**Знайомство з нейрониими мережами та їх побудова**

План

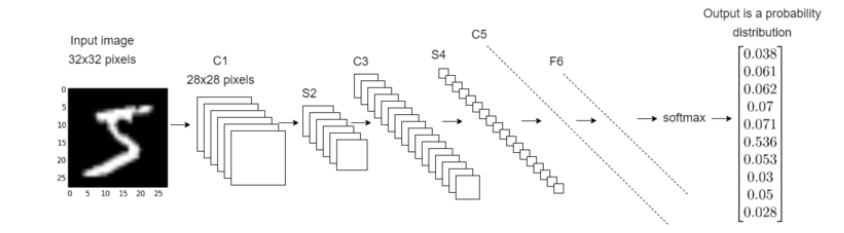
1. Основні поняття згорткових нейронних мереж
2. Що таке MNIST, Keras, TensorFlow
3. Установка залежностей та комнонентів
4. Побудова мережі

**Основні поняття згорткових нейронних мереж**

Нейронні мережі сьогодні всюди, ключовий представник, відомий всім це ChatGPT – мовна модель, **з якої ви можете спробувати скатать цю лабу, але ось вам спойлер – не вийде, тому вчитуйтесь уважно.** Сьогодні, ми навчимо холодну і бездушну машину, яка розуміє лише дві цифри, розпізнавати аж десять, а саме задача буде полягати в тому, щоб створити нейронну мережу яка буде ідентифікувати рукописні цифри від 0 до 9. Це буквально «Hello, World» в машинному навчанні.

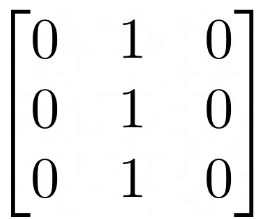
Нейронна мережа - це навчальна система. Вона діє не тільки відповідно до заданого алгоритмом і формулами, але і виходячи з минулого досвіду. Така собі дитина, який з кожним разом складає пазл, роблячи все менше і менше помилок.

**Зго́рткові нейро́нні мере́жі** (**ЗНМ**, [англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *convolutional neural network, CNN, ConvNet*) в [машинному навчанні](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B5_%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F) — це клас глибинних [штучних нейронних мереж](https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B0_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0) [прямого поширення](https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0_%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%B3%D0%BE_%D0%BF%D0%BE%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%BD%D1%8F), який успішно застосовувався до аналізу візуальних зображень. Згорткові мережі взяли за основу [біологічний](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F) процес, а саме схему з'єднання [нейронів](https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD) [зорової кори](https://uk.wikipedia.org/wiki/%D0%97%D0%BE%D1%80%D0%BE%D0%B2%D0%B0_%D0%BA%D0%BE%D1%80%D0%B0) тварин. Окремі [нейрони кори](https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD_%D0%BA%D0%BE%D1%80%D0%B8) реагують на [стимули](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%B8%D0%BC%D1%83%D0%BB_(%D1%84%D1%96%D0%B7%D1%96%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F)&action=edit&redlink=1) лише в обмеженій області [зорового поля](https://uk.wikipedia.org/wiki/%D0%97%D0%BE%D1%80%D0%BE%D0%B2%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5), відомій як [рецептивне поле](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D1%86%D0%B5%D0%BF%D1%82%D0%B8%D0%B2%D0%BD%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5).

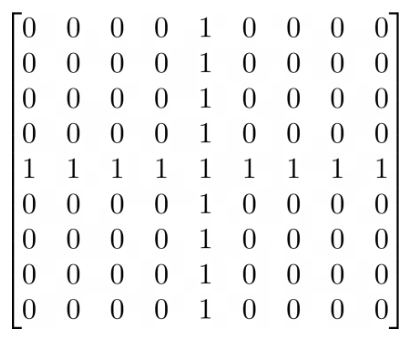


Ключовим будівельним блоком згорткової нейронної мережі є згортковий рівень. Ми можемо візуалізувати згортковий шар як багато маленьких квадратних шаблонів, які називаються згортковими ядрами, які ковзають по зображенню та шукають шаблони. Якщо ця частина зображення відповідає шаблону ядра, ядро ​​повертає велике додатне значення, а якщо відповідності немає, ядро ​​повертає нуль або менше значення.

