КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Машинне розпізнавання хвороб за рентгенівськими знімками грудної клітини

Дипломна робота магістра
студента II курсу ОР «Магістр»
спеціальності «Комп'ютерні системи та мережі»
В'ячеслава КРАВЧЕНКА

_____

Науковий керівник:
к. ф.-м. н., асистент **Андрій КОНОВАЛОВ**

_____

Рецензент:
к. ф.-м. н., доцент кафедри нанофізики
конденсованих середовищ
Інституту високих технологій **Іван ІВАНОВ**

_____

До захисту допускаю: Завідувач кафедрою, к. ф.-м. н.,
доцент **Юрій БОЙКО**

Ухвалено на засіданні кафедри "____" _____ 2022 р., протокол № ___

Київ 2022

Реферат

Робота присвячена побудові класифікаторів рентгенівських знімків грудної клітини для розпізнавання захворювань людини з використанням глибоких нейронних мереж сучасних архітектур. Запропонована методика розпізнавання захворювань грудної клітини на основі моделі згорткової нейронної мережі Xception, яка дозволяє досягнути середніх значень метрик якості AUC ROC 84%, AU PRC 77.6%, F-міри 76%, що перевищує відомі з літератури відповідні значення, отримані на основі ансамблю нейронних мереж.

Дипломна робота магістра складається з 87 сторінок, 12 рисунків, 2 таблиць, 42 посилань на джерела, 2 додатків.

Ключові слова: хвороби органів грудної клітини, рентгенівські знімки, машинне навчання, класифікація, глибокі нейронні мережі, TensorFlow, Keras, Google Colaboratory, Xception, DenseNet, EfficientNet, MobileNet, CCT.

Зміст

Вступ

Згідно даних Всесвітньої організації охорони здоров'я [1] перші 4 та 6-те місце у рейтингу захворювань як причин смерті людей займають хвороби органів грудної клітини: серця та легень. Зокрема в Україні статистика схожа – 1 місце займають серцево-судинні хвороби, а 2 та 4 – хвороби органів дихання [2].

Рентген грудної клітки для діагностики захворювань серця та легень є зазвичай першим дослідженням пацієнта [3], а у випадку, наприклад, пневмонії відіграє вирішальну роль у клінічному догляді [4] та епідеміологічних дослідженнях [5].

Наприклад, для людей похилого віку рівень смертності вищий в 1.9 разів за наявності кардіомегалії [6]. Також вища ймовірність серцевого нападу за наявності цієї хвороби для людей середнього віку, особливо за наявності зайвої ваги [7], що вказує на важливість не лише окремо взятої хвороби, а й комплексного стану здоров'я та зв'язку між хворобами.

Щодо зв'язку між хворобами, то правильне визначення наявності однієї хвороби з певною ймовірністю може вказати на інше захворювання: згідно дослідження [8] у 24% хворих на SARS-CoV-2 (Ковід) було також виявлено ателектаз.

Точне та раннє виявлення хвороби важливе ще й для того, щоб захворювання не перейшло у гостру фазу. Наприклад для легеневого набряку у гострій формі відсоток смертності у лікарні дорівнює 46%, а також має негативні наслідки для здоров'я, що підвищує рівень смертності до 70% протягом 6 років після лікування, відповідно до даних у дослідженні [9].

Статистика невтішна і при виявленні плеврального випоту, коли протягом місяця від виявлення хвороби відсоток смертності дорівнює 15%, а протягом року – 32% згідно роботи [10].

Автоматизована класифікація (за допомогою машинного розпізнавання) має великий потенціал у сфері діагностики хвороб шляхом швидкого аналізу

надзвичайно великої кількісті даних, зокрема зображень, та проведення класифікацій й опрацювання даних, складних для людини (навіть експертів) [11].

Рання діагностика важлива для багатьох захворювань, а медиків-експертів не завжди достатньо там, де вони потрібні, але на допомогу можуть прийти досягнення сучасної науки та техніки, а саме – машинне навчання (ML) та нейронні мережі (NN), які з розвитком потужності комп'ютерних компонентів, зокрема графічних прискорювачів (GPU), які прискорюють процес навчання моделі нейронної мережі при роботі з графічними зображеннями, порівняно із навчанням, використовуючи центральні процесори (CPU).

Тому, зважаючи на наявну проблему високої захворюваності та смертності від хвороб органів грудної клітини, необхідність складного аналізу для діагностування хвороб та наявність програмних і технічних ресурсів для спроби вирішення проблеми та допомоги медикам, було обрано таку тему.

1 Аналітичний огляд літератури

1.1 Набір даних

Набір даних CheXpert (на даний момент найбільший набір даних з рентгенівськими знімками грудної клітини у відкритому доступі), представлений у роботі [12], складається із 224 316 багатоміткових (1 знімок може містити кілька класів) рентгенівських знімків від 65 240 пацієнтів. Зображення поділені на 14 підкласів, їх список з кількістю знімків та відсотком від всієї кількості знімків наведено у таблиці 1.1 з роботи [12].

Таблиця 1.1.Розподіл знімків за класами із вказаною кількістю знімків та відсотком відносно загальної кількості знімків набору даних [12].

Pathology	Positive (%)	Uncertain (%)	Negative (%)
No Finding	16627 (8.86)	0 (0.0)	171014 (91.14)
Enlarged Cardiom.	9020 (4.81)	10148 (5.41)	168473 (89.78)
Cardiomegaly	23002 (12.26)	6597 (3.52)	158042 (84.23)
Lung Lesion	6856 (3.65)	1071 (0.57)	179714 (95.78)
Lung Opacity	92669 (49.39)	4341 (2.31)	90631 (48.3)
Edema	48905 (26.06)	11571 (6.17)	127165 (67.77)
Consolidation	12730 (6.78)	23976 (12.78)	150935 (80.44)
Pneumonia	4576 (2.44)	15658 (8.34)	167407 (89.22)
Atelectasis	29333 (15.63)	29377 (15.66)	128931 (68.71)
Pneumothorax	17313 (9.23)	2663 (1.42)	167665 (89.35)
Pleural Effusion	75696 (40.34)	9419 (5.02)	102526 (54.64)
Pleural Other	2441 (1.3)	1771 (0.94)	183429 (97.76)
Fracture	7270 (3.87)	484 (0.26)	179887 (95.87)
Support Devices	105831 (56.4)	898 (0.48)	80912 (43.12)

Список класів:

1. No Finding – відсутні хвороби.
2. Enlarged Cardiomediastinum – збільшення кардіосередостіння.
3. Cardiomegaly (кардіомегалія, іноді мегакардія або мегалокардія) – це захворювання, яке характеризується станом серця, коли воно та/або його клапани збільшені.
4. Lung Lesion – ураження легень.

5. Lung Opacity – помутніння легень.

6. Edema – набряк у легенях.

7. Consolidation – ущільнення легень.

8. Pneumonia – пневмонія.

9. Atelectasis (ателектаз) легені – патологічний стан, при якому внаслідок закриття просвіту бронхів та наступного розсмоктування повітря нижче місця закриття відбувається спадання легеневої тканини, частіше частки або сегменту легені та її ущільнення.

10. Pneumothorax (пневмоторакс)– скупчення газу (внаслідок розриву плеври).

11. Pleural Effusion (плевральний випіт) – виверження лімфи у плеврі.

12. Pleural Other – інші проблеми з плеврою.

13. Fracture – перелом.

14. Support Devices – наявність сторонніх пристроїв (наприклад катетер).

Окрім збору значної кількості знімків, увага приділена методології процесу позначення знімків. Для цього авторами набору даних було розроблено програмний комплекс для збору інформації про знімки від спеціалістів та подальшому автоматичному позначенні знімків на відповідні вказані класи.

У таблиці 1 є 3 колонки з підписами *positive* (з англ. позитивний)– це коли у тексті, наданому спеціалістами, точно вказана наявність певного класу, *negative* (з англ. негативний) – коли явно вказано відсутність певного класу та *uncertain* (з англ. непевний) – ця мітка надається знімку, якщо про вказаний клас була згадка у тексті, але не можна чітко сказати про наявність або відсутність певного класу на знімку. Якщо у комірці з відміткою класу відсутня будь-яка позначка, значить про цей клас не було згадки у тексті.

Для зручнішої роботи зі знімками, їх мітки (класи) та інші дані містяться у csv файлах, які йдуть разом з набором даних.

Варто зазначити, що окрім відміток про класи (наявність, відсутність або невизначеність наявності хвороби), у таблиці з даними також міститься додаткова інформація, така-як: стать пацієнта, його вік та сторона зйомки рентгенограми, що можна використати для додаткового аналізу та навчання моделі нейронної мережі.

Важлива відмінність від попереднього найбільшого набору даних рентгенівських знімків грудної клітини з роботи [13] полягає у тому, що цей набір даних містить рентгенівські знімки не лише у фронтальній проекції (див рис. 1.1), а й у бічній (14.5% знімків навчального набору даних) (див рис. 1.2), які необхідні до 15% випадків для точного діагнозу, згідно даних у роботі [14].

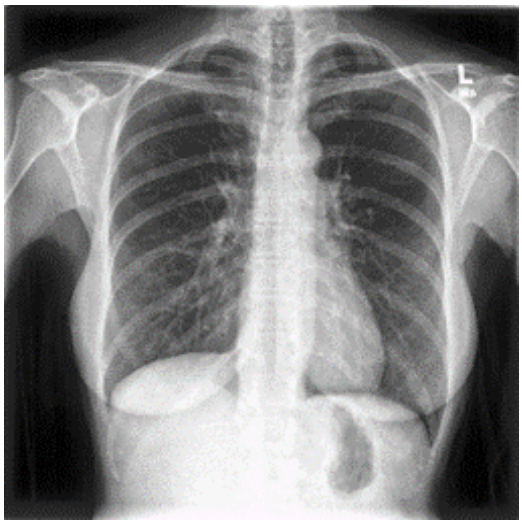

Рисунок 1.1 Фронтальний рентгенівський знімок з набору даних.

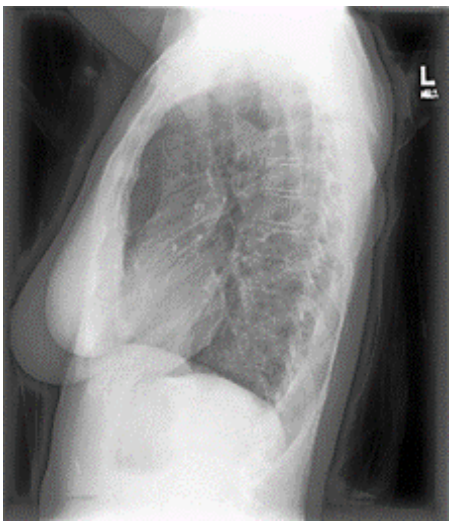

Рисунок 1.2 Бічний рентгенівський знімок з набору даних.

Середній розмір знімків у наборі даних – 2828x2320 пікселів, розмір повного набору даних – 439 GB, але автори також надають оригінальний набір даних, але зі зменшеною роздільною здатністю до 386x320 пікселів у середньому. Зменшений набір даних займає лише 11 GB.

Дані поділено авторами на навчальну та валідаційну (перевірочну) вибірку, а тестова вибірка недоступна публічно, але для такої перевірки можна

завантажити свою роботу та позмагатися з іншими на сторінці проекту, який представляє як сам набір даних, так і змагання [15].

На рис. 1.3 зображені приклади рентгенівських знімків грудної клітини з деякими хворобами органів грудної клітини. Такий набір даних з понад 100 тисяч позначених знімків від понад 32 тисяч пацієнтів уперше був представлений у наборі даних ChestX-ray8 [13].

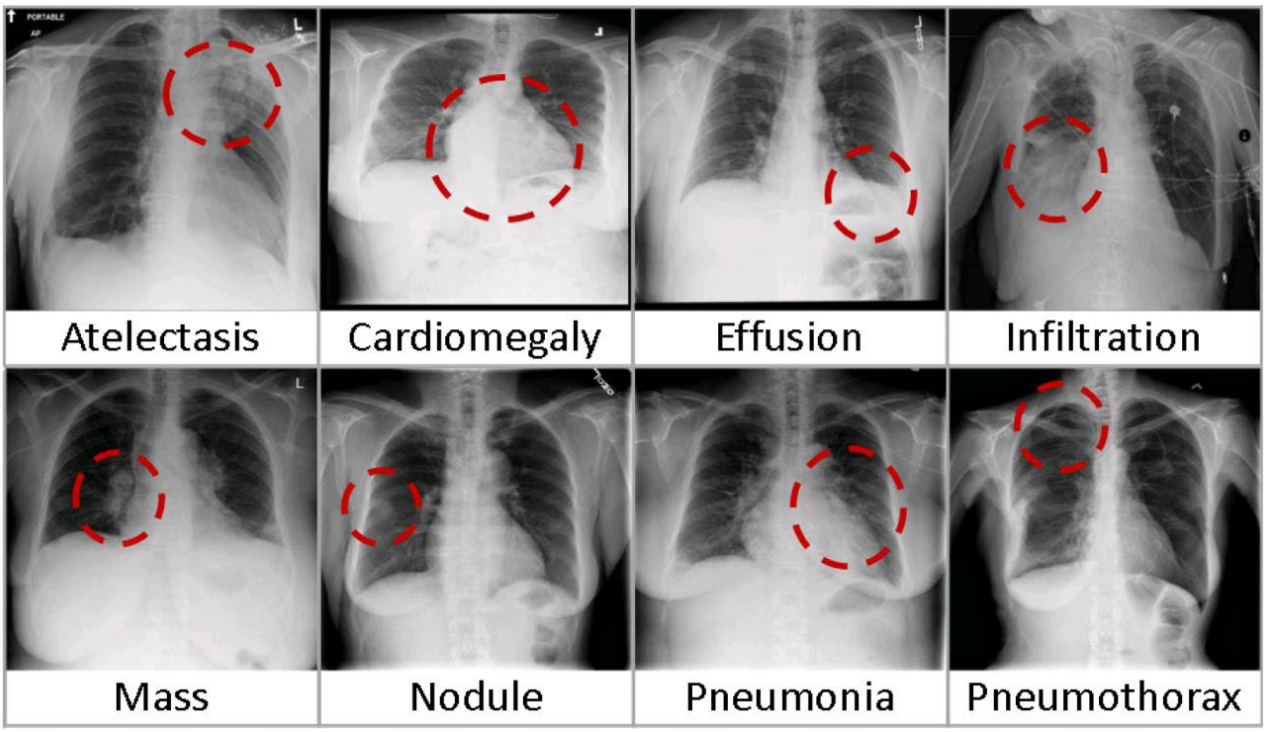

Рис. 1.3 Хвороби органів грудної клітини з ChestX-ray8 [13].

У наборі даних представлені такі хвороби, як-от:

1. Ателекта́з (англ. atelectasis) легені – спадання легеневої тканини та її ущільнення.
2. Кардіомегалія (іноді мегакардія або мегалокардія) (з англ. cardiomegaly) – збільшений розмір серця та/або його клапанів.
3. Перикардіальний випіт (виверження) (з англ. pulmonary effusion) – виверження лімфи у грудній клітині.
4. Інфільтрація (з англ. infiltration) легеневої тканини.
5. Утворення легеневої маси (наприклад рак або ліпоїдна пневмонія).

6. Ущільнення легень, добро- або злоякісні утворення на легенях (з англ. nodule).
7. Пневмонія.
8. Пневмоторакс (з англ. pneumothorax) – внаслідок розриву плеври скупчення газу (найчастіше, повітря) у плевральній порожнині з одночасним підвищенням тиску в ній, відбувається поступовий колапс легені.

У 2017 році автори набору даних ChestX-ray8 розширили його, додавши 3 тисячі знімків та 5 хвороб, позначили знімки та додали дані у ту ж роботу [13].Так ChestX-ray8 став ChestX-ray14.

Додані хвороби:

1. Набряк легенів (заповнення рідиною) (з англ. edema).
2. Емфізема легень (з англ. emphysema) – збільшення об'єму альвеол за рахунок руйнування перетинок між ними. Легені збільшуються в об'ємі, не спадаються, стають млявими, дихальні проходи звужуються.
3. Фібро́з (з англ. fibrosis) ущільнення сполучної тканини з появою рубцевих змін у різних органах.
4. Плеврит (потовщення плеври) (з англ. pleural thickening).
5. Легенева грижа (з англ. hernia).

На рисунку 1.3 не так просто помітити різницю між зображеннями, а, відповідно, й хворобами навіть з позначеними місцями осередку хвороби, що значить для кожного захворювання потрібен нетривіальний аналіз [16] під кожен випадок навіть для експертів.

