РОЗРОБКА АВТОЕНКОДЕРУ МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №8**

з дисципліни «**Нейронні мережі**»

# на тему: «РОЗРОБКА АВТОЕНКОДЕРУ»

**Виконав:**

студент гр. БС-03

Затуловський Г. А.

**Перевірив:**

ас. каф. БМК Дюмін О.Д.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

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

(підпис викладача)

Київ-2023

# Варіант 6

# Практична частина

1. Створити автоенкодер для датасету зображень відповідно до варіанту завдань.
2. Дослідити вплив та порівняти результати роботи різних оптимізаторів (Adam, SGD, RMSprop, функцій активації (relu, elu, sigmoid, tanh, і т.д.).
3. Спробувати іншу архітектуру базової мережі, збільшити або зменшити кількість шарів, розміри, тощо (згорткових та пулінгових, і т.д.).
4. Аналіз результатів: Візуалізувати криві навчання та втрат, розрахувати метрики, проаналізувати результати, надати рекомендації щодо можливих покращень моделі.
5. Вимоги до звіту: опис використовуваного датасету, його особливостей та завдань, візуалізація попередньої обробки даних, детальний опис структури розробленої моделі, графіки навчання та втрат, висновки та рекомендації.

**Варіант завдання:**

[3, **6**, 9, 12, 15, 18, 21, 24] KMNIST (Kuzushiji-MNIST):

Датасет історичних японських рукописних символів.

70,000 зображень у 10 категоріях.

Зображення мають розмір 28x28 пікселів, монохромні.

Використовуйте це посилання та відповідний репозиторій (посилання на код додане) для завантаження даних: http://codh.rois.ac.jp/kmnist/, https://github.com/rois-codh/kmnist/blob/master/download\_data.py, при завантаженні треба обрати необхідний набір даних. Після завантаження можна користуватись даними наступним чином попередньо встановивши idx2numpy:

*import gzip import idx2numpy*

*def load\_kmnist():*

*with gzip.open('path\_to\_downloaded\_files/train-images-idx3-ubyte.gz', 'r') as f:*

*x\_train = idx2numpy.convert\_from\_file(f) with gzip.open('path\_to\_downloaded\_files/train-labels-idx1-ubyte.gz', 'r') as f:*

*y\_train = idx2numpy.convert\_from\_file(f) with gzip.open('path\_to\_downloaded\_files/t10k-images-idx3-ubyte.gz', 'r') as f:*

*x\_test = idx2numpy.convert\_from\_file(f) with gzip.open('path\_to\_downloaded\_files/t10k-labels-idx1-ubyte.gz', 'r') as f:*

*y\_test = idx2numpy.convert\_from\_file(f) return (x\_train, y\_train), (x\_test, y\_test)*

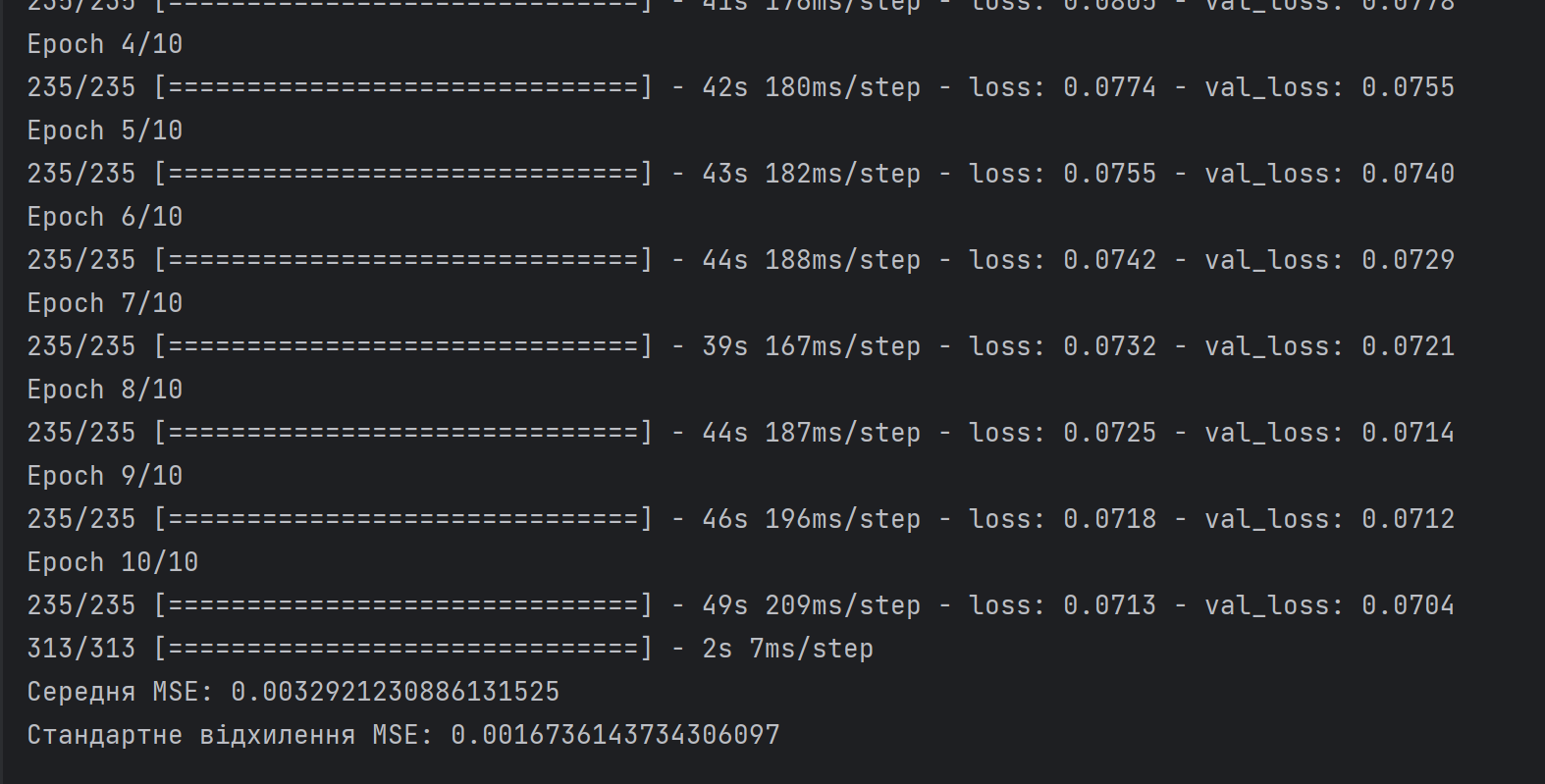
*(x\_train, y\_train), (x\_test, y\_test) = load\_kmnist()*

**Хід роботи**

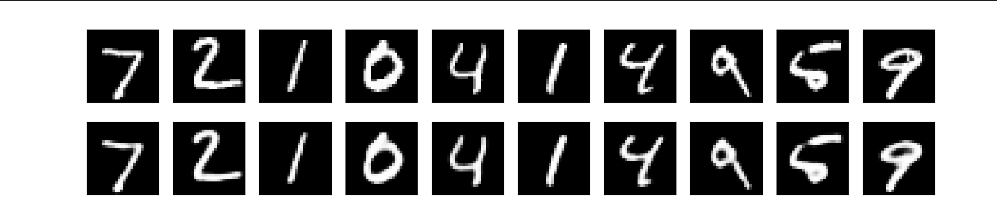
Код програми:

import gzip  
import idx2numpy  
import numpy as np  
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras.datasets import mnist  
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape, Conv2D, MaxPooling2D, UpSampling2D  
from tensorflow.keras.models import Model  
from tensorflow.keras.optimizers import RMSprop  
from tensorflow.keras import backend as K  
from tensorflow.keras.regularizers import l1, l2  
import matplotlib.pyplot as plt  
from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score  
  
def load\_kmnist():  
 with gzip.open('train-images-idx3-ubyte.gz', 'r') as f:  
 x\_train = idx2numpy.convert\_from\_file(f)  
 with gzip.open('train-labels-idx1-ubyte.gz', 'r') as f:  
 y\_train = idx2numpy.convert\_from\_file(f)  
 with gzip.open('t10k-images-idx3-ubyte.gz', 'r') as f:  
 x\_test = idx2numpy.convert\_from\_file(f)  
 with gzip.open('t10k-labels-idx1-ubyte.gz', 'r') as f:  
 y\_test = idx2numpy.convert\_from\_file(f)  
 return (x\_train, y\_train), (x\_test, y\_test)  
  
  
(x\_train, y\_train), (x\_test, y\_test) = load\_kmnist()  
  
# Завантаження даних  
(x\_train, \_), (x\_test, \_) = mnist.load\_data()  
# Нормалізація даних  
x\_train = x\_train.astype('float32') / 255.  
x\_test = x\_test.astype('float32') / 255.  
x\_train = x\_train.reshape((len(x\_train), 28, 28, 1))  
x\_test = x\_test.reshape((len(x\_test), 28, 28, 1))  
#Створення моделі автоенкодера.  
  
# Параметри вхідних даних  
input\_img = Input(shape=(28, 28, 1))  
# Архітектура енкодера  
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input\_img)  
x = MaxPooling2D((2, 2), padding='same')(x)  
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)  
encoded = MaxPooling2D((2, 2), padding='same')(x)  
# Архітектура декодера  
x = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)  
x = UpSampling2D((2, 2))(x)  
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)  
x = UpSampling2D((2, 2))(x)  
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)  
# Модель автоенкодера  
autoencoder = Model(input\_img, decoded)  
autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')  
#Навчання автоенкодера  
history = autoencoder.fit(x\_train, x\_train,  
 epochs=10,  
 batch\_size=256,  
 shuffle=True,  
 validation\_data=(x\_test, x\_test))  
#Візуалізація результатів  
decoded\_imgs = autoencoder.predict(x\_test)  
n = 10 # скільки цифр ми відобразимо  
plt.figure(figsize=(20, 4))  
for i in range(n):  
 # Відображення оригіналу  
 ax = plt.subplot(2, n, i + 1)  
 plt.imshow(x\_test[i].reshape(28, 28))  
 plt.gray()  
 ax.get\_xaxis().set\_visible(False)  
 ax.get\_yaxis().set\_visible(False)  
 # Відображення відтворення  
 ax = plt.subplot(2, n, i + 1 + n)  
 plt.imshow(decoded\_imgs[i].reshape(28, 28))  
 plt.gray()  
 ax.get\_xaxis().set\_visible(False)  
 ax.get\_yaxis().set\_visible(False)  
plt.show()  
#Розрахунок метрик якості та візуалізація навчання.  
# Відображення графіка функції втрат  
plt.figure(figsize=(10, 5))  
plt.plot(history.history['loss'], label='Train Loss')  
plt.plot(history.history['val\_loss'], label='Validation Loss')  
plt.title('Model Loss')  
plt.ylabel('Loss')  
plt.xlabel('Epochs')  
plt.legend(loc='upper right')  
plt.show()  
# Розрахунок та відображення MSE між оригінальними та відтвореними зображеннями  
mse = np.mean(np.square(x\_test - decoded\_imgs), axis=(1,2,3))  
print(f"Середня MSE: {np.mean(mse)}")  
print(f"Стандартне відхилення MSE: {np.std(mse)}")  
plt.figure(figsize=(10, 5))  
plt.hist(mse, bins=50)  
plt.xlabel('MSE')  
plt.ylabel('Кількість зображень')  
plt.show()

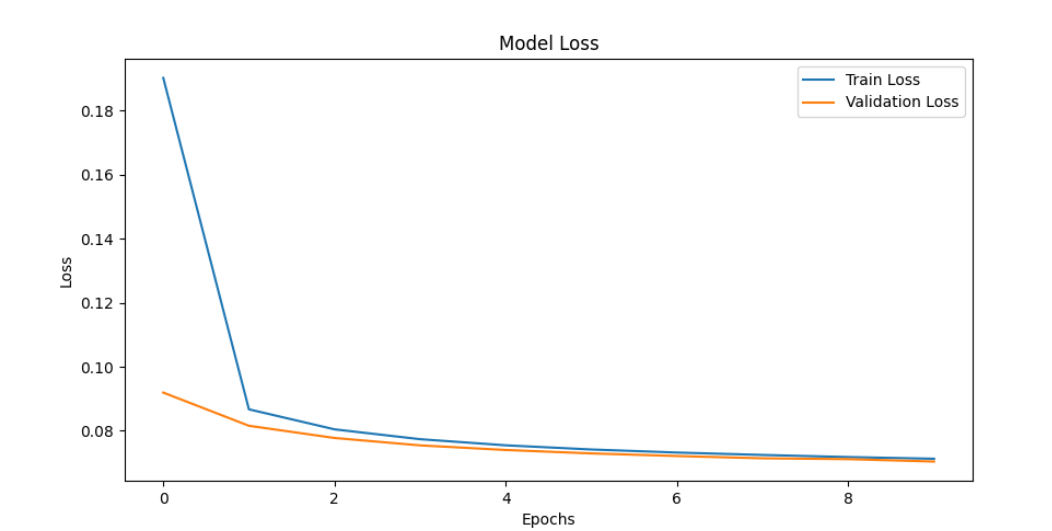
Результати:

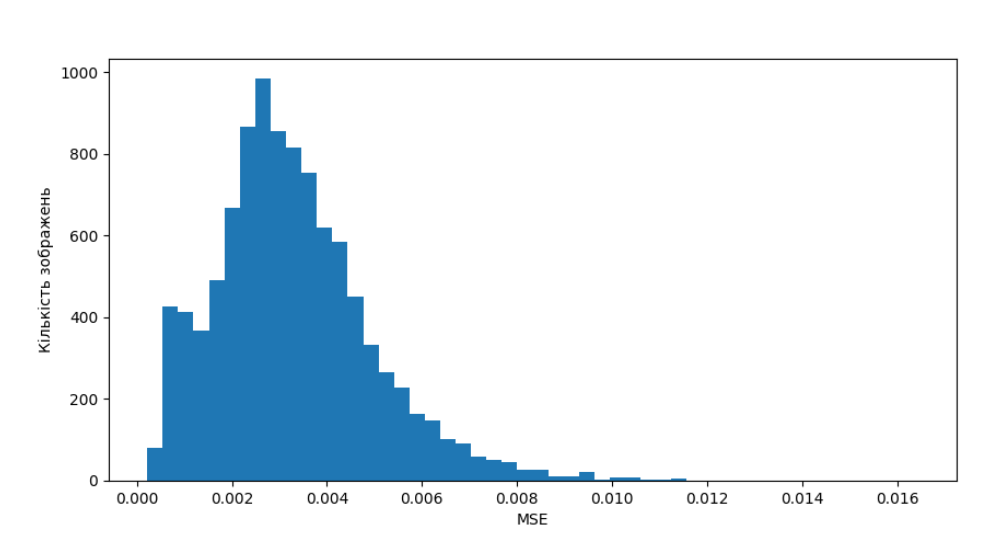


Візуалізація результатів до та після автоенкодерування:



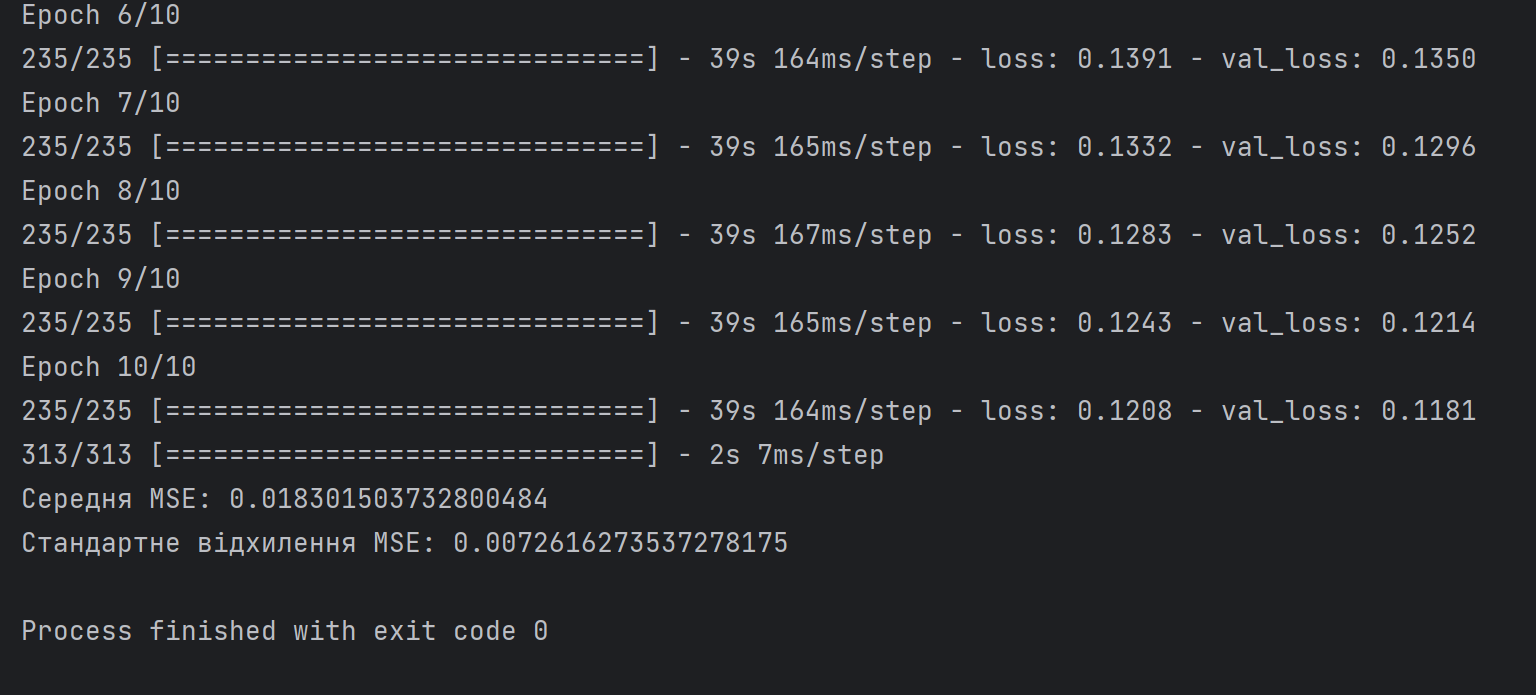
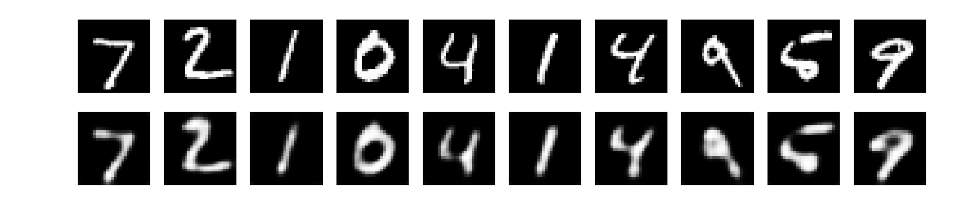
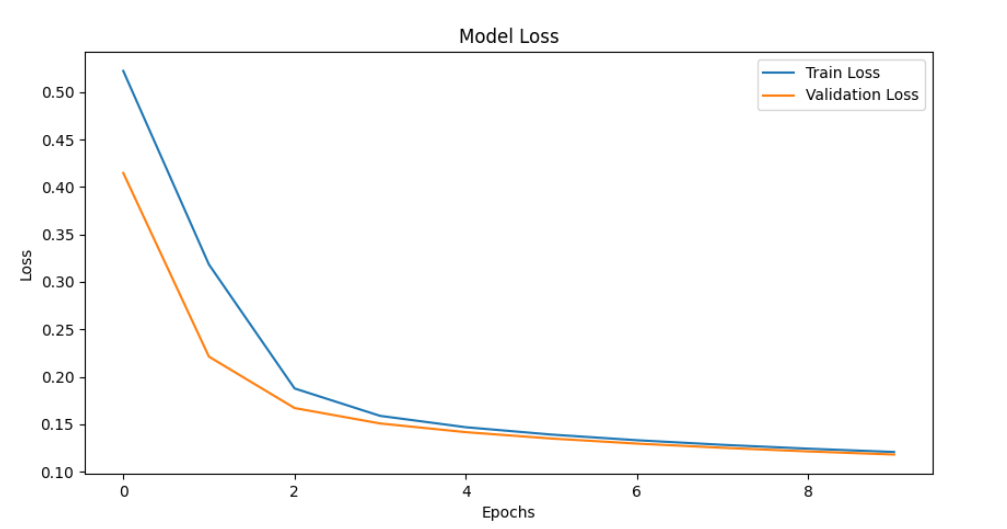
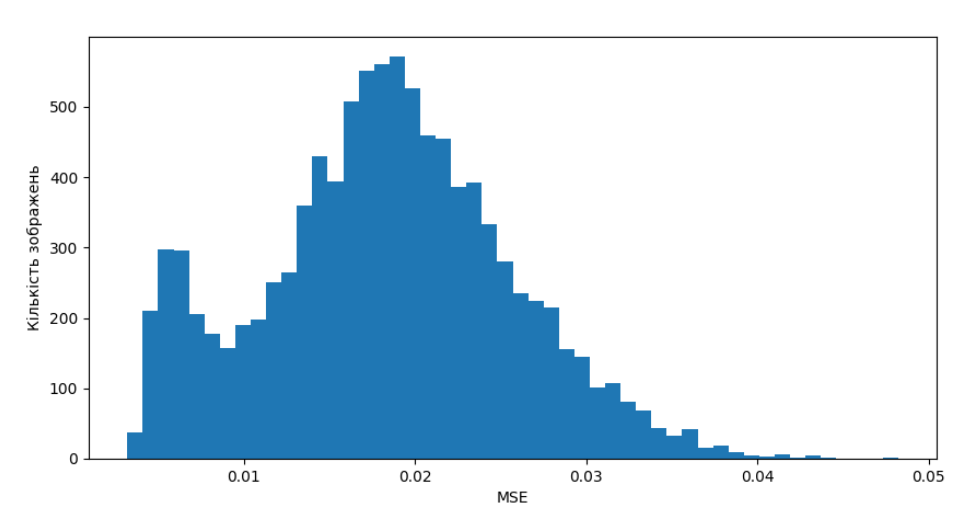
Відображення графіка функції втрат на тренувальній та тестовій вибірці:

