Київський національний університет імені Тараса Шевченка

Факультет комп'ютерних наук та кібернетики

Кафедра моделювання складних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня бакалавра

за спеціальністю 113 «Прикладна математика»

на тему:

Застосування нейронних мереж для виявлення захворювань за результатами рентгенодіагностики

студента 4 курсу

Таргонського Валерія Андрійовича

_____

Науковий керівник:

доцент, кандидат фізико–математичних наук

Лівінська Г.В.

_____

Робота заслухана на засіданні кафедри моделювання складних систем та рекомендована до захисту, протокол № 10 від 08 травня 2021 р.

Завідувач кафедри МСС канд. фіз.-мат. наук, доц. Черній Д.І.

Київ – 2021

РЕФЕРАТ

Обсяг роботи 35 сторінок, 10 ілюстрацій, 12 джерел посилань.

НЕЙРОННІ МЕРЕЖІ, РЕНТГЕНОДІАГНОСТИКА, МЕТОДИ РЕГУЛЯРИЗАЦІЇ, МЕТОДИ ОПТИМІЗАЦІЇ, ФУНКЦІЇ АКТИВАЦІЇ.

Об’єктом роботи є нейронні мережі. Предметом роботи є розробка нейронних мереж для класифікації рентген знімків, аналіз функцій активації, методів оптимізації та регуляризації.

Метою роботи є розробка двох нейронних мереж, згорткової та повнозв’язної, які дають принаймні точність 90% в задачі класифікації рентген знімків.

Методи розроблення: аналіз існуючих методів розроблення нейронних мереж, методів оптимізації та регуляризації, огляд популярних та ефективних функцій активації, а також створення власних нейронних мереж на основі проаналізованої інформації.

Інструменти розроблення. Для розробки реалізації було використано мову програмування Python. Під час розробки в якості середовища було використано Google Colab.

Результат роботи. Було досліджено уже існуючі методи побудови нейронних мереж для реалізації власних двох нейронних мереж, описано необхідний математичний апарат для побудови алгоритмів, розроблено згорткову та повнозв’язну мережу, які дають точність вище 90% на тестовій вибірці.

ЗМІСТ

- Вступ 4
- 1. Дані та попередня обробка 7
- 1.1 Попередня обробка даних для повнозв’язної мережі 8
- 1.2 Попередня обробка даних для згорткової мережі мережі 8
- 1.3 Рентгенодіагностика пневмонії 9
- 2. Оптимізація нейронних мереж 10
- 2.1 Стохастичний градієнтний спуск 11
- 2.2 Стохастичний градієнтний спуск з імпульсом 11
- 2.3 Параметри ініціалізації 13
- 2.4 Алгоритм Adam 14
- 3. Архітектура нейронних мереж 16
- 3.1 Повнозв’язні мережі 16
- 3.2 Згорткові мережі 18
- 3.3 Важливі види шарів згорткової мережі 19
- 4. Регуляризація 20
- 4.1 L^2 регуляризація 20
- 4.2 L^1 регуляризація 21
- 4.3 Процедура відсіву 22
- 5. Функції активації 23
- 5.1 Softmax 23
- 5.2 ReLU 24
- 5.3 Sigmoid 24
- 6. Опис алгоритмів 26
- 6.1 Повнозв’язна нейронна мережа 26
- 6.2 Згорткова нейронна мережа 29
- Висновки 33
- Список використаних джерел 35
- Додаток А 36
- Додаток Б 40

ВСТУП

Дослідники з давніх часів мріяли створити машину, яка вміє мислити. Ці бажання ще йдуть з Давньої Греції, адже міфічні фігури Дедала, Гефеста та Пігмаліона можна інтерпретувати як легендарних дослідників.

Перші ґрунтовні ідеї в цьому напрямку починаються з 1940 року. Моделі, розроблені в цей час, були мотивовані нейробіологічною точкою зору, тобто людським мозком, ці моделі були лінійними, виду

$$f(\mathrm{x},\mathrm{w}) = x_1 w_1 + \cdots + x_n w_n ,$$

де $x_1, \dots, x_n$ – це вхідні дані, пов'язані з виходом y, ваги $w_1, \dots, w_n$ та обчислювальна функція f. З цього почалася перша хвиля досліджень нейронних мереж, яка називалися кібернетикою. В 1950-х роках перцептрон Розенблата став першою моделлю, яка могла вивчати ваги, що визначають категорії, на основі вхідних даних з кожної категорії. Проте, лінійні моделі мають багато обмежень. Тому критики, які спостерігали за цими недоліками, викликали негативну реакцію в бік біологічного натхнення навчання у цілому в шістдесятих роках. Це було перше серйозне падіння популярності нейронних мереж ([1]).

Друга хвиля популярності у 1980 – 1990 роках називалася коннекціонізмом. Головною ідеєю цього напрямку було те, що об'єднавши багато простих обчислювальних одиниць в одну мережу, можна досягнути інтелектуальної поведінки. Деякі концепції, які виникли в цей час, залишаються базовими і в сучасному глибокому навчанні. Одним з досягнень цього руху було успішне використання зворотного поширення для навчання глибоких нейронних мереж і популяризація цього алгоритму. Проте проекти, які були засновані на нейронних мережах, висували занадто амбітні вимоги для пошуку інвестицій. З часом інвестори розчарувалися в цих дослідженнях, вирішивши, що вони не виправдовують їхні очікування. Одночасно з цим почали розвиватися і інші галузі машинного навчання, які давали гарні результати в багатьох задачах. Все це призвело до падіння популярності нейронних мереж до 2007 року [1].

З 2007 року по наш час – це третя хвиля популярності нейронних мереж, яка має назву глибоке навчання. Сучасний термін “глибоке навчання” виходить за рамки нейробіологічної точки зору. Основна причина зниження ролі нейробіології у сучасному дослідженні глибокого навчання, полягає в тому що у нас недостатньо інформації про мозок, щоб використовувати її в якості орієнтира. Адже ми далекі від розуміння навіть самих простих та гарно вивчених ділянок мозку.

Виділимо основні причини популярності нейронних мереж в останні роки [1]:

а) Збільшення кількості доступних навчальних матеріалів. В наш час майже кожний має вільний доступ до інтернету та соціальних мереж, що суттєво збільшує обсяг інформації.

б) Збільшення розмірів моделей глибокого навчання, в зв’язку з вдосконаленням комп’ютерної інфраструктури.

в) Нейронні мережі допомагають розв’язувати все більш важкі задачі зі зростаючою точністю. Адже виникають нові і потужні методи навчання глибоких мереж.

Людина добре розпізнає зображення, проте довгий час для світу та машинного навчання зокрема, було проблемою робота зі знімками. В цій роботі буде продемонстровано, що нейронні мережі розв’язують цю проблему і показують гарний результат у подібних задачах.

Актуальність роботи. Під час коронавірусу та особливо коли він лише починався, окрім великої кількості хворих, було багато остраху та будь які невеликі симптоми лякали людей і вони хотіли дізнатися чи справді в них є певна хвороба. Відповідно лікарям приходилося стикатися з великою кількістю пацієнтів, не обов’язково хворих. Як відомо, коронавірус в якості ускладнення може викликати пневмонію. Саме тому актуальною і важливою проблемою є швидко та точно діагностувати по рентген-зображенням здорова людина чи ні.

Метою даної роботи є досягнення точності принаймні 90% в задачі класифікації здорових легень та хворих пневмонією на тестовій вибірці.

Завданнями роботи є написання двох нейронних мереж: повнозв'язної та згорткової. Опис необхідного математичного апарату. Налаштування необхідних гіперпараметрів.

Об'єктом дослідження є нейронні мережі.

Предметом дослідження є алгоритми оптимізації у машинному навчанні, методи регуляризації, попередня обробка даних, архітектури нейронних мереж, функції втрат, функції активації та графік швидкості навчання.

Для виконання роботи було використано мову програмування Python, та наступні пакети: tensorflow, os, matplotlib, google.colab, numpy, sklearn.

Аналіз структури роботи та новизна отриманих результатів. Перший розділ присвячений опису даних, з якими йде подальша робота, та їх обробці для подання на вхід моделям. Деякі визначення та формулювання для цього розділу були взяті з першоджерела [7]. Основні теоретичні відомості для другого розділу були взяті з першоджерела [3]. В цьому розділі мова йде про оптимізацію мереж. Основні ідеї третього розділу були взяті з курсу лекцій про архітектури нейронних мереж, прослуханого автором. Теоретичні відомості для четвертого розділу були взяті з першоджерела [2], мова у цьому розділі йде про методи регуляризації. П'ятий розділ присвячений функціям активації, більшість інформації взята з першоджерел [4] та [5]. Основний, шостий розділ, складається з опису двох алгоритмів написаних самостійно автором. В цьому розділі йде мова про деталі реалізації, про те, чим було обумовлено те чи інше використання певних параметрів та шарів, продемонстровані остаточні результати навчання, валідації та результат на тестовій вибірці. Опис сучасної архітектури MobileNetV2 взятий з першоджерела [6].

1. ДАНІ ТА ПОПЕРЕДНЯ ОБРОБКА

Датасет із зображеннями здорових та хворих легень, був взятий на інтенсиві “Machine learning в MedTech”, який відбувався 26-27 грудня 2020 року. Кожне зображення має певний розмір, наприклад 1857x1317x3, де остання цифра 3 – це кількість каналів зображення, а саме червоний, зелений та блакитний. До кожного зображення є відповідь – здорові ці легені чи ні, тобто наявна так звана "мітка". У програмі здоровим легеням у відповідність буде ставитися мітка один, хворим нуль.

