КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки

на тему:

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ ТА ЇХ ПРАКТИЧНЕ ЗАСТОСУВАННЯ

Виконав студент 4-го курсу
Дерябін Микита Євгенійович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Кузенко Володимир Федорович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
« 01 » червня 2022 р.,
Протокол № 10
Нікітченко М.С.

(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 69 сторінок, 43 ілюстрації, 1 таблиця, 20 джерел посилань.

ГЛИБОКІ МЕРЕЖІ, ЗГОРТКОВІ МЕРЕЖІ, РОЗПІЗНАВАННЯ ОБ'ЄКТІВ, МЕРЕЖА ПРЯМОГО ПОШИРЕННЯ, НЕЙРОЕВОЛЮЦІЯ, ГЕНЕТИЧНІ АЛГОРИТМИ, НЕЙРОН, НЕЙРОНА МЕРЕЖА, РЕКУРЕНТНІ МЕРЕЖІ, ФУКНЦІЯ АКТИВАЦІЇ.

Об'єктом роботи є аналіз засобів та принципів розробки штучних нейронних мереж, процес їх побудови та практичне застосування.

Предметом роботи є програмні застосунки, які демонструють роботу штучних нейронних мереж, а саме: стилізація зображення, кольоризація монохромного зображення, розпізнавання об'єктів на зображенні, а також застосунки, які використовують алгоритми нейроеволюції, для відтворення руху об'єктів заданою траєкторією, та проходження перешкод.

Метою даного проєкту є створення програмних застосунків, які використовують моделі штучних нейронних мереж для обробки або розпізнавання об'єктів зображення, а також застосунків, побудованих на базі алгоритмів нейроеволюції.

Інструменти розроблення: операційна система – Windows 11, середовище програмування – PyCharm. Мова програмування Python, версія мови програмування – python 3.10, набір модулів та пакетів – numpy, pygame, math, os, neat-python, sys, keras, tensorflow, matplotlib, PIL, skimage.

Результати роботи: виконано загальний огляд засобів та принципів побудови, тренування, для різних моделей штучних нейронних мереж. Було успішно проведено створення та навчання певних моделей мереж, використання існуючих натренованих мереж.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Programing Interface, прикладний програмний інтерфейс;
AI – Artificial Intelligence, штучний інтелект;
BPTT – Backpropagation through time, зворотне поширення в часі;
CNN – Convolutional Neural Network, згорткова нейронна мережа;
CPU – Central Processing Unit, процесор;
GPU – Graphics Processing Unit, графічний процесор;
GRU – Gated recurrent unit, закриті нейронні мережі;
RNN – Recurrent neural network, рекурентна нейронна мережа;
NEAT – NeuroEvolution of Augmenting Topologies, нейроеволюція розширюючих топологій;
IDE – Integrated Design Environment, інтегроване середовище розробки;
DL – Deep Learning, глибоке навчання;
DNN – Deep Neural Network, глибокі нейронні мережі;
LSTM – Long Short Term Memory, мережі довготривалої пам’яті;
MSE – Mean Square Error, середнє значення квадратів помилок;
NLP – Natural Language Processing, обробка природної мови.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Основна мета штучних нейронних мереж – навчатися, автоматично корегуючи себе, щоб у майбутньому виконувати складні завдання, які не можна вирішити за допомогою класичного програмування на основі певних правил та алгоритмів. Обсяг призначення нейронних мереж доволі широкий, завдяки своїй роботі вони здатні апроксимувати будь-яку функцію, що існує, при достатньому рівні навчання. В основному нейронні мережі являють собою модифікації для задач прогнозування та класифікації. За допомогою нейронних мереж AI досягнув значного прогресу в 20-их роках XXI століття. Було відзначено багато досягнень AI в NLP, комп’ютерному зорі та робототехніці. Найкращі досягнення AI сьогодення включають вдосконалення автоматичної генерації тексту, розпізнавання обличчя та мови, виявлення жестів руху, відкриття нових ліків і квантової переваги.

Актуальність роботи та підстави для її виконання. Спектр діяльності штучних нейронних мереж широкий і потрібний сьогодні, вони використовуються в багатьох сферах сучасного життя таких як економіка, де вони можуть допомогти передбачити наскільки ціни будуть змінюватися з часом або навіть у медицині, при діагностуванні різні проблем зі здоров’ям.

Нейронні мережі стали ключовою частиною розвитку машинного навчання та штучного інтелекту, це одна з основних областей досліджень і та, яка з часом розвивається та створює все більш складні та ефективні рішення, які вже зараз перевершують у фізичних та розумових здібностях людей.

Засоби розробки. Для розробки штучних нейронних мереж було використано дві основні бібліотеки: TensorFlow та Keras, а для використання нейроеволюційного алгоритму NEAT – модуль Neat-Python.

Keras – це API для штучних нейронних мереж. Він дозволяє проводити швидкі дослідження на мові програмування Python. Його можна використовувати для простого та швидкого прототипування застосунку. Підтримує згорткові мережі, є можливість комбінування двох різних мереж, можна запустити на CPU та GPU, що пришвидшить процес навчання моделі.

TensorFlow – відкрита програмна бібліотека для машинного навчання, розроблена компанією Google. Містить у собі пакет Keras. Має зручну та гнучку екосистему інструментів, бібліотек та ресурсів, які дозволяють дослідникам впроваджувати найсучасніші технології машинного навчання.

Neat-Python – модуль, що надає змогу використання генетичного алгоритму для визначення кращої та мінімально необхідної топології нейронної мережі. У розроблених застосунках використовується для відтворення руху об’єктів заданою траєкторією та проходження перешкод.

Мета й завдання роботи. Метою даної роботи є розробка застосунків, які демонструють практичне застосування штучних нейронних мереж. Завданням є створення:

- стилізації зображення: надає можливість привести зображення до певного стилю, тобто із двох картинок отримати одне, яке матиме об’єднану структуру та інформацію зображень;
- кольоризації зображення: дає змогу, із застарілих фотографій у відтінках сірого, отримати кольорове зображення формату RGB;
- розпізнавання об’єктів зображення: дозволяє виявити на зображенні певні класи об’єктів;
- нейроеволюційних застосунків, що використовують генетичні алгоритми навчання.

Можливі сфери застосування. Перелік принципів використання у різних сфера застосування:

а) класифікація;
   1) у сільському господарстві: використовується для сортування фруктів, овочів та інших продовольчих товарів;
   2) в обороні: використовується для класифікації радіолокаційних та гідроакустичних зображень;
   3) у маркетингу: використовується у класифікації певних патернів;
   4) у медицині: використовується в медичній діагностиці та класифікації зображень електрокардіограми;

б) прогнозування та розпізнавання;
   1) у фінансах: використовується у валютному курсі та у прогнозуванні фондового ринку;
   2) у телекомунікації та обчислювальній техніці: використовується для розпізнавання мовлення, рукописного тексту та комп'ютерного зору;
   3) у фінансах: використовується під час перевірки підпису;
   4) у метеорології: використовується для прогнозування погоди;

в) оцінювання;
   1) у машинобудуванні: використовується для контролю якості продукції;
   2) у правоохоронних органах безпеки: використовується при виявленні руху та при знятті відбитків пальців.

РОЗДІЛ 1 ШТУЧНА НЕЙРОНА МЕРЕЖА ТА ЇЇ ОСНОВНІ СКЛАДОВІ ЧАСТИНИ

1.1 Поняття штучної нейронної мережі

Штучна нейронна мережа – це обчислювальна навчальна система, яка використовує мережу функцій, для розуміння вхідних даних в певній початковій формі із подальшим перетворенням отриманої інформації у бажаний вихід, як правило, дещо відмінний від початкової форми.

Концепція штучних нейронних мереж запозичена у біологічних нейронів людини, які в людському мозку функціонують разом, щоб зрозуміти сенсорний вплив людських органів чуття. Таким чином, нейронна мережа – це набір алгоритмів, які намагаються визначити закономірності та зв’язки, зрозуміти інформацію, що надходить, за допомогою процесів, подібних до біологічних.

1.2 Складові частини

Основними компонентами будь-якої нейронної мережі є набір нейронів та зв’язків між ними, при чому кожен окремий зв’язок має свій власний ваговий коефіцієнт. У кожній мережі виділяють три рівні:

- рівень вхідних даних (input layer);
- прихований рівень (hidden layer);
- вихідний рівень (output layer).

Розглянемо кожну частину штучної нейронної мережі окремо та визначимо основні особливості компонентів.

1.2.1 Нейрон

Штучна нейронна мережа складається з сукупності нейронів, з’єднаних за допомогою зв’язків. Кожен окремий нейрон приймає вхідні

дані, виконує з ними певні математичні маніпуляції, а потім передає результат виконання іншим нейронам (рис. 1). Якщо нейрон відноситься до input layer, то він не виконує ніяких математичних дій, а просто передає вхідні дані через зв’язки нейронам рівня hidden.

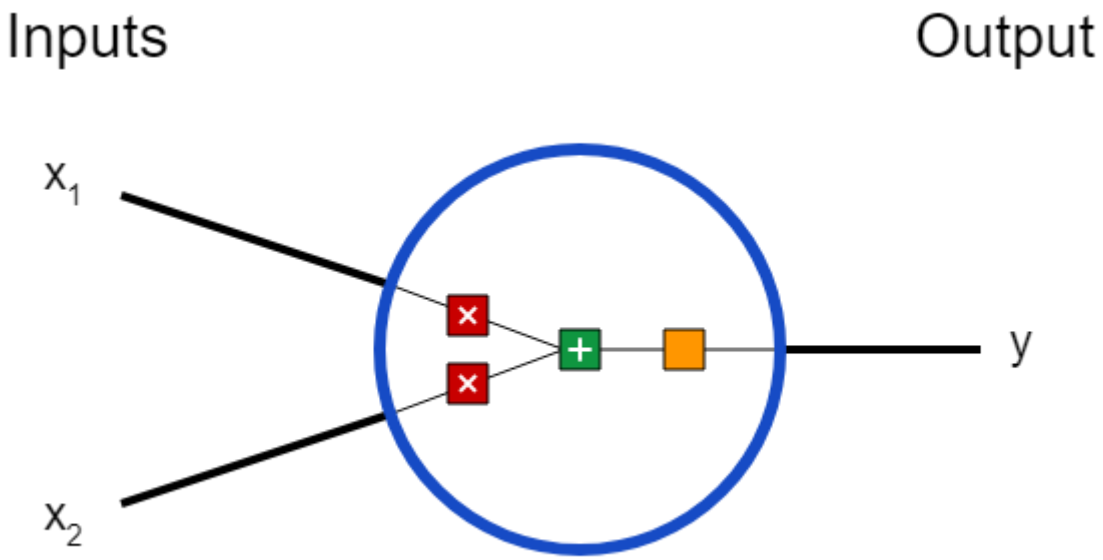


Рисунок 1 – Модель окремого нейрона штучної нейронної мережі

Операції, які виконує окремо взятий нейрон досить прості. Спочатку, він бере суму значень кожного нейрона з попереднього стовпця, з яким він з’єднаний зв’язком [3].

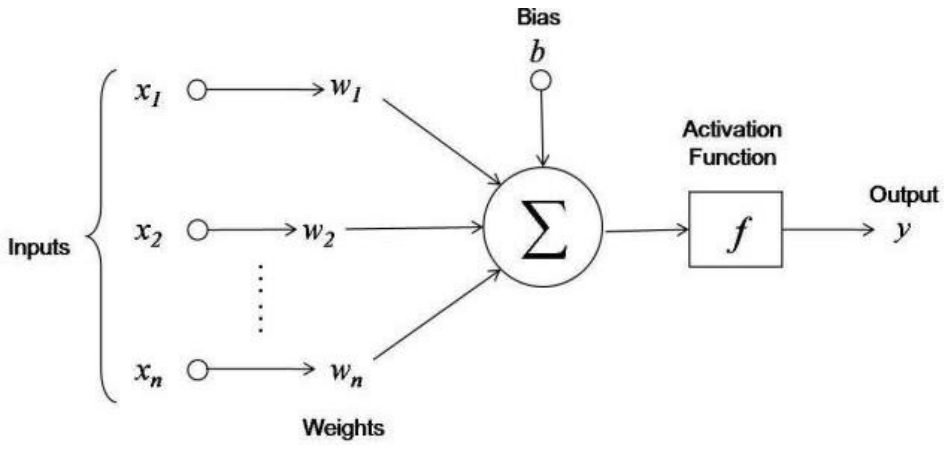


Рисунок 2 – Обробка інформації нейроном

На рис. 2 видно, що до нейрону надходить *n* вхідних параметрів (деякі числові значення), а саме: x_1, x_2, …, x_n, таким чином до нейрону підключено *n* вузлів із попереднього стовпця. Кожне із значень x_1, x_2, …, x_n перед додаванням множиться на свій ваговий коефіцієнт (w_1, w_2, …, w_n), який визначається між двома нейронами з'єднанням. Кожен такий зв’язок нейронної мережі має власний ваговий коефіцієнт, який буде змінюватися в процесі навчання.

Оскільки нейронна мережа займається саме знаходженням форми прогнозованої функції для вхідних даних, то для того, щоб змістити знайдену функцію в сторону, таким чином, щоб вона співпала з базовою, потрібно ввести додатковий параметр, який буде відповідати за зміщення. Цей параметр називається «bias». Зміщення додається до результуючого значення суми перемножених вагових коефіцієнтів на вхідні параметри нейрона. Таким чином маємо наступну формулу:

$$(x_1 \cdot w_1) + (x_2 \cdot w_2) + \cdots + (x_n \cdot w_n) + b, \qquad (1.1)$$

де x_1, x_2, …, x_n – вхідні параметри;

w_1, w_2, …, w_n – вагові коефіцієнти зв’язків;

b – bias (зміщення).

Після виконання попередньої операції, до отриманого значення застосовують функцію активації. У результаті ми отримаємо вихідне значення нейрона, яке далі буде вхідним параметром для наступних нейронів, або буде одним із остаточних вихідних значень нейронної мережі (якщо нейрон належить до output layer). Дана процедура виконується для всіх нейронів прихованого та вузлів вихідного рівня.

1.2.2 Основні три рівні нейронної мережі

Як зазначалося раніше, нейронна мережа має три основні рівні [2]: input layer, hidden layer та output layer (рис. 3). Основну роботу виконує прихований рівень (hidden layer).

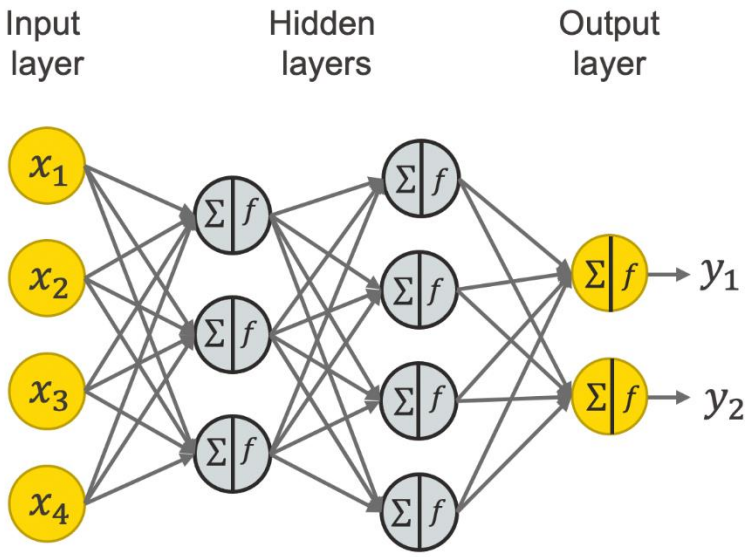


Рисунок 3 – Рівні нейронної мережі

Вхідний рівень (input layer), також відомий як вхідні вузли, це початкові дані із зовнішнього світу, які надаються моделі для вивчення та отримання висновків. Вхідні вузли передають інформацію до наступного рівня – прихованого.

Прихований рівень (hidden layer) – це набір нейронів, де обчислення виконуються з вхідними даними. У нейронній мережі може бути будь-яка кількість прихованих. Найпростіша мережа складається з одного прихованого шару.

Вихідний рівень (output layer) – це вихідні дані моделі, отримані з урахуванням всіх виконаних обчислень. У вихідному шарі може бути один або кілька вузлів. Якщо ми маємо проблему бінарної класифікації, то достатньо лише одного вихідного вузла, але у випадку багатокласової класифікації вихідних вузлів повинно бути більше ніж один. Такий процес

передачі вхідних даних для отримання результату називається прямим розповсюдженням (feedforward).

1.2.3 Функція активації

Функції активації прив’язані до кожного нейрона і є математичними рівняннями, які визначають, чи повинен бути активований окремий нейрон [4]. Це залежить від того, чи має значення використання нейрона для передбачення моделі чи ні.

Функція активації зазвичай призначена для приведення загального значення отриманого за формулою (1.1), до діапазону [0; 1] або [-1; 1], в залежності від обраної функції. Таким чином відбувається нормування вхідних даних. Вибір функції активації залежить від поставленої задачі для нейронної мережі. Існує велика кількість функцій активацій, але давайте розглянемо лише деякі із них.

