Зображення, що містить текст

Автоматично згенерований опис

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Лабораторна робота №5**

Прикладні задачі машинного навчання

**Тема:** Проектування та навчання штучної нейронної мережi для задач

класифiкацiї

Виконав Перевірив:

студент групи ІП-11: Нестерук А. О

Панченко С. В.

Київ 2023

Table of Contents

[Мета лабораторної роботи 6](#__RefHeading___Toc3319_204808571)

[Завдання 7](#__RefHeading___Toc5937_3437555080)

# Мета лабораторної роботи

Навчитись реалізовувати навчання та проектування нейронної мережі.

# Завдання

1. Виконати завдання iз прикладу i отримати файл iз навченою моделлю для розпiзнавання рукописних цифр. В будь якому графiчному редакторi створити файл iз рукописною цифрою i розпiзнати її. Пояснити результат.
2. Спроектувати i розробити нейронну мережу на основi таких наборiв даних iмплементованих в Keras. Потрiбно розробити архiтектуру нейронної мережi, навчити її на тестових прикладах, i продемонструвати її роботу на кiлькох зображеннях.

# Завдання

## Виконати завдання iз прикладу i отримати файл iз навченою моделлю для розпiзнавання рукописних цифр. В будь якому графiчному редакторi створити файл iз рукописною цифрою i розпiзнати її. Пояснити результат.

Імпортуємо значення набору даних MNIST з пакету keras. Вхідні дані мають розмір у 60 000 екземплярів та розширення у 28 \* 28 пікселів.

  
  
Рисунок 3.1 - Завантаження датасету MNIST

Переглянемо перші 25 зображень та відповідні їм індекси за допомогою модуля matplotlib.pyplot.

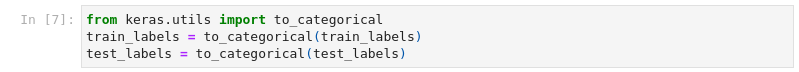
  
  
Рисунок 3.2 - Зображення перших двадцяти п'яти елементів

Робимо перетворення тривимірного масиву в двовимірний масив типу float 32. Далі нормалізовуємо його, щоб отримати значення в інтервалі від [0, 1]. Оскільки маємо колір, то зрозуміло, що його можна закодувати у вигляді байта, пригадуючи RGB, де є 255 \* 255 \* 255 кольорів для червоного, зеленого, блакитного кольорів. Тому треба розділити датасет на навчальний та тестові набори і розділити кожний з елементів на 255.

  
  
Рисунок 3.3 - Розбиття на навчальні та тестові дані і їхня нормалізація

У результаті маємо, що кожне зображення - це вектор з 784 елементів від 0 до 1.

Тепер підготуємо мітки, а саме будемо завантажувати масиви із відгуками на кожне тестове зображення. Після цього передамо нейронній мережі навчальні дані train\_images та train\_labesls. Так мережа навчиться зіставлятии зображення з мітками. to\_categorical - функція, що дозволяє перетворити масив NumPy, що містить числа різних категорій, у матрицю, яка має двійкові значення та має стовпці, що дорівнюють кількості категорій у даних.

  
  
Рисунок 3.4 - Підготування міток

Оберемо архітектуру моделі у вигляді Sequential, оскільки вона є простою послідовністю шарів - вихідного шару, який складається із 28 \* 28 = 784 нейронів, тобто кожному пікселю зображення відповідає один нейрон. За допомогою методу add створимо прихований шар, який є функцією, на вхід якої подається тензор і на виході отримується тензор, можливо, іншого розміру. Використаємо щільні шари, де кожен нейрон одного шару зв'язаний з нейроном іншого. Прихований шар має 512 нейронів, вихідний - 10. Функціями активації є relu та softmax.

  
  
Рисунок 3.5 - Побудова архітектури моделі

Скомпілюємо мережу, привівши її до сумісного вигляду із TensorFlow. Налаштуємо три параметри.

По-перше, оптимізатор - це конкретний алгоритм, що буде оновлювати ваги в процесі навчання.

