Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

# КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
к. т. н., старший викладач
Бучко О.А.
*(прізвище та ініціали)*

_____
*(підпис)*
“____” _____ 2021 р.

Виконав студент
Сахаров А.Ю.
*(прізвище та ініціали)*
“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

к.ф. – м.н., доцент О. П. Жежерун ____

*(підпис)*

„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту__Сахарову А.Ю.__ факультету _інформатики_ _____4-го____курсу

Розробити Комп’ютерну систему класифікації зображень з використанням нейронних мереж

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Анотація
Вступ
1 Огляд теми, існуючих рішень та даних для розробки нейронної мережі
2 Побудова нейронної мережі, створення моделі та її оцінка
3 Розробка алгоритму програми використання моделі
4 Аналіз результатів
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2020 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

*Календарний план виконання роботи*

| No п/п | Назва етапу дипломного проекту (роботи) | Термін виконання етапу | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 15.09.2020 | |
| 2 | Аналіз літератури, пошук джерел | 30.10.2020 | |
| 3 | Аналіз технологій та додатків для створення програми | 01.12.2020 | |
| 4 | Розробка алгоритму, архітектура системи | 01.02.2021 | |
| 5 | Створення програмного коду | 03.03.2021 | |
| 6 | Аналіз створеного продукту з керівником | 20.03.2021 | |
| 7 | Написання текстової частини до курсової роботи | 03.03.2021 | |
| 8 | Перегляд текстової частини керівником | 20.03.2021 | |
| 9 | Написання доповіді, слайдів, останні правки | 27.03.2021 | |
| 10 | Завантаження курсової роботи | 12.04.2021 | |
| 11 | Захист курсової роботи | 15.04.2021 | |

Студент _____

Керівник _____

“_____” _____

*Зміст*

*Анотація*

У даній роботі досліджуються можливості нейронних мереж для класифікації зображень та використання їх для власного застосунку. Метою було створення нейронної мережі, тренування та використання її для розпізнавання одягу на зображеннях, тобто відношення зображення до певної категорії.

Сам застосунок використовує вже натреновану нейронну мережу та надає можливість просто перевірити певне зображення та віднести його до певної категорії. На першому кроці даного дослідження була створена та натренована нейронна мережа.

В основній частині описані принципи роботи нейронних мереж та на які результати вона спроможна. Крім того, було розглянуто процес тренування мережі, набори даних, які були використані для цього. Також було приділено увагу впровадженню мережі у результуючий застосунок.

У розділі «аналіз результатів» описано як користуватися застосунком, які результати та з якою точністю він надає, а також яку саме вдалося досягти точність.

В роботі використано технології Google Colaboratory для тренування мережі, а також PyTorch, Fastai як технології тренування на наборі даних DeepFashion. Для подальшого використання моделі використано мову Python, за допомогою якої вдалося не лише створити алгоритм використання моделі, а і перенести її у Telegram бот. Це зроблено для того, щоб можна було протестувати роботу моделі, перевірити її на різних вхідних даних, переконатися, що вона натренована на високому рівні.

**Ключові слова:** розпізнавання, категорії, одяг, нейронні мережі, схожість, зображення, тренування, тестування, результати пошуку, машинне навчання, глибоке навчання.

*Вступ*

Коли мова йде про світ сучасних технологій, то очевидним є невпинний їх розвиток та неможливість навіть на секунду відвернутися від прогресу – за цю ж секунду його неможливо буде наздогнати. Особливо це стосується технологій, які почали розвиватися не так давно, серед яких й нейронні мережі. Насправді, технологія машинного навчання є чимось далеким та невідомим для великої кількості людей, через що, і на ринку праці, зокрема, фахівці цієї сфери високо цінуються. Але, насправді, з нейронними мережами люди стикаються у повсякденні мало не щодня. Використовуючи систему розблокування смартфону (розпізнання обличчя чи відбитку пальця), ввівши запит до пошукової системи, читаючи новини в інтернеті, просто користуючись своїм смартфоном для буденних задач – мало не в усіх технологіях почали все частіше зустрічатися нейронні мережі, як важлива складова їх роботи. Важливо зазначити, що люди іноді навіть не підозрюють про те, що саме машинне навчання використовується як технологія, що спрощує їх користування певним застосунком або ж системою, і це свідчить тільки про одне – наскільки неймовірно цінним є цей розділ комп'ютерних наук.

В ситуації, що зараз склалася в світі, в умовах пандемії та серйозних соціальних змін, що зазнало суспільство, не відставали й технології у своєму розвитку та намаганні підлаштуватися під людей у складному становищі. Звісно, перехід на дистанційний режим роботи/навчання – одна з основних заслуг прогресу, який уможливив існування в умовах ізоляції, але не менш важливим є розуміння того, що серйозна частина сучасного життя перейшла у сферу послуг, зокрема придбання певних товарів онлайн та їх доставка. Ситуація з ринком онлайн магазинів товарів, продуктів та одягу вже давно вийшла на дуже висококонкурентний рівень та дала зрозуміти, що назад дороги немає, і людство й справді мусить звикати саме до такого способу покупок у майбутньому. Але проблема, з якою стикаються ці технології, це правильна та зручна організація

своїх магазинів, яка дуже скоро стане визначальним фактором того, де будуть купувати та де це буде робити зручно.

В основі даної роботи лежить вирішення одної з основних проблем, що виникає у таких підприємств, а саме автоматизація роботи з товарами на рівні комп'ютерів, що могла б допомогти легко опрацьовувати великі об'єми даних та полегшити організацію товарів на онлайн просторі. Зокрема, проблема, з якою стикається велика кількість людей, з ким мені довелося поговорити в ході виконання цієї роботи – це неможливість знайти чогось необхідного саме їм, однією з причин чого є неправильна класифікація товару на полицях онлайн магазину. Зокрема, це стосується одягу, коли людина може не знати, як називається те, що їй потрібно, або ж магазин неправильно відніс товар до тієї чи іншої категорії. Саме тому, в рамках цієї роботи, я вирішив дослідити можливості нейронних мереж у класифікації товарів, а конкретніше одягу, з використанням його зображення. Якщо коротко: *зображення одягу на вході, вид/тип/клас одягу на виході*. Я вважаю, що така архітектура має обов'язково бути якісно побудована у будь-якому інтернет магазині одягу, для того щоб уникати помилок неправильної класифікації, а також пришвидшувати пошук товару для користувачів.

Робота складається з чотирьох розділів.

Перший розділ присвячено аналізу нейронних мереж, їх видам, вибору використаного виду саме в цій роботі, а також пояснюється вибір набору даних та чому модель, що розроблена на ньому, буде працювати для вирішення проблематики цієї роботи.

Другий розділ описує розробку нейронної мережі, оцінку результатів її роботи, її показників, ефективності, точності тощо, а також процес створення, проектування та тренування нейронної мережі, що була використана в цій роботі.

В третьому розділі розглядаються принцип роботи застосунку, алгоритм, з яким працює розроблена нейронна мережа, а також її використання у фінальній технології, яка буде представлена користувачам, щоб з нею можна було

попрацювати та оцінити будь-кому, досвідченому чи недосвідченому користувачу.

Четвертий розділ присвячено аналізу результатів програми та опису того, як можна буде в подальшому розвивати створений застосунок.

Створено модель, яка забезпечує можливість класифікувати зображення одягу та визначити, до якої категорії він відноситься.

Постановка задачі:

1. Провести аналіз різних видів нейронних мереж та обрати найбільш ефективний для завдання класифікації одягу.
2. Спроектувати нейронну мережу, натренувати її на наборах даних, виконати аналіз показників створеної моделі, покращити, якщо можливо.
3. Розробити застосунок з використанням моделі та надати можливість користувачам оцінити рівень точності класифікації мережею.
4. Провести аналіз результатів, оцінити роботу застосунку, протестувати на різних зображеннях.