Сигмоподібна або логістична функція активації (рис. 4) використовується для моделей, де ми повинні передбачити ймовірність як результат. Основна причина, чому доцільно її використовувати в даному випадку це область значень, яка відповідає діапазону [0;1].

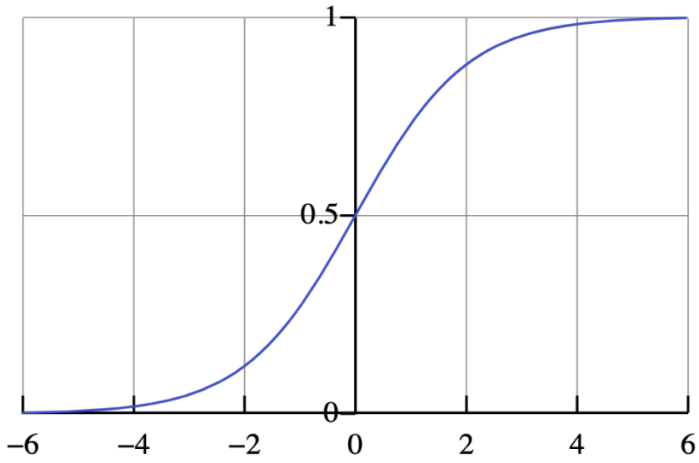


Рисунок 4 – Сигмоподібна функція

Дана функція диференційована. Це означає, що ми можемо знайти нахил кривої сигмоїди в будь-яких двох точках. Функція монотонна, а похідна – ні. Формула функції наступна:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Tanh або функція активації гіперболічного тангенсу схожий формою на логістичну сигмоїду, але діапазон функції – [-1; 1] (рис. 5). Має сенс використовувати гіперболічний тангенс, лише тоді, коли значення можуть бути як від'ємними, так і додатними. Використовувати цю функцію лише з додатними значеннями недоцільно, оскільки це значно погіршить результати нейронної мережі.

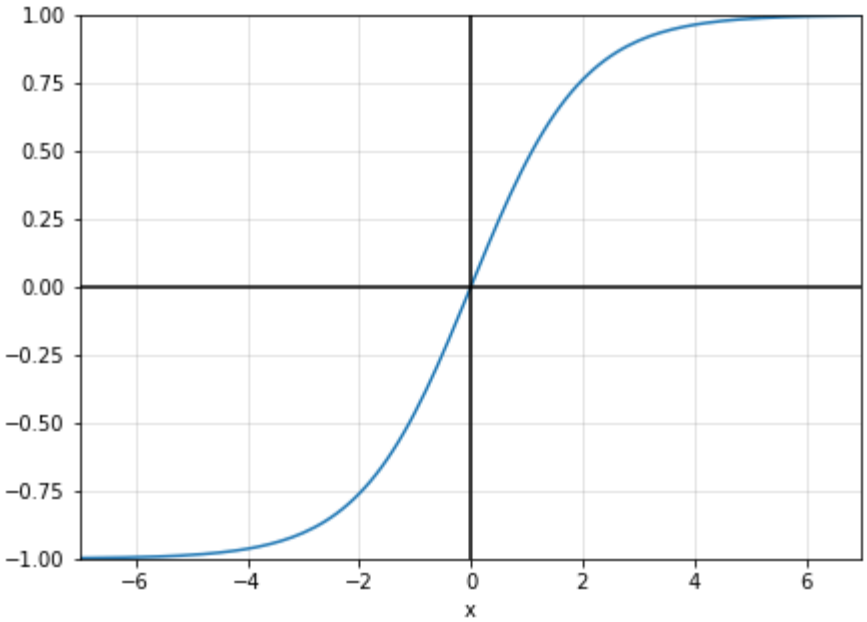


Рисунок 5 – Функція гіперболічного тангенсу

Функція диференційована, монотонна, а її похідна – не монотонна [4]. Функція tanh переважно використовується для розподілення на два класи. І

tanh, і логістична сигмоподібна функція активації використовуються в мережах із прямим зв’язком. Формула функції наступна:

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Функція активації ReLU є найбільш вживаною функцією активації (рис. 6). Вона використовується майже у всіх згорткових нейронних мережах та DL. Діапазон функції – [0; ∞].

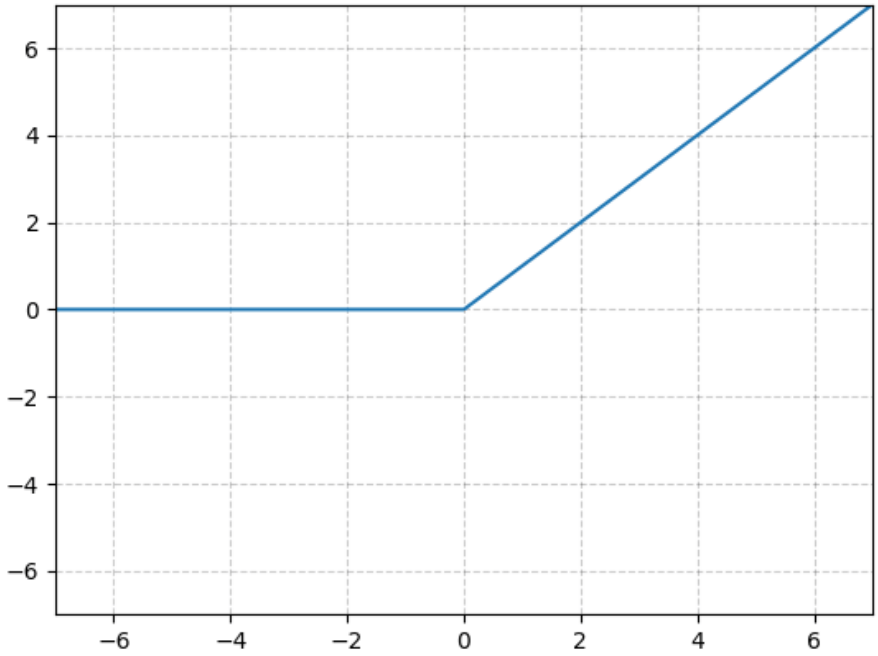


Рисунок 6 – Функція ReLU

Як можна побачити, функція ReLU випрямлена наполовину знизу. Значення функції $f(x)$ дорівнює нулю, коли значення аргументу менше нуля, і $f(x)$ дорівнює x, коли x більше або дорівнює нулю.

$$f(x) = \max(0, x).$$

Проблема даної функції активації полягає в тому, що всі від’ємні значення стають рівними нулю, що знижує здатність моделі правильно навчатися. Це означає, що будь-яке від’ємне вхідне значення, задане функцією активації ReLU, негайно перетворюється на нуль на графіку, що, у свою чергу, впливає на результуючий графік, не відображаючи від’ємні значення належним чином.

1.3 Метод зворотного поширення помилки (Backpropagation)

Зворотне поширення – це процес оновлення та пошуку оптимальних значень вагових коефіцієнтів, що допомагає моделі мінімізувати помилку, тобто знизити різницю між фактичними та прогнозованими значеннями.

Для того, щоб оновлювати ваги зв’язків використовуються оптимізатори. За допомогою бібліотеки Keras можна обрати певний оптимізатор та використати його при навчанні нейронної мережі.

Перелік деяких оптимізаторів:

- SGD ітераційний метод для оптимізації цільової функції з відповідними властивостями гладкості;
- Adagrad – оптимізація на основі квадратів градієнтів;
- RMSProp та Adadelta – подібні до Adagrad, але намагаються боротися із надмірним накопиченням квадратів градієнтів;
- Adam – поєднання алгоритму з моментом і квадратів градієнтів;
- Adamax – варіант оптимізації за Adam, але без обмежень на норму;
- Nadam – комбінація алгоритма Adam із нестерівським моментом;
- Ftrl – оптимізатор, що реалізує Ftrl-алгоритм:

Перед тим як почати використовувати оптимізатор потрібно обрати спосіб оцінки того, наскільки вдало нейронна мережа прогнозує остаточні результати. Для цього потрібно обрати функцію втрат (loss function), яка

буде рахувати різницю між фактичними та прогнозованими значеннями. Функція втрат одномірна і не є вектором, оскільки вона оцінює, наскільки добре нейронна мережа працює загалом.

Деякі відомі функції втрат:

– MSE (середньоквадратичне відхилення);
– крос-ентропія;
– експонентна (AdaBoost);
– відстань Кульбака – Лейблера.

Середньоквадратичне відхилення (рис. 7) – одна з найпростіших та найуживаніших функцій втрат. MSE задається так:

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_{fact}-y_{pred}\right)^2$$

де n – кількість об’єктів, що розглядаються;

y_{fact} – фактичне значення, до якого має наближатися прогнозоване;

y_{pred} – прогнозоване значення нейронної мережі.

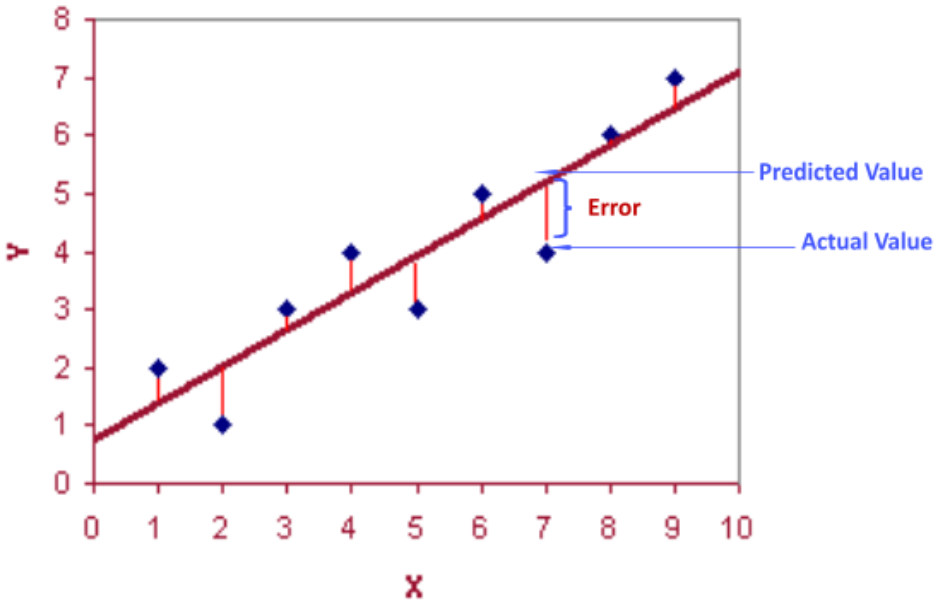


Рисунок 7 – MSE

Різницю між фактичним і прогнозованим значенням називають квадратичною помилкою. Таким чином дана функція знаходить середнє значення по всім квадратичним помилкам. Чим краще прогнозування, тим нижче будуть втрати. Функція втрат нейронної мережі повинна задовольняти двом умовам:

– функція втрат має повертати середнє значення;
– функція втрат не повинна залежати від будь-яких активаційних значень нейронної мережі, крім значень, що отримуються на виході.

Тепер, маючи функцію оцінки прогнозованих значень, нам потрібно використати оптимізатор, який буде мінімізувати втрати. Розглянемо як працює оптимізатор градієнтного спуску Gradient Descent для мінімізації значення деякої функції $J(w)$.

На рис. 8 зображено криву функції $J(w)$. Потрібно мінімізувати помилку таким чином, щоб J_{min} (глобальний мінімум) був досягнутий.

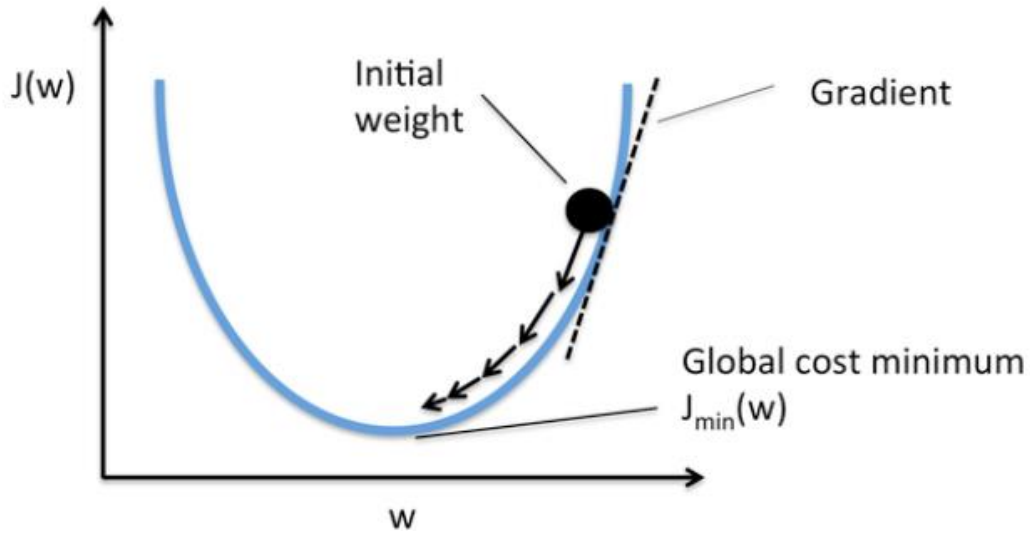


Рисунок 8 – Функція $J(w)$ із кроками градієнтного спуску

Давайте розглянемо алгоритм знаходження глобального мінімуму. Із самого початку вагові коефіцієнти ініціалізуються випадковим чином. Далі, після першого обчислення вихідних значень мережі, виконується обрахунок

часткової похідної помилки для кожного вагового коефіцієнта. Отримані значення демонструють внесок кожної ваги у загальну помилку (total loss). Потім нові коефіцієнти розраховуються за наступною формулою:

$$w_i = w_i - \left(\eta \cdot \frac{\partial(err)}{\partial(w_i)}\right),$$

де w_i – ваговий коефіцієнт окремого зв'язку нейронів;

err – функція втрат (loss function);

η – параметр швидкості навчання, також відомий, як learning rate [6].

Даний процес обчислення, оновлення вагових коефіцієнтів триває до тих пір, доки значення функції втрат не стане достатньо малим (рис. 9).

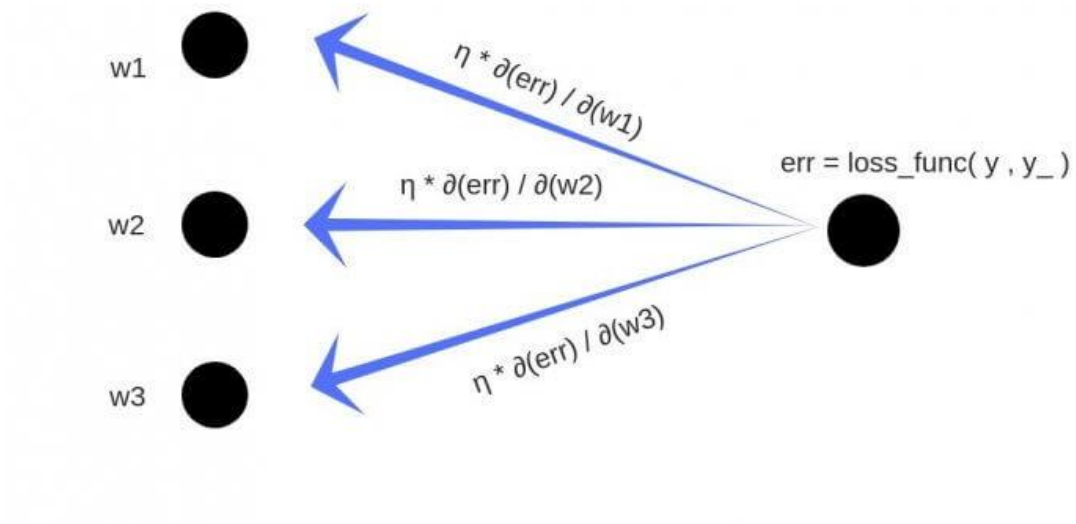


Рисунок 9 – Корегування вагових коефіцієнтів

Потрібно зазначити, що швидкість навчання $\eta \in [0; 1]$ – це гіперпараметр, який ми маємо обрати на основі моделі. Швидкість навчання являє собою розмір кроку, який робиться при градієнтному спуску. Швидкість навчання визначає, як швидко модель адаптується до проблеми. Замалі значення навчання вимагають більшої кількості епох навчання,

враховуючи менші зміни, що вносяться у ваги при кожному оновленні, тоді як великі швидкості навчання призводять до швидких змін і вимагають меншої кількості епох навчання (рис. 10). Занадто велика швидкість навчання може призвести до дуже швидкої збіжності моделі неоптимального рішення, тоді як занадто низька швидкість навчання може призвести до зависання процесу.

