МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

**КОМП’ЮТЕРНИЙ ПРАКТИКУМ 3.**

**ВИКОРИСТАННЯ ЗГОРТКОВОЇ  
НЕЙРОННОЇ МЕРЕЖІ CNN ДЛЯ ЗАДАЧІ КЛАСИФІКАЦІЇ  
ЗОБРАЖЕНЬ**

Підготував: студент групи ПДМ-51

Гапей Максим Юрійович

Перевірив: викладач

Садовенко Володимир Сергійович

Київ 2024

**Мета роботи:** Потрібно використати бібліотеки мови програмування Python для завантаження, дослідження та аналізу набору даних Fashion-MNIST для задачі класифікації зображень. Після цього потрібно провести попередню обробку даних: змінити розміри, масштабувати, перетворити мітки у вектори кодування та розділити дані на навчальні та тестові набори. Зробивши все це, потрібно побудувати модель нейронної мережі CNN. Далі потрібно компілювати, навчити та оцінити побудовану модель, візуалізуючи точність і графіки втрат.

Завантажте дані:

from keras.datasets import fashion\_mnist

(train\_X,train\_Y), (test\_X,test\_Y) = fashion\_mnist.load\_data()

Проаналізуйте дані:

import numpy as np

from keras.utils import to\_categorical

import matplotlib.pyplot as plt

%matplotlib inline

print('Training data shape : ', train\_X.shape, train\_Y.shape)

print('Testing data shape : ', test\_X.shape, test\_Y.shape)

*Який результат ви отримали?*Training data shape : (60000, 28, 28) (60000,)

Testing data shape : (10000, 28, 28) (10000,)

*Які зображення є даними для навчання та які є тестовими даними?*

60000 — для навчання, 10000 — для тестування

*Який розмір навчальних та тестових зразків?*

28x28

Знайдіть унікальні номери міток тренувального набору:

classes = np.unique(train\_Y)

nClasses = len(classes)

print('Total number of outputs : ', nClasses)

print('Output classes : ', classes)

*Який результат ви отримали?*

Total number of outputs : 10

Output classes : [0 1 2 3 4 5 6 7 8 9]

Прогляньте зображення у наборі даних:

plt.figure(figsize=[5,5])

plt.subplot(121)

plt.imshow(train\_X[0,:,:], cmap='gray')

plt.title("Ground Truth : {}".format(train\_Y[0]))

plt.subplot(122)

plt.imshow(test\_X[0,:,:], cmap='gray')

plt.title("Ground Truth : {}".format(test\_Y[0]))

Попередня обробка даних:

train\_X = train\_X.reshape(-1, 28,28, 1)

test\_X = test\_X.reshape(-1, 28,28, 1)

train\_X.shape, test\_X.shape

*Який результат ви отримали?*

(60000, 28, 28, 1) (10000, 28, 28, 1)

Дані зараз у форматі int8, тому перед подачею їх у мережу вам потрібно

перетворити їх тип на float32, а також змінити масштаб значень пікселів у

діапазоні від 0 до 1 включно:

train\_X = train\_X.astype('float32')

test\_X = test\_X.astype('float32')

train\_X = train\_X / 255.

test\_X = test\_X / 255.

Перетворіть навчальні та тестові мітки у вектори кодування:

train\_Y\_one\_hot = to\_categorical(train\_Y)

test\_Y\_one\_hot = to\_categorical(test\_Y)

print('Original label:', train\_Y[0])

print('After conversion to one-hot:', train\_Y\_one\_hot[0])

*Який результат ви отримали?*

Original label: 9   
After conversion to one-hot: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]

Щоб модель добре узагальнювала дані, ви повинні розділит навчальні дані на дві частини, одну призначену для навчання, а іншу для перевірки. У цьому випадку ви навчатимете модель на 80% навчальних даних і перевіряєте її на 20% решти дани:

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

train\_X,valid\_X,train\_label,valid\_label = train\_test\_split(train\_X,

train\_Y\_one\_hot, test\_size=0.2, random\_state=13)

