3), activation='relu', padding='same')(x)

model = keras.Model(input\_img, decoded)

model.compile(optimizer='adam', loss='mean\_squared\_error',

metrics=['accuracy'])

model.summary()

hist = model.fit(X\_train, y\_train, epochs=7, batch\_size=32, validation\_data=(X\_test, y\_test))

colorized\_image = np.zeros((image\_height, image\_width, 3), dtype=np.uint8)

**for** y **in** range(0, image\_height - patch\_size[0] + 1, step):

**for** x **in** range(0, image\_width - patch\_size[1] + 1, step):

    patch = image\_test\_gray[y:y + patch\_size[0], x:x + patch\_size[1]]

    patch = np.expand\_dims(np.expand\_dims(patch, axis=0), axis=3)

    colorized\_patch = model.predict(patch)

    colorized\_patch = colorized\_patch.astype(np.uint8)

    colorized\_image[y:y + patch\_size[0], x:x + patch\_size[1]] = colorized\_patch[0]

mse = np.mean((image\_test\_gray - cv2.cvtColor(colorized\_image, cv2.COLOR\_BGR2GRAY)) \*\* 2)

print(f"Mean Squared Error (MSE): {mse}")

plt.imshow(colorized\_image)

plt.show()

plt.plot(hist.history['accuracy'])

plt.plot(hist.history['val\_accuracy'])

plt.title('Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.savefig('accuracy.png')

plt.show()

plt.plot(hist.history['loss'])

plt.plot(hist.history['val\_loss'])

plt.title('Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.savefig('loss.png')

plt.show()

image\_test\_gray = cv2.resize(image\_gray, (150, 150))

patch\_size = (16, 16)

step = 2

image\_height, image\_width = image\_test\_gray.shape

colorized\_image = np.zeros((image\_height, image\_width, 3), dtype=np.uint8)

**for** y **in** range(0, image\_height - patch\_size[0] + 1, step):

**for** x **in** range(0, image\_width - patch\_size[1] + 1, step):

    patch = image\_test\_gray[y:y + patch\_size[0], x:x + patch\_size[1]]

    patch = np.expand\_dims(np.expand\_dims(patch, axis=0), axis=3)

    colorized\_patch = model.predict(patch)

    colorized\_patch = colorized\_patch.astype(np.uint8)

    colorized\_image[y:y + patch\_size[0], x:x + patch\_size[1]] = colorized\_patch[0]

mse = np.mean((image\_test\_gray - cv2.cvtColor(colorized\_image, cv2.COLOR\_BGR2GRAY)) \*\* 2)

print(f"Mean Squared Error (MSE): {mse}")

plt.imshow(image\_test\_gray, cmap=plt.cm.gray)

plt.show()

plt.imshow(colorized\_image)

plt.show()

cv2.imwrite('train\_colored.jpg', colorized\_image)

Завдання 2

(X\_train, y\_train), (\_, \_) = fashion\_mnist.load\_data()

selected\_classes = [9, 8, 7]

X\_train\_select = X\_train[np.isin(y\_train, selected\_classes)]

y\_train\_select = y\_train[np.isin(y\_train, selected\_classes)]

X\_train\_shuff, \_, y\_train\_shuff, \_ = train\_test\_split(X\_train\_select, y\_train\_select, test\_size=0.1, random\_state=42, stratify=y\_train\_select)

X\_train\_shuff = X\_train\_shuff[:900]

y\_train\_shuff = y\_train\_shuff[:900]

class\_names = ['Tshirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

**def** **display\_one\_image\_per\_class**(images, labels, class\_names):

    plt.figure(figsize=(15, 5))

**for** i, class\_id **in** enumerate(selected\_classes):

        class\_indices = np.where(labels == class\_id)[0]

        image\_index = class\_indices[0]

        plt.subplot(1, len(selected\_classes), i + 1)

        plt.xticks([])

        plt.yticks([])

        plt.grid(False)

        plt.imshow(images[image\_index], cmap=plt.cm.binary)

        plt.xlabel(class\_names[class\_id])

    plt.show()

display\_one\_image\_per\_class(X\_train\_shuff, y\_train\_shuff, class\_names)

patch\_sizes = [(7, 7), (8, 8)]

orb = ORB()

**def** **orb\_features**(image\_patch):

    orb.detect\_and\_extract(image\_patch)