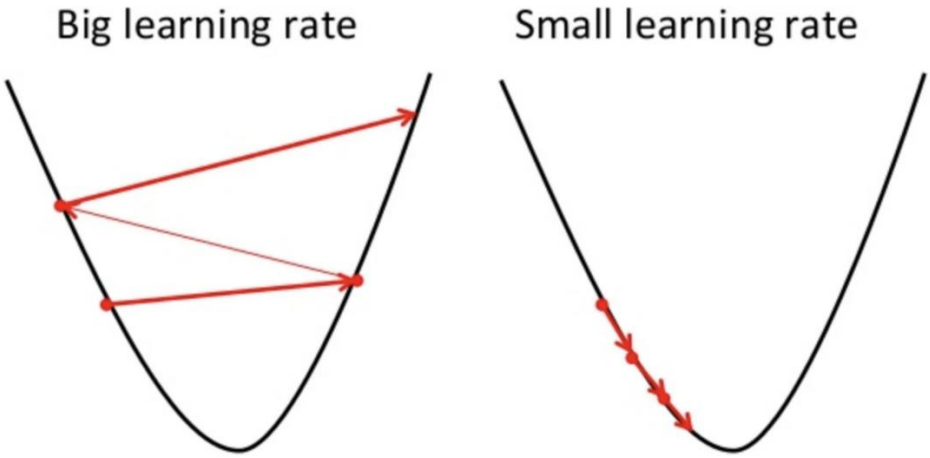


Рисунок 10 – Збіжність при великому та малому learning rate

РОЗДІЛ 2 ПРИКЛАДИ МОДЕЛЕЙ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

2.1 Нейронні мережі глибокого навчання (DNN)

Глибоке навчання (DL) – це клас алгоритмів машинного навчання, які навчаються більш абстрактніше розуміти дані. DL має наступні ознаки:

- використовується каскад (конвеєр, як потік, що послідовно передається) з безлічі обробних рівнів (нелінійних) для вилучення і перетворення ознак;
- базується на вивченні ознак (подання інформації) даних без навчання з учителем. Функції вищого рівня (які знаходяться в останніх рівнях) виходять із функцій нижнього рівня (які знаходяться у початкових рівнях);
- вивчає багаторівневі представлення, що відповідають різним рівням абстракції; рівні утворюють ієрархію уявлення.

У найпростішому випадку нейронна мережа, яка складається не менше ніж із двох наборів нейронів прихованого рівня, кваліфікується як глибока нейронна мережа. DNN обробляють дані складним чином, використовуючи складне математичне моделювання.

Розглянемо модель нейронної мережі, у якої прихований рівень складається із одного набору нейронів (рис. 11). У цій моделі навчається лише один рівень нейронів (зображені зеленим кольором), а потім результат просто передається на вихід.

Якщо ж подивитись на нейронну мережу із двома наборами нейронів прихованого рівня (рис. 11), то можна побачити, що незалежно від того, як навчається зелений прихований шар, він потім передається на синій, де продовжує навчатися. Таким чином, чим більша кількість прихованих шарів, тим більше можливостей навчання мережі.

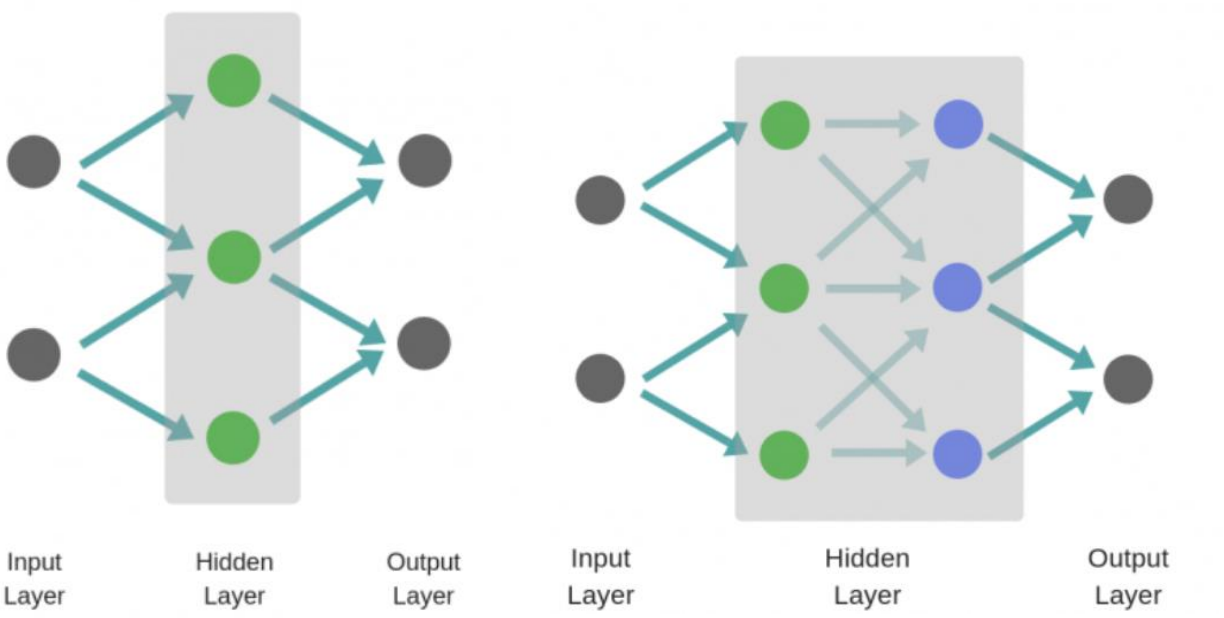


Рисунок 11 – Дві моделі нейронних мереж

DNN не слід плутати із широкою нейронною мережею, де кількість нейронів в одному шарі може бути досить великою, і це призводить до вивчення більшої кількості ознак, але не до глибокого розуміння даних.

Наприклад, під час вивчення англійської граматики, потрібно знати безліч понять. В цьому випадку одношарова широка нейронна мережа працює набагато краще, ніж глибока нейронна мережа, яка значно менша за кількістю нейронів. Але у випадку вивчення перетворення Фур'є, нейронна мережа має бути глибокою, тому що не так багато понять, які потрібно знати, але кожне з них є досить складним і вимагає глибокого розуміння.

Основні недоліки DNN:

– вимагає значно більшої кількості даних для навчання, щоб досягти мінімальної бажаної точності;
– має експоненційну складність;
– занадто глибока нейронна мережа може робити помилкові припущення і намагатиметься знайти псевдозалежності, яких не

існує для простої задачі, таким чином результат буде відрізнятися від бажаного.

2.1.1 Перенавчання та способи його вирішення

Перенавчання – це проблема, яка виникає під час навчання DNN, коли модель починає сприймати помилки у якості надійної інформації. Спочатку розглянемо критерій, який вказує, що відбулося перенавчання мережі, а потім проаналізуємо засоби вирішення даної проблеми.

Помилка під час навчання є функцією різниці між прогнозованим результатом і фактичним результатом кожної точки навчальних даних. На рис. 12 показані помилки навчання та валідації глибокої нейронної мережі під час навчання [7].

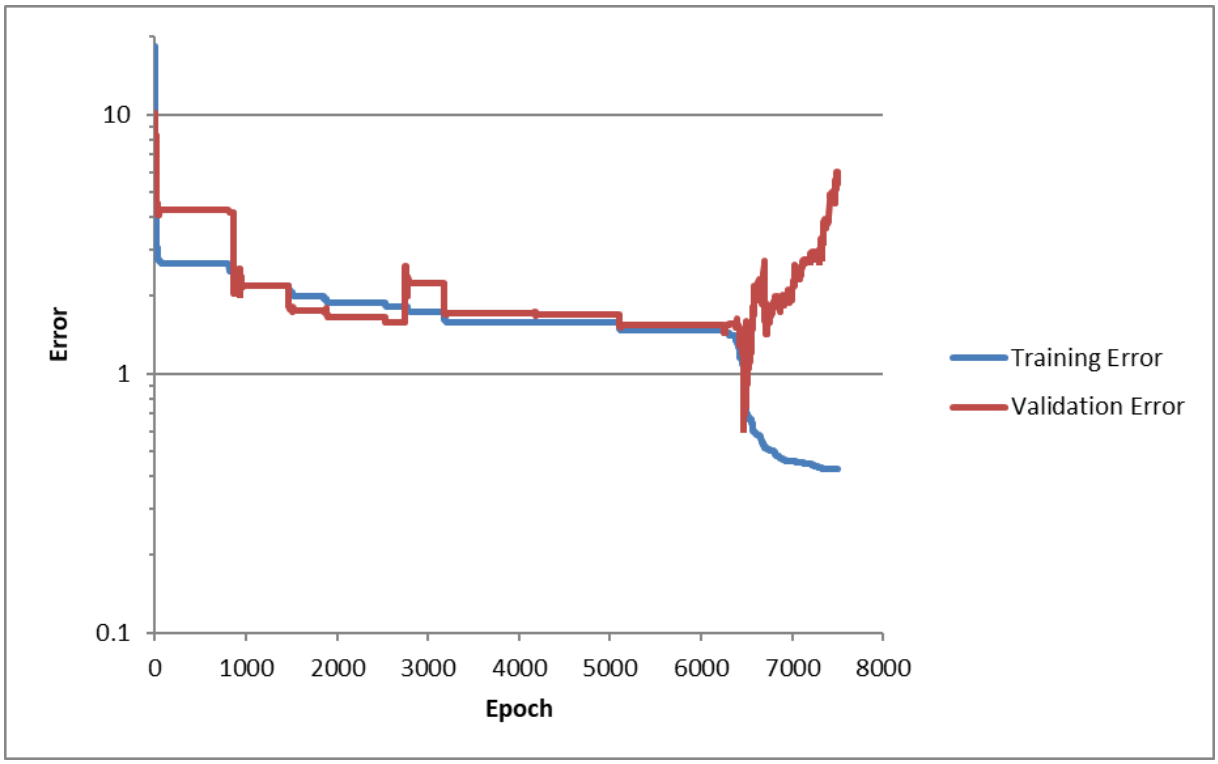


Рисунок 12 – Значення функції втрат для тренувальної та валідаційної вибірок

Помилки тренувальної та валідаційної вибірок мають тенденцію до зниження приблизно до 6500-ї епохи. Потім ці два показника починають розходитися. Саме це розходження і означає, що відбулося перенавчання нейронної мережі. Оскільки помилка тренування продовжує зменшуватись, а помилка валідації збільшується, то це означає, що мережа добре вивчає дані для навчання, але не узагальнює нові.

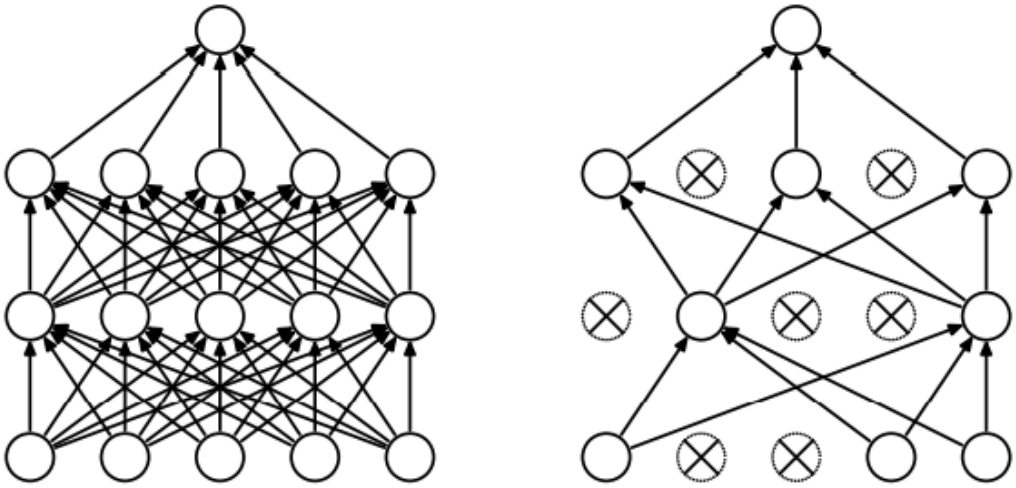
Для того, щоб уникнути процесу перенавчання DNN, можна виконати наступні дії:

- використовувати мінімальну необхідну кількість нейронів у нейронній мережі;
- розбити всю множину спостережень на три вибірки: тренувальну (training), валідаційну (validation), тестову (test);
- можна зупинити процес навчання нейронної мережі, якщо спостерігається розходження в точності вихідних значень тренувальної та валідаційної вибірок;
- використати метод Dropout (див. підрозділ 2.1.2), якщо зменшення кількості нейронів недопустиме;
- використати метод нормалізації вхідних даних (batch normalization) (див. підрозділ 2.1.3), окремо від методу Dropout;
- виконати регуляризацію: в обох типах (training, validation) до помилки навчання додати штраф в залежності від величини вагових коефіцієнтів. Ваги більшого значення можуть бути ознакою перенавчання, коли мережа намагається вивчити зашумлені особливості набору тренувальних даних. Регуляризація зменшує вагові коефіцієнти, зводячи помилку до мінімуму.

2.1.2 Метод Dropout

Глибокі нейронні мережі з великою кількістю параметрів є дуже потужними системами машинного навчання. Основною серйозною проблемою таких мереж, як зазначалося у раніше, є перенавчання. Велика кількість нейронів та зв’язків мережі досить повільні у використанні та навчанні, і саме цей фактор ускладнює боротьбу з перенавчанням, шляхом поєднання прогнозів багатьох різних великих нейронних мереж під час тестування. Для того, щоб вирішити дану проблему зазвичай використовують метод Dropout. Ключова ідея полягає в тому, щоб випадковим чином відкидати деякі нейрони разом із зв’язками, під час, кожної зміни вагових коефіцієнтів нейронної мережі. Таким чином, даний спосіб дозволяє запобігти надмірній коадаптації системи.

Dropout вирішує дві проблеми одночасно: запобігає перенавчанню та забезпечує ефективний спосіб комбінування багатьох різних нейронних мереж. Під вилучанням нейрона, мається на увазі його тимчасове видалення з мережі разом з усіма його вхідними та вихідними з’єднаннями, як зображено на рис. 13 [8].



а) б)

Рисунок 13 – Застосування методу Dropout: а – початкова DNN; б – DNN із застосованим Dropout

Вилучення нейронів є випадковим. У найпростішому випадку кожен нейрон зберігається з фіксованою ймовірністю p, незалежною від інших нейронів, де p просто встановлюють 0,5 або обирають за допомогою набору перевірок. Зазвичай значення 0,5 є близьким до оптимального, для широкого діапазону мереж і завдань. Однак для вхідних нейронів оптимальна ймовірність видалення зазвичай ближче до 1, ніж до 0,5.

При застосуванні методу Dropout, із нейронної мережі виділяється вибірка зменшених у розмірі мереж. Розріджені мережі складаються з усіх нейронів, які залишилися (рис. 13.б). Таким чином нейронна мережа з n одиниць розглядається як сукупність із 2^n можливих розріджених нейронних мереж. При цьому загальна кількість параметрів може або залишитись незмінною $O(2^n)$ або стати меншою [8]. Для кожного кроку навчання, випадково обирається та навчається нова розріджена мережа. Тому навчання початкової нейронної мережі можна розглядати як навчання набору із 2^n розріджених мереж, де кожна навчається або дуже рідко або взагалі не навчається, через ймовірнісну складову.

Варто зазначити, що під час процесу навчання розріджених нейронних мереж, кількість вхідних зв'язків кожного нейрона зменшується пропорційно вірогідності p (рис. 14). Таким чином, вхідне значення нейрона

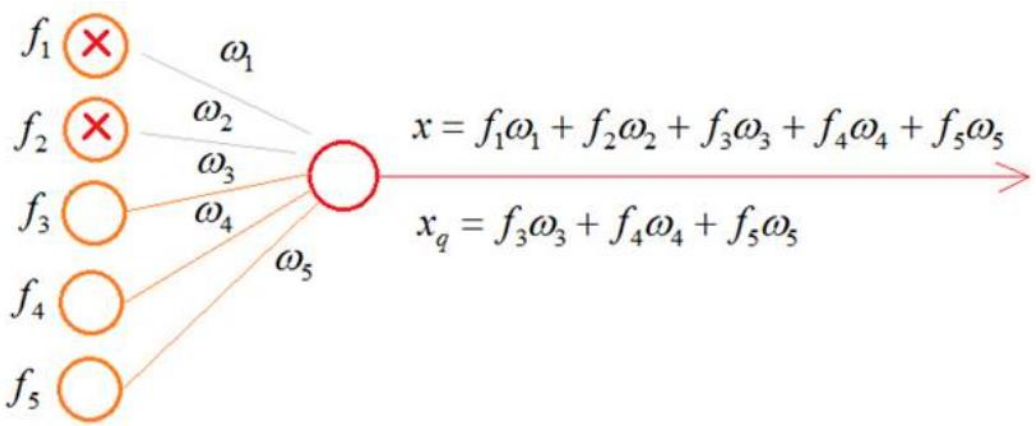


Рисунок 14 – Зменшення кількості вхідних зв'язків нейрона

буде x, а під час навчання x_q. Неважко помітити, що в даній ситуації $x_q < x$. Цей аспект приводить до спотворення вхідних значень, а отже загальний вихід нейронної мережі буде невірним. Для уникнення такого спотворення обчислюють середню кількість вилучених нейронів на поточному рівні. Нехай їх буде n. Тоді значення середнього буде математичним очікуванням:

$$M\{X\} = \sum_{i=1}^{n} x_i \cdot p_i,$$

де X – випадкова величина , що відображає кількість вилучених нейронів на поточному рівні;

x_i = 1 – кількість нейронів, до якої застосовується вірогідність;

p_i = p – вірогідність вилучення.