Тренувальна вибірка складається з 1735 елементів, серед яких 1416 зображень здорових легень, а інші 319 хворих пневмонією. Тестова вибірка складається з 624 зображень, з яких 390 прикладів хворих і 234 здорових легень. Зрештою, усі ці зображення переводяться у масив та додаються в один список, після чого розділяються у відношенні 70:15:15 на відповідно тренувальні дані, валідаційні та тестові. Отримані три списка переводяться у масиви. Перша частина даних йде на вхід моделі, тобто саме на цих даних вона буде навчатися, на другій частині даних буде відбуватися валідація моделі, тобто підбиратимуться гіперпараметри, і остаточну точність моделі зможемо оцінити на третій частині. Мітки класів обробляються функцією to_categorical з модуля tensorflow.keras.utils, яка замінює число нуль на вектор $(1; 0)$, а число 1 на $(0; 1)$.

На рисунку 1.1 можна побачити приклади зображень до обробки.

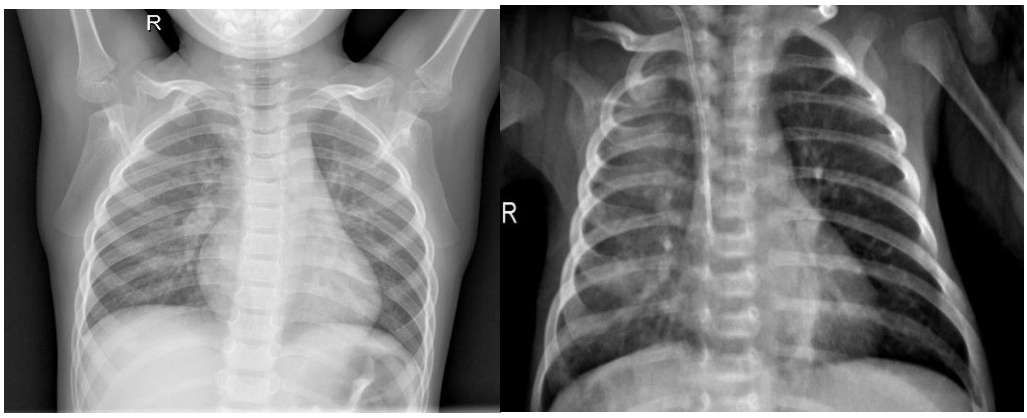

Рисунок 1.1 – Зображення до обробки.

Примітка. На усіх зображеннях в цьому розділі, зліва зображенні приклади здорових легень, відповідно справа – хворих.

1.1 Попередня обробка даних для повнозв’язної мережі

Для повнозв’язної мережі змінимо розмір даних до 100 x 100 x 3. Отримаємо наступні зображення.

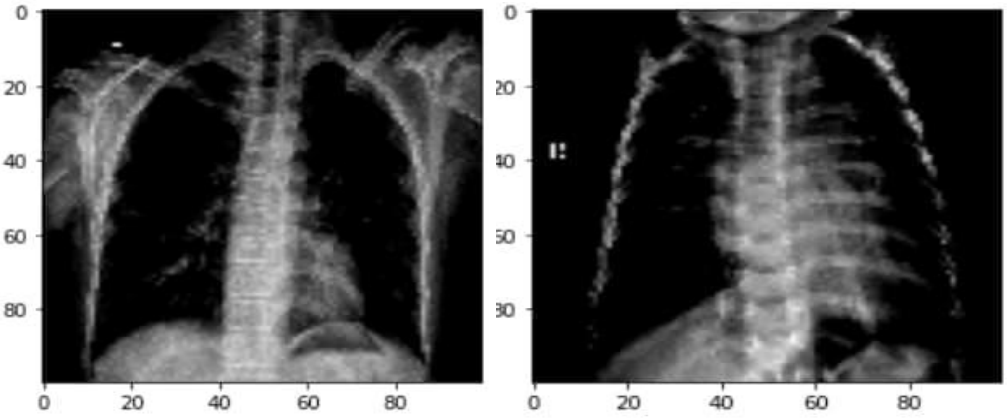

Рисунок 1.2 – Зображення розміром 100 x 100 x 3.

Після розбиття даних на три частини, елементи у кожній частині нормуються, шляхом операції ділення кожного пікселя на двісті п’ятдесят п’ять. Так як у кожному фільтрі яскравість пікселя змінюється у діапазоні від 0 до 255. Відповідно після нормування усі ці величини булуть знаходитися у діапазоні від 0 до 1 і тоді будуть краще опрацьовуватися нейронною мережею.

1.2 Попередня обробка даних для згорткової мережі

Враховуючи, що за основу згорткової мережі взята модель MobileNetV2, то і для обробки даних використовується функція preprocess_input з модуля tensorflow.keras.applications.mobilenet_v2. Ця функція застосовується перед тим, як усі зображення додаються в один список. Ще одною особливістю даної моделі є те, що вхідні дані повинні мати розмірність 224 x 224 x 3. Зображення даної розмірності, мають вигляд, зображений на Рис. 1.3.

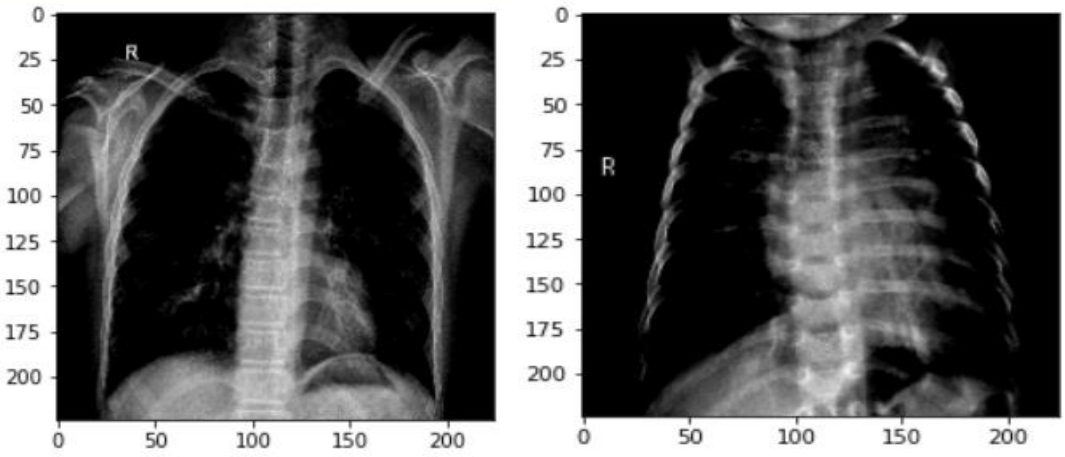

Рисунок 1.3 – Зображення розміром 224 x 224 x 3.

1.3 Рентгенодіагностика пневмонії

Пневмонія - це запальне захворювання легень, яке в першу чергу вражає невеликі повітряні мішечки, відомі як альвеоли. Зазвичай симптомами є деякі комбінації продуктивного або сухого кашлю, біль в грудях, лихоманка та затруднене дихання. Рентгенограма грудної клітини доволі часто використовується в діагностиці. Людям з легкою формою захворювання візуалізація необхідна тільки тоді, коли є потенційні ускладнення, коли лікування не покращує загальний стан або тоді коли причина хвороби не виявлена. Також якщо пацієнт хворий і потребує госпіталізації, рекомендується зробити йому рентгенограму грудної клітини. Проте, інколи результати можуть не відповідати тяжкості захворювання і не дозволяють однозначно розділити бактеріальну і вірусну інфекцію ([7]).

Рентгенологічні прояви пневмонії можуть бути класифіковані як крупозна пневмонія, бронхопневмонія, лобулярная пневмонія і інтерстиціальна пневмонія ([7]).

В даній роботі класифікація проводиться лише на 2 класи: здорова людина і людина з пневмонією. Проте, при наявності необхідних даних (для кожного з класів, про які йшла мова в попередньому абзаці), можна поліпшити результати роботи і провести багатокласову класифікацію.

2. ОПТИМІЗАЦІЯ НЕЙРОННИХ МЕРЕЖ

В цьому розділі наведемо алгоритми оптимізації, які використовуються для глибоких моделей. В чому ж відмінність між алгоритмами такої оптимізації та класичними алгоритмами?

В більшості випадків у машинному навчанні метою є ефективність деякої міри P. Цю задача може бути важко розв'язати. Тому, в такому випадку, мінімізують деяку цільову функцію $J(\theta)$ в припущенні що це покращить значення P. В цьому і є основна відмінність з класичною оптимізацією, де головною метою є мінімізація J [3].

Кількість тренувальних прикладів, які використовує алгоритм за один підхід, називають пакетом [3]. На основі очікуваного значення функції втрат, алгоритми оптимізації машинного навчання зазвичай швидко обчислюють кожне оновлення параметрів. Найпопулярнішими та найефективнішими методами у глибокому навчанні на даний момент є модифікації градієнтного спуску, про них і піде мова [3].

