

## **Лабораторна робота №3**

### **‘Запуск проєкту за допомогою Docker та аналіз його роботи’**

**Виконав студент групи ТК-41  
Соловей Євгеній Романович**

**2025**

## Кроки виконання задачі

Було встановлено Docker для виконання даної лабораторної роботи. Також потрібно встановити образ з Docker Hub за допомогою команди `docker pull tensorflow/tensorflow:latest-jupyter`. Тепер можемо запустити Docker image за допомогою команди `docker run -it -p 8888:8888 -v <Path to our dir>:/tf tensorflow/tensorflow:latest-jupyter`, після цього запуститься образ на порті 8888 та ми зможемо виконати наш код.

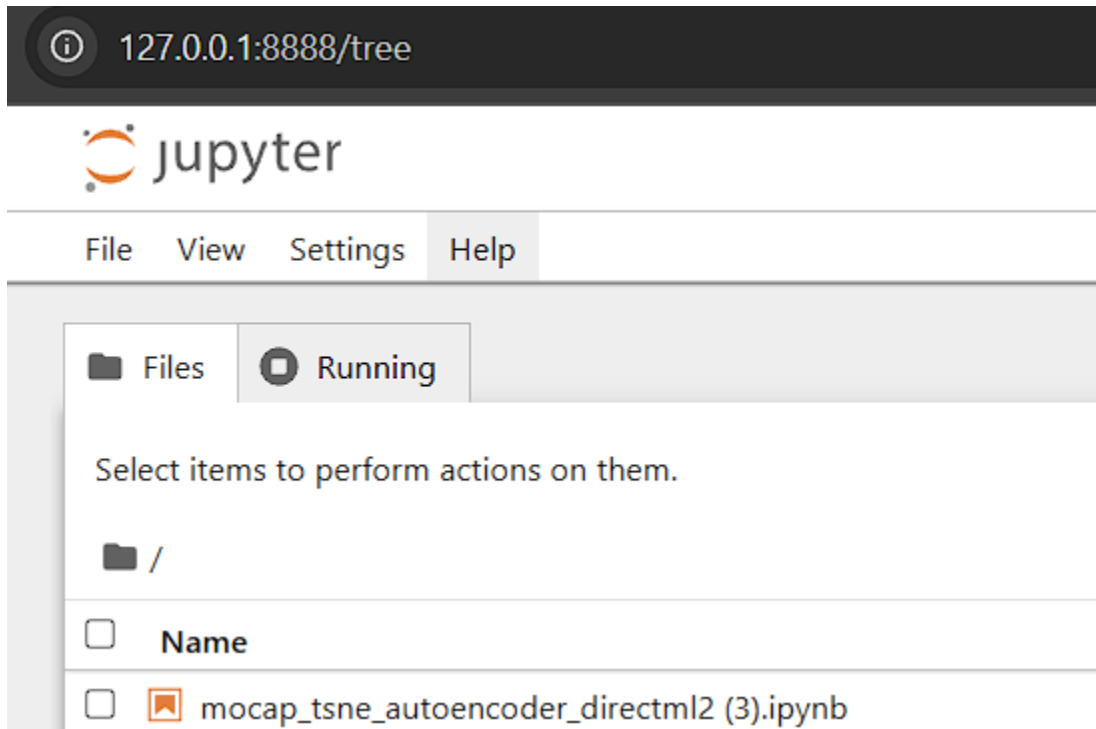


Рис. 1. Jupyter Notebook запущений на 8888 порту.

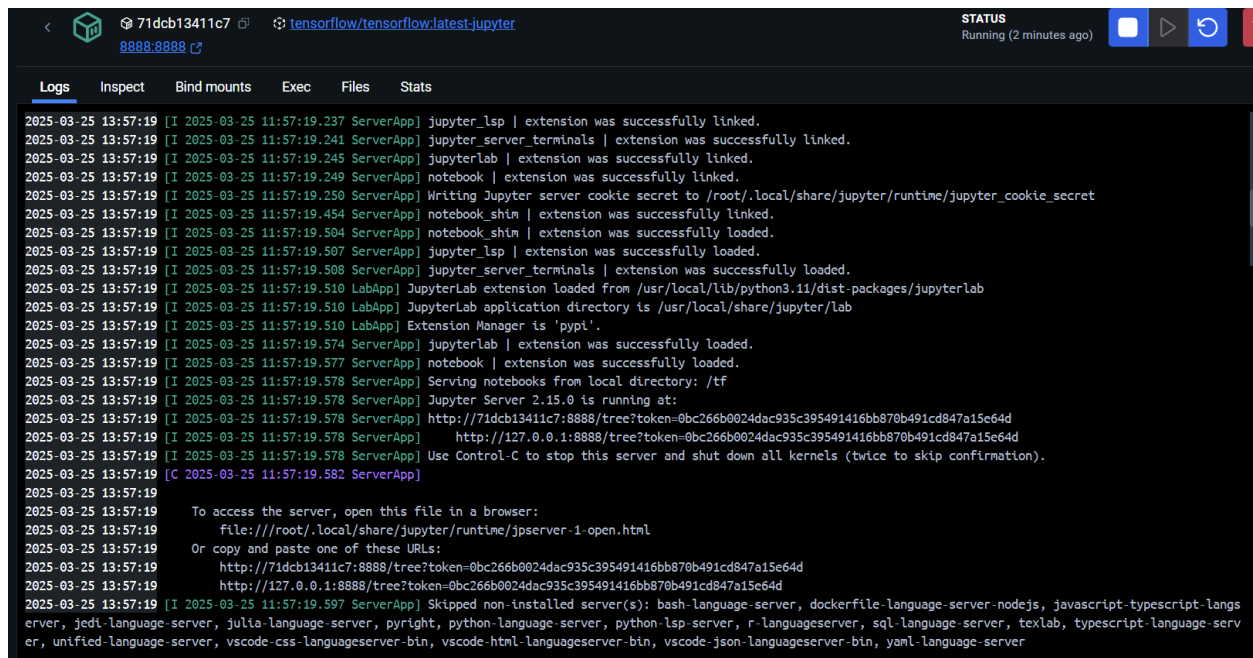
The image shows a terminal window of a Docker container. The top bar indicates the container ID '71dcb13411c7' and the image 'tensorflow/tensorflow:latest-jupyter'. The status is 'Running (2 minutes ago)'. The terminal output shows a series of logs for JupyterLab extensions being loaded successfully, including 'jupyter\_lsp', 'jupyter\_server\_terminals', 'jupyterlab', 'notebook', and 'notebook\_shim'. It also shows the JupyterLab application directory and the extension manager. The terminal ends with a message to access the server via a browser, providing a URL: 'http://127.0.0.1:8888/tree?token=0bc266b0024dac935c395491416bb870b491cd847a15e64d'.