Таким чином середнє значення буде наступним:

$$M\{X\} = \sum_{i=1}^{n} p_i = n \cdot p.$$

А середня кількість нейронів, що залишилася:

$$n - \ n \cdot p = n \cdot (1 - p) = n \cdot q,$$

де q – вірогідність того, що нейрон не буде вилучений.

Звідси отримуємо, що середнє сумарне вхідне значення нейронів наступних рівнів, у середньому, буде менше на:

$$\frac{x_q}{x} \sim \frac{n \cdot q}{n} = q.$$

Тепер потрібно значення x_q розділити на величину q для того, щоб зберегти масштаб суми:

$$x_q \cdot \frac{1}{q} \sim x.$$

Завдяки такій операції, в середньому, вихідне значення нейронної мережі буде відповідати істинним значенням при її застосуванні, із усіма нейронами без виключень.

2.1.3 Пакетна нормалізація (batch normalization)

Навчання глибоких нейронних мереж із десятками прихованих шарів, є складним завданням. Одним з аспектів цієї проблеми є те, що модель оновлюється порівнево, тобто від вихідного до вхідного, із використанням оцінки помилки, яка передбачає, що ваги в шарах до поточного рівня фіксовані.

Навчання DNN також ускладнюється тим, що розподіл вхідних даних кожного рівня змінюється під час навчання, оскільки змінюються параметри попередніх рівнів. Це уповільнює навчання, вимагаючи нижчої швидкості тренування та ретельної ініціалізації параметрів, а також ускладнює навчання моделей із насиченою нелінійністю. Зміна розподілу внутрішніх вузлів DNN під час навчання називається внутрішнім коваріантним зміщенням (internal covariate shift) [9].

Batch normalization пропонується в якості метода, який допомагає координувати оновлення декількох рівнів моделі нейронної мережі. Нормалізація – це спосіб попередньої обробки даних, який застосовується для приведення числових даних до загального масштабу без зміни їхньої форми. Як правило, коли дані надходять до алгоритму глибокого навчання, то вони схильні змінювати значення до збалансованої шкали. Причина,

через яку доцільно виконати нормалізацію, частково полягає в тому, щоб гарантувати належне узагальнення нашої моделі. Batch normalization дозволяє зробити нейронні мережі більш швидкими та стабільними за рахунок додавання нового рівня до DNN. Новий рівень виконує операції нормалізації та стандартизації на вході, що надходить із попереднього рівня.

Оскільки, зазвичай нейронна мережа навчається з допомогою певних наборів вхідних даних (пакетів), то процес нормалізації відбувається партіями.

Таким чином алгоритм batch normalization виконує приведення математичного очікування та дисперсії до значень 0 та 1. Нехай маємо вхідні дані певного пакету отриманого з рівня h. Тоді ми можемо порахувати середнє значення для прихованої активації:

$$\mu = \frac{1}{m} \cdot \sum_{i=1}^{m} h_i,$$

де m – кількість нейронів на рівні h;

h_i – значення суми на вході кожного нейрона рівня h.

Далі потрібно порахувати стандартне відхилення:

$$\sigma = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} (h_i - \mu)^2}\,.$$

Тепер, маючи середнє значення та стандартне відхилення, ми можемо порахувати нормалізацію. Для цього віднімемо середнє значення μ від кожного h_i, а потім поділимо на суму стандартного відхилення σ та коефіцієнта згладжування ε:

$$h_{i(norm)} = \frac{h_i - \mu}{\sigma + \varepsilon}.$$

Кінцевим етапом пакетної нормалізації є зміна масштабу та зміщення вхідних значень. Для цього використовуються два параметри γ та β, для масштабування та зміщення відповідно, значення з попередньої операції:

$$h_i = \gamma \cdot h_{i(norm)} + \beta.$$

Ці два параметри оновлюються, під час навчання нейронної мережі, яка забезпечує використання оптимальних значень γ і β.

Таким чином, batch normalization прискорює сам процес навчання, а також вирішує проблему внутрішнього коваріантного зміщення. Завдяки цьому гарантується, що вхідні дані для кожного рівня розподіляються навколо одного і того ж середнього значення та стандартного відхилення. Ще одним позитивним фактором є згладжування функції втрат, що у свою чергу також позитивно впливає на швидкість навчання моделі.

2.2 Згорткові нейронні мережі (CNN)

Згорткова нейронна мережа, є нейронною мережею з глибоким навчанням (DNN), що призначена для аналізу даних, що мають сіткову топологію, наприклад зображення. Цифрове зображення є бінарним представленням візуальних даних. Воно містить серію пікселів, розташованих у вигляді сітки, що позначають, наскільки яскравим і якого кольору має бути кожен піксель. CNN широко використовуються в комп’ютерному зорі і стали сучасними для багатьох візуальних додатків, наприклад класифікація зображень, а також досягли успіху в NLP для класифікації тексту та роботі з аудіосигналами.

Згорткові нейронні мережі досить добре розпізнають патерни вхідного зображення, такі як лінії, градієнти, кола та навіть інші фігури. Саме ця властивість згорткових нейронних мереж робить їх корисними у використані комп'ютерним зором. На відміну від ранніх алгоритмів комп'ютерного зору, згорткові нейронні мережі можуть працювати безпосередньо з необробленим зображенням.

CNN є feed-forward мережею, що має, в основному, 30 рівнів. Згорткові нейронні мережі містять безліч згорткових рівнів, накладених один на одного, кожен з яких здатний розпізнавати складніші форми зображення. Сьогодні достатньо чотирьох рівнів згортки, щоб мати можливість розпізнавати рукописні цифри, а за допомогою 25 рівнів можна розпізнати людське обличчя.

2.2.1 Архітектура та принцип роботи згорткової нейронної мережі

Архітектура згорткових нейронних мереж має три основні рівні, а саме [13]:

- згортковий рівень (convolutional layer);
- об'єднуючий рівень (pooling layer);
- повнозв'язний рівень (fully-connected layer).

Згортковий рівень – це перший рівень згорткової мережі. За згортковими рівнями можуть слідувати як додаткові інші згорткові рівні, так і об'єднуючі рівні. Повнозв'язний же рівень є останнім. З кожним рівнем складність CNN збільшується, ідентифікуючи великі частини зображення. У міру того, як дані зображення проходять через рівні CNN, вони починають розпізнавати великі елементи або форми об'єкта, доки, нарешті, не ідентифікують передбачуваний об'єкт.

На рис. 15 можна побачити модель простої CNN, це рання мережа LeNet-5, опублікована Яном Лекуном у 1998 році. LeNet здатна розпізнавати рукописні символи [15].

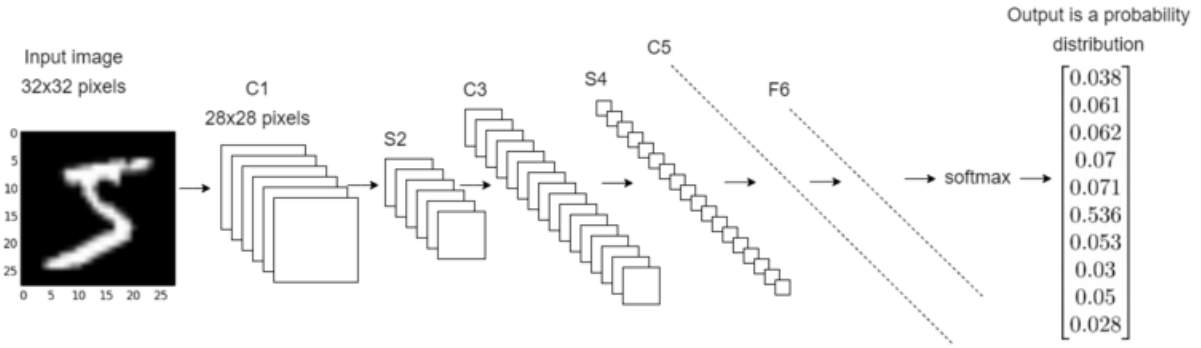


Рисунок 15 – Архітектура LeNet

Convolutional layer є основним для CNN [13]. Він займає основну частину обчислювального навантаження мережі. Для цього потрібні кілька компонентів: вхідні дані, фільтр та карта об’єктів. Припустимо, що на вході маємо кольорове зображення, що складається із матриці пікселів у 3D форматі. Це означає, що вхідні дані матимуть три виміри – висоту, ширину та глибину – яка відповідає кольорам пікселя зображення формату RGB. У нас також є детектор ознак, відомий як ядро або фільтр, який буде переміщатися полями зображення, перевіряючи, чи присутня певна ознака у зображення. Цей процес відомий як згортка.

Ядро ознак є двовимірним масивом вагових коефіцієнтів. Розмір фільтра зазвичай є матрицею із трьома рядками та трьома стовпцями. Фільтр (ядро) застосовується до певної області зображення шляхом обчислення скалярного добутку між вхідними пікселями та фільтром. Отримане значення скалярного добутку потім подається у вихідний масив. Після цього фільтр зміщується на певний крок та повторює процес доти, доки ядро не охопить усе зображення. Остаточний результат серії точкових добутків вхідних даних та фільтра називають картою ознак (рис. 16).

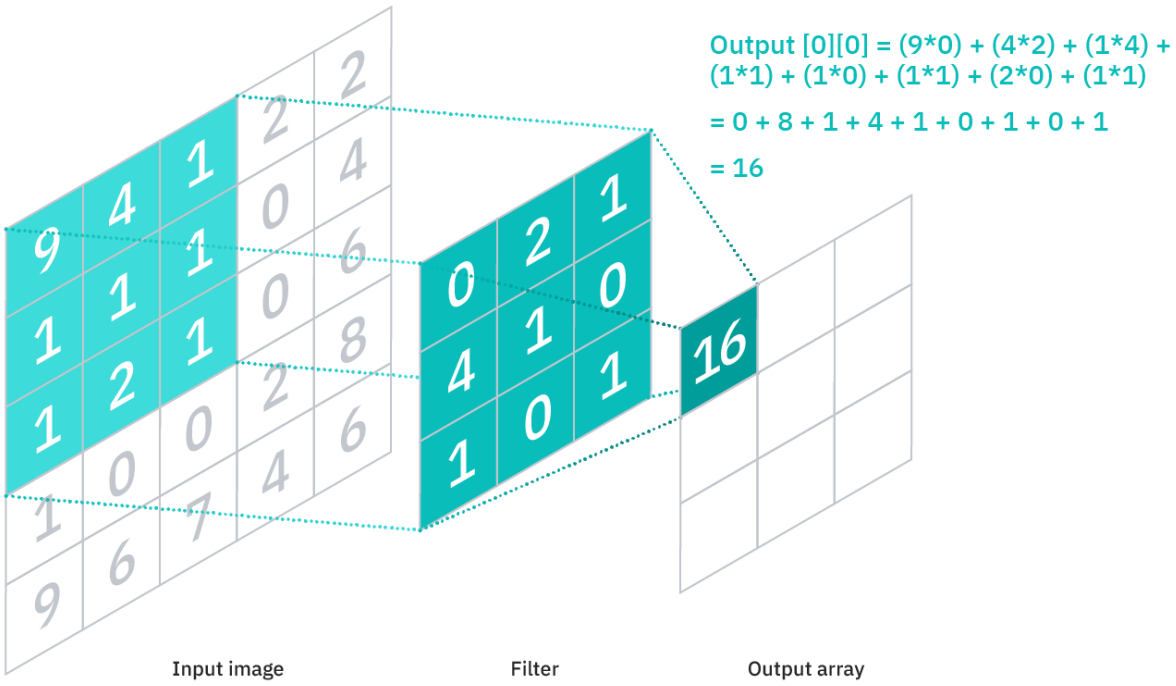


Рисунок 16 – Процес отримання карти ознак

Можна помітити, що на рис. 16 кожне вихідне значення на карті ознак не обов'язково має бути пов'язаним із кожним пікселем у вхідному зображенні. Оскільки вихідній матриці не потрібно прямо зіставляти кожне вхідне значення, згорткові рівні зазвичай називають «частково зв'язаними».

Слід зазначити, що вагові коефіцієнти в детекторі ознак залишаються фіксованими доки ядро переміщується зображенням. Деякі параметри, такі як значення ваг, корегуються під час навчання за допомогою процесу зворотного розповсюдження помилки та градієнтного спуску (див. підрозділ 1.3). Однак є три гіперпараметри, які впливають на розмір об'єму вихідної матриці, які необхідно встановити перед початком навчання нейронної мережі, а саме:

– кількість фільтрів, яка впливає на глибину виведення. Наприклад, за допомогою трьох різних фільтрів отримаємо три різні карти ознак, створивши глибину, рівну трьом;

- крок зміщення – це відстань або кількість пікселів, на яку ядро зміщується перед виконанням скалярного множення. Зазвичай крок зміщення дорівнює одиниці;
- заповнення нулями, яке використовується, якщо фільтри не відповідають вхідному зображенню. Дана процедура робить всі елементи, що виходять за межі вхідної матриці, рівними нулю, що призводить до більшого або однакового розміру вихідних даних.

Зрештою згортковий рівень перетворює зображення на числові значення, що дозволяє нейронній мережі інтерпретувати та виявляти відповідні закономірності.

Об’єднуючий рівень замінює вихідні дані мережі у певних місцях, отримуючи зведену статистику найближчих вихідних даних. Це допомагає виконати аналіз порахованих ознак на більш крупному масштабі та зменшити просторовий розмір карт ознак, що у свою чергу зменшує необхідний обсяг обчислень. Таким чином, розмірність карт ознак зменшують за допомогою однієї із наступних операцій:

- MaxPooling – відбір найбільших значень (рис. 17);
- MinPooling – відбір найменших значень;
- AveragePooling – відбір середніх значень.

Завдяки даним операціям, в результаті отримуємо аналіз даних на більшому масштабі, і нейрони наступного рівня здатні виділяти більш загальні ознаки на зображенні. Як правило, великі значення відповідають наявності певної ознаки на зображенні, а малі – її відсутності. Таким чином, відбираючи максимальні числа, ми тим самим відбираємо знайдені ознаки і зберігаємо їх для подальшого аналізу на більшому масштабі. Саме тому

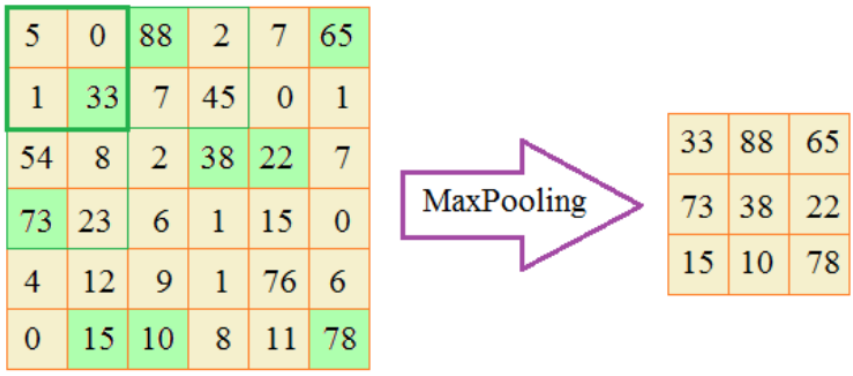


Рисунок 17 – Зменшення карти ознак вдвічі (операція MaxPooling)

операція MaxPooling, в основному, використовується в CNN при аналізі зображень.

Як зазначалося раніше, значення пікселів вхідного зображення не пов'язані безпосередньо з вихідним рівнем на частково пов'язаних рівнях. Однак на повнозв'язному рівні кожен вузол вихідного шару безпосередньо з'єднується з вузлом попереднього рівня. Даний рівень виконує завдання класифікації на основі ознак, отриманих за допомогою попередніх рівнів та їх ядер. У той час як згорткові та об'єднуючі рівні зазвичай використовують функції ReLu (рис. 6), повнозв'язні рівні мають справу в більшості випадків із функцією активації soft-max, яка для відповідної класифікації вхідних даних, на виході повертає значення ймовірності від 0 до 1.

2.3 Рекурентні нейронні мережі (RNN)

Рекурентні нейронні мережі – це клас нейронних мереж, які допомагають моделювати дані послідовностей. RNN демонструють поведінку, аналогічну тому, як функціонує мозок людини. Дана модель нейронних мереж надає прогнозовані результати у послідовних даних, в той час, як інші алгоритми не здатні на це. RNN є потужним і надійним типом нейронної мережі і відносяться до найбільш перспективних алгоритмів, оскільки це єдиний алгоритм з внутрішньою пам'яттю. Завдяки даній

особливості RNN можуть запам’ятовувати важливі аспекти про отримані дані, що в свою чергу дозволяє їм дуже точно передбачати наступні кроки. Ось чому вони є одним із найкращих виборів для послідовних даних, таких як часові ряди, текст, NLP, фінанси, погода, відео, аудіо. Рекурентні нейронні мережі можуть набагато глибше формувати розуміння послідовності та її контексту порівняно із іншими моделями нейронних мереж.