Математично ядро ​​— це матриця ваг. Наприклад, наступне ядро ​​3x3 виявляє вертикальні лінії.

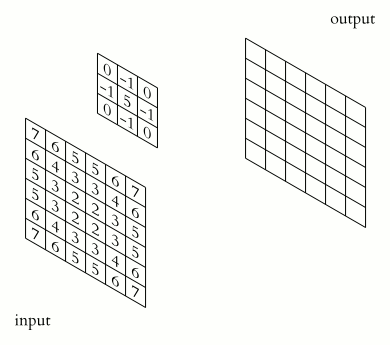


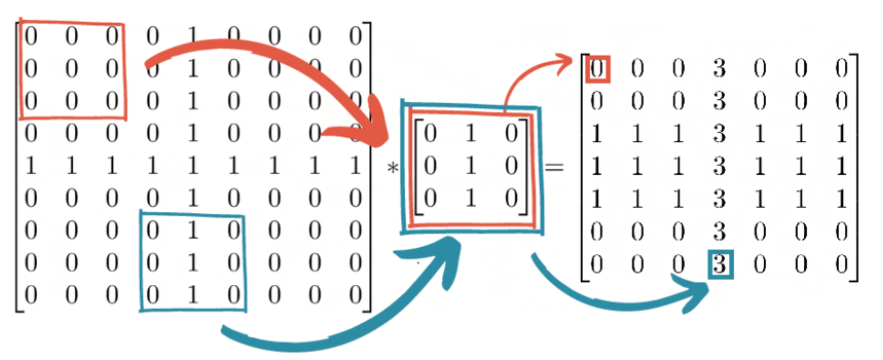
Уявімо вхідне зображення 9x9 зі знаком плюс. Тут є два види ліній, горизонтальні та вертикальні, а також перехресна лінія. У матричному форматі зображення виглядало б так:



Уявіть, що ми хочемо перевірити ядро ​​детектора вертикальної лінії на зображенні зі знаком плюс. Щоб виконати згортку, ми проводимо ядро ​​згортки по зображенню. У кожній позиції ми множимо кожен елемент ядра згортки на елемент зображення, яке воно охоплює, і підсумовуємо результати.

Оскільки ядро ​​має ширину 3, його можна розташувати лише в 7 різних положеннях по горизонталі в зображенні шириною 9. Отже, кінцевим результатом операції згортки на зображенні розміром 9x9 із ядром згортки 3x3 є нове зображення розміром 7x7 .





Отже, у наведеному вище прикладі спочатку ядро (матриця) ​​розміщується у верхньому лівому куті, і кожен елемент ядра множиться на кожен елемент у червоному полі у верхньому лівому куті вихідного зображення. Оскільки всі ці значення дорівнюють 0, результатом для цієї комірки є 0 у верхній лівій частині вихідної матриці.

Тепер давайте розглянемо положення синього прямокутника у наведеному вище прикладі. Він містить частину вертикальної лінії. Коли ядро ​​розміщене над цією вертикальною лінією, воно збігається та повертає 3.

Нагадаємо, що це ядро ​​згортки є детектором вертикальних ліній. Для частин вихідного зображення, які містили вертикальну лінію, ядро ​​повернуло значення 3, тоді як воно повернуло значення 1 для горизонтальної лінії та 0 для порожніх областей зображення.

На практиці ядро ​​згортки містить як ваги, так і зміщення, подібно до формули для [лінійної регресії](https://deepai.org/machine-learning-glossary-and-terms/linear-regression) . Отже, вхідний піксель множиться на вагу, а потім додається зсув.

У повністю підключеній нейронній мережі прямого зв’язку кожен вузол у вхідних даних прив’язаний до кожного вузла першого рівня тощо. Немає ядра згортки.