Хоча аналіз рентгенівського знімку, на який здатні моделі штучних нейронних мереж не містить додаткових клінічних ознак пацієнта (наприклад кашель), які будуть у лікаря. Але метод розпізнавання за рентгенівським знімком

лишається ключовим у первинній діагностиці – й правильна класифікація значно пришвидшує визначення як діагнозу так і пріоритету лікування.

Враховуючи більшу кількість знімків, що добре для узагальнюючої здібності моделі нейронної мережі, наявність рентгенівських знімків бічної проекції, а також якісний інструмент виділення міток для знімків, для даної роботи було обрано CheXpert [12] як набір даних для досліджень.

1.2 Нейронні мережі

До сучасного вигляду технологія нейронних мереж (штучних нейронних мереж, штучного інтелекту, машинної класифікації) пройшла довгий шлях з 40-х років минулого століття, та майже до 90-х років головним обмеженням були апаратні ресурси обчислювальної техніки. А з розвитком комп'ютерної техніки, зокрема графічних процесорів, нарешті з'явилася можливість побудови глибинних нейронних мереж для вирішення складних та цікавих задач, як от відбудова зображень та визначення положень обличь (2003 рік) [17]. З розвитком технологій стали можливі й задачі розпізнавання високорівневих понять, наприклад котів, лише завдяки навчанні на немічених зображеннях, взятих з відео хостингу YouTube (2011 рік) [18].

Штучні нейронні мережі (ШНМ) – це спроба програмно реалізувати нейронні структури людського мозку, який містить своєрідні органічні тригери (перемикачі) – нейрони. Тип переданого сигналу може змінюватися у залежності від походження джерела сигналу. У мозку людини нейронна мережа – це величезна система пов'язаних між собою нейронів, у якій сигнал, що проходить через один нейрон, може розповсюдитися на велику частину мережі за потреби. Ймовірність отримання правильного (потрібного) результату за певних вхідних даних зростає під час навчання, яке реалізовано повторною активацією деяких синапсів (нейронних з'єднань), які «зміцнюються» – стають щільнішими при закріпленні асоціації (отриманні підходящого результату).

Архітектура нейронної мережі прямого поширення глибинного навчання з послідовністю згорткових шарів та шарів агрегування (максимізаційного), слідом за якими йдуть декілька частково- або повнозв'язних класифікуючих шарів, з 2011 року є передовою для свого типу [19, 20].
Нейронна мережа з такою архітектурою називається згортковою та була першою, що досягла в певних задачах продуктивності, порівняної з людською [21].

Саме такого типу й використовуються архітектури нейронних мереж у цій роботі та у дослідженнях з підрозділу, окрім однієї, яка поєднує у собі як переваги згорткових мереж, так і механізму уваги [22], який замість повторення сигналів (як працює зміцнення синапсів у людському мозку) шукає важливі елементи відразу на усій послідовності даних.

Відтворення та моделювання нелінійних процесів – завдяки цим здібностям ШНМ виявилися корисними у різних сферах: від керування транспортними засобами, передбаченнями траєкторії, перемоги чемпіонів у іграх, оптимізації фінансових процесів до моделювання природніх та інженерних моделей, розпізнавання образів та об'єктів, медичної діагностики та багато інших.

У контексті теми даної роботи варто зазначити, що було показано, що для діагностики різних хвороб за медичними зображеннями машинне навчання працює на рівні медичних експертів [23]. Програмні продукти починають сертифікуватись для клінічного використання [24]. Машинне навчання, як наріжний камінь сучасної революції штучного інтелекту, може бути ключем до реалізації бачення ШІ в медицині, накресленого кілька десятиліть тому [25].

1.3 Результати сучасних досліджень

У всіх роботах, результати яких будуть представлені для порівняння метрик якості класифікатора, використовується набір даних CheXpert [12].

У роботі [12] від Stanford ML Group, де й представлено набір даних CheXpert, для роботи з нейронною мережею знімки приведено до розміру

320x320 пікселів. У ході дослідження найефективнішою виявилася архітектура DenseNet-121 [26]. Результуючий класифікатор складається з ансамблю 3 моделей обраної архітектури DenseNet-121. У результаті дослідження підходів з обробки міток «невпевненості (uncertain)» класів було виявлено, що, наприклад, підхід *U-Ones* (коли мітки невпевненості вважаються як наявність хвороби) отримує кращі результати для виявлення ателектазу, значення метрики класифікатора AUC ROC (площа, обмежена кривою частки помилкових та позитивних класифікацій) – 85.8% на відміну від підходу U-Zeros (коли мітки невпевненості вважаються як відсутність хвороби), для кого AUC ROC – 81.1% , тому різні підходи можуть з різною точністю виявляти окремі хвороби. А середнє значення (macro-усереднення для цього та наступних випадків) AUC ROC для 5 хвороб, обраних для змагання – **90.6%**. Середнє значення метрики AU PRC (площа, обмежена precision-recall-кривою) – **70.2%.**

У роботі [27] знімки приведено до розміру 256x256 пікселів, а потім виділено грудну клітину з розміром знімку 224x224 пікселів. Проведено підбір гіперпараметрів. Використано техніку регуляризації міток класів для міток невпевненості. Кращі результати отримано при використані ансамблю 6 архітектур нейронних мереж, попередньо натренована на наборі даних ImageNet [28], та отримано середній для усіх класів показник AUC ROC – **93%**.

У роботі [29] обраний розмір знімків – 320x320 пікселів. Використано деякі з методик роботи [27] та власний алгоритм як максимізації метрики AUC ROC, так і оптимізація похибки. Модель являє собою ансамбль 5 архітектур нейронних мереж. Середнє значення AUC ROC – **93,05.**

Робота [30] не використовує знімки, де присутні мітки невизначеності класу. Розмір знімків – 224x224 пікселів. Перевірено 3 підходи до залежностей між мітками класів. Архітектура моделі - DenseNet-121 [26], попередньо натренована на наборі даних ImageNet [28]. Серед досліджених підходів найкращий досягає середнього значення AUC ROC – **81.2%** для усіх класів набору даних та **82.8%** для 5 відібраних для змагання хвороб.

Проаналізувавши сучасні дослідження, було помічено, що лише в роботі [27] було виділено грудну клітину на знімку, але у всіх роботах відсутнє дослідження впливу коректного приведення фото до квадратної форми з виділенням центральної частини, ефективність чого було доведено у попередніх дослідженнях.

Також варто зазначити, що не в усіх роботах було використано сучасні (на час публікації статті) архітектури нейронних мереж. Зокрема важливим дослідженням могло б бути дослідження архітектур різної складності та якість окремо взятої архітектури, на відміну від ансамблів, використаних у деяких роботах, які, хоч і призводять до підвищення точності класифікації, але вимагають величезної кількості ресурсів як часових, так і апаратних.

У жодній з робіт не наведено обґрунтування вибору розміру зображення. Тому важливо дослідити це питання, так як це безумовно впливає на різні аспекти як часу навчання моделі НМ, вимоги до ресурсів обладнання, часу прототипування моделі та іншого.

Враховуючи важливість попередньої обробки даних та максимально можливого використання доступних ознак знімків, які надає набір даних було помічено, що у всіх роботах відсутні дослідження впливу на метрики якості класифікатора як присутності рентгенівських знімків бічної проекції, так і статі пацієнта для розподілу навчання за статтю.

1.4 Мета і задачі дослідження

Таким чином, **метою** дипломної роботи магістра є розробка методики машинного розпізнавання хвороб за рентгенівськими знімками грудної клітини на основі глибоких нейронних мереж сучасних архітектур.

Для досягнення сформульованої мети поставлені такі задачі:

1. Побудова моделей класифікаторів рентгенівських знімків грудної клітини на основі глибоких нейронних мереж сучасних архітектур.
2. Встановлення впливу розміру зображень рентгенівських знімків на час навчання моделей нейронних мереж та на їх основні метрики якості.
3. Визначення метрик якості моделей, навчених на різних підвибірках навчальної вибірки зображень, поділених за ознаками статі пацієнта та типу проекції рентгенівських знімків.

2 Методика дослідження

2.1 Попередня обробка даних

Враховуючи розміри знімків у інших роботах та ресурсоємність повнорозмірного набору даних, для дослідження було обрано оригінальний набір даних зменшеної роздільної здатності, який також надають автори [12]. Максимальна роздільна здатність знімків набору – 386x320 пікселів. Обсяг набору даних – 11 ГБ.

З оригінального набору даних сформовано новий з коректним приведенням знімків до квадратної форми з виділенням центральної частини розміром 320x320 пікселів. Ефективність такого коректного приведення була підтверджена під час виконання попередніх досліджень.

Для перевірки впливу розміру зображень на результати навчання нейронних мереж обрані такі розміри: 100x100, 160x160, 224x224, 320x320 пікселів.

Для оцінювання метрик якості побудованої моделі класифікатора використовується публічно-доступний тестовий (називається валідаційним у оригінальній роботі) набір даних (234 знімки від 200 пацієнтів), який йде разом з оригінальним набором даних та має окремий табличний файл з мітками класів та додатковими ознаками.

Як і в роботах [12, 27, 29, 30], для можливості порівняльного аналізу при побудові моделей класифікаторів використовувались рентгенівські знімки з мітками хоча б однієї з 5 захворювань (ателектаз, кардіомегалія, легеневе ущільнення, легеневий набряк та плевральний випіт). Список хвороб обрано авторами використаного набору даних [12], зважаючи на їх клінічну важливість та розповсюдженість.

Для міток невизначеності обрано підхід *U-Ignore* [12], тобто класи з такою міткою ігноруються. Таким чином навчання проводиться на 223 414 знімках (які перемішуються випадковим чином перед початком навчання), що лише

на 902 зображення менше, чим повний набір даних. Кількість знімків у фронтальній проекції – 191 027 (85.5%), у бічній – 32 387(14.5%).

У якості валідаційної множини (використовується для оцінки метрик якості під час навчання моделі та не бере участь у навчанні та тестуванні) автоматично відбирається з навчальної множини 5% знімків випадковим чином з перемішуванням.

2.2 Середовище виконання

Для проведення досліджень було обрано середовище Google Colaboratory [31] та його PRO версія за платною підпискою з таким характеристикам:

- Потужні апаратні ресурси:
  - CPU: Intel(R) Xeon(R) CPU @ 2.30 ГГц (1 ядро).
  - Оперативна пам’ять: 13 ГБ (25 ГБ у PRO версії).
  - Об’єм жорсткого диску: ≈78 ГБ (≈168 ГБ для PRO).
  - Наразі середовище виділяє один з таких **GPU** у залежності від наявних вільних ресурсів (порядок у зростанні потужності, яка впливає на швидкість навчання моделі): Nvidia Tesla K80 (12 ГБ відеопам’яті, найчастіший варіант для безкоштовної версії), Nvidia Tesla T4 (15 ГБ відеопам’яті надається при доступності в безкоштовній версії та при вичерпанні квоти більш потужної відеокарти для PRO версії), Nvidia Tesla P100 (16 ГБ відеопам’яті, найпотужніша відеокарта сервісу та найчастіше надається для PRO версії). GPU дають найбільший приріст до швидкості навчання моделі на графічних зображеннях порівняно з CPU (різниця у 5–10 разів).
- Можливість безкоштовного використання.
- Зручність використання – запуск коду мовою Python у комірках та зберігання коду в форматі Jupyter Notebook [32].
- Віртуальна машина на операційній системі Ubuntu найновішої стабільної версії, що дає змогу використовувати команди ОС.

- Наперед встановлені основні бібліотеки машинного навчання (наприклад Tensorflow [33] та).
- Хмарність – можливість підключення до віртуальної машини свого Google Drive (наприклад для доступу до набору даних) у 2 кліки, що й використовувалося під час виконання роботи.

Для проведення досліджень також використовувався локальний комп'ютер з такими характеристиками:

- CPU: Intel Core I7-6700HQ.
- Оперативна пам'ять: 16 ГБ DDDR3.
- Об'єм SSD – 465 ГБ, жорсткого диску – 1TB.
- GPU: **Nvidia GeForce 960M** з 4 ГБ GDDR5 відеопам'яті, що помітно менше, ніж у найпростішої відеокарти Google Colab, тому доступний менший розмір пакету (batch size) знімків.

Переваги використання локального комп'ютера, порівняно з Google Colab:

- відпадає необхідність завантаження набору даних на початку кожної сесії,
- не потрібно кожної сесії довстановлювати необхідні бібліотеки,
- можливість використання SSD замість HDD,
- необмежений час сесії та використання ресурсів ПК,
- доступно більше ядер процесору,
- відсутня необхідність постійного підключення до Інтернету.

Варто зазначити, що при використанні будь-якого середовища іноді доводиться стикатися з конфліктами версій різних пакетів. На локальному ПК можна з повним доступом пробувати різні версії пакетів, різні середовища. Зручність наперед встановлених сумісних пакетів у Google Colab допомагає, але можна стикнутися з тим, що деякі пакети чи версії при ручному встановленні не підтримуються або конфліктують, що може призвести до додаткових маніпуляцій з конфігурацією середовища, тому треба бути до цього готовим.

2.3 Архітектури нейронних мереж

Для усіх наведених далі архітектур нейронних мереж змінювався вхідний шар для подання на вхід нейронної мережі знімків потрібного розміру та кольорового формату (градації сірого).

Спільні характеристики навчання для усіх використаних архітектур:

- Випадкова ініціалізація початкових ваг.
- Функція активації нейронів вихідного шару – сигмоїда. Відбувається класифікація багатоміткових даних, тому обрано таку функцію для того, щоб отримати незалежну для кожного класу (хвороби) ймовірність присутності на знімку.
- Початкова швидкість навчання – 0.01.
- Функція помилки – бінарна крос-ентропія.

2.3.1 DenseNet-121

У роботах [12, 27, 30], найкращою архітектурою нейронної мережі визначена DenseNet-121 (2017 рік) [26]. Тому було вирішено і для цієї роботи спробувати дану архітектуру для відтворення результатів та спроби покращення метрик класифікатора.

Ця архітектура демонструє високу точність при невеликій кількості параметрів ≈8 млн. Одна з найлегших сучасних архітектур [34].

2.3.2 Xception

Архітектура Xception (2017 рік) [35] має ≈21 млн параметрів, що у 2.25 разів більше за DenseNet-121, тому цікаво порівняти архітектури різної складності. Також у неї якісне співвідношення розміру до результатів точності порівняно з іншими архітектурами [34].

2.3.3 EfficientNet

Архітектура EfficientNet [36] представлена у 2019 році та її перевагами є: високі результати метрик якості на наборі зображень ImageNet [28], найшвидша класифікація даних при меншому розмірі моделі. А також головна зручність –

можливість вибору складності архітектури (8 варіантів від ≈5 млн до ≈67 млн параметрів) простим завантаженням іншої моделі без необхідності змінювати увесь інший програмний код.

У 2021 році представлено другу версію цієї архітектури EfficientNetV2 [37]. Приріст швидкості навчання, зменшення розміру моделі зі збереженням якості метрик. 7 варіантів моделей від ≈7 млн до ≈119 млн параметрів. Це одна з найсучасніших опублікованих та доступних архітектур нейронних мереж.

2.3.5 MobileNetV2

MobileNetV2 [38] – сучасна архітектура, представлена у 2018 році. Найлегша за кількістю параметрів (лише 3.5 млн) архітектура серед доступних у бібліотеці глибинного навчання Keras [34]. Простота поєднана з високими показниками метрик якості класифікатора.

Також однією з причин вибору стало те, що жодна з порівнюваних робіт не використовувала цю архітектуру.

2.3.6 CCT

Compact Convolutional Transformer (CCT) [39] – сучасна архітектура, представлена у 2021 році. Поєднує краще з двох світів для класифікації зображень – згорткову нейронну мережу з механізмом уваги Transformer [22].

Зручність використання даної архітектури полягає у можливості гнучкого налаштування як гіперпараметрів, так й складових структури архітектури. Наприклад можна змінювати кількість шарів трансформерів.

Для цієї архітектури використання такі значення гіперпараметрів:

- `positional_emb = True,`
- `conv_layers = len(output_channels),`
- `output_channels = [32, 64],`
- `projection_dim = 64,`
- `num_heads = 1,`

- `transformer_units = [projection_dim, ],`
- `transformer_layers = 1,`
- `stochastic_depth_rate = 0.1,`
- `learning_rate = 0.01,`
- `weight_decay = 0.0001.`

У другому наборі гіперпараметрів усі значення вище повторюються, окрім:

- `num_heads = 2,`
- `transformer_layers = 2`

Хоча така архітектура й містить лише 200 тисяч параметрів при таких гіперпараметрах, але через особливість роботи механізму уваги трансформера, такі моделі потребують багато відеопам'яті – 16 ГБ при розміру знімку 160x160 пікселів та розміру порції (batch size) – 128.