У традиційних нейронних мережах вхідні та вихідні дані не залежать один від одного, тоді як вихідні дані в RNN залежать від попередніх елементів у послідовності. Рекурентні мережі також спільно використовують параметри кожного рівня мережі. У мережах із прямим зв’язком кожен вузол має різні вагові коефіцієнти, у той час як RNN має однакові ваги на кожному рівні мережі [16]. Загальний вигляд RNN представлений на рис. 18. Для прикладу, якщо ми хочемо спрогнозувати

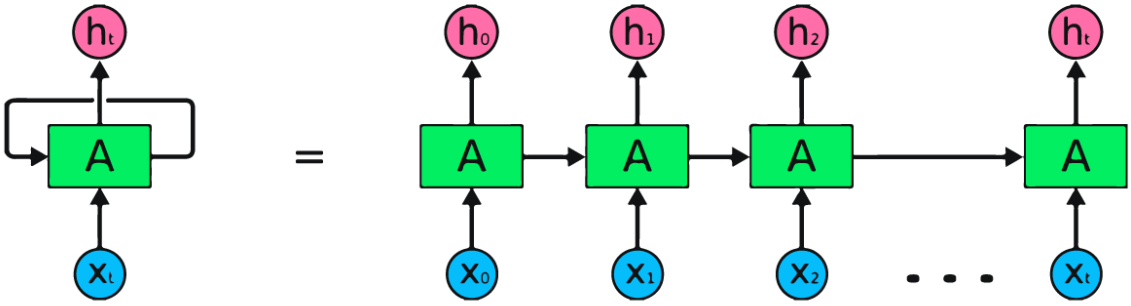


Рисунок 18 – Загальне представлення RNN

ціни акцій, використовуючи певний масив цін, то кожен вхід від x_0 до x_t буде містити попереднє значення. Тобто x_0 буде позначати перший елемент масиву, x_1 – другий, і так далі. Таким чином значення x_i будуть використовуватися для передбачення наступного значення послідовності.

У RNN інформація циклічно повторюється, тому результат визначається поточним та раніше отриманими вхідними даними. Вхідний

рівень x_t обробляє вхідні дані і передає його на середній рівень A (рис. 18). Середній рівень, у свою чергу складається з кількох прихованих рівнів, кожен зі своїми функціями активації, ваговими коефіцієнтами та зміщеннями (bias). Ці параметри стандартизовані для прихованого рівня, тому замість створення кількох прихованих рівнів створюється один рівень, який далі повторюється.

Замість використання традиційного зворотного поширення рекурентні нейронні мережі використовують алгоритми зворотного поширення в часі (BPTT) для визначення градієнта. BPTT підсумовує помилку на кожному тимчасовому кроці, оскільки RNN розділяє параметри на кожному рівні.

2.3.1 Типи рекурентних нейронних мереж

Існує чотири типи RNN на основі різної довжини вхідних та вихідних даних (рис. 19).

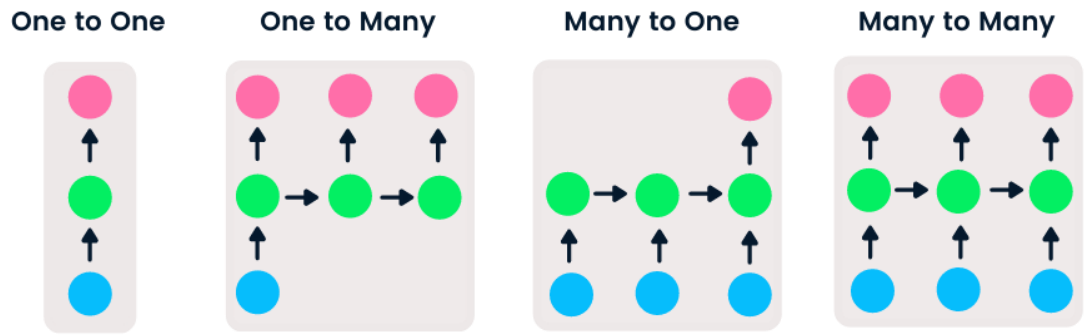


Рисунок 19 – Типи RNN

Яку архітектуру потрібно обрати залежить від прикладного завдання. Наприклад, при перекладі однієї мови на іншу маємо довільну послідовність слів як на вході так і на виході, отже, в даному випадку слід обрати архітектуру Many to Many.

One to one – це найпростіша RNN. Вона зазвичай використовується для завдань машинного навчання, які мають один вхід та вихід. Застосовується зазвичай дуже рідко.

Many to One має декілька вхідних значень та одне вихідне. Цей тип використовується для аналізу емоційного забарвлення тексту (на вході текст, на виході категорії: позитивний, нейтральний, негативний) [17].

One to Many має один вхід та кілька виходів. Використовується для створення опису зображення, тобто на вхід отримуємо карту ознак зображення, а на виході – текст опису.

2.3.2 Обмеження рекурентних нейронних мереж

Прості RNN моделі зазвичай стикаються з двома основними проблемами. Ці проблеми пов’язані з градієнтом, який є нахилом функції втрат разом з функцією помилок.

Проблема зникаючого градієнта, виникає, коли градієнт стає настільки маленьким, що оновлення параметрів стає незначним. Врешті-решт алгоритм перестає навчатися.

Протилежна ситуація виникає, коли градієнт стає занадто великим. Це робить модель нестабільною. У цьому випадку накопичуються великі градієнти помилок, і вагові коефіцієнти моделі стають занадто великими. Ця проблема може призвести до збільшення часу навчання та зниження продуктивності моделі.

Вирішенням цих проблем є або зменшення кількість прихованих рівнів нейронної мережі, що знизить складність RNN, або використання передових архітектур RNN, таких як LSTM мереж та GRU.

РОЗДІЛ 3 РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА ЗОБРАЖЕННІ

Розпізнавання об'єктів зображення відбувалося за допомогою згорткової нейронної мережі VGG-16, яка натренована на більш ніж 10 мільйонів зображень бази даних ImageNet. Дана мережа може класифікувати зображення на 1000 категорій об'єктів.

VGG-16 – модель згорткової нейронної мережі, яка призначена для класифікації та виявлення об'єктів на зображенні. У 2014 році відбулося змагання із розпізнавання зображень, на якому VGG-16 досягла неймовірної точності, а саме 92,7% [12]. Цей показник близький до людського.

3.1 Архітектура та процес розпізнавання мережею VGG-16

Архітектуру VGG-16 зображено на рис. 20. На вхід до нейронної

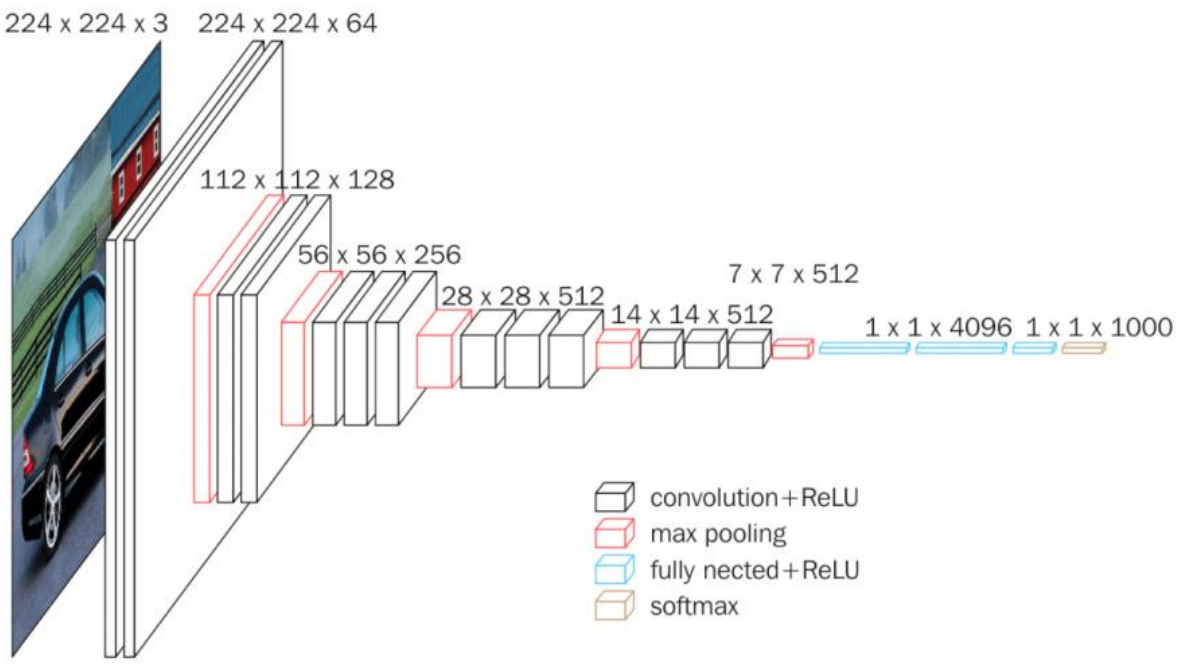


Рисунок 20 – Архітектура згорткової нейронної мережі VGG-16

мережі надходить зображення формату RGB розміром 224 на 224 пікселя. Далі зображення проходить через перші два згорткових рівні, де кожен рівень має 64 фільтри розміром 3 на 3 (див. підрозділ 2.2.1). Далі виконується операція MaxPooling (див. підрозділ 2.2.1), яка зменшує розмір отриманих карт ознак удвічі. Таким чином матриця кожної із карт ознак матиме розмір 112 на 112 елементів. Результат передається наступним двом згортковим рівням, які у свою чергу мають 128 фільтрів, і так далі. Таким чином, вхідне зображення проходить через 13 згорток та 5 операцій MaxPooling. Врешті, отримуємо тензор розмірності 7 на 7 із 512 картами ознак, який надходить, у якості вхідних даних, до повнозв'язної нейронної мережі, яка складається з двох прихованих рівнів (кожен містить 4096 нейронів) та вихідного рівня (містить 1000 нейронів). Усі приховані рівні мають функцію активації ReLU (рис. 6), а вихідний рівень – функцію soft-max. Кількість нейронів вихідного рівня відповідає кількості класів зображень, на яких відбувалося навчання VGG-16. Таким чином на виході міститься 1000 значень, які демонструють ймовірність належності розпізнаного об'єкта певному класу.

Кожен номер вихідного нейрона відповідає власному класу зображень. Результатом буде номер нейрона з найбільшим значенням. Конфігураційна схема VGG-16 представлена на рис. 21.

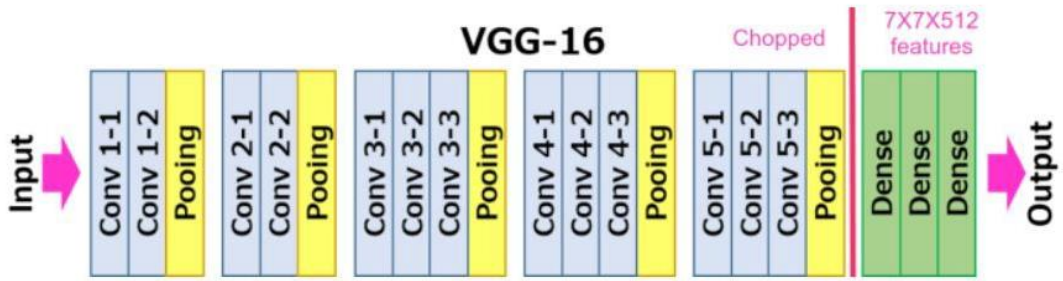


Рисунок 21 – Загальна схема VGG-16

Варто пояснити, навіщо у моделі підряд ідуть два або більше згорткових рівні та чому не можна використати натомість один. Такий підхід використовуються через те, що в даному випадку кількість параметрів, що корегуються буде меншою.

Для прикладу, замість перших двох згорткових рівнів із розмірністю фільтра 3 на 3 можна взяти одну згортку, у якої фільтр буде розмірності 5 на 5. В даному випадку кількість параметрів буде рівна 26 (ядро розмірності 5 на 5 та ще один параметр – bias). Якщо ж використати дві згортки із ядром розмірності 3 на 3, то кількість параметрів буде рівна 20.

3.2 Реалізація та результати

Для того, щоб використати мережу VGG-16, при реалізації задачі розпізнавання, потрібно встановити пакет Keras. Для підключення моделі використовується метод keras.applications.VGG16(), який повертає модель згорткової мережі VGG-16. Модель можна досить зручно налаштувати під власні потреби.

На рис. 22 зображено основні параметри, які потрібні для налаштування моделі:

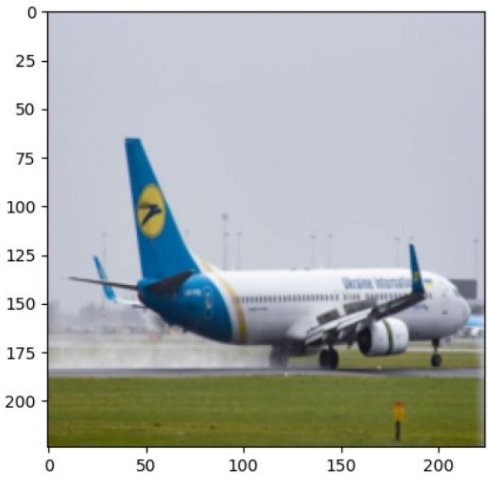
```
keras.applications.VGG16(
    include_top=True,
    weights="imagenet",
    classes=1000,
    classifier_activation="softmax",
)
```

Рисунок 22 – Налаштування моделі VGG-16

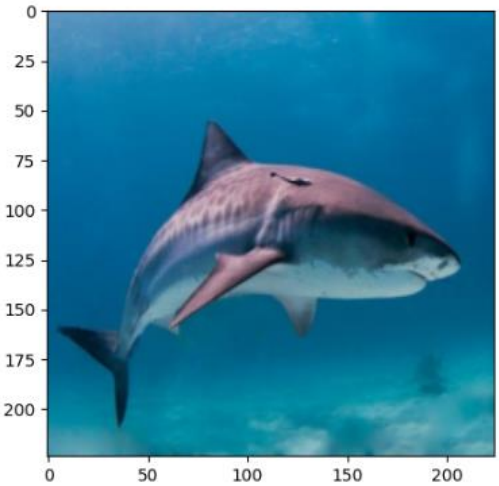
– include_top – булеве значення, яке вказує чи потрібно використовувати повнозв’язні рівні;
– weights вказує на джерело значень вагових коефіцієнтів;

– classes відповідає за кількість вихідних нейронів мережі;
– classifier_activation – функція активації, яка буде застосована до вихідних нейронів.

Для перевірки результату виконання було створено вибірку зображень різних класів. Деякі результати розпізнавання об’єктів на зображенні можна побачити на рис. 23.




```
1/1 [==============================] - 1s 547ms/step
 'airliner'
```




```
1/1 [==============================] - 0s 420ms/step
 'tiger shark, Galeocerdo cuvieri'
```

Рисунок 23 – Результати розпізнавання

РОЗДІЛ 4 КОЛЬОРИЗАЦІЯ ЗОБРАЖЕНЬ

4.1 Алгоритм

Задача кольоризації полягає в тому, щоб на основі зображення у відтінках сірого отримати кольорове зображення. Кольорові зображення для даної задачі зручніше за все представляти не у форматі RGB, а у форматі Lab, L позначає рівень яскравості (градації сірого), a і b – кольорові спектри зелено-червоний та синьо-жовтий (рис. 24). При чому пікселі рівня L знаходяться в діапазоні від 0 до 100, a і b – від -128 до 127. Таким чином нейронній мережі буде достатньо спрогнозувати лише два канали: a і b, адже канал L наявний у якості вхідних даних.

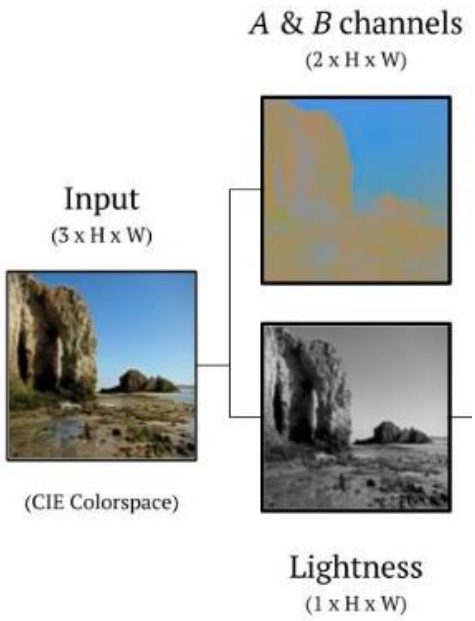


Рисунок 24 – Розбиття зображення на дві складові

Весь (спрощений) процес кольоризації можна резюмувати так [18]:

- перетворення всієї тренувальної вибірки зображення з формату RGB у формат Lab;
- використання каналу L як вхідного значення у мережу;

- навчання мережі передбачати канали a та b;
- об’єднання вхідного канал L із передбаченими каналами a та b;
- перетворення отриманого зображення у початковий формат RGB.

4.2 Реалізація, результати та модель нейронної мережі

Для початку потрібно перетворити формат зображення із RGB у Lab. Для цього потрібно використати пакет skimage, який надає наступні два методи:

- rgb2lab – перетворює зображення з RGB в Lab;
- lab2rgb – перетворює зображення з Lab на RGB.