Отже, у наведеному вище прикладі зображення 9x9 на вході та зображення 7x7 як вихід першого рівня, якби це було реалізовано як повнопідключену нейронну мережу прямого зв’язку, було б



Однак, коли це реалізовано як згортковий рівень з одним згортковим ядром 3x3, є



Зрозуміло, що згорточна нейронна мережа використовує набагато менше параметрів, ніж еквівалентна повністю зв’язана нейронна мережа прямого зв’язку з тими ж розмірами шару. Це тому, що параметри мережі повторно використовуються, коли ядро ​​згортки ковзає по зображенню. Інтуїтивно це пояснюється тим, що згорточна нейронна мережа повинна мати можливість виявляти особливості зображення незалежно від того, де вони розташовані. Ця стійкість згорткових нейронних мереж називається «трансляційною інваріантністю».

**Що таке MNIST, Keras, TensorFlow**

[MNIST](https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/) — це набір даних, розроблений Янном Лекуном, Корінною Кортес і Крістофером Берджесом для оцінки моделей машинного навчання в задачі класифікації рукописних цифр.

Набір даних було створено з ряду сканованих документів, доступних у [Національному інституті стандартів і технологій](http://www.nist.gov/) (NIST). Саме звідси походить назва набору даних, модифікований набір даних NIST або MNIST.

Зображення цифр були взяті з різних відсканованих документів, нормалізовані за розміром і вирівняні по центру. Це робить його чудовим набором даних для оцінки моделей, дозволяючи розробнику зосередитися на машинному навчанні з мінімальними потребами очищення чи підготовки даних.

Кожне зображення є квадратом 28 × 28 пікселів (загалом 784 пікселя). Для оцінки та порівняння моделей використовується стандартне розділення набору даних, де 60 000 зображень використовуються для навчання моделі, а окремий набір із 10 000 зображень використовується для її тестування.

Це завдання на розпізнавання цифр. Таким чином, існує десять цифр (від 0 до 9) або десять класів для прогнозування. Результати повідомляються з використанням похибки передбачення, яка є не чим іншим, як оберненою точністю класифікації.

Відмінні результати досягають похибки передбачення менше 1%. За допомогою великих згорткових нейронних мереж можна досягти найсучаснішої похибки прогнозування приблизно в 0,2%.

Keras — це API, призначений для людей, а не для машин. Keras дотримується найкращих практик для *зменшення когнітивного навантаження* : він пропонує узгоджені та прості API, мінімізує кількість дій користувача, необхідних для типових випадків використання, і надає чіткі та дієві повідомлення про помилки. Keras також надає найвищий пріоритет створенню чудової документації та посібників для розробників.

TensorFlow - це бібліотека відкритого вихідного коду, розроблена Google, для виконання чисельних обчислень з використанням графів обчислень. TensorFlow є одним з найпопулярніших фреймворків для створення нейронних мереж та глибокого навчання.

Keras є високорівневим API для побудови нейронних мереж, а TensorFlow – це низькорівневий фреймворк для виконання обчислень. Keras надає зручний інтерфейс для створення та навчання нейронних мереж, але не може виконувати обчислення самостійно. Для виконання обчислень Keras може використовувати різні бекенди, включаючи TensorFlow.

**Установка залежностей та комнонентів**

Для повної установки залежностей та компонентів Keras та TensorFlow в RStudio потрібно виконати наступні кроки:

1. Встановіть останню версію R: <https://www.r-project.org/>
2. Встановіть останню версію RStudio: <https://www.rstudio.com/products/rstudio/download/>
3. Відкрийте RStudio та виконайте наступну команду, щоб встановити пакет reticulate:

install.packages(“reticulate”)

Цей пакет дозволить R взаємодіяти з Python та використовувати його бібліотеки, такі як TensorFlow та Keras.