Перевірте форму набору для навчання та перевірки:

train\_X.shape,valid\_X.shape,train\_label.shape,valid\_label.shape

*Який результат ви отримали?*

(48000, 28, 28, 1) (12000, 28, 28, 1) (48000, 10) (12000, 10)

Нейронна мережа. Змоделюйте дані:

from keras.models import \*

from keras.layers import \*

Навчайте мережу протягом 20 епох:

batch\_size = 64

epochs = 20

num\_classes = 10

Архітектура нейронної мереж:

fashion\_model = Sequential()

fashion\_model.add(Conv2D(32, kernel\_size=(3,3),

activation='linear',input\_shape=(28,28,1),padding='same'))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(MaxPooling2D((2, 2),padding='same'))

fashion\_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(MaxPooling2D(pool\_size=(2, 2),padding='same'))

fashion\_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(MaxPooling2D(pool\_size=(2, 2),padding='same'))

fashion\_model.add(Flatten())

fashion\_model.add(Dense(128, activation='linear'))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(Dense(num\_classes, activation='softmax'))

Компіляція моделі. Вкажіть метрику як точність, яку хочете проаналізувати під час навчання моделі:

fashion\_model.compile(loss=tf.keras.losses.categorical\_crossentropy,

optimizer=tf.keras.optimizers.Adam(),metrics=['accuracy'])

Вконайте візуалізацію шарів, які було створено на вищезгаданому кроці за

допомогою функції підсумовування:

fashion\_model.summary()

*Який результат ви отримали?*

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 28, 28, 32) 320

leaky\_re\_lu (LeakyReLU) (None, 28, 28, 32) 0

max\_pooling2d (MaxPooling2D (None, 14, 14, 32) 0

)

conv2d\_1 (Conv2D) (None, 14, 14, 64) 18496

leaky\_re\_lu\_1 (LeakyReLU) (None, 14, 14, 64) 0

max\_pooling2d\_1 (MaxPooling (None, 7, 7, 64) 0

2D)

conv2d\_2 (Conv2D) (None, 7, 7, 128) 73856

leaky\_re\_lu\_2 (LeakyReLU) (None, 7, 7, 128) 0

max\_pooling2d\_2 (MaxPooling (None, 4, 4, 128) 0

2D)

flatten (Flatten) (None, 2048) 0

dense (Dense) (None, 128) 262272

leaky\_re\_lu\_3 (LeakyReLU) (None, 128) 0

dense\_1 (Dense) (None, 10) 1290

=================================================================

Total params: 356,234

Trainable params: 356,234

Non-trainable params: 0

Тренування моделі:

fashion\_train = fashion\_model.fit(train\_X, train\_label,

batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(valid\_X, valid\_label))

Оцінка моделі на тестовому наборі:

test\_eval = fashion\_model.evaluate(test\_X, test\_Y\_one\_hot, verbose=0)

print('Test loss:', test\_eval[0])

print('Test accuracy:', test\_eval[1])