Алгоритми оптимізації поділяються на три типи: детерміновані, онлайн методи та стохастичні. Методи детермінованого градієнту використовують всі навчальні приклади одночасно. Онлайн методи використовують для свого навчання лише один приклад за раз. Стохастичні методи використовують більше одного прикладу, але не усі тренувальні. Для того щоб підібрати оптимальний розмір пакету можна використовувати наступні фактори:

а) Більший розмір пакету дає більш точну оцінку градієнта, але без лінійної віддачі.

б) Очевидно, об'єм використаної пам'яті залежить від розміру пакету, тому це є обмеженням для вибору розміру пакету та для багатьох налаштувань пристроїв.

в) Невеликі розміри партії можуть давати ефект регуляризації, можливо через шум, який вони додають у процесі навчання. [3]

Враховуючи, що детерміновані методи схильні до перенавчання та потребують великого об’єму пам’яті, а онлайн методи дають невисоку точність, то зазвичай використовуються методи саме стохастичного градієнту та його модифікації. Про них і піде мова. [3]

2.1 Стохастичний градієнтний спуск

Найважливішою властивістю цього алгоритму і подібних до нього є те, що час обчислень на оновлення не збільшується зі збільшенням кількості навчальних прикладів. Збіжність алгоритму відбувається навіть тоді, коли кількість тренувальних прикладів стає дуже великою. Для досить великого набору даних стохастичний градієнтний спуск може зійтися в межах деякої фіксованої похибки на тестових даних, перш ніж він обробить всю навчальну вибірку. [3]

Алгоритм стохастичного градієнтного спуску:

а) Задаємо параметри інтенсивності навчання $\epsilon_1, \epsilon_2, \ldots$, векторний параметр зсуву та ваг θ та встановлюємо лічильник ітерацій $k = 1$.

б) В циклі, поки не буде виконаний критерій зупинки, виконуються наступні операції:

1) обирається підвибірка розміру m з навчальних даних $\{x^{(1)}, \ldots, x^{(m)}\}$, з розміткою $y^{(i)}$ – вектор міток відповідних даних;

2) обчислюється оцінка градієнту: $\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$, де L – це деяка функція втрат для відповідного рівня, а f – певна перетворююча функція;

3) оновлюється параметр θ: $\theta - \epsilon_k, \hat{g} \rightarrow \theta$;

4) перехід до наступної ітерації: $k + 1 \rightarrow k$. [3]

2.2 Стохастичний градієнтний спуск з імпульсом

Цей метод розроблений для того, щоб прискорювати швидкість навчання в порівнянні зі звичайним стохастичним градієнтним спуском.

Алгоритм імпульсу накопичує ковзне середнє минулих градієнтів, яке зменшується експоненціально і продовжує рух в їхньому напрямку. Формально алгоритм імпульсу вводить векторну змінну v – це напрямок і швидкість, з якими параметри переміщаються в просторі параметрів. Швидкість встановлена як середнє значення негативного градієнта, який зменшується експоненціально. [3]

Алгоритм стохастичного градієнтного спуску з імпульсом:

а) Задаємо інтенсивність навчання ∈, параметр імпульсу α, векторний параметр ваг та зсуву θ та векторний параметр v швидкості та напрямку.

б) В циклі, поки не буде виконаний критерій зупинки, виконуються наступні операції:

1) обирається підвибірка розміру m з навчальних даних $\{x^{(1)}, \dots, x^{(m)}\}$, з розміткою $y^{(i)}$;

2) обчислюється оцінка градієнту: $g = \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$;

3) оновлюється параметр v: $\alpha v - \in g \rightarrow v$;

4) оновлюється параметр θ: $\theta + v \rightarrow \theta$. [3]

Модифікацію цього алгоритму світ побачив у 2013 році, за ідею було взято метод прискореного градієнту Нестерова. Правила оновлення в модифікованому алгоритмі наступні:

$$\alpha v - \in \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)}; \theta + \alpha v\right), y^{(i)}\right) \rightarrow v;$$

$$\theta + v \rightarrow \theta$$

де L – це функція втрат певного рівня, f – певна перетворююча функція.

Відмінність імпульсу Нестерова і стандартного імпульсу полягає в оцінюванні градієнту. У модифікованому алгоритмі градієнт обчислюється з урахуванням поточної швидкості. Тобто можна сказати, що алгоритм Нестерова намагається ввести деякий коефіцієнт як корекцію до стандартного методу імпульсу. Доволі часто цей алгоритм показує кращий результат ніж класичний. [3]

2.3 Параметри ініціалізації

Зазвичай моделі глибокого навчання є ітеративними, тому користувач має вказати початкову точку, з якої слід починати ітерації. Слід пам’ятати, що вибір початкової точки часом суттєво впливає на те, чи буде алгоритм збігатися в цілому. Коли ж навчання дійсно збігається, то початкова точка може визначати, наскільки швидко буде відбуватися збіжність. [3]

Сучасні стратегії ініціалізації доволі прості та евристичні, адже оптимізація нейронних мереж ще недостатньо вивчена галузь. Наприклад, великі початкові ваги можуть призвести до суттєвішого порушення симетрії, допомагаючи уникнути надмірних одиниць. Вони також допомагають уникнути втрати сигналу при прямому або зворотному поширенні через лінійну складову кожного шару, бо великі значення в матриці призводять до більшого результату множення матриці. Однак занадто великі початкові ваги можуть призвести до збільшення значень під час прямого або зворотного поширення.

Початкові значення W – матриці ваг, можна ініціалізувати нулями, проте це не дуже гарний підхід, адже множення на нуль неефективне. Тому краще ці значення задати або випадковим чином, або наприклад випадковими значеннями з нормального розподілу, з нульовим середнім та одиничною дисперсією. Початкові значення b – вектор зсуву, можна задавати нулями, або також випадковими числами.

Зауважимо, що немає якоїсь загальної і оптимальної стратегії визначення параметрів ініціалізації, проте дослідники нейронних мереж давно усвідомили, що швидкість навчання – одна з найскладніших і найважливіших гіперпараметрів, оскільки вона істотно впливає на продуктивність моделі. Функція вартості часто буває чутлива до одних напрямів в просторі параметрів і нечутлива до інших. Алгоритм імпульсу

може дещо пом'якшити ці проблеми, але робить це за рахунок введення ще одного гіперпараметру.

Отже, якщо вважати, що напрямки чутливості в якійсь мірі збігаються з осями координат, чи має сенс використовувати окрему швидкість навчання для кожного параметра і автоматично адаптувати ці швидкості навчання протягом усього навчання? [3]

2.4 Алгоритм Adam

Adam – це адаптивний алгоритм швидкості навчання, розроблений в 2014 році. Тобто адаптації індивідуальної швидкості навчання для параметрів моделі під час навчання. Ранні підходи адаптивних алгоритмів, були засновані на простій ідеї: якщо частинна похідна втрат по заданому параметру моделі залишається того ж знаку, то швидкість навчання повинна збільшитися. Якщо ця частинна похідна змінює знак, то швидкість навчання повинна зменшуватися, проте таке правило застосовується лише до детермінованої оптимізації.

Перевага Adam в тому, що цей алгоритм досить стійкий до вибору гіперпараметрів, хоча швидкість навчання іноді необхідно дещо змінити із запропонованою за замовчуванням. [3]

Алгоритм Adam:

а) Задаємо коефіцієнт кроку при навчанні ϵ; експонентні швидкості згасання для оцінок моментів ρ_1, ρ_2 з проміжку $[0,1)$; малу сталу величину δ, яка використовується для чисельної стабілізації; векторний параметр ваг та зсуву θ; номер ітерації $t = 0$; першу та другу моменті змінні: $s = 0, r = 0$;

б) В циклі, поки не буде виконаний критерій зупинки, виконуються наступні операції:

1) обирається підвибірка розміру m з навчальних даних $\{x^{(1)}, \dots, x^{(m)}\}$, з розміткою $y^{(i)}$;

2) обчислюється оцінка градієнту: $g = \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right)$;

3) крок збільшується на одиницю, $t + 1 \rightarrow t$;

4) оновлюється оцінка першого моменту: $\rho_1 s + (1 - \rho_1)g \rightarrow s$;

5) оновлюється оцінка другого моменту: $\rho_2 r + (1 - \rho_2)g^2 \rightarrow r$;

6) корегується зсув першого моменту: $\frac{s}{1-\rho_1^t} \rightarrow \hat{s}$;

7) корегується зсув другого моменту: $\frac{r}{1-\rho_2^t} \rightarrow \hat{r}$;

8) оновлюється оцінка градієнту: $\nabla_\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$;

9) застосовується оновлення параметру θ: $\nabla_\theta + \theta \rightarrow \theta$. [3]

3. АРХІТЕКТУРА НЕЙРОННИХ МЕРЕЖ

Інформація для цього розділу, була взята з курсу лекцій “Global Logic Machine Learning”. Мова піде про те, як поетапно проходить навчання глибокої (тобто мережі, яка містить один або більше прихованих шарів) повнозв’язної нейронної мережі в загальному випадку; про те, що таке згортка і що з себе представляє згортковий шар; які параметри містять такі шари та як відбувається навчання. Окрім цього будуть згадані і інші види шарів, які використовуються при підході з використанням згорткових шарів.

3.1 Повнозв’язні мережі

На вхід модель отримує набір даних:

$$x_i, \qquad i = 1, 2, \dots, n$$

$x_i -$ вектор певної довжини, та набір міток $y_i,\ i = 1, 2, \dots, n$ (у випадку бінарної класифікації це набір нулів та одиниць).