*Розділ 1: Огляд теми та використаних даних для розробки нейронної мережі*

*1.1 Нейронна мережа та її види*

Для вирішення проблеми цієї роботи, яка полягає в розпізнаванні певного елементу на зображенні, вирішено було працювати з технологіями нейронних мереж. Причиною такого вибору стало те, що саме нейронні мережі та сфера машинного навчання спеціалізується на створенні алгоритмів для розпізнавання елементів на зображеннях і не тільки [1]. Нейронна мережа, як технологія, по-перше, є передовою в плані того, що її можна навчити робити те, що від неї потрібно, а по-друге через можливість підкорегувати її роботу в залежності від результатів та специфічних потреб. Крім того, коли мова йде про поставлену задачу у роботі, то необхідно обрати таку технологію, яка вміє *класифікувати* дані, а не тільки їх розпізнавати. Нейронні мережі є одним з ефективних методів класифікації вхідних даних, особливо коли мова йде про великі дані, які необхідно обробити під час тренування мережі [5]. Під час роботи з класифікацією даних нейронні мережі почали набувати популярності ще від початку їх активного використання у 1985 році, і задача класифікації є одна з двох домінуючих сфер, у яких використовують таку технологію.

Традиційно для будь-якого процесу розпізнавання зображень використовують згорткові нейронні мережі/convolutional neural networks (CNN), адже вони використовують згортки та sub-sampling layers для пришвидшення процесу розпізнавання конкретних елементів зображення [6]. Згорткові мережі побудовані зі зменшенням кількості вузлів у кожному наступному шарі, що і суттєво пришвидшує час роботи з зображеннями та використовує меншу кількість математичних обрахунків та параметрів. Простіше кажучи, для того, щоб провести детекцію (знайти певний елемент на зображенні), необхідно працювати у різних шарах нейронної мережі саме над частинами кожного зображення, що і відбувається у згортковій мережі та її sub-sampling layers.

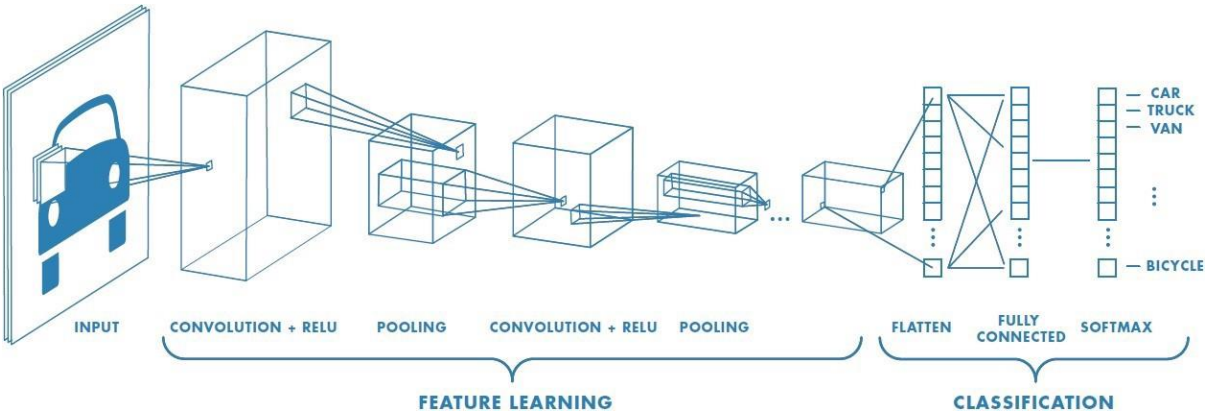



*Рисунок 1.1.1 – Ілюстрація роботи згорткової нейронної мережі*

З іншого боку, нейронні мережі розвиваються достатньо стрімко та все новіші підходи до розв'язання звичних задач з'являються кожного дня. Через «простоту» звичайних нейронних мереж в свій час вирішили розроблювати архітектуру згорткових, але згодом в них помітили певні недоліки, і розвиток призвів до виникнення нового виду – так званих, залишкових нейронних мереж/residual neural networks [8]. Їх особливістю є те, що вони роблять додаткові переходи між шарами мережі та дозволяють більш ефективно натреновувати мережі, мати більшу кількість шарів та використовувати виконані розрахунки до більшої кількості вузлів. Використовуючи дані не тільки минулого шару, а і минулих до нього, при розрахункові back-propagation не відбувається затухання градієнту, що дозволяє поглиблювати та подовжувати навчання моделі, що і робить залишкові мережі більш ефективними та вирішує проблему згорткових мереж [9].

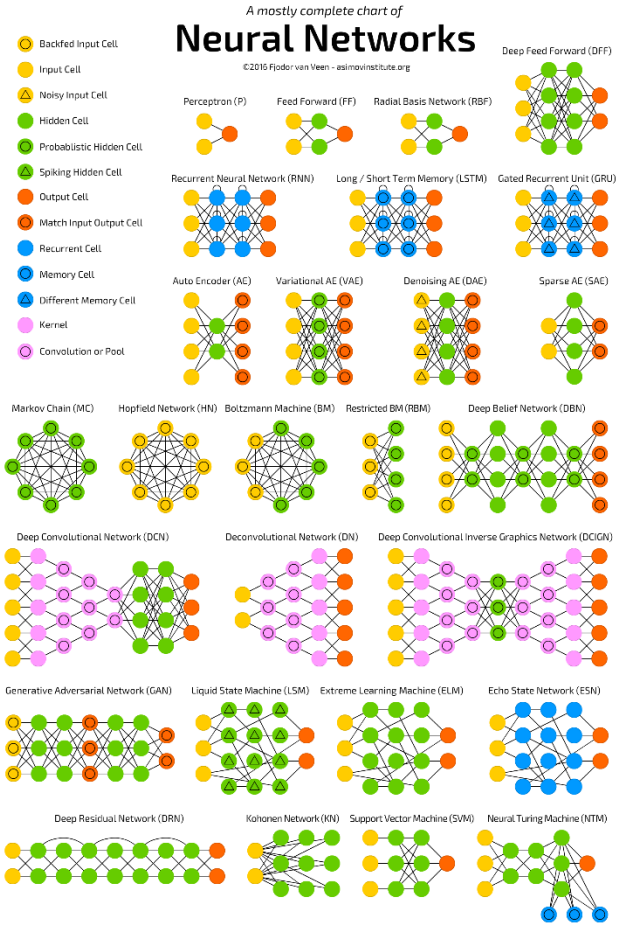



*Рисунок 1.1.2 – Види нейронних мереж*

*1.2 Набори даних*

При роботі з машинним навчанням важливою складовою для необхідного результату є якісно підібрані дані, на якому буде тренуватися розроблена нейронна мережа. Саме тому суттєвою задачею цієї роботи було підібрати правильний набор даних, який би містив достатню кількість даних, а також в якому було би правильно розподілено дані серед класів. Для задач класифікації не завжди буде правильним просто підібрати максимальну велику кількість даних для тренування, або ж далеко не будь-який набір даних підійшов би через нелогічний розподіл даних всередині нього між класами, що він містить. Справа в тому, що деякі набори даних, навіть будучи дуже великими та справляючи враження ефективних ресурсів для навчання нейронних мереж, насправді лише можуть спровокувати явище перенавчання/overfitting або ж передискретизації/ oversampling [10]. Перший датасет, який завжди трапляється при розв'язанні задачі класифікації, це датасет MNIST [11]. Він використовується для навчання простої моделі класифікації зображень цифр, для розпізнавання почерку. Так само існує кілька класичних датасетів для класифікації одягу, оскільки це достатньо популярна тематика для задач класифікації. Зокрема, існує Fashion MNIST, що складається з 70 тисяч зображень низького рівня якості, однакового розміру та у чорно-білому кольору для навчання моделі задачі класифікації одягу, що включає 10 класів [12]. Але класифікатор на 10 класів є недостатнім для вирішення проблеми поставленої у цій роботі, тому було вирішено використовувати інший датасет. DeepFashion – набір даних, що містить 800 тисяч кольорових зображень невисокої якості з реальних сайтів магазинів одягу, що ідеально підходить під проблему цієї роботи [4]. Крім того такі дані дозволяють розробити класифікатор, що працюватиме на 50 класів та категорій жіночого одягу, що є достатньо високим показником для класифікатора. Сам датасет показує здатність до тренування мереж високого рівня ефективності та точності, що для певних задач перевищує 90% [4].