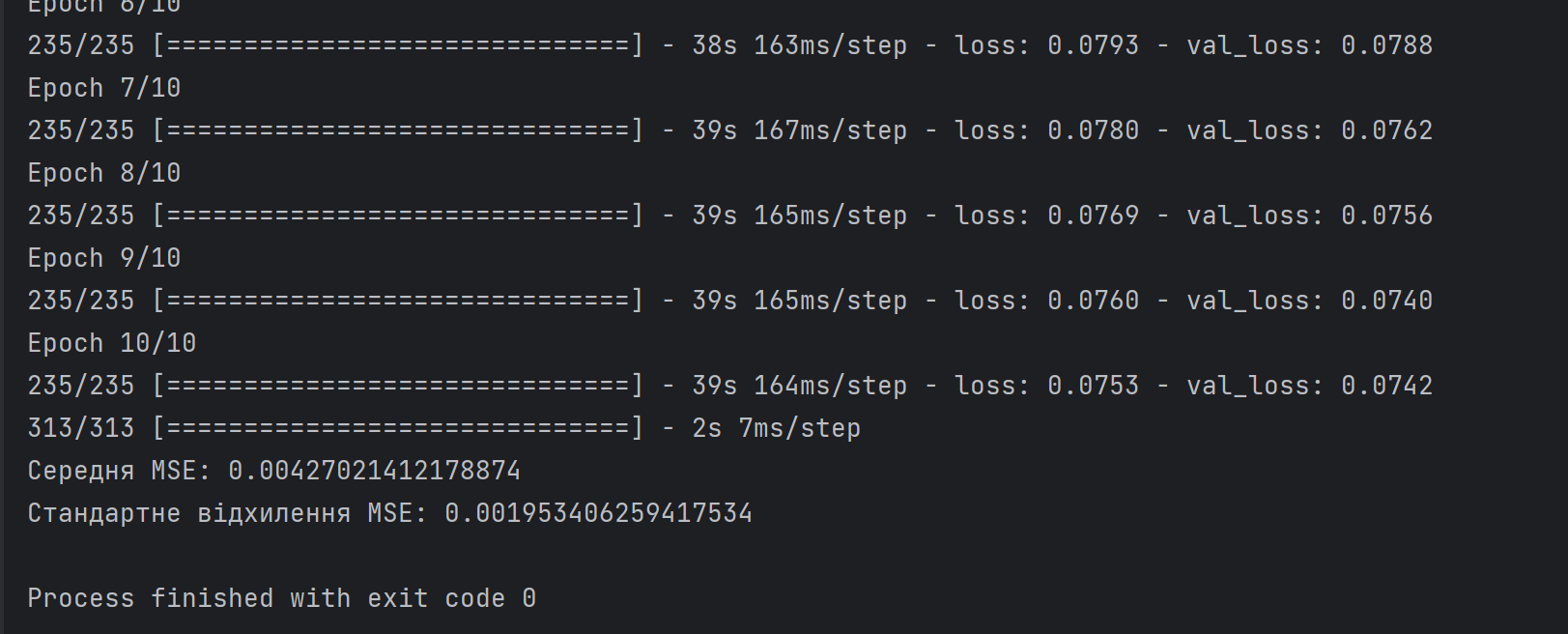
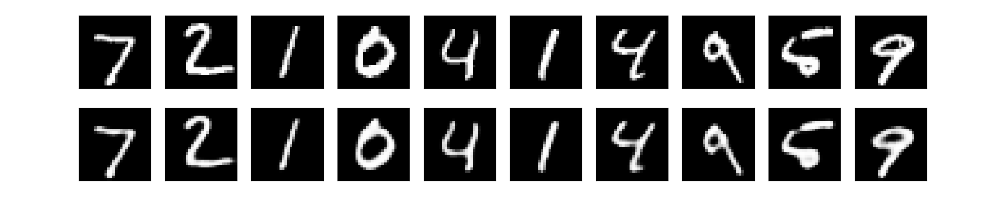
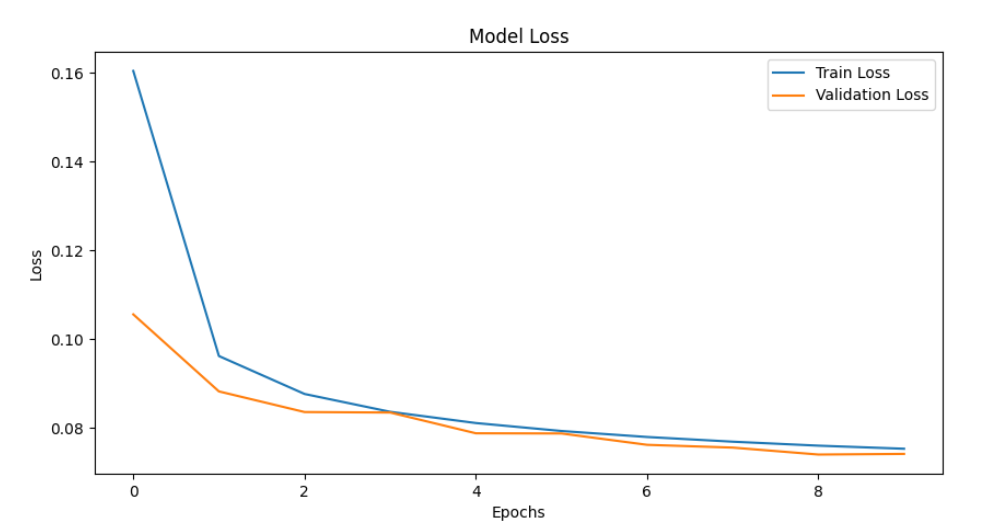


Розрахунок та відображення MSE між оригінальними та відтвореними зображеннями:

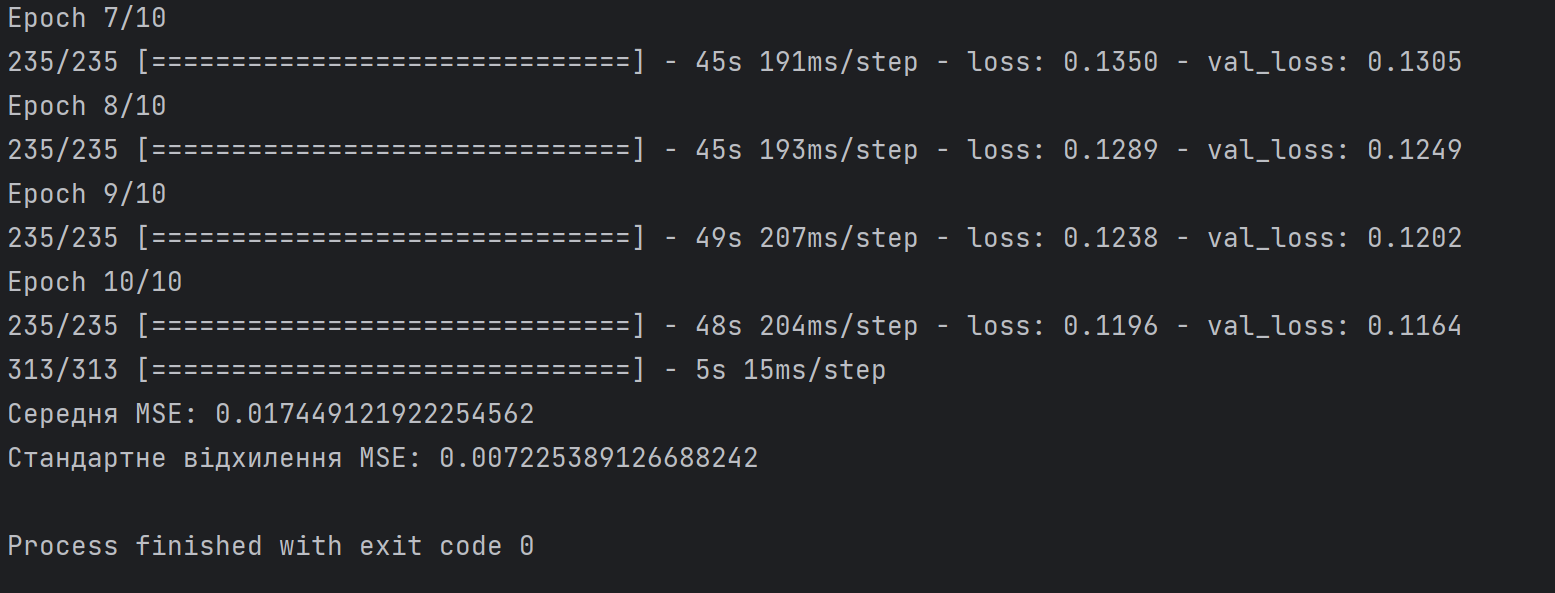
Дослідимо вплив роботи різних оптимізаторів та різних функцій активації:

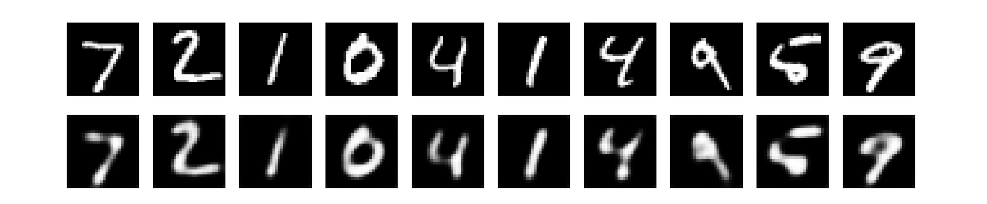
Оптимізатор SGD, функцій активації relu

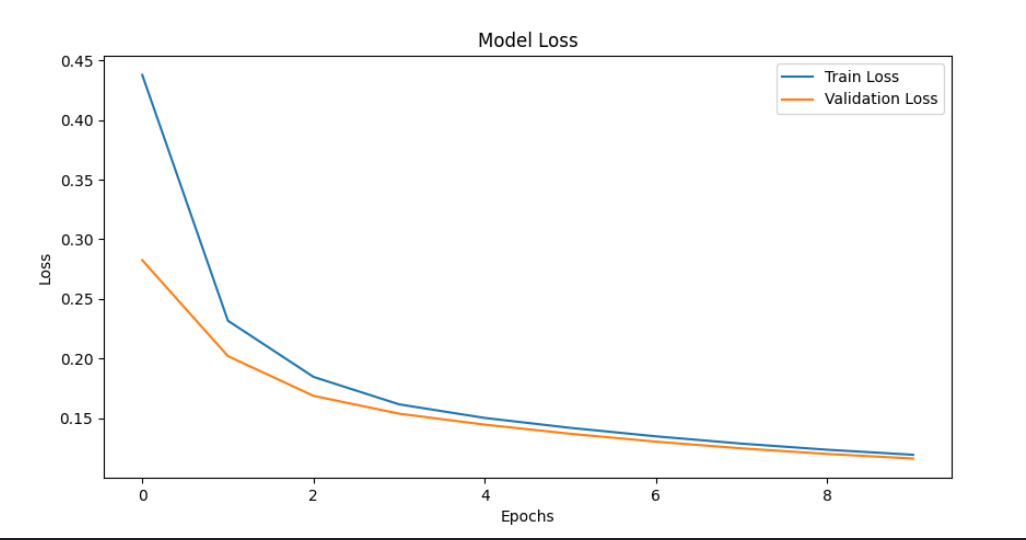
   

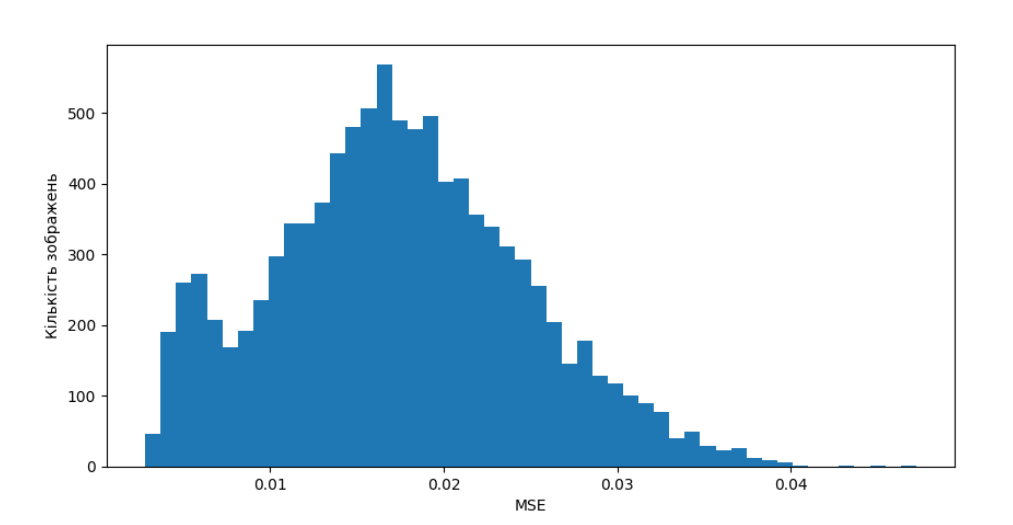
оптимізатор RMSprop, функцій активації relu  
   

оптимізатор SGD, функцій активації elu

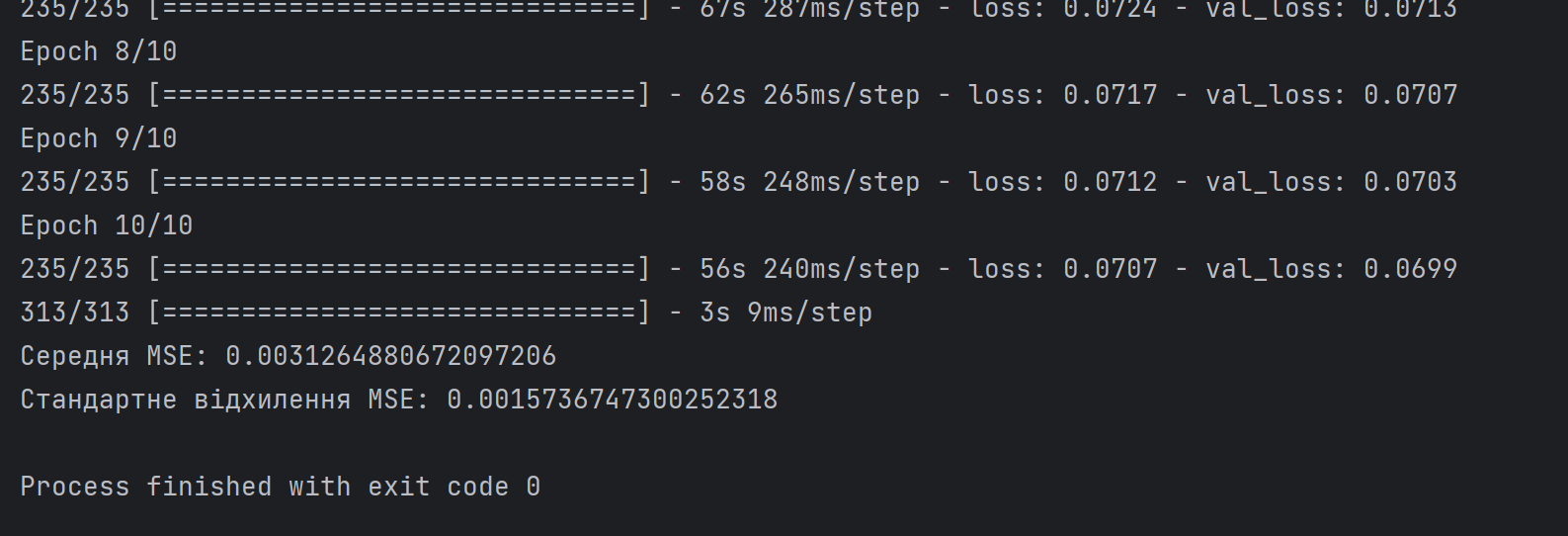
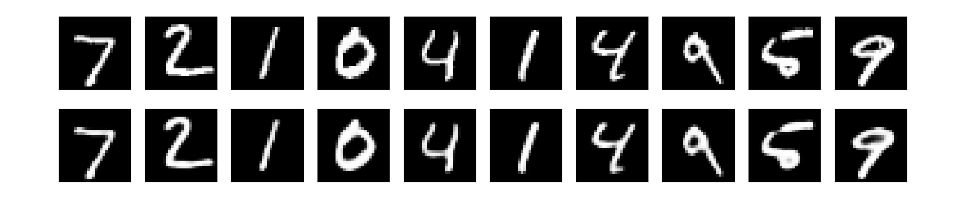
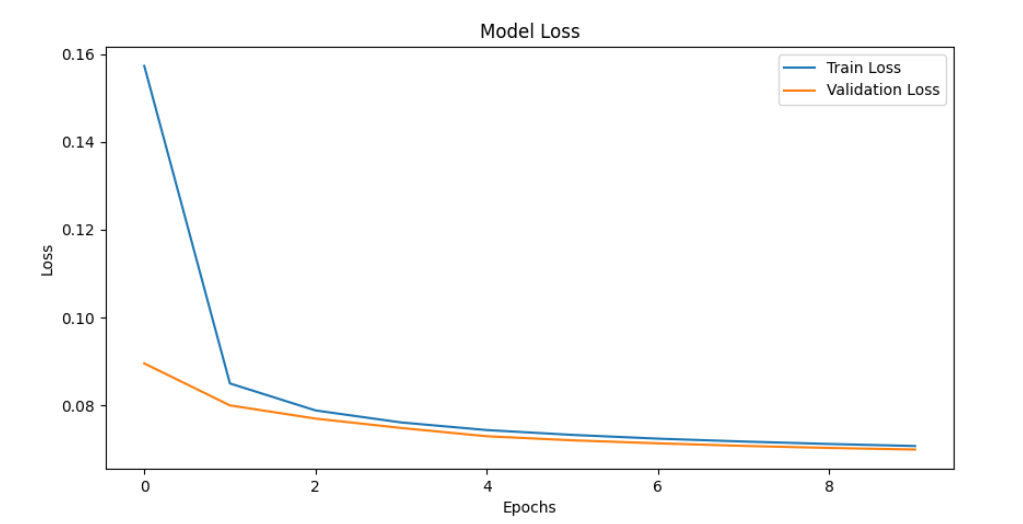
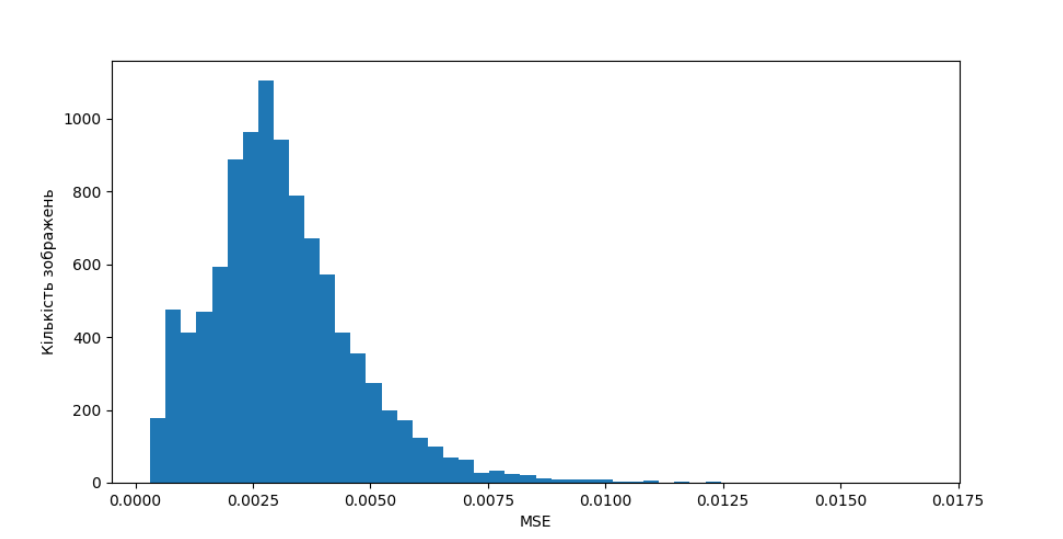