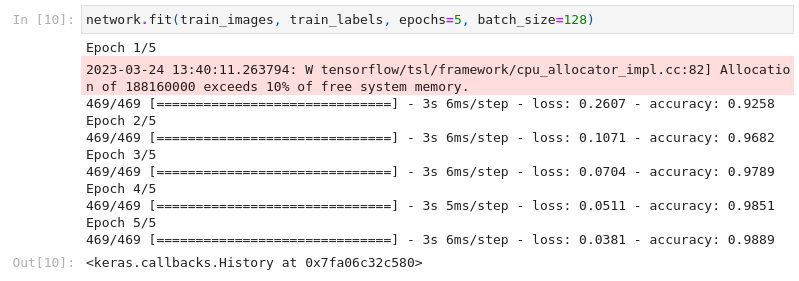
По-друге, функція втрат - функція, яку оптимізатор використовує для оцінювання якості та коригування навчання.

По-третє, метрики - об'єкти, що оцінюють результати навчання за такими параметрами як точність, коректність, повнота.

Після визначення цих параметрів процес компіляції виконується методом .compile(). Оскільки ми вирішуємо задачу класифікації із багатьма класами, то використаємо такі параметри.

  
  
Рисунок 3.6 - Компіляція моделі

Навчання моделі. Викличемо метод fit для навчання, де процес адаптації відбувається у подання пакетів фіксованого розміру до моделі, яка ітеративно підганяє вагові коефіцієнти відповідно до виставлених міток з файлу міток. Кожна ітерація є епохою. За кожної епохи виправляється порушення відгуків новими змінними вагових коефіцієнтів.

  
  
Рисунок 3.7 - Навчання моделі

Перевіримо модель на контрольному наборі даних.

  
  
Рисунок 3.8 - Перевірка моделі

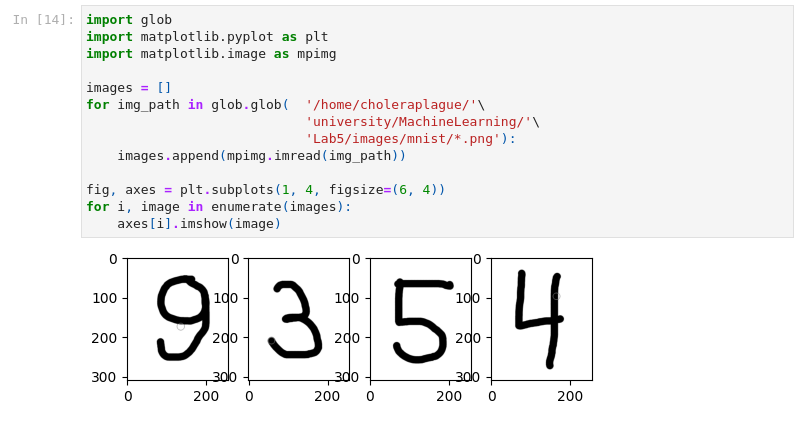
Зробимо збереження моделі командою model.save у форматі HDF5, що містить архітектуру, ваги, конфігурацію тренувань, стан оптимізатора.

  
  
Рисунок 3.9 - Збереження моделі

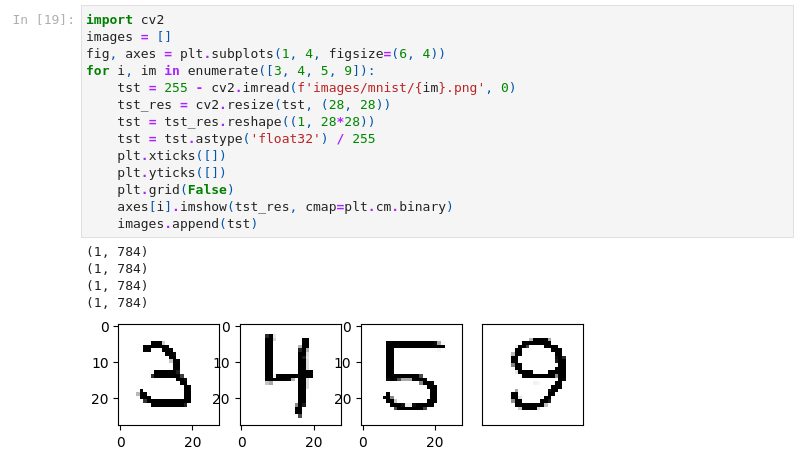
Завантажуємо модель з файлу.