2.4 Навчання моделі нейронної мережі

Програмний код для усієї необхідної роботи розміщено у Додатку А CNN для моделей нейронних мереж з використанням архітектур, наведених у пунктах 2.3.1-2.3.5, а для архітектури CCT – Додаток Б CCT.

Робота з архітектурами нейронних мереж, навчанням та тестуванням моделей виконувалася з використанням бібліотеки-надбудови Keras [33, 40]. Надбудова – API (Application Programming Interface - це набір готових класів, процедур, функцій, та іншого, що надаються додатком (бібліотекою, сервісом) для використання у зовнішніх програмних продуктах) для зручнішого використання бібліотеки глибинного машинного навчання TensorFlow 2.0 [33].

Навчання проводилися на знімках таких розмірів: 100x100, 160x160, 224x224 та 320x320 пікселів. Знімки у наборі даних у градаціях сірого – у такому ж кольоровому форматі вони й подавалися на вхід НМ.

Також було використано функцію зворотного виклику ReduceLROnPlateau водночас із критерієм зупинки EarlyStopping (Метод ранньої зупинки). Про них детальніше у розділі 2.4.1 Додаткові параметри навчання.

Проведено навчання для розпізнавання 5 хвороб (Atelectasis, Cardiomegaly, Consolidation, Edema, Pleural Effusion).

Для дослідження впливу на метрики якості класифікатора навчання та тестування проводилися на різних підвибірках навчальної і тестової вибірок, поділених за ознаками статі пацієнта та типу проекції рентгенівських знімків.

2.4.1 Додаткові параметри навчання

Принцип роботи стратегії зміни швидкості навчання ReduceLROnPlateau (Reduce learning rate on plateau) – при зупинці зменшення помилки на валідаційному наборі даних коефіцієнт швидкості навчання зменшується. Принцип роботи стратегії продемонстрований на рис. 2.1 (Ви демонструєте дуже спрощений тезисний підхід до написання розділу 2):

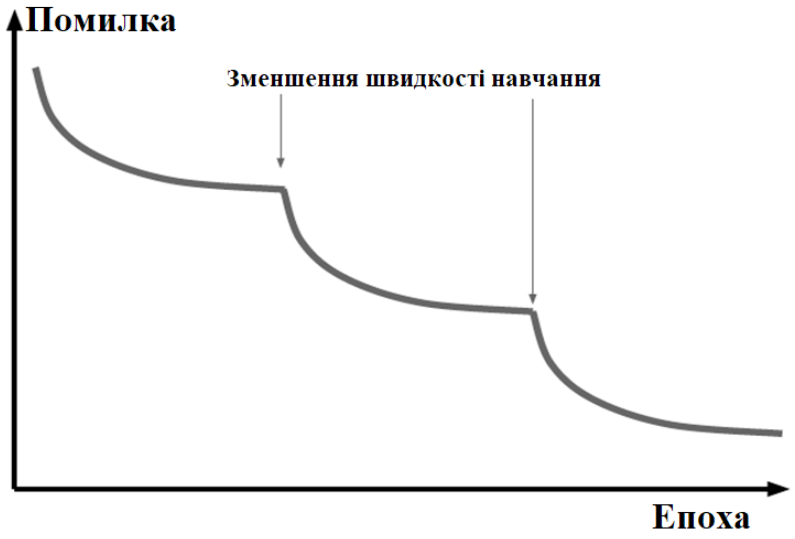

Рис. 2.1 Принцип роботи ReduceLROnPlateau.

При роботі з Keras було використано такий код для задання даної функції зворотного виклику:

```
reduce_learning_rate = ReduceLROnPlateau
(monitor='val_loss', factor=0.2, patience=5, cooldown=2,
min_lr=0.00001, verbose=1)
```

Також були проведенні навчання моделей зі значенням параметру `patience – 1`0 для дослідження впливу на якість навчання.

Використані такі параметри стратегії:

• `monitor = 'val_loss'` – означає слідкувати за покращеннями (зменшенням) загальної помилки нейронної мережі при перевірці валідаційної множини даних під час навчаня.

• `factor = 0.2` – це фактор зменшення темпу навчання мережі, тобто при відсутності покращень у характеристиці, за якою ведеться спостереження, протягом епох, кількість яких вказано у наступному параметрі patience (витримка). Значення фактору 0.2 означає, що темп навчання зменшиться у 5 разів.

• `patience = 10 # (або 20 у залежності від архітектури)` – (витримка) - це кількість епох, протягом яких не буде змінюватися темп навчання, якщо немає покращень у результатах.

• `cooldown = 2` – (втихомірювання) – кількість епох до повернення до нормального режиму роботи навчання після зменшення темпу навчання.

• `min_lr = 0.00001`- мінімальний темп навчання після якого вже не можна зменшувати темп.

• `verbose = 1` – середній рівень виводу інформації про те, що відбувається.

Принцип роботи методу ранньої зупинки (EarlyStopping) зупинки полягає у вчасній зупинці навчання, щоб не відбулося перенавчання, коли на навчальному наборі буде високе значення метрики якості класифікатор, а на валідаційному – гірше.

Місце необхідної ранньої зупинки продемонстровано на наступному рисунку 2.2:

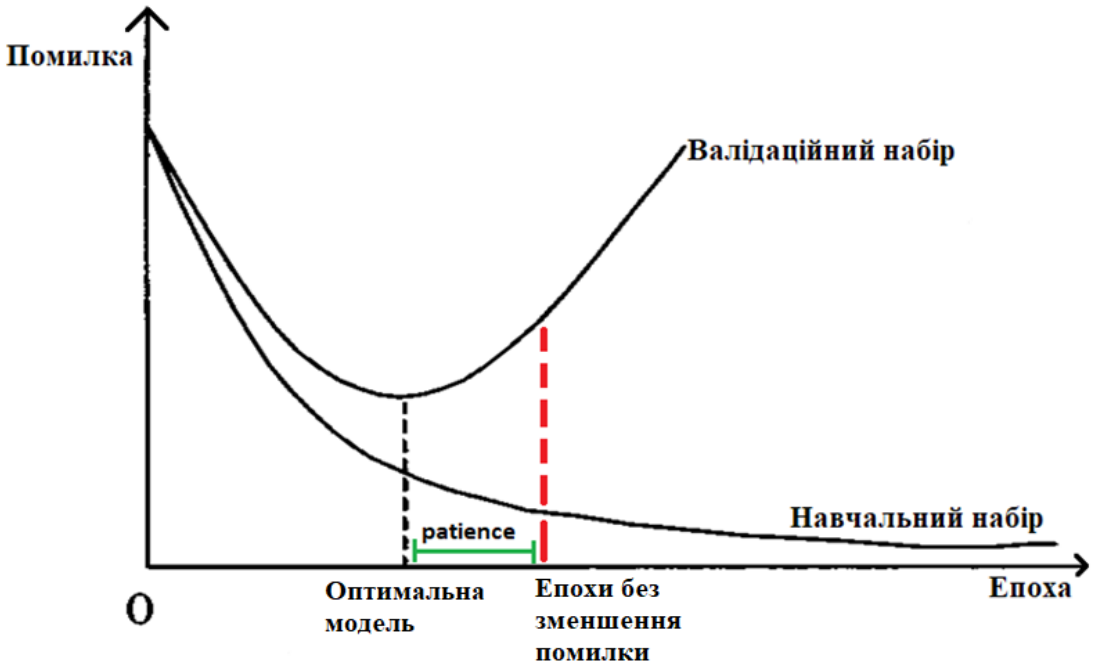

Рис. 2.2 Принцип роботи EarlyStopping.

У цій роботі було використано такий код для задання критерія зупинки:

```
early_Stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)
```

Параметри EarlyStopping:

• `monitor='val_loss'` – тут означає спостереженням за зменшенням втрат на перевірочній множині.

• `patience=10 (або 20 у залежності від архітектури)` – витримувати (чекати) 10 (20) епох без покращень до зупинки.

• `verbose=1` – як і в описі параметрів попереднього критерія зупинки.

• `restore_best_weights=True` – означає, що після завершення навчання будуть використані ваги не з останньої епохи, а найкращі, тобто з кращої за результатами епохи.

2.5 Метрики якості класифікатора

Precision (точність) – відсоток правильних класифікацій обраного класу із множини усіх відповідей, які позначені класифікатором як обраний клас, $precision = \frac{TP}{TP+FP}$.

Де TP (true positive, істинно позитивний) – правильно класифіковані об'єкти (зображення) обраного класу (хвороби). FP (false positive, хибно позитивний) – кількість хибно визначених об'єктів обраного класу, коли класифікатор вказує на наявність хвороби там, де її нема.

Recall (sensivity, повнота, чутливість) – відсоток правильних класифікацій обраного класу із множини усіх об'єктів обраного класу, $recall = \frac{TP}{TP+FN}$.

Традиційна формула F-score (F-міра) – середнє гармонійне: $F_1 = 2 * \frac{precision*recall}{precision+recall}$.

AU PRC (area under precision-recall curve – площа, обмежена PR-кривою) – метрика якості оцінки належності об'єкту до позитивного класу, яку повертає алгоритм. Наглядність та зручність даної метрики полягає в тому, що при візуалізації кривої видно значення точності та чутливості класифікатора для певного класу від їх мінімальних до максимальних значень, тому керуючи порогом класифікації можна максимізувати точність (підтвердити наявність хвороби, менше невірних спрацювань) або чутливість (не пропустити наявність хвороби, менше невірних пропусків) у залежності від потреби медика у випадку діагностики хвороб.

Поріг класифікації – поріг ймовірності, що використовується для визначення класу (для віднесення зображення до обраного класу). Наприклад, якщо класифікатор надав для знімку ймовірність належності до класу 30%, а поріг класифікації – 50%, то вважатиметься, що цей клас відсутній на знімку.

AUC скорочено від AUC ROC (area under curve of receiver operating characteristic) - площа, обмежена ROC-кривою (робоча характеристика

приймача, крива помилок). ROC-крива показує значення TPR, FPR (True/False Positive Rate) для різних порогів класифікації. Де $TPR = \frac{TP}{TP+FN}$, тобто те саме, що й чутливість (recall). $FPR = \frac{FP}{FP+TN}$ – доля помилкових спрацювань на негативних об’єктах. TN (true negative, істинно негативний) – правильно класифіковані об’єкти негативного (відсутність хвороби) класу. AUC ROC не залежить від балансу класів.

Чим більше значення площі під кривою (і для AU PRC, і AUC ROC), тим якісніше працює класифікатор.

Для більш детальної інформації щодо метрик оцінки якості класифікатора можна ознайомитися з інформацією, наведеною у бібліотеці Scikit-learn [41]. З цієї бібліотеки було використано функції для отримання вище описаних метрик якості побудованих у роботі класифікаторів.

3 Результати досліджень та їх обговорення

3.1 Залежності часу навчання та метрик якості від розміру зображення

На рис 3.1 продемонстровано залежності часу навчання від розміру зображення для різних архітектур при навчанні на рентгенівських знімках фронтальної проекції пацієнтів чоловічої статі. Ця підвибірка використовується і для тестування для демонстрації результатів на рис. 3.2. Таке спрощення щодо об'єму навчальних даних використано для пришвидшення ресурсномісткого дослідження.

Для архітектур Xception та CCT на рис. 3.3 дані наведено при використанні GPU Nvidia Tesla P100, для DenseNet-121 – Nvidia GeForce GTX 960M. Розмір порції знімків (batch size) було підібрано для використання максимально доступного об'єму відеопам'яті.

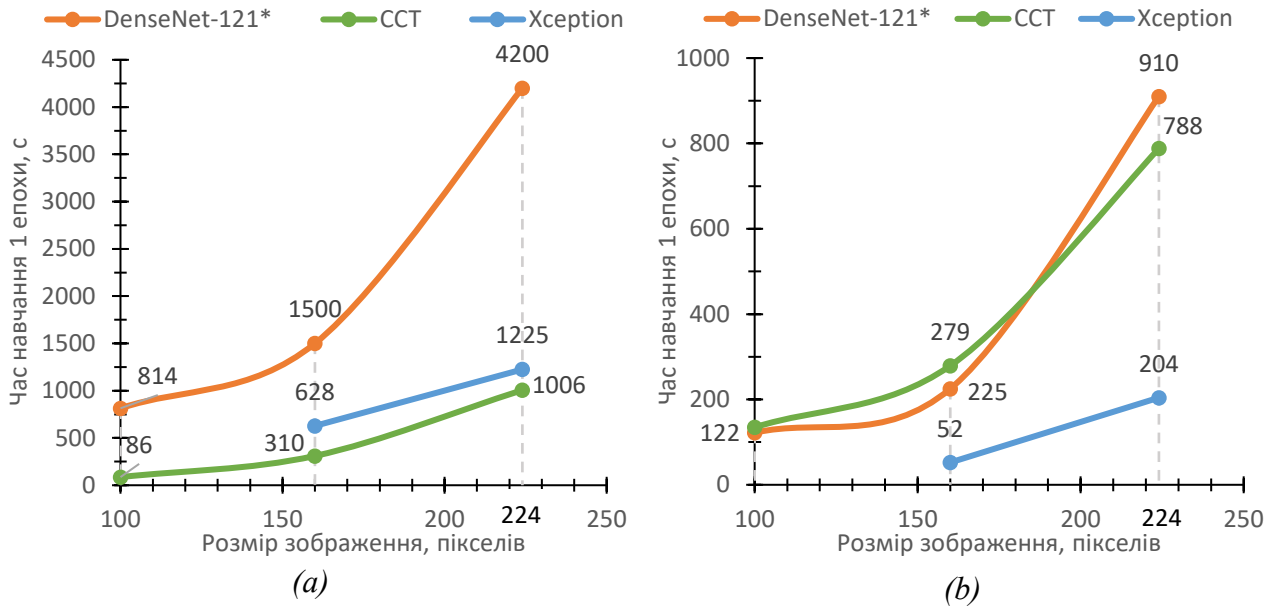

Рис. 3.3 Залежності часу навчання 1 епохи (a) та загального часу навчання (b) від розміру зображення для різних архітектур нейронних мереж на 100% навчальної підвибірки

На рис. 3.4 зображено залежність метрики AUC ROC від розміру зображення відповідно до архітектури нейронної мережі при чоловічих фронтальних знімках. З графіку помітно, що навчання на знімках розміром понад 160x160 пікселів не призводить до помітного покращення точності класифікації, враховуючи на скільки зростає час навчання як зображено на рис. 3.1 та зростають вимоги до ресурсів.

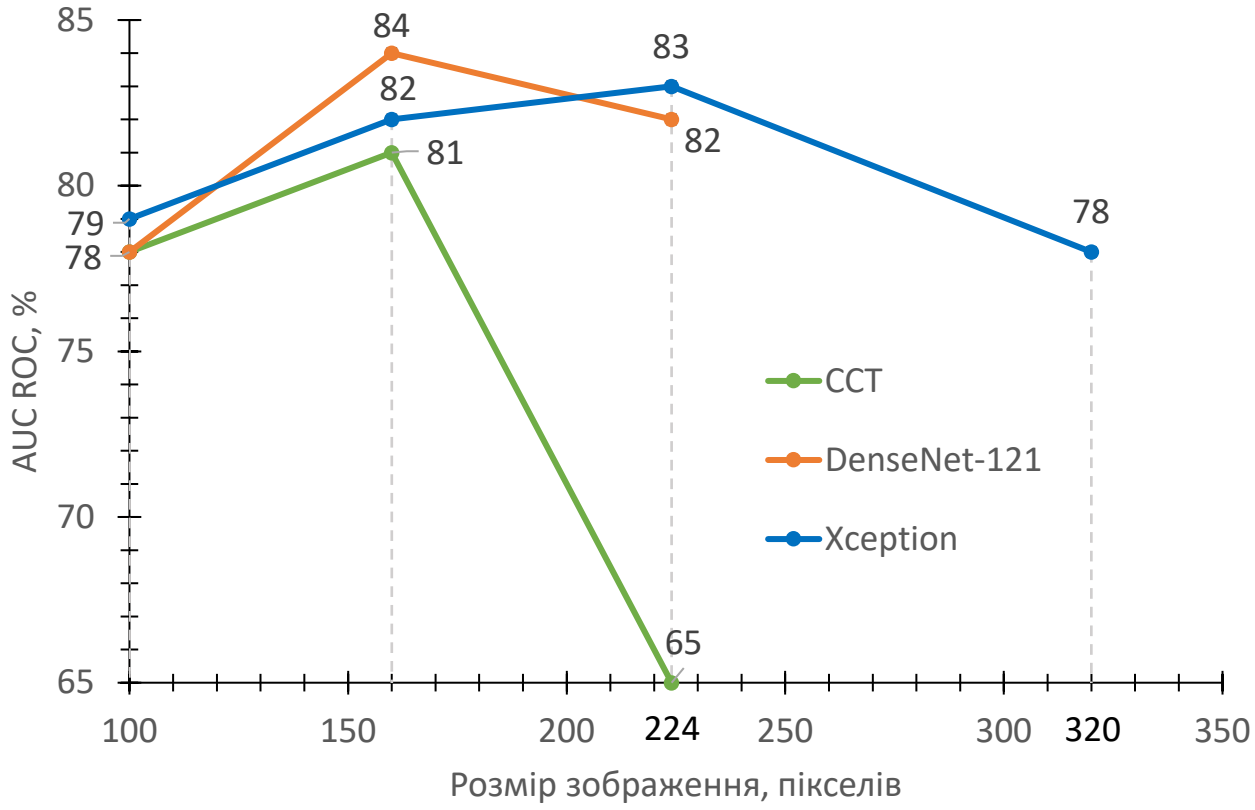

Рис. 3.4 Залежності метрики AUC ROC від розміру зображення та архітектури нейронної мережі на чоловічих фронтальних знімках.

