**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**КАФЕДРА КОМП'ЮТЕРНИХ НАУК**

# КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

**на тему:**

**«Інформаційна технологія розпізнавання шкіряних захворювань за зображенням»**

**Завідувач**
**випускаючої кафедри** **Довбиш А.С.**

**Керівник роботи** **Москаленко В.В.**

**Студента групи ІН.м – 81н** **Слєпченко Д.М.**

**СУМИ 2020**

*Сумський державний університет*
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Комп'ютерні науки»*

Затверджую:
зав.кафедрою *Довбиш А.С.*
_____
"_____" _____ 20___р.

# ЗАВДАННЯ
# НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

*Слєпченку Дмитру Миколайовичу*
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія розпізнавання шкіряних захворювань за зображенням*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) *звіт з переддипломної практики, наукові публікації, статті, технічна документація та перелік літературних джерел з матеріалами опису*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) аналіз проблеми та постановка задачі; 2) опис інформаційної технології класифікаційного аналізу дермоскопічних зображень; 3) програмна реалізація та аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
*69 сторінок, 15 рисунків, 5 табл., 1 додаток, 20 джерел*

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів дипломного проекту (роботи) | Термін виконання проекту (роботи) | Примітка |
|---|---|---|---|
| *1.* | *Аналіз проблеми та постановка задачі.* | | |
| *2.* | *Опис інформаційної технології класифікаційного аналізу дермоскопічних зображень.* | | |
| *3.* | *Програмна реалізація та аналіз результатів.* | | |
| *4.* | *Оформлення пояснювальної записки до дипломної роботи.* | | |

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

# РЕФЕРАТ

**Записка:** 69 стор., 15 рис., 5 табл., 1 додаток, 20 джерел.

**Об’єкт дослідження** — процес класифікаційного аналізу шкіряних захворювань на дермоскопічних зображеннях.

**Мета роботи** — розробка інформаційної технології класифікаційного аналізу шкіряних захворювань.

**Методи дослідження** — методи машинного навчання глибоких нейронних мереж, методи валідації моделей детектування та класифікації об'єктів інтересу на зображеннях.

**Результати** — розроблено модель, алгоритм функціонування та програмну реалізацію інформаційної технології класифікаційного аналізу шкіряних захворювань на дермоскопічних знімках. В основі даної технології лежить використання глибоких згорткових нейронних мереж з детектуючим шаром YOLO. Розроблений алгоритм реалізовано у формі скриптів на мові програмування Python 3.

КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, НАВЧАЛЬНА ВИБІРКА, ДЕТЕКТУВАННЯ, МАШИННЕ НАВЧАННЯ, ФУНКЦІЯ ВТРАТ, ВАЛІДАЦІЯ

**ЗМІСТ**

# ВСТУП

У порівнянні з іншими формами раку шкіри злоякісна меланома зустрічається порівняно рідко, однак захворюваність на меланому зростає швидше, ніж будь-який інший вид раку, і саме вона є причиною більшості випадків смерті від раку шкіри. У пацієнтів з меланомою стадії I виживання становить більше 95% протягом 5 років у порівнянні зі стадією IV, де рівень виживання знаходиться в діапазоні 8 -25%, що обумовлює актуальність ранньої діагностики захворювань шкіри.

Людина з підозрілим пігментованим ураженням шкіри проходить кілька етапів перед остаточною постановкою діагнозу меланоми: самооцінка, оцінка лікаря первинної медичної допомоги, оцінка фахівця, а також ексцизія і оцінка за гістопатологією. Сучасні практичні керівництва рекомендують відповідним чином навченим медичним працівникам оцінювати всі підозрілі ураження шкіри за допомогою дермоскопії. Методи підвищення точності діагностики можуть використовуватися на кожному етапі для поліпшення диференціації між нешкідливими і потенційно небезпечними ураженнями, тим самим знижуючи навантаження на наступні служби у зв'язку з ураженнями, які були невиправдано перенаправлені далі. Група Кокрейн з діагностики раку шкіри нещодавно опублікувала комплексну серію перегляду точністних характеристик різних діагностичних методів, включаючи візуальну оцінку за дерматоскопією або без неї, рефлекторну конфокальну мікроскопію, теле-дерматологію, методи комп'ютерної діагностики,  а також додатки для смартфонів. Було встановлено, що дермоскопія підвищує діагностичну точність в порівнянні з візуальним оглядом, а рефлекторна конфокальна мікроскопія є більш точною, ніж дермоскопія. Проте дані, що підтверджують ефективність та широке поширення теледерматології, методів діагностики за допомогою комп'ютерів або смартфонів є обмежені або неякісні [1]. Точність клінічної діагностики також

залежить від досвіду експертів, а обладнання, необхідне для рефлекторної конфокальної мікроскопії, є дорогим.