*1.3 Пояснення принципу роботи нейронних мереж*

Для продовження пояснення використаних технологій та процесу проектування нейронної мережі необхідно пояснити принцип її роботи та який результат врешті-решт буде отримано після її тренування. Нейронна мережа – багаторівнева технологія, що містить декілька шарів, що містять по декілька вузлів для роботи з різними математичними функціями. Мережа працює з матрицями та має матрицю числових значень (модель «вагів»), яку використовує в математичних розрахунках в декілька ітерацій, що покращують показники моделі щоразу. Коли мережа отримує на вхід зображення з датасету, то розглядає його як матрицю значень у діапазоні [0;255], що у випадку даної роботи не тільки розкладені по пікселям, а і по трьом кольорам RGB. Надалі мережа використовує матрицю зображення та модель у математичних операціях на кожному шарі мережі. В залежності від архітектури мережі, на різних шарах використовуються так звані функції активації, що вираховують за допомогою двох матриць значення цих функцій (forward propagation), а потім корегують значення вагів у моделі (backward propagation). В залежності від функції активації мережа нормалізує матрицю зображення до діапазону [0;1] або [-1;1] перед подачею до моделі. З кожним новим зображенням ці операції повторюються, що врешті-решт призводить до створення моделі, яка містить оптимальні ваги, для того, щоб видавати внаслідок останнього математичного розрахунку на вихід нейронної мережі ймовірність того, що таке зображення містить той чи інший клас одягу. Фактично, мережа «підганяє» ваги у моделі для отримання найвищого показника точності при тренуванні на конкретному датасеті для конкретної задачі. Для класифікації обирається клас одягу, що отримав найвище число ймовірності внаслідок роботи алгоритму з моделлю та зображенням, що тестують користувачі. [1] [5] [8] [9]

Важливою частиною роботи нейронної мережі, яка суттєво змінює швидкість навчання та покращує результати, є використання функцій активації [18]. Їх

основна задача – математично перетворити вхідні дані до одного шару мережі у такі, що можна буде далі подати до наступного. Схема розрахунку:

$$output_k = activation(\sum W_{k,i} * input_i)$$

*Кожен вузол k певного шару отримує число, що є результатом використання функції активації до суми добутків вузлів і минулого шару на відповідні числа у матриці вагів.*

Використовується ціла низка функцій, в різних комбінаціях, в залежності від моделі та задачі, що ставиться перед розробниками. Серед них є 6 найпопулярніших:

1. Лінійна $f(x) = ax$ – при backward propagation буде завжди використовуватися одне і те саме значення похідної (в данному випадку константа).
2. Двійкова ступінчата $f(x) = 1, x \geq 0 \quad or \quad f(x) = 0, x < 0$ – не містить компоненту $x$, через що функція містить нульовий градієнт.
3. Сигмоїд $\frac{1}{e^{-x}}$ – дуже популярна функція серед нелінійних, має значення [0;1], тому і використовується зазвичай в останньому шарі мережі, щоб видати число, яке може виражати ймовірність передбачення моделі.
4. Тангенс tanh(x) – симетрична відносно нуля, має значення [-1;1], через що краще «врівноважує» числа на виході з шару мережі.
5. ReLu $f(x) = x, x \geq 0 \quad or \quad f(x) = 0, x < 0$ – одна з найпопулярніших нелінійних функцій, адже вона активує не усі нейрони одночасно, лише певну їх кількість.
6. Leaky ReLu $f(x) = x, x \geq 0 \quad or \quad f(x) = 0.01x, x < 0$ – через зміну у ситуаціях з від'ємними значеннями, така функція вирішує проблему нульового градієнту, що іноді виникає при використанні ReLu.

Існують також інші функції активації, які у різних комбінаціях надають різну ефективність навчання. Корисним є експериментальний підбір порядку та функцій для знаходження найкращої комбінації навчання між шарами мережі.

### *1.4 Огляд існуючих технологій, що реалізують задачу*

Існує декілька технологій, що реалізують задачу розпізнавання зображень одягу та видають результати високої точності. Але, мало не усі технології, до яких є доступ, займаються не розпізнаванням класу одягу, а пошуком ідентичного предмету в мережі, повертаючи ресурси на яких можна придбати саме те, що було на зображенні. Якщо говорити про задачу, поставлену саме в цій роботі, то вона стосується лише розробки частини технології, що потенційно може використовуватися у складніших застосунках, які б знаходили конкретну одиницю одягу та посилання на її купівлю. З іншого боку, задача пошуку конкретної одиниці одягу може бути не питанням класифікації, а питанням розпізнаванням конкретного зображення по великій індексованій колекції товарів. Зокрема, такі платформи як Google чи Amazon надають можливість скористатися технологіями Google Lens та Amazon StyleSnap, що мають доступ до своїх власних баз даних товарів (або ж технології Google Product), що і ставить під сумнів релевантність розгляду таких технологій у роботі.

Технологія, що привертає увагу, це до прикладу Asos Style Match. Британський сайт для придбання одягу онлайн дозволяє завантажувати зображення до ресурсу, що повертає *схожі* одиниці з наявних у себе в базі. Крім того, що вони мають обмежену базу даних, а отже з меншою ймовірністю роблять акцент на пошуку ідентичної одиниці одягу, вони повертають саме рекомендації того, що може сподобатись, внаслідок аналізу зображення. Самі вони на сторінці пояснення роботи технології пишуть, що вони аналізують зображення на предмет «colour, pattern and type of clothing so that we can make recommendations to you», що свідчить про реалізований класифікатор типів одягу, яка є частиною більшої системи.

*Розділ 2: Побудова нейронної мережі, створення моделі та її оцінка*

*2.1 Проектування нейронної мережі у Google Colaboratory*

Для проектування нейронної мережі та її тренуванні на великій кількості даних, такій як у датасеті DeepFashion, необхідно використовувати ресурс зі швидким доступом до зчитування файлів та можливістю використання великого сховища. Одним з таких ресурсів є Google Colaboratory. Створення середовища там дозволяє працювати з графічним відображенням файлів, швидко до них доступатися, а також використовувати сховище розміром до 150Гб. Завантаживши туди датасет, необхідно створити архітектуру доступу до даних, які використовуються для тренування. Крім самих зображень, існують також csv-файли, що зберігають шлях до файлу, та певну інформацію про нього. Наприклад, використаний файл train.csv містив у собі таблицю з іменами зображень та класу одягу, до якого він відноситься. Це необхідно для тренування мережі, адже вона використовує вже labeled data для свого навчання. Для процесу тренування моделі було використано бібліотеки Fastai [2] та PyTorch [3]. Fastai необхідний для роботи з даними, зокрема для зчитування файлів csv, а також для використання методів навчання мережі, зокрема fastai.vision.learner.cnn_learner, fastai.vision.learner.fine_tune та fastai.vision.learner.fit_one_cycle. Усі ці методи дозволяють будувати нейронну мережу, особливо не заглиблюючись в її архітектуру. Фактично, використовуючи ці три методи можна навчити робити мережу те, що необхідно. Крім того, важливим елементом навчання є вибір типу мережі, і як було зазначено вище, було обрано залишкову мережу. Для уникнення ризику перенавчання [10] було обрано ResNet34 [14] з бібліотеки torchvision.models, особливістю якої є 34 шари та попереднє тренування матриці на датасеті ImageNet – даних, що використовуються для створення моделі, що розпізнає образи на зображеннях, без конкретної класифікації.