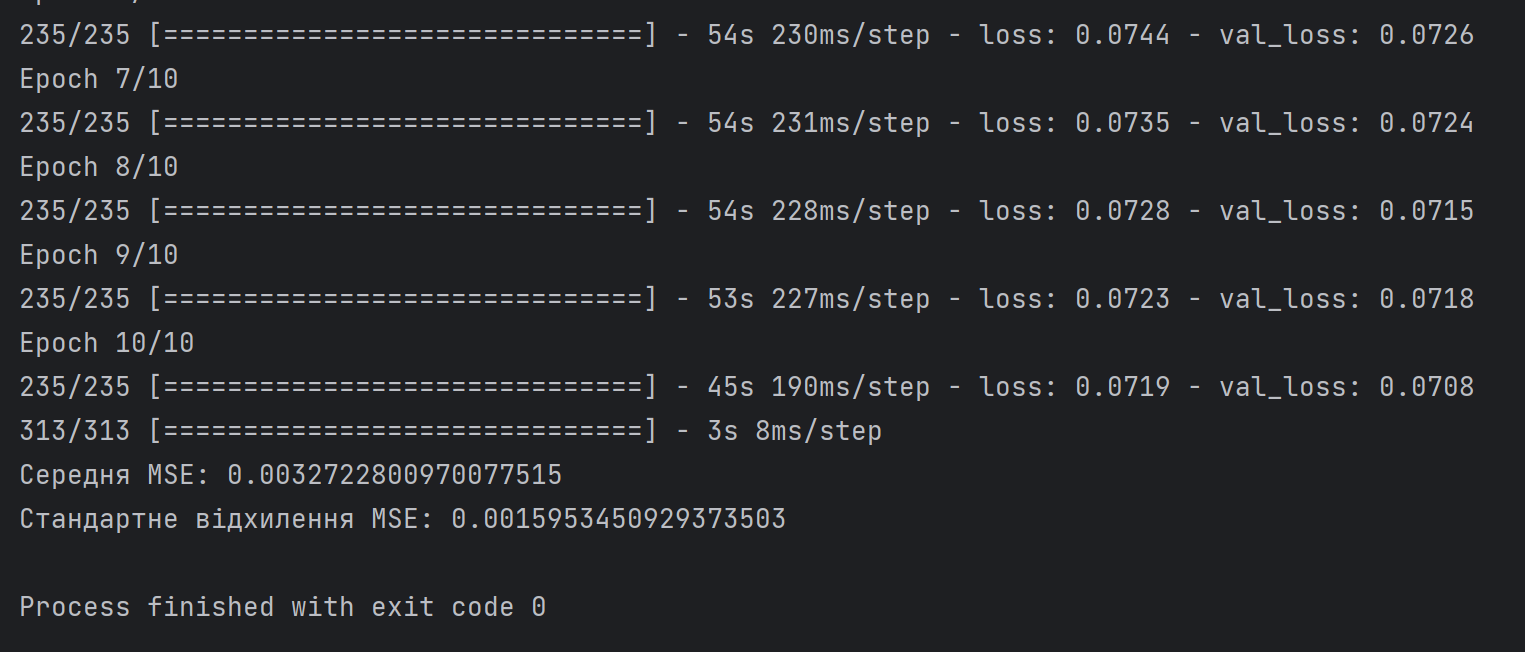
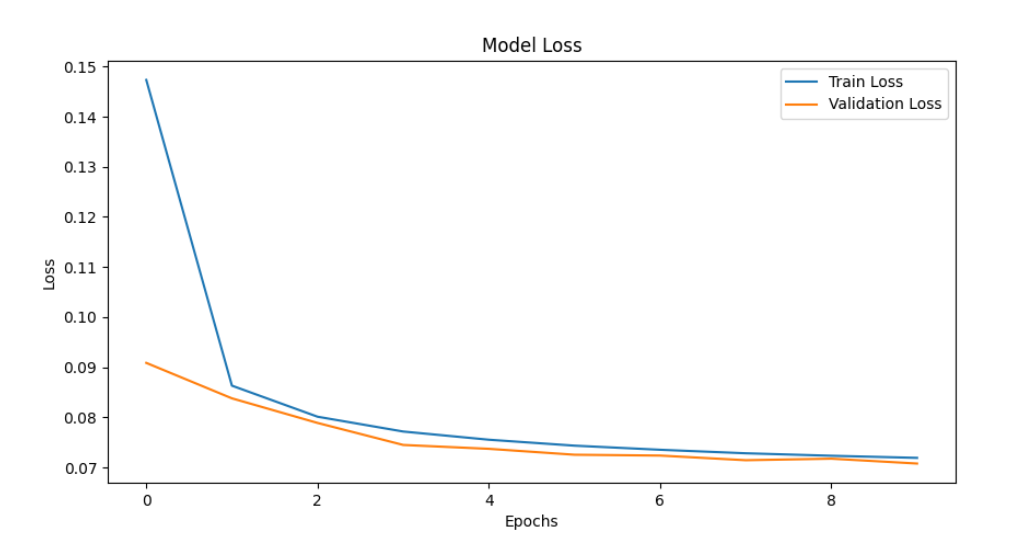
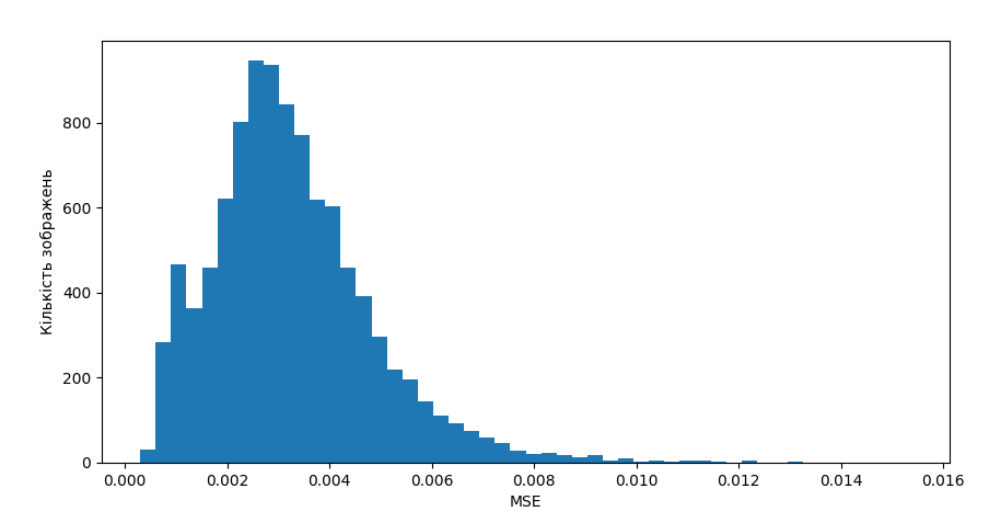




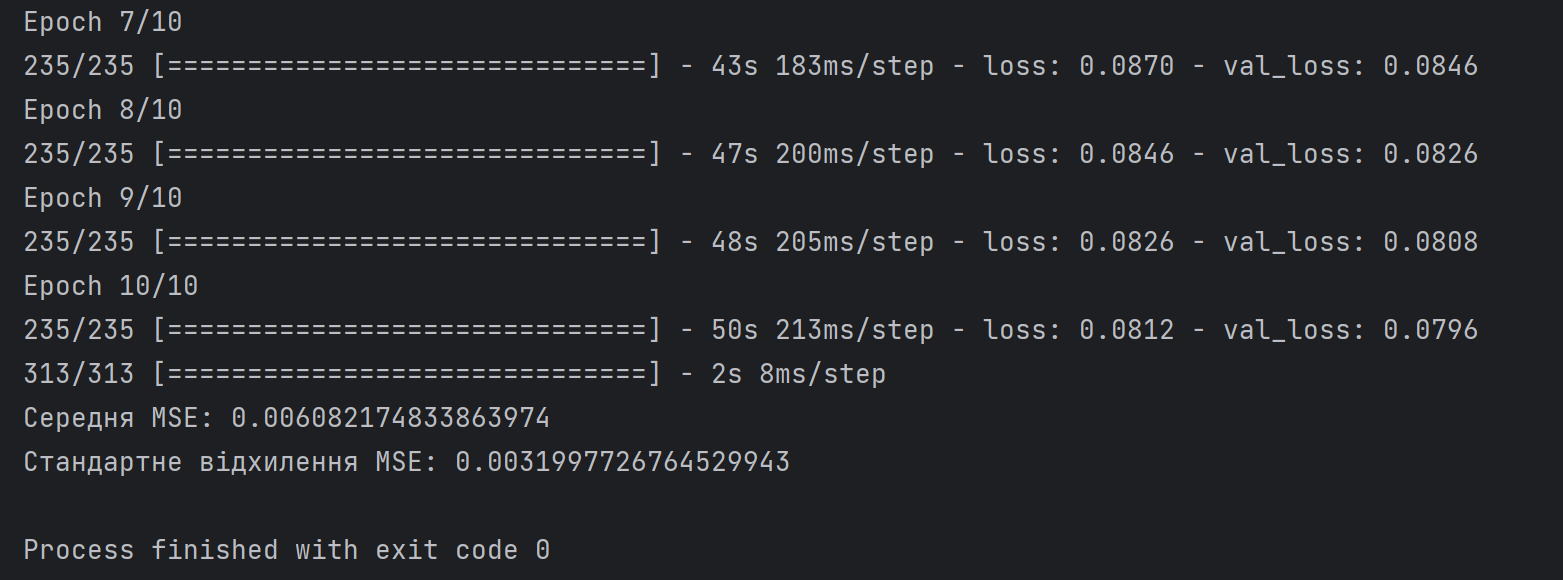
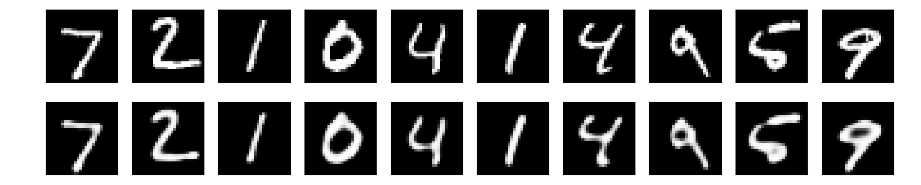
оптимізатор adam, функцій активації elu

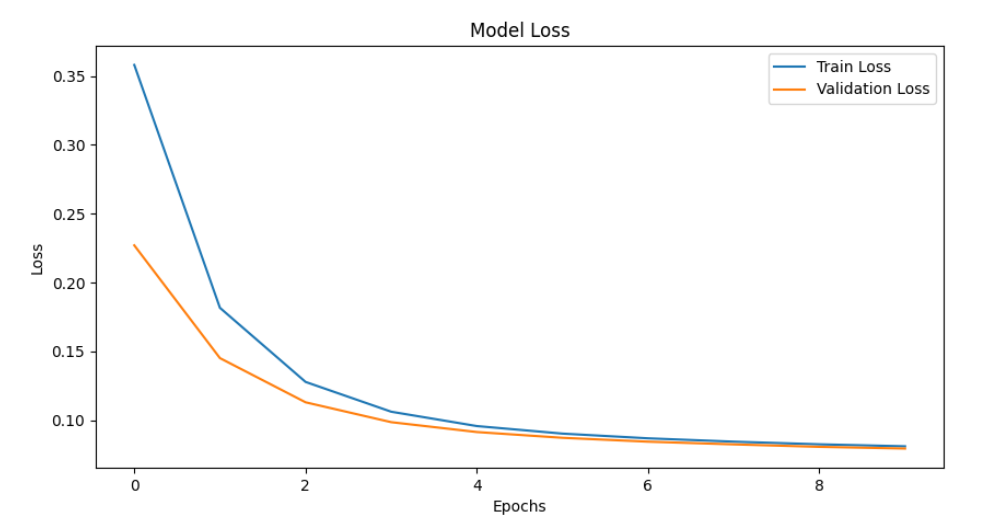
   

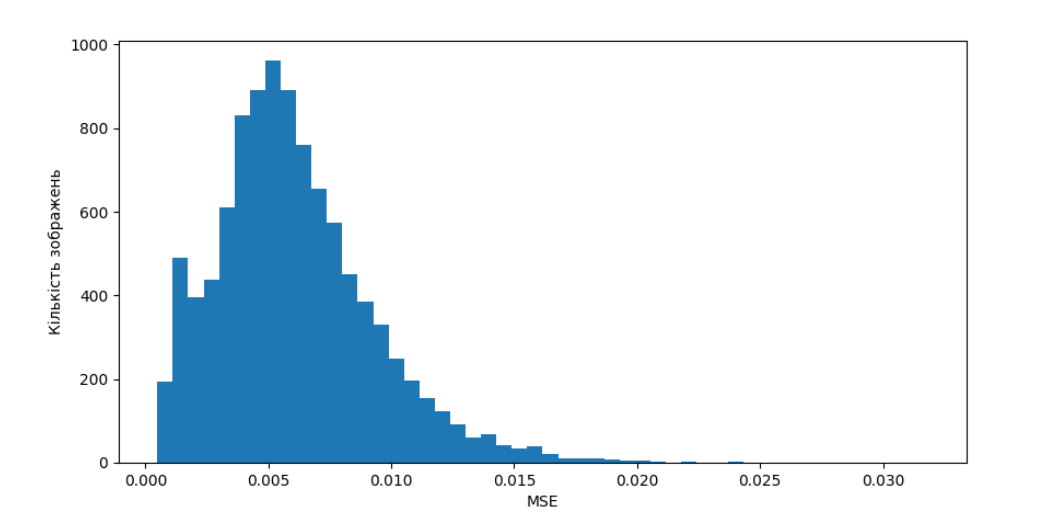
оптимізатор RMSprop, функцій активації elu

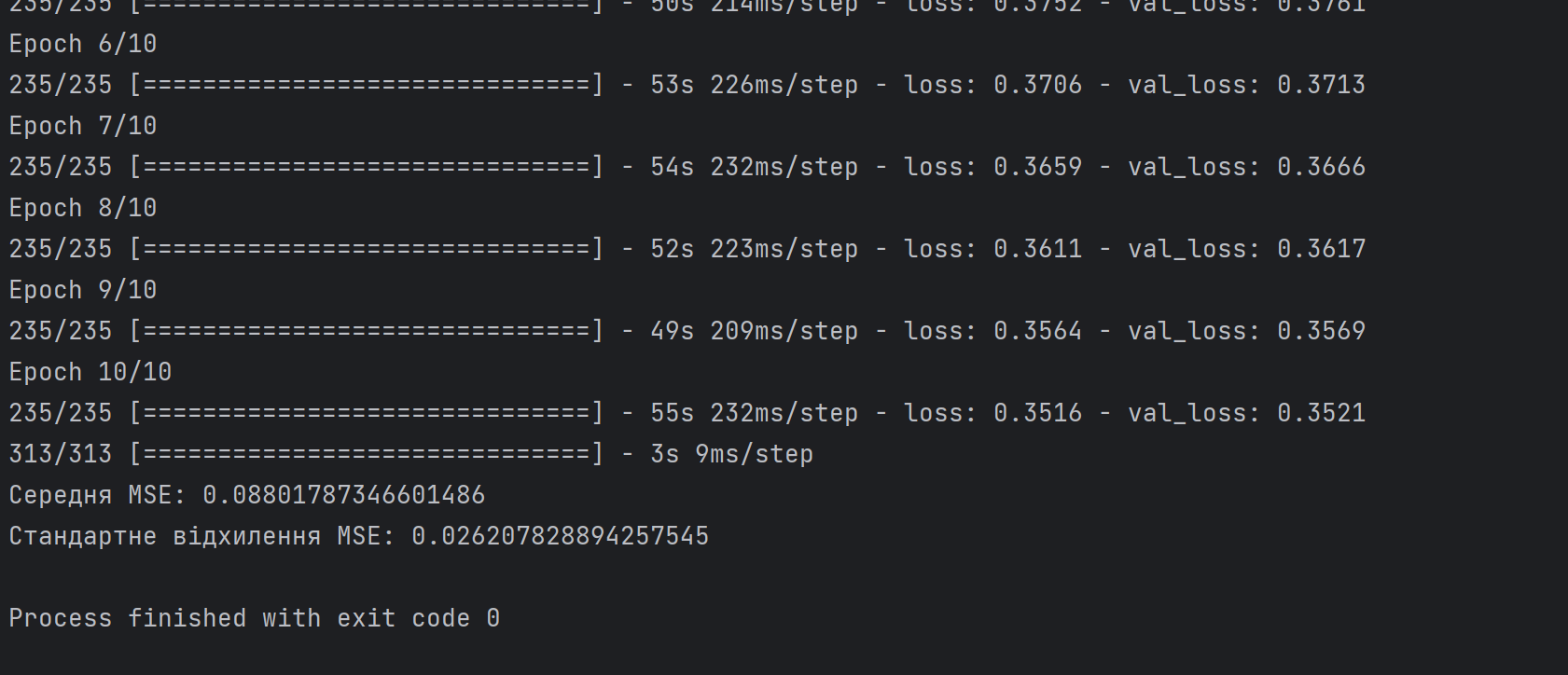
оптимізатор adam, функцій активації sigmoid

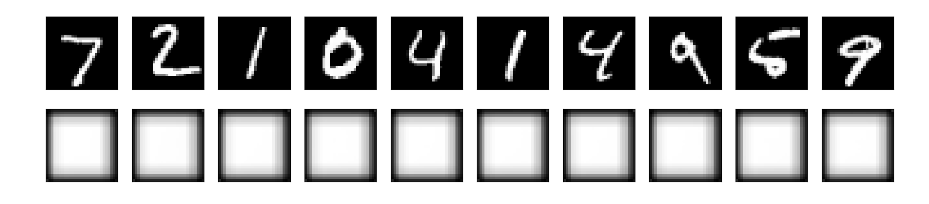
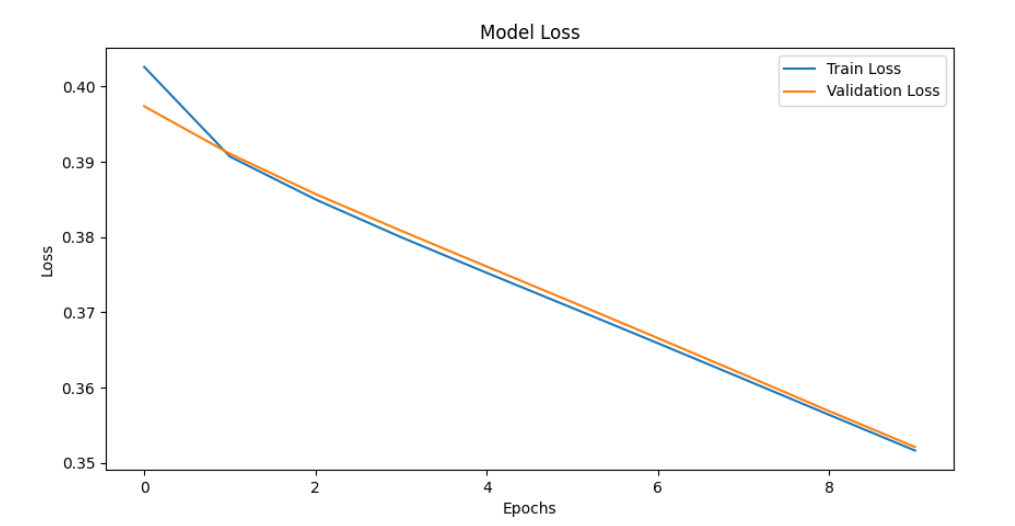
 



