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 алгоритмів інтелектуального аналізу даних для обробки та аналізу цифрових зображень.

Завдання на роботу: колоризація зображень у відтінках сірого через автокодувальник. Навчання ознак для розв’язання задачі семантичної роз мітки зображень.

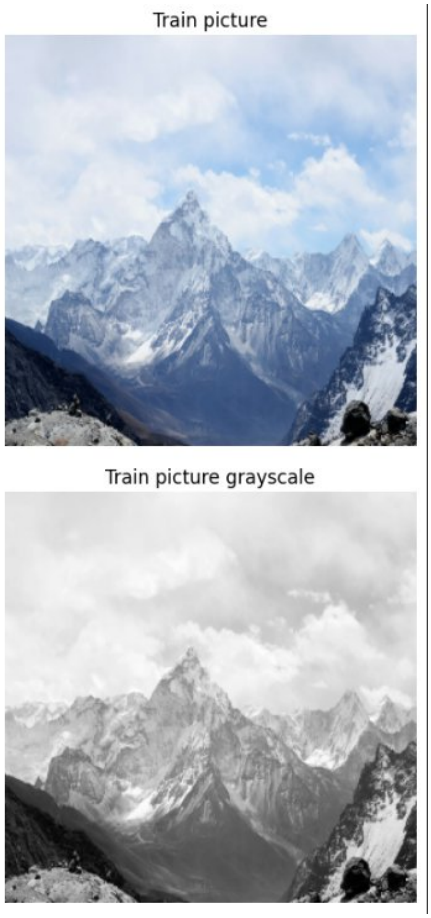
Завдання 1

1) Обрані картинки для колоризації:

Для тренування

Для тесту

Переводимо у відтінки сірого:



Переводимо у відтінки сірого:

2) Налаштування моделі автоенкодеру:

Кількість епох: 7

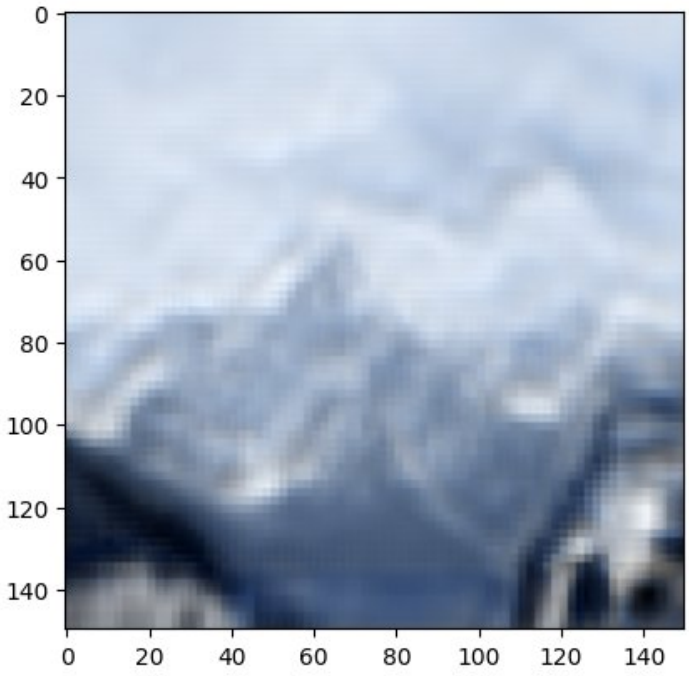
Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 16, 16, 1)]	0
conv2d_7 (Conv2D)	(None, 16, 16, 16)	160
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 16)	0
conv2d_8 (Conv2D)	(None, 8, 8, 8)	1160
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 8)	0
conv2d_9 (Conv2D)	(None, 4, 4, 8)	584
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 8)	0
conv2d_10 (Conv2D)	(None, 2, 2, 8)	584
up_sampling2d_3 (UpSampling2D)	(None, 4, 4, 8)	0
conv2d_11 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_4 (UpSampling2D)	(None, 8, 8, 8)	0
conv2d_12 (Conv2D)	(None, 8, 8, 16)	1168
up_sampling2d_5 (UpSampling2D)	(None, 16, 16, 16)	0
conv2d_13 (Conv2D)	(None, 16, 16, 3)	435

Total params: 4675 (18.26 KB)
Trainable params: 4675 (18.26 KB)
Non-trainable params: 0 (0.00 Byte)