Ці дані надходять на перший шар нейронної мережі (відповідно його розмірність має бути такою ж, як розмірність вектора вхідних даних), та виконуються наступні 2 операції:

$$Z_1 = W_1^T X + b_1,$$

$$A_1 = \sigma_1(Z_1),$$

де W_1 – матриця ваг (вагова матриця) нейронів першого прихованого шару, її розмірність: кількість вхідних даних на кількість нейронів першого прихованого шару, на початку ініціалізується певним чином (наприклад, випадковими числами з нормального розподілу, з нульовим математичним сподіванням та одиничною дисперсією); X – матриця вхідних даних; b_1 – зсув першого прихованого шару, на початку ініціалізується певним чином (наприклад, нулем); σ_1 – деяка нелінійна функція активації першого прихованого шару (про функції активації буде сказано пізніше).

Параметри наступних прихованих шарів обчислюються ітеративно таким чином:

$$Z_i = W_i^T A_{i-1} + b_i$$

$$A_i = \sigma_i(Z_i),\ i = 2, 3, \dots, l-1$$

де W_i – це матриця ваг нейронів i-го прихованого шару (розміру: кількість нейронів на попередньому шарі на кількість нейронів цього прихованого шару), з початковими значеннями, що ініціалізуються певним чином; l – кількість шарів нейронної мережі; b_i – параметр зсуву i-го прихованого шару, початкове значення ініціалізується певним чином; σ_i – деяка нелінійна функція активації i-го прихованого шару.

Далі виконуються наступні операції з останнім, вихідним шаром, який має розмірність рівну кількості класів:

$$Z_l = W_l^T Z_{l-1} + b_l$$

$$A_l = \sigma_l(Z_l)$$

Зазвичай функція σ_l – це сигмоїд (sigmoid) або softmax. Якщо це була остання ітерація, то A_l – це матриця передбаченнь даної моделі для заданих початкових даних. У задачі бінарної класифікації це число від 0 до 1. Відповідно, встановивши поріг 0.5, можна дати бінарну відповідь нуль чи один.

Якщо ж це не була остання ітерація, то на даний момент було описано лише алгоритм прямого розповсюдження. Тоді тут Z_i – це результат лінійного перетворення, а A_i – результат нелінійного перетворення, $i = 1, 2, 3, \dots, l$.

Далі йде сам процес навчання, так зване зворотне поширення помилки, в результаті якого матимемо оцінки параметрів, які мінімізують задану цільову функцію. Отже, задається деяка цільова функція задачі, ми намагаємося її оптимізувати, знайти точку мінімуму. За допомогою градієнтного спуску і його модифікацій це зазвичай вдається зробити, знайшовши величини, які задіяні в алгоритмі та оновивши параметри ваг та зсуву. Більш детально це описано, в розділі “Оптимізація нейронних мереж”.

Після виконання всіх необхідних ітерацій модель є навченою. Тобто ті ваги і зсуви, які вона має після останньої ітерації, є остаточними, вони мінімізують цільову функцію. Отже тепер модель можна використовувати на практиці, тобто робити прогнози на деяких інших даних. Для того щоб отримати прогноз для нових даних, потрібно повторити увесь процес прямого поширення з використанням отриманих оцінок параметрів.

3.2 Згорткові мережі

В нейронних мережах згорткою (або крос-кореляцією) деяких функцій f та g називають наступну операцію:

$$(f * g)[n] = \sum_m f[m]g[n-m]$$

Згортка, якщо говорити грубо, демонструє певний ступінь перетину того що ми шукаємо, з тим що ми маємо.

Наприклад, ми маємо вхід (1,3,3,0,1,2) та застосуємо до нього фільтр (2,0,1). В результаті 1d згортки, отримаємо вихід:

$$5\ (5 = 1 \cdot 2 + 3 \cdot 0\ + 3 \cdot 1);\ 6\ (6 = 3 \cdot 2 + 3 \cdot 0 + 0 \cdot 1);$$
$$7\ (7 = 3 \cdot 2 + 0 \cdot 0 + 1 \cdot 1);\ 2\ (2 = 0 \cdot 2 + 1 \cdot 0 + 2 \cdot 1)$$

Отже вихідний вектор (5,6,7,2).

В даній роботі ми маємо справу з кольоровим зображенням, тому на вхід подається не вектор, а тензор. Тобто для кожного кольору (каналу) будуть свої фільтри. І метою використання згорткової мережі є знайти найбільш оптимальний фільтр за допомогою зворотного поширення. Для певного розуміння того як відбувається зворотне поширення, розглянемо формули для знаходження градієнтів:

$$\frac{\delta L}{\delta F} = \text{Згортка}(X, \frac{\delta L}{\delta O})$$

$$\frac{\delta L}{\delta X} = \text{Повна згортка}(\text{повернений на 180 градусів фільтр } F, \frac{\delta L}{\delta O})$$

де L – функція втрат.

Також є такі важливі параметри згорткової мережі як відступ (padding) та крок (stride).

Відступ (padding) – це певний відступ або розширення даних по краях нулями, для того аби не втрачати інформацію, яка знаходиться на кінцях наших даних. Зазвичай цей параметр встановлюють рівним одиниці.

Крок (stride) – це крок з яким обирається наступний фільтр, він застосовується для певного контролю інформації, яка передається. Тобто, наприклад, якщо обрати занадто малий крок, то рівень абстракції може зменшуватися не так швидко, як це може бути необхідно.

Формула для обчислення кількості змінних після застосування параметрів відступу та кроку:

$$n_{out} = \left[\frac{n_{in} + 2p - k}{s}\right] + 1$$

де n_{out} – кількість змінних на виході, n_{in} – кількість змінних на вході, p – значення параметру відступу, s – значення параметру кроку, k – розмір фільтру, [] – ціла частина числа.

3.3 Важливі види шарів згорткової мережі

Згладжуючий (Flatten) шар – це шар, який переводить дані у вектор. Наприклад (None,20,30,2) у (None,1200), де None означає будь-яке число вхідних даних у модель.

Збірний (pooling) шар – це шар який використовується для зменшення кількості інформації. Цей шар не бере участі в тренуванні і є сталим. Наприклад, якщо є матриця 4x4 і розмір збірки 2x2, то матриця 4x4 перетвориться на матрицю 2x2. Окрім max-pooling, це збірний шар, який кожну підматрицю замінює на найбільший елемент у ній, існують і інші способи обрати елементи з підматриці, наприклад average-pooling. Цей вид збірки заміняє підматрицю на середнє арифметичне значення усіх елементів у ній.

4. РЕГУЛЯРИЗАЦІЯ

Регуляризацією називають будь-які кроки, призначені для боротьби з перенавченням моделі. Адже ціллю моделі є не ідеальне вивчення і підгонка до тренувальних даних, а те щоб вона зрозуміла природу і основні закономірності цих даних, тобто щоб ця модель мала сенс у реальному житті. Зазвичай в результаті регуляризації може падати точність на навчальній вибірці, але вона буде зростати на тестовій.

Основним методом регуляризації є введення штрафу за надто велике значення параметру $\Omega(\theta)$ цільової функції J , величина якого збільшується при збільшенні визначеної норми параметру.. В результаті отримаємо регуляризовану цільову функцію $\tilde{J}$:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta) ,$$

де $0 \leq \alpha < +\infty$ - гіперпараметр, який зважує відносний вклад нормального штрафу відносно стандартної цільової функції. Якщо $\alpha = 0$, то регуляризація відсутня, відповідно чим більше цей гіперпараметр, тим більша регуляризація. [2]

4.1 L^2 регуляризація

При L^2-регуляризації штрафом за величину параметра є зменшення вагового коефіцієнта при цьому параметрі. Стратегія цієї регуляризації наближує ваги моделі до початку координат шляхом додавання члена регуляризації $\Omega(\theta) = \frac{1}{2}||w||_2^2$. Тут $||w||_2^2$ – це сума квадратів усіх ваг.

Якщо припустити, що параметр зсуву відсутній, то отримаємо наступну цільову функції:

$$\tilde{J}(w; X, y) = J(w; X, y) + \frac{\alpha}{2} w^T w$$

Відповідний параметр градієнта буде мати наступний вигляд:

$$\nabla_w \tilde{J}(w; X, y) = \nabla_w J(w; X, y) + \alpha w$$

Для одного кроку градієнта для оновлення ваг виконуємо таке оновлення:

$w \rightarrow \mathrm{w} - \in (\nabla_w J(\mathrm{w}; \mathrm{X}, \mathrm{y}) + \alpha w)$.

Розглянемо використання L^2-регуляризації на прикладі лінійної регресії. Функція втрат представляє собою суму квадратів помилок:

$$(Xw - y)^T (Xw - y)$$

Після додавання регуляризації цільова функція стане наступною:

$$(Xw - y)^T (Xw - y) + \frac{\alpha}{2} w^T w$$

Тому, $w = (X^T X + \alpha I)^{-1} X^T y$.

Розв'язок без регуляризації: $w = (X^T X)^{-1} X^T y$.