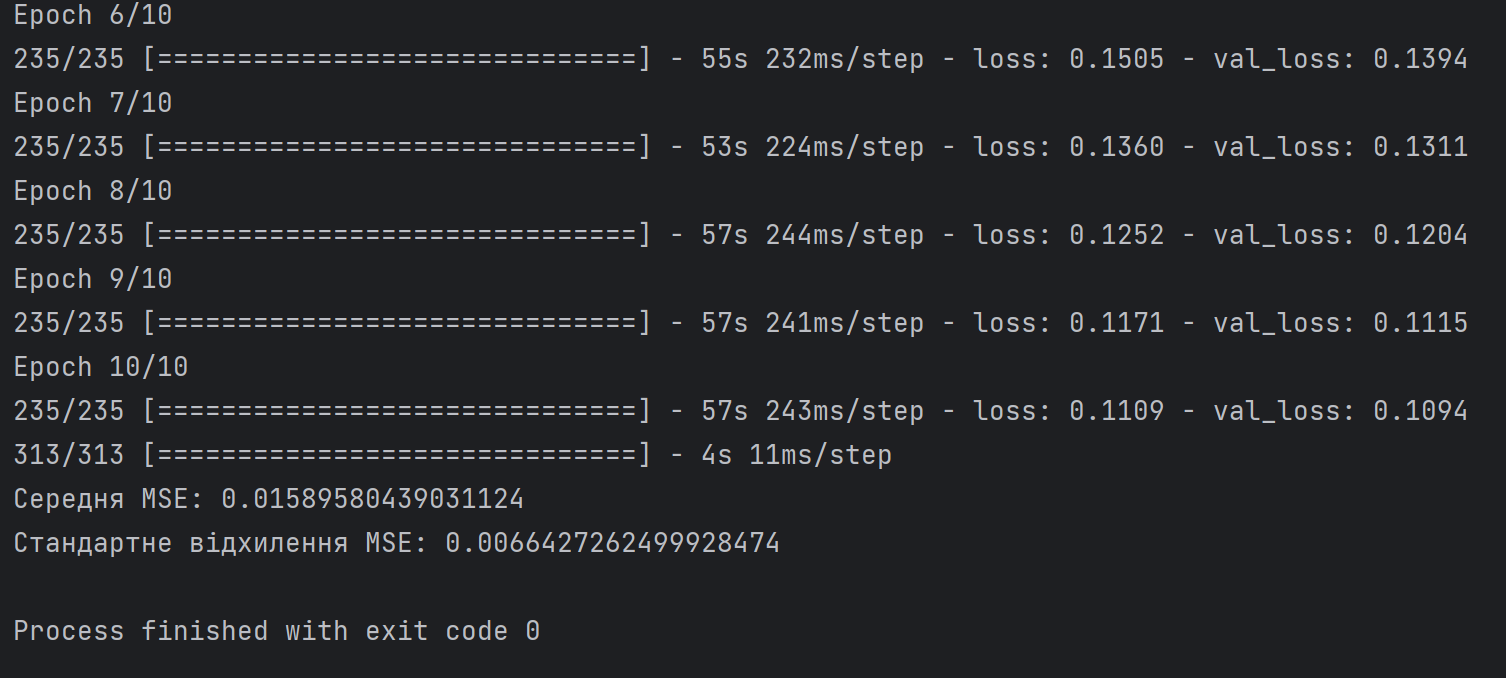


оптимізатор SGD, функцій активації sigmoid



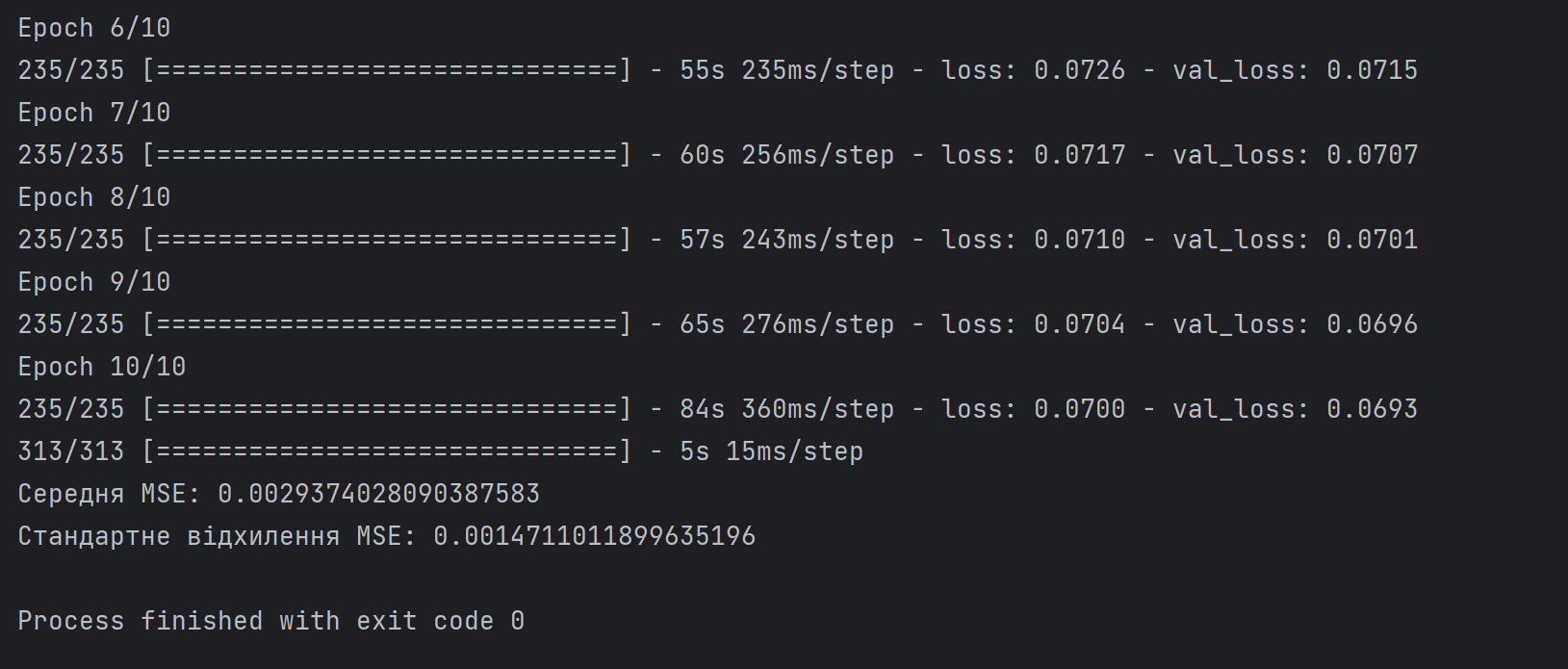
  

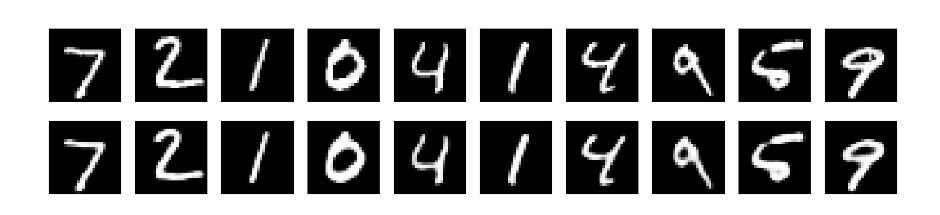
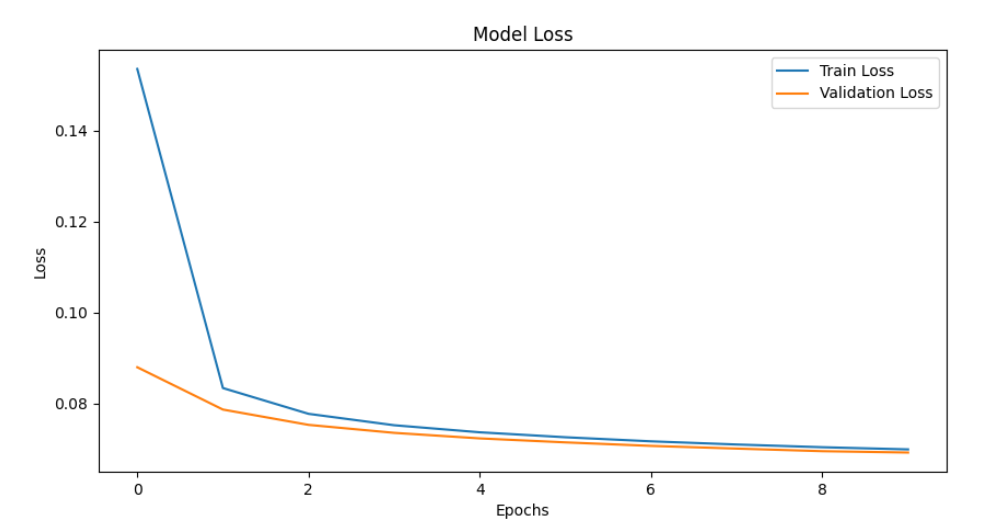
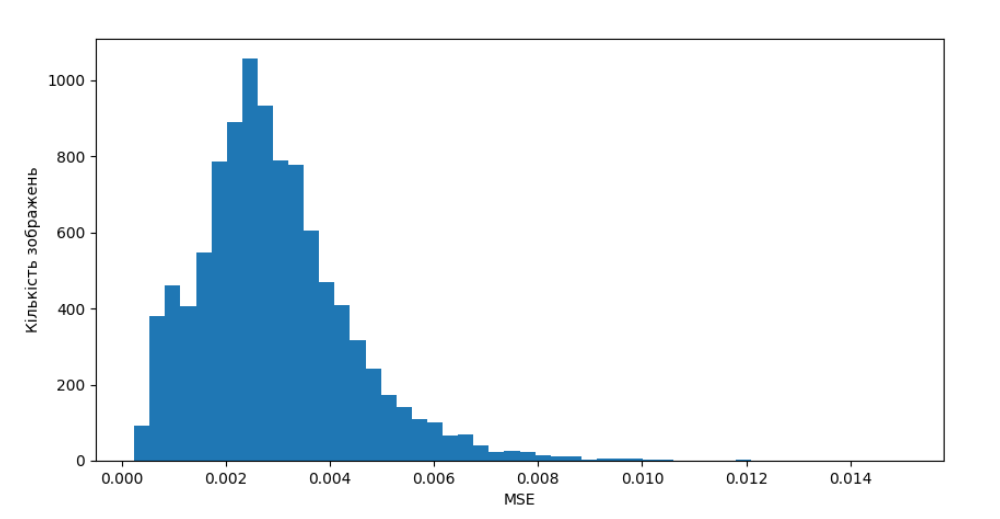
оптимізатор RMSprop, функцій активації sigmoid



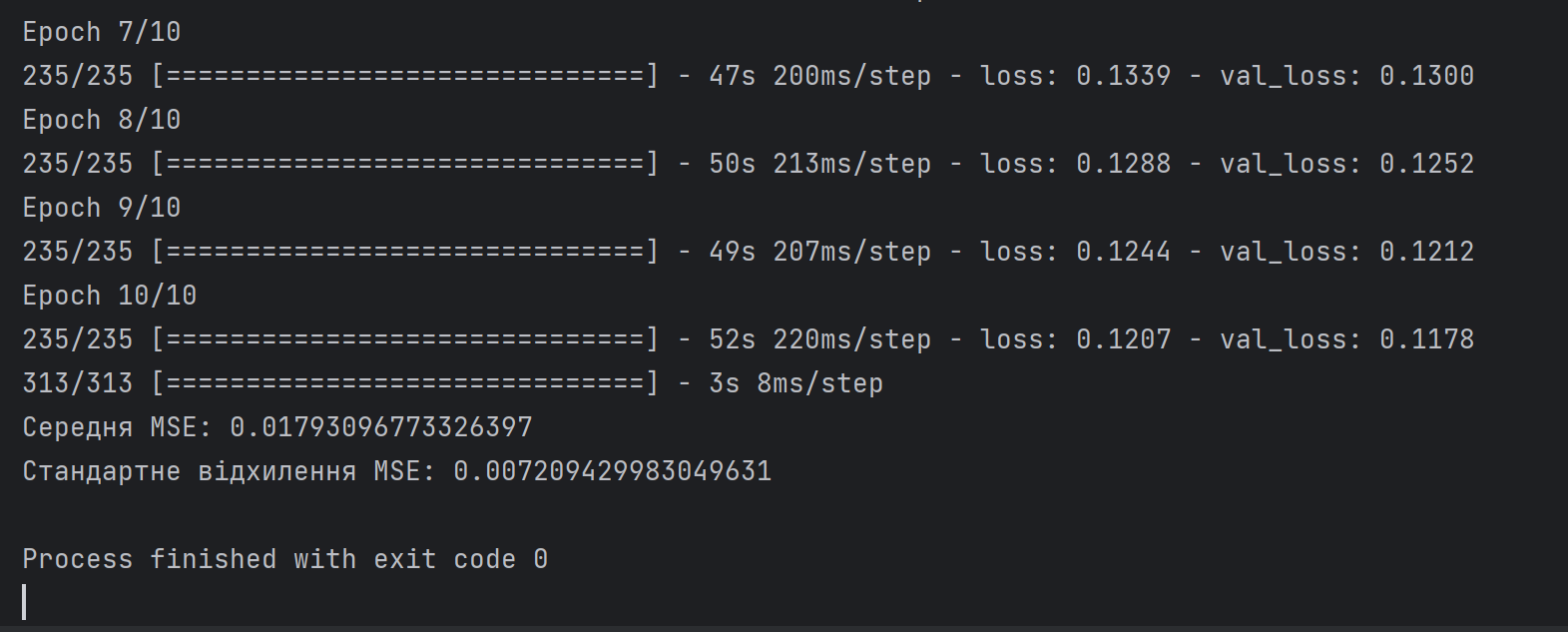


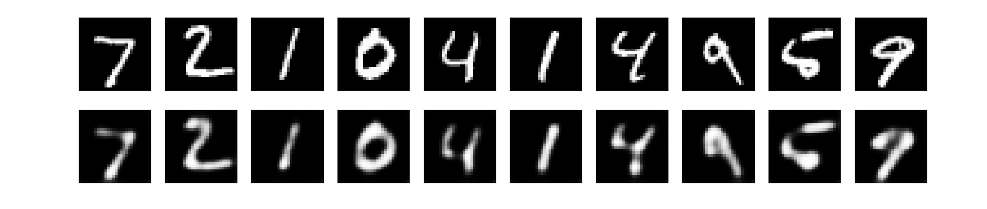
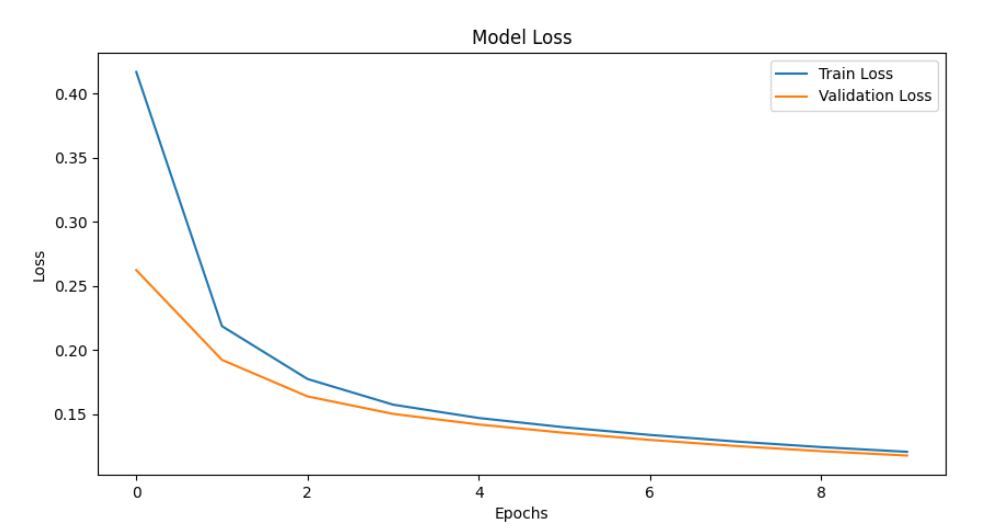
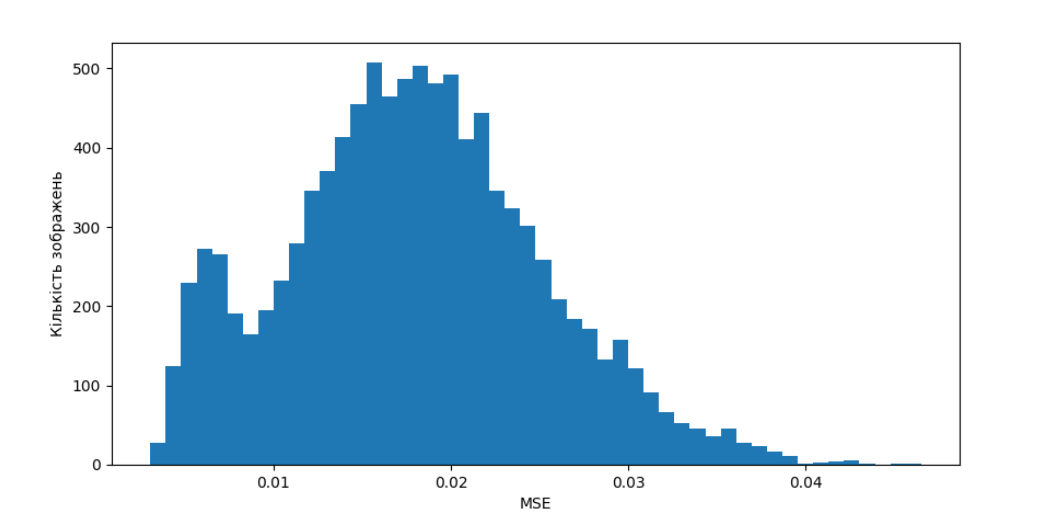
оптимізатор adam, функцій активації tanh



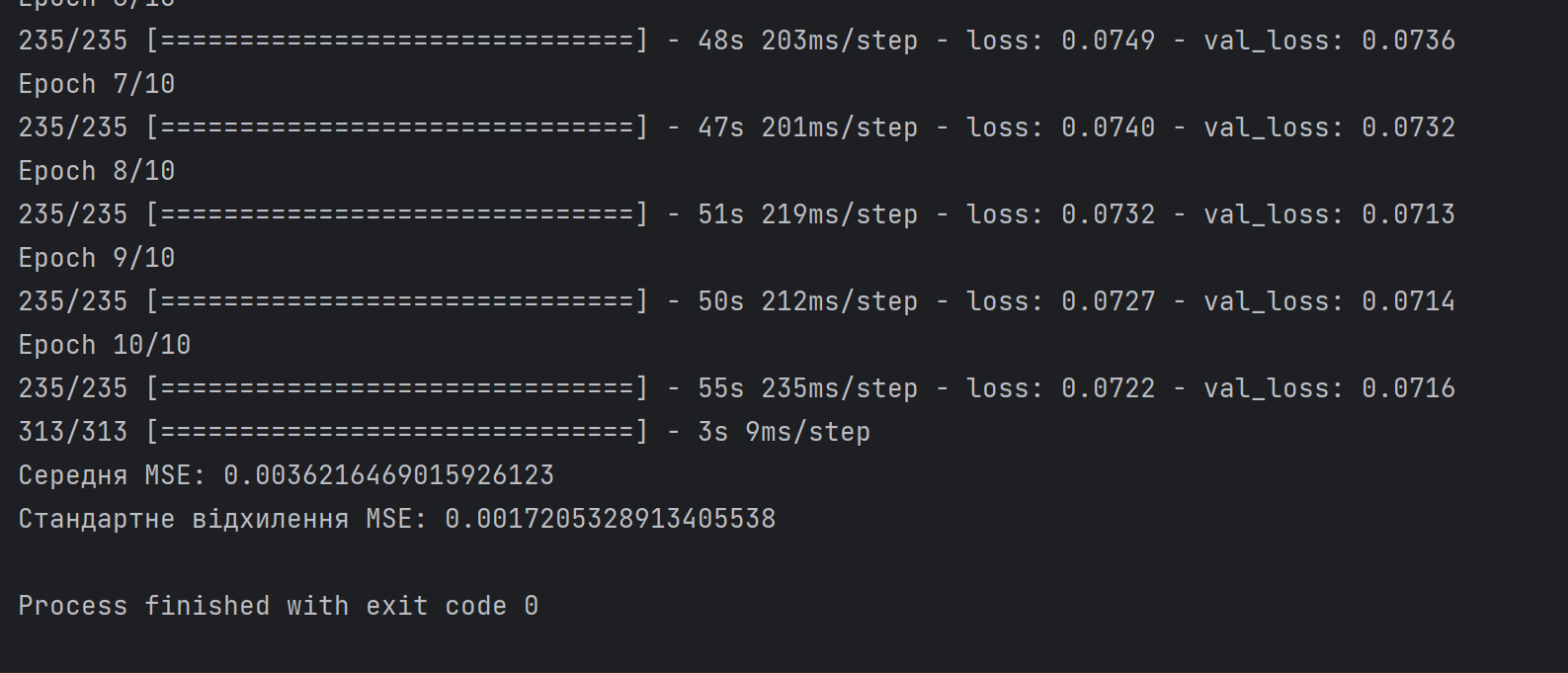
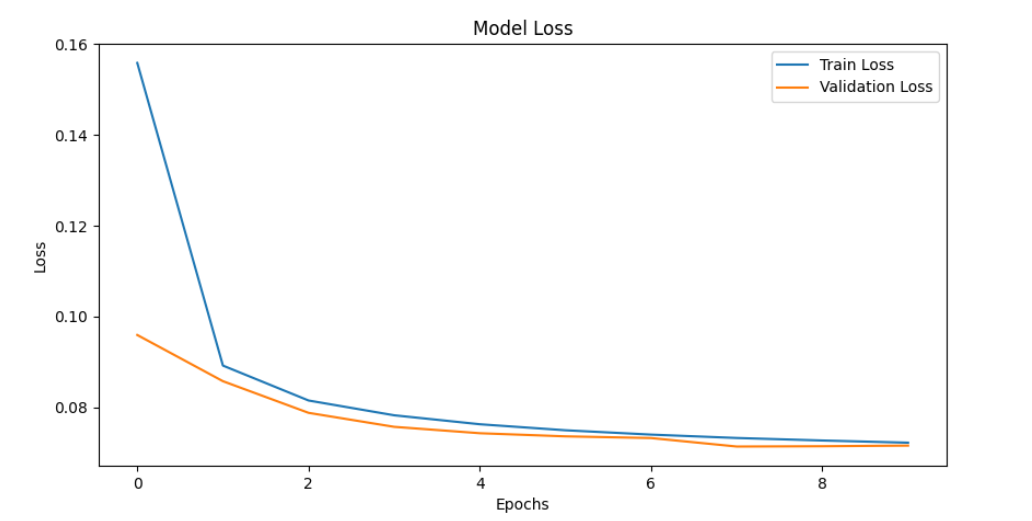
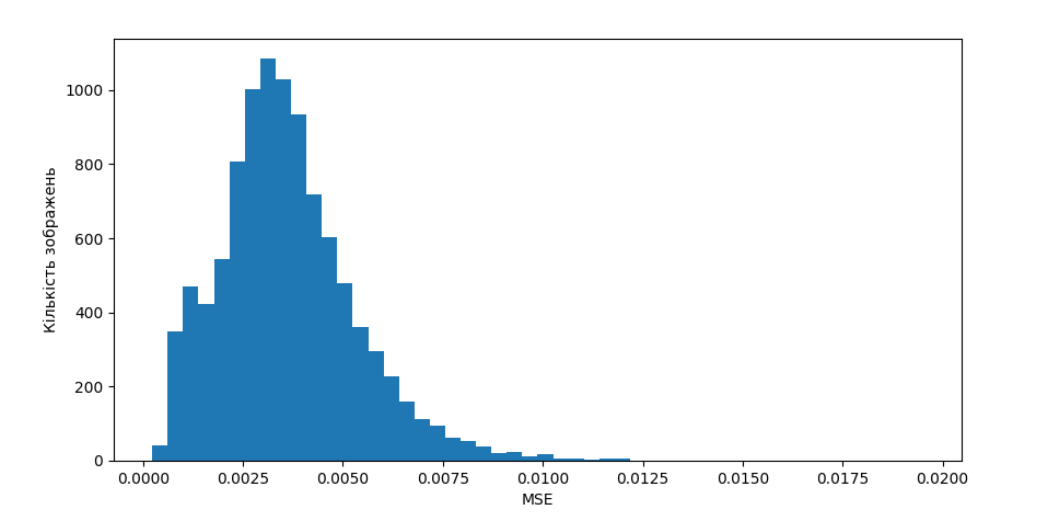
  

оптимізатор SGD, функцій активації tahn



оптимізатор RMSprop, функцій активації tanh

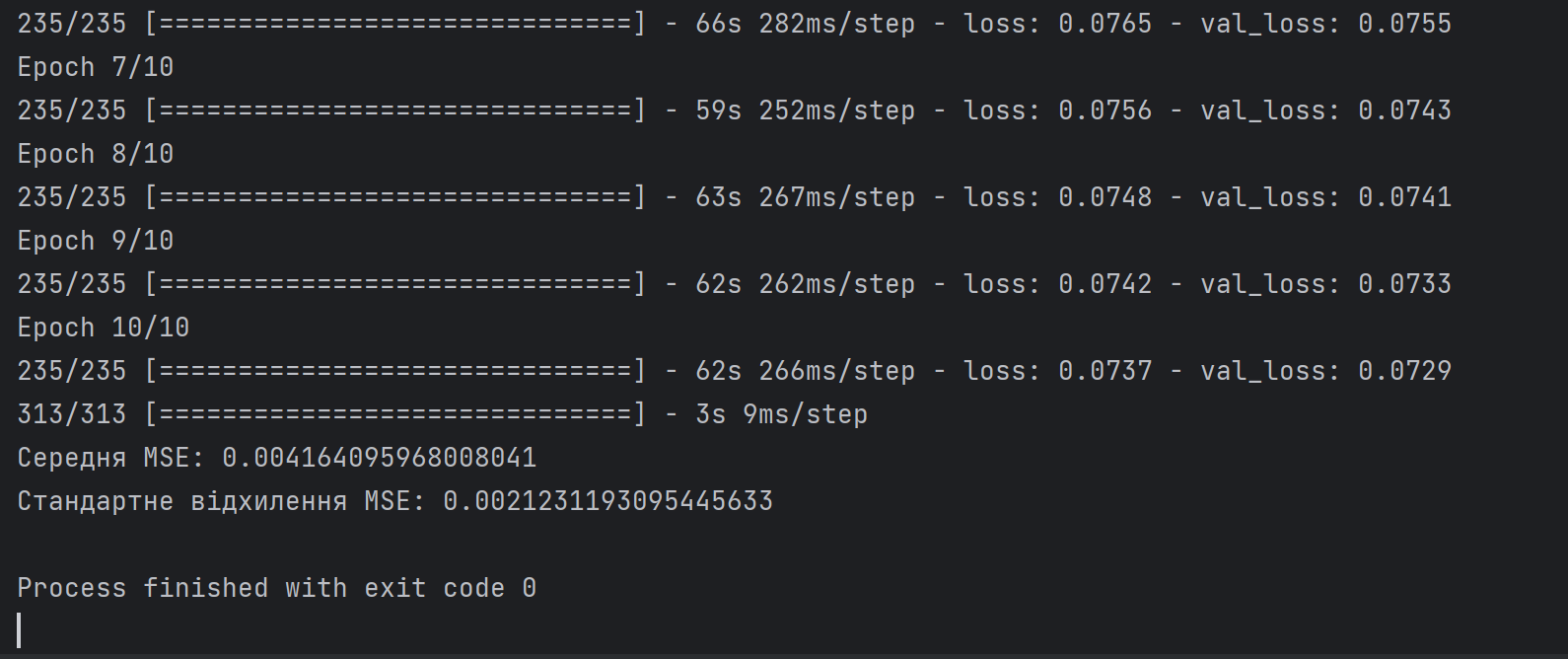
Розглянувши результати експерементів, можна зробити висновок найкращим результат був отриманий під час оптимізатор adam, функцій активації elu (середня середня MSE: 0.00313, стандартне відхилення MSE: 0.001574) та оптимізатор adam, функцій активації relu (середня середня MSE: 0.00329, стандартне відхилення MSE: 0.001674). Найгірший результат був отриманий під час використання оптимізатор RMSprop, функцій активації sigmoi (середня середня MSE: 0.0159, стандартне відхилення MSE: 0.006643). Експеримент, в якому використовувався оптимізатор SGD та функції активації sigmoid, вивів некоректні результати.