Під час розробки нейронної мережі є декілька речей, на які необхідно звернути увагу для її покращення. По-іншому, це називають гіперпараметрами нейронної

мережі, від зміни яких залежить рівень оптимізації мережі [16]. На деякі з них звертати увагу необхідно через те, що вони дозволяють біль ефективно тренувати мережу, на меншій кількості ітерацій, з більшим прогресом між ними. Зокрема, таким параметром є коефіцієнт швидкості навчання / learning rate. Нейронна мережа дозволяє обирати коефіцієнт та передавати його як параметр для навчання мережі. Але попередньо необхідно зрозуміти який коефіцієнт необхідно взяти. За допомогою методу learner.lr_find() будуємо графік залежності втрат при навчанні від коефіцієнту швидкості навчання мережі.

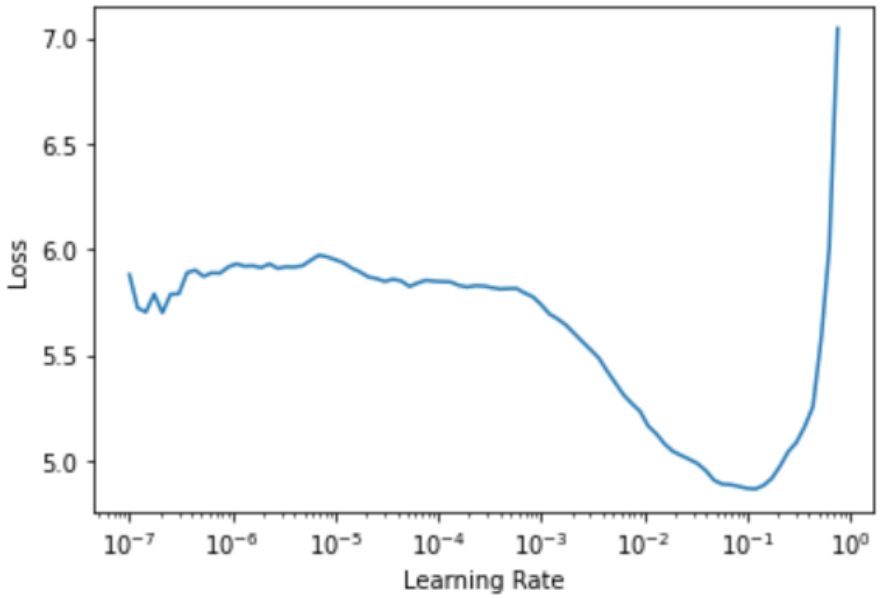



*Рисунок 2.1.1 – Графік залежності втрат при навчанні від коефіцієнту швидкості навчання*

З погляду теорії, найкращим коефіцієнтом для навчання мережі на початкових етапах є такий коефіцієнт на графіку, при якому кут спуску графіка буде якомога крутішим [16]. З графіка було зроблено висновок, що приблизним значенням такого коефіцієнту буде число в межах від $10^{-3}$ та $10^{-2}$. Або ж можна знайти це число за допомогою методу learner.lr_steep(). Вивід цього методу дав результат: 9.12e-03, що фактично є числом 0.009, і що справді лежить в межах від $10^{-3}$ та $10^{-2}$, ближче до останнього. Таким чином на перших ітераціях використовувався метод learn.fine_tune(2, base_lr=9.12e-03), де перший параметр означає кількість ітерацій навчання, а другий – власне знайдений оптимальний коефіцієнт швидкості навчання мережі. Сам метод fine_tune використовується у ситуаціях,

коли необхідно адаптувати модель до нових умов. Фактично, модель тренується всередині мережі кілька разів, щоб з кожним разом показувати все кращі результати. Як зазначалося вище, модель ResNet, що використана як база для тренування нейронної мережі, є натренованою на базових образах, тому використання методу fine_tune з першої ітерації є *адаптація натренованої моделі під конкретну задачу*.

Але, використання одного й того самого коефіцієнту швидкості навчання мережі не є правильною тактикою для більшої кількості ітерацій, адже це може призвести до ефекту перенавчання [10], тому з цим коефіцєнтом буде проведено лише дві ітерації. Це так само і підтверджує видозмінений графік залежності втрат від коефіцієнту швидкості навчання, який побудовано вже після проходження перших ітерацій:

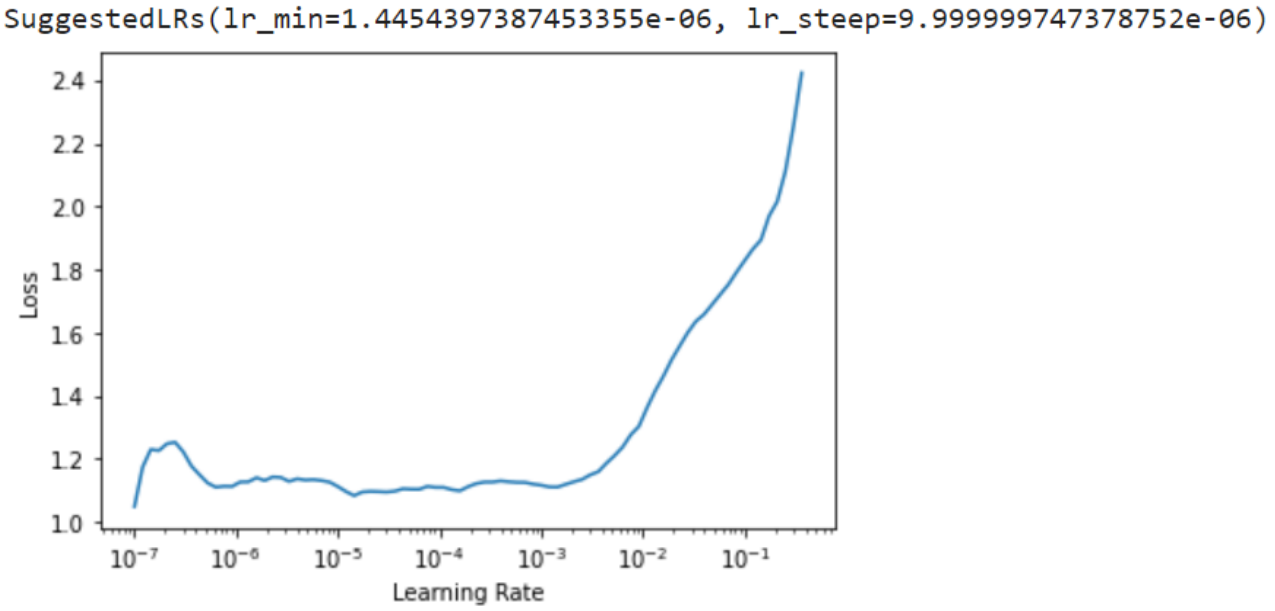



*Рисунок 2.1.2 – Новий графік залежності втрат при навчанні від коефіцієнту швидкості навчання*

Система сама пропонує коефіцієнти, які варто використати при продовженні навчання. Вона посилається на дані залежності, де, як і в минулому випадку, найкраще використовувати значення коефіцієнту, за якого втрати зменшуються, або ж майже не змінюються. При цьому, для продовження тестування буде використано метод learn.fit_one_cycle(5, lr_max=slice(1e-7, 5e-3)), який надає можливість не тільки задавати коефіцієнт швидкості навчання, а і задавати діапазон в якому він буде змінюватися від перших ітерацій до останніх. Це

необхідно для того, щоб коли навчання продовжувалося, то коефіцієнт підвищувався, адже навчання на дуже малому коефіцієнті, за якого втрати знижуються (в даному випадку в межах від $10^{-7}$ до $10^{-6}$) призведе до малоефективного навчання [16], якщо робити це на усіх ітераціях. Саме тому, у метод fit_one_cycle() є можливість передати діапазон коефіцієнтів швидкості навчання, за допомогою яких можна підвищити його ефективність та розмаїття. В цій ситуації, і за графіком, і за пропозицією системи, буде обрано діапазон від $10^{-7}$, що є числом 1e-7, до 0.005, місцем, де починає рости графік, серединою між $10^{-3}$ до $10^{-2}$, що є числом 5e-3. Перший параметр методу – знову кількість ітерацій. Причина обрання саме п'яти ітерацій – експериментальна, це детально описано в наступній частині, але в цілому – експеримент показав, що більша кількість ітерацій не є корисною для навчання.