Було випущено велику кількість додатків для виявлення меланоми за допомогою смартфонів, хоча є мало свідчень щодо клінічної валідації розглянутих додатків для діагностики раку шкіри: 19 були сфотографовані за допомогою смартфона, а 4 надали оцінку ймовірності виникнення злоякісної пухлини, але жодне з них не було оцінено з точки зору діагностичної точності [2]. Користувацькі програми, які погано розроблені, неточними або вводять в оману, можуть завдати шкоди пацієнтам.

Однак сучасні досягнення в обчислювальній техніці та алгоритмах обробки даних відкривають перспективи щодо підвищити точність медичної діагностики. Існуючі алгоритми штучного інтелекту (ШІ) здатні розпізнавати зображення і відносити їх до одного з класів алфавіту, розмір якого може перевищувати декілька тисяч. При цьому існує багато прикладів успішного використання глибокого машинного навчання в медичній діагностиці [3]. Основною умовою застосування алгоритмів глибокого машинного навчання є великий обсяг розмічених навчальних даних.

Метою даного дослідження є розробка та оцінка ефективності алгоритму розпізнавання онкологічних захворювань шкіри за дермоскопічним зображенням біопсивних та небіопсивних пігментованих зразків шкіри. За основу пропонується взяти сучасні методи глибокого машинного навчання та методи валідації на розміченій тестовій вибірці. Розробка є перспективною для впровадження в клініках дерматології та пластичної хірургії.

# 1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Аналіз проблеми

Рак шкіри — один з найпоширеніших видів раку в світі. Існують різні типи раку шкіри, такі як базальноклітинна карцинома, меланома, інтраепітеліальна карцинома, плоскоклетинна карцинома і ін. Шкіра людини складається з трьох тканин, званих дермою, епідермісом і підшкірної клітковиною. В епідермісі є меланоцити, які за певних умов можуть виробляти меланін з досить незвичайною швидкістю. Наприклад, тривалий вплив сильного ультрафіолетового випромінювання сонячного світла викликає вироблення меланіну. Незвичайне зростання меланоцитів викликає меланому — смертельний тип раку шкіри. Згідно оцінок річного звіту Американського товариства з боротьби з раком протягом 2019 р. зареєстровано близько 96 480 нових випадків меланоми і 7230 осіб помруть від цієї хвороби [4]. Меланома також вважається найбільш смертоносним типом раку шкіри з рівнем смертності 1,62% серед інших видів раку шкіри. Рання діагностика меланоми дуже важлива з точки зору лікування. Якщо меланому діагностувати на ранніх стадіях, то частка пацієнтів, що виживають протягом п’яти років, становить 92%. Однак візуальна схожість між доброякісними і злоякісними ураженнями шкіри є основною проблемою для виявлення меланоми. Саме цьому діагностика меланоми може бути ускладнена навіть для добре підготованого фахівця. Визначити тип ураження неозброєним оком — дуже складне завдання. Тому протягом багатьох років використовуються різні методи візуалізації, одним з яких є дермоскопія. Дермоскопія — неінвазивний метод візуалізації, що дозволяє візуалізувати поверхню шкіри за допомогою пігменту. Вона є однією з найбільш поширених методик візуалізації в дерматології і з досвіду спостерігача збільшила ефективність діагностики злоякісних новоутворень на 50%. Однак використання тільки людського зору для виявлення меланоми на дерматологічних

зображеннях може бути неефективним внаслідок суб'єктивності, втоми чи недостатнього рівня досвіду у дерматолога. Точність діагностики меланоми за дермоскопічним зображенням у недосвідченого спеціаліста становить від 75% до 84%. Для подолання труднощів, що виникають під час діагностики меланоми, необхідні системи комп'ютерної діагностики (CAD), що допомагають фахівцям в постановці діагнозу. В CAD-системах існує чотири етапи ідентифікації ураження шкіри меланомою: попередня обробка, сегментація, виділення ознак і класифікація. Для надійної ідентифікації меланоми сегментація ураження є фундаментальним етапом в CAD-системах. Однак етап сегментації є складним процесом через великі відмінностей в кольорі, текстурі, розміщенні і розмірах уражень шкіри на дермоскопічних зображеннях. Крім того, низький контраст зображення перешкоджає диференціації суміжних тканин. Крім того, додаткові фактори, такі як повітряні бульбашки, волосся, ебенові рамки, лінійки, кровоносні судини і кольорове підсвічування підвищують складність сегментації ураження. На рисунку 1.1.1 показано кілька зразків дермоскопічних зображень з набору даних з різними артефактами. Перший ряд зображень на рисунку 1.1.1 відповідає артефактам спричинених волосям, маркером чорнил та темних країв. Другий ряд зображені відповідає артефактам, що спричинені маркерами лінійки, гелевими бульбашками і засвічуванням.