1. Перевірте, чи встановлено Python на вашому комп'ютері. Відкрийте командний рядок або термінал та введіть наступну команду:

python --version

1. Встановіть бібліотеку TensorFlow за допомогою Python. Відкрийте командний рядок або термінал та введіть наступну команду:

pip install tensorflow

1. Встановіть бібліотеку Keras за допомогою Python. Відкрийте командний рядок або термінал та введіть наступну команду:

pip install keras

1. Після встановлення TensorFlow та Keras встановіть Python в RStudio. Відкрийте RStudio та виконайте наступну команду:

library(reticulate)

path\_to\_python <- install\_python() pip install keras

virtualenv\_create("r-reticulate", python = path\_to\_python)

Зауважу, що якщо у вас уже встановлено Python, вам не потрібно викликати install\_python(), а замість цього можна просто вказати абсолютний шлях до виконуваного файлу Python.

Потім скористайтеся install\_tensorflow() функцією для встановлення TensorFlow, вона встановить його в /usr/bin/python.

library(tensorflow)

install\_tensorflow(envname = "r-reticulate")

Можна використовувати keras::install\_keras(), який встановлює Tensorflow, на додаток до деяких часто використовуваних пакетів, таких як «scipy» і «tensorflow-datasets».

install.packages("keras")

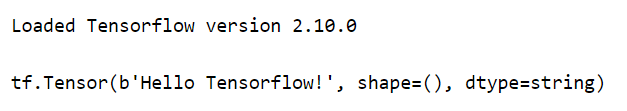
library(keras)

install\_keras(envname = "r-reticulate")

Перевіримо, чи правильно ми встановили бібліотеку, створимо змінну:

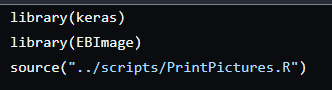
library(tensorflow)

tf$constant("Hello Tensorflow!")



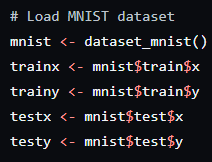
Якщо все встановлено корректно, то ви побачите це повідомлення, зазвичай проблеми під час встановлення виникають з неправильно вказанаим шляхом до пайтону, або з його інсталяцією спочатку, але іноді на це впливає версія мови R чи середи розробки.

**Побудова мережі**

****

Перші два рядки цього коду завантажують два пакети R: keras і EBImage.

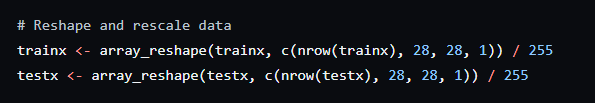
EBImage — це пакет для обробки та аналізу зображень у R. Він надає інструменти для читання, запису та обробки зображень у різних форматах.



Даний код завантажує набір даних MNIST із зберіганням даних для навчання та тестування у змінні **trainx**, **trainy**, **testx** та **testy.** Цей набір даних, складається з рукописних цифр зображених у чорно-білому вигляді. Кожен зразок у наборі складається з **28x28** пікселів, де кожен піксель представлений значенням інтенсивності від **0** до **255**.

Змінна mnist містить функції, які дозволяють завантажити набір даних MNIST. Змінна **mnist$train** містить дані для навчання моделі, а **mnist$test** містить дані для тестування моделі.

Змінні trainx та testx містять зображення цифр, а trainy та testy містять відповідні мітки (0-9) для цих зображень.

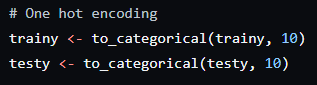


Даний код перетворює дані про зображення цифр у відповідний формат та нормалізує їх.

Спочатку, змінні trainx та testx, які містять зображення цифр, перетворюються з одномірного масиву у чотири-вимірний масив. Перші два розміри відповідають розміру зображення (28 на 28), третій розмір відповідає кількості каналів (один, оскільки ми маємо чорно-білі зображення), а четвертий розмір відповідає кількості зображень у масиві.

Наприкінці, дані нормалізуються, щоб значення інтенсивності пікселів були у діапазоні від 0 до 1. Для цього кожен елемент у масивах trainx та testx ділиться на 255, яке є максимальним значенням для інтенсивності пікселів у зображенні.