  
  
Рисунок 3.10 - Завантаження моделі

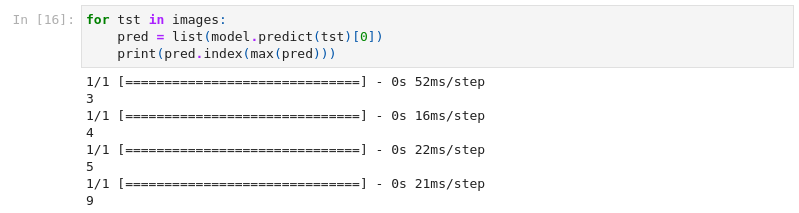
Покажемо тестові картинки, на яких ми будемо тестувати модель. Картинки не стиснені, імпортуємо клас Image з модуля IPython для відображення.

  
  
Рисунок 3.11 - Тестові зображення

Завантажимо їх в OpenCV як сіре зображення і отримуємо дані у форматі ndarray. Потім переформатуємо цей масив до розміру, на якому відбувалося тренування нашої моделі.

  
  
Рисунок 3.12 - Обробка зображень

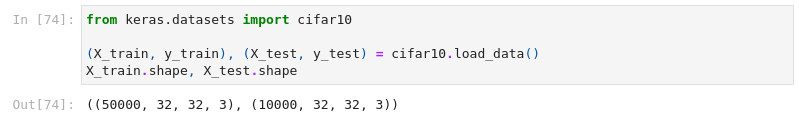
Спрогнозуємо результати, провівши тестування моделі.

  
  
Рисунок 3.13 - Спрогнозовані результати

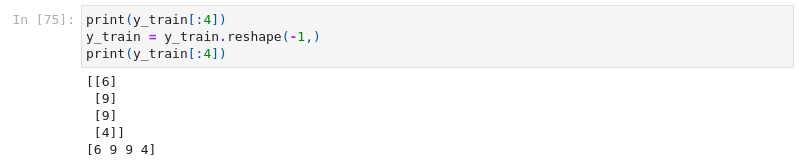
Отже, тестування свідчать про успішну роботу нейронної мережі. Це означає, що набір даних, на якому вона тренувала, був якісним. До того ж параметри були підібрані правильно, а саме: optimizer='rmsprop', loss='categorical\_crossentropy', metrics='accuracy', - які чудово підходять для задач класифікації із багатьма класами, наприклад множина цифр, де кожний елемент - окремий клас зображень. Однак, для повної оцінки роботи мережі потрібно провести тестування на більш широкому діапазоні зображень цифр та провести аналіз результатів.

## Спроектувати i розробити нейронну мережу на основi набору даних Cifar10

Імпортуємо значення набору даних MNIST з пакету keras. Вхідні дані мають розмір у 60 000 екземплярів та розширення у 32 \* 32 пікселів.

  
  
Рисунок 3.14 - Завантаження датасету Cifar10

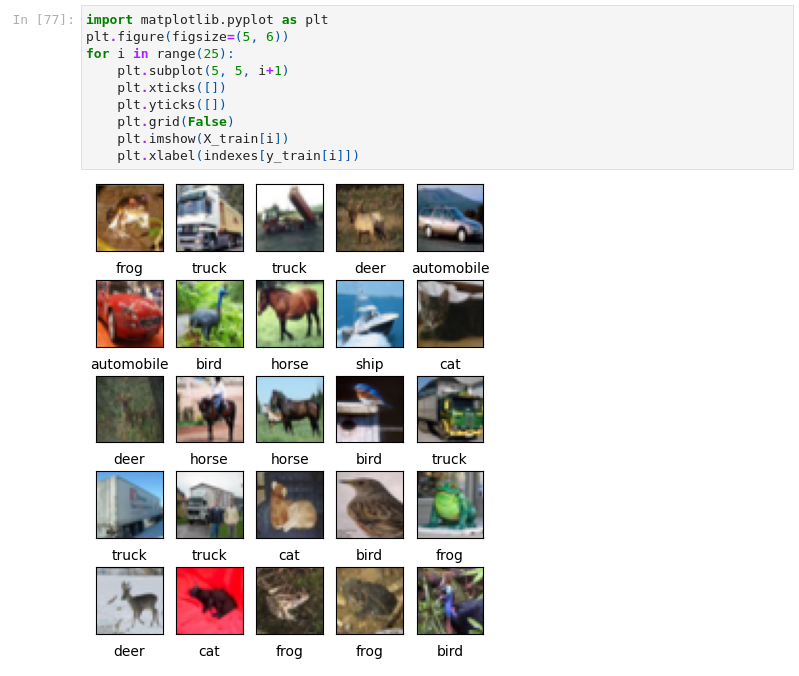
Змінимо розмірність y\_train, оскільки він є двовимірним масивом, у якому кожен рядок має лише один елемент. Тож приберемо цю незручність і перетворимо його в одновимірний масив.