Для підвищення ефективності діагностики меланоми у дермоскопії використовується неінвазивна методика візуалізації шкіри, що дозволяє отримати збільшене і освітлене зображення ділянки шкіри, тобто зображення підвищеної чіткості пігментних плям на шкірі. Дермоскопія широко використовується в діагностиці меланоми і забезпечує набагато більшу точність, ніж огляд неозброєним оком. Тож в цілому роз’язання задачі діагностування полягає у застосуванні методів попередньої обробки зображень для їх нормалізації, екстрації ознакового опису, локалізації і класифікації місця ураження шкіри.

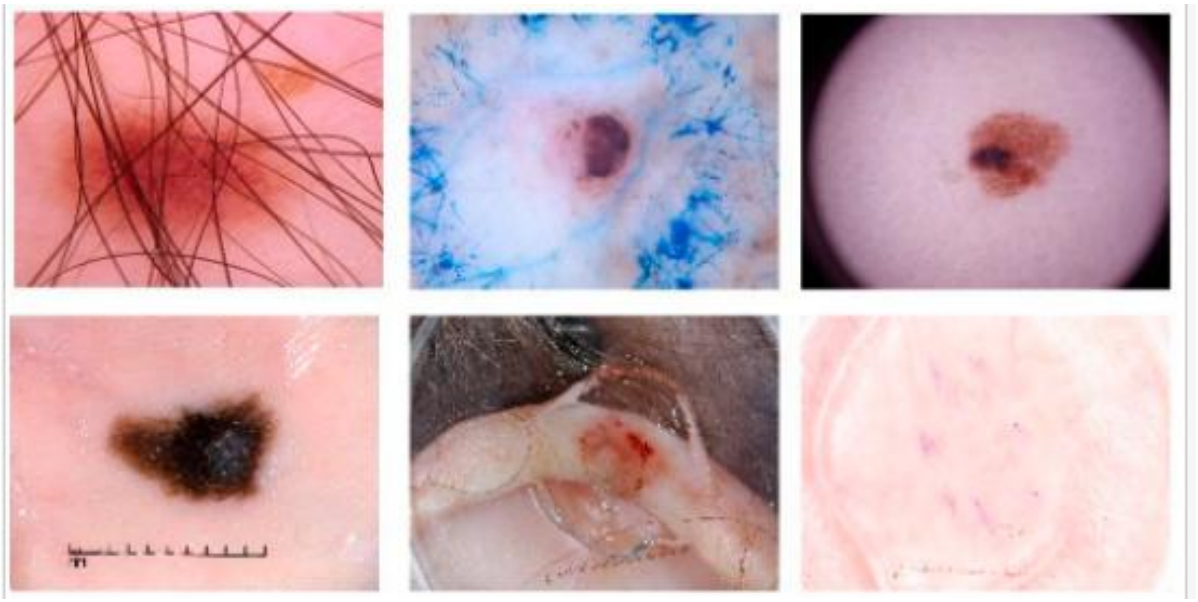

Рисунок 1.1.1 — Приклади дермоскопічних зображень з артефактами

Існуючі підходи, основані на технологіях машинного зору та машинного навчання, здатні вирішувати задачі всіх етапів комп’ютерної діагностики. Розроблено велику кількість архітектур глибоких нейронних мереж, методів і протоколів їх начання. Однак в галузі діагностики уражень шкіри покищо досліджувалися або нейронні мережі з входом низької роздільної здатності, або мережі невеликої інформаційної ємності. Тому дослідежння ефективності розпізнавання дермоскопічних зображень з використанням нових моделей і методів технології глибокого машинного навчання є актуальним і перспективним.

### 1.2 Підходи до локалізації об’єктів інтересу на зображенні

Глибоке навчання набуло популярності в медичних дослідженнях, включаючи Магнітно-резонансну томографію (МРТ) мозоку, ультразвукову діагностику раку грудей, розпізнавання стадії діабету, сегментацію виразки стоп,

тощо. U-Net є популярним підходом глибокого навчання в біомедичній візуалізації. В U-Net використовуються аугментовані дані, в тому числі отримані шляхом мякої деформації, для підвищення ефективності навчання за умов обмеженого обсягу анотованих зображень.

Дослідники зробили значний внесок, розробивши велику кількість різноманітних методи глибокого навчання для аналізу біологічних тканин і об’єктів. Набули широкого використання глибокі згортокові мережі з більш ніж 50 шарами для двоетапної процедури сегментації об’єктів інтересу на зображенні з подальшою класифікацією. Було доведено, що більш глибокі мережі створюють більш інформативне ознакове подання для задач розпізнавання образів. Досить відомими аріхітектурами мереж є з 16-шарова марежа VGG-16, 22-шарова мережа GoogleNet та 50-шарова ResNet-50. Дані мережі показують багатообіцяючі результати в різних задачах медичної діагностики та візуалізації, однак їх функціонування в режимах навчання та екзамену є обчислювально дорогим. Для зниження обчислювальної складності глибоких моделей були запропоновані багатошарові повністю згорткову мережі (FCNs), для яких було підтверджено ефективність в задачі сегментації зображень. Подальші удосконалення стосувалися розробки багатоетпаних і багатоступінчастих моделей обробки даних, суть яких полягала у навчанні моделі грубій локалізації об’єкта інтересу на зображенні на ранніх етапах і в деталізації меж виділених об’єктів на пізнії стадіях обробки та аналізу.