Рис. 2. Запущений Docker контейнер

## Аналіз роботи коду

Цей код виконує такі основні завдання:

1. Імпортує необхідні бібліотеки: TensorFlow, scikit-learn, NumPy, Matplotlib та інші.
2. Завантажує та обробляє дані:
  - a. Завантажує shared.pickle з використанням pickle.
  - b. Використовуємо TruncatedSVD для зменшення розмірності до 10 компонентів.
  - c. Кластеризує дані за допомогою MiniBatchKMeans.
3. Навчання класифікаторів:
  - a. Реалізує функції train\_model і train\_model2, які навчають та перевіряють точність класифікаторів.
  - b. Використовує train\_test\_split для поділу вибірки.
4. Автоенкодер на TensorFlow/Keras:
  - a. Визначає архітектуру автоенкодера (create\_dense\_ae).
  - b. Кодує дані у 2-вимірний простір.

- c. Використовує adam як оптимізатор та функцію втрат binary\_crossentropy.
  - d. Використовує GPU для навчання(в моєму випадку використовується CPU).
5. Візуалізація результатів:
- a. Використовує TruncatedSVD для зменшення розмірності перед кластеризацією.
  - b. Будує графіки розташування кластерів. Порівнює кодування вхідних даних у двовимірному просторі.

Після виконання команди `autoencoder.summary()` отримає інформацію про архітектуру моделі та можемо зробити висновок, що: Model: "autoencoder", вхідний шар з 10 нейронами, прихований шар з 2 нейронами та має 4442 параметри та вихідний шар, що має 4450 параметрів.

**Model: "autoencoder"**

Layer (type)	Output shape	Param #
input_layer (InputLayer)	(None, 10)	0
encoder (Functional)	(None, 2)	4,442
decoder (Functional)	(None, 10)	4,450

**Total params: 8,892** (34.73 KB)

**Trainable params: 8,892** (34.73 KB)

**Non-trainable params: 0** (0.00 B)

*Рис. 3. Інформація отримана в результаті виконання команди `autoencoder.summary()`*

Запускаємо навчання автоенкодера:

```
autoencoder.fit(train_x, train_x,
                epochs=500,
                batch_size=50,
```

`shuffle=True,`

`validation_data=(valid_x, valid_x))`

`epochs` - кількість епох

`batch_size` - розмір пакета, тобто скільки зразків обробляється за один раз

`shuffle` - перемішування даних перед кожною епохою

`validation_data` - валідаційні дані

Цей код навчає автоенкодер на `train_x`, щоб він міг відтворювати ці дані, а також контролює якість навчання на `valid_x`.

Під час навчання в 500 епох можемо зробити висновок, що виконання першої ітерації займає більше часу чи наступні, а всі наступні ітерації займають в середньому 14-15 мілісекунд.

```
Epoch 1/500
5/5 ————— 2s 59ms/step - accuracy: 0.0294 - loss: 0.8282 - val_accuracy: 0.0400 - val_loss: 0.4411
Epoch 2/500
5/5 ————— 0s 18ms/step - accuracy: 0.2776 - loss: 0.5646 - val_accuracy: 0.5333 - val_loss: 0.2062
Epoch 3/500
5/5 ————— 0s 14ms/step - accuracy: 0.4799 - loss: 0.2530 - val_accuracy: 0.5333 - val_loss: -0.0211
Epoch 4/500
5/5 ————— 0s 15ms/step - accuracy: 0.4680 - loss: 0.0542 - val_accuracy: 0.5333 - val_loss: -0.2393
Epoch 5/500
5/5 ————— 0s 15ms/step - accuracy: 0.4727 - loss: -0.1888 - val_accuracy: 0.5333 - val_loss: -0.4502
Epoch 6/500
5/5 ————— 0s 15ms/step - accuracy: 0.4563 - loss: -0.3686 - val_accuracy: 0.5333 - val_loss: -0.6641
Epoch 7/500
5/5 ————— 0s 14ms/step - accuracy: 0.4444 - loss: -0.5130 - val accuracy: 0.5333 - val loss: -0.8814
```

*Рис. 4. Ітерації*