Як бачимо, нова матриця $(X^T X + \alpha I)^{-1}$ аналогічна до початкової, але з додаванням α до діагональних елементів. Ці елементи відповідають дисперсії кожної вхідної характеристики. Тому L^2-регуляризація змушує стискати ваги тих ознак, коваріація яких з цільовою функцією є низькою в порівнянні з цією доданою дисперсією. [2]

Отже, в результаті L^2-регуляризації незмінними залишаються лише ті напрямки, уздовж яких параметри вносять суттєвий вклад у зменшення цільової функції. Компоненти вектора ваг, які відповідають напрямкам, які не сприяють зменшенню цільової функції, зменшуються за рахунок використання регуляризації протягом усього навчання. [2]

4.2 L^1 регуляризація

Ця регуляризація додає до цільової функції компоненту $\Omega(\theta) = ||w||_1$, тобто суму абсолютних значень окремих параметрів. Подивимося на вплив L^1 регуляризації на просту модель лінійної регресії без зсуву. Як і у випадку зменшення ваг L^2 регуляризації, зменшення ваг L^1 також контролює силу регуляризації шляхом масштабування штрафу Ω з використанням додатного гіперпараметру α. Таким чином цільова функція буде наступною:

$$\tilde{J}(\mathrm{w}; \mathrm{X}, \mathrm{y}) = J(\mathrm{w}; \mathrm{X}, \mathrm{y}) + \alpha ||w||_1.$$

Відповідно градієнт матиме вигляд:

$$\nabla_w \tilde{J}(\mathrm{w}; \mathrm{X}, \mathrm{y}) = \nabla_w J(\mathrm{w}; \mathrm{X}, \mathrm{y}) + \alpha \mathrm{sign}(w),$$

де $\mathrm{sign}(w)$ – це знак w, який шукається поелементно.

Як бачимо, відмінність цього підходу з L^2-регуляризацією в тому, що вклад в градієнт більше не масштабується з кожним w_i, замість цього з'являється постійний множник, який рівний знаку w_i. Наслідком цього буде те, що ми не завжди побачимо алгебраїчні розв'язки квадратичних наближень $J(\mathrm{X}, \mathrm{y}; \mathrm{w})$. [2]

Що ж можна отримати використавши L^1-регуляризацію? В порівнянні з L^2, ця регуляризація дає більш розріджений розв'язок, тобто деякі параметри мають оптимальне нульове значення. В загальному L^2 не призводить до того, що параметри стають розрідженими, натомість L^2 може це робити при достатньо великих значеннях α. Ця властивість широко використовувалася як механізм відбору ознак. Адже штраф L^1 призводить до того, що деяка підмножина ваг стає нульовою, а це означає, що від відповідних функцій можна безпечно позбутися. [2]

4.3 Процедура відсіву

Відсів або "dropout" – це прохід усіх шарів нейронної мережі та встановлення ймовірності зберегти певні вузли чи позбутися (відсіяти) їх. Задається лише поріг, а ймовірність зберегти кожен вузол визначається випадковим чином. Відсів є корисним, оскільки мережа не може покладатися на жоден вхідний вузол, так як він може бути відкинутий. Тому модель не зможе надавати велику вагу певним характеристикам.

5. ФУНКЦІЇ АКТИВАЦІЇ

Функції активації грають ключову роль у ефективності нейронних мереж, адже саме вони створюють нелінійність і допомагають при розв'язанні важких задач. В цьому розділі будуть описані основні і найпопулярніші на даний момент функції активації, деякі з яких, зокрема будуть використані у даній роботі.

5.1 Softmax

Функція softmax, або нормалізована експоненціальна функція [10], є узагальненням логістичної функції на декілька вимірів. Softmax використовується в поліноміальній логістичній регресії, але найважливіше для нас, що вона використовується в якості останньої функції активації нейронної мережі для того, щоб нормалізувати вихідні дані мережі до розподілу ймовірностей за прогнозованими вихідними класами на основі аксіоми вибору Люса. Нагадаємо, що таке аксіома вибору Люса. [5]

У теорії ймовірностей аксіома вибору Люса, яка була сформульована Дунканом Люсом у 1959 році, стверджує, що ймовірність вибору одного елемента з набору замість іншого, який складається з багатьох елементів, не залежить від наявності або відсутності якихось інших елементів. Тобто, ймовірність вибору елемента i, з набору який містить j елементів, дорівнює:

$$P(i) = \frac{w_i}{\sum_j w_j}$$

де w – це міра деяких типових характерних властивостей конкретного предмету. [11]

Повернемося до softmax. Ця функція приймає в якості аргументу вектор з N дійсних чисел, після чого нормалізує його в розподіл ймовірностей, який складається з N ймовірностей, пропорційних експонентам вхідних чисел. Softmax є зручною функцією, оскільки після її застосування кожну компоненту можна буде сприймати як ймовірність приналежності до відповідного класу, бо кожна з них буде належати

інтервалу $(0; 1)$, та їх сума буде рівною 1. Одинична або стандартна функція softmax має наступний вигляд:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

де $i = 1, 2, \dots, N$ та $z = (z_1, \dots, z_N)$. [5]

5.2 Relu

Функція активації Relu (rectified linear unit) має наступний вигляд:

$$f(x) = x^+ = \max(0, x)$$

де x – вхід нейрона. [4]

В 2011 році було виявлено, що ця активаційна функція допомагає краще навчати глибокі нейронні мережі. На даний момент часу це одна з найпопулярніших функцій активації і застосовується на усіх шарах нейронної мережі, за винятком вихідного.

Сформулюємо основні переваги цієї функції, щоб зрозуміти, чому вона користується такою популярністю.

а) Розріджена активація. Тобто, у випадково заданої нейронної мережі активується тільки близько 50% прихованих модулів (з ненульовим виходом).

б) Більш якісне поширення градієнта. Виникає менше проблем зі зникаючим градієнтом в порівнянні з сигмоїдальними функціями активації, які насичуються в обох напрямках. [8]

в) Швидкодія. Серед математичних операцій використовується лише множення, додавання та порівняння. [4]

Звісно, relu має і певні недоліки, оскільки для кожної конкретної задачі необхідно аналізувати яка функція буде більш ефективною.

5.3 Sigmoid

Функція sigmoid - це математична функція, яка має характерну S-подібну криву або сигмовидну криву. Однією з найтиповіших sigmoid-функцій є логістична функція:

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

З математичної точки зору сигмоїдальна функція – це обмежена диференційована дійсна функція, яка визначена для всіх дійсних вхідних значень, має невід’ємну похідну в кожній точці, а також рівно одну точку перегину. Ці властивості є важливими для використання цієї функції в якості функції активації. [9]

6. ОПИС АЛГОРИТМІВ

В цьому розділі будуть описані моделі, побудовані автором роботи, їхні параметри, оцінки, графіки навчання та остаточна оцінка на тестовій вибірці.

6.1 Повнозв'язна нейронна мережа

Схема обраної архітектури проілюстрована на рисунку 6.1.

Рисунок 6.1 – Архітектура повнозв'язної моделі.

У колонці "інформація про шар" в якості інформації виступає або функція активації поточного шару, або ймовірність відсіву чи регуляризація ядра на цьому шарі. В центральній частині, в дужках коло назви шару вказано кількість змінних у шарі. В правій колонці вказано кількість параметрів відповідного шару. Всі параметри моделі піддаються навчанню і їх кількість рівна 15492354.

Як зображено на рисунку 6.1, модель містить 5 шарів. Вхідний шар має розмір 30000. Далі йде перший прихований шар, який містить 512 нейронів,

функцією активації виступає relu, кількість параметрів цього шару: 15360512 (512x30001). Після цього йде шар відсіву, з ймовірністю 0,2. Потім другий прихований шар, який містить 256 нейронів, функцією активації виступає relu, використовується регуляризація L^2 (з параметром α = 0,01) кількість параметрів цього шару: 131328. Останній вихідний шар, містить 2 нейрони, функція активації – softmax, кількість параметрів: 514.

Оптимізатор даної моделі – Adam, з усіма параметрами за замовчуванням, окрім кроків при навчанні (∈), цей параметр має свій графік. Навчання відбувається 22 етапи (епохи) (∈ = 0.001, з першого до 9 етапу, ∈ = 0.0001, з десятого до п’ятнадцятого етапу, ∈ = 0.00001 до кінця навчання). Для того щоб зменшувати параметр ∈ використовується функція LearningRateScheduler, це зумовлено тим, що нашою метою є отримати максимальне наближення до оптимальної точки. Розмір міні пакетів рівний 32, цей параметр був обраний методом перебору і дає необхідний результат. Для вимірювання точності класифікації (так звана метрика accuracy) ми рахуємо кількість разів, коли передбачення моделі збігається з істинною міткою. Результатом даної метрики буде відношення цього числа до загальної кількості передбачень. Цільовою функцією або функцією втрат є CategoricalCrossentropy, яка в загальному має наступний вигляд:

$$J(w) = -\frac{1}{N}\sum_{n=1}^{N} y_n \log \widehat{y_n} + (1 - y_n) \log(1 - \widehat{y_n})$$

де y_n – істинна мітка, $\widehat{y_n}$ – передбачення моделі. Також було розглянуто функції втрат BinaryCrossentropy та SparseCategoricalCrossentropy, проте обрана функція CategoricalCrossentropy на думку автора найкраще підходить до даної задачі.

Графік зміни точності моделі на тренувальній і валідаційній вибірці в залежності від етапу (червоний – тренувальний, блакитний – валідаційній) можна побачити на рисунку 6.2.