  
  
Рисунок 3.15 - Перетворення y\_train в одновимірний масив

Позначимо індекси для кращого розуміння, що кожне число в y\_train означає.

  
  
Рисунок 3.16 - Індекси, тобто імена класів

Переглянемо перші 25 зображень та відповідні їм індекси за допомогою модуля matplotlib.pyplot.

  
  
Рисунок 3.17 - Зображення перших двадцяти п'яти елементів

Нормалізовуємо датасет, щоб отримати значення в інтервалі від [0, 1]. Оскільки маємо колір, то треба розділити кожний з елементів на 255.

  
  
Рисунок 3.18 - Нормалізація даних

За архітектуру моделі оберемо Sequential. Використаємо щільні шари, де кожен нейрон одного шару зв язаний з нейроном іншого. Прихований шар має по 64 нейрони. Оскільки у нас є 10 індексів, то і вихідний шар міститиме 10 нейронів.

Функціями активації є relu та softmax. Використаємо Flatten для перетворення багатовимірних вхідних данних до одновимірних.

Використаємо паттерн CNN( Convolutional Neural Network ) - це мережа, яка є одним із методів класифікації та розпізнавання зображень у нейронних мережах. Вона призначена для обробки даних кількома шарами масивів. Цей тип нейронної мережі використовується в таких програмах, як розпізнавання зображень. Основна відмінність між CNN та іншими нейронними мережами полягає в тому, що CNN приймає вхідні дані як двовимірний масив. І він працює безпосередньо з зображеннями, а не зосереджується на виділенні ознак, як це роблять інші нейронні мережі.

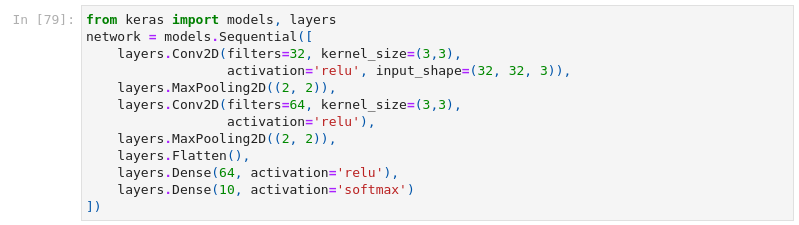
Convolutional Neural Networks мають 4 шари: convolutional, relu шар, pooling, fully connected.

Початковий крок у вилученні функцій із вхідного зображення виконується згортковим шаром( convolutional layer ). Використовуючи вхідну матрицю, згортковий рівень підтримує зв’язок між пікселями під час вивчення характеристик зображення. Цей математичний процес включає два входи: матрицю зображення та ядро або фільтр.

Функція активації Rectified Linear Unit (ReLU) запускає вузол лише тоді, коли вхід перевищує певний поріг. Коли вхід нижче нуля, вихід дорівнює нулю; однак, як тільки він перевищує поріг, він зберігає лінійний зв'язок із залежною змінною.

Під час попередньої обробки зображення шар об’єднання( pooling layer ) є ключовим компонентом, який допомагає зменшити параметри зображення, коли воно стає надмірно великим. Це «зменшення масштабу» зображення досягається шляхом об’єднання, яке можна розглядати як стиснення зображення для зменшення його щільності, подібно до його зменшення.

Максимальне об’єднання( max pooling ) — це процес дискретизації на основі вибірки. Основна мета максимального об’єднання полягає в тому, щоб зменшити масштаб представлення вхідних даних, зменшивши його розмірність і дозволивши зробити припущення щодо функції, що міститься в групуванні підобласті.