### *2.2 Оцінка роботи мережі*

Початкові ітерації тренування показали серйозні покращення результату як у точності моделі, так і у зменшенні втрат.

| epoch | train_loss | valid_loss | accuracy | time |
|---|---|---|---|---|
| 0 | 1.551926 | 1.412060 | 0.589022 | 33:11 |
| **epoch** | **train_loss** | **valid_loss** | **accuracy** | **time** |
| 0 | 1.338351 | 1.221115 | 0.650659 | 37:45 |
| 1 | 1.161446 | 1.072970 | 0.691656 | 37:43 |

*Рисунок 2.2.1 – Перші дві ітерації тренування нейронної мережі*

Важливо підмітити, що не тільки підвищилась точність розпізнавання на 10%, а ще і зменшились втрати, майже у півтора рази. Але, подальші ітерації мережі не показали суттєвих змін, тому для уникнення ризику перенавчання, що може почати зменшити точність, навіть якщо втрати будуть знижуватися, після 5 ітерацій навчання було припинено.

| epoch | train_loss | valid_loss | accuracy | time |
|---|---|---|---|---|
| 0 | 1.102886 | 1.072060 | 0.691174 | 37:42 |
| 1 | 1.104428 | 1.063833 | 0.694908 | 37:41 |
| 2 | 1.105640 | 1.055722 | 0.696234 | 37:39 |
| 3 | 1.067472 | 1.048353 | 0.698121 | 37:40 |
| 4 | 1.064548 | 1.043491 | 0.700249 | 37:56 |

*Рисунок 2.2.2 – Після ще 5 ітерацій результати майже перестали змінюватись*

Саме 5 ітерацій виявилось достатньо для навчання моделі для високого рівня точності у виборі топ-1 ймовірності для результату. Пошук ідеальної кількості ітерацій – процес експериментальний. Приклади [10] [13] показують, що на етапах подальшого навчання за такої динаміки виникають проблеми перенавчання моделі, адже починають рости втрати навіть за невеликого збільшення відсотка точності, що не компенсує факт того, що втрати при навчанні – показник того, що мережа тренується неповністю.

| epoch | train_loss | valid_loss | accuracy | time |
|---|---|---|---|---|
| 0 | 1.156718 | 1.115304 | 0.682220 | 26:32 |
| 1 | 1.204249 | 1.118955 | 0.684027 | 26:38 |
| 2 | 1.141088 | 1.095323 | 0.685593 | 26:34 |
| 3 | 1.087024 | 1.091421 | 0.685633 | 26:32 |
| 4 | 1.073382 | 1.073441 | 0.692620 | 26:27 |
| 5 | 1.082264 | 1.070563 | 0.692258 | 26:27 |

*Рисунок 2.2.3– Приклад тренування на 6 ітераціях*

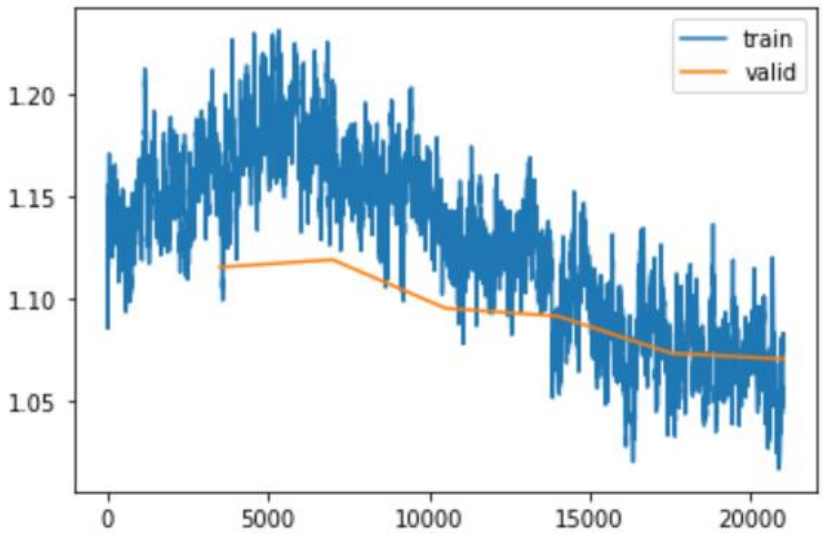



*Рисунок 2.2.4 – Графік втрат*

Як видно з таблиці, на етапі шостої ітерації і зменшилася точність моделі, і збільшилися втрати. Також збільшення втрат прослідковується на етапі другої ітерації, але надалі вони знову почали зменшуватися. Як видно з графіку втрат, подекуди показники навіть почали перевищувати втрати попередніх ітерацій, що свідчить про те, що навчання перестало приносити результати.

Також за допомогою бібліотеки matplotlib можна побудувати графік зміни точності моделі внаслідок навчання та побачити, що саме на шостій ітерації точність почала знижуватися.

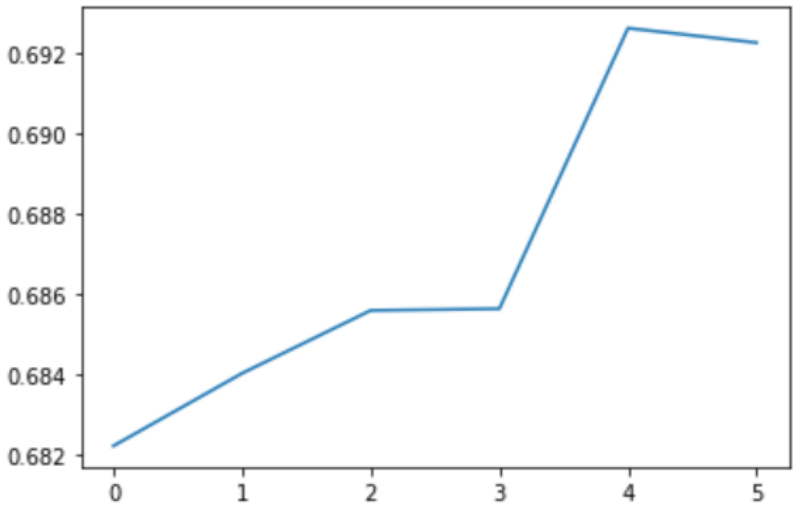



*Рисунок 2.2.5 – Графік зміни точності моделі*

Потенційні наслідки такого навчання – дилема зміщення та дисперсії [13]. Наприклад, якщо тренувати мережу велику кількість разів та проганяти через неї

один і той самий датасет зі сподіваннями, що вона ще чомусь навчиться, це може призвести до ситуації, що вона перестане сприймати будь-які інші вхідні дані, які зрештою і мають розпізнаватися натренованою моделлю в майбутньому. Враховуючи те, які однотипні зображення зазвичай використовуються у датасеті, це призведе до того, що система перестане бути чутливою, адже стане дуже заточена саме на дані з датасету, які можуть не відповідати тому, що буде передано користувачем в майбутньому на вхід до моделі. Модель, що так працює, називають моделлю з *високою дисперсією*.