*Який результат ви отримали?*

Epoch 1/20   
750/750 [==============================] - 230s 304ms/step - loss: 0.4637 - accuracy: 0.8324 - val\_loss: 0.3470 - val\_accuracy: 0.8693   
Epoch 2/20   
750/750 [==============================] - 217s 289ms/step - loss: 0.2905 - accuracy: 0.8925 - val\_loss: 0.2824 - val\_accuracy: 0.8990   
Epoch 3/20   
750/750 [==============================] - 208s 278ms/step - loss: 0.2412 - accuracy: 0.9110 - val\_loss: 0.2585 - val\_accuracy: 0.9057   
Epoch 4/20   
750/750 [==============================] - 184s 246ms/step - loss: 0.2108 - accuracy: 0.9221 - val\_loss: 0.2371 - val\_accuracy: 0.9133   
Epoch 5/20   
750/750 [==============================] - 182s 243ms/step - loss: 0.1860 - accuracy: 0.9300 - val\_loss: 0.2404 - val\_accuracy: 0.9140   
Epoch 6/20   
750/750 [==============================] - 177s 236ms/step - loss: 0.1603 - accuracy: 0.9392 - val\_loss: 0.2299 - val\_accuracy: 0.9189   
Epoch 7/20   
750/750 [==============================] - 175s 234ms/step - loss: 0.1381 - accuracy: 0.9485 - val\_loss: 0.2377 - val\_accuracy: 0.9209   
Epoch 8/20   
750/750 [==============================] - 175s 234ms/step - loss: 0.1205 - accuracy: 0.9548 - val\_loss: 0.2402 - val\_accuracy: 0.9194   
Epoch 9/20   
750/750 [==============================] - 175s 233ms/step - loss: 0.1024 - accuracy: 0.9618 - val\_loss: 0.2585 - val\_accuracy: 0.9194   
Epoch 10/20   
750/750 [==============================] - 176s 234ms/step - loss: 0.0869 - accuracy: 0.9682 - val\_loss: 0.2567 - val\_accuracy: 0.9216   
Epoch 11/20   
750/750 [==============================] - 203s 271ms/step - loss: 0.0758 - accuracy: 0.9712 - val\_loss: 0.2994 - val\_accuracy: 0.9181   
Epoch 12/20   
750/750 [==============================] - 296s 394ms/step - loss: 0.0632 - accuracy: 0.9765 - val\_loss: 0.3168 - val\_accuracy: 0.9202   
Epoch 13/20   
750/750 [==============================] - 204s 272ms/step - loss: 0.0524 - accuracy: 0.9806 - val\_loss: 0.3344 - val\_accuracy: 0.9196   
Epoch 14/20   
750/750 [==============================] - 193s 258ms/step - loss: 0.0486 - accuracy: 0.9813 - val\_loss: 0.3749 - val\_accuracy: 0.9227   
Epoch 15/20   
750/750 [==============================] - 182s 243ms/step - loss: 0.0438 - accuracy: 0.9838 - val\_loss: 0.3851 - val\_accuracy: 0.9138   
Epoch 16/20   
750/750 [==============================] - 182s 242ms/step - loss: 0.0392 - accuracy: 0.9855 - val\_loss: 0.3829 - val\_accuracy: 0.9193   
Epoch 17/20   
750/750 [==============================] - 184s 245ms/step - loss: 0.0333 - accuracy: 0.9884 - val\_loss: 0.4058 - val\_accuracy: 0.9226   
Epoch 18/20   
750/750 [==============================] - 183s 244ms/step - loss: 0.0311 - accuracy: 0.9884 - val\_loss: 0.4364 - val\_accuracy: 0.9195   
Epoch 19/20   
750/750 [==============================] - 184s 245ms/step - loss: 0.0330 - accuracy: 0.9887 - val\_loss: 0.4256 - val\_accuracy: 0.9139   
Epoch 20/20   
750/750 [==============================] - 186s 248ms/step - loss: 0.0308 - accuracy: 0.9887 - val\_loss: 0.4978 - val\_accuracy: 0.9094   
Test loss: 0.5175475478172302   
Test accuracy: 0.9101999998092651

*Чи ці результати хороші?*

Ні, модель перенавчена, оскільки дає погані результати на тестовій виборці.