3) Колоризовані тестове та тренувальне зображення

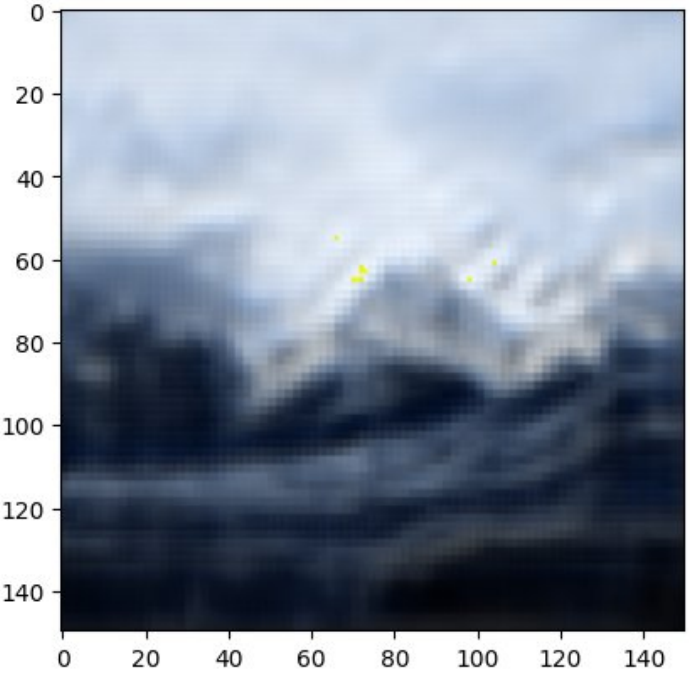
Тренувальне

Похибка: 34153.51

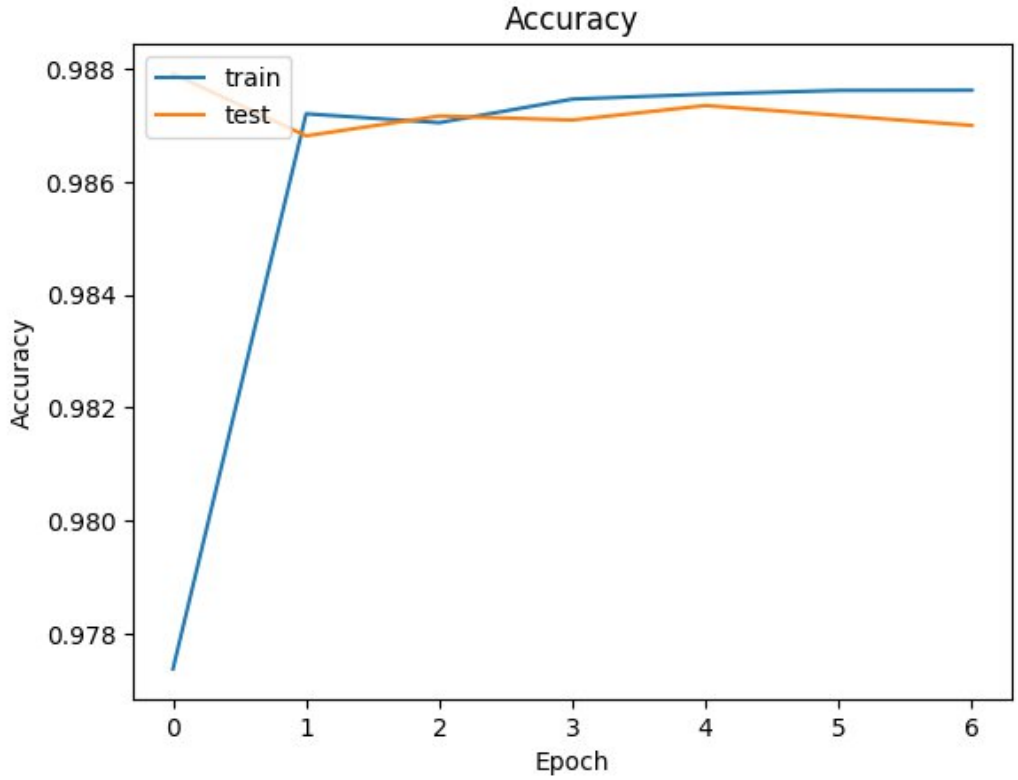


Тестове

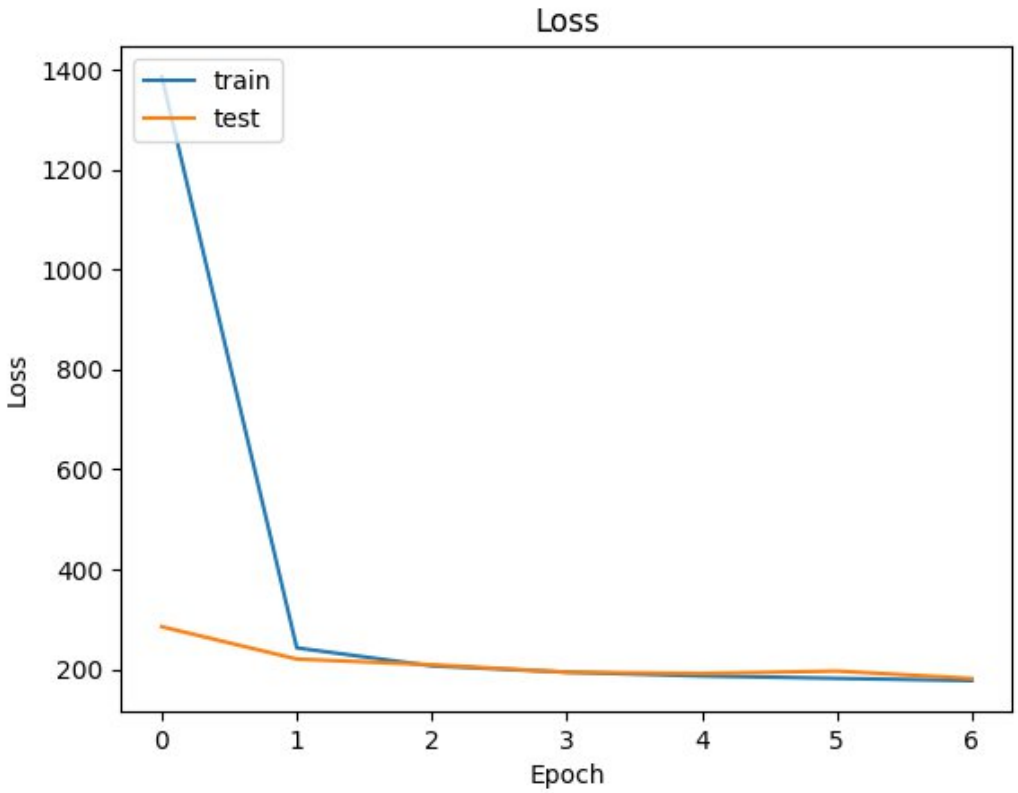
Похибка: 24795.953