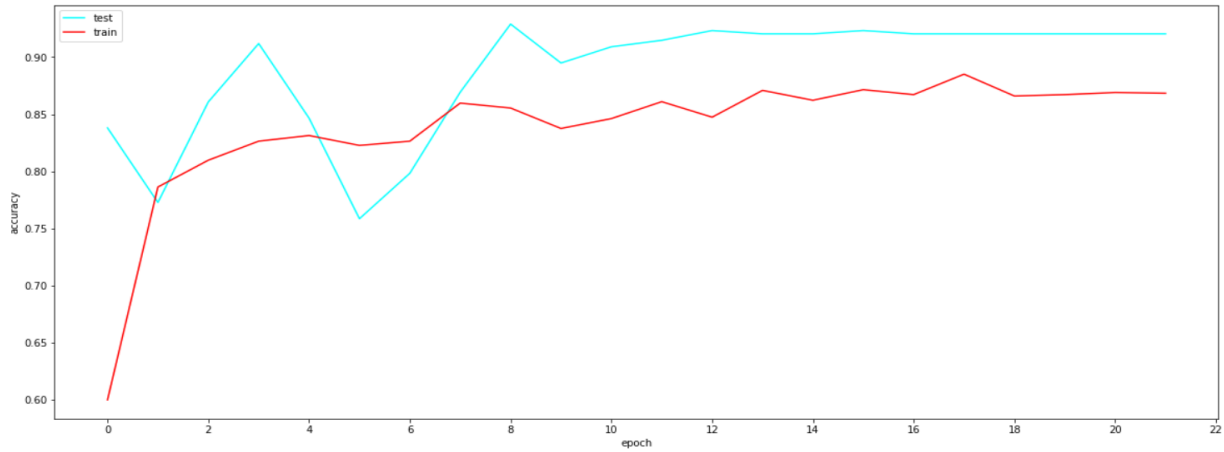

Рисунок 6.2 – Графік зміни точності першої моделі.

На даному графіку видно, що після сьомого етапу точність на валідаційній вибірці краща. Це пов’язано з використанням регуляризації. Також на графіку можна побачити, що за останні 10 етапів точність майже не змінювалася, це свідчить про те, що навчання можна вважати завершеним, адже надалі модель не буде показувати кращі результати.

Графік зміни функції втрат моделі на тренувальній і валідаційній вибірці в залежності від етапу (червоний – тренувальний, блакитний – валідаційний) можна побачити на рисунку 6.3.

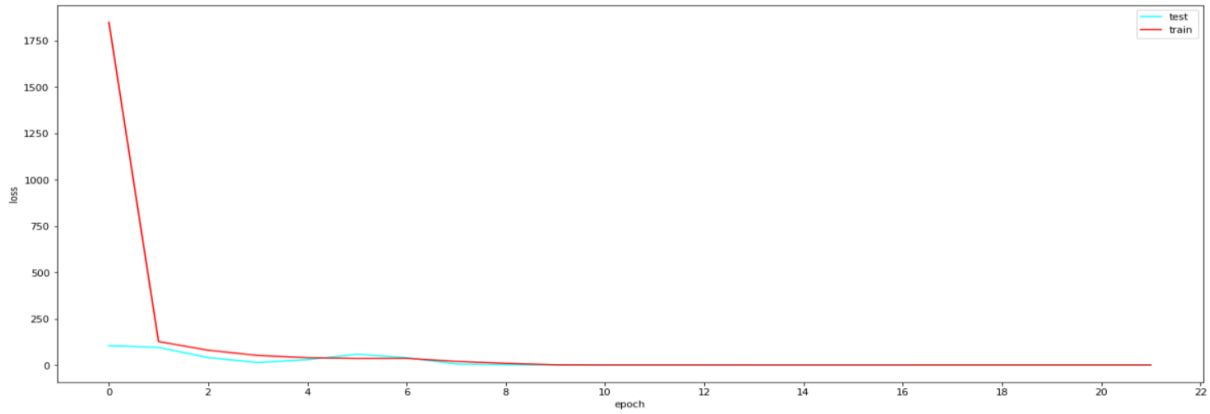

Рисунок 6.3 – Графік зміни функції втрат першої моделі.

Результат на тестових даних: точність – 0.9068, функція втрат – 2.3014. Це – результат після того, як тренування було завершено. За допомогою функції ModelCheckpoint було збережено ваги моделі в той момент часу, коли функція втрат на валідаційній вибірці була мінімальною. Це означає, що в майбутньому, завантаживши ці ваги в модель аналогічної архітектури, можна буде отримати таку ж точність з певним відхиленням.

6.2 Згорткова нейронна мережа

Тренувальних параметрів 164226, тобто усі параметри з MobileNetV2 не піддаються навчанню. Це означає, що усі нейрони з цих шарів будуть брати участі у прямому поширенні, а в зворотному – ні, і не будуть оновлюватися. Це пов’язано з швидкістю тренування, а також тому, що необхідного результату було досягнуто.

MobileNetV2 – це архітектура нейронної мережі. Розглянемо, чим вона відрізняється від звичайної згорткової нейронної мережі.

Архітектуру обраної моделі можна побачити на рисунку 6.4.

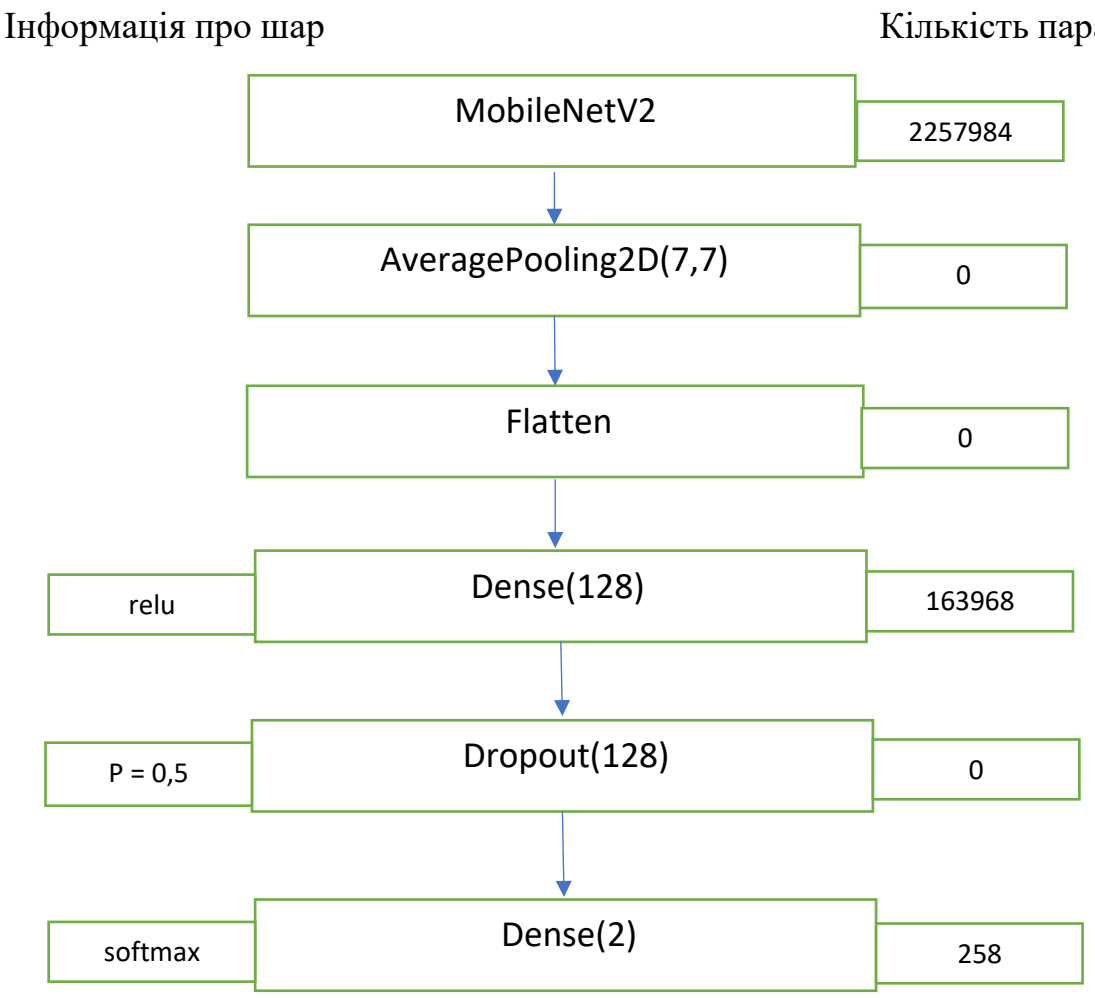

Рисунок 6.4 – архітектура другої моделі.