### 1.2.1 Регіональна сегментація

Порогова сегментація є найпростішим методом сегментації зображень, а також одним з найбільш поширених паралельних методів сегментації. Це загальний алгоритм сегментації, який здійснює безпосереднє розділення шляхом обробки візуальної інформації в форматі сірого зображення на основі значення порогових значень сірого для різних об’єктів. Порогова сегментація може бути

розділена на локальний пороговий метод і глобальний пороговий метод. Глобальний пороговий метод ділить зображення на дві області інтересу і фон одним пороговим методом. Локальний пороговий метод вимагає вибору декількох порогів сегментації і ділить зображення на кілька зон інтересу і фон з кількома пороговими значеннями.

Найбільш поширеним алгоритмом порогової сегментації є метод найбільшої міжкласової дисперсії, який вибирає глобально оптимальний поріг шляхом максимізації дисперсії між класами. Крім того, існує метод порогової сегментації на основі ентропії, метод мінімальної похибки, метод матриці кооперації, метод збереження моменту, простий статистичний метод, метод релаксації ймовірності, метод нечітких множин та порогові методи, поєднані з іншими методами [6]. Перевагою порогового методу є простота розрахунку і швидкість роботи. Зокрема, коли об’єкти інтересу і фон мають високий контраст, можна отримати ефект сегментації. Недоліком є те, що важко отримати точні результати сегментації зображення, коли на зображенні немає значної різниці в шкалі сірого або великого перекриття значень шкали сірого. Оскільки цей метод враховує тільки інформацію в сірих відтінках без врахування просторової інформації зображення, він є чутливим до шумів і нерівностей шкали сірого, що обумовлює необхідність комбінування даного методу з іншими методами.

Метод регіонального зростання є типовим послідовним алгоритмом сегментації регіону і його основна ідея полягає в тому, щоб пікселі з подібними властивостями знаходилися разом для формування регіону. Метод вимагає спочатку вибрати піксель зростання, а потім об'єднати схожі пікселі навколо нього в область. На рис. 1.2.1 показано приклад обраної точки для області вирощування. На рис. 1.2.1 (а) показана необхідність поділу зображення. Обирається два стартових пікселя (помічені сірим) для регіонального зростання. Правило об’єднання пікселів полягає у включенні пікселя в область якщо абсолютне значення різниці між пікселем сірого і пикселем поруч менше певного

порогу T. На рис. 1.2.1 (б) показані результати регіонального зростання при T = 3, і всю ділянку, що чітко розділена на два регіони. На рис. 1.2.1 (в) показані результати зростання регіону при T = 6, і всю ділянку виділену як область. Таким чином, вибір порогу дуже важливий .

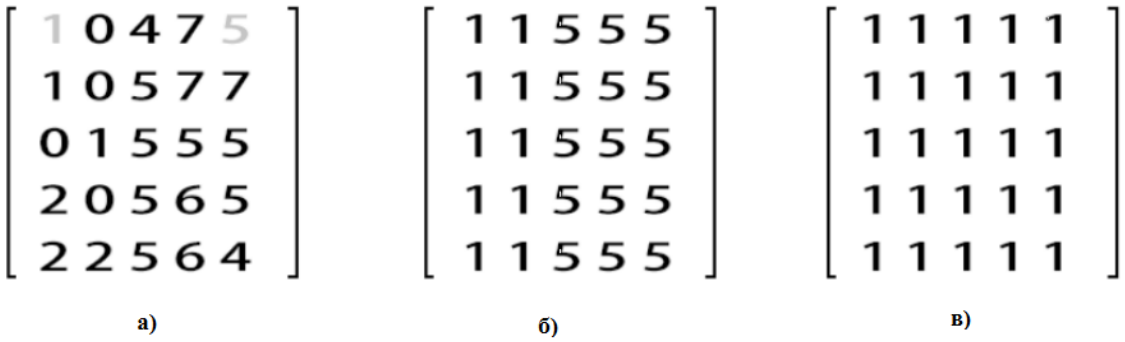


Рисунок 1.2.1 – Приклади регіонального росту

Перевага регіонального зростання полягає в тому, що він, як правило, розділяє пов'язані з ним регіони з однаковими характеристики і забезпечує чіткість меж і результатів сегментації. Ідея регіонального зростання проста і вимагає всього лише декількох стартових точок для продовження. І правила зростання в процесі вирощування можуть бути вільно задані. Також, допустимо використовувати декілька правил і критерії одночасно. Однак обчислювальна складність таких методів є високою. Крім того наявність шуму і нерівномірності градації сірого може призвести до втрати деяких регіонів і надмірного поділу. Одним з них є ефект тіні на зображенні, що є не дуже добре для даного методу.