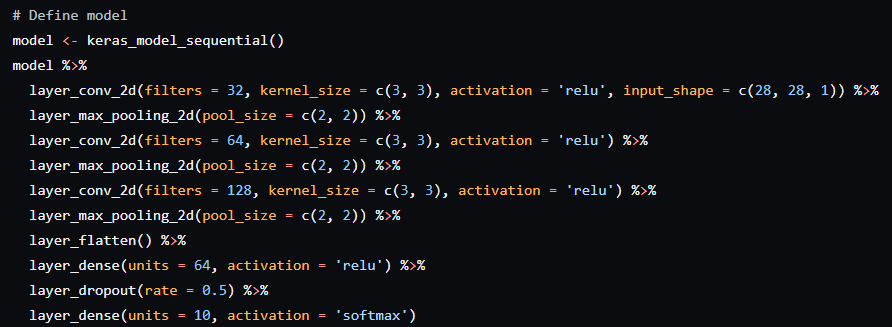
Ця нормалізація допомагає збільшити швидкість навчання та збільшити точність класифікації моделей, оскільки вона зменшує вплив шуму та покращує збіжність алгоритму.



Даний код перетворює вектори міток trainy та testy з цілочисельних значень у вектори бінарних значень за допомогою процедури one-hot encoding.

One-hot encoding - це процес перетворення категоріальних змінних у бінарні вектори, де тільки один елемент у векторі має значення 1, а всі інші елементи мають значення 0. Для задачі класифікації зображень цифр, кожне зображення має мітку від 0 до 9. One-hot encoding перетворює ці числові мітки в бінарні вектори довжиною 10, де індекс елемента зі значенням 1 відповідає мітці зображення.

Функція to\_categorical з бібліотеки keras допомагає виконати процедуру one-hot encoding. Параметр 10, переданий у функцію, вказує на кількість унікальних значень, які містяться у векторах trainy та testy (від 0 до 9).



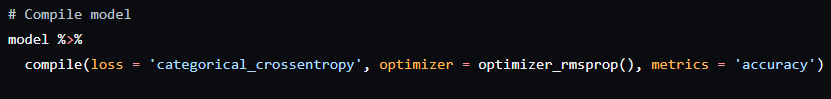
Даний код визначає архітектуру нейромережі за допомогою бібліотеки keras.

Ця нейромережа використовує 9 шарів, які розташовані в послідовному порядку:

Даний код визначає архітектуру нейромережі за допомогою фреймворку Keras.

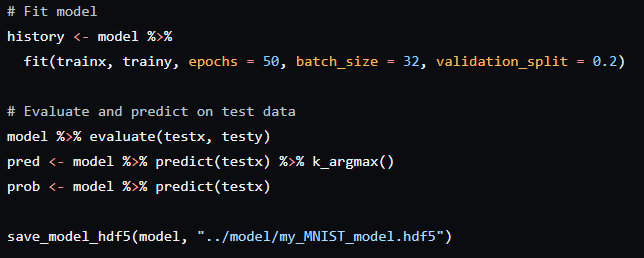
1. Створення моделі: спочатку ми створюємо модель за допомогою функції keras\_model\_sequential (), яка дозволяє визначити модель як послідовність різних шарів.
2. Додавання шарів: з допомогою оператора %>% ми додаємо до моделі шари в правильному порядку. Починаємо зі шару layer\_conv\_2d з 32 фільтрами та ядром розміром 3х3, що використовує функцію активації ReLU та отримує на вхід зображення розміром 28х28х1 (1 канал для чорно-білих зображень). Після цього додаємо шар layer\_max\_pooling\_2d з пулінгом розміру 2х2, що допомагає зменшити розмір зображення та зберегти тільки найбільш значущі ознаки. Потім додаємо ще два шари layer\_conv\_2d та layer\_max\_pooling\_2d, збільшуючи кількість фільтрів у кожному шарі (64 та 128 відповідно) та зменшуючи розмір зображення.
3. Плоский шар: далі додаємо шар layer\_flatten(), який перетворює вихід з попереднього шару у вектор для подальшого використання в повнозв'язних шарах.
4. Повнозв'язний шар: додаємо повнозв'язний шар layer\_dense з 64 нейронами та функцією активації ReLU. Для запобігання перенавчанню використовуємо шар\_dropout з швидкістю відкидання 50%.
5. Вихідний шар: на останок додаємо вихідний повнозв'язний шар layer\_dense з 10 нейронами та функцією активації softmax, який використовується для класифікації зображень на 10 класів (цифр від 0 до 9).