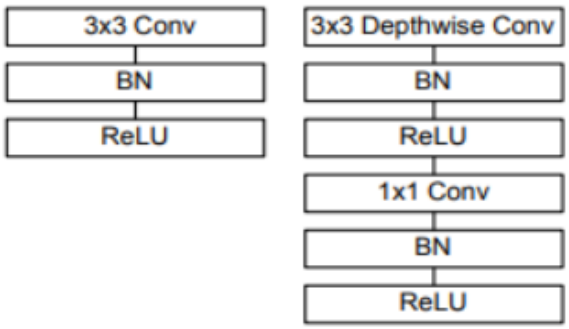

Рисунок 6.5 – Блок звичайної згорткової мережі та блок MobileNet. [6]

MobileNet складається з одного згорткового шару 3x3 та 13 блоків зображених на рисунку 6.5 праворуч. Особливістю даної архітектури є відсутність традиційний max-pooling шарів. Також тут фігурує шар Depthwise Conv, його ідея полягає в тому, щоб розкласти такий шар на depthwise-згортку, тобто на поканальний фільтр та 1x1 згортку. Щоб застосувати такий шар необхідно виконати наступну кількість операцій:

$$(D_k^2 + C_{out})C_{in}\, D_f^2 \ ,$$

де D_k – це розмір фільтру згортки, D_f – це висота і ширина шару, C_{in} – це кількість каналів на вході, C_{out} – кількість каналів на виході. [6]

Відмінністю між MobileNetV2 та MobileNet є останній шар у блоці, який називається bottleneck layer. Цей шар є згорткою 1x1 з лінійною функцією активації, який зменшує кількість каналів. На вході цей шар приймає тензор, який має розмірність:

$$\frac{D_f}{s} * \frac{D_f}{s} * C_in t \ ,$$

де s – шаг згортки, t – гіперпараметр, який називається розширенням. А на виході повертає тензор розмірності:

$$\frac{D_f}{s} * \frac{D_f}{s} * C_{out}$$

[6].

Після цього йде пулінг-шар розміру 7x7, який замінює кожну матрицю такого розміру на середнє значення у ній. Далі шар Flatten, у ньому відбувається вирівнювання, тобто ми переводимо дані з матричного вигляду у вектор і підготовлюємо до звичайного повнозв’язного шару. Наступний

шар містить 128 нейронів, функцією активації є relu, кількість параметрів – 163968, усі параметри підлягають тренуванню. Після цього йде шар відсіву, який використовуються для боротьби з перенавчанням, ймовірність відсіву рівна 0,5. Останній шар моделі містить 2 нейрони, функція активації – softmax, кількість тренувальних параметрів рівна 258.

На початку ваги MobileNetV2 ініціалізуються на основі ImageNet. Проект ImageNet – це велика візуальна база даних, яка використовується для дослідження програмного забезпечення та для розпізнавання візуальних об'єктів. [12]

Функцією втрат обрана categorical crossentropy, як і у випадку першої моделі. Оптимізатор – Adam, з фіксованим кроком при навчанні, рівним 0,001. Він увесь процес навчання залишається незмінним, адже це майже ніяк не впливає на остаточний результат. Метрика – accuracy, для того, щоб можна було порівняти результати у першому і другому підході. Розмір міні пакету також 32. Навчання відбувається 12 етапів. Це число обрано саме таким чином, аби можна було пересвідчитися, що сенсу продовжувати навчання немає.

Графік зміни точності на валідаційному та навчальному наборі даних, в залежності від епохи (червоний – навчальний, блакитний - валідаційний) можна побачити на рисунку 6.6.

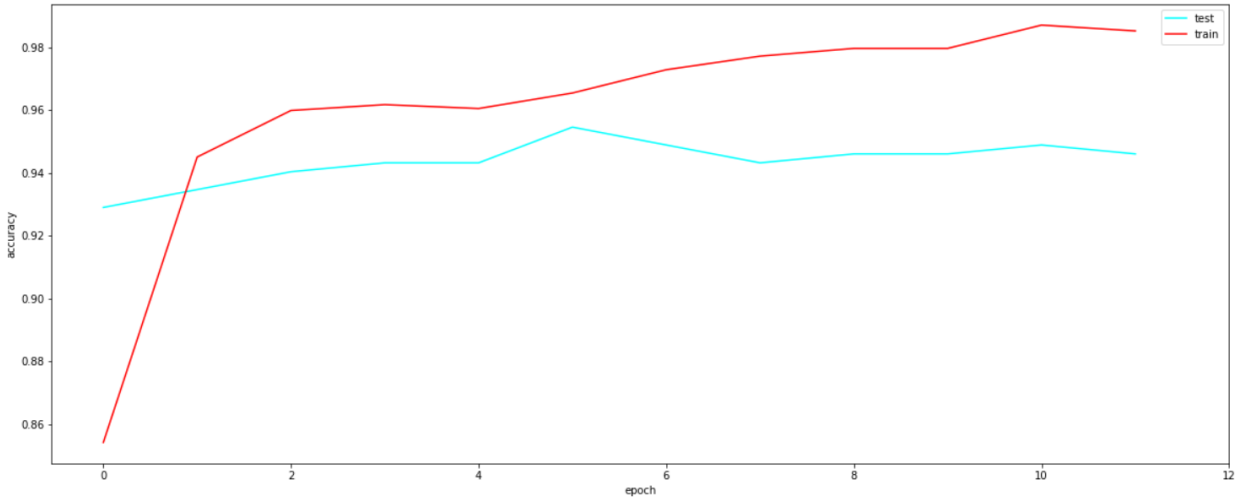

Рисунок 6.6 – графік зміни точності другої моделі.

Як бачимо, точність на тренувальній вибірці невпинно зростає, але в той же час на валідації є майже стаціонарною, тобто якщо продовжувати навчання, то можуть виникнути проблеми з перенавчанням.

Графік зміни функції втрат в залежності від епохи (блакитний – валідація, червоний - тренування) можна побачити на рисунку 6.7.

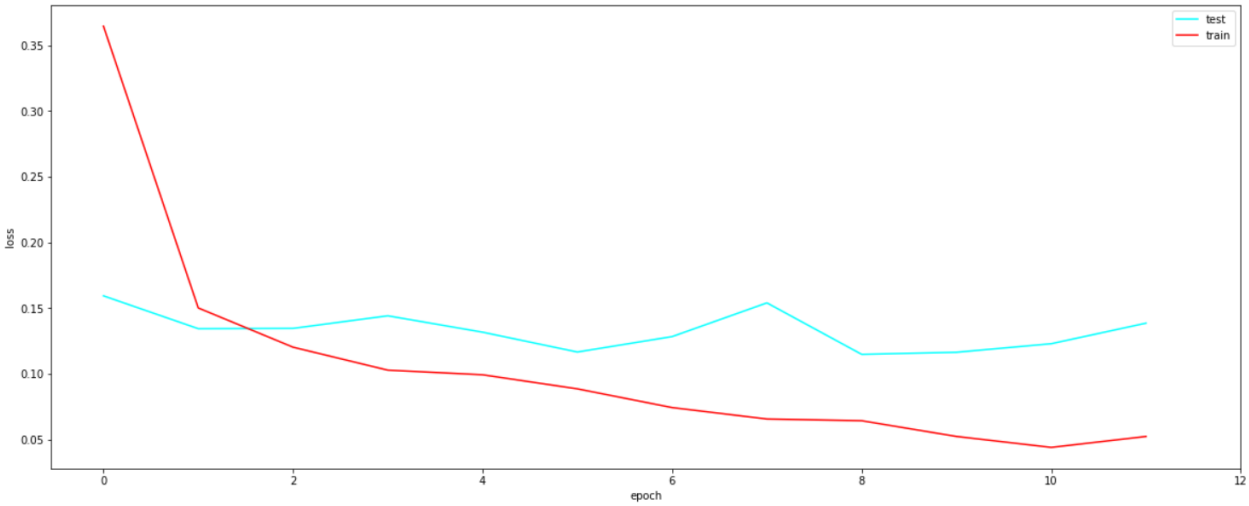

Рисунок 6.7- графік зміни функції втрат другої моделі.

Точність на тестових даних – 0,9576, значення функції втрат – 0,1013. Ця модель показує кращі результати, хоча і тренувальних параметрів використовується набагато менше. Це і є головною перевагою згорткових мереж і другого підходу зокрема.

ВИСНОВКИ

В даній роботі були описані два підходи до розв’язання задачі класифікації. Основним завданням автора було донести інтуїтивне розуміння того, як відбувається весь процес навчання нейронної мережі та методи вирішення проблем, які виникають в процесі навчання мереж. В деяких частинах роботи акцент також робився на математичній складовій.

Було розібрано літературу, необхідну для написання програми. Після чого автором було самостійно написано код програми з описом основних кроків та прийомів у ньому. Обидва підходи показали доволі гарний результат, точність класифікації понад 90% на тестових даних. Це означає, що ці моделі можуть мати практичне застосування. Поза сумнівом, практичне застосування подібних мереж не означає, що ці моделі можуть замінити досвідченого лікаря. Проте вони можуть бути гарними помічниками-порадниками лікарю, наприклад, при наявності великого потоку пацієнтів.

Дійсно, нейронні мережі дуже якісно справляються з подібними задачами, на відміну від традиційних методів або методів, які були відомі кілька десятків років тому. Але не слід вважати нейронні мережі “універсальною зброєю”. Доволі часто такий підхід є занадто “важким” та погано інтерпретується. Найголовніше розуміти з якими даними пов’язано задача і від них відштовхуватися.

Повнозв’язна мережа містить 15492354 параметрів для навчання, в той час як згорткова мережа лише 164226. Цей факт демонструє, наскільки ефективнішим є використання саме згортокової мережі у випадку задачі класифікації зображень. Окрім цього згорткова модель показала кращу точність і на тренувальних, і на валідаційних, і на тестових даних, причому за меншу кількість етапів. Але не можна сказати, що повнозв’язні мережі це те, від чого немає жодної користі і про що можна забути. Адже навіть у згортковій мережі, описаній у програмі, “класифікатором”, тобто одним з останніх шарів, був повнозв’язний шар. Окрім того, повнозв’язні мережі

мають важливий історичний зміст та є гарним першим кроком для розуміння сучасних архітектур нейронних мереж.