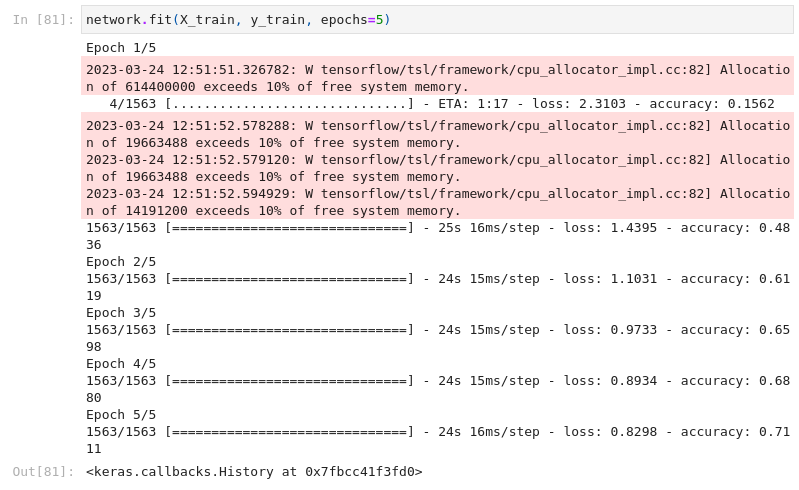
  
  
Рисунок 3.19 - Побудова архітектури моделі

Скомпілюємо мережу, привівши її до сумісного вигляду із TensorFlow. Налаштуємо оптимізатор, функцію втрат, метрики.

Використаємо оптимізатор "adam", який працює на основі градієнтного спуску. У якості функції втрат використаємо Sparse Categorical Crossentropy, у якій передбачені ймовірності порівнюються з істинними мітками, де істинні мітки є цілими числами, що відповідають індексу правильного класу. Це на відміну від категоріальної крос-ентропії, де істинні мітки є векторами, закодованими одним способом.

  
  
Рисунок 3.20 - Компіляція моделі

Навчання моделі.

  
  
Рисунок 3.21 - Навчання моделі

Перевіримо модель на контрольному наборі даних.

  
  
Рисунок 3.22 - Перевірка моделі

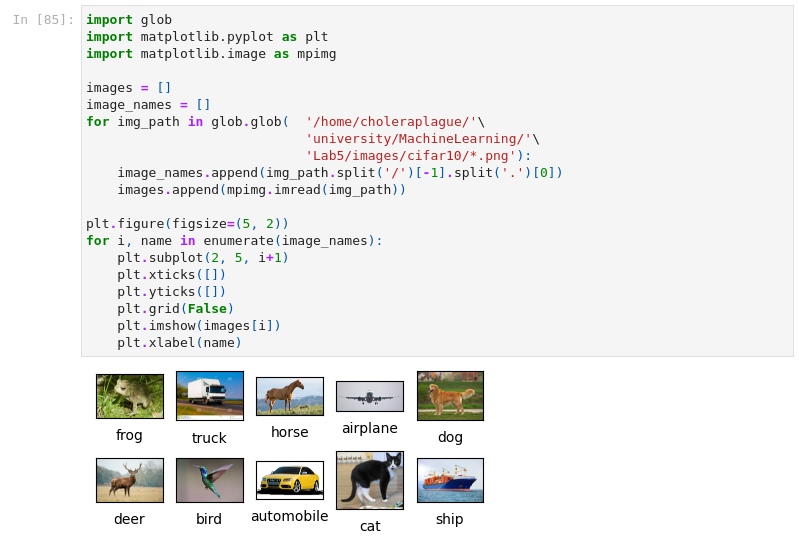
Зробимо збереження моделі командою model.save у форматі HDF5, що містить архітектуру, ваги, конфігурацію тренувань, стан оптимізатора.

  
  