### 1.2.2 Сегментація по границі

Край об'єкта визначається як уривчасті локальні ознаки на зображенні, тобто найбільш істотні частини зображення, що змінюють свою локальну яскравість, наприклад, значення сірого кольору мутації (колірна мутація), змінють текстуру чи тому подібне. Також можна використати розриви для

виявлення краю об’єкта і досягнення задач сегментації зображення. Між двома сусідніми областями зображення завжди є сірий край з різними значеннями сірого, і існує випадок, коли значення сірого не є безперервним. Цей розрив часто може бути виявлений за допомогою похідних операцій, а похідні можуть бути обчислені за допомогою диференціальних операторів. Паралельне виявлення границь часто виконується за допомогою диференціального оператора просторової області для виконання сегментації зображення шляхом згортки його шаблону і зображення. Паралельне визначення меж зазвичай використовується як метод попередньої обробки зображення. Широко розповсюдженими диференціальними операторами першого порядку є оператори Prewitt, Roberts і Sobel.

Оператор Sobel використовується в основному для виявлення країв, і технічно — це дискретний диференціальний оператор, який використовується для обчислення апроксимації градієнта функції яскравості зображення. Оператор Sobel є типовим оператором детектування країв на основі першої похідної. В результаті впровадження оператором аналогічної локальної операції, шум має згладжувальний ефект і таким чином можна ефективно усунути вплив шуму. Вплив оператора Sobel на положення пікселя зважується краще ніж у оператора Prewitt і оператора Roberts. Оператор Sobel складається з двох наборів матриць 3x3, які представляють собою поперечний і поздовжній шаблони і накладаються на площину зображення для отримання різниці між горизонтальною і вертикальною різницею.

Апроксимації горизонтального і вертикального градієнтів кожного пікселя зображення можна комбінувати для розрахунку розміру градієнта за такою формулою

$$\mathrm{G} = \sqrt[2]{G_x^2 + G_y^2}\,.$$

Градієнт можна розрахувати за наступною формулою

$$\theta = \arctan\left(\frac{G_y}{G_x}\right).$$

У наведеному вище прикладі, якщо вищенаведений кут Θ дорівнює нулю, то зображення має поздовжній край, а лівий край темніше правого.

Оператор Лапласа — ізотропний оператор, оператор диференціала другого порядку. Його використання є більш доречним, коли мова йде тільки про позиціювання границі об’єкту, незалежно від різниці в сірій шкалі пікселів навколо нього. Реакція оператора Лапласа на ізольовані пікселі є сильнішою, ніж на край або лінію, і тому його варто застосовувати тільки для безшумових зображень. При наявності шуму Лапласівському оператору необхідно виконати низкочастотную фільтрацію перед виявленням краю. Тому звичайний алгоритм сегментації об'єднує Лапласівський оператор з оператором згладжування для створення нового шаблону. Лапласівський оператор також є найпростішим ізотропним диференційним оператором з обертальною інваріантністю. Перетворення Лапласа двовимірної функції зображення являє собою ізотропну другу похідну, найбільш підходящу для цифрової обробки зображень.

Лапласівський оператор використовується для поліпшення ефекту розмиття, так як він відповідає моделі градієнтного спуску. Ефект дифузії часто виникає в процесі візуалізації, як показано на рисунку 1.2.2.

а) б) в)

Рисунок 1.2.2 – Приклади відображення зображень різних операторів:
а — оригінал; б — оператор Собеля; в — Лапласіан

### 1.2.3 Сегментація на основі кластеризації

Немає загальної теорії сегментації зображень. Однак з появою безлічі нових теорій і методів різних дисциплін з'явилося багато методів сегментації зображень у поєднанні з деякими специфічними теоріями і методами. Регіони в задачі сегментації можуть розгядатися як клас, тобто як колекції подібних елементів. При цьому кластеризація відповідає певним вимогам і законам класифікації речей в процесі їх аналізу. Метод кластеризації простору ознак використовується для сегментації пікселів в просторі зображення як деяких точок простору ознак. Відповідно до групування пікселів в просторі ознак, простір ознак сегментується, а потім вони картуються назад в простір вихідного зображення для отримання результату сегментації. Метод K-середніх — один з найбільш поширених алгоритмів кластеризації. Основна ідея K-середніх полягає в тому, щоб збирати вибіркові дані в різні кластери в залежності від відстані. Тож алгоритм полягає в пошуку центрів кластерів та наборів елементів кожного кластера. Як функція, що виражає якість поточного розбиття множини на К кластерів, використовується квадратичне відхилення елементів кластерів від центрів цих кластерів. Тож в процесі обчислення значенн центрів нових кластерів необхідно намагатися манімізувати цю функцію [7].