4) Графік залежності точності від кількості епох



5) Графік залежності похибки від кількості епох

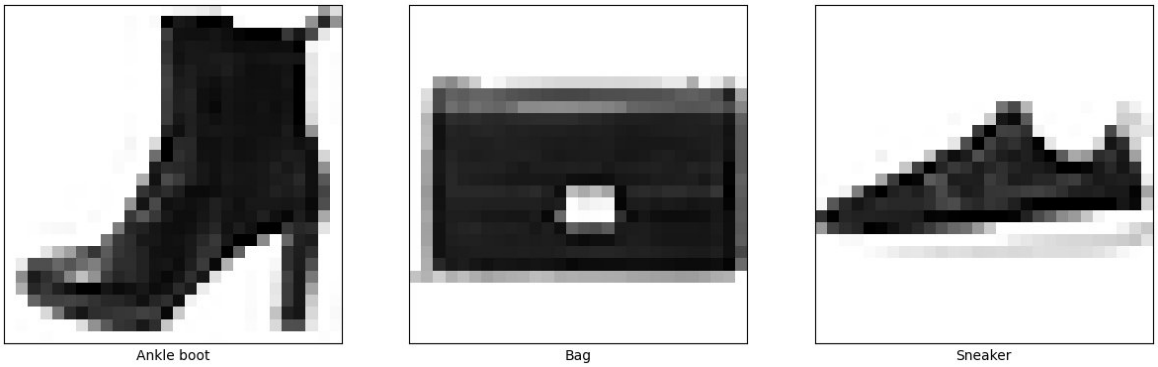


Завдання 2

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

Беру **900** зображень

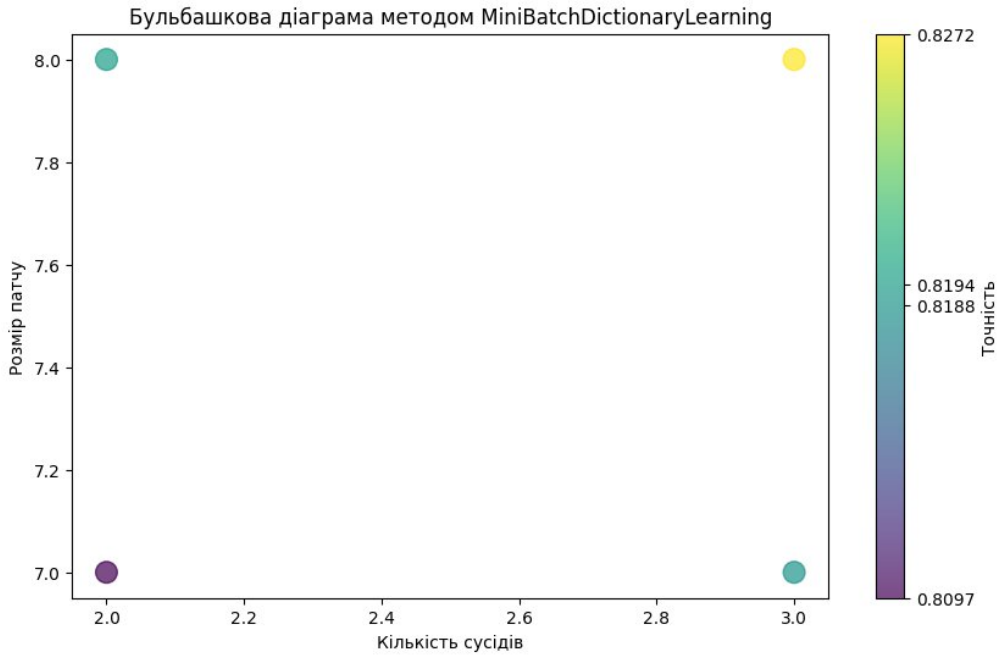