Тепер, маючи можливість робити перетворення форматів кольорів, потрібно привести зображення до певного виду даних, які будуть надходити на вхідний рівень нейронної мережі. В першу чергу зображення потрібно привести до розміру 256 на 256 пікселів, а потім перетворити його на масив numpy. Далі, за допомогою функції rgb2lab, приводимо зображення до формату Lab. Варто зауважити, що на вхід функція rgb2lab отримує зображення у вигляді дійсних чисел у діапазоні від 0 до 1, через це потрібно значення пікселів помножити на нормуючий множник. Далі виділяємо окремо компоненту L, яка буде вхідним значенням нейронної мережі, та компоненти a і b, які будуть прогнозованими значеннями для тренувальної вибірки.

Наступним кроком виконаємо нормалізацію вихідних значень, тобто приведемо їх до діапазону від -1 до 1. Дану операцію потрібно виконати, оскільки функція активації для вихідних значень буде повертати числа діапазону від -1 до 1.

Тепер давайте розглянемо саму модель нейронної мережі, яка є згортковою (рис. 25).

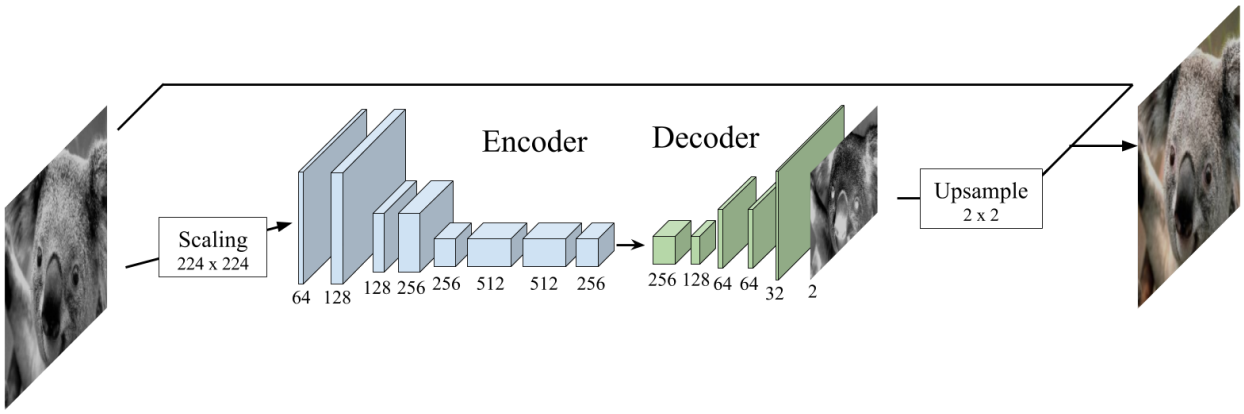


Рисунок 25 – Модель згорткової нейронної

За допомогою пакету Keras можна побудувати дану нейронну мережу (рис. 26).

```
model = Sequential()
model.add(InputLayer(input_shape=(None, None, 1)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))
model.add(UpSampling2D((2, 2)))
```

Рисунок 26 – Побудова моделі згорткової нейронної мережі

Можна помітити, що другий рівень згортки має 128 фільтрів із розмірністю 3 на 3. Третя згортка має аналогічну кількість рівнів, але крок зміщення ядра (див. підрозділ 2.2.1) дорівнює двом. В такий спосіб відбувається масштабування ознак за рахунок збільшення кроку (рис. 27). Може виникнути питання, чому не використати операцію MaxPooling, яка має той же сенс – масштабування ознак зображення. Справа в тому, що MaxPooling добре визначає значиму частину особливостей зображення, але це дещо спотворює взаємне розташування пікселів. У задачах кольоризації таке спотворення недопустиме.

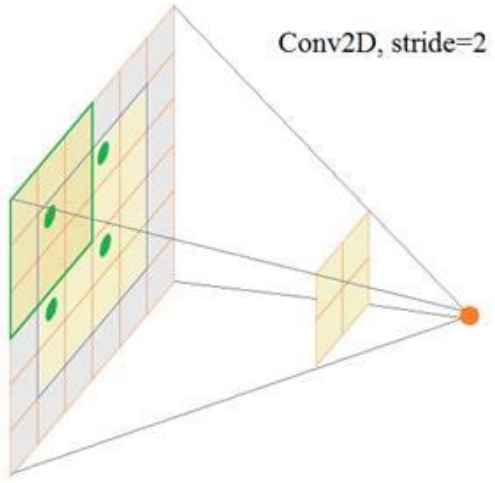


Рисунок 27 – Зміщення ядра із кроком 2

Процес проходження через згортки продовжується, доки дані не дійдуть до рівня UpSampling2D. Завдання цього рівня збільшити розмір кожного елемента карт ознак. Збільшення відбувається шляхом масштабування кожної клітинки матриці. Значення нових клітинок дорівнюють значенню початкової клітинки до масштабування. У решті решт, отримуємо збільшене грубе представлення карт ознак на кожному каналі (рис. 28).

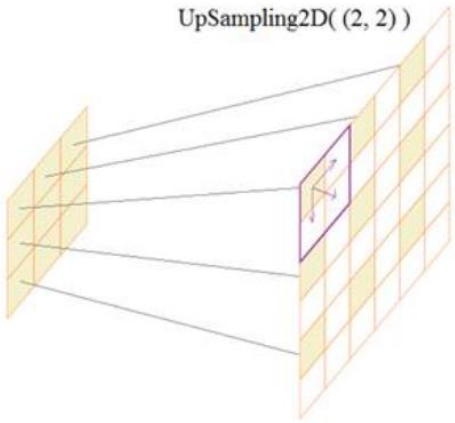


Рисунок 27 – Схематичне представлення UpSampling2D

На останньому вихідному рівні отримуємо два канали a і b, розміри яких відповідають розмірам вхідного зображення. У якості функції активації обирається tanh (див. підрозділ 1.2.3), для того щоб значення виходу відповідало діапазону від -1 до 1. У якості оптимізатора було обрано оптимізацію за Adam (див. підрозділ 1.3), а критерій якості – MSE.

Під час виконання даної задачі було проведено 100 епох навчання нейронної мережі на вибірці із 140 зображень певного виду птахів. Також було проведено порівняння кольоризації для різної кількості епох навчання (рис. 28).

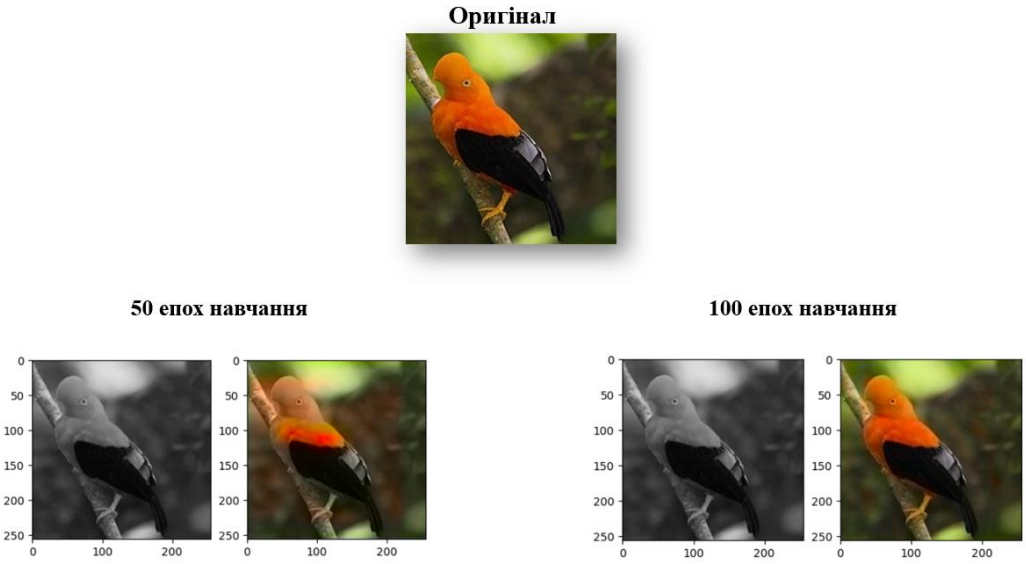


Рисунок 28 – Результати виконання кольоризації

4.3 Кольоризація різних класів зображень

Алгоритм кольоризації та побудована нейронна мережа, що описані в попередньому підрозділі, можуть досить гарно працювати для однотипних зображень, наприклад тварин певного виду, зображень лісів або водоймищ. Якщо обрати декілька різних типів зображень і навчити мережу, то результат не буде відповідати очікуванням. Саме тому для кольоризації різних типів зображень було запропоновано концепцію, яка покращує даний алгоритм. Рішення полягає в додаванні окремої нейронної мережі, яка виконує звичайну класифікацію зображення (рис. 29). Таким чином, класифікувавши зображення, нейронна мережа матиме можливість визначити, яким способом потрібно виконати кольоризацію об'єктів на зображенні. Для класифікації зображення було використано одну із навчених нейронних мереж VGG-19 (подібна до VGG-16).

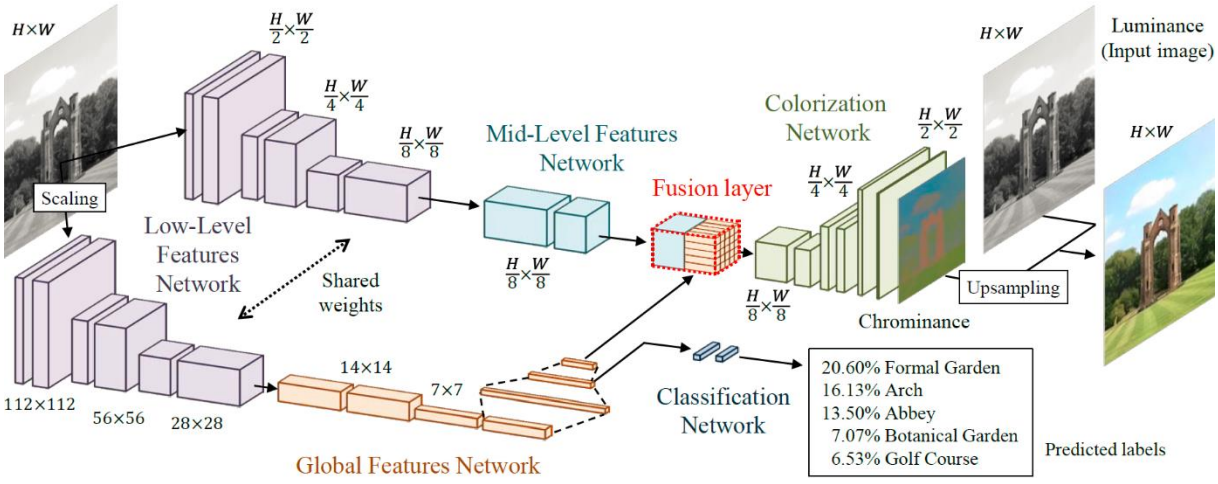


Рисунок 29 – Схема поєднаних нейронних мереж класифікації та кольоризації

РОЗДІЛ 5 СТИЛІЗАЦІЯ ЗОБРАЖЕНЬ

Стилізація зображення – це метод змішування стилю одного зображення з іншим із збереженням його вмісту. Дана процедура надає можливість привести зображення до певного стилю, тобто із двох картинок отримати одне, яке матиме об’єднану структуру та інформацію зображень.

5.1 Алгоритм

Ідея алгоритму полягає в тому, щоб обрати певне зображення та розглядати пікселі, як параметри, які можна коригувати (аналогічно як це виконується із ваговими коефіцієнтами). Критерій якості має бути обраний таким чином, щоб він зменшувався при наближенні початкового зображення до стилізованого [19]. Таким чином, основна задача нейронної мережі полягає у обчисленні критерія якості. Для цього використовуються згорткові нейронні мережі.

Загальний критерій якості розподіляється на дві незалежні частини:

- ступінь відповідності згенерованого зображення до оригінального;
- ступінь стилізації згенерованого зображення.

Ці два показники обчислюються незалежно один від одного, а потім формують загальну оцінку якості.

Розглянемо кожен критерій окремо. Для початку розглянемо ступінь перенесення стилю. Перш за все потрібно мати змогу оцінювати зображення на абстрактному рівні, виділяючи певні ознаки об’єктів. Для цього використаємо натреновану CNN VGG-19 [19], що подібна до VGG-16, але має на три згортки більше. Виконаємо наступні дії:

- пропустимо через VGG-19 початкове зображення об’єкта та отримаємо на виході набір певних ознак;

– пропустимо через VGG-19 згенероване зображення (із застосуванням стилю) та отримаємо на виході інший набір ознак;

– в кінці порахуємо показник якості. Чим краще результат тим значення буде нижче, оскільки ми мінімізуємо значення помилки.

Таким чином візьмемо останній рівень мережі VGG-19 (рис. 30) та порахуємо суму різниць квадратів ознак:

$$L = \frac{1}{4 \cdot n_H \cdot n_W \cdot n_C} \cdot \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} \sum_{k=1}^{n_C} \left(p_{i,j,k} - t_{i,j,k}\right)^2, \qquad (5.1)$$

де $\{p_{i,j,k}\}$ – набір ознак згенерованого зображення;

$\{t_{i,j,k}\}$ – набір ознак оригінального зображення об’єкта;

n_H, n_W – висота та ширина карт ознак;

n_C – кількість карт ознак, тобто каналів.

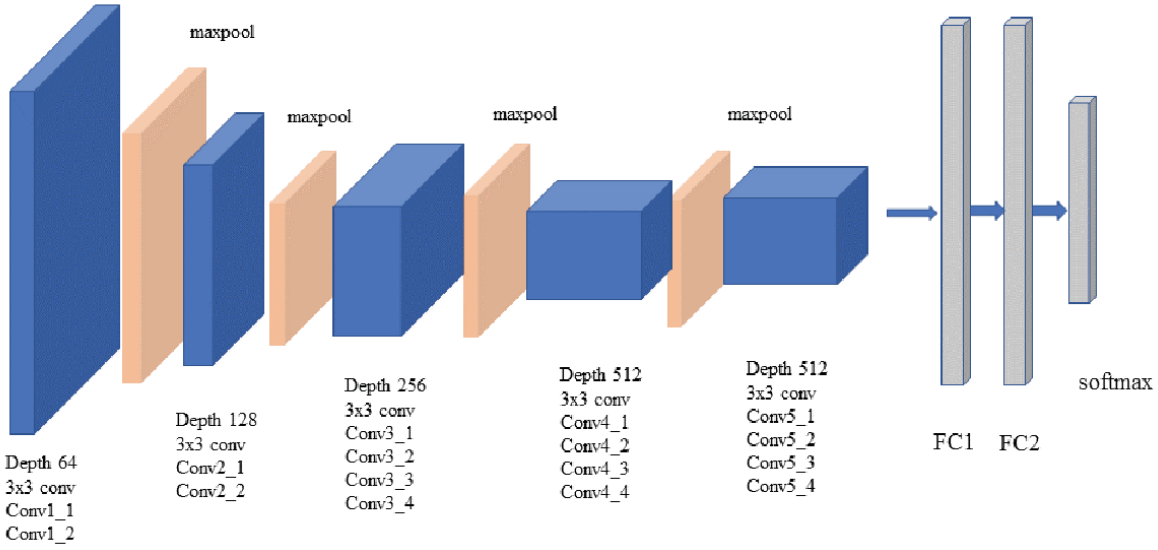


Рисунок 30 – Схема моделі VGG-19

Тепер розглянемо ступінь стилізації генерованого зображення. Тут нам також знадобиться VGG-19, яка буде визначати тимчасовий параметр величини перенесення стилю. Пропустимо зображення стилю через мережу та витягнемо дані з рівня Conv5-1 (рис. 30). Весь тензор карт ознак перетворимо у матрицю для зручності наступних обчислень. Таким чином розмір матриці буде $n_H \cdot n_W$ стовпців та n_C рядків, де кожен рядок матриці буде вектором, отриманим із конкретної карти ознак. Наступним кроком потрібно отримати матрицю Грама. Для цього потрібно перемножити матрицю карт ознак на саму себе, але в транспонованому вигляді (рис. 31).

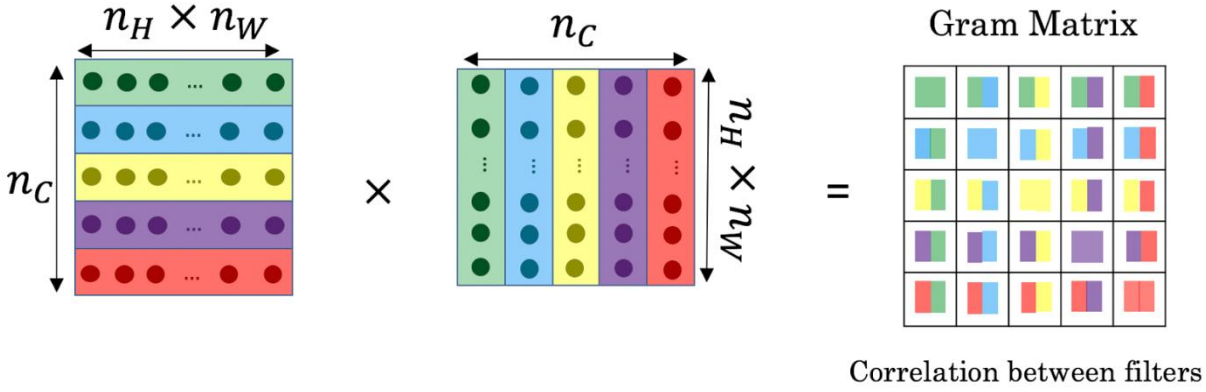


Рисунок 31 – Процес отримання матриці Грама

Таким чином, елементами матриці Грама є скалярні добутки векторів ознак для різних каналів. Така операція дає нам певну проекцію одного вектора на інший, що у свою чергу є деякою мірою схожості ознак.