3.2 Вплив навчання на різних підвибірках на метрики якості

На рис. 3.5 та 3.6 продемонстровано метрики якості AUC ROC та AU PRC для моделей, побудованих на архітектурі Xception. На рис 3.5 результати моделей навчених і оцінених на різних підвибірках навчальної і тестової вибірок, поділених за ознаками статі пацієнта та типу проекції рентгенівських знімків. А на рис. 3.6 – навчених на усіх знімках, а оцінених на різних підвибірках.

Згідно рисунку 3.5 встановлено, що для підвищення точності класифікації фронтальних знімків у якості навчальної підвибірки краще використовувати рентгенівські знімки лише у фронтальній проекції. Навчання додатково на знімках у бічній проекції не дає приросту точності класифікатора.

Також з рис. 3.5 видно, що класифікація жіночих знімків складніша. Тому варто зауважити, що на рис 3.6 показано: якщо класифікатор навчений розпізнавати усі знімки, то краще вміє розпізнавати як довільні знімки, так і особливо помітно приріст точності у розпізнаванні зображень пацієнтів жіночої

статі, що свідчить про краще узагальнення ознак, що стосуються хвороби, саме при навчанні не лише на жіночих знімках.

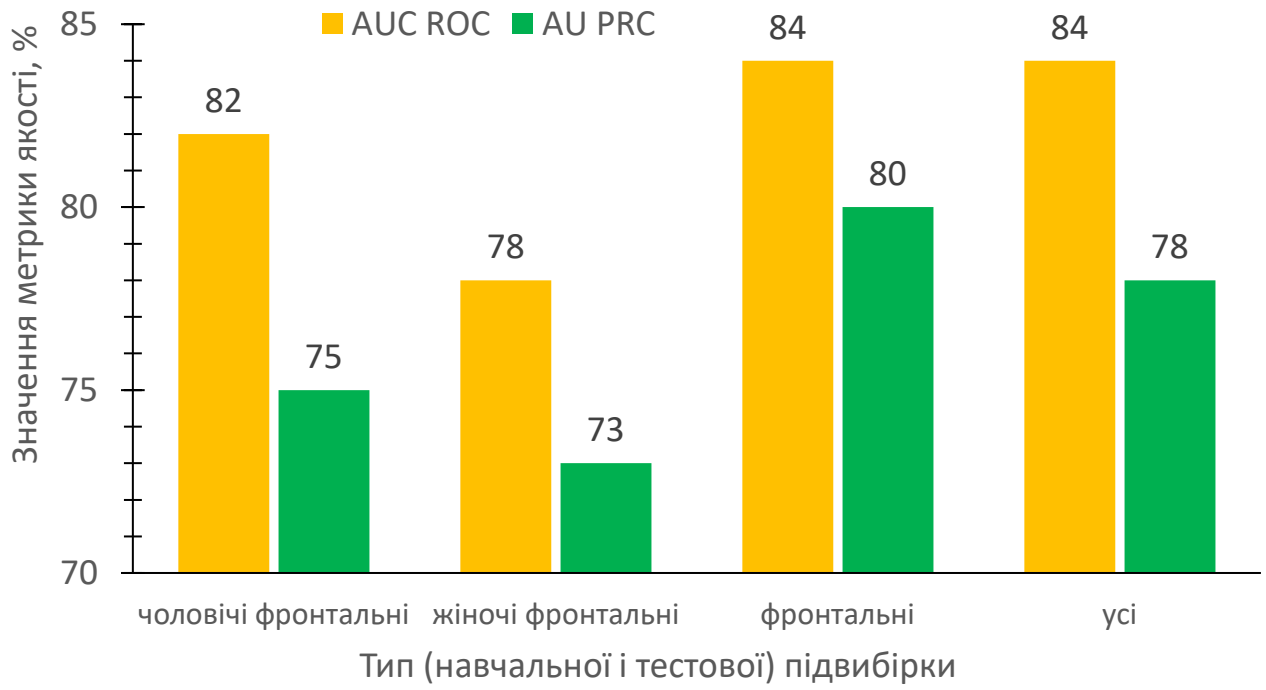

Рис.3.5 Метрики якості моделей, побудованих на архітектурі Xception, при навчанні та тестуванні на різних підвибірках даних.

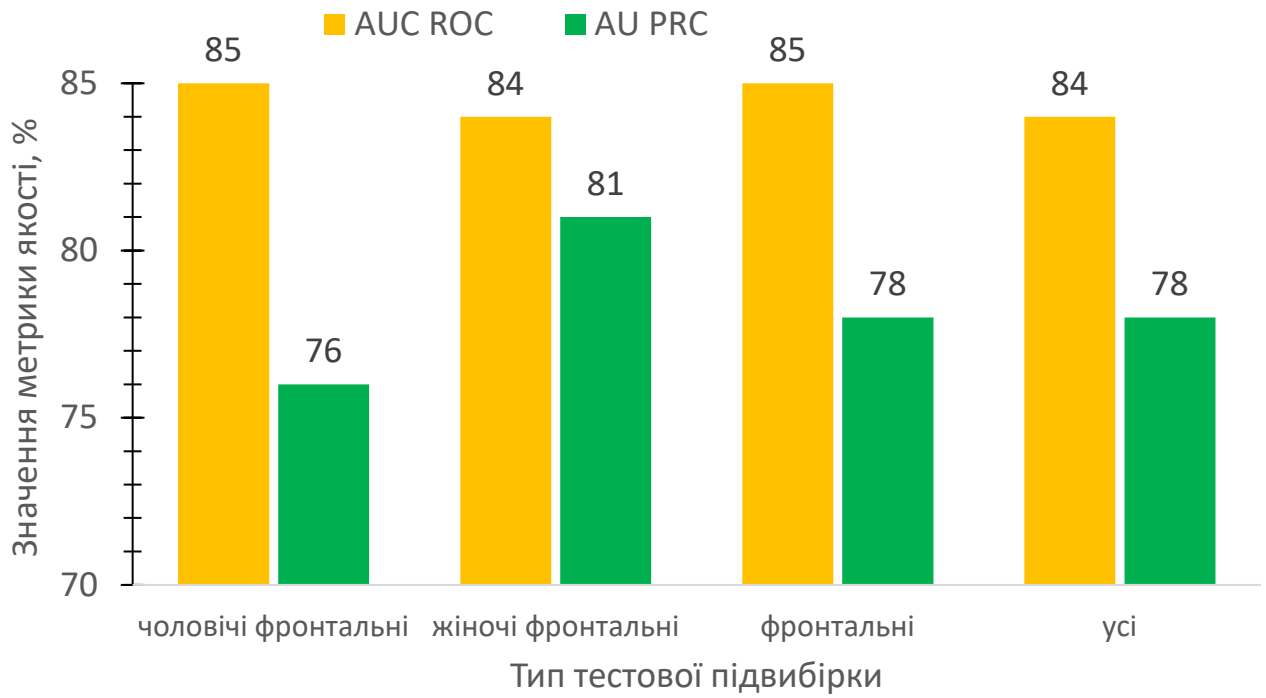

Рис.3.6 Метрики якості моделей, побудованих на архітектурі Xception, при навчанні на усіх знімках, а тестуванні на різних підвибірках.

3.3 Оптимальна модель

У ході виконання роботи та після оцінки метрик якості класифікаторів, серед досліджених архітектур нейронних мереж, використання різних розмірів зображення та підходів до попередньої обробки й розподілу даних, кращі результати було отримано при використанні архітектури Xception, розміру знімків 160x160 пікселів та навчанні на повному наборі даних.

Досягнуто таких показників метрик якості класифікатора:

- AUC ROC середнє значення – 84%.
- AU PRC середнє значення – 77.6%.
- F-міра – 76%.

Таке значення F-міри досягнуто при використанні різних порогів класифікації для кожної хвороби: 16.3% для ателектазу, 3.4% для кардіомегалії, 11.3% для легеневого ущільнення, 29.1% для легеневого набряку та 20.7% для плеврального випоту.

На наступному рис. 3.7 зображено PR-криві для кожного класу та усереднену за усіма класами криву (micro-average на графіку) оптимальної моделі.

А також продемонстровано як змінюються precision, recall та F-міра на PR-кривій ателектазу у залежності від значення порогу класифікації на 3 точках: a, b та c.

Наприклад, якщо необхідно визначитися чи давати пацієнту ліки з сильними побічними ефектами, то краще впевнитися у наявності хвороби – мати високе значення precision, для чого треба збільшити поріг активації. Такому випадку відповідає точка (a) з порогом класифікації 23.1%, у якій значення precision – 93%, recall – 34%, а F-міри –50%.

У іншому випадку, коли треба не пропустити наявність хвороби (щоб точно перестрахуватися та провести додаткові аналізи, наприклад), то треба високе

значення чутливості (recall). Це випадок точки (b) з порогом класифікації 8.9%, у якій значення precision – 63%, recall – 93%, а F-міри –75%.

А найвищому значенню середнього гармонійного від точності та чутливості, тобто F-мірі, відповідає точка (c) з порогом класифікації 16.3%, у якій значення precision – 81%, recall – 81% та F-міри –81%.

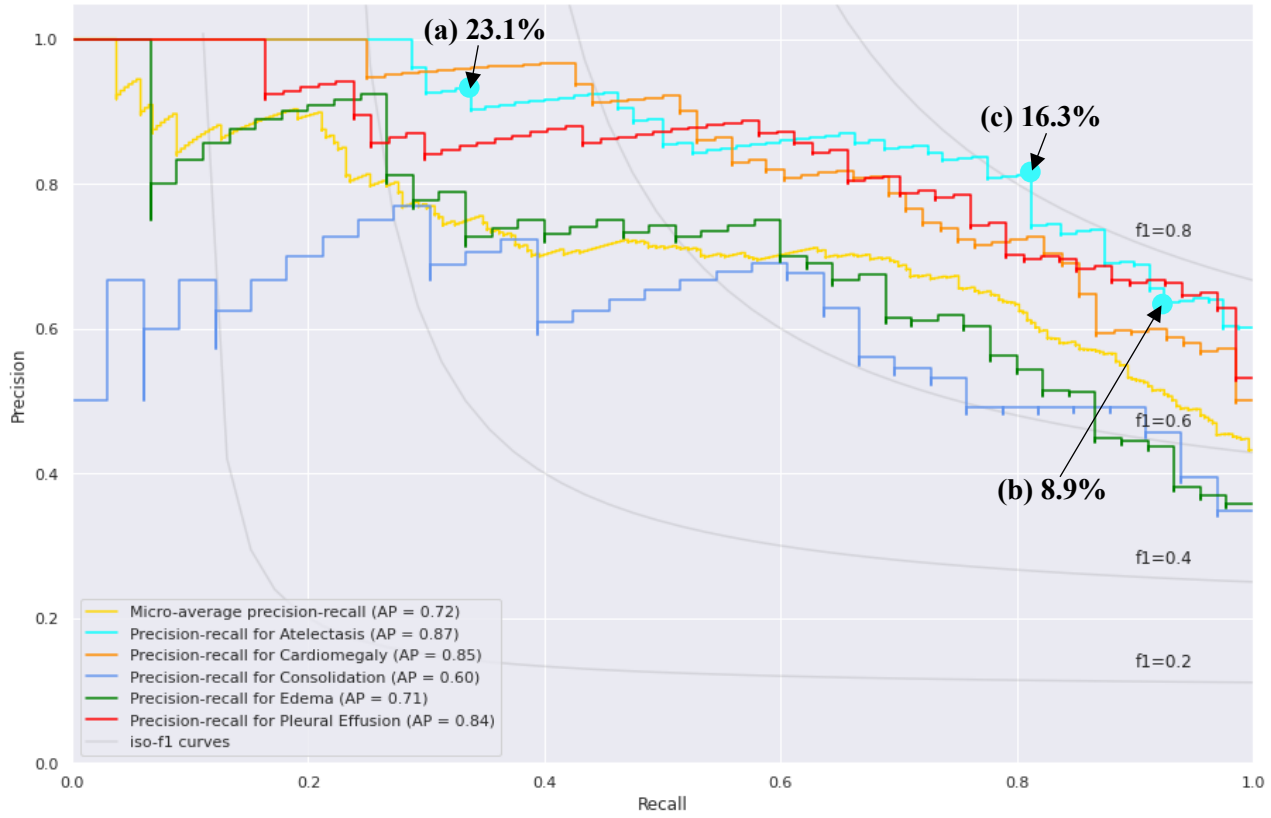

Рис. 3.7 PR-криві для оптимальної моделі.

На рис. 3.8 для оптимальної моделі зображено ROC-криві для кожного класу (захворювання) та усереднені за класами криві. Вказані значення площі під кривими.

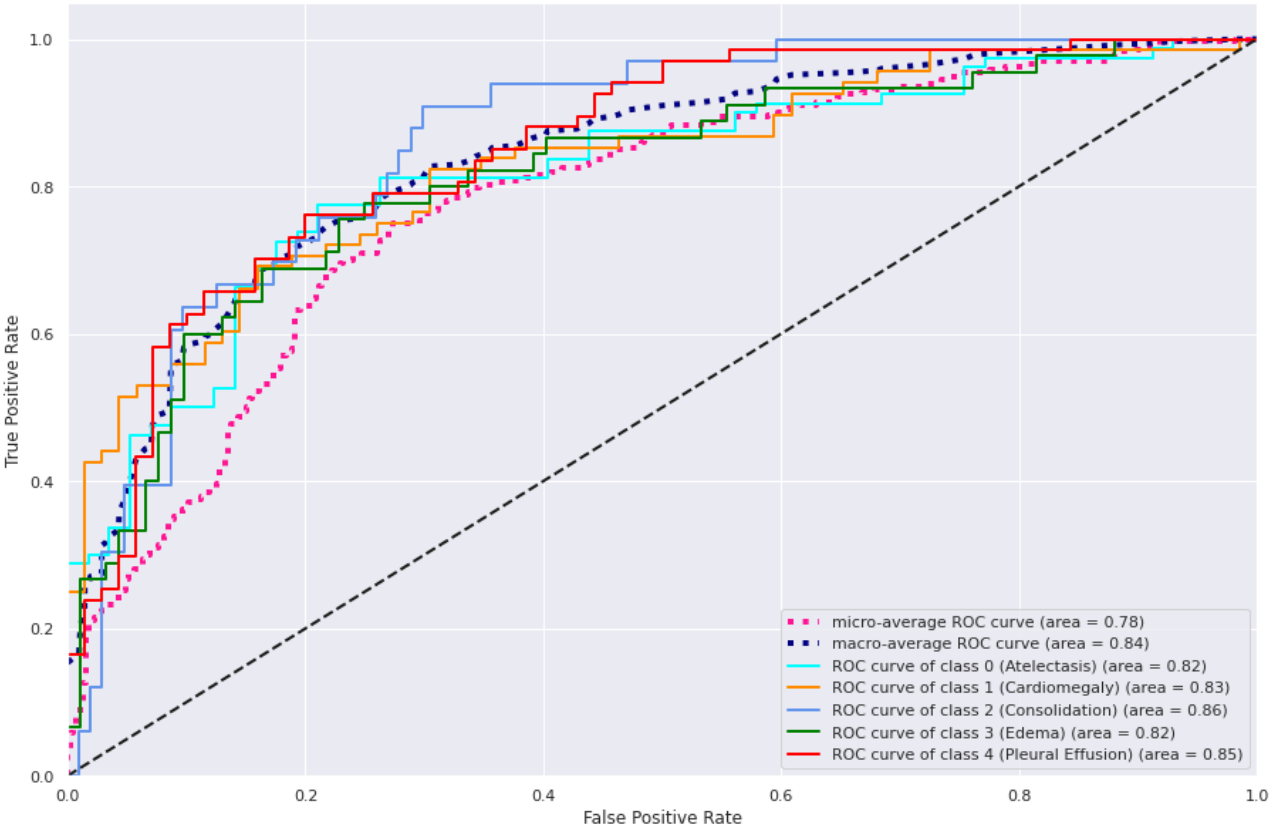

Рис. 3.8 ROC-криві для оптимальної моделі.