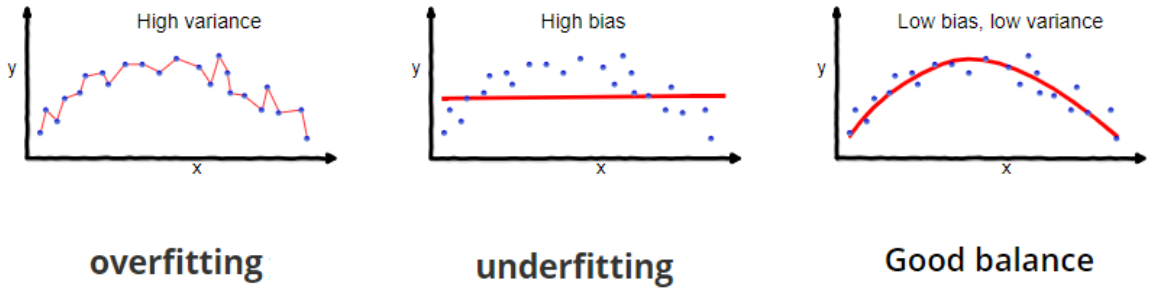



*Рисунок 2.2.6 – Ілюстрація до дилеми зміщення та дисперсії*

З іншого боку, недостатня кількість ітерацій, з неправильно визначеними коефіцієнтами навчання або іншими гіперпараметрами може спровокувати недонавчання та *високе зміщення.* В такій ситуації модель навпаки ще не стала достатньо чутливою для розпізнавання та класифікації майбутніх вхідних даних, через що і важливо початково правильно будувати архітектуру мережі, вибирати найкращі інструменти для її навчання, правильні датасети та експериментувати з різними параметрами для досягнення хорошого результату.

Крім того, якщо порівнювати досягнуту точність у 70% при знаходженні топ-1 результату з точністю, що була описана, як точність 82.6% знаходження топ-3 самої системи набору даних, можна зробити висновки, що модель справді натренувалася до достатньо високого рівня.

| | Category | |
|---|---|---|
| | top-3 | top-5 |
| WTBI [3] | 43.73 | 66.26 |
| DARN [10] | 59.48 | 79.58 |
| FashionNet+100 | 47.38 | 70.57 |
| FashionNet+500 | 57.44 | 77.39 |
| FashionNet+Joints [34] | 72.30 | 81.52 |
| FashionNet+Poselets [34] | 75.34 | 84.87 |
| FashionNet (Ours) | **82.58** | **90.17** |

*Рисунок 2.2.7 – Дані класифікатора системи датасету для топ-3 та топ-5 результатів*

Простим прикладом для пояснення розрахунку точності моделі є приклад моделі з двох класів, Positive та Negative. Для розрахунку модель будує так звану confusion matrix, яка містить в собі числа, що характеризують кілька комбінацій. Ці комбінації поділяють на 4 види: True Positive (TP), True Negative (TN), False Positive (FN), False Negative (FN) [17].

| | Positive | Negative |
|---|---|---|
| Positive | 3 | 1 |
| Negative | 2 | 4 |

В даному випадку:

True Positive – передбачення збіглося з класом, обидва Positive.

True Negative – передбачення збіглося з класом, обидва Negative.

False Positive – передбачено Positive, а насправді Negative.

False Negative – передбачено Negative, а насправді Positive.

Тоді точність моделі буде вираховуватися за формулою:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Аналогічна матриця будувалася під час оцінки натренованої моделі, де значення, що використані в чисельнику – значення на головній діагоналі, а у знаменнику – усі числа матриці. Фактично, точність моделі можна подати у вигляді відношення правильні передбачення / усі передбачення [17].

Для розрахунку втрат під час навчання та тестування нейронної мережі використовується різниця між реальним класом зображення та передбаченням, яке було зроблено моделлю. Розраховується коефіцієнт втрат за допомогою формули [1]:

$$Cost\ function = \frac{1}{m} \sum L(y', y)$$

*m – кількість зображень, що використано при ітерації, L – функція втрат для кожного конкретного зображення, де y' – результат передбачення, y – реальний клас.*

Для кожного зображення розраховується значення втрат за формулою:

$$Loss\ function = -y * \log y' - (1 - y) * \log (1 - y')$$

Таким чином мережа розраховує втрати, які на етапі навчання зробила модель, а також які вона зробила на етапі закінчення ітерації навчання. В ситуації ефективного навчання, втрати train_loss та validation_loss мають максимально наблизитись одне до одного. У випадку, якщо між ними велика різниця, це свідчить про проблему недонавчання або перенавчання.

За допомогою методу learner.show_results() можна також оцінити роботу натренованої моделі. Цей метод з бібліотеки Fastai видає декілька передбачень щодо зображень з набору даних, та порівнює їх з реальними класами цих файлів. Метод використовує будь-які зображення з набору даних, тим самим показуючи різні класи, роботу моделі для передбачення різних результатів, та виділяє ті передбачення, що не збіглися зі справжніми класами.

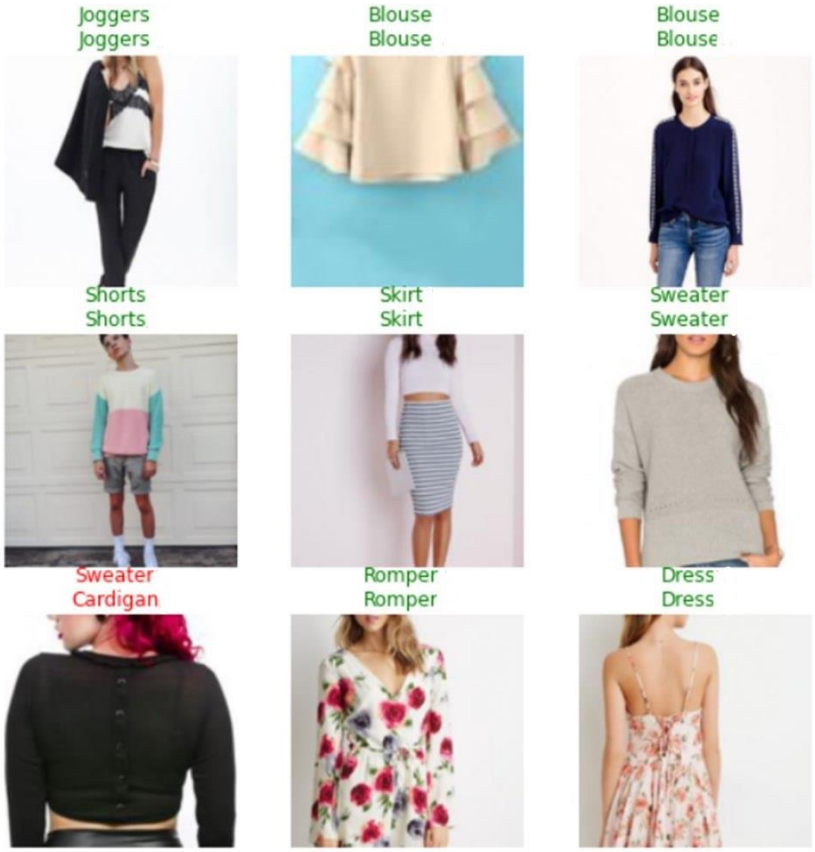



*Рисунок 2.2.8 – Приклад передбачень моделлю над зображеннями датасету*

Як показують результати, модель справді натренована на передбачення та розпізнавання достатньо високої точності. Далі, вже на алгоритмі використання моделі вдалося також переконатися, що вона справді працює. Єдине питання, що стоїть стосовно результатів роботи натренованої моделі – вищезгадане питання зміщення та дисперсії, яке є достатньо поширеним для теорії машинного навчання [13]. В даному дослідженні набір даних був натренований виключно на зображеннях, які були початково взяті з сайтів інтернет-магазинів одягу, які мало схожі на зображення з реального життя. Тож система справді буде показувати роботу достатньо точного класифікатора, коли мова бути йти про тестування таких зображень, а от якщо модель отримає звичайне зображення, від звичайної людини – такі ж показники гарантувати не можна.