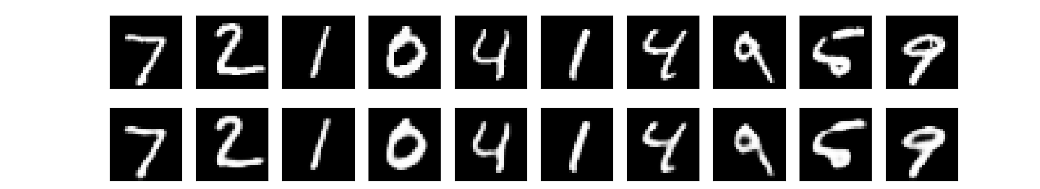
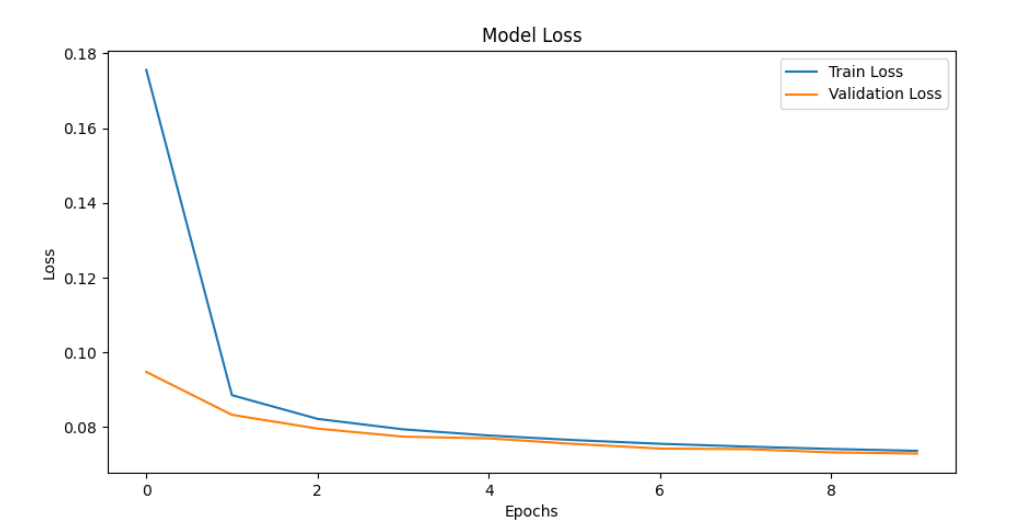
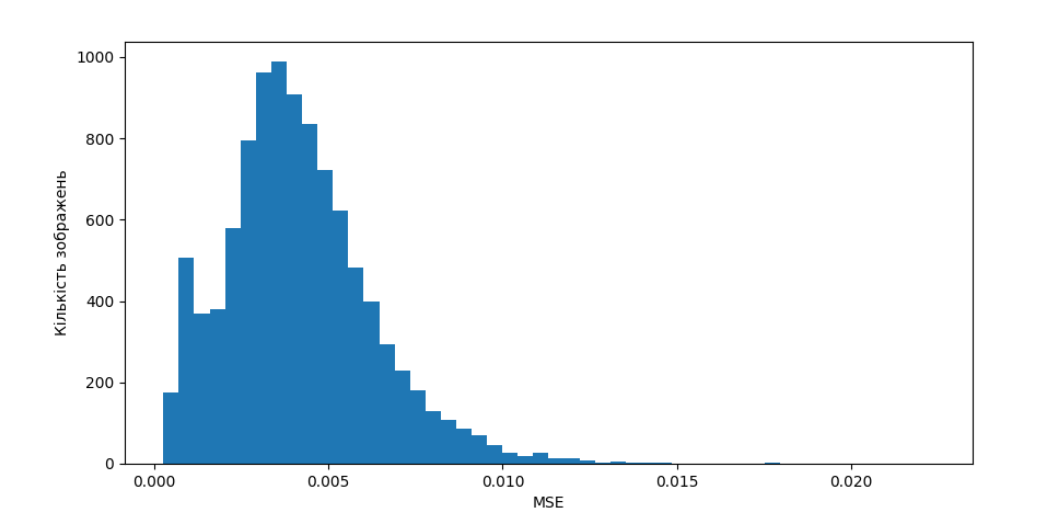
Спробуваємо змінити архітектуру нашого автоенкодеру

збільшимо кількість згорткових шарів



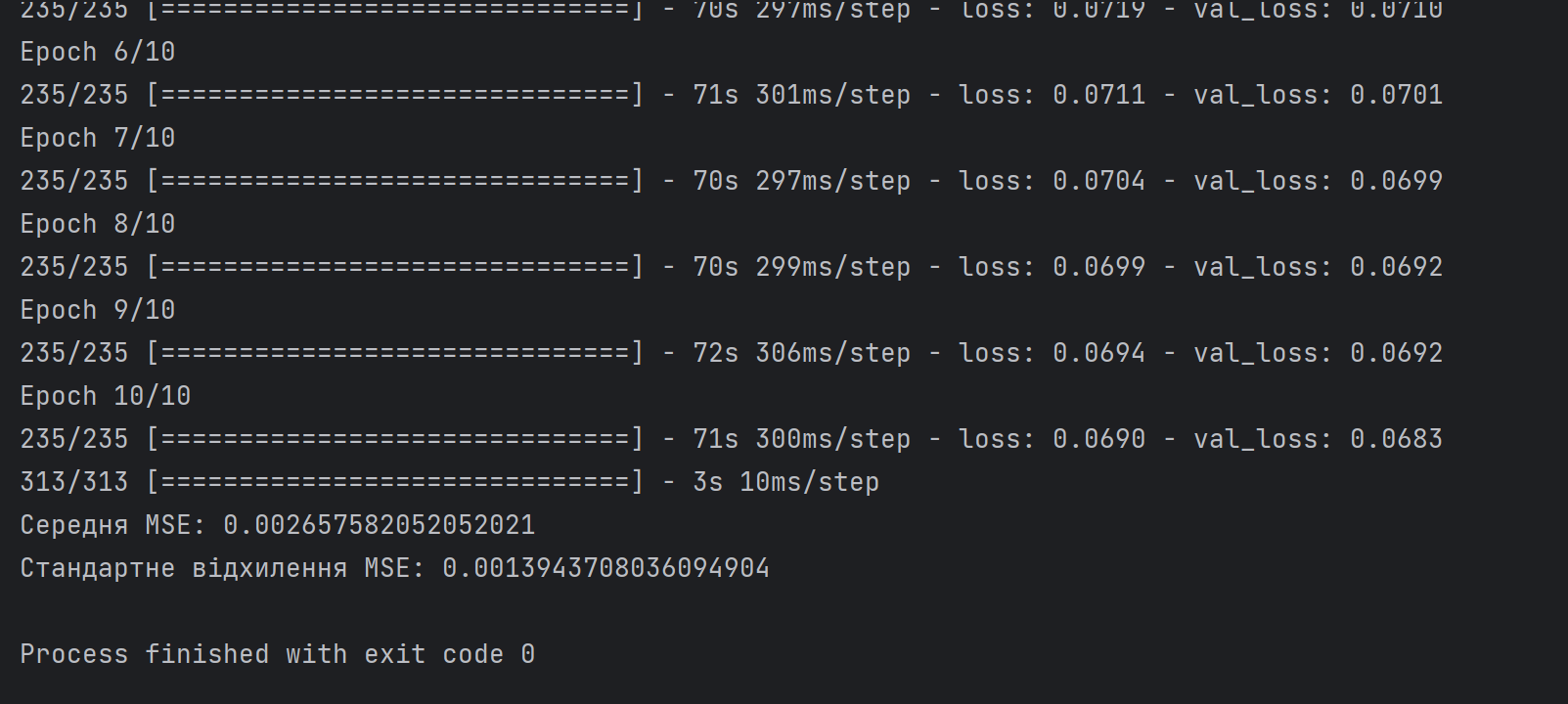
Результат

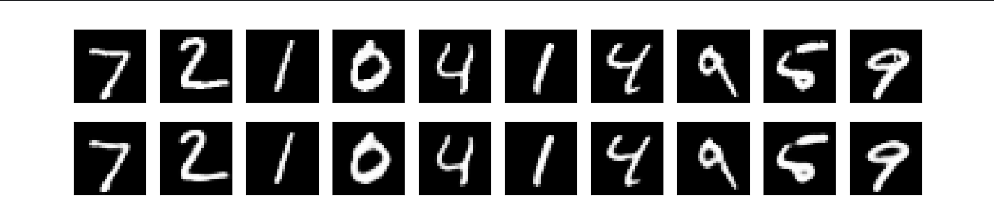
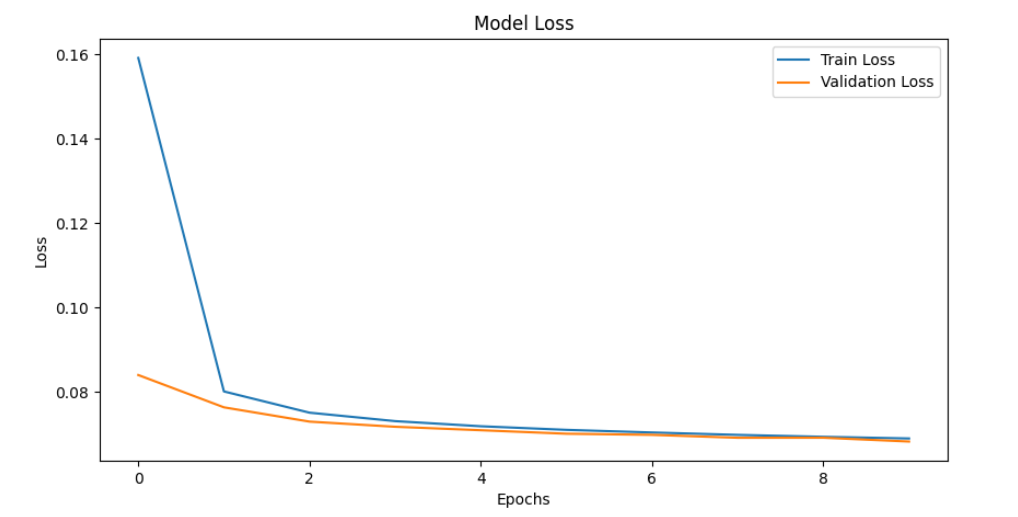
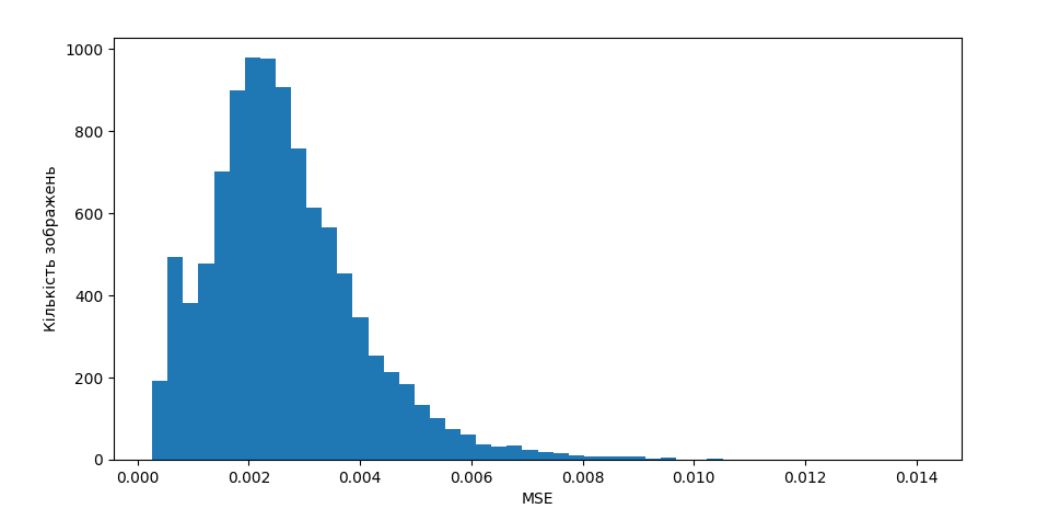


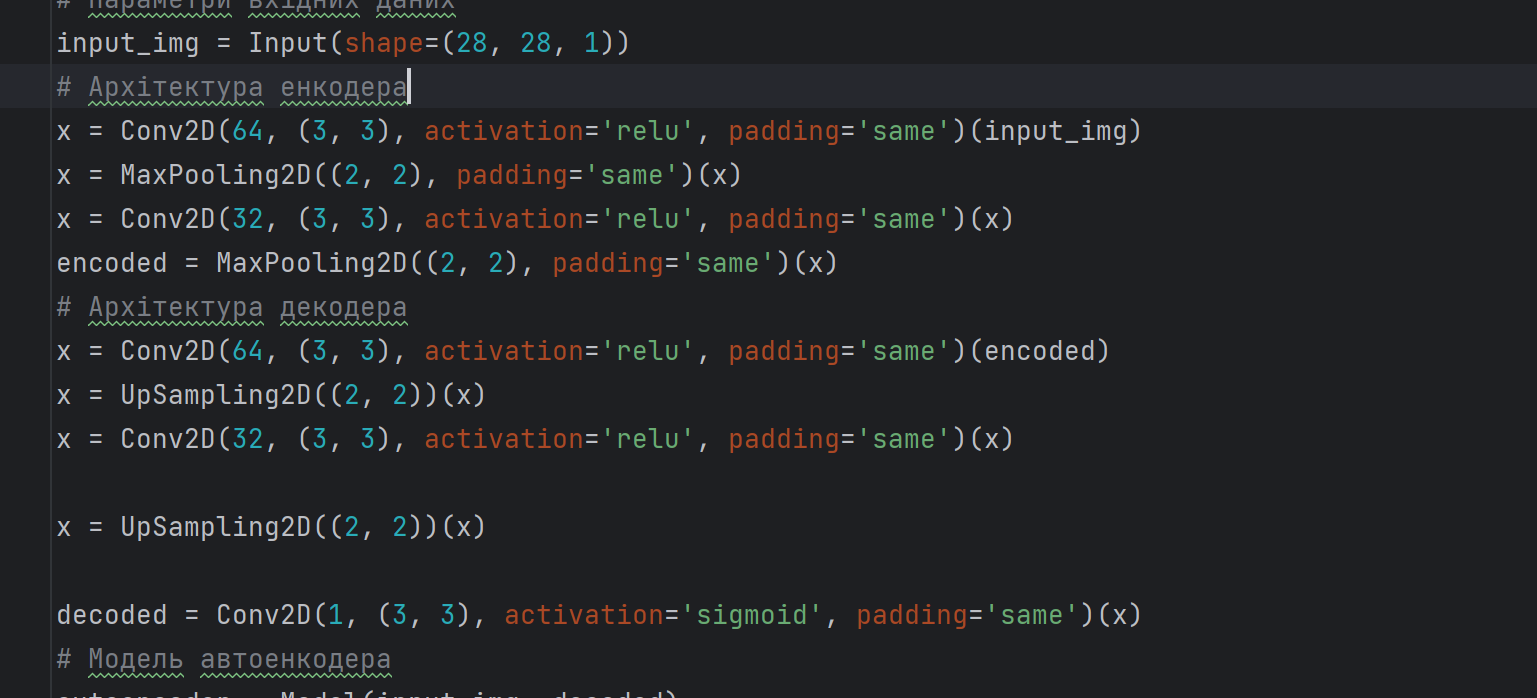
збільшимо кількість згорткових шарів, згорткових збільшимо розміри шарів