Розгляньте модель оцінки в перспективі та побудуйте графіки точності та втрат між даними навчання та перевірки:

val\_accuracy = fashion\_train.history['val\_accuracy']

loss = fashion\_train.history['loss']

val\_loss = fashion\_train.history['val\_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training accuracy')

plt.plot(epochs, val\_accuracy, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

Додавання Dropout до нейронної мережі. Знову створіть, компілюйте та навчайте мережу, але цього разу з використанням Dropout. І запустіть його протягом 20 епох із розміром партії 64:

batch\_size = 64

epochs = 20

num\_classes = 10

fashion\_model = Sequential()

fashion\_model.add(Conv2D(32, kernel\_size=(3, 3),activation='linear',padding='same',input\_shape=(28,28,1)))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(MaxPooling2D((2, 2),padding='same'))

fashion\_model.add(Dropout(0.25))

fashion\_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(MaxPooling2D(pool\_size=(2, 2),padding='same'))

fashion\_model.add(Dropout(0.25))

fashion\_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(MaxPooling2D(pool\_size=(2, 2),padding='same'))

fashion\_model.add(Dropout(0.4))

fashion\_model.add(Flatten())

fashion\_model.add(Dense(128, activation='linear'))

fashion\_model.add(LeakyReLU(alpha=0.1))

fashion\_model.add(Dropout(0.3))

fashion\_model.add(Dense(num\_classes, activation='softmax'))

fashion\_model.summary()

*Який результат ви отримали?*

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 28, 28, 32) 320

leaky\_re\_lu (LeakyReLU) (None, 28, 28, 32) 0

max\_pooling2d (MaxPooling2D (None, 14, 14, 32) 0

)

dropout (Dropout) (None, 14, 14, 32) 0

conv2d\_1 (Conv2D) (None, 14, 14, 64) 18496

leaky\_re\_lu\_1 (LeakyReLU) (None, 14, 14, 64) 0

max\_pooling2d\_1 (MaxPooling (None, 7, 7, 64) 0

2D)

dropout\_1 (Dropout) (None, 7, 7, 64) 0

conv2d\_2 (Conv2D) (None, 7, 7, 128) 73856

leaky\_re\_lu\_2 (LeakyReLU) (None, 7, 7, 128) 0

max\_pooling2d\_2 (MaxPooling (None, 4, 4, 128) 0

2D)

dropout\_2 (Dropout) (None, 4, 4, 128) 0

flatten (Flatten) (None, 2048) 0

dense (Dense) (None, 128) 262272

leaky\_re\_lu\_3 (LeakyReLU) (None, 128) 0

dropout\_3 (Dropout) (None, 128) 0

dense\_1 (Dense) (None, 10) 1290

=================================================================

Total params: 356,234

Trainable params: 356,234

Non-trainable params: 0

fashion\_model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

fashion\_train\_dropout = fashion\_model.fit(train\_X, train\_label,

batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(valid\_X,

valid\_label))