Повний список досліджених метрик та інших характеристик натренованих моделей можна переглянути у таблиці онлайн за посиланням [42].

3.4 Порівняння з іншими роботами

На рис. 3.9 продемонстровано порівняння метрики AU PRC з роботою [12] авторів використаного набору даних. Отримано приріст якості класифікатора у даній метриці у середньому на **7.4%** відсотки та отримано кращі результати для 4 з 5 хвороб при використанні оптимальної моделі.

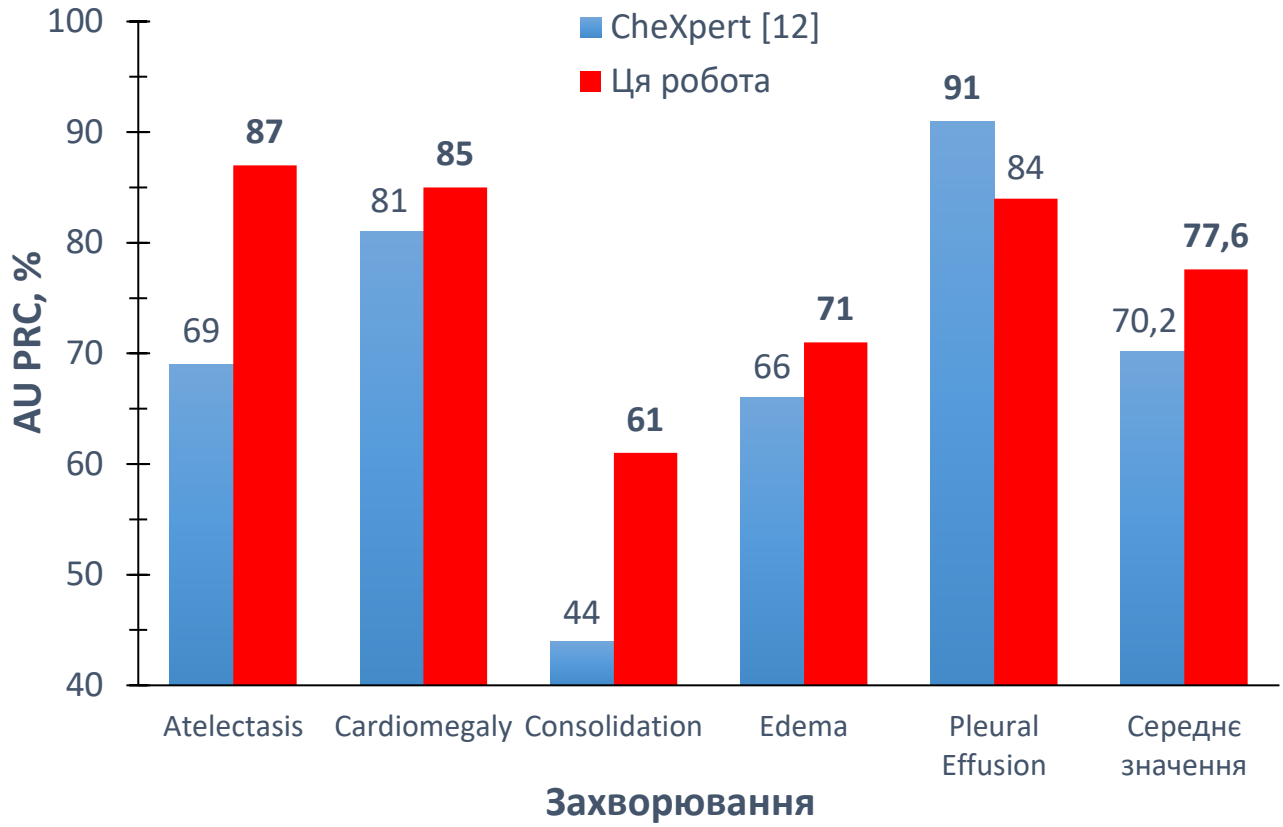

Рис. 3.9 Порівняння метрики AU PRC з роботою [12].

На рисунку 3.10 показано порівняння метрики AUC ROC з роботою [30]. На жаль, у роботі не деталізовано інші метрики для доступності порівняння. Покращені результати отримано для 2 хвороб та у середньому на **1.2%** відсотки. Результати цікаві тим, що у цій роботі ігноруються мітки невпевненості класів, а у роботі [30] ігнорується (не потрапляє у навчальну вибірку) весь знімок при наявності мітки невпевненості.

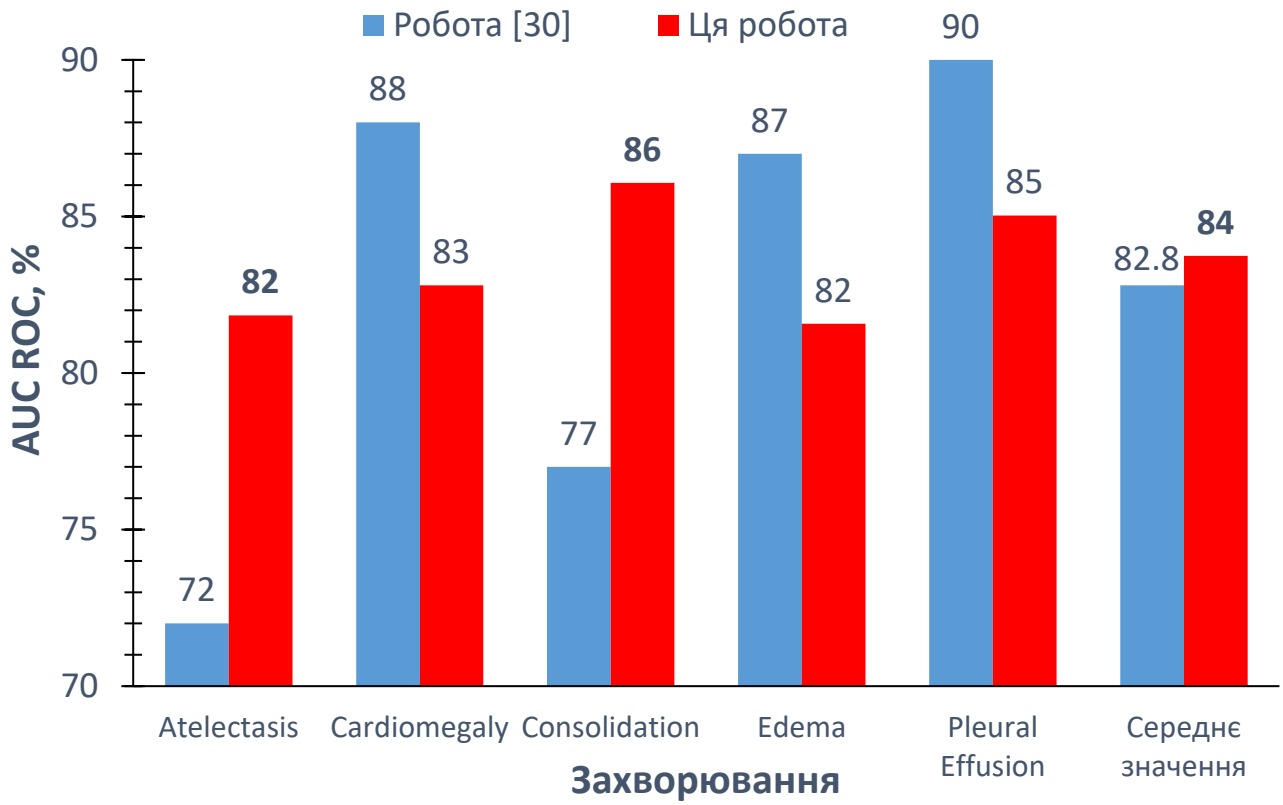

Рис. 3.10 Порівняння метрики AUC ROC з роботою [30].

Для подальшого покращення метрик якості класифікатора можна розглянути використання ансамблю навчених моделей нейронних мереж та аугментації знімків при навчанні, як це демонструвалося в [12]. Але у багатьох випадках такі методи використовуються для невеликого, але максимально можливого значення певної метрики, важливою у конкурсі. Ще треба враховувати часові затрати та апаратні потужності для таких задач, особливо при використанні ансамблю. А у даній роботі зосереджено увагу на дослідженнях для визначення кращої окремо взятої архітектури нейронної мережі, дослідження сучасних архітектур та різних підходів до обробки (попередньої підготовки) даних та іншого, вище описаного у роботі. Звісно, і для таких досліджень наявний широкий простір для подальших досліджень при продовженні роботи над темою .

Висновки

У результаті виконання дипломної роботи магістра:

1. Побудовані моделі класифікаторів рентгенівських знімків грудної клітини на основі глибоких нейронних мереж сучасних архітектур DenseNet-121, Xception, MobileNetV2, EfficientNetV2 і CCT.
2. Встановлені залежності метрик якості побудованих моделей AUC ROC і AU PRC на тестовій вибірці, а також часу навчання моделей від розміру зображень знімків. Визначені метрики якості моделей при використанні для їх навчання різних підвибірок навчальної вибірки, поділених за ознаками статі пацієнта та типу проекції рентгенівських знімків.
3. Запропонована методика розпізнавання захворювань грудної клітини (модель Xception, коректне приведення зображень до квадратної форми розміром 160x160 пікселів, навчання на повній навчальній вибірці), яка дозволяє досягнути середніх значень метрик якості AUC ROC 84%, AU PRC 77.6%, F-міри 76%, що перевищує відомі з літератури відповідні значення, отримані на основі ансамблю нейронних мереж.

Список використаних джерел

1. World Health Organization (WHO), «The top 10 causes of death,» 9.12.2020. [Онлайн]. Доступний за: https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death. [Дата звернення: 05.05.2022].

2. Опендатабот, «Рекордне скорочення населення за 11 років,» 26.01.2022. [Онлайн]. Доступний за: https://opendatabot.ua/analytics/death-in-november-2021. [Дата звернення: 05.05.2022].

3. А. С. Свінціцький та інші, Внутрішні хвороби. Підручник, заснований на принципах доказової медицини 2018/19, 2019.

4. T. Franquet, "Imaging of pneumonia: trends and algorithms," *European Respiratory Journal,* vol. 1, no. 18, pp. 196-208, 2001.

5. T. Cherian et al., "Standardized interpretation of paediatric chest radiographs for the diagnosis of pneumonia in epidemiological studies," *Bulletin of the World Health Organization,* vol. 83, no. 5, pp. 353-359, 2005.

6. William H. Frishman et al., "Cardiomegaly on chest x-ray: prognostic implications from a ten-year cohort study of elderly subjects: a report from the Bronx Longitudinal Aging Study," *American Heart Journal,* vol. 124, no. 4, pp. 1026-1030, 1992, doi: 10.1016/0002-8703(92)90987-7.

7. F. Tavora, Y. Zhang, M. Zhang, et al., «Cardiomegaly is a common arrhythmogenic substrate in adult sudden cardiac deaths, and is associated with obesity,» *Pathology, vol*. 44, № 3, pp. 187-191, 2012, doi: 10.1097/PAT.0b013e3283513f54.

8. Á. Mingote at al., «Prevalence and clinical consequences of atelectasis in SARS-CoV-2 pneumonia: a computed tomography retrospective cohort study,» *BMC Pulmonary Medicine, vol*. 21, 2021, doi: 10.1186/s12890-021-01638-9.

9. R. S. Wiener et al., "Hospital and long-term survival of patients with acute pulmonary edema associated with coronary artery disease," *The American journal of cardiology,* vol. 60, no. 1, pp. 33-35, 1987, doi:10.1016/0002-9149(87)90979-9.

10. A. S. Kookoolis et al., «Mortality of Hospitalized Patients with Pleural Effusions,» *Journal of Pulmonary & Respiratory Medicine,* vol. 4, № 3, p. 184, 2014, doi: 10.4172/2161-105X.1000184.

11. D. S. Kermany et al., "Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning," *Cell,* vol. 5, no. 172, pp. 1122-1131, 02 2018, doi: 10.1016/j.cell.2018.02.010.

12. J. Irvin, P. Rajpurkar, M. Ko, et al., "CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison," in *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, Hawaii, USA, 2019, doi: 10.1609/aaai.v33i01.3301590.

13. X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, R. M. Summers, «ChestX-Ray8: Hospital-Scale Chest X-Ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases,» *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 3462-3471, 2017, doi: 10.1109/CVPR.2017.369.

14. S. Raoof et al., «Interpretation of plain chest roentgenogram,» *Chest, vol*. 141, № 2, pp. 545-558, 2012, doi: 10.1378/chest.10-1302.

15. Stanford ML Group, «CheXpert: A Large Chest X-Ray Dataset And Competition,» [Онлайн]. Доступний за: https://stanfordmlgroup.github.io/competitions/chexpert/. [Дата звернення: 05.05.2022].

16. R. Smithuis, «Chest X-Ray - Lung disease,» 01 02 2014. [Онлайн]. Доступний за: https://radiologyassistant.nl/chest/chest-x-ray/lung-disease. [Дата звернення: 05.05.2022].

17. S. Behnke, "Hierarchical Neural Networks for Image Interpretation," *Lecture Notes in Computer Science,* vol. 2766, 2003.

18. Quoc V. Le, Marc'Aurelio Ranzato, et al., «Building High-level Features Using Large Scale Unsupervised Learning,» 2011.

19. D. C. Ciresan, U. Meier, et al., "Flexible, High Performance Convolutional Neural Networks for Image Classification," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, Barcelona, Catalonia, Spain, 2011, doi: 10.5591/978-1-57735-516-8/IJCAI11-210.

20. H. P. Martinez, Y. Bengio, G. N. Yannakakis, «Learning deep physiological models of affect,» *IEEE Computational Intelligence Magazine,* vol. 8, № 2, pp. 20-33, 2013, doi: 10.1109/MCI.2013.2247823.

21. D. Cireşan, U. Meier, J. Schmidhuber, «Multi-column Deep Neural Networks for Image Classification,» *IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* pp. 3642-3649, 2012, doi: 10.1016/j.neunet.2012.02.023.

22. A. Vaswani et al., "Attention is All you Need," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, Long Beach, California, USA, 2017, doi: 10.48550/arXiv.1706.03762.

23. L. Xiaoxuan et al., «A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis,» *The Lancet. Digital health,* vol. 1, № 6, pp. 271-297, 2019, doi: 10.1016/s2589-7500(19)30123-2.

24. Eric J Topol, «High-performance medicine: the convergence of human and artificial intelligence,» *Nature medicine,* vol. 25, № 1, pp. 44-56, 2019, doi: 10.1038/s41591-018-0300-7.

25. W. B. Schwartz et al., «Artificial intelligence in medicine. Where do we stand?,» *The New England journal of medicine,* vol. 316, № 11, pp. 685-688, 1987, doi: 10.1056/nejm198703123161109.

26. G. Huang et al., "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, Hawaii, USA, 2017, doi: 10.48550/arXiv.1608.06993.

27. Hieu H. Pham, Tung T. Le, Dat Q. Tran, et al., «Interpreting chest X-rays via CNNs that exploit hierarchical disease dependencies and uncertainty labels,» *Neurocomputing,* vol. 4, № 437, pp. 186-194, 2021, doi: 10.1016/j.neucom.2020.03.127.

28. Jia Deng et al., "ImageNet: a Large-Scale Hierarchical Image Database," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Miami, Florida, USA, 2009, doi: 10.1109/CVPR.2009.5206848.

29. Z. Yuan, Y. Yan, M. Sonka, T. Yang, «Large-scale Robust Deep AUC Maximization: A New Surrogate Loss and Empirical Studies on Medical Image Classification,» in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada, 2021, doi: 10.1109/ICCV48922.2021.00303.

30. I. Allaouzi, M. Ben Ahmed, "A Novel Approach for Multi-Label Chest X-Ray Classification of Common Thorax Diseases," *IEEE Access,* vol. 7, pp. 64279-64288, 2019, doi: 10.1109/ACCESS.2019.2916849.