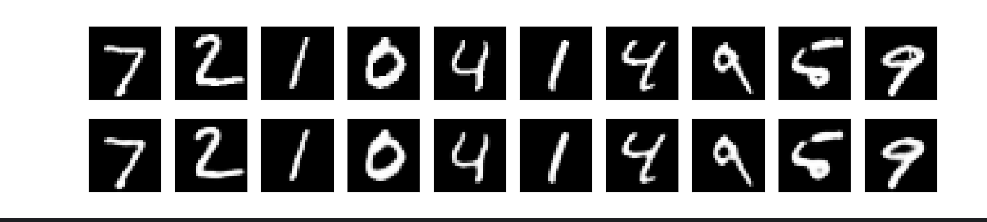


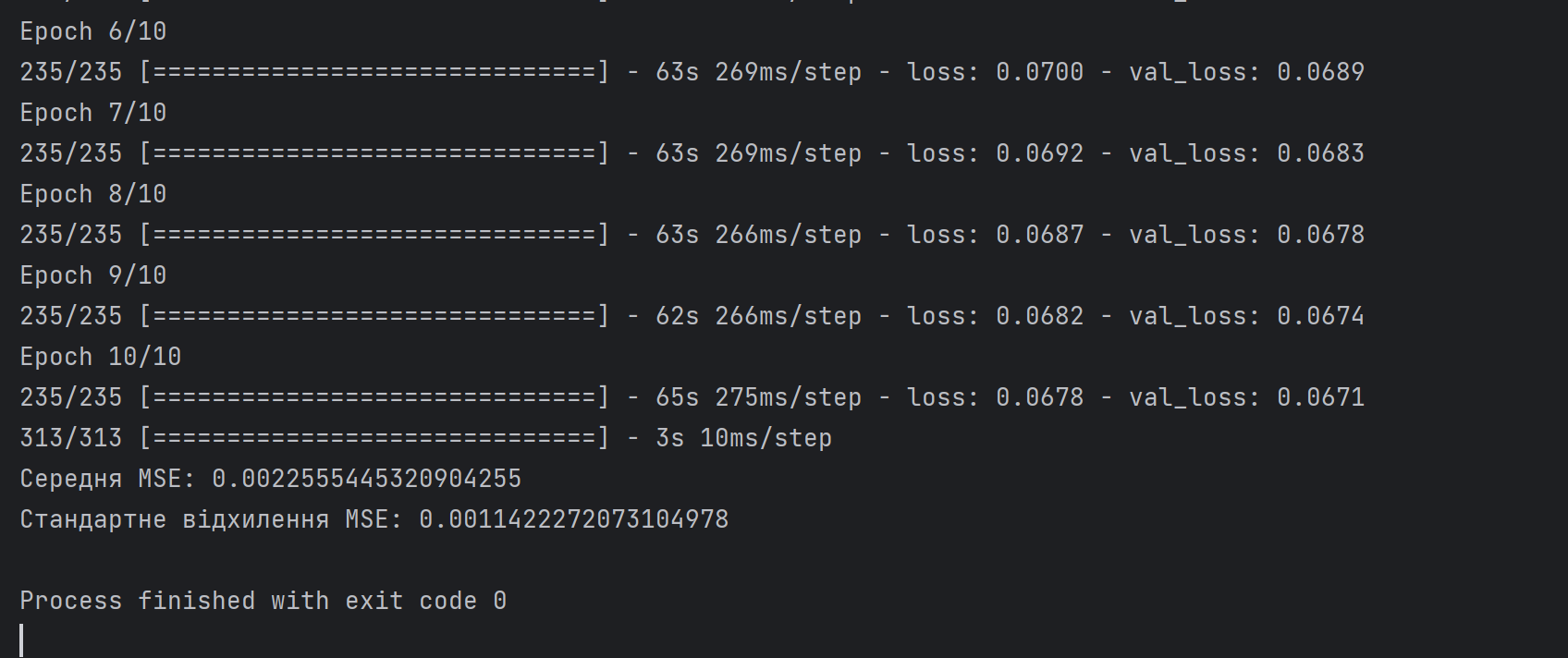


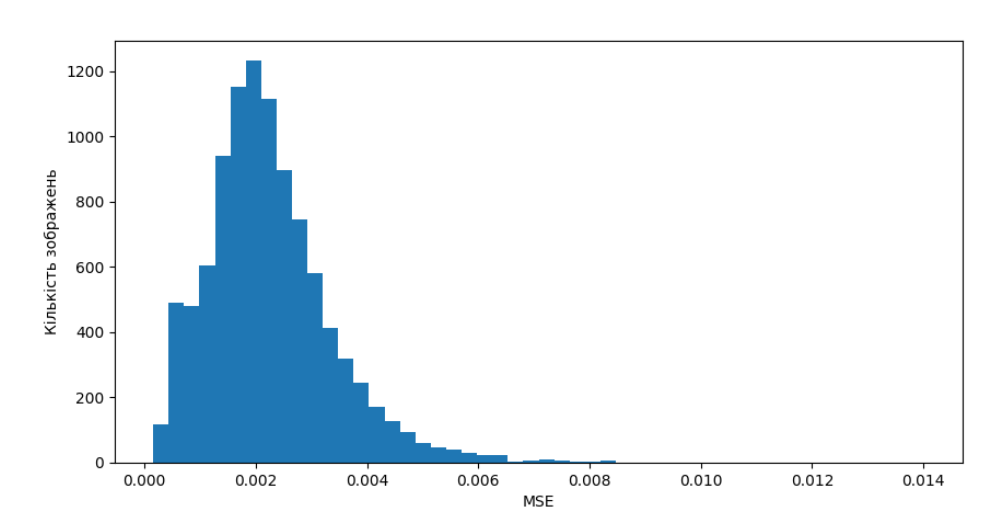
  

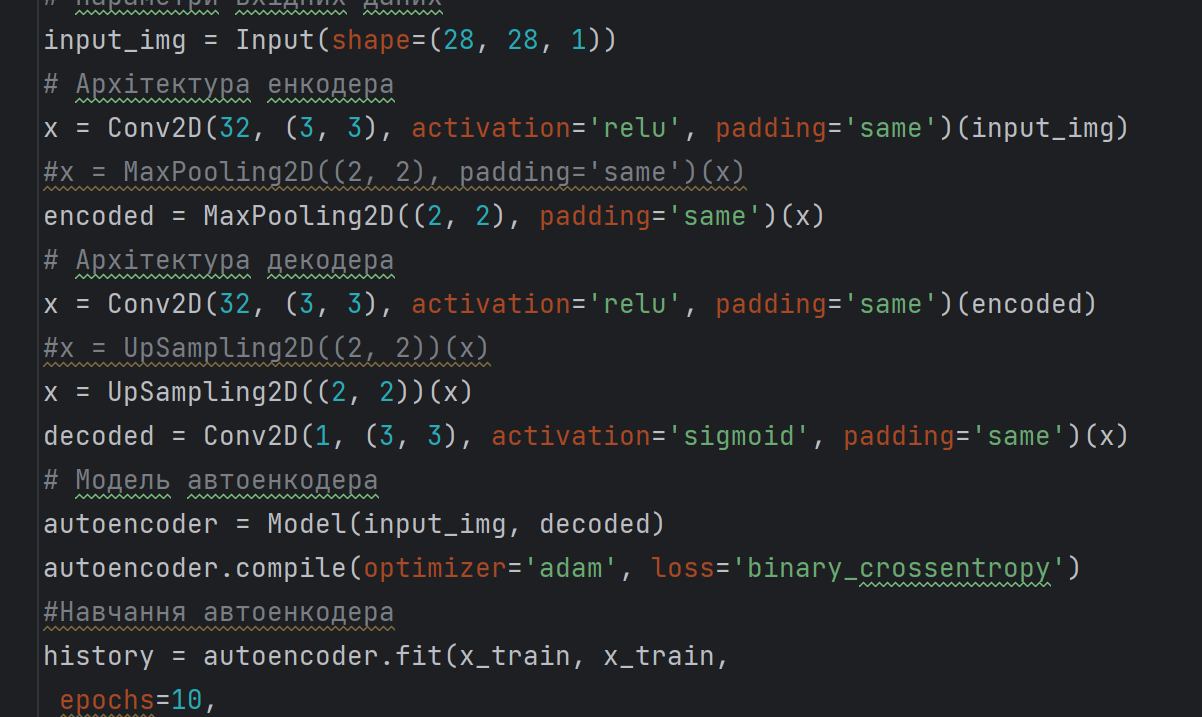
збільшимо розміри згорткових шарів

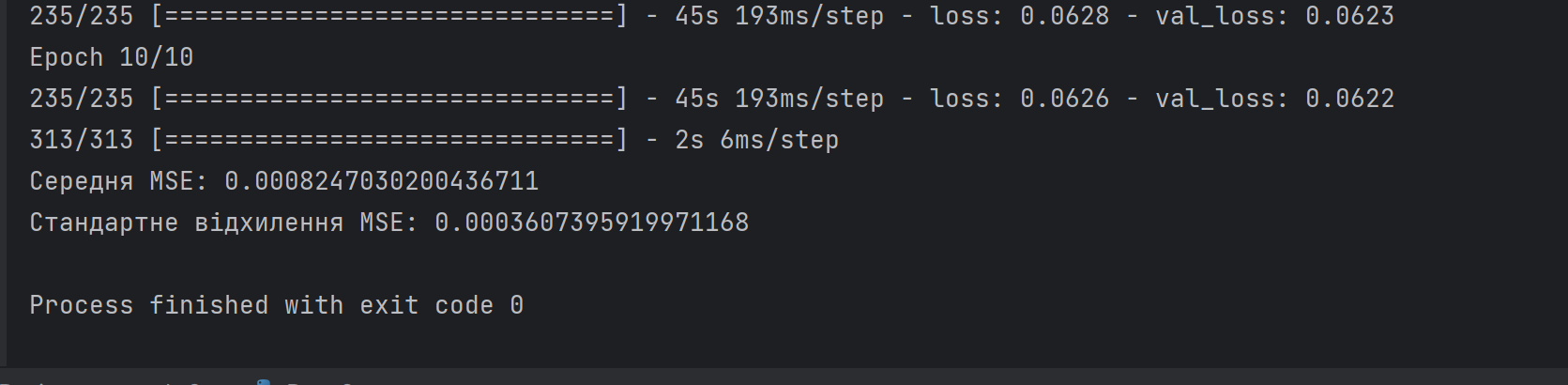


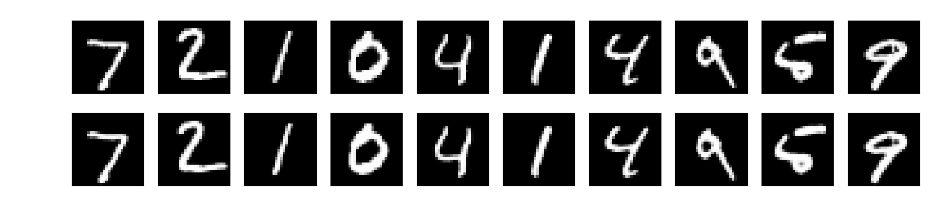


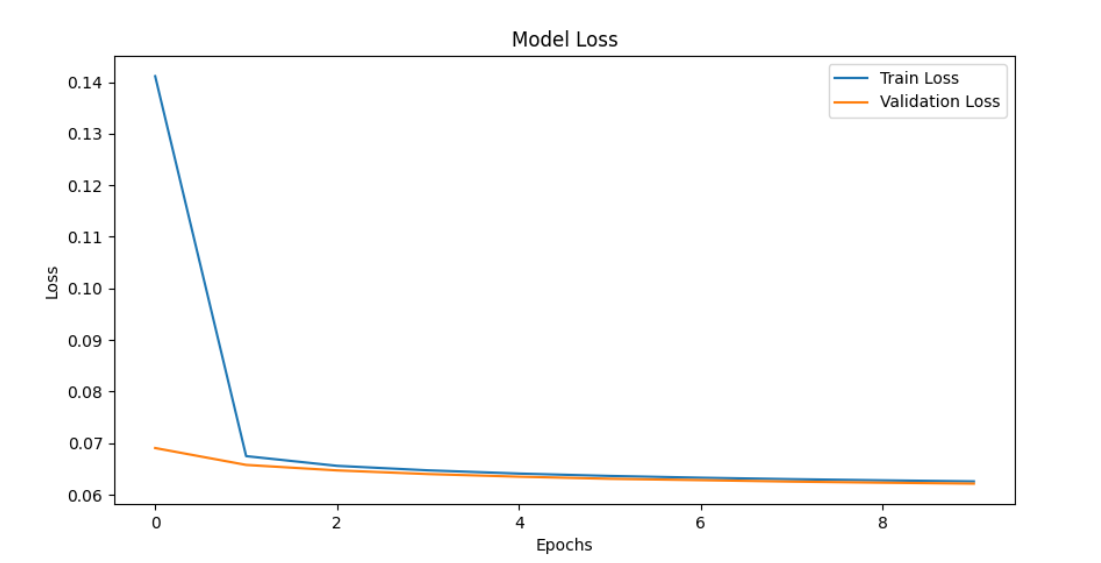


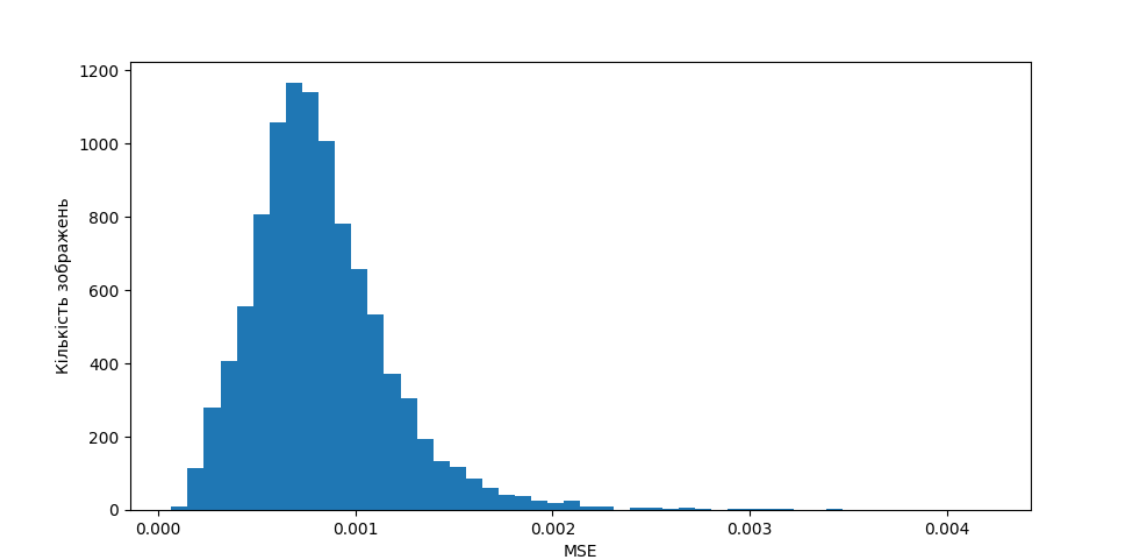


зменшимо кількість згорткових та пулінгових шарів

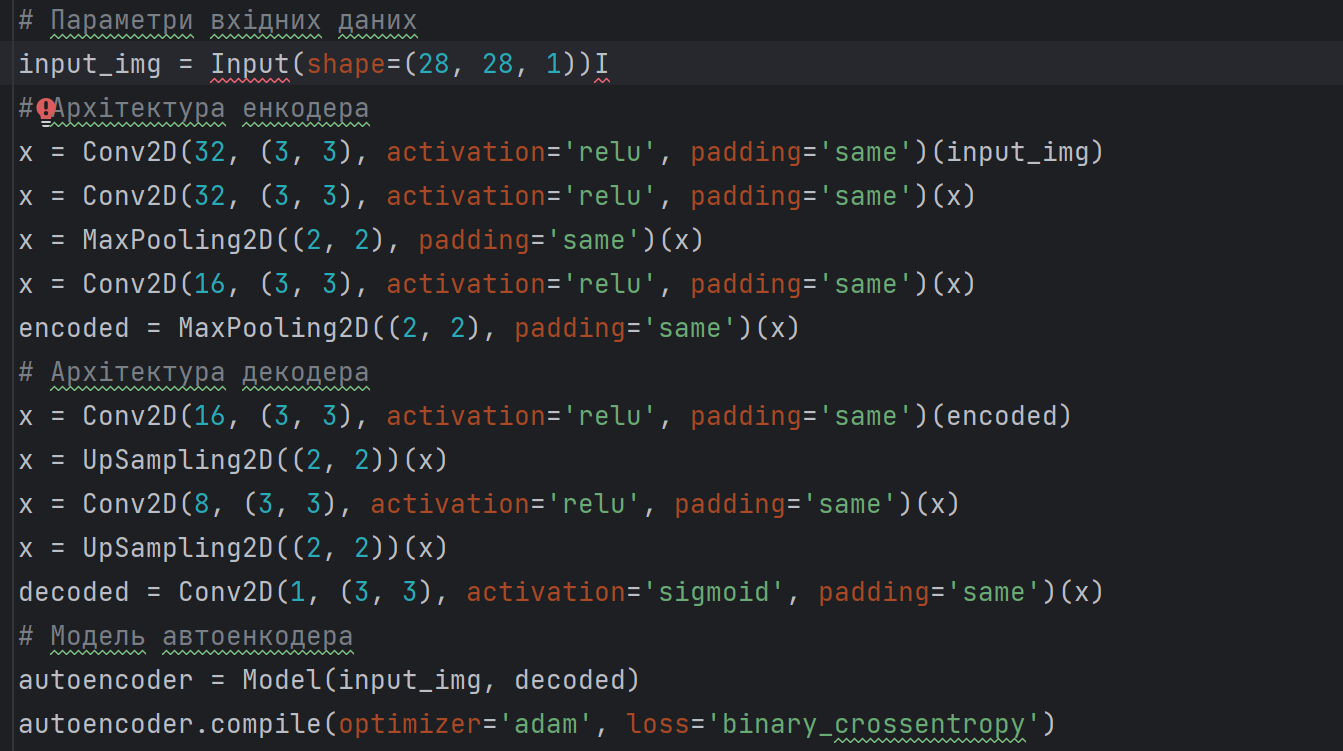


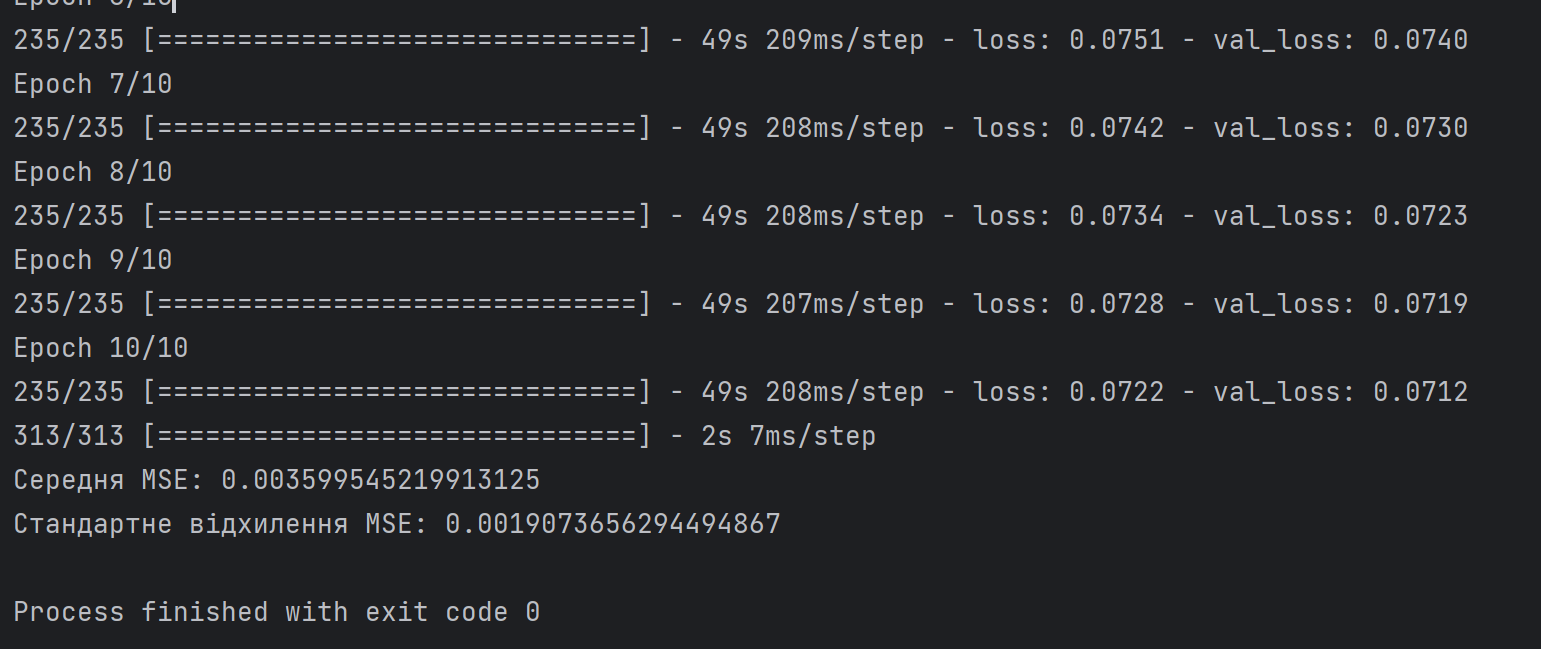


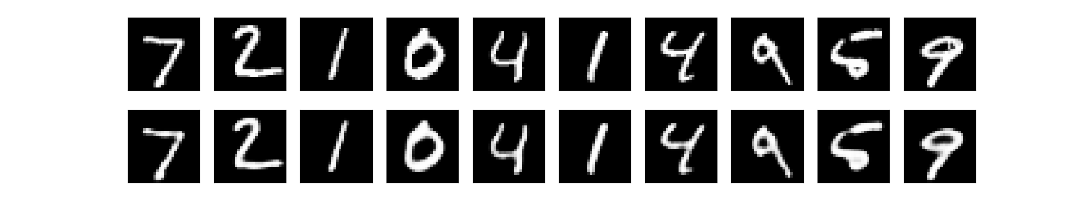


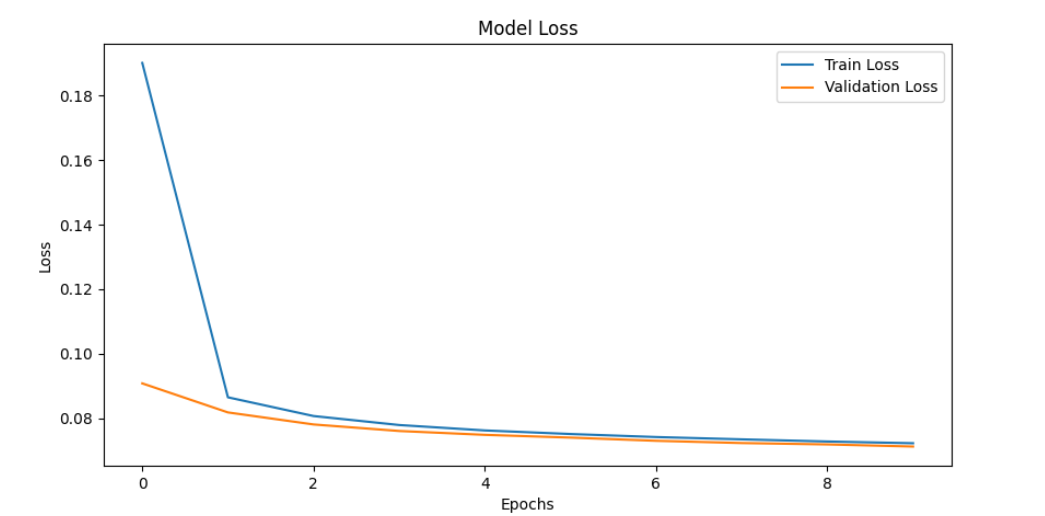


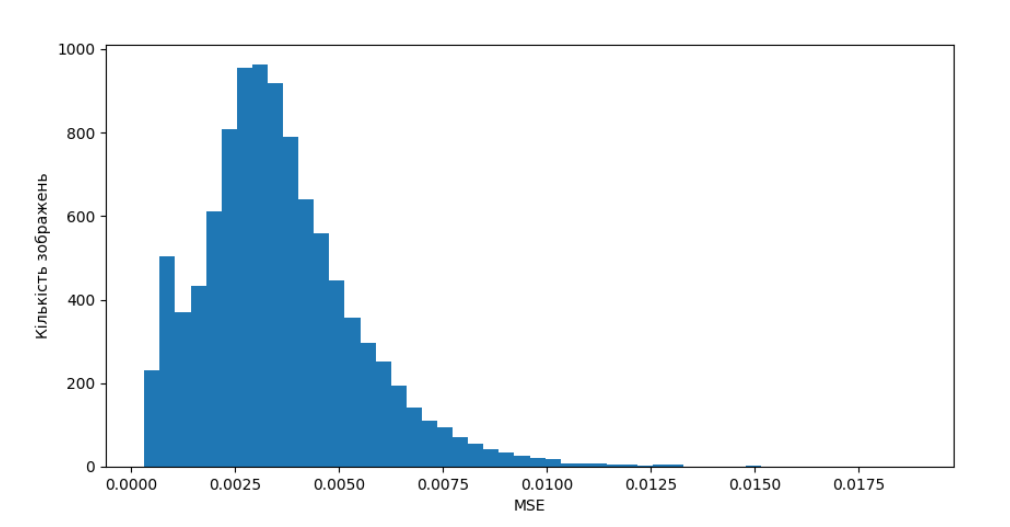
Збільшимо кількість згорткових та зменшимо кількість пулінгових шарів в архітектурі енкодера, та зменшимо розміри шарів в архітектурі декодера