Отже, основні завдання які ставилися при написанні роботи досягнені.

В якості подальшої роботи можна було б побудувати не бінарний класифікатор, а класифікатор з більшою кількістю класів, який би дозволив ставити більш точний діагноз за результатом рентгенодіагностики. Для цього потрібна вибірка з відповідними мітками великого об'єму, достатнього для тренування нейронної мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://www.deeplearningbook.org/contents/intro.html
2. https://www.deeplearningbook.org/contents/regularization.html
3. https://www.deeplearningbook.org/contents/optimization.html
4. https://en.wikipedia.org/wiki/Rectifier_(neural_networks)
5. https://en.wikipedia.org/wiki/Softmax_function
6. https://habr.com/ru/post/352804
7. https://en.wikipedia.org/wiki/Pneumonia
8. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf
9. https://en.wikipedia.org/wiki/Sigmoid_function
10. Bishop, Christopher M. (2006). Pattern Recognition and Machine Learning. Springer.
11. https://en.wikipedia.org/wiki/Luce%27s_choice_axiom
12. https://en.wikipedia.org/wiki/ImageNet

Додаток А

```
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from google.colab import drive
drive.mount('/gdrive')
import os
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

fold = r"/gdrive/My Drive/data/diplom.test/NORMAL"
d = []
labels = []
for img in os.listdir(fold):
  image = load_img("/gdrive/My Drive/data/diplom.test/NORMAL/{}".format(i
mg), target_size=(100,100))
  image = img_to_array(image)
  d.append(image)
  labels.append(1)
fold_pn = r"/gdrive/My Drive/data/diplom.test/PNEUMONIA"
for img in os.listdir(fold_pn):
  image = load_img("/gdrive/My Drive/data/diplom.test/PNEUMONIA/{}".form
at(img), target_size=(100, 100))
  image = img_to_array(image)
  d.append(image)
  labels.append(0)
d_train = []
labels_train = []
```

```
fold1_pn = r"/gdrive/My Drive/data/diplom.train/PNEUMONIA"
k = 0
for img in os.listdir(fold1_pn):
  image = load_img("/gdrive/My Drive/data/diplom.train/PNEUMONIA/{}".for
mat(img), target_size=(100,100))
  image = img_to_array(image)
  d_train.append(image)
  labels_train.append(0)
fold1 = r"/gdrive/My Drive/data/diplom.train/NORMAL"
for img in os.listdir(fold1):
  image = load_img("/gdrive/My Drive/data/diplom.train/NORMAL/{}".format(
img), target_size=(100,100))
  image = img_to_array(image)
  d_train.append(image)
  labels_train.append(1)


D = d + d_train
Labels = labels + labels_train
x_train, x_tes, y_train, y_tes = train_test_split(D, Labels, test_size = 0.3, stratify
 = Labels)
x_test, x_validation, y_test, y_validation = train_test_split(x_tes, y_tes, test_size
 = 0.5, stratify = y_tes)
x_test = np.array(x_test).reshape(354, 30000)
x_train = np.array(x_train).reshape(1651, 30000)
x_validation = np.array(x_validation).reshape(354, 30000)
y_train = tf.keras.utils.to_categorical(y_train, 2)
y_test = tf.keras.utils.to_categorical(y_test, 2)
y_validation = tf.keras.utils.to_categorical(y_validation, 2)
def lr_scheduler(epoch, lr):
    if 9 < epoch < 16:
```

```
        lr = 0.0001
    elif 15 < epoch < 23:
      lr = 0.00001
    return lr
first_way = "/gdrive/My Drive/data/" + "first_weights" + ".h5"
callbacks = [tf.keras.callbacks.LearningRateScheduler(lr_scheduler, verbose=1),

             tf.keras.callbacks.ModelCheckpoint(filepath=first_way, save_weights_o
nly=True, mode = "min", monitor ="val_loss", save_best_only=True)]
l2_reg = tf.keras.regularizers.l2(l2=0.01)
def first_model():
  inputs = tf.keras.Input(shape=(30000,))
  x = tf.keras.layers.Dense(512, activation='relu')(inputs)
  x = tf.keras.layers.Dropout(0.2)(x)
  x = tf.keras.layers.Dense(256, activation='relu', kernel_regularizer = l2_reg)(x)
  outputs = tf.keras.layers.Dense(2, activation='softmax')(x)
  model = tf.keras.Model(inputs=inputs, outputs=outputs)
  return model
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.CategoricalCrossentropy()
model = first_model()
model.compile(optimizer=optimizer, loss=loss_object, metrics=[tf.keras.metrics
.CategoricalAccuracy()])
model.summary()
history = model.fit(x_train, y_train, batch_size=32,
    steps_per_epoch = len(x_train) // 32,
    epochs = 22,
    validation_data=(x_test, y_test),
    validation_steps = len(x_test) // 32,
    callbacks = callbacks)
```

```
plt.rcParams['figure.figsize'] = [20,8]
plt.plot(history.history["val_categorical_accuracy"],color = "cyan")
plt.plot(history.history["categorical_accuracy"],color = "red")
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.legend(["test", "train"])
plt.xticks(range(0,23,2))

plt.plot(history.history["val_loss"],color = "cyan")
plt.plot(history.history["loss"],color = "red")
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend(["test", "train"])
plt.xticks(range(0,23,2))

model.evaluate(x_validation, y_validation)
```

Додаток Б

```
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from google.colab import drive
drive.mount('/gdrive')
import os
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

from tensorflow.keras.applications import MobileNetV2
d = []
labels = []
for img in os.listdir(fold):
  image = load_img("/gdrive/My Drive/data/diplom.test/NORMAL/{}".format(i
mg)) target_size=(224,224))
  image = img_to_array(image)
  image = preprocess_input(image)
  d.append(image)
  labels.append(1)
fold_pn = r"/gdrive/My Drive/data/diplom.test/PNEUMONIA"
for img in os.listdir(fold_pn):
  image = load_img("/gdrive/My Drive/data/diplom.test/PNEUMONIA/{}".form
at(img), target_size=(224, 224))
  image = img_to_array(image)
  image = preprocess_input(image)
  d.append(image)
  labels.append(0)
```

```
d_train = []
labels_train = []
fold1_pn = r"/gdrive/My Drive/data/diplom.train/PNEUMONIA"
k = 0
for img in os.listdir(fold1_pn):
  image = load_img("/gdrive/My Drive/data/diplom.train/PNEUMONIA/{}".for
mat(img), target_size=(224,224))
  image = img_to_array(image)
  image = preprocess_input(image)
  d_train.append(image)
  labels_train.append(0)
fold1 = r"/gdrive/My Drive/data/diplom.train/NORMAL"
for img in os.listdir(fold1):
  image = load_img("/gdrive/My Drive/data/diplom.train/NORMAL/{}".format(
img), target_size=(224,224))
  image = img_to_array(image)
  image = preprocess_input(image)
  d_train.append(image)
  labels_train.append(1)
D = d + d_train
Labels = labels + labels_train
x_train, x_tes, y_train, y_tes = train_test_split(D, Labels, test_size = 0.3, stratify
 = Labels)
x_test, x_validation, y_test, y_validation = train_test_split(x_tes, y_tes, test_size
 = 0.5, stratify = y_tes)
x_train = np.array(x_train)
x_test = np.array(x_test)
x_validation = np.array(x_validation)
y_train = tf.keras.utils.to_categorical(y_train, 2)
y_test = tf.keras.utils.to_categorical(y_test, 2)
```

```
y_validation = tf.keras.utils.to_categorical(y_validation, 2)
model = MobileNetV2(weights = "imagenet", include_top = False, input_tensor
= tf.keras.layers.Input(shape = (224,224,3)))
h_model = model.output
h_model = tf.keras.layers.AveragePooling2D(pool_size = (7,7))(h_model)
h_model = tf.keras.layers.Flatten(name = "flatten")(h_model)
h_model = tf.keras.layers.Dense(128, activation="relu")(h_model)
h_model = tf.keras.layers.Dropout(0.5)(h_model)
h_model = tf.keras.layers.Dense(2, activation="softmax")(h_model)
main_model = tf.keras.models.Model(inputs = model.input, outputs = h_model)
for layer in model.layers:
  layer.trainable = False
main_model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(),
    metrics=['accuracy']
)
hist = main_model.fit(
    x_train, y_train, batch_size=32,
    steps_per_epoch = len(x_train) // 32,
    epochs=12,
    validation_data=(x_test, y_test),
    validation_steps = len(x_test) // 32
)
plt.rcParams['figure.figsize'] = [20,8]
plt.plot(hist.history["val_accuracy"],color = "cyan")
plt.plot(hist.history["accuracy"],color = "red")
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.legend(["test", "train"])
```

```
plt.xticks(range(0,13,2))
plt.plot(hist.history["val_loss"],color = "cyan")
plt.plot(hist.history["loss"],color = "red")
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend(["test", "train"])
plt.xticks(range(0,13,2))
main_model.evaluate(x_validation, y_validation)
```