Алгоритм К-середніх складається з наступних кроків:

1) випадковий вибір К точок як центрів кластерів;

2) розрахунок відстані від кожного зразка до кожного центру кластера і віднесення кожного зразка до найближчого центру кластеризації;

3) перевірка кількості зразків в кластері (в кожному кластері повинно міститися не менше одного зразка) та зміна центру кожного кластера на усереднений вектор всіх зразків кластера;

4) повторення кроків 2 – 3 до тих пір, доки кластерний центр не припинить змінюватися або не буде досягнено заданого числа ітерацій.

На рис. 1.2.3 показано результат сегментації методом K-середніх для випадку двох кластерів (K=2), а на рис. 1.2.4 розглянуто випадок з трьома кластерами (K=3)/


Рисунок 1.2.3 — Приклади відображення різних K-середніх зображень для значення параметру k = 2 : а — оригінал; б — сегментація 1; в — сегментація 2

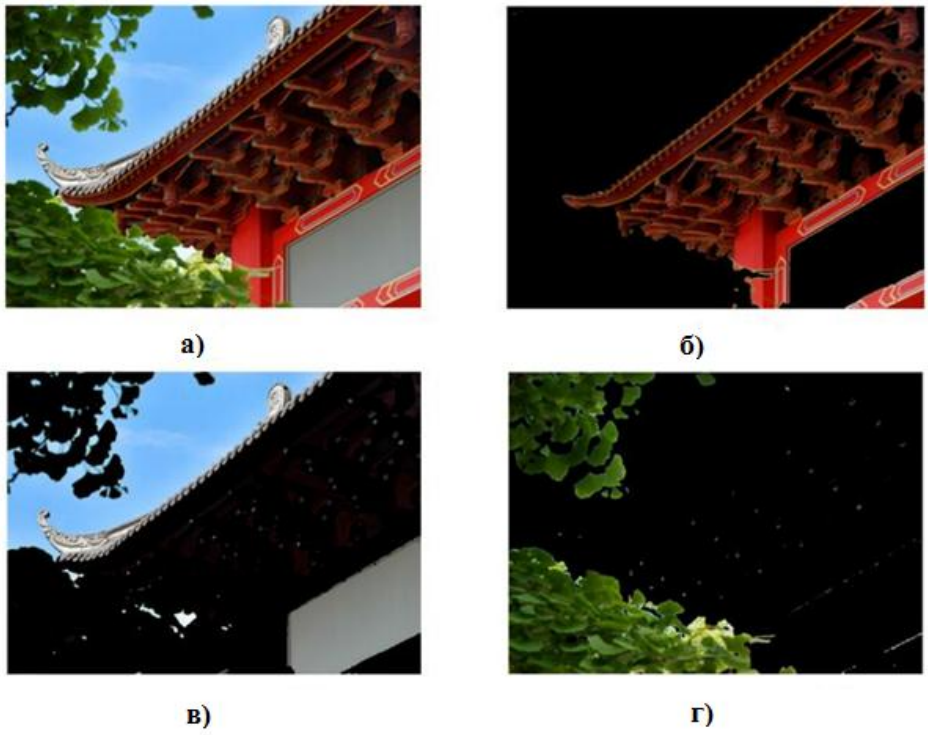


Рисунок 1.2.4 — Приклади відображення різних K-середніх зображень (k = 3): а — оригінал; б — сегментація 1; в — сегментація 2; г – сегментація 3

Перевага алгоритму кластеризації K-середніх полягає в тому, що він швидкий і простий, а також високоефективний і масштабований для великих масивів даних. А його обчислювальна складність близька до лінійної і є придатною для обробки великих масивів даних. Недоліком K-середніх є вибір кількості кластерів розбиття є слабоформалізованою задачею, оскільки немає прямих критеріїв. По-друге, зі структури алгоритму K-середніх видно, що кожна ітерація алгоритму обробляє всю вибірку, що збільшує ресурсні потреби алгоритму. Отже, алгоритм K-середніх є методом розбиття, що оснований на розрахунку відстані. Він застосовується лише до набору даних, який які можна розбити на кластери круглої форми.

### 1.2.4 Сегментація на основі згорткових мереж

В останні роки методи глибокого навчання знайшли місце в області класифікації зображень, детектування, сегментації, генерації зображень з високою роздільною здатністю і в багатьох інших областях, де привели до проривних результатів. В області сегментації зображень алгоритм глибокого навчання згорткових нейронних мереж є досить ефективним підходом.

U-Net вважається однією із стандартних архітектур згорткових нейронних мереж для задач сегментації зображень, якщо потрібно не тільки визначити клас зображення вцілому, але і сегментувати його області відповідно до класу, тобто створити маску, яка буде розділяти зображення на кілька класів. Архітектура складається з стягуючого шляху для захоплення контексту і симетричного розширюючого шляху, який дозволяє здійснити точну локалізацію.

Мережа навчається наскрізним способом на невеликій кількості зображень. Крім того, ця мережа характеризується відносно високою оперативністю функціонування в режимах навчання та екзамену. Сегментація зображення 512 × 512 займає менше секунди на сучасному графічному процесорі.