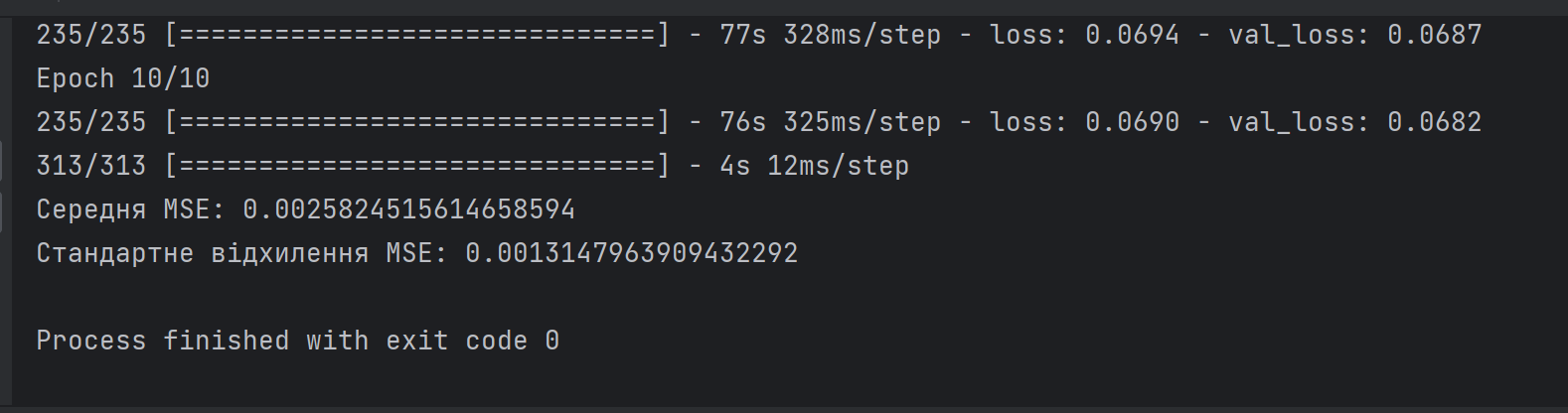


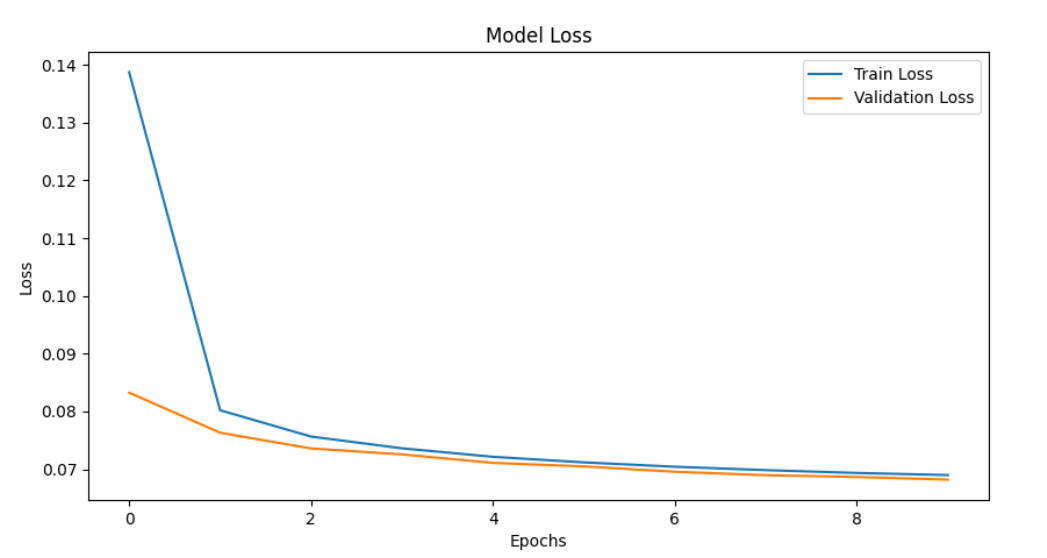
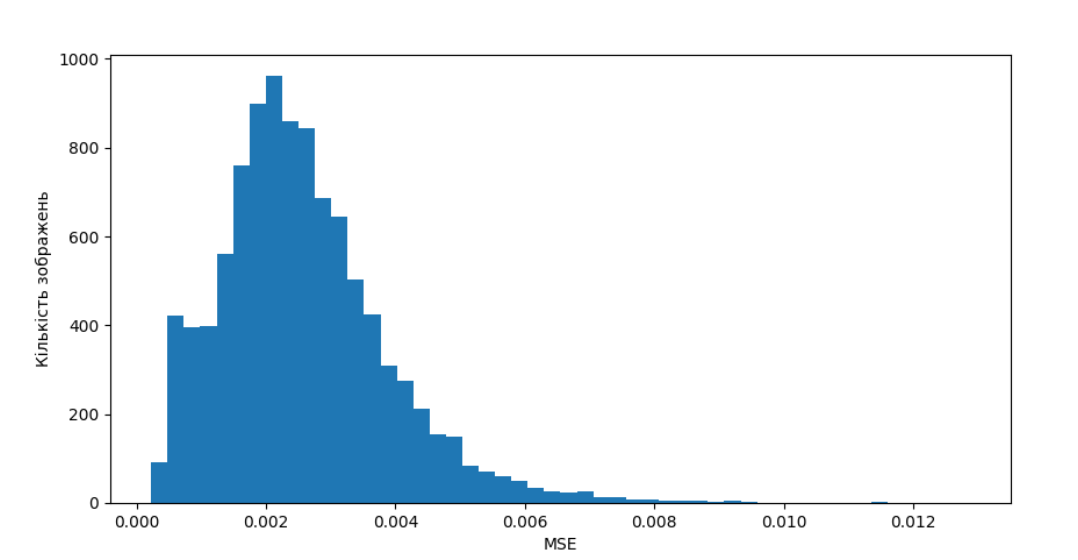


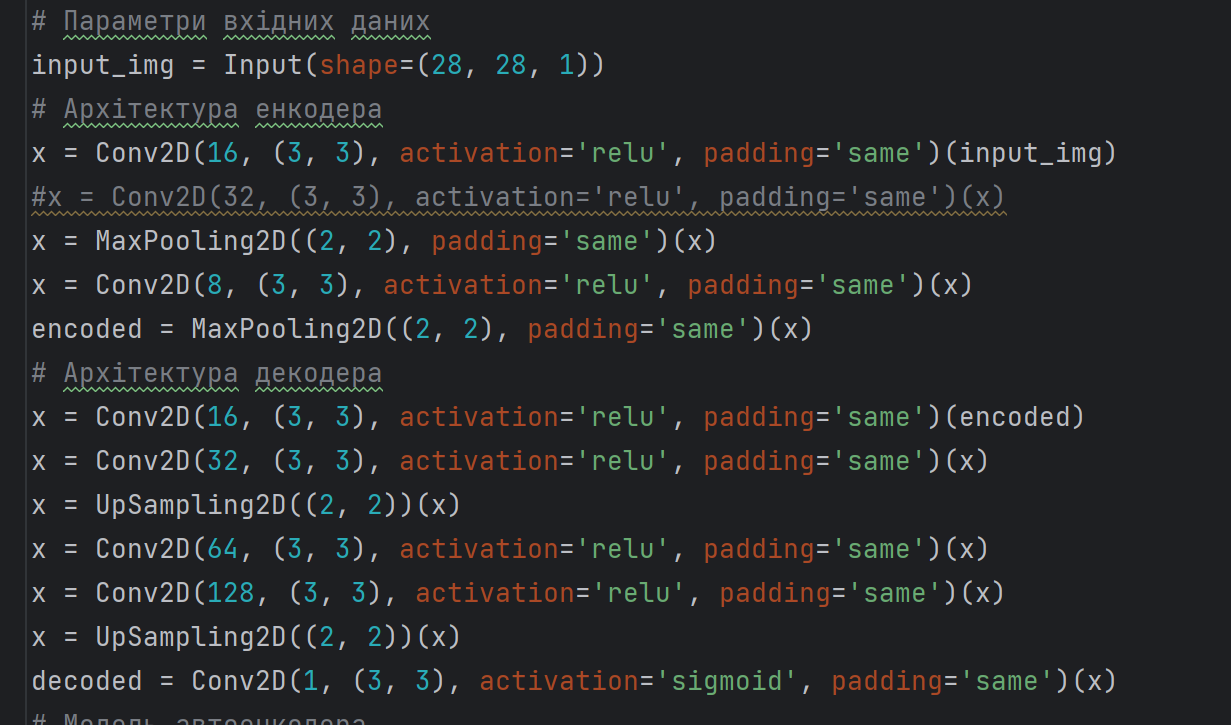


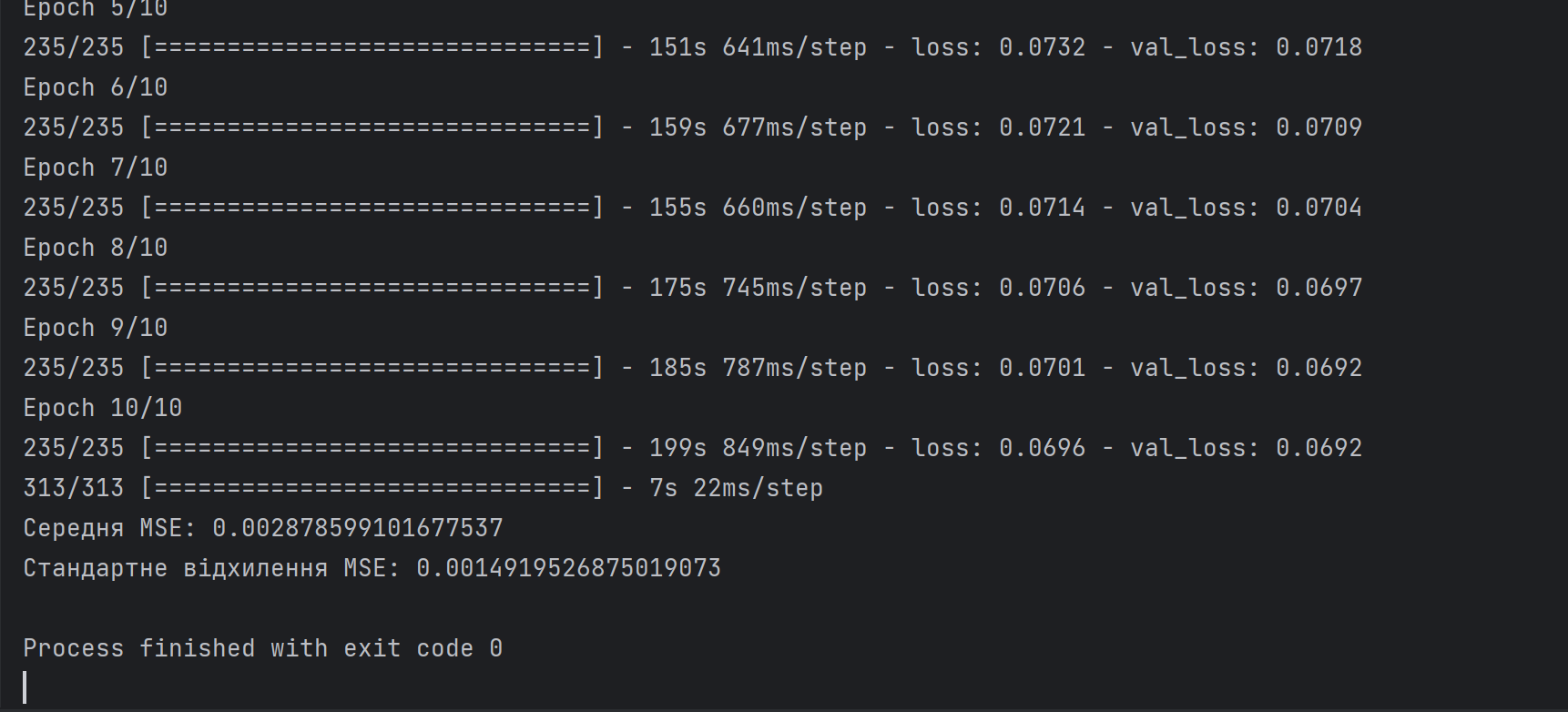
Збільшимо кількість згорткових шарів в архітектурі декодера



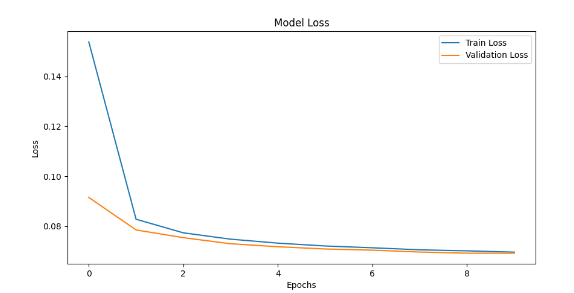


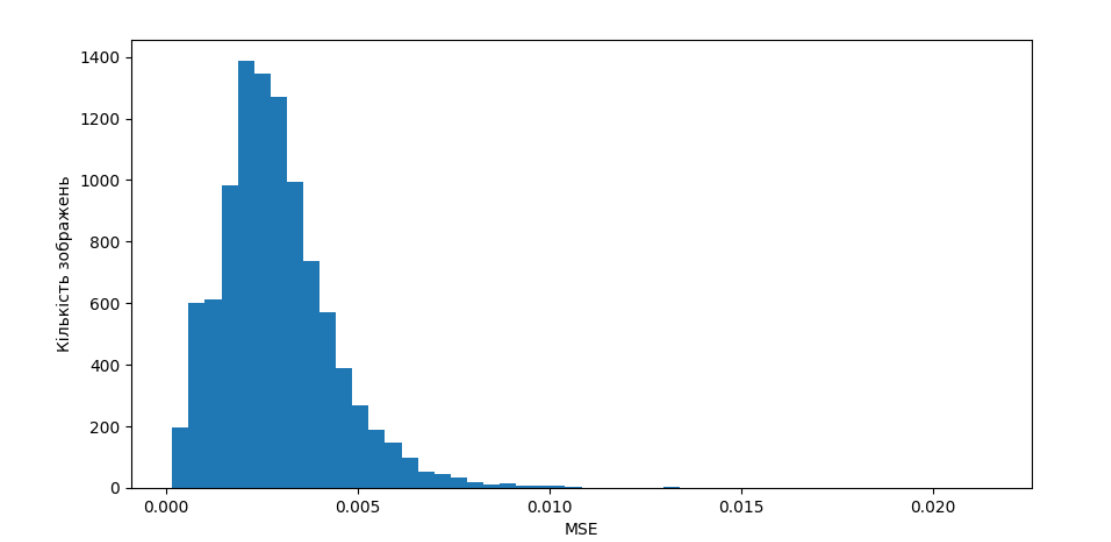
  

Збільшимо кількість згорткових шарів в архітектурі декодера, та зменшимо розміри згорткових шарів в архітектурі енкодера



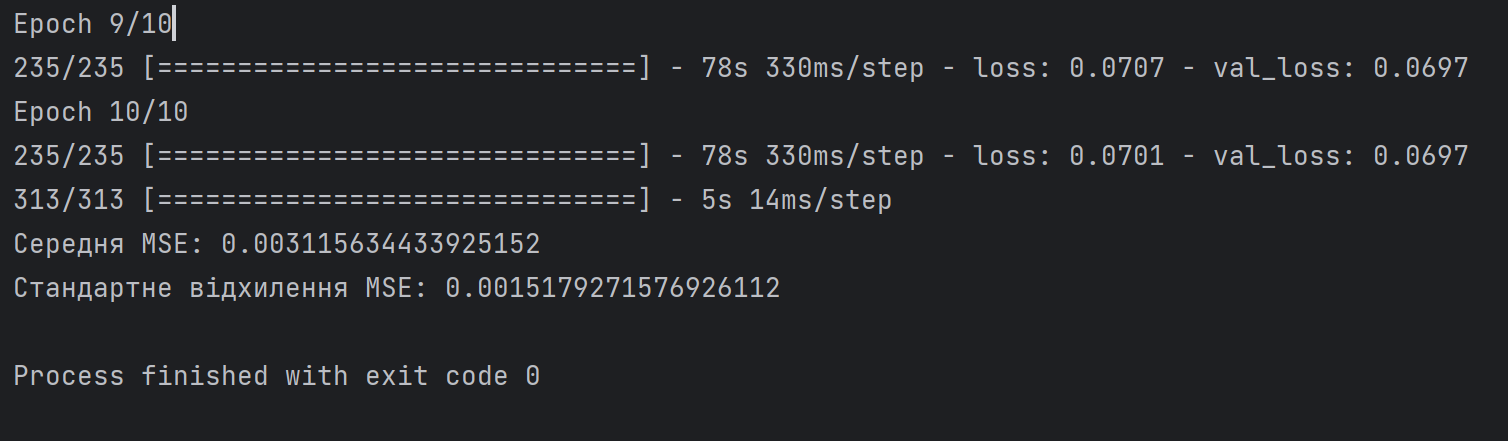


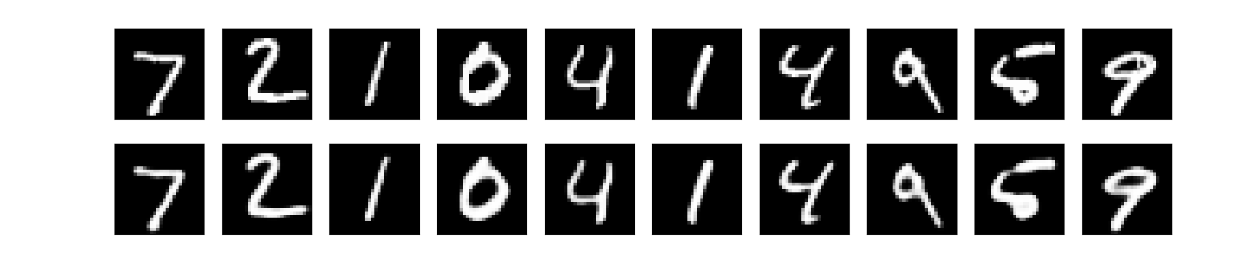


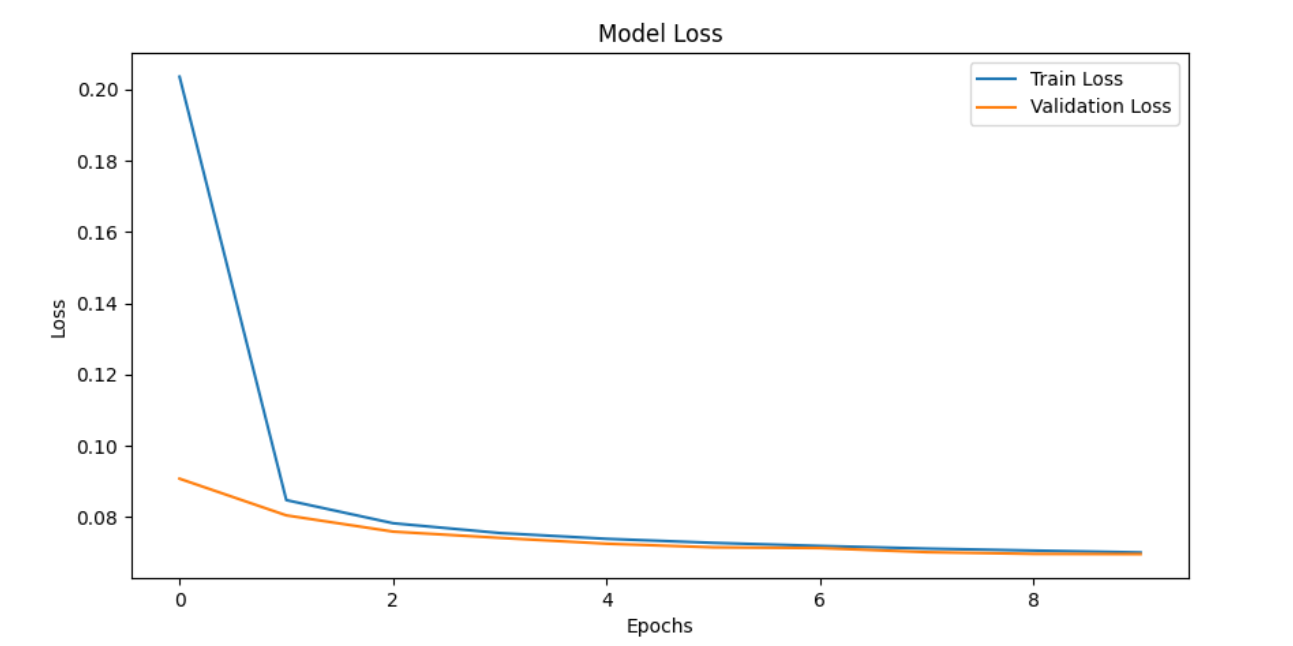


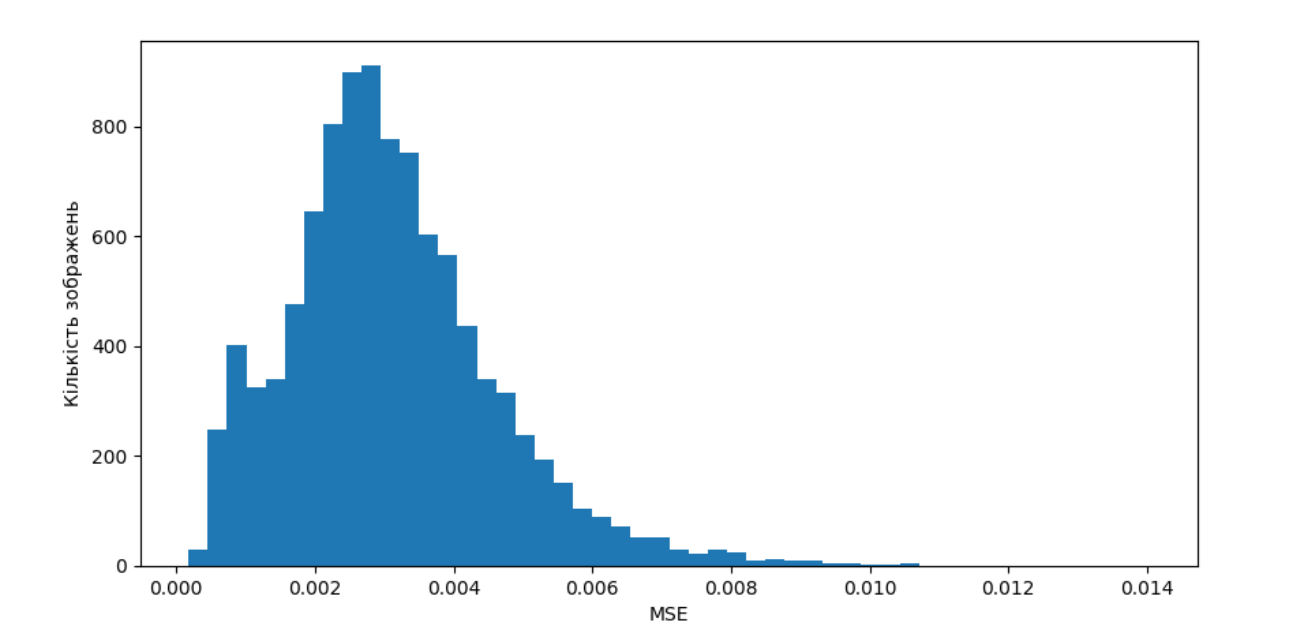
Додамо до архітектури енкодера та декодера додаткові шари