## *Розділ 3: Розробка алгоритму програми використання моделі*

### *3.1 Алгоритм використання моделі, нормалізація*

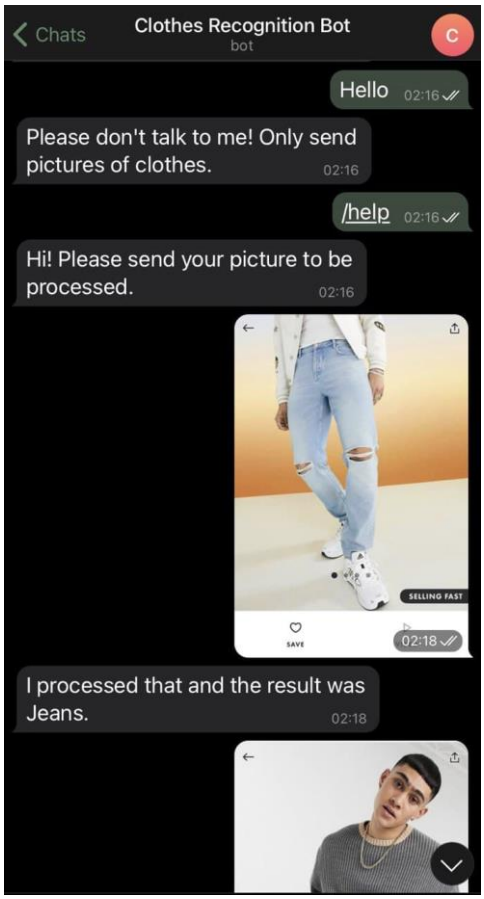
Використання розробленої моделі потребує певної «обгоротки», з допомогою якої можна було б дозволити її доступ до звичайних користувачів. Але насамперед необхідно створити алгоритм, який дозволяє передавати картинку на обробку та аналіз у модель, щоб отримувати необхідні ймовірності на вихід, та обирати серед них ту, яка і буде вихідними даними. Перш за все необхідно видозмінити зображення у таке, яке можна передати на обробку моделі, а також «нормалізувати» його для моделі ResNet [14]. Для цього відбувається нормалізація та зміна вимірів зображення за допомогою методу transforms.Compose([transforms.Resize(),transforms.ToTensor(),transforms.Normalize([0.485, 0.456, 0.406],0.229, 0.224, 0.225])]), де числові дані – стандартні для саме такої нейронної мережі. Сама модель зчитується з файлу у модель Sequential з бібліотеки PyTorch [3], яка дає можливість будувати модель шар за шаром, і відповідно є однією з найпростіших варіантів як для роботи з моделями, так і для зчитування даних зображення і їх використання при розрахунках ймовірності. Далі, відповідно, картинка передається через центральний процесор до моделі, яка за допомогою методу model(inp)виконує математичні розрахунки, що закладені в мережу, відповідно видаючи числа для різних класів – ймовірності того, що це зображення розпізнається за тим чи іншим класом. Нарешті, відсортувавши ймовірності за зростанням, з вихідних даних обирається найбільша ймовірність, що і вказує на клас одягу, який модель розпізнала на вхідному зображенні.

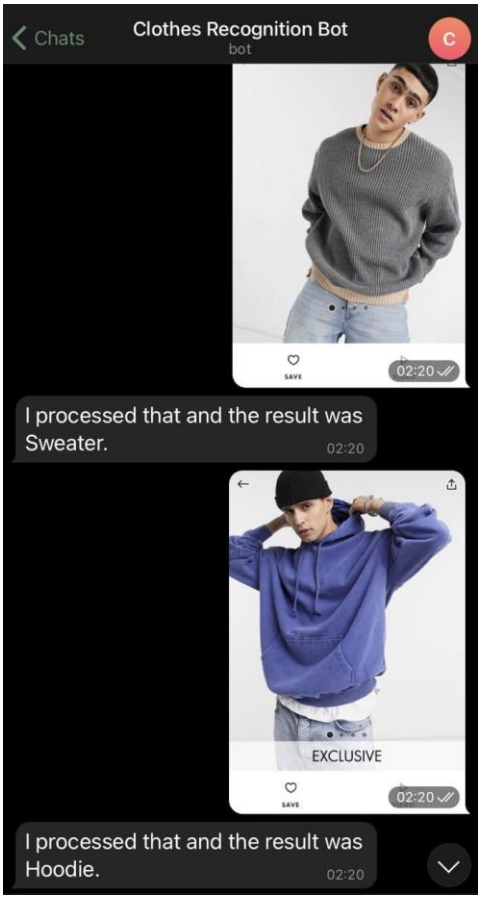
Алгоритм використовує декілька бібліотек Python, зокрема PyTorch [3], що містить вищеописані методи для роботи з моделлю, передачі у неї зображення, а також виведення результатів та найвищої ймовірності. Також використовується torchvision, як частина PyTorch, що дозволяє провести маніпуляції з нормалізацією зображення перед його введенням до моделі за допомогою блоку transforms. Додатково використано Python Imaging Library, для зчитування

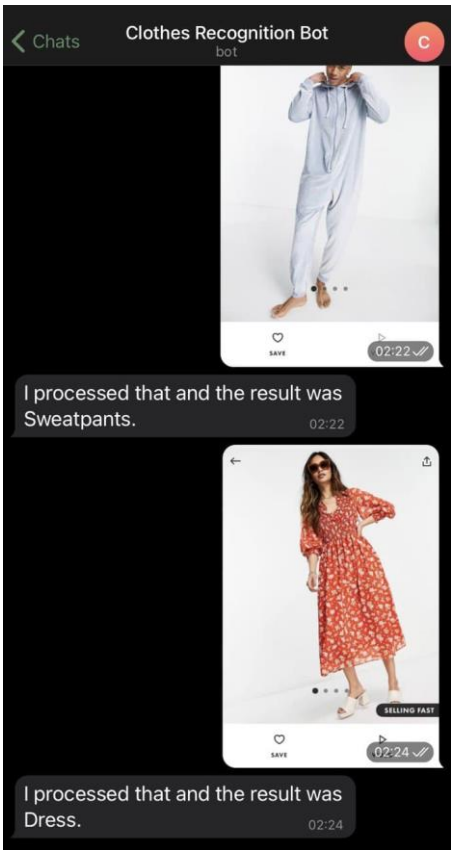
вхідного зображення та перетворення його на формат, з яким надалі зручно працювати (фактично – матриця), і Fastai [2], що бере участь у зчитуванні та використанні моделі ResNet.

### *3.2 Telegram бот*

Для доступу користувачів до інтерфейсу використання моделі та тестування її роботи та ефективності необхідно було обрати простий у користуванні ресурс, через який можна було би проводити операції з використанням алгоритму, написаного на мові Python. Оскільки, фактично, ніяких функцій, крім повернення класу одягу алгоритм не виконує, достатньо зручним ресурсом для такої задачі є бот у Telegram [15]. Імплементація такого боту є достатньо простою та не потребує зайвих операцій щодо проведення підключення алгоритму на Python до боту. За допомогою бібліотеки python-telegram-bot та модулів Updater, CommandHandler, MessageHandler, Filters бот може приймати повідомлення, реагувати лише на ті, на які у нього поставлені фільтри, та надсилати відповідь у вигляді виводу результату роботи розробленого алгоритму використання моделі. Передавши унікальний токен боту до коду програми, вона отримує доступ до повідомлень, що були надіслані користувачем до боту. На цьому етапі роботи, бот вживає заходів лише при появі текстових повідомлень, на які він відповідає інструкцією, що необхідно завантажувати картинку, а також, власне, на зображення, а назад повертає результат функції розпізнавання зображення у вигляді повідомлення, що система розпізнала на ньому [клас одягу]. Telegram бот ідеально підходить для такого завдання, адже має простий інтерфейс, та підтримує як раз той функціонал, що необхідний для зручного використання розробленої моделі.