Аналогічну операцію потрібно виконати для генерованого зображення. Пропустимо його через VGG-19 та знайдемо матрицю Грама. Таким чином, маючи дві матриці Грама, можна порахувати величину розходження стилів на поточному рівні нейронної мережі:

$$J^{(5)} = \frac{1}{{n_C}^2 \cdot (n_H \cdot n_W)^2} \cdot \sum_{j=1}^{n_W} \sum_{k=1}^{n_C} \left(G^P_{i,j,k} - G^S_{i,j,k}\right)^2, \qquad (5.2)$$

де G^P – матриця Грама для генерованого зображення;

G^S – матриця Грама для зображення стилю;

$J^{(5)}$ – результат обчислення для 5-го рівня (Conv5-1).

Стиль, на відміну від об'єктів, має важливі деталі на різних рівнях абстракції. Саме тому потрібно повторити розглянуті вище операції для початкових рівнів VGG-19, а саме: Conv1-1, Conv2-1, Conv3-1, Conv4-1. У результаті матимемо наступні значення: $J^{(1)}$, $J^{(2)}$, $J^{(3)}$,$J^{(4)}$,$J^{(5)}$. Далі кожне таке значення множиться на власний ваговий коефіцієнт. Врешті решт, шляхом додавання $J^{(1)}$, $J^{(2)}$, $J^{(3)}$,$J^{(4)}$,$J^{(5)}$, отримаємо загальне значення J розходження стилів на всіх обраних рівнях.

Тепер, маючи J та L, можна порахувати загальний показник якості:

$$G = \alpha \cdot L + \beta \cdot J, \tag{5.3}$$

де α та β – параметри, які вказують, наскільки важливо враховувати інформацію основного зображення та стилю. Підбираючи ці значення, можна отримувати різні ступені стилізації.

Визначивши критерій якості, алгоритм градієнтного спуску змінює пікселі основного зображення, таким чином, щоб мінімізувати G. Шляхом виконання ітерацій обирається найкращий показник G, за яким буде згенероване вихідне зображення.

5.2 Реалізація та результати

Для початку потрібно завантажити два зображення (основне та стиль), а потім привести їх до розміру 224 на 224 пікселі. Наступним кроком потрібно привести формат RGB до формату BGR, а потім зменшити середні значення кожного каналу. Далі потрібно під'єднати модель VGG-19, але без

повнозв’яного рівня, оскільки нас цікавлять лише певні згортки. Для того, щоб вибирати карти ознак певних згорток мережі VGG-19, потрібно створити нову нейронну мережу на основі VGG-19. Таким чином нова мережа буде мати лише потрібні копії згорток мережі VGG-19. Перелік згорток, які потрібно обрати зображено на рис. 32.

```
main_picture_layer = ["block5_conv2"]
style_layer = ["block1_conv1",
               "block2_conv1",
               "block3_conv1",
               "block4_conv1",
               "block5_conv1"]
```

Рисунок 32 – Масиви назв обраних згорток мережі VGG-19

Тепер, маючи зменшену копію VGG-19, можна пропускати через модель зображення та отримувати карти ознак на відповідних рівнях.

Наступним кроком створимо функції втрат. Перша функція буде відповідати за показник якості відповідності згенерованого зображення та оригінального. Ця функція обчислюється за формулою (5.1). Реалізацію зображено на рис. 33, де метод square() повертає тензор квадратів різниць між ознаками генерованого та основного зображень. Метод reduce_mean() повертає середнє арифметичне.

```
def loss_func(primary_cont, final_cont):
    loss = tf.reduce_mean(tf.square(primary_cont - final_cont))
    return loss
```

Рисунок 33 – Функція втрат контенту

Для обчислення втрат за ступенем стилізації, потрібно визначити функцію, яка буде повертати матрицю Грама. Реалізація представлена на

рис. 34. Спочатку отримуємо кількість каналів, потім тензор приводимо до матричного вигляду, і в кінці множимо матрицю на саму себе у транспонованому вигляді. Повертаємо результат поділений на кількість рядків матриці, таким чином відбувається усереднення значень.

```
def get_gram_matrix(tensor):
    channels = int(tensor.shape[-1])
    tensor_2d = tf.reshape(tensor, [-1, channels])
    n_elements = tf.shape(tensor_2d)[0]
    gram = tf.matmul(tensor_2d, tensor_2d, transpose_a=True)
    return gram / tf.cast(n_elements, tf.float32)
```

Рисунок 34 – Обчислення матриці Грама

Тепер можемо визначити функцію, яка буде рахувати втрати для зображення стилю та генерованого зображення, згідно формули (5.2). Реалізація представлена на рис. 35.

```
def style_loss_func(primary_style, target_gram):
    gram_style = get_gram_matrix(primary_style)
    loss = tf.reduce_mean(tf.square(gram_style - target_gram))
    return loss
```

Рисунок 35 – Функція втрат стилю

Завершальним етапом є обчислення загального значення втрат згідно формули (5.3), та запуск алгоритму формування стилізованого зображення (рис. 36). Спочатку отримуємо загальне значення втрат для генерованого зображення. Після цього, рахуємо градієнт для параметрів, що будуть коригуватися, мінімізуючи загальні втрати. Оскільки такими параметрами є пікселі зображення, то будемо застосовувати обчислений градієнт саме для їх зміни. Даний процес потрібно повторити близько 100 разів. Результатом буде зображення, у якого найкращий показник якості.

```
for i in range(iterations):
    with tf.GradientTape() as tape:
        all_loss = common_loss(**configuration)

    loss, style_loss, picture_loss = all_loss
    grads = tape.gradient(loss, primary_image)
    optimizer.apply_gradients([(grads, primary_image)])
    clipped = tf.clip_by_value(primary_image, clip_min, clip_max)
    primary_image.assign(clipped)

    if loss < less_loss:
        less_loss = loss
        best_image = bgr2rgb(primary_image.numpy())
```

Рисунок 36 – Формування стилізації

Приклади стилізації зображено на рис. 37.

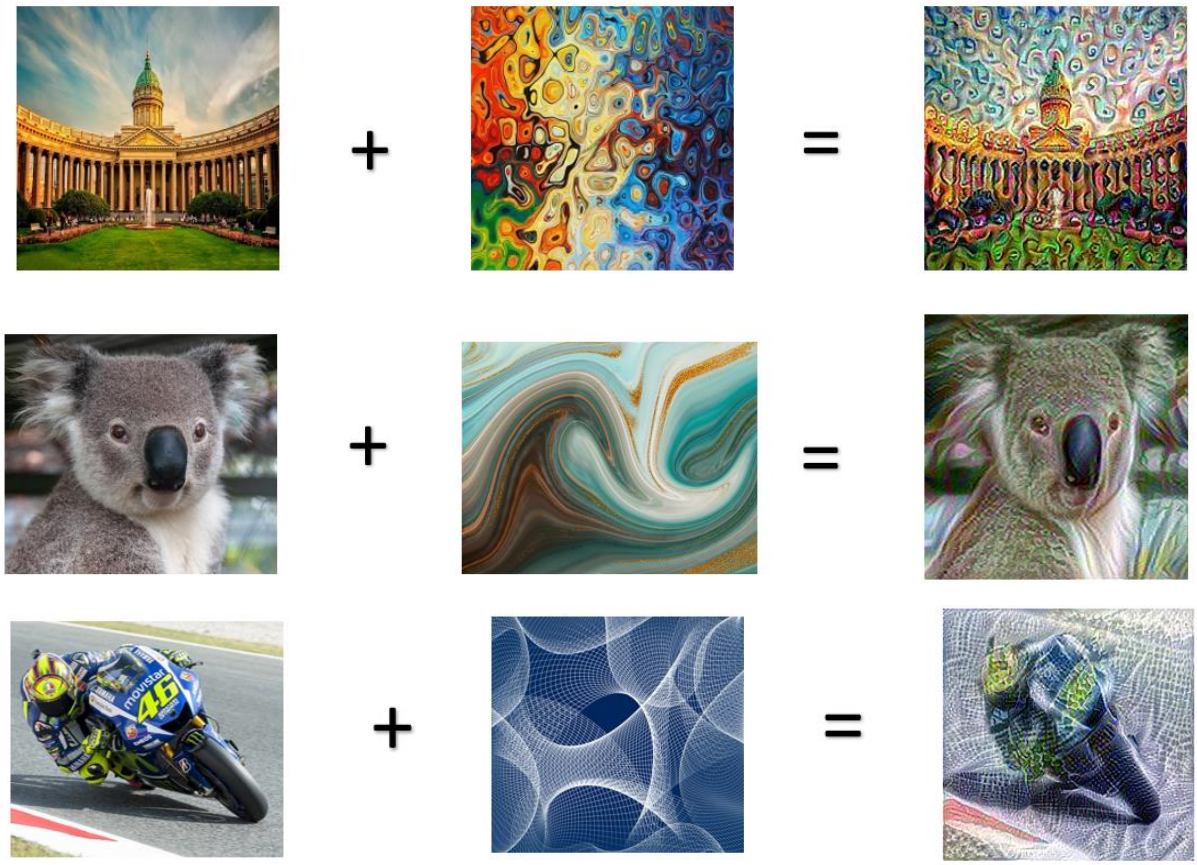


Рисунок 37 – Результати перенесення стилю на зображення

РОЗДІЛ 6 НЕЙРОЕВОЛЮЦІЯ

Нейроеволюція – це метод машинного навчання, який використовує еволюційні алгоритми для створення штучних нейронних мереж. Принцип дії запозичений у еволюції природних біологічних нервових систем. Порівняно з іншими методами навчання нейронних мереж, нейроеволюція має досить загальний характер, вона дозволяє навчатися без участі явно заданих цілей. Нейроеволюція – це ефективний підхід для вирішення проблем навчання з підкріпленням, який зазвичай використовується в еволюційній робототехніці та штучному інтелекті.

6.1 Алгоритм NEAT

NEAT – це еволюційний алгоритм, що створює штучні нейронні мережі. NEAT розроблено спеціально для вирішення трьох проблем [1], а саме:

– чи існує генетичне представлення, яке дозволяє різним топологіям виконувати процес кросинговеру (відтворення нових геномів, шляхом об’єднання двох інших). Рішенням є використання певних позначок, які будуть однаковими для генів з однаковим походженням;
– як захистити окремий нововведений геном, якому потрібно декілька поколінь для оптимізації, від передчасного зникання. Рішення – розділити кожне нововведення на різні види;
– як можна мінімізувати топологію протягом еволюції без необхідності використання функції пристосованості. Рішення – почати з мінімально можливої структури та ускладнювати її лише тоді, коли це необхідно.

Еволюційний алгоритм використовує геном, у якості формування фенотипу штучної нейронної мережі. Кожен геном містить перелік генів з’єднання, кожен з яких відповідає двом генам вузлам (рис. 38) [1]. Кожен ген

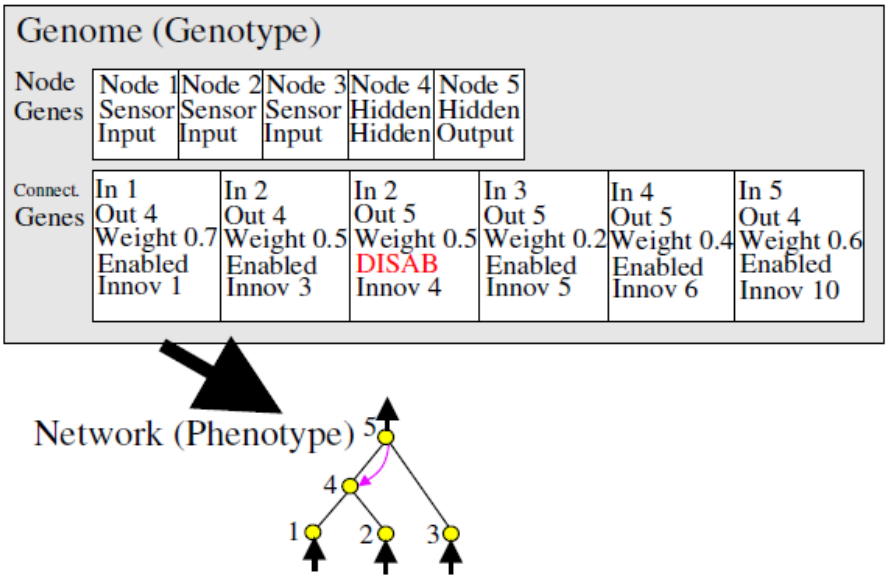


Рисунок 38 – Компоненти генома та приклад побудованого фенотипу

з’єднання визначає вхідний і вихідний вузол, вагу з’єднання, а також номер, під яким було введено цей тип геному, який дозволяє знайти відповідні гени під час кросинговеру.

Процес мутації в NEAT може змінити як вагу з’єднань, так і саму структуру мережі. Вага з’єднань змінюється, як і в будь-якій системі нейроеволюції, при цьому кожне з’єднання або збуджене, або ні. Структурні мутації, які розширюють геном, відбуваються двома способами (рис. 39). Під час першого виду мутації відбувається додавання з’єднання, що у свою чергу пов’язує між собою два вузли, які раніше не були з’єднанні. Під час другого виду мутації, існуюче з’єднання розділяється, а новий вузол розміщується там, де раніше знаходилось початкове з’єднання. Таким чином до геному додається два нових з’єднання, та видаляється одне старе.

Цей метод додавання вузлів призначений для того, щоб швидко інтегрувати нові вузли в мережу. За допомогою мутації створюються геноми різного розміру, іноді з абсолютно різними зв’язками [1].

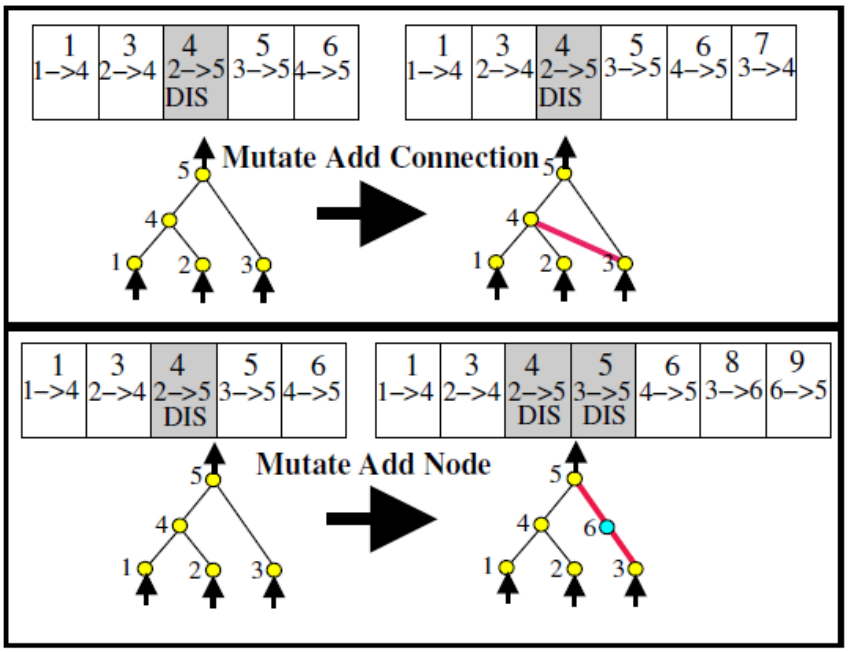


Рисунок 39 – Два види мутації геному

Окрім мутації, алгоритм NEAT застосовує метод кросинговеру, що означає схрещування двох мереж різної структури. NEAT справляється з цим, відстежуючи походження вузлів за допомогою ідентифікаційного номера (нові, вищі номери генеруються для кожного додаткового вузла). Ті вузли, що є гомологічними (походять від загального пращура), зіставляються для кросинговеру.

Саме в такий спосіб відбувається зміна структури нейронних мереж, які використовуються під час нейроеволюційного навчання.

6.2 Проходження об’єктів через перешкоди

Для демонстрації процесу проходження об’єктів через перешкоди було реалізовано міні-гру (рис. 40). Спочатку було реалізовано загальну

фізику застосунку, а вже потім відбувалося накладання алгоритму нейроеволюції, який самостійно навчався проходити через перешкоди.