31. E. Bisong, "Google Colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Berkeley, CA, Apress, 2019, doi: 10.1007/978-1-4842-4470-8_7.

32. Thomas Kluyver et al., «Jupyter Notebooks – a publishing format for reproducible computational workflows,» in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, Göttingen, Germany, 2016, doi: 10.3233/978-1-61499-649-1-87.

33. E. Bisong, «TensorFlow 2.0 and Keras,» in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Berkeley, CA, Apress, 2019, doi: 10.1007/978-1-4842-4470-8_30.

34. «Keras Applications,» [Онлайн]. Доступний за: https://keras.io/api/applications/. [Дата звернення: 05.05.2022].

35. F. Chollet, «Xception: Deep Learning With Depthwise Separable Convolutions,» in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawaii, USA, 2017, doi: 10.48550/arXiv.1610.02357.

36. Mingxing Tan, Quoc Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, California, USA, 2019, doi: 10.48550/arXiv.1905.11946.

37. Mingxing Tan, Quoc Le, "EfficientNetV2: Smaller Models and Faster Training," in *International Conference on Machine Learning*, Virtual, 2021, doi: 10.48550/arXiv.2104.00298.

38. M. Sandler et al., «MobileNetV2: Inverted Residuals and Linear Bottlenecks,» in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, 2018, doi: 10.48550/arXiv.1801.04381.

39. Ali Hassani et al., «Escaping the Big Data Paradigm with Compact Transformers,» 13.08.2021, doi: 10.48550/arXiv.2104.05704.

40. F. Chollet et al., «Keras,» 2015. [Онлайн]. Доступний за: https://keras.io/. [Дата звернення: 05.05.2022].

41. F. Pedregosa et al., «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research,* vol. 12, № 85, pp. 2825-2830, 2011.

42. В. В. Кравченко, «Таблиця в Google Drive з повним об'ємом результатів дослідження,» [Онлайн]. Доступний за: https://docs.google.com/spreadsheets/d/19u8cdLV7WTmhjYsceyWWuUEGW_8vULkX/edit?usp=sharing&ouid=107318636244775930171&rtpof=true&sd=true. [Дата звернення: 05.05.2022].

Додаток А CNN

```
# -*- coding: utf-8 -*-
"""universal_CNN_CheXpert.ipynb"
Automatically generated by Colaboratory.
Original file is located at

https://colab.research.google.com/drive/14KQHG3UsJF64wwD50KtA3v3zDzwC_Bhx
# Start and GPU check
"""
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
# clear GPU memory
from numba import cuda
device = cuda.get_current_device()
device.reset()
"""# Additional Installation"""
!pip install scikit-learn-intelex
"""# Google Drive"""
from google.colab import drive
drive.mount('/content/drive')
!cp -vr /content/drive/MyDrive/mag/data/full/cropped/CheXpert-v1.0-
small.zip /content/sample_data/
!unzip /content/sample_data/CheXpert-v1.0-small.zip -d /content/CheXpert-
v1.0-small > /dev/null
from pathlib import Path
#chestxrays_root = Path(r"C:\univ\data\short\cropped")
chestxrays_root = Path(r"/content/CheXpert-v1.0-small/")
data_path = chestxrays_root
save_path = '/content/drive/MyDrive/mag/models/multi/efficientB0V2/'
#save_path = Path(r"C:\univ\models\check")
```

```
"""# Imports"""
# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os
import random

from tensorflow.keras.applications import EfficientNetV2B0 #для
EfficientNetV2
# from tensorflow.keras.applications.xception import Xception # для
Xception
# from tensorflow.keras.applications.densenet import DenseNet121 # для
DenseNet121
# from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2,
preprocess_input # для MobileNetV2
from tensorflow.keras.layers import
Input,Conv2D,Dense,Flatten,Dropout,GaussianDropout,MaxPooling2D,GlobalAve
ragePooling2D,ActivityRegularization
#from keras.preprocessing.image import ImageDataGenerator
from keras_preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping,
ModelCheckpoint
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.metrics import  AUC
import tensorflow.keras.regularizers
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow import keras
import tensorflow as tf
from sklearnex import patch_sklearn
```

```
patch_sklearn()
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.preprocessing import MultiLabelBinarizer
# %matplotlib inline
from tensorboard import notebook
# Load the TensorBoard notebook extension
# %load_ext tensorboard
import datetime, os
# tf.compat.v1.enable_eager_execution()
auc_metric = keras.metrics.AUC(multi_label=True)
autotune = tf.data.AUTOTUNE
"""# Model Naming and Config"""
log_dir = save_path + "/efficientB0V2/logs"
logdir = os.path.join(log_dir, datetime.datetime.now().strftime("%d%m%Y-
%H%M%S"))

model_count = "colab_multi_nosex_nosides_efficientV2B0_1_"
size = 1
img_size = 160
image_channels = 1
image_height = img_size
image_width  = img_size
batch_size   = 256
seed = 42
no_of_epochs = 100
channel_name = "RGB" if image_channels == 3 else ''
colorMode = "rgb" if image_channels == 3 else "grayscale"
model_name = '/{0}_{1}_{2}_amount_{3}'.format(model_count, img_size,
channel_name, size)
chexnet_targets = ['No Finding',
       'Enlarged Cardiomediastinum', 'Cardiomegaly', 'Lung Opacity',
       'Lung Lesion', 'Edema', 'Consolidation', 'Pneumonia',
'Atelectasis',
       'Pneumothorax', 'Pleural Effusion', 'Pleural Other', 'Fracture',
```

```
       'Support Devices']
chexpert_targets = ['Atelectasis', 'Cardiomegaly', 'Consolidation',
'Edema', 'Pleural Effusion']
u_one_features = ['Atelectasis', 'Edema']
u_zero_features = ['Cardiomegaly', 'Consolidation', 'Pleural Effusion']
targets = chexpert_targets
num_classes = len(targets)

model_name = '{0}_{1}'.format(model_name, 'full') if len(targets) ==
len(chexnet_targets) else '{0}_{1}'.format(model_name, 'short')
model_name
"""# Feature Extraction methods"""
def feature_string(row):
    feature_list = []
    for feature in u_one_features:
        if row[feature] in [-1,1]:
            feature_list.append(feature)

    for feature in u_zero_features:
        if row[feature] == 1:
            feature_list.append(feature)

    return ';'.join(feature_list)
def feature_string_list(row):
    feature_list = []
    for feature in u_one_features:
        if row[feature] in [-1,1]:
            feature_list.append(feature)

    for feature in u_zero_features:
        if row[feature] == 1:
            feature_list.append(feature)

    return feature_list
```

```
def full_feature_string(row):
    feature_list = []
    for feature in targets:
        if row[feature] == 1:
            feature_list.append(feature)

    return ';'.join(feature_list)

def full_feature_string_list(row):
    feature_list = []
    for feature in targets:
        if row[feature] == 1:
            feature_list.append(feature)

    return feature_list

def full_feature_string_arr(row):
    feature_arr = np.array([])
    for feature in targets:
        if row[feature] == 1:
            feature_arr = np.append(feature_arr, feature)

    return feature_arr

"""# DataFrame data config"""

full_train_df = pd.read_csv(data_path/'CheXpert-v1.0-small/train.csv',
sep=',')
full_valid_df = pd.read_csv(data_path/'CheXpert-v1.0-small/valid.csv',
sep=',')

current_train_df = full_train_df.copy()
current_train_df = current_train_df.loc[(current_train_df['Sex'] ==
'Male') & (current_train_df['Frontal/Lateral'] == 'Frontal')]
```

```
current_test_df = full_valid_df.copy()
current_test_df = current_test_df.loc[(current_test_df['Sex'] == 'Male')
& (current_test_df['Frontal/Lateral'] == 'Frontal')]
current_train_df['train_valid'] = False
current_test_df['train_valid'] = True
current_train_df['patient'] =
current_train_df.Path.str.split('/',3,True)[2]
current_train_df  ['study'] =
current_train_df.Path.str.split('/',4,True)[3]
current_test_df['patient'] =
current_test_df.Path.str.split('/',3,True)[2]
current_test_df  ['study'] =
current_test_df.Path.str.split('/',4,True)[3]
current_train_df['feature_string'] =
current_train_df.apply(full_feature_string_list,axis = 1).fillna('')
current_test_df['feature_string'] =
current_test_df.apply(full_feature_string_list,axis = 1).fillna('')
full_df = pd.concat([current_train_df, current_test_df])
full_df['feature_string'] = full_df.apply(full_feature_string_list,axis =
1).fillna('')
def get_sample_df(df, sample_perc = 0.05):

    train_only_df = df.copy()
    unique_patients = train_only_df.patient.unique()
    mask = np.random.rand(len(unique_patients)) <= sample_perc
    sample_patients = unique_patients[mask]
    sample_df =
train_only_df[current_train_df.patient.isin(sample_patients)]
    return sample_df
train_df = get_sample_df(current_train_df, size)
train_df["Path"] = train_df["Path"].apply(lambda x: str(data_path) + "/"
+ x)
test_df = current_test_df.copy()
```

```
test_df["Path"] = test_df["Path"].apply(lambda x: str(data_path) + "/" +
x)
test_df_no_empty = test_df.copy()
test_df_no_empty =
test_df_no_empty[test_df_no_empty['feature_string'].map(lambda f: len(f))
> 0]
"""# ImageDataGenerator"""
datagen = ImageDataGenerator(validation_split=0.05) # для EfficientNet,
MobileNetV2
datagen = ImageDataGenerator(rescale=1./255., validation_split=0.05) #
для Xception, DenseNet
train_generator=datagen.flow_from_dataframe(
dataframe=train_df,
directory=None,
x_col= "Path",
y_col="feature_string",
subset="training",
batch_size=batch_size,
seed=seed,
shuffle=True,
class_mode="categorical",
target_size=(img_size,img_size),
color_mode='grayscale')
valid_generator=datagen.flow_from_dataframe(
dataframe=train_df,
directory=None,
x_col="Path",
y_col="feature_string",
subset="validation",
batch_size=batch_size,
seed=seed,
shuffle=True,
class_mode="categorical",
target_size=(img_size,img_size),
```

```
color_mode='grayscale')
test_datagen=ImageDataGenerator() #rescale=1./255.
test_generator=test_datagen.flow_from_dataframe(
dataframe=test_df_no_empty,
directory=None,
x_col="Path",
y_col=None,
batch_size=batch_size,
seed=seed,
shuffle=False,
class_mode=None,
target_size=(img_size,img_size),
color_mode='grayscale')

train_ds = tf.data.Dataset.from_generator(
    lambda: train_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes = ([None, img_size, img_size, image_channels],
                     [None, num_classes]))
train_ds = train_ds.prefetch(buffer_size=autotune)
# train_ds = train_ds.cache()
valid_ds = tf.data.Dataset.from_generator(
    lambda: valid_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes = ([None, img_size, img_size, image_channels],
                     [None, num_classes]))
valid_ds = valid_ds.prefetch(buffer_size=autotune)
# valid_ds = valid_ds.cache()
test_ds = tf.data.Dataset.from_generator(
    lambda: test_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes = ([None, img_size, img_size, image_channels],
                     [None, num_classes]))
test_ds = test_ds.prefetch(buffer_size=autotune)
```

```
# test_ds = test_ds.cache()
"""# Callbacks"""
# for validation AUC ROC maximization
reduce_learning_rate = ReduceLROnPlateau(monitor='val_auc',
                                         mode='max',
                                         factor=0.2,
                                         patience=10,
                                         cooldown=2,
                                         min_lr=0.00001,
                                         verbose=1)
early_Stop = EarlyStopping(monitor='val_auc',
                           mode='max',
                           patience=20,
                           verbose=1,
                           restore_best_weights=True)
model_checkpoint = ModelCheckpoint(filepath=str(save_path) + model_name
+"_{epoch:02d}",
                                   monitor="val_auc",
                                   mode="max",
                                   save_best_only=True,
                                   verbose=1)
tensorboard_callback = keras.callbacks.TensorBoard(logdir,
histogram_freq=1, write_graph=True)
my_callbacks = [reduce_learning_rate, early_Stop, model_checkpoint,
tensorboard_callback]
# for validation loss minimization
reduce_learning_rate = ReduceLROnPlateau(monitor='val_loss',
                                         mode='min',
                                         factor=0.2,
                                         patience=10,
                                         cooldown=2,
                                         min_lr=0.00001,
                                         verbose=1)
early_Stop = EarlyStopping(monitor='val_loss',
```

```
                              mode='min',
                              patience=20,
                              verbose=1,
                              restore_best_weights=True)
model_checkpoint = ModelCheckpoint(filepath=str(save_path) + model_name
+"_{epoch:02d}",
                                   monitor="val_loss",
                                   mode="min",
                                   save_best_only=True,
                                   verbose=1)
tensorboard_callback = keras.callbacks.TensorBoard(logdir,
histogram_freq=1, write_graph=True)
my_callbacks = [reduce_learning_rate, early_Stop, model_checkpoint,
tensorboard_callback]
"""# Model Creation"""
# для EfficientNet
class_count = len(targets)
inputTensor = Input(shape=(image_height, image_width, image_channels))
base_model = EfficientNetV2B0(
    include_top=False,
    weights=None,
    input_tensor=inputTensor,
    input_shape=(image_height, image_width, image_channels),
    pooling='avg',
    classes=class_count)
x = base_model.output
predictions = Dense(class_count, activation='sigmoid')(x)
# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
# compile the model (should be done *after* setting layers to non-
trainable)
model.compile(optimizer=RMSprop(learning_rate=0.01),
loss=keras.losses.BinaryCrossentropy(),  metrics=[auc_metric])
# для Xception
```

```
class_count = len(targets)
inputTensor = Input(shape=(image_height, image_width, image_channels))
base_model = Xception(input_tensor=inputTensor,
                      input_shape=(image_height, image_width,
image_channels),
                      weights = None,
                      include_top=False,
                      classes=class_count)
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(class_count, activation='sigmoid')(x)
# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
# compile the model (should be done *after* setting layers to non-
trainable)
model.compile(optimizer=RMSprop(learning_rate=0.01),
loss=keras.losses.BinaryCrossentropy(),  metrics=[auc_metric])
# для DenseNet121
class_count = len(targets)
inputTensor = Input(shape=(image_height, image_width, image_channels))

base_model =
DenseNet121(input_tensor=inputTensor,input_shape=(image_height,
image_width, image_channels),
                      weights = None,  include_top=False,
classes=class_count)
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(class_count, activation='sigmoid')(x)
# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
# compile the model (should be done *after* setting layers to non-
trainable)
```

```
model.compile(optimizer=RMSprop(learning_rate=0.01),
loss=keras.losses.BinaryCrossentropy(),  metrics=[auc_metric])
# для MobileNetV2
class_count = len(targets)
inputShape = (image_height, image_width, image_channels)
inputs = Input(shape=inputShape)
inputsScaled = keras.layers.Rescaling(scale=1./127.5, offset=-1)(inputs)
# Rescale inputs
base_model = MobileNetV2(input_shape=inputShape, alpha=1.0,
include_top=False, weights = None,
                         input_tensor=inputs, pooling='avg',
classes=class_count)(inputsScaled)
x = base_model
predictions = Dense(class_count, activation='sigmoid')(x)
# this is the model we will train
model = Model(inputs=inputs, outputs=predictions)
# compile the model (should be done *after* setting layers to non-
trainable)
model.compile(optimizer=RMSprop(learning_rate=0.01),
loss=keras.losses.BinaryCrossentropy(),  metrics=[auc_metric])
#base_model.summary()
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
"""# Training"""
#%tensorboard --logdir logdir
history = model.fit(
    x=train_ds,
    # initial_epoch = 10,
    steps_per_epoch=STEP_SIZE_TRAIN,
    validation_data=valid_ds,
    validation_steps=STEP_SIZE_VALID,
    callbacks=my_callbacks,
    epochs=no_of_epochs)
```

```
model.save(str(save_path) + model_name)
"""# History"""
history.history.keys()
df = pd.DataFrame({'epochs'       : history.epoch,
                   'auc'     : history.history['auc'],
                   'val_auc' : history.history['val_auc'],
                   'loss'         : history.history['loss'],
                   'val_loss'     : history.history['val_loss'],
                   'lr'           : history.history['lr']})
sns.set(rc = {'figure.figsize':(15,8)})
g = sns.pointplot(x="epochs", y="val_auc", data=df, fit_reg=False,
color='green')
g = sns.pointplot(x="epochs", y="auc", data=df, fit_reg=False)
g = sns.pointplot(x="epochs", y="val_loss", data=df, fit_reg=False,
color='orange')
g = sns.pointplot(x="epochs", y="loss", data=df, fit_reg=False,
color='yellow')
g = sns.pointplot(x="epochs", y="lr", data=df, fit_reg=False,
color='red')
"""# Load model"""
# modelToLoad =
'colab_multi_male_frontal_efficientV2B0_3__160__amount_1_short_10'
# model =
keras.models.load_model(Path(r"/content/drive/MyDrive/mag/models/multi/ef
ficientB0V2/"+modelToLoad))
#model.summary()
"""# TEST model"""
def printer_y(yId):
  print(orig_test_labels.iloc[yId])
  print(trans_y[yId])
  print(preds_prob[yId])
  print(preds[yId])
# predictions with threshold
# індекси значень, більших за поріг
```

```
# [рядок, індекс значення]
# вектор порогів -- щоб по кожному класу окремий поріг
def threshold_predictions(pred_threshold):
  pred_threshold = np.asarray(pred_threshold)
  shape = (num_classes,)
  pred_threshold = np.reshape(pred_threshold, (pred_threshold.size))
  if pred_threshold.size != num_classes:
    pred_threshold = np.full(shape, pred_threshold[0], dtype=float)
  arg_preds = np.argwhere(preds_prob > pred_threshold)
  preds_int = np.zeros(shape=preds_prob.shape, dtype=np.int8)
  for pred in arg_preds:
    preds_int[pred[0]][pred[1]] = 1
  return preds_int
# Get predictions
preds_prob = model.predict(test_generator, verbose=1)
preds=np.argmax(preds_prob, axis=1)
orig_test_labels = test_df_no_empty.feature_string
orig_test_labels_int = [targets.index(x[0]) for x in orig_test_labels]
print(preds_prob.shape)
print(orig_test_labels.shape)
generator_classes = np.array([x for x in
train_generator.class_indices.keys()])
generator_classes_list = [x for x in
train_generator.class_indices.keys()]
y = orig_test_labels
trans_y = MultiLabelBinarizer(classes=generator_classes).fit_transform(y)
arg_y = np.argwhere(trans_y == 1)
count_y_all = np.count_nonzero(trans_y)
count_y = np.count_nonzero(trans_y, axis = 0)
print(trans_y.shape)
# TP, FP
preds_thresholded = preds_int
count_preds = np.count_nonzero(preds_thresholded, axis = 0)
# [all_y, all_predicted as true, TP, FP]
```

```
tfpr = np.zeros((num_classes,4), dtype=np.int32)
for y in arg_y:
  if(preds_thresholded[y[0]][y[1]] == 1):
    tfpr[y[1]][2] += 1 #count TP
for class_ind in range(num_classes):
  tfpr[class_ind][0] = count_y[class_ind] # all_y
  tfpr[class_ind][1] = count_preds[class_ind] # all_predicted as true
  tfpr[class_ind][3] = tfpr[class_ind][1] - tfpr[class_ind][2] # FP
# Precision #Recall #F1 score
PRF = np.zeros((num_classes,3), dtype=np.float32)
for class_ind in range(num_classes):
  PRF[class_ind][0] = (tfpr[class_ind][2] / tfpr[class_ind][1])
  PRF[class_ind][1] = (tfpr[class_ind][2] / tfpr[class_ind][0])
  PRF[class_ind][2] = 2 * ((PRF[class_ind][0] * PRF[class_ind][1]) /
(PRF[class_ind][0] + PRF[class_ind][1]))
preds_int = threshold_predictions(0.5)
preds_thresholded = preds_int
axis_classes = generator_classes
print(classification_report(y_true=trans_y, y_pred=preds_thresholded,
target_names=generator_classes))

preds_int = threshold_predictions(0.25)
preds_thresholded = preds_int
axis_classes = generator_classes
print(classification_report(y_true=trans_y, y_pred=preds_thresholded,
target_names=generator_classes))
preds_int = threshold_predictions(0.1)
preds_thresholded = preds_int
axis_classes = generator_classes
print(classification_report(y_true=trans_y, y_pred=preds_thresholded,
target_names=generator_classes))
y_test = trans_y
y_prob = preds_prob
```

```
macro_roc_auc_ovo = roc_auc_score(y_test, y_prob, multi_class="ovo",
average="macro")
weighted_roc_auc_ovo = roc_auc_score(
    y_test, y_prob, multi_class="ovo", average="weighted"
)
macro_roc_auc_ovr = roc_auc_score(y_test, y_prob, multi_class="ovr",
average="macro")
weighted_roc_auc_ovr = roc_auc_score(
    y_test, y_prob, multi_class="ovr", average="weighted"
)
print(
    "One-vs-One ROC AUC scores:\n{:.6f} (macro),\n{:.6f} "
    "(weighted by prevalence)".format(macro_roc_auc_ovo,
weighted_roc_auc_ovo)
)
print(
    "One-vs-Rest ROC AUC scores:\n{:.6f} (macro),\n{:.6f} "
    "(weighted by prevalence)".format(macro_roc_auc_ovr,
weighted_roc_auc_ovr)
)
# for PR curve
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score

# For each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(trans_y[:, i],
preds_prob[:, i])
    average_precision[i] = average_precision_score(trans_y[:, i],
preds_prob[:, i])
```

```
# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(
    trans_y.ravel(), preds_prob.ravel()
)
average_precision["micro"] = average_precision_score(trans_y, preds_prob,
average="micro")
average_precision["macro"] = average_precision_score(trans_y, preds_prob,
average="macro")
import matplotlib.pyplot as plt
from itertools import cycle

# setup plot details
colors = cycle(["aqua", "darkorange", "cornflowerblue", "green", "red"])
_, ax = plt.subplots(figsize=(15, 10))
f_scores = np.linspace(0.2, 0.8, num=4)
lines, labels = [], []
for f_score in f_scores:
    x = np.linspace(0.01, 1)
    y = f_score * x / (2 * x - f_score)
    (l,) = plt.plot(x[y >= 0], y[y >= 0], color="gray", alpha=0.2)
    plt.annotate("f1={0:0.1f}".format(f_score), xy=(0.9, y[45] + 0.02))
display = PrecisionRecallDisplay(
    recall=recall["micro"],
    precision=precision["micro"],
    average_precision=average_precision["micro"],
)
display.plot(ax=ax, name="Micro-average precision-recall", color="gold")

for i, color in zip(range(num_classes), colors):
    display = PrecisionRecallDisplay(
        recall=recall[i],
        precision=precision[i],
        average_precision=average_precision[i],
    )
```

```
    display.plot(ax=ax, name="Precision-recall for
{0}".format(targets[i]), color=color)

# add the legend for the iso-f1 curves
handles, labels = display.ax_.get_legend_handles_labels()
handles.extend([l])
labels.extend(["iso-f1 curves"])
# set the legend and the axes
ax.set_xlim([0.0, 1.0])
ax.set_ylim([0.0, 1.05])
ax.legend(handles=handles, labels=labels, loc="best")
ax.set_title("Extension of Precision-Recall curve to multi-class")
print(average_precision)
plt.show()
x = preds_prob
y = trans_y
roc_auc_score(y, x, average=None)
```