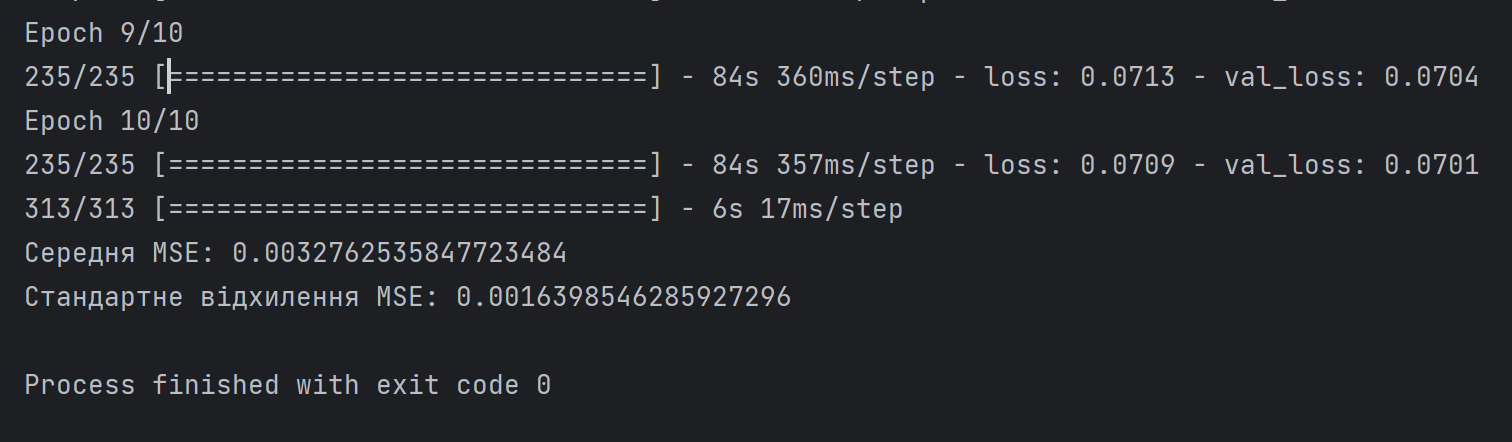
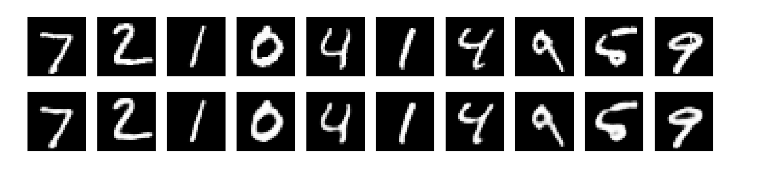
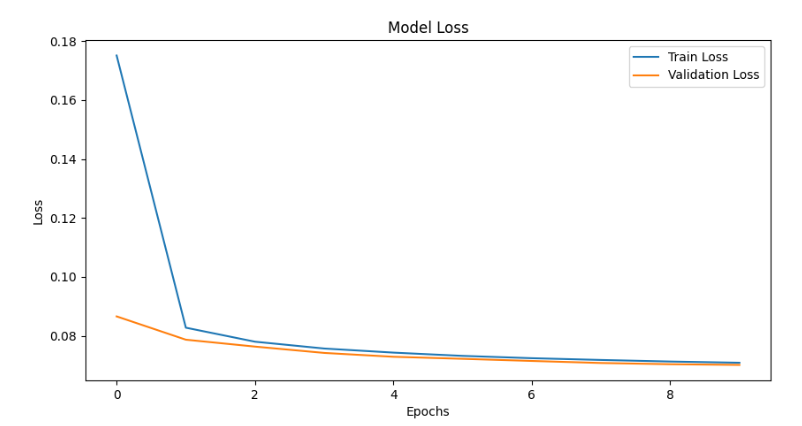
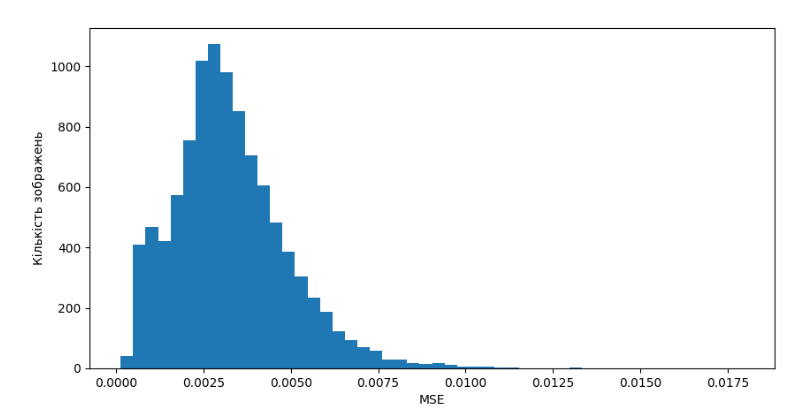






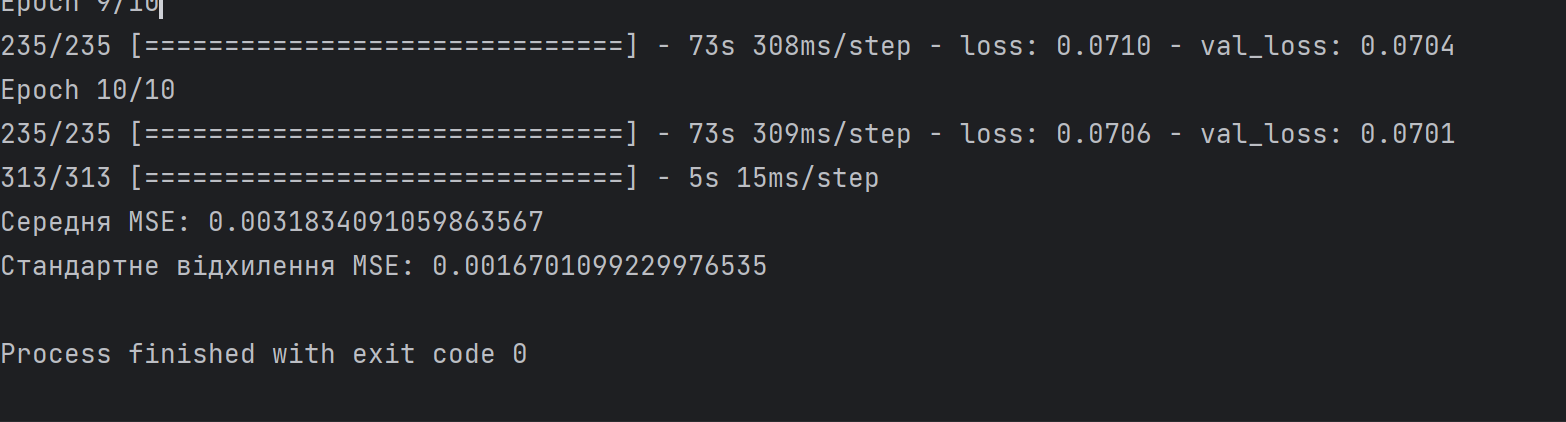


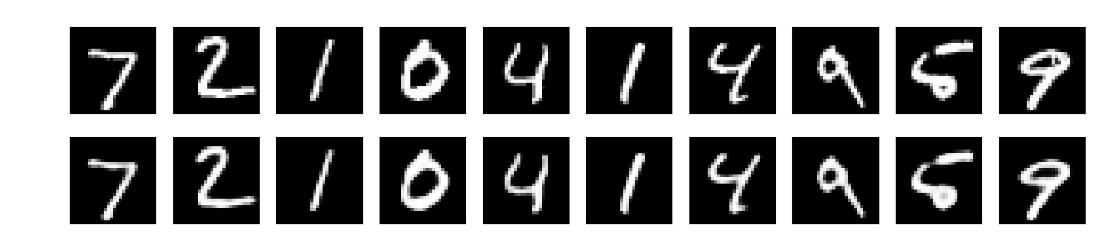
Переставимо додаткові шари у архітектурі

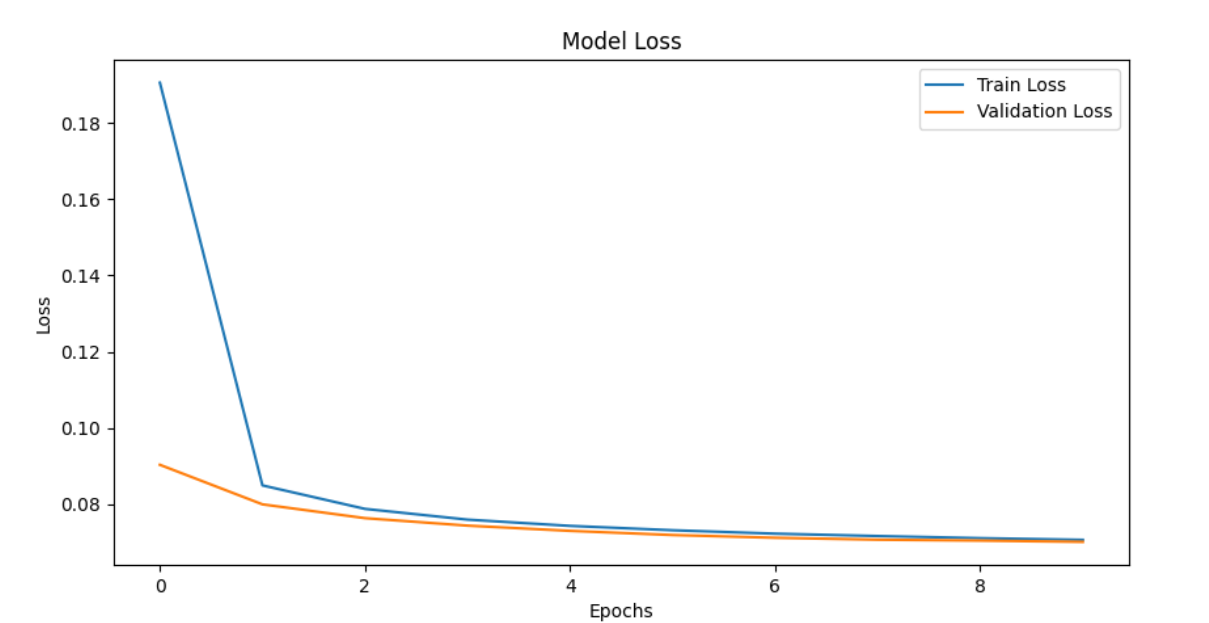
    

Видалимо додатковий шар з декодеру



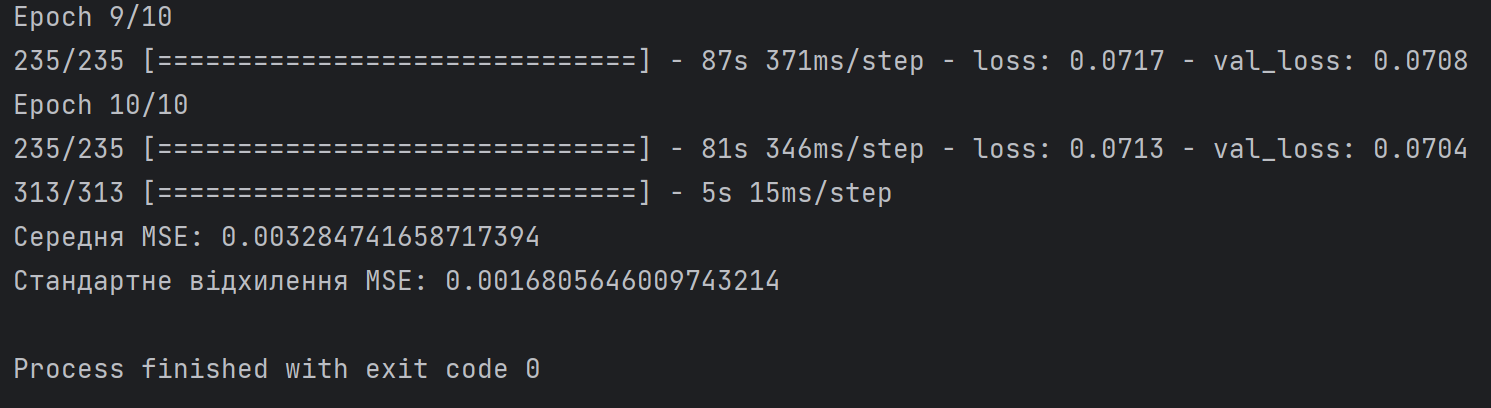


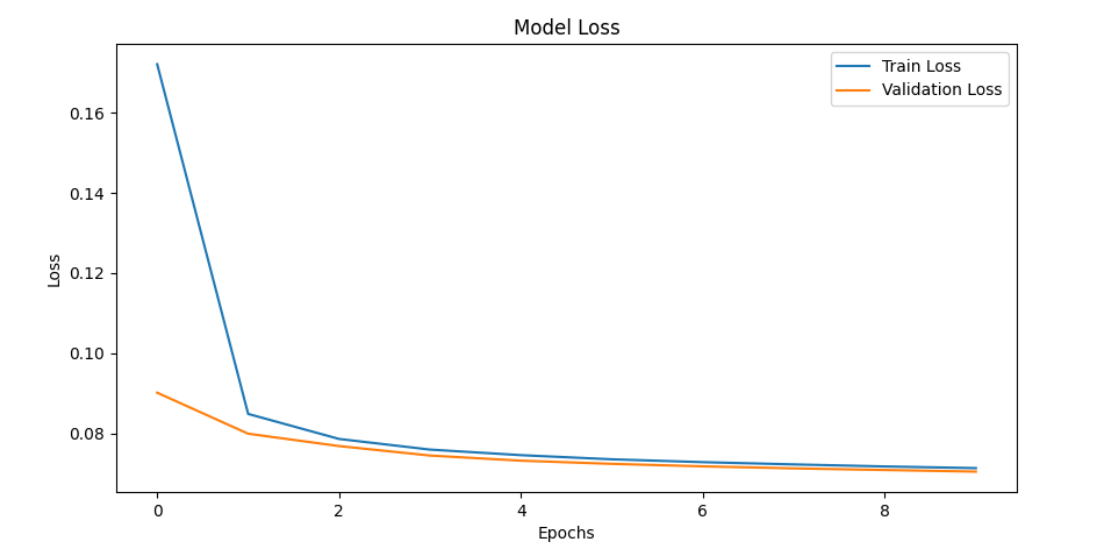
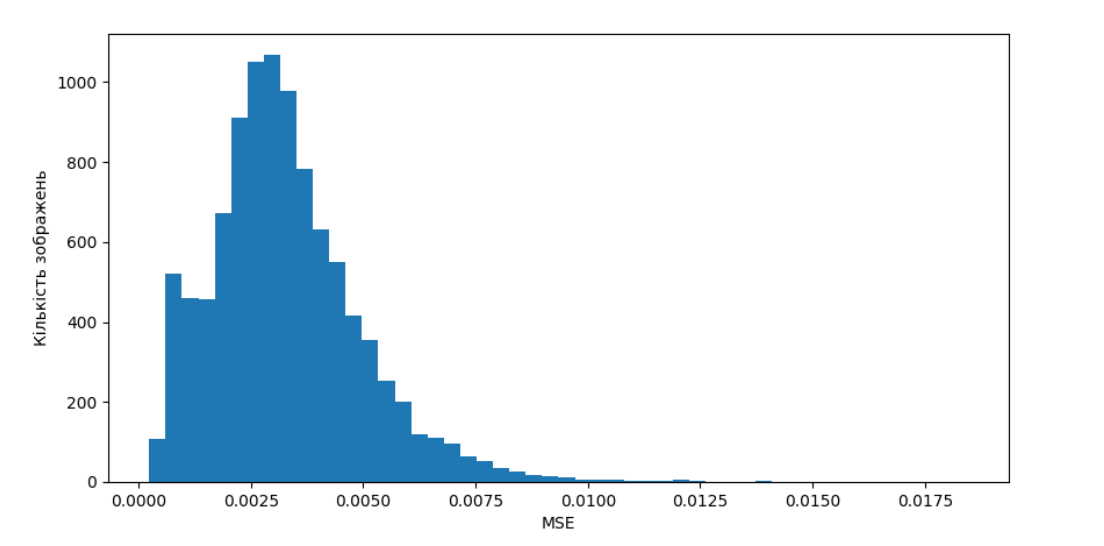






Видалимо додатковий шар з енкодеру

Під час проведення експерементів над архітектурою енкодера та декодера, можно зробити висновок, що під час збільшення кількость та розмірів згорткових шарів енкодера та декодера покращили результат (Середня MSE = 0.002658, Стандартне відхилення MSE = 0.00139 – додавання нових згорткових шарів, Середня MSE = 0.002256, Стандартне відхилення MSE = 0.00114 – додавання нових згорткових шарів та збільшення їх розмірів), але додавання згорткових шарів з маленьким розміром погіршують результати (Середня MSE = 0.004164, Стандартне відхилення MSE = 0.002123). Також, додавання до декодера додаткових згорткових шарів покращився результат (Середня MSE = 0.002583, Стандартне відхилення MSE = 0.001315 – збільшення декодеру). Для покращення можна рекомендується використати: збільшити розмірність моделі; використовувати різні функції активації для прихованих шарів; використовувати меншу функцію втрати для забезпечення того, що автоенкодер навчається ефективно стискати та відтворювати вхідні дані; додавати шуму може зробити автоенкодер менш чутливим до шуму в вхідних даних та поліпшити його здатність узагальнення; експериментуйте із розмірністю коду; збільшити кількості епох та зменшення швидкості навчання може допомогти в уникненні перенавчання та покращити загальну ефективність.

**Контрольні питання**

1. **Що таке автоенкодер?**

Автоенкодер – це тип нейронної мережі, яка має за мету ефективно кодувати вхідні дані таким чином, щоб потім можна було їх відновити. Основна ідея полягає в тому, щоб навчити мережу стискаємо представляти вхідний сигнал у вигляді компактного внутрішнього представлення (зазвичай з меншим розміром) і потім декодувати його назад до близько оригінального або точного вихідного сигналу

1. **Для яких задач можуть бути корисними автоенкодери?**

Автоенкодери можуть бути корисні для ряду задач:

• Зменшення розміру даних (Dimensionality Reduction): Автоенкодери можуть використовуватися для зменшення розміру даних, аналогічно до методу головних компонент (PCA), але в нелінійних просторах.

• Зменшення шуму (Denoising): Автоенкодери можуть бути навчені відновлювати зашумлені зображення, видаляючи шум.

• Виявлення аномалій: Якщо автоенкодер навчений на "нормальних" даних, він може мати проблеми із відтворенням аномальних даних, що робить його корисним для виявлення аномалій.

• Генерація нових даних: В комбінації з іншими техніками, такими як VAE (варіаційний автоенкодер), автоенкодери можуть генерувати нові, раніше не бачені приклади даних.

1. **Які основні частини автоенкодера?**

Автоенкодер складається з двох основних частин:

• Енкодер: Ця частина мережі приймає вхідний сигнал і перетворює його у внутрішнє, зазвичай стисле, представлення.

Задача: Змінити розмір вхідного сигналу, зазвичай стискаючи його до меншої розмірності.

Структура: Енкодер складається з одного або декількох шарів (зазвичай повнозв'язних, хоча для зображень можуть бути використані згорткові шари). Чим глибше шар, тим менша його розмірність, що дозволяє стиснути вхідні дані. Вивід: Внутрішнє представлення даних, іноді називається "латентний" або "прихований" простір.

• Декодер: Декодер бере внутрішнє представлення, надане енкодером, і перетворює його назад у вихідний сигнал, який має бути якомога ближчим до оригінального входу.

Задача: Відновити дані з внутрішнього представлення, яке створив енкодер, до їх оригінальної форми (або до форми, яка їй найбільше відповідає).

Структура: Декодер, як правило, має архітектуру, зворотну архітектурі енкодера. Якщо енкодер зменшує розмірність даних через кожен шар, то декодер збільшує їх. Вивід: Дані, що були відтворені з внутрішнього представлення

1. **Яка мета функції втрат у автоенкодері?**

Мета функції втрат у автоенкодері – це мінімізувати різницю між вхідним сигналом та його відтвореним вихідним сигналом після проходження через енкодер та декодер. Тобто, функція втрат оцінює, наскільки добре автоенкодер може відтворити вхідні дані після того, як вони були стиснені і розпаковані назад.

1. **Які можуть бути проблеми при тренуванні автоенкодерів?**

Перенавчання (Overfitting): Автоенкодер може занадто добре відтворювати тренувальний набір даних, втрачаючи здатність ефективно кодувати нові, невідомі дані.

• Недостатнє кодування (Underfitting): Якщо архітектура автоенкодера занадто проста, він може не вміти відтворювати вхідні дані ефективно.

• Втрата інформації: Оскільки автоенкодери намагаються стиснути вхідні дані, можлива втрата частини інформації при кодуванні.

• Занадто добре відтворення: Якщо автоенкодер відтворює вхідні дані занадто добре без значущого стискання, це може бути індикатором того, що кодування не вдалося.

• Труднощі з оптимізацією: В залежності від архітектури та даних, градієнти можуть "вимирати" або ставати занадто великими

1. **Що таке bottleneck encoding в контексті автоенкодерів?**

Bottleneck encoding (пляшкове шийко) відноситься до тієї частини автоенкодера, де дані стискаються до найбільш компактного представлення. Це є середнім шаром між енкодером та декодером і відіграє ключову роль у визначенні ефективності кодування. Чим менший розмір "пляшкового шийка ", тим менше інформації може бути закодовано, але це також може забезпечити більше стискання даних

1. **Як розрізнити "добрий" автоенкодер від "поганого" з точки зору якості відтворення?**

Розрізнити "добрий" автоенкодер від "поганого" можна завдяки таким параметрам як:

* Якість відтворення: На перший погляд, "добрий" автоенкодер відтворює вхідні дані з високою точністю. Порівняння вхідних та відтворених зображень може допомогти визначити якість.

• Втрати: Низький рівень функції втрат може вказувати на "добрий" автоенкодер, але необхідно також перевірити на перенавчання.

• Здатність до узагальнення: "Добрий" автоенкодер ефективно кодує нові, раніше невідомі дані.

• Специфікація завдання: В деяких завданнях, таких як видалення шуму, "добрий" автоенкодер не просто відтворює вхідні дані, але й покращує їх.

• Візуальна інтерпретація: Це особливо корисно для зображень. Якщо відтворене зображення виглядає чітким і відповідає оригіналу, це гарний знак.

• Стійкість до шуму: "Добрий" автоенкодер зможе відновити оригінальні дані, навіть якщо до вхідних даних було додано шу