Рисунок 3.23 - Збереження моделі

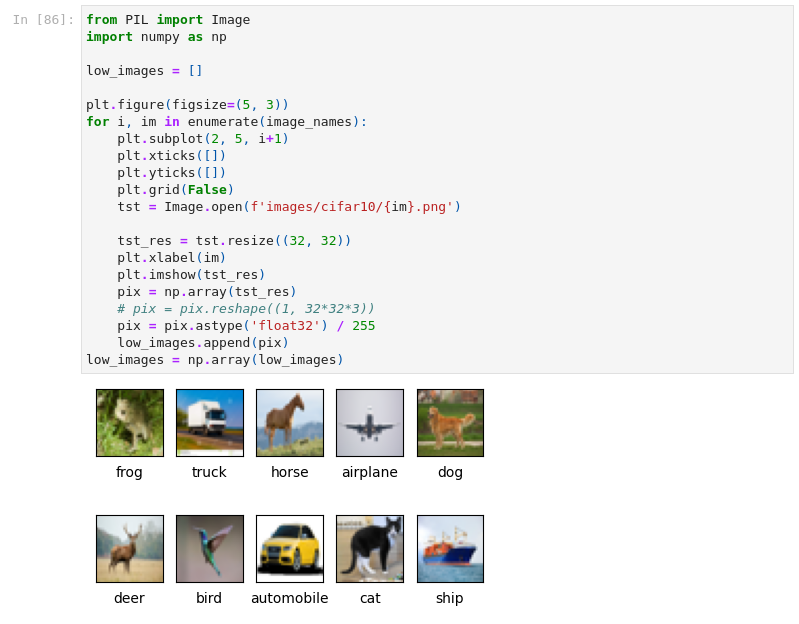
Завантажуємо модель з файлу.

  
  
Рисунок 3.24 - Завантаження моделі

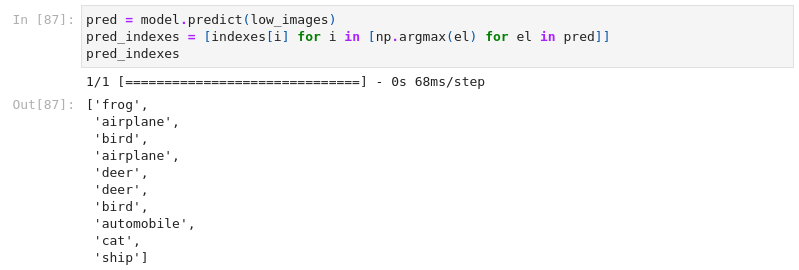
Покажемо тестові картинки, на яких ми будемо тестувати модель.

  
  
Рисунок 3.25 - Тестові зображення

Завантажимо їх в PIL у форматі ndarray. Потім переформатуємо цей масив до розміру, на якому відбувалося тренування нашої моделі.

  
  
Рисунок 3.26 - Обробка зображень

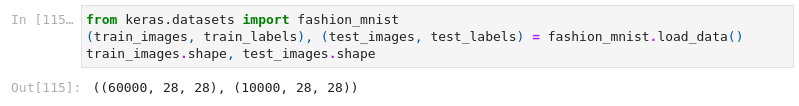
Спрогнозуємо результати, провівши тестування моделі.

  
  
Рисунок 3.27 - Спрогнозовані результати

Отже, тестування свідчать про відносно успішну роботу нейренної мережі. Маємо те, що мережа сплутала 3 об'єкти: вантажівку з літаком, собаку з оленем та коня з птахою. Як ми побачили точність нейронної мережі оцінюється в 67.52 %. Це означає, що хоч підбір даних був і якісним, але архітектура мережі не дала змоги досягти більш виоких результатів. Тож треба зробити шарів у мережі та витратити час на підбір оптимального набору даних.

## Спроектувати i розробити нейронну мережу на основi набору даних FMNIST

Імпортуємо значення набору даних MNIST з пакету keras. Вхідні дані мають розмір у 60 000 екземплярів та розширення у 32 \* 32 пікселів.

  
  
Рисунок 3.28 - Завантаження датасету FMNIST

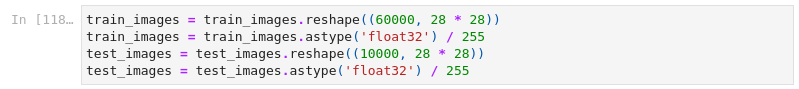
Позначимо індекси для кращого розуміння, що кожне число в y\_train означає.

  
  
Рисунок 3.29 - Індекси, тобто імена класів

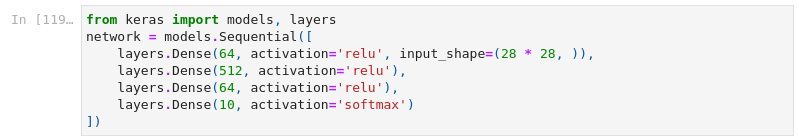
Переглянемо перші 25 зображень та відповідні їм індекси за допомогою модуля matplotlib.pyplot.