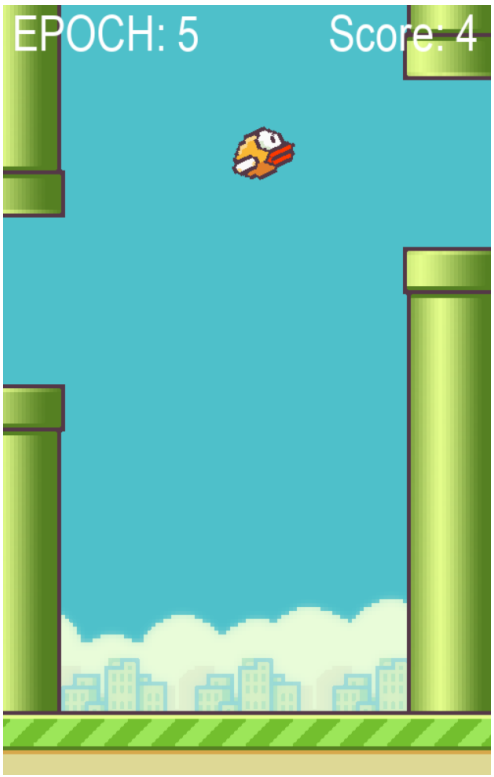


Рисунок 40 – Демонстрація застосунку

Процес за яким відбувається еволюційне навчання наступний: із самого початку створюється популяція видів певного розміру. Далі відбувається навчанням із підкріпленням. Кожен вид навчається шляхом отримання винагороди за виконання певної дії. Чим далі об’єкт зможе просунутися та чим більше перешкод буде подолано, тим буде вище показник отриманої винагороди, та тим краще буде пристосованість даного виду популяції. Якщо популяція помирає, то за нею створюється нова, у якій види мають характеристики попередньої популяції. Зазвичай обирається генотип найбільш пристосованих видів. Процедура продовжується до тих пір, поки не буде досягнуто ліміт на створення нових популяцій або певний вид досягне верхньої межі пристосованості.

Для тренування нейронної мережі, було обрано розмір популяції рівний 25 об’єктам, критерій пристосованості – максимум із значень (кращий той геном, у якого пристосованість вища), межа пристосованості – 100, функція активації – tanh (див. підрозділ 1.2.3), максимальне та мінімальне значення вагових коефіцієнтів – 30 та -30 відповідно, кількість вхідних даних – 2, кількість вихідних – 1. Взагалі повний перелік параметрів налаштування доволі великий, і це одна із переваг модуля NEAT-Python [20], що надає реалізацію алгоритму NEAT. Цей модуль надає спеціальний конфігураційний файл [20], який можна налаштовувати під власні потреби (додаток А).

Таким чином, залишилося зрозуміти, які саме дані надходять до мережі, та яким чином ці дані обробляються. Нейронна мережа для даного застосунку має три вхідних і один вихідний вузли. На вхід надходять висота положення об’єкта у просторі та відстань до нижньої і верхньої найближчих точок перешкод (рис. 41). Виходом мережі є значення на діапазоні від 0 до 1. Якщо значення більше за 0,5 тоді об’єкт повинен виконати переміщення вгору, інакше він нічого не робить. Таким чином, задача нейронної мережі полягає в умінні визначати коли потрібно здійснити переміщення вгору.

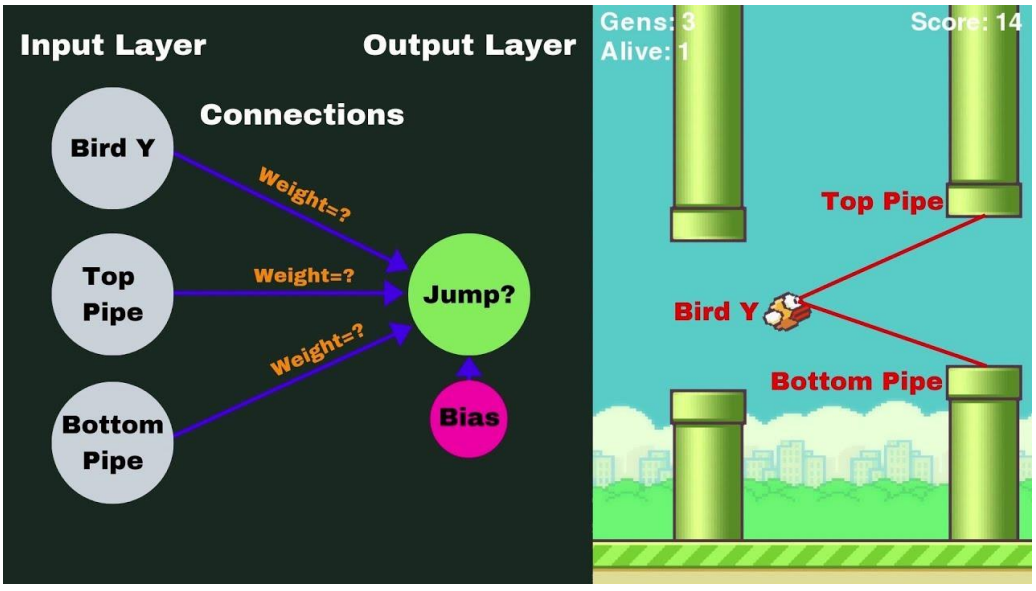


Рисунок 41 – Вхідні дані, на яких виконується навчання мережі

Тепер у процесі навчання нейронної мережі кожного із об’єктів, потрібно обрати ті, які виконують поставлене завдання найкращим чином. Для цього визначається функція пристосування. Чим далі об’єкт просунеться тим вище буде у нього нагорода, і тим краще він буде пристосований. Саме така логіка дозволяю алгоритму NEAT обрати найкращі геноми для кожної епохи, а потім використати їх для кросинговеру у наступних популяціях.

Для порівняння результатів навчання наведено таблицю 1, яка демонструє середню кількість епох, яка необхідна для проходження 20 перешкод (якщо об’єкт подолає дану кількість перешкод, то на цьому процес навчання можна зупинити).

Таблиця 1 – Порівняння навчання для різних початкових параметрів

Відстань між верхньою та нижньою частинами перешкод у пікселях	Швидкість переміщення перешкод (зміщення у пікселях одного такту виконання)	Розмір популяції для кожної епохи	Середня кількість епох навчання
200	8	25	3
200	10	25	5
150	8	25	8
150	8	15	12
150	10	15	16

6.3 Рух об’єктів за заданими траєкторіями

Для демонстрації процесу руху об’єкта за заданою траєкторією, було реалізовано застосунок, в якому виконується переміщення моделі автомобіля. Модель рухається вздовж дороги, обмеженої білою розміткою.

Для створення даного застосунку, спочатку було розроблено три різні види доріг (рис. 42) та реалізовано основний фізичний процес переміщення об’єкта. Кінцевим етапом реалізації, було інтегрування NEAT алгоритму в застосунок, за допомогою модуля NEAT-Python [20].

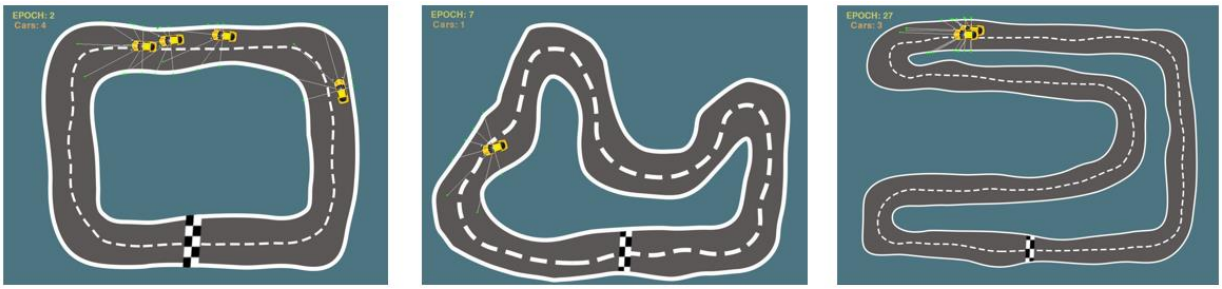


Рисунок 42 – Види траєкторій

Процес навчання подібний до попереднього застосунку – проходження об’єктів через перешкоди (див. підрозділ 6.2), тобто створюється популяція видів певної кількості та виконується навчання з підкріпленням. Чим далі об’єкт зможе просунутися тим вище буде показник отриманої винагороди. Вийшовши за межі траєкторії руху, об’єкт видаляється.

Розмір популяції було обрано рівним 10, критерій пристосованості – вибір максимуму, значення верхньої межі пристосованості – 10000, функція активації – tanh (див. 1.2.3), атрибут функції агрегації – значення суми, початкове з’єднання вузлів геному – повне (кожен вузол з’єднаний із іншим). Кількість вхідних даних – 8, кількість вихідних – 3 (додаток Б).

Модель автомобіля має 7 сенсорів (рис. 43), кожен з яких обчислює відстань від об’єкта, до межі, де закінчується дорога, ці данні подаються на вхід нейронній мережі. Останнім значенням, яке надходить до нейронної мережі, є параметр швидкості руху об’єкта вздовж траєкторії. На виході повертається три значення. Перше значення відповідає за зміну напрямку руху об’єкта проти годинникової стрілки, друге – за годинниковою стрілкою, а третій – за змінну швидкості руху. Кожне вихідне значення знаходиться на діапазоні від 0 до 1.

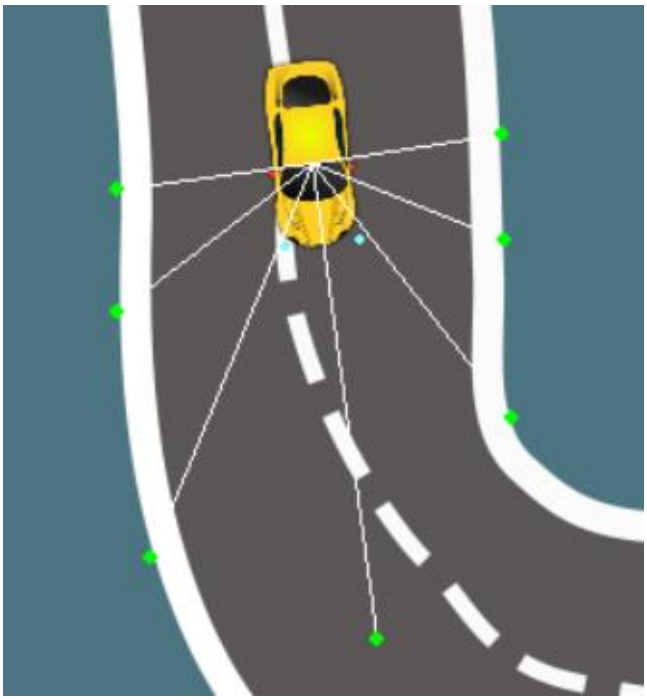


Рисунок 43 – Сенсори моделі автомобіля, які відслідковують відстань до меж траєкторії

Таким чином, нейрона мережа кожного геному об’єкта навчається визначати оптимальний вектор напрямку та швидкості. Початкова швидкість кожної моделі автомобіля дорівнює зміщенню на 3 пікселі за один такт відображення кадру. Мінімальне та максимальне обмеження значень зміщення дорівнюють 2 та 4 пікселі, відповідно, за один такт відображення кадру. В процесі навчання, нейронна мережа виділяє певні

ознаки траєкторії та виконує зміну швидкості руху. Такий підхід дозволяє знизити або збільшити швидкість руху, в залежності від наявності або відсутності різкого повороту, що у свою чергу додає гнучкості системі, при проходженні траєкторії.

Якщо модель автомобіля проходить дистанцію повністю, тоді можна зупинити процес навчання.

Для порівняння результату навчання та кількості епох, необхідної для проходження повної дистанції, було створено три різні траєкторії руху (рис. 42). Таким чином маємо три рівні складності.

Перша траєкторія найлегша, вона має достатньо широку межу дороги та не має досить різких поворотів, відносно інших траєкторій. Для того, щоб виконати повний оберт для першої траєкторії, алгоритму достатньо, в середньому від 2 до 5 епох навчання, при розмірі популяції рівній 10 об’єктам.

Друга траєкторія має досить різкі повороти, порівняно із першою, що в свою чергу є ускладненням при тренуванні нейронної мережі. Ширина дороги залишається, приблизно того ж розміру, що й у першому випадку. Таким чином для проходження цієї траєкторії, достатньо в середньому від 5 до 12 епох навчання. Розмір популяції залишається без змін.

Третя траєкторія руху є найскладнішою. Деякі повороти мають кут у 180°. Ширина дороги менша ніж у попередніх двох. Для того, щоб об’єкт пройшов повну дистанцію знадобилося більше 25 епох навчання.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було:

- продемонстровано загальний огляд роботи штучних нейронних мереж;
- описано процес оновлення та пошуку оптимальних значень вагових коефіцієнтів методом зворотного поширення (Backpropagation);
- використано засоби розробки, такі як: модуль NEAT-Python та бібліотеки TensorFlow і Keras, які було застосовано для побудови та навчання нейронних мереж;
- проаналізовано різні моделі штучних нейронних мереж та їх практичне призначення;
- описано процес перенавчання глибоких нейронних мереж та способи його усунення;
- досліджено загальний принцип роботи існуючих натренованих моделей згорткових нейронних мереж VGG-16 та VGG-19;
- реалізовано програму розпізнавання об’єктів на зображенні за допомогою моделі VGG-16 та бази даних зображень ImageNet. Описано процес реалізації, а також наведено приклади виконання розпізнавання об’єктів.
- реалізовано кольоризацію монохромного зображення, шляхом побудови моделі штучної нейронної мережі. Описано алгоритм та реалізацію перетворення зображення. Наведено приклади результатів;
- реалізовано процес стилізації зображення об’єктів за допомогою моделі VGG-19. Детально описано алгоритм обчислення критерія якості, який є основним у реалізації даного

застосунку. Продемонстровано результати виконання стилізаціїї штучної нейронної мережі;

– описано генетичний алгоритм навчання нейронних мереж NEAT. Використано модуль NEAT-Python, що надає доступ до реалізації даного алгоритму;
– розроблено застосунок, який на базі нейроеволюційного алгоритму, демонструє процес навчання проходження об’єктів через перешкоди. Описано принцип роботи нейронної мережі для даної задачі, виконано огляд параметрів конфігурації. Наведено порівняльну таблицю, що демонструє необхідну кількість епох навчання при різних початкових умовах для подолання 20 перешкод.
– розроблено застосунок, що демонструє процес навчання переміщення об’єктів за заданими траєкторіями. Описано алгоритм навчання нейронної мережі. Створено три види траєкторій різної складності, для порівняння процесу навчання штучної нейронної мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kenneth O. S. Eficient Evolution of Neural Network Topologies / Kenneth O. Stanley – Austin. – 6 с.
2. Guide to Artificial Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2021/05/beginners-guide-to-artificial-neural-network/.
3. Kenneth O. S. First neural network. Understand and create a Perceptron [Електронний ресурс] / O. S. Kenneth, M. Risto – Режим доступу до ресурсу: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf.
4. Kenneth O. S. Activation Functions in Neural Networks [Електронний ресурс] / O. S. Kenneth, M. Risto – Режим доступу до ресурсу: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.
5. Keras API reference / Optimizers / Adam [Електронний ресурс] – Режим доступу до ресурсу: https://keras.io/api/optimizers/adam/.
6. Understand the Impact of Learning Rate on Neural Network Performance [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.
7. Machine Learning Overtraining [Електронний ресурс] – Режим доступу до ресурсу: https://vortarus.com/machine-learning-overtraining/.
8. Dropout: A Simple Way to Prevent Neural Networks from Overfitting / Nitish Srivastava – Toronto, Ontario, M5S 3G4, Canada., 2014. – 30 с.
9. Sergey Ioffe. Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift / Sergey Ioffe., 2015.

10. Introduction to Batch Normalization [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/.
11. A Gentle Introduction to Batch Normalization for Deep Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.
12. VGG16 – Convolutional Network for Classification and Detection [Електронний ресурс] – Режим доступу до ресурсу: https://neurohive.io/en/popular-networks/vgg16/.
13. Convolutional Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/convolutional-neural-networks.
14. CNN Explainer [Електронний ресурс] – Режим доступу до ресурсу: https://poloclub.github.io/cnn-explainer/.
15. Convolutional Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network.
16. Recurrent Neural Network Tutorial (RNN) [Електронний ресурс] – Режим доступу до ресурсу: https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network#rdl.
17. A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks [Електронний ресурс] – Режим доступу до ресурсу: https://builtin.com/data-science/recurrent-neural-networks-and-lstm.
18. Coloring Black & White Images Using Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://medium.datadriveninvestor.com/coloring-black-white-images-using-deep-learning-984e6f4ddf14.

19. Leon A. Gatys. Image Style Transfer Using Convolutional Neural Networks / Leon A. Gatys.. – 10 с.
20. NEAT-Python [Електронний ресурс] – Режим доступу до ресурсу: https://neat-python.readthedocs.io/en/latest/index.html.

ДОДАТКИ

ДОДАТОК А

Параметри конфігурації нейроеволюційного застосунку «Проходження об’єктів через перешкоди»

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 100
pop_size              = 25
reset_on_extinction   = False

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 15
species_elitism      = 2

[DefaultReproduction]
elitism            = 2
survival_threshold = 0.2

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.0
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
```

```
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = full

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 0
num_inputs              = 3
num_outputs             = 1

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1
```

ДОДАТОК Б

Параметри конфігурації нейроеволюційного застосунку «Рух об’єктів за заданими траєкторіями»

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 10000
pop_size              = 10
reset_on_extinction   = False

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.0
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = full

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 0
num_inputs              = 8
num_outputs             = 3

# node response options
```

```
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 2

[DefaultReproduction]
elitism            = 2
survival_threshold = 0.2
```