Отже, ця модель має застосувати три послідовні шари з пулінгом та три послідовні шари конволюції для екстракції властивостей зображень, далі згладжує вектори, а потім застосовує два Dense шари з випадковим відключенням нейронів за допомогою Dropout. Останній шар має 10 нейронів та використовує функцію активації softmax для визначення ймовірності належності зображення до кожного з 10 класів.



В даному коді, після визначення моделі, використовується функція compile, яка налаштовує процес тренування моделі. Параметри, які передаються в цю функцію включають:

* **loss:** функція втрат, яка вимірює, наскільки далеко передбачені значення відповідають дійсним значенням під час тренування. У даному випадку використовується функція **categorical\_crossentropy**, яка є стандартним вибором для багатокласової класифікації.
* **optimizer:** алгоритм оптимізації, який оновлює ваги моделі на основі значень втрат та градієнтів. У даному випадку використовується алгоритм RMSprop, який є популярним вибором для глибинного навчання.
* **metrics:** метрика, яка вимірює, наскільки точні передбачені значення моделі під час тренування. У даному випадку використовується метрика accuracy, яка є стандартним вибором для багатокласової класифікації.



У цьому коді виконується тренування моделі з використанням функції fit. Параметри, які передаються в цю функцію включають:

* **trainx:** вхідні дані для тренування моделі.
* **trainy:** відповідні цілі значення для тренування моделі.
* **epochs:** кількість епох, які будуть використані під час тренування. Одна епоха відповідає одному проходу моделі через всі тренувальні дані.
* **batch\_size:** кількість зразків даних, які будуть використані під час одного проходу моделі. Більші значення можуть пришвидшити процес тренування, але можуть призвести до меншої точності.
* **validation\_split**: частка тренувальних даних, які будуть використовуватися для валідації моделі під час тренування.

Після тренування моделі, виконується оцінка її точності на тестових даних за допомогою функції **evaluate**, яка приймає вхідні дані та відповідні цілі значення. Потім модель використовується для передбачення класів на тестових даних за допомогою функції **predict**, яка повертає вектор передбачених класів та вектор ймовірностей для кожного класу.



Цей код містить наступні дії:

1. Завантаження моделі з файлу **my\_MNIST\_model.hdf5**.
2. Завантаження нових даних з папки **../tests**.
3. Обробка нових зображень: зчитування файлу, перетворення у чорно-біле, інвертування кольорів, зміна розміру до **28x28** пікселів, і перетворення на необхідну форму **(15, 28, 28, 1).**
4. Передбачення класу зображення на основі завантаженої моделі.
5. Виведення таблиці з результатами передбачень та їх порівняння зі змінними **pred** та **newy**. Це дозволяє оцінити точність моделі на нових даних.

Зверніть увагу, що дана модель була навчена на даних з набору даних MNIST, який складається з рукописних цифр. Тому, передбачення на інших типах зображень може бути неправильним. Крім того, якщо нові дані мають інший формат або розмір, то їх також потрібно буде відповідним чином обробити перед передбаченням за допомогою моделі.

Посилання:

1. <https://github.com/AndryVolynets/Digit-Recognition>
2. <https://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi>
3. <https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network#:~:text=Convolutional%20neural%20networks%20contain%20many%20convolutional%20layers%20stacked,layers%20it%20is%20possible%20to%20distinguish%20human%20faces>.
4. <https://ppt-online.org/108201>
5. <https://www.coursera.org/learn/convolutional-neural-networks-ru/home/week/1>
6. <https://tensorflow.rstudio.com/install/>