Додаток Б CCT

```
# -*- coding: utf-8 -*-
"""universal_CCT_CheXpert.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1RAf1c72uv-
Dhfblx32iTnyTsfEi9xNJ0

# Start and GPU check
"""

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)

# clear GPU memory
from numba import cuda
device = cuda.get_current_device()
device.reset()

"""# Additional Installation"""

!pip install scikit-learn-intelex
!pip install -U -q tensorflow-addons

"""# Google Drive"""

from google.colab import drive
drive.mount('/content/drive')
```

```
!cp -vr /content/drive/MyDrive/mag/data/full/cropped/CheXpert-v1.0-small.zip /content/sample_data/
!unzip /content/sample_data/CheXpert-v1.0-small.zip -d /content/CheXpert-v1.0-small > /dev/null

from pathlib import Path

#chestxrays_root = Path(r"C:\univ\data\short\cropped")
chestxrays_root = Path(r"/content/CheXpert-v1.0-small/")
data_path = chestxrays_root

save_path = '/content/drive/MyDrive/mag/models/multi/'
#save_path = Path(r"C:\univ\models\check")

"""# Imports"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os
import random

from tensorflow.keras import layers
from keras_preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.metrics import  AUC
```

```
import tensorflow.keras.regularizers
from tensorflow.keras.optimizers import RMSprop, Adam

from tensorflow import keras

import tensorflow_addons as tfa
import tensorflow as tf

from sklearnex import patch_sklearn
patch_sklearn()
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.preprocessing import MultiLabelBinarizer

# %matplotlib inline

from tensorboard import notebook
# Load the TensorBoard notebook extension
# %load_ext tensorboard
import datetime, os

# tf.compat.v1.enable_eager_execution()

auc_metric = keras.metrics.AUC(multi_label=True, from_logits=False)

autotune = tf.data.AUTOTUNE

"""# Model Naming and Config"""

log_dir = save_path +"/cct/logs"
logdir = os.path.join(log_dir, datetime.datetime.now().strftime("%d%m%Y-
%H%M%S"))

model_count = "colab_multi_nosex_nosides_CCT_1_"
size = 1
```

```
img_size = 160
img_channels = 1
image_height = img_size
image_width  = img_size
batch_size = 128
seed = 42
no_of_epochs = 100
num_epochs = no_of_epochs

input_shape = (img_size, img_size, img_channels)

image_size = img_size
image_channels = img_channels

channel_name = "RGB" if image_channels == 3 else ''
colorMode = "rgb" if image_channels == 3 else "grayscale"
model_name = '/{0}_{1}_{2}_amount_{3}'.format(model_count, img_size,
channel_name, size)

chexnet_targets = ['No Finding',
       'Enlarged Cardiomediastinum', 'Cardiomegaly', 'Lung Opacity',
       'Lung Lesion', 'Edema', 'Consolidation', 'Pneumonia',
'Atelectasis',
       'Pneumothorax', 'Pleural Effusion', 'Pleural Other', 'Fracture',
       'Support Devices']

chexpert_targets = ['Atelectasis', 'Cardiomegaly', 'Consolidation',
'Edema', 'Pleural Effusion']

u_one_features = ['Atelectasis', 'Edema']
u_zero_features = ['Cardiomegaly', 'Consolidation', 'Pleural Effusion']

targets = chexpert_targets
```

```
class_count = len(targets)
num_classes = class_count

model_name = '{0}_{1}'.format(model_name, 'full') if len(targets) ==
len(chexnet_targets) else '{0}_{1}'.format(model_name, 'short')

model_name

"""## Hyperparameters and constants"""

positional_emb = True # змінюємо # True

output_channels = [32, 64] # змінюємо # [64, 128]
conv_layers = len(output_channels)

projection_dim = 64 # змінюємо # 128

num_heads = 1 # змінюємо # 2
transformer_units = [
    projection_dim, # змінюємо
    # projection_dim
]
transformer_layers = 1 # 2
stochastic_depth_rate = 0.1

learning_rate = 0.01 # 0.001
weight_decay = 0.0001

#model.summary()

"""# Feature Extraction methods"""

def feature_string(row):
    feature_list = []
```

```
    for feature in u_one_features:
        if row[feature] in [-1,1]:
            feature_list.append(feature)

    for feature in u_zero_features:
        if row[feature] == 1:
            feature_list.append(feature)

    return ';'.join(feature_list)

def feature_string_list(row):
    feature_list = []
    for feature in u_one_features:
        if row[feature] in [-1,1]:
            feature_list.append(feature)

    for feature in u_zero_features:
        if row[feature] == 1:
            feature_list.append(feature)

    return feature_list

def full_feature_string(row):
    feature_list = []
    for feature in targets:
        if row[feature] == 1:
            feature_list.append(feature)

    return ';'.join(feature_list)

def full_feature_string_list(row):
    feature_list = []
    for feature in targets:
        if row[feature] == 1:
```

```
            feature_list.append(feature)

    return feature_list

def full_feature_string_arr(row):
    feature_arr = np.array([])
    for feature in targets:
        if row[feature] == 1:
            feature_arr = np.append(feature_arr, feature)

    return feature_arr

"""# DataFrame data config"""

full_train_df = pd.read_csv(data_path/'CheXpert-v1.0-small/train.csv',
sep=',')
full_valid_df = pd.read_csv(data_path/'CheXpert-v1.0-small/valid.csv',
sep=',')

current_train_df = full_train_df.copy()
current_train_df = current_train_df.loc[(current_train_df['Sex'] ==
'Male') & (current_train_df['Frontal/Lateral'] == 'Frontal')]
current_test_df = full_valid_df.copy()
current_test_df = current_test_df.loc[(current_test_df['Sex'] == 'Male')
& (current_test_df['Frontal/Lateral'] == 'Frontal')]

current_train_df['train_valid'] = False
current_test_df['train_valid'] = True

current_train_df['patient'] =
current_train_df.Path.str.split('/',3,True)[2]
current_train_df  ['study'] =
current_train_df.Path.str.split('/',4,True)[3]
```

```
current_test_df['patient'] =
current_test_df.Path.str.split('/',3,True)[2]
current_test_df  ['study'] =
current_test_df.Path.str.split('/',4,True)[3]

current_train_df['feature_string'] =
current_train_df.apply(full_feature_string_list,axis = 1).fillna('')
current_test_df['feature_string'] =
current_test_df.apply(full_feature_string_list,axis = 1).fillna('')

full_df = pd.concat([current_train_df, current_test_df])
full_df['feature_string'] = full_df.apply(full_feature_string_list,axis =
1).fillna('')

def get_sample_df(df, sample_perc = 0.05):

    train_only_df = df.copy()
    unique_patients = train_only_df.patient.unique()
    mask = np.random.rand(len(unique_patients)) <= sample_perc
    sample_patients = unique_patients[mask]

    sample_df =
train_only_df[current_train_df.patient.isin(sample_patients)]
    return sample_df

train_df = get_sample_df(current_train_df, size)
train_df["Path"] = train_df["Path"].apply(lambda x: str(data_path) + "/"
+ x)

test_df = current_test_df.copy()
test_df["Path"] = test_df["Path"].apply(lambda x: str(data_path) + "/" +
x)

test_df_no_empty = test_df.copy()
```

```
test_df_no_empty =
test_df_no_empty[test_df_no_empty['feature_string'].map(lambda f: len(f))
> 0]

"""# ImageDataGenerator"""

datagen = ImageDataGenerator(rescale=1./255., validation_split=0.05)
train_generator=datagen.flow_from_dataframe(
dataframe=train_df,
directory=None,
x_col= "Path",
y_col="feature_string",
subset="training",
batch_size=batch_size,
seed=seed,
shuffle=True,
class_mode="categorical",
target_size=(img_size,img_size),
color_mode='grayscale')

valid_generator=datagen.flow_from_dataframe(
dataframe=train_df,
directory=None,
x_col="Path",
y_col="feature_string",
subset="validation",
batch_size=batch_size,
seed=seed,
shuffle=True,
class_mode="categorical",
target_size=(img_size,img_size),
color_mode='grayscale')

test_datagen=ImageDataGenerator(rescale=1./255.)
```

```
test_generator=test_datagen.flow_from_dataframe(
dataframe=test_df_no_empty,
directory=None,
x_col="Path",
y_col=None,
batch_size=batch_size,
seed=seed,
shuffle=False,
class_mode=None,
target_size=(img_size,img_size),
color_mode='grayscale')

train_ds = tf.data.Dataset.from_generator(
    lambda: train_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes = ([None, img_size, img_size, image_channels],
                     [None, num_classes]))
train_ds = train_ds.prefetch(buffer_size=autotune)
# train_ds = train_ds.cache()

valid_ds = tf.data.Dataset.from_generator(
    lambda: valid_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes = ([None, img_size, img_size, image_channels],
                     [None, num_classes]))
valid_ds = valid_ds.prefetch(buffer_size=autotune)
# valid_ds = valid_ds.cache()

test_ds = tf.data.Dataset.from_generator(
    lambda: test_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes = ([None, img_size, img_size, image_channels],
                     [None, num_classes]))
test_ds = test_ds.prefetch(buffer_size=autotune)
```

```
# test_ds = test_ds.cache()

"""# Callbacks"""

# for validation AUC ROC maximization
reduce_learning_rate = ReduceLROnPlateau(monitor='val_auc',
                                         mode='max',
                                         factor=0.2,
                                         patience=10,
                                         cooldown=2,
                                         min_lr=0.00001,
                                         verbose=1)

early_Stop = EarlyStopping(monitor='val_auc',
                           mode='max',
                           patience=20,
                           verbose=1,
                           restore_best_weights=True)

model_checkpoint = ModelCheckpoint(filepath=str(save_path) + model_name
+"_{epoch:02d}",
                                   monitor="val_auc",
                                   mode="max",
                                   save_best_only=True,
                                   verbose=1)

tensorboard_callback = keras.callbacks.TensorBoard(logdir,
histogram_freq=1, write_graph=True)

my_callbacks = [reduce_learning_rate, early_Stop, model_checkpoint,
tensorboard_callback]

# for validation loss minimization
reduce_learning_rate = ReduceLROnPlateau(monitor='val_loss',
```

```
                                        mode='min',
                                        factor=0.2,
                                        patience=10,
                                        cooldown=2,
                                        min_lr=0.00001,
                                        verbose=1)

early_Stop = EarlyStopping(monitor='val_loss',
                           mode='min',
                           patience=20,
                           verbose=1,
                           restore_best_weights=True)

model_checkpoint = ModelCheckpoint(filepath=str(save_path) + model_name
+"_{epoch:02d}",
                                   monitor="val_loss",
                                   mode="min",
                                   save_best_only=True,
                                   verbose=1)

tensorboard_callback = keras.callbacks.TensorBoard(logdir,
histogram_freq=1, write_graph=True)

my_callbacks = [reduce_learning_rate, early_Stop, model_checkpoint,
tensorboard_callback]

"""# The CCT tokenizer"""

class CCTTokenizer(layers.Layer):
    def __init__(
        self,
        kernel_size=3,
        stride=1,
        padding=1,
```

```
        pooling_kernel_size=3,
        pooling_stride=2,
        num_conv_layers=conv_layers,
        num_output_channels=output_channels,
        positional_emb=positional_emb,
        **kwargs,
    ):
        super(CCTTokenizer, self).__init__(**kwargs)

        # This is our tokenizer.
        self.conv_model = keras.Sequential()
        for i in range(num_conv_layers):
            self.conv_model.add(
                layers.Conv2D(
                    num_output_channels[i],
                    kernel_size,
                    stride,
                    padding="valid",
                    use_bias=False,
                    activation="relu",
                    kernel_initializer="he_normal",
                )
            )
            self.conv_model.add(layers.ZeroPadding2D(padding))
            self.conv_model.add(
                layers.MaxPool2D(pooling_kernel_size, pooling_stride,
"same")
            )

        self.positional_emb = positional_emb

    def call(self, images):
        outputs = self.conv_model(images)
```

```
        # After passing the images through our mini-network the spatial
dimensions
        # are flattened to form sequences.
        reshaped = tf.reshape(
            outputs,
            (-1, tf.shape(outputs)[1] * tf.shape(outputs)[2],
tf.shape(outputs)[-1]),
        )
        return reshaped

    def positional_embedding(self, img_size, img_channels):
        # Positional embeddings are optional in CCT. Here, we calculate
        # the number of sequences and initialize an `Embedding` layer to
        # compute the positional embeddings later.
        if self.positional_emb:
            dummy_inputs = tf.ones((1, img_size, img_size, img_channels))