*Який результат ви отримали?*

Epoch 1/20   
750/750 [==============================] - 233s 308ms/step - loss: 0.6063 - accuracy: 0.7754 - val\_loss: 0.3727 - val\_accuracy: 0.8602   
Epoch 2/20   
750/750 [==============================] - 231s 308ms/step - loss: 0.3783 - accuracy: 0.8609 - val\_loss: 0.3069 - val\_accuracy: 0.8862   
Epoch 3/20   
750/750 [==============================] - 223s 297ms/step - loss: 0.3327 - accuracy: 0.8778 - val\_loss: 0.2824 - val\_accuracy: 0.8942   
Epoch 4/20   
750/750 [==============================] - 223s 297ms/step - loss: 0.3022 - accuracy: 0.8879 - val\_loss: 0.2654 - val\_accuracy: 0.9033   
Epoch 5/20   
750/750 [==============================] - 225s 300ms/step - loss: 0.2849 - accuracy: 0.8933 - val\_loss: 0.2654 - val\_accuracy: 0.9023   
Epoch 6/20   
750/750 [==============================] - 236s 315ms/step - loss: 0.2704 - accuracy: 0.8991 - val\_loss: 0.2397 - val\_accuracy: 0.9105   
Epoch 7/20   
750/750 [==============================] - 225s 300ms/step - loss: 0.2547 - accuracy: 0.9044 - val\_loss: 0.2324 - val\_accuracy: 0.9118   
Epoch 8/20   
750/750 [==============================] - 218s 291ms/step - loss: 0.2505 - accuracy: 0.9059 - val\_loss: 0.2404 - val\_accuracy: 0.9106   
Epoch 9/20   
750/750 [==============================] - 192s 256ms/step - loss: 0.2412 - accuracy: 0.9103 - val\_loss: 0.2250 - val\_accuracy: 0.9159   
Epoch 10/20   
750/750 [==============================] - 224s 298ms/step - loss: 0.2332 - accuracy: 0.9128 - val\_loss: 0.2202 - val\_accuracy: 0.9187   
Epoch 11/20   
750/750 [==============================] - 224s 298ms/step - loss: 0.2300 - accuracy: 0.9129 - val\_loss: 0.2202 - val\_accuracy: 0.9178   
Epoch 12/20   
750/750 [==============================] - 227s 303ms/step - loss: 0.2256 - accuracy: 0.9168 - val\_loss: 0.2194 - val\_accuracy: 0.9201   
Epoch 13/20   
750/750 [==============================] - 224s 299ms/step - loss: 0.2195 - accuracy: 0.9177 - val\_loss: 0.2195 - val\_accuracy: 0.9204   
Epoch 14/20   
750/750 [==============================] - 226s 302ms/step - loss: 0.2142 - accuracy: 0.9187 - val\_loss: 0.2203 - val\_accuracy: 0.9217   
Epoch 15/20   
750/750 [==============================] - 224s 299ms/step - loss: 0.2102 - accuracy: 0.9202 - val\_loss: 0.2287 - val\_accuracy: 0.9153   
Epoch 16/20   
750/750 [==============================] - 229s 305ms/step - loss: 0.2108 - accuracy: 0.9200 - val\_loss: 0.2348 - val\_accuracy: 0.9167   
Epoch 17/20   
750/750 [==============================] - 229s 305ms/step - loss: 0.2050 - accuracy: 0.9228 - val\_loss: 0.2125 - val\_accuracy: 0.9247   
Epoch 18/20   
750/750 [==============================] - 208s 277ms/step - loss: 0.2048 - accuracy: 0.9223 - val\_loss: 0.2097 - val\_accuracy: 0.9255   
Epoch 19/20   
750/750 [==============================] - 217s 290ms/step - loss: 0.2017 - accuracy: 0.9251 - val\_loss: 0.2152 - val\_accuracy: 0.9227   
Epoch 20/20   
750/750 [==============================] - 230s 307ms/step - loss: 0.1987 - accuracy: 0.9247 - val\_loss: 0.2147 - val\_accuracy: 0.9231

Зауважте, що ви також можете зберігати модель після кожної епохи, щоб у випадку, якщо виникла якась проблема, яка зупиняє навчання в епоху, вам не довелось починати навчання з початку:

fashion\_model.save("fashion\_model\_dropout.h5py")

Оцінка моделі на тестовому наборі. Оцініть нову модель і подивіться, як вона працює:

test\_eval = fashion\_model.evaluate(test\_X, test\_Y\_one\_hot, verbose=1)

print('Test loss:', test\_eval[0])

print('Test accuracy:', test\_eval[1])

*Який результат ви отримали?*

313/313 [==============================] - 15s 49ms/step - loss: 0.2253 - accuracy: 0.9188   
Test loss: 0.22531403601169586   
Test accuracy: 0.9187999963760376

Додавання Dropout в модель повинно спрацювати, хоча точність тесту значно не покращилася, але втрата тесту зменшилася порівняно з попередніми результатами. В останній раз побудуйте графіки точності та втрат між даними навчання та перевірки:

accuracy = fashion\_train\_dropout.history['accuracy']

val\_accuracy = fashion\_train\_dropout.history['val\_accuracy']

loss = fashion\_train\_dropout.history['loss']

val\_loss = fashion\_train\_dropout.history['val\_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training accuracy')

plt.plot(epochs, val\_accuracy, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