Для U-Net є характерними наступні особливості:

- досягнення високих результатів в різних реальних задачах, особливо для біомедичних застосувань;
- використання невеликої кількості даних для досягнення хороших результатів.

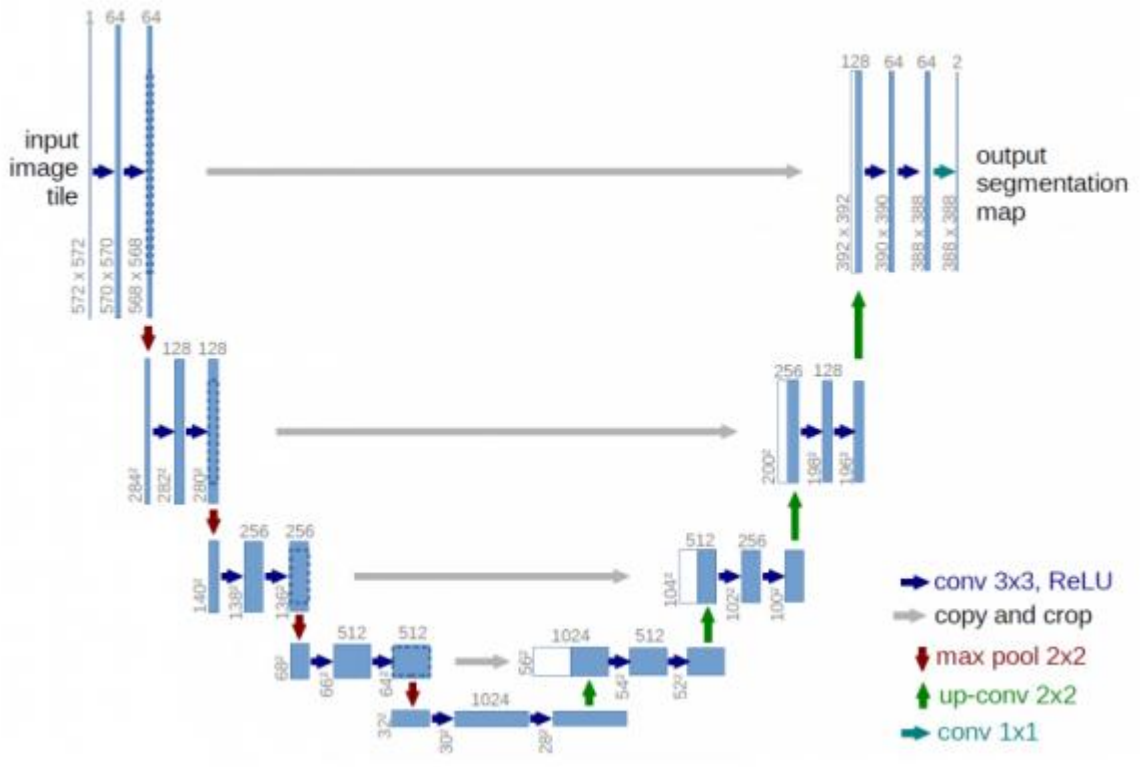


Рисунок 1.2.5 — Архітектура мережі U-Net

Архітектура мережі приведена на рис.1.2.5. Кожен синій квадрат відповідає багатоканальної карті ознак. Кількість каналів приведено у верхній частині квадрату. Розмір х-у наведено в нижньому лівому краю квадрата. Білі квадрати є копіями карти ознак. Стрілки позначають різні операції.

Архітектура складається з шляху, що звужується (зліва) і шляху, що розширюється (праворуч). Звужуючийся шлях — типова архітектурі згорткової нейронної мережі для екстрації ознак. Він складається з повторного застосування двох згорток 3 × 3, за якими слідують випрямляючий лінійний блок (ReLU) і

операція максимального об'єднання (2 × 2 зкроком 2) для зниження дозволу.

На кожному етапі понижувальної дискретизації канали ознак подвоюються. Кожен крок у розширенні шляху складається з операції, яка підвищує дискретизації карти властивостей, за якою слідують:

- згортка 2 × 2, яка зменшує кількість каналів властивостей;
- об'єднання з відповідним чином обрізаної картою ознак;
- дві 3 × 3 згортки, за якими слід випрямляючий лінійний блок.

Обрізування необхідне через втрату граничних пікселів при кожній згортці.