  
  
Рисунок 3.30 - Зображення перших двадцяти п'яти елементів

Робимо перетворення тривимірного масиву в двовимірний масив типу float 32. Далі нормалізовуємо його, щоб отримати значення в інтервалі від [0, 1].

  
  
Рисунок 3.31 - Нормалізація даних

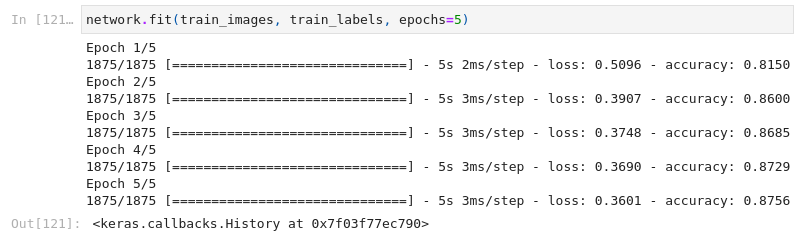
За архітектуру моделі оберемо Sequential. Використаємо щільні шари, де кожен нейрон одного шару зв язаний з нейроном іншого. Прихований шар маює по 512 нейронів. Оскільки у нас є 10 індексів, то і вихідний шар міститиме 10 нейронів.

  
  
Рисунок 3.32 - Побудова архітектури моделі

Скомпілюємо мережу, привівши її до сумісного вигляду із TensorFlow. Налаштуємо оптимізатор, функцію втрат, метрики.

  
  
Рисунок 3.33 - Компіляція моделі

Навчання моделі.

  
  
Рисунок 3.34 - Навчання моделі

Перевіримо модель на контрольному наборі даних.

  
  
Рисунок 3.35 - Перевірка моделі

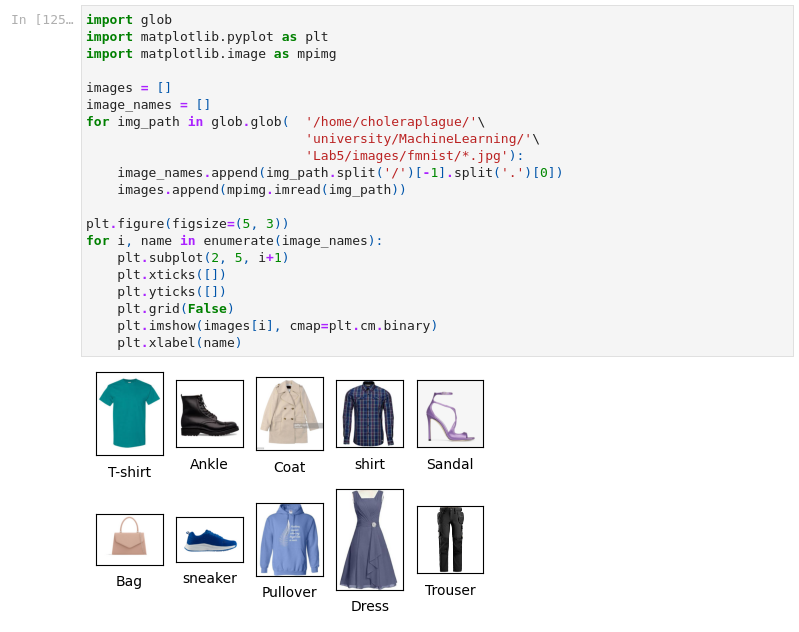
Зробимо збереження моделі командою model.save у форматі HDF5, що містить архітектуру, ваги, конфігурацію тренувань, стан оптимізатора.