*Рисунки 3.2.1, 3.2.2 та 3.2.3 – Зображення роботи інтерфейсу бота*

*Розділ 4: Аналіз результатів*

Кінцевим результатом, що презентується користувачам, є Telegram бот, в якому користувачі можуть спробувати завантажити зображення одягу, та отримати на вихід передбачення того, що саме з класів одягу зображено на ньому. Система є простою у користуванні та дає результат для користувача у бот достатньо швидко, звідки складається враження, що алгоритм роботи достатньо простий та швидкий. Сам алгоритм складається з методів бібліотеки PyTorch, Fastai та допоміжних, який використовує файл натренованої моделі як ресурс, до якого завантажує отримане від користувача через бот зображення, та повертає через бот результат роботи класифікатора моделі. Модель була натренована до високого показника точності у 70% за допомогою датасету DeepFashion, що є великим датасетом з 800 тисячами зображень та 50 класами одягу, які може розпізнавати отримана модель. Нейронна мережа, що була спроектована для тренування моделі, отримує модель ResNet34 як основу для тренування моделі, що початково містить матрицю, яка натренована на розпізнавання загальних образах на зображенні. Тестування отриманої системи показали, що вона справляється з поставленою задачею у цій роботі, та робить це з високим показником точності, допускаючи помилки, причини яких згадано у цій роботі. Всі розробки проведені на мові програмування Python, що є однією з найзручніших мов для роботи у сфері машинного навчання.

*Висновки*

При виконанні даної курсової роботи, можна з впевненістю сказати, що найскладнішим етапом розробки алгоритму було тренування моделі та проведення експериментів з різними гіперпараметрами для досягнення кращого результату. Бібліотеки PyTorch та Fastai є дуже зручними ресурсами для побудови нейронних мереж та тренування моделей. Вони дозволяють будувати їх швидко навіть недосвідченим розробникам у сфері машинного навчання, адже не потребують глибокого знання математичних формул, що лежать в основі роботи мережі, та архітектури і схеми роботи мережі. Також важливим етапом роботи було дослідження існуючих наборів даних для розв'язання поставленої задачі, але це виявилося простішим через те, що на етапі постановки задачі було заздалегідь визначено якого масштабу має бути натренована модель. Це не було задачею спробувати натренувати модель-класифікатор, це було задачею тренування моделі для конкретних потреб користувачів, для яких необхідно було обрати якісно складений набір даних з великою кількістю даних. Вдалося також створити інтерфейс для роботи з моделлю у вигляді Telegram бота, з яким раніше працювати не доводилося. Насправді, підключити модель до інтерфейсу виявилось нескладно, адже Telegram API надає можливість без особливих складнощів працювати з підключенням алгоритмів на різних мовах програмування до логіки своєї роботи.

Систему можна покращити по декількох показниках та критеріях. Перш за все, немає ідеалу у тренуванні моделі, можна продовжувати тренувати її з використанням інших технологій, інших моделей в основі, інших параметрів чи наборів даних. Хоча показник точності 70% є достатньо високим результатом тренування мережі, його можна спробувати підвищити, навіть на декілька відсотків. Крім того, кількість класів 50 не обмежує потенціал розвитку класифікатора для розпізнавання інших категорій одягу, які не було включено до набору даних. Тому використання більших даних, хоч і вимагає довшого

навчання та ще більшої кількості експериментів, але варте того, щоб створювати та покращувати систему.

З недоліків: через те, що система має обмежену кількість класів у наборі даних та натренована лише на них, іноді вона може розпізнавати на зображенні те, чого там немає. Зокрема, це спричинено тим, що деякі класи, яких немає у наборі даних, візуально схожі з тими, що є в ньому, а саме тому користувачам все ж можуть видаватися результати з неточностями. Наприклад, якщо уявити ситуацію, що користувач завантажує зображення піжамної майки, а система повертатиме у відповідь майка (Tank), то ці дві категорії хоч і мають дуже схожі візуальні характеристики (контур, форму, розмір), але по суті створюють неправильне враження про те, що є на зображенні насправді.

Загалом, система надає передбачення з точністю 70% та розпізнає те, що знаходиться на зображенні. Поставлену задачу цієї роботи *побудувати нейронну мережу та натренувати її на розпізнавання та класифікацію зображень одягу* можна вважати виконаною.

*Додатки*

1. Лістинг коду

Modeluse.py

```
from PIL import Image
from torchvision import transforms
from torch.autograd import Variable
import torch
import torch.nn as nn



class Model:

    def __init__(self):
        self.model = nn.Sequential(torch.load("models/model.pkl"))
        self.labels = open("classes.txt", 'r').read().splitlines()
        return

    def predict(self, image_path):
        test_transforms = transforms.Compose([transforms.Resize(224),
                                              transforms.ToTensor(),
                                              transforms.Normalize([0.485,
0.456, 0.406],
                                                                   [0.229,
0.224, 0.225])
                                              ])
        img = Image.open(image_path)
        inp = Variable(test_transforms(img).float().unsqueeze_(0))
        inp = inp.to(torch.device("cpu"))
        index = self.model(inp).data.cpu().numpy().argmax()
        return self.labels[index]


def do(file):
    learner = Model()
    return learner.predict(file)
```

main.py

```
import logging
from io import BytesIO

from telegram.ext import Updater, CommandHandler, MessageHandler, Filters
from modeluse import do
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s',
                    level=logging.INFO)
logger = logging.getLogger(__name__)


def start(update, context):
    update.message.reply_text('Hi! Please send your picture to be processed.')


def echo(update, context):
    update.message.reply_text("Please don't talk to me! Only send pictures of
clothes.")


def photo(update, context):
    file = context.bot.get_file(update.message.photo[-1].file_id)
    f = BytesIO(file.download_as_bytearray())
    result = do(f)
    response = 'I processed that and the result was ' + result + '.'
    context.bot.send_message(chat_id=update.message.chat_id, text=response)


def main():
    updater = Updater("1758984507:AAHoVWVQ0qhU-c1TMWuaOBL-YJwj74-NHNg",
use_context=True)

    dp = updater.dispatcher
    dp.add_handler(CommandHandler("start", start))
    dp.add_handler(MessageHandler(Filters.text, echo))
    dp.add_handler(MessageHandler(Filters.photo, photo))
    updater.start_polling()
    updater.idle()

if __name__ == '__main__':
    main()
```

*Список використаних джерел*


1. A. Y. Ng Deep learning course. Режим доступу: https://www.coursera.org/learn/neural-networks-deep-learning/home/welcome
2. Документація бібліотеки Fastai. Режим доступу: https://docs.fast.ai/
3. Документація бібліотеки Pytorch. Режим доступу: https://pytorch.org/
4. DeepFashion Dataset. Режим доступу: http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html
5. B. D. Ripley Neural Networks and Related Methods for Classification – 1-15.
6. J. Bouvrie Notes on Convolutional Neural Networks – 3-5.
7. Z. Liu DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations / Z. Liu, P. Luo, S. Qiu, X. Wang, X. Tang.
8. M. Thorpe Deep Limits of Residual Neural Networks / M. Thorpe, Y. van Gennip – 1-6.
9. K. He Deep Residual Learning for Image Recognition / K. He, X. Zhang, S. Ren, J. Sun
10. M. S. Santos Cross-Validation for Imbalanced Datasets: Avoiding Overoptimistic and Overfitting Approaches / M. S. Santos, J. P. Soares, P. H. Abreu, H. Araújo, J. Santos.
11. MNIST Dataset. Режим доступу: http://yann.lecun.com/exdb/mnist/

Fashion MNIST Dataset. Режим доступу: https://www.kaggle.com/zalando-research/fashionmnist

13. T. G. Dietterich Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms / T. G. Dietterich, E. B. Kong – 1-5.
14. ResNet Documentation. Режим доступу: https://pytorch.org/hub/pytorch_vision_resnet/
15. Telegram bot API. Режим доступу: https://core.telegram.org/bots/api
16. A.Y. Ng On Optimization Methods for Deep Learning / Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A. Y. Ng
17. P. Baldi Assessing the accuracy of prediction algorithms for classification: an overview / P. Baldi, S. Brunak, Y. Chauvin, C. A. F. Andersen, H. Nielsen.
18. S. Sharma Activation functions in neural networks / S. Sharma, S. Sharma.