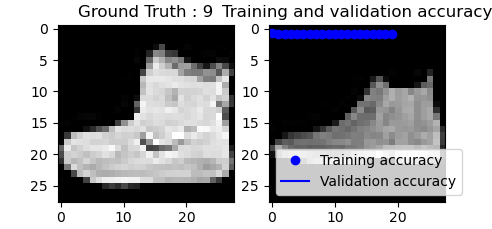
plt.plot(epochs, val\_loss, 'b', label='Validation loss')

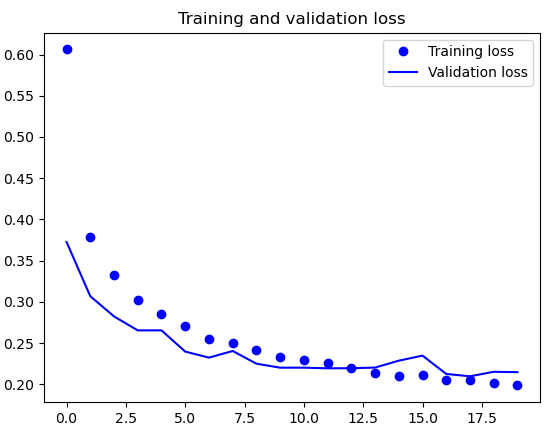
plt.title('Training and validation loss')

plt.legend()

plt.show()

*Який результат ви отримали?*

**



Прогнозування міток:

predicted\_classes = fashion\_model.predict(test\_X)

predicted\_classes = np.argmax(np.round(predicted\_classes),axis=1)

predicted\_classes.shape, test\_Y.shape

*Який результат ви отримали?*

313/313 [==============================] - 16s 51ms/step   
(10000,) (10000,)

correct = np.where(predicted\_classes==test\_Y)[0]

print("Found " + str(len(correct)) + " correct labels")

for i, correct in enumerate(correct[:9]):

plt.subplot(3,3,i+1)

plt.imshow(test\_X[correct].reshape(28,28), cmap='gray', interpolation='none')

plt.title("Predicted {}, Class {}".format(predicted\_classes[correct],

test\_Y[correct]))

plt.tight\_layout()

*Який результат ви отримали?*

Found 9147 correct labels

incorrect = np.where(predicted\_classes!=test\_Y)[0]

print("Found " + str(len(incorrect)) + " incorrect labels")

for i, incorrect in enumerate(incorrect[:9]):

plt.subplot(3,3,i+1)

plt.imshow(test\_X[incorrect].reshape(28,28), cmap='gray', interpolation='none')

plt.title("Predicted {}, Class {}".format(predicted\_classes[incorrect], t est\_Y[incorrect]))

plt.tight\_layout()

*Який результат ви отримали?*

Found 853 incorrect labels

Звіт про класифікацію:

from sklearn.metrics import classification\_report

target\_names = ["Class {}".format(i) for i in range(num\_classes)]

print(classification\_report(test\_Y, predicted\_classes, target\_names=target\_names))

*Поясніть отриманий результат.*

precision recall f1-score support

Class 0 0.75 0.93 0.83 1000

Class 1 0.99 0.98 0.99 1000

Class 2 0.86 0.90 0.88 1000

Class 3 0.93 0.92 0.92 1000

Class 4 0.88 0.86 0.87 1000

Class 5 0.99 0.97 0.98 1000

Class 6 0.85 0.66 0.74 1000

Class 7 0.95 0.99 0.97 1000

Class 8 0.99 0.99 0.99 1000

Class 9 0.98 0.96 0.97 1000

accuracy 0.91 10000

macro avg 0.92 0.91 0.91 10000

weighted avg 0.92 0.91 0.91 10000

Для класів 4 та 6 класифікатору не вистачає точності та запам’ятовування. Для класу 0 і класу 2 класифікатору не вистачає лише точності.

**Висновки:** Під час виконання практичної роботи, було створено, зкомпільовано та навчено модель нейронної мережі CNN, а також та оцінено візуалізуючи точність і графіки втрат.