Обрав такі мітки: **Ankle boot, Bag, Sneaker**



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

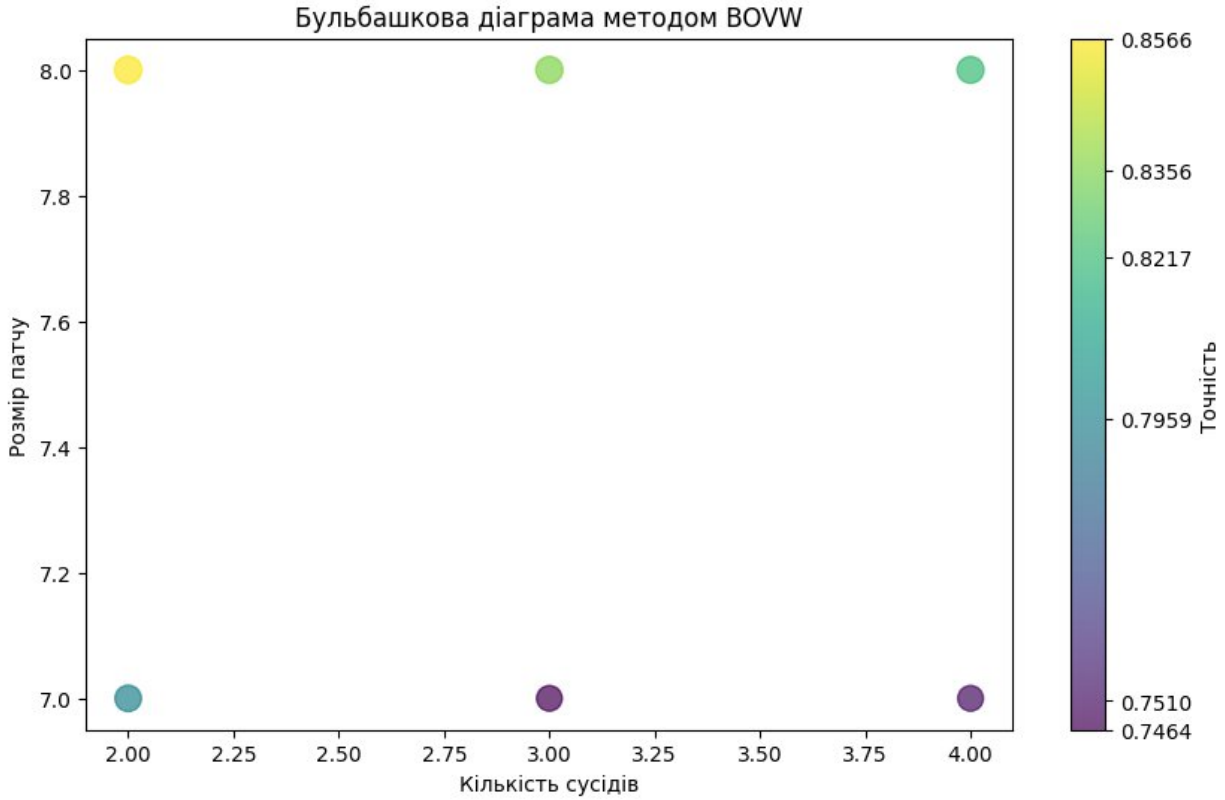
Побудування бульбашкової діаграми



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()
```