  
  
Рисунок 3.36 - Збереження моделі

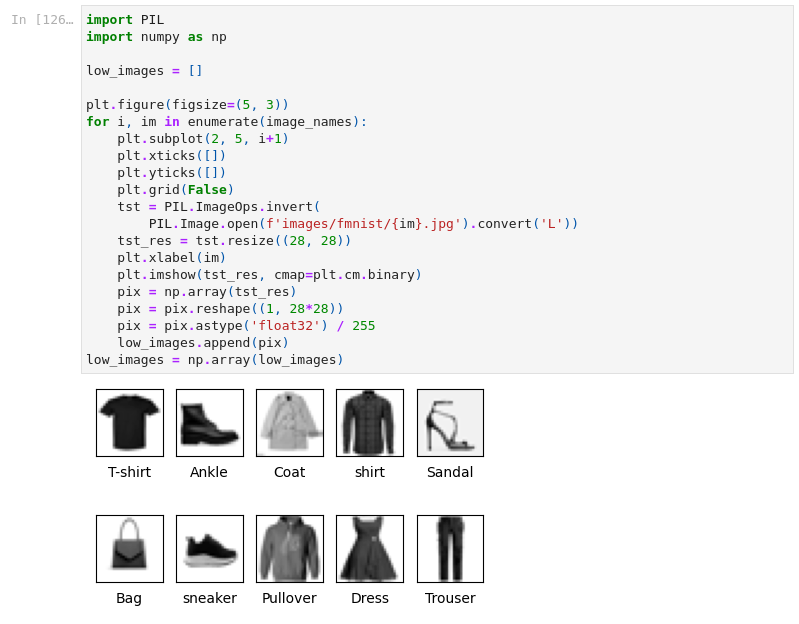
Завантажуємо модель з файлу.

  
  
Рисунок 3.37 - Завантаження моделі

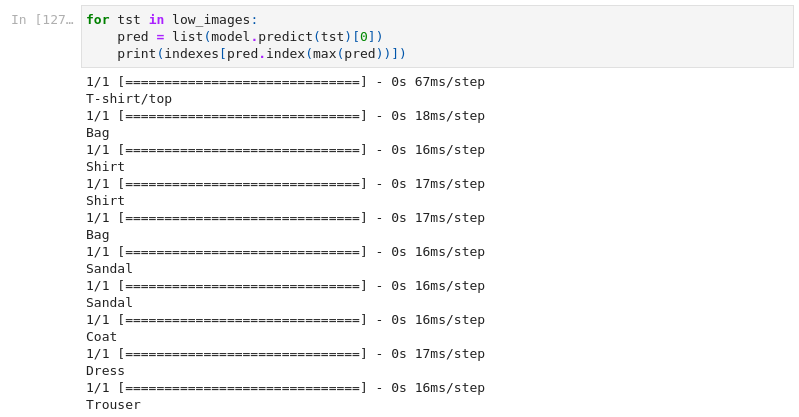
Покажемо тестові картинки, на яких ми будемо тестувати модель.

  
  
Рисунок 3.38 - Тестові зображення

Завантажимо їх в PIL у форматі ndarray. Потім переформатуємо цей масив до розміру, на якому відбувалося тренування нашої моделі.

  
  
Рисунок 3.39 - Обробка зображень

Спрогнозуємо результати, провівши тестування моделі.

  
  
Рисунок 3.40 - Спрогнозовані результати

Отже, тестування свідчать про відносно неуспішну роботу нейренної мережі. Як ми побачили точність на тестових даних нейронної мережі оцінюється в 87.68 %. Це означає, що хоч підбір даних був якісним, але треба мати на увазі, що мережа по суті сплутала два об'єкти: сандаль з сумкою та чобіт із сумкою. На мій погляд, достатньо складно для нейромережі пояснити, чим пальто відрізняється від пуловера чи сорочки, оскільки на малюнках у 28 \* 28 піеселів ця різниця і деталі втрачаються. Я вважаю, що треба збільшувати роздільну здатність картинок та будувати складнішу архітектуру моделі.

# Висновок

Під час виконання цієї лабораторної роботи здобув базові навички

з використання класифікатора KNN та обробкою вбудованих наборів даних з пакету sklearn на прикладі digits. Було візуалізовано зображення digits, розбито дані на тренувальні та тестові у відношеннях 75% до 25% та 80% до 20% , створено та начено модель KNN, спрогнозовано класи. Дізналися точність оцінювання моделі KNN та зобразили матрицю невідповідностей, де побачили, що високі значення концентруються навколо головної діагоналі, що показує високу точність моделі. Додатково виведено звіт класифікації. Окремим завданням налаштували гіперпараметри за допомогою GeidSearchCV для SVC та GaussianNB класифікаторів. Також це було зроблено для KNN, але окрім цього знайшли найкращі параметри за допомогою циклу. Виявилося, що KNN має найвищу точність серед усіх трьох моделей.