**return** orb.descriptors

data = []

**for** patch\_size **in** patch\_sizes:

    patches\_all = []

    labels = []

**for** \_, (img, label) **in** enumerate(zip(X\_train\_shuff, y\_train\_shuff)):

        patches = extract\_patches\_2d(img, patch\_size)

        patches = patches.reshape(patches.shape[0], -1).astype("float32")

        c\_mean = np.mean(patches, axis=0)

        c\_std = np.std(patches, axis=0)

        patches -= c\_mean

        patches /= c\_std

        patches\_all.extend(patches)

        labels.extend([label] \* len(patches))

    dictionary = MiniBatchDictionaryLearning(

        n\_components=100, batch\_size=512, max\_iter=10, random\_state=42

    )

    sparsecode = dictionary.fit\_transform(patches\_all)

    patches\_train, patches\_test, labels\_train, labels\_test = train\_test\_split(

        sparsecode, labels, test\_size=0.2, random\_state=42

    )

**for** k **in** range(2, 4):

        knn = KNeighborsClassifier(n\_neighbors=k)

        knn.fit(patches\_train, labels\_train)

        predictions = knn.predict(patches\_test)

        score = accuracy\_score(labels\_test, predictions)

        data.append((patch\_size[0], k, score))

mini\_batch\_df = pd.DataFrame(data, columns=["patch\_size", "n\_neighbors", "accuracy"])

mini\_batch\_df

x\_values = mini\_batch\_df["n\_neighbors"]

y\_values = mini\_batch\_df["patch\_size"]

sizes = mini\_batch\_df["accuracy"] \* 200

colors = mini\_batch\_df["accuracy"]

plt.figure(figsize=(10, 6))

scatter = plt.scatter(x\_values, y\_values, s=sizes, c=colors, cmap="viridis", alpha=0.7)

plt.title("Бульбашкова діаграма методом MiniBatchDictionaryLearning")

plt.xlabel("Кількість сусідів")

plt.ylabel("Розмір патчу")

cbar = plt.colorbar(scatter, label="Точність")

cbar.set\_ticks(colors)

cbar.set\_ticklabels(["{:.4f}".format(val) **for** val **in** colors])

plt.show()

data\_bovw = []

**for** patch\_size **in** patch\_sizes:

    descriptors = []

    labels = []

**for** \_, (img, label) **in** enumerate(zip(X\_train\_shuff, y\_train\_shuff)):

        patches = extract\_patches\_2d(img, patch\_size)

        patches = patches.reshape(patches.shape[0], -1).astype("float32")

        c\_mean = np.mean(patches, axis=0)

        c\_std = np.std(patches, axis=0)

        patches -= c\_mean

        patches /= c\_std

        des = orb\_features(patches)

        descriptors.extend(des)

        labels.extend([label] \* len(des))

    descriptors\_train, descriptors\_test, labels\_train, labels\_test = train\_test\_split(

        descriptors, labels, test\_size=0.2, random\_state=42

    )

**for** k **in** range(2, 5):

        knn = KNeighborsClassifier(n\_neighbors=k)

        knn.fit(descriptors\_train, labels\_train)

        predictions = knn.predict(descriptors\_test)

        accuracy = accuracy\_score(labels\_test, predictions)

        data\_bovw.append((patch\_size[0], k, accuracy))

bovw\_batch\_df = pd.DataFrame(

    data\_bovw, columns=["patch\_size", "n\_neighbors", "accuracy"]

)

bovw\_batch\_df

x\_values = bovw\_batch\_df["n\_neighbors"]

y\_values = bovw\_batch\_df["patch\_size"]

sizes = bovw\_batch\_df["accuracy"] \* 200

colors = bovw\_batch\_df["accuracy"]

plt.figure(figsize=(10, 6))

scatter = plt.scatter(x\_values, y\_values, s=sizes, c=colors, cmap="viridis", alpha=0.7)

plt.title("Бульбашкова діаграма методом BOVW")

plt.xlabel("Кількість сусідів")

plt.ylabel("Розмір патчу")

cbar = plt.colorbar(scatter, label="Точність")

cbar.set\_ticks(colors)

cbar.set\_ticklabels(["{:.4f}".format(val) **for** val **in** colors])

plt.show()