            dummy_outputs = self.call(dummy_inputs)
            sequence_length = tf.shape(dummy_outputs)[1]
            projection_dim = tf.shape(dummy_outputs)[-1]

            embed_layer = layers.Embedding(
                input_dim=sequence_length, output_dim=projection_dim
            )
            return embed_layer, sequence_length
        else:
            return None

"""## Stochastic depth for regularization"""

# Referred from: github.com:rwightman/pytorch-image-models.
class StochasticDepth(layers.Layer):
    def __init__(self, drop_prop, **kwargs):
        super(StochasticDepth, self).__init__(**kwargs)
        self.drop_prob = drop_prop
```

```
    def call(self, x, training=None):
        if training:
            keep_prob = 1 - self.drop_prob
            shape = (tf.shape(x)[0],) + (1,) * (len(x.shape) - 1)
            # shape = (tf.shape(x)[0],) + (1,) * (len(tf.shape(x)) - 1)
            random_tensor = keep_prob + tf.random.uniform(shape, 0, 1)
            random_tensor = tf.floor(random_tensor)
            return (x / keep_prob) * random_tensor
        return x
#TypeError: len is not well defined for symbolic Tensors. (Shape_1:0)
#Please call x.shape rather than len(x) for shape information.

"""## MLP for the Transformers encoder"""

def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x

"""## Data augmentation"""

# Note the rescaling layer. These layers have pre-defined inference
behavior.
data_augmentation = keras.Sequential(
    [
        layers.Rescaling(scale=1.0 / 255),
        layers.RandomCrop(img_size, img_size),
        layers.RandomFlip("horizontal"),
    ],
    name="data_augmentation",
)
```

```
"""# The final CCT model"""

def create_cct_model(
    img_size=img_size,
    img_channels=img_channels,
    input_shape=input_shape,
    num_heads=num_heads,
    projection_dim=projection_dim,
    transformer_units=transformer_units,
):

    inputs = layers.Input(input_shape)

    # Augment data.
    # augmented = data_augmentation(inputs)

    # Encode patches.
    cct_tokenizer = CCTTokenizer()
    # encoded_patches = cct_tokenizer(augmented)
    encoded_patches = cct_tokenizer(inputs)

    # Apply positional embedding.
    if positional_emb:
        pos_embed, seq_length =
cct_tokenizer.positional_embedding(img_size, img_channels)
        positions = tf.range(start=0, limit=seq_length, delta=1)
        position_embeddings = pos_embed(positions)
        encoded_patches += position_embeddings

    # Calculate Stochastic Depth probabilities.
    dpr = [x for x in np.linspace(0, stochastic_depth_rate,
transformer_layers)]

    # Create multiple layers of the Transformer block.
```

```
    for i in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-5)(encoded_patches)

        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)

        # Skip connection 1.
        attention_output = StochasticDepth(dpr[i])(attention_output)
        x2 = layers.Add()([attention_output, encoded_patches])

        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-5)(x2)

        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)

        # Skip connection 2.
        x3 = StochasticDepth(dpr[i])(x3)
        encoded_patches = layers.Add()([x3, x2])

    # Apply sequence pooling.
    representation = layers.LayerNormalization(epsilon=1e-
5)(encoded_patches)
    attention_weights = tf.nn.softmax(layers.Dense(1)(representation),
axis=1)
    weighted_representation = tf.matmul(
        attention_weights, representation, transpose_a=True
    )
    weighted_representation = tf.squeeze(weighted_representation, -2)

    # Classify outputs
```

```
    #logits = layers.Dense(num_classes)(weighted_representation)
    logits = layers.Dense(num_classes,
activation='sigmoid')(weighted_representation)
    # Create the Keras model.
    model = keras.Model(inputs=inputs, outputs=logits)
    return model

model = create_cct_model()

#optimizer = tfa.optimizers.AdamW(learning_rate=learning_rate,
weight_decay=weight_decay)
optimizer = Adam(learning_rate=learning_rate)
#optimizer = RMSprop(learning_rate=0.01)

model.compile(optimizer=optimizer,
loss=keras.losses.BinaryCrossentropy(from_logits=False),
metrics=[auc_metric])

#model.compile(optimizer=optimizer,
loss=keras.losses.CategoricalCrossentropy(from_logits=True,
label_smoothing=0.1),  metrics=[auc_metric])

# model.summary()

#keras.utils.plot_model(model, show_shapes=True)

"""# Training"""

STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size

#%tensorboard --logdir logdir

history = model.fit(
```

```
    x=train_ds,
    # initial_epoch = 109,
    steps_per_epoch=STEP_SIZE_TRAIN,
    validation_data=valid_ds,
    validation_steps=STEP_SIZE_VALID,
    callbacks=my_callbacks,
    epochs=no_of_epochs)

model.save(str(save_path) + model_name)

"""# History"""

history.history.keys()

df = pd.DataFrame({'epochs'       : history.epoch,
                   'auc'     : history.history['auc'],
                   'val_auc' : history.history['val_auc'],
                   'loss'         : history.history['loss'],
                   'val_loss'     : history.history['val_loss'],
                   'lr'           : history.history['lr']})
sns.set(rc = {'figure.figsize':(15,8)})
g = sns.pointplot(x="epochs", y="val_auc", data=df, fit_reg=False,
color='green')
g = sns.pointplot(x="epochs", y="auc", data=df, fit_reg=False)

g = sns.pointplot(x="epochs", y="val_loss", data=df, fit_reg=False,
color='orange')
g = sns.pointplot(x="epochs", y="loss", data=df, fit_reg=False,
color='yellow')

g = sns.pointplot(x="epochs", y="lr", data=df, fit_reg=False,
color='red')

"""# Load model"""
```

```
# modelToLoad = 'colab_multi_male_CCT_6__160__amount_1_short'
# model =
keras.models.load_model(Path(r"/content/drive/MyDrive/mag/models/multi/"+
modelToLoad))
#model.summary()

"""# TEST model"""

def printer_y(yId):
  print(orig_test_labels.iloc[yId])
  print(trans_y[yId])
  print(preds_prob[yId])
  print(preds[yId])

# Get predictions
preds_prob = model.predict(test_generator, verbose=1)
preds=np.argmax(preds_prob, axis=1)
orig_test_labels = test_df_no_empty.feature_string
orig_test_labels_int = [targets.index(x[0]) for x in orig_test_labels]
print(preds_prob.shape)
print(orig_test_labels.shape)

# predictions with threshold
# індекси значень, більших за поріг
# [рядок, індекс значення]
# вектор порогів -- щоб по кожному класу окремий поріг
def threshold_predictions(pred_threshold):
  pred_threshold = np.asarray(pred_threshold)
  shape = (num_classes,)
  pred_threshold = np.reshape(pred_threshold, (pred_threshold.size))
  if pred_threshold.size != num_classes:
    pred_threshold = np.full(shape, pred_threshold[0], dtype=float)
  arg_preds = np.argwhere(preds_prob > pred_threshold)
```

```
  preds_int = np.zeros(shape=preds_prob.shape, dtype=np.int8)
  for pred in arg_preds:
    preds_int[pred[0]][pred[1]] = 1
  return preds_int

generator_classes = np.array([x for x in
train_generator.class_indices.keys()])
generator_classes_list = [x for x in
train_generator.class_indices.keys()]
y = orig_test_labels
trans_y = MultiLabelBinarizer(classes=generator_classes).fit_transform(y)
arg_y = np.argwhere(trans_y == 1)
count_y_all = np.count_nonzero(trans_y)
count_y = np.count_nonzero(trans_y, axis = 0)
print(trans_y.shape)

# TP, FP
preds_thresholded = preds_int
count_preds = np.count_nonzero(preds_thresholded, axis = 0)
# [all_y, all_predicted as true, TP, FP]
tfpr = np.zeros((num_classes,4), dtype=np.int32)
for y in arg_y:
  if(preds_thresholded[y[0]][y[1]] == 1):
    tfpr[y[1]][2] += 1 #count TP

for class_ind in range(num_classes):
  tfpr[class_ind][0] = count_y[class_ind] # all_y
  tfpr[class_ind][1] = count_preds[class_ind] # all_predicted as true
  tfpr[class_ind][3] = tfpr[class_ind][1] - tfpr[class_ind][2] # FP

# Precision #Recall #F1 score
PRF = np.zeros((num_classes,3), dtype=np.float32)
for class_ind in range(num_classes):
  PRF[class_ind][0] = (tfpr[class_ind][2] / tfpr[class_ind][1])
```

```
  PRF[class_ind][1] = (tfpr[class_ind][2] / tfpr[class_ind][0])
  PRF[class_ind][2] = 2 * ((PRF[class_ind][0] * PRF[class_ind][1]) /
(PRF[class_ind][0] + PRF[class_ind][1]))

preds_int = threshold_predictions(0.5)
preds_thresholded = preds_int
axis_classes = generator_classes
print(classification_report(y_true=trans_y, y_pred=preds_thresholded,
target_names=generator_classes))

preds_int = threshold_predictions(0.25)
preds_thresholded = preds_int
axis_classes = generator_classes
print(classification_report(y_true=trans_y, y_pred=preds_thresholded,
target_names=generator_classes))

preds_int = threshold_predictions(0.1)
preds_thresholded = preds_int
axis_classes = generator_classes
print(classification_report(y_true=trans_y, y_pred=preds_thresholded,
target_names=generator_classes))

y_test = trans_y
y_prob = preds_prob
macro_roc_auc_ovo = roc_auc_score(y_test, y_prob, multi_class="ovo",
average="macro")
weighted_roc_auc_ovo = roc_auc_score(
    y_test, y_prob, multi_class="ovo", average="weighted"
)
macro_roc_auc_ovr = roc_auc_score(y_test, y_prob, multi_class="ovr",
average="macro")
weighted_roc_auc_ovr = roc_auc_score(
    y_test, y_prob, multi_class="ovr", average="weighted"
)
```

```
print(
    "One-vs-One ROC AUC scores:\n{:.6f} (macro),\n{:.6f} "
    "(weighted by prevalence)".format(macro_roc_auc_ovo,
weighted_roc_auc_ovo)
)
print(
    "One-vs-Rest ROC AUC scores:\n{:.6f} (macro),\n{:.6f} "
    "(weighted by prevalence)".format(macro_roc_auc_ovr,
weighted_roc_auc_ovr)
)

from itertools import cycle
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
#from scipy import interp #scipy.interp is deprecated and will be removed
in SciPy 2.0.0, use numpy.interp instead

y_test = trans_y
y_score = preds_prob
n_classes = len(targets)

fig_size_x = 15
fig_size_y = 10
lw = 2
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
```

```
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(),
y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(figsize=(fig_size_x, fig_size_y))
plt.plot(
    fpr["micro"],
    tpr["micro"],
    label="micro-average ROC curve (area =
{0:0.2f})".format(roc_auc["micro"]),
    color="deeppink",
    linestyle=":",
    linewidth=4,
)

plt.plot(
    fpr["macro"],
    tpr["macro"],
```

```
    label="macro-average ROC curve (area =
{0:0.2f})".format(roc_auc["macro"]),
    color="navy",
    linestyle=":",
    linewidth=4,
)

colors = cycle(["aqua", "darkorange", "cornflowerblue", "green", "red"])
for i, color in zip(range(n_classes), colors):
    plt.plot(
        fpr[i],
        tpr[i],
        color=color,
        lw=lw,
        label="ROC curve of class {0} ({1}) (area = {2:0.2f})".format(i,
generator_classes[i], roc_auc[i]),
    )

plt.plot([0, 1], [0, 1], "k--", lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Some extension of Receiver operating characteristic to
multiclass")
plt.legend(loc="lower right")

print(roc_auc)
plt.show()
plt.close()

import matplotlib.pyplot as plt
from itertools import cycle
# for PR curve
```

```
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score

# For each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(trans_y[:, i],
preds_prob[:, i])
    average_precision[i] = average_precision_score(trans_y[:, i],
preds_prob[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(
    trans_y.ravel(), preds_prob.ravel()
)
average_precision["micro"] = average_precision_score(trans_y, preds_prob,
average="micro")
average_precision["macro"] = average_precision_score(trans_y, preds_prob,
average="macro")

# setup plot details
colors = cycle(["aqua", "darkorange", "cornflowerblue", "green", "red"])

_, ax = plt.subplots(figsize=(15, 10))

f_scores = np.linspace(0.2, 0.8, num=4)
lines, labels = [], []
for f_score in f_scores:
    x = np.linspace(0.01, 1)
    y = f_score * x / (2 * x - f_score)
    (l,) = plt.plot(x[y >= 0], y[y >= 0], color="gray", alpha=0.2)
```

```
    plt.annotate("f1={0:0.1f}".format(f_score), xy=(0.9, y[45] + 0.02))

display = PrecisionRecallDisplay(
    recall=recall["micro"],
    precision=precision["micro"],
    average_precision=average_precision["micro"],
)
display.plot(ax=ax, name="Micro-average precision-recall", color="gold")

for i, color in zip(range(num_classes), colors):
    display = PrecisionRecallDisplay(
        recall=recall[i],
        precision=precision[i],
        average_precision=average_precision[i],
    )
    display.plot(ax=ax, name="Precision-recall for
{0}".format(targets[i]), color=color)

# add the legend for the iso-f1 curves
handles, labels = display.ax_.get_legend_handles_labels()
handles.extend([l])
labels.extend(["iso-f1 curves"])
# set the legend and the axes
ax.set_xlim([0.0, 1.0])
ax.set_ylim([0.0, 1.05])
ax.legend(handles=handles, labels=labels, loc="best")
ax.set_title("Extension of Precision-Recall curve to multi-class")

print(average_precision)
plt.show()
```