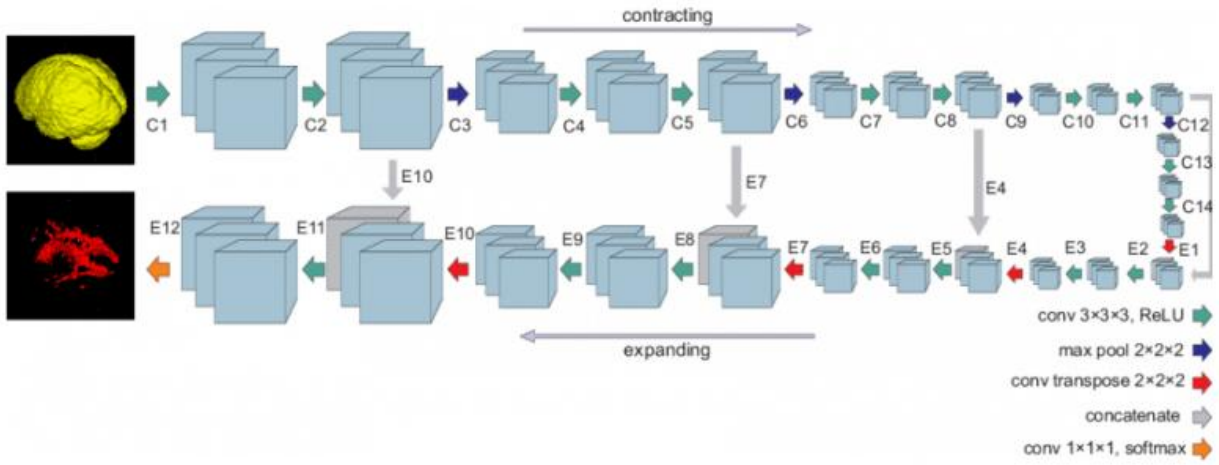


Рисунок 1.2.6 — Схема мережі U-Net

Як видно с рис. 1.2.6, на останньому шарі використовується згортка 1 × 1 для зіставлення кожного 64-компонентного вектора властивостей з бажаною кількістю класів. Всього мережа містить 23 згорткових шари. Мережа навчається методом стохастичного градієнтного спуску на основі вхідних зображень і відповідних їм карт сегментації. Архітектура U-Net має досить гарну продуктивность і точність в досить різних додатках біомедичної сегментації. Метод вимагає десятки помічених зображень для тренування і має прийнятний час навчання.

## 1.3 Методи класифікаціїї зображень

Класифікація зображень є важливим етапом у розпізнванні об'єкта інтересу та аналізі зображень. Виведення етапу класифікації зображення може бути остаточним рішенням або проміжним. До теперішнього часу було запропоновано багато методів класифікації зображень. Були проведені різні дослідження для того, щоб зробити висновок про кращу методику класифікації знімків. Складно визначити якусь одну методику як кращу серед всіх, тому що результати і їх точність залежать від ряду факторів. Протягом останніх декількох десятиліть спостерігається постійна модифікація традиційних методів, а також винахід нових методів класифікації зображень з метою отримання максимально точних результатів. Кожен з методів класифікації має свої переваги і недоліки. В даний час, дослідження зосереджені на комбінуванні бажаних характеристик цих методик з метою підвищення їх ефективності. Оскільки жорсткі класифікатори не можуть впоратися з проблемою змішаних пікселів, використовуються м'які класифікатори. Але у м'яких класифікаторів є свої недоліки.

Задача класифікації зображень полягає у віднесенні вхідних зображень, що не брали участі у навчанні класифікатора, до однієї з категорій для вимірювання точності. Під час побудови класифікаційної моделі існує проблема узагальнення даних, оскільки зображення мають високий рівень варіативності за рахунок різних точок спостереження, масштабу та освітлення, а також внаслідок різного роду викривлень та шумових артефактів.

Найбільш популярною архітектурою, використовуваною для класифікації зображень, є згорткові нейронні мережі. Згорткова нейронна мережа, як правило, починаються з вхідного "сканера", який не призначений для одночасного розбору всіх навчальних даних. Наприклад, для введення зображення розміром 100 х 100 пікселів не потрібен шар з 10 000 вузлами.

Швидше, ви створите скануючий вхідний шар розміром, скажімо, 10 х 10, який буде обробляти перші 10 х 10 пікселів зображення. Після того, як ви пропустили цей вхідний шар, ви подаєте його наступним 10 х 10 пікселям, переміщаючи сканер на один піксель вправо. Ця техніка відома як розсувні вікна.

Потім ці вхідні дані подаються через згорткові шари, а не через звичайні шари. Кожен вузол відноситься тільки до прилеглих сусідніх осередків. Ці згорткові шари також мають тенденцію до звуження в міру того, як вони стають глибшими, в основному за рахунок легко помітних факторів вхідних даних. Крім цих згорткових шарів, в них також часто присутні об'єднуючі шари. Пул — це спосіб відфільтрувати деталі: частою технікою пулу є максимальний пул, при якому ми беремо, скажімо, 2 х 2 пікселя і передаємо пікселю з найбільшою кількістю певного атрибута.

Більшість технік класифікації зображень в даний час навчені на ImageNet, наборі даних з приблизно 1.2 мільйонами навчальних зображень з високою роздільною здатністю. Тестові зображення будуть представлені без початкових анотацій (без сегментації і міток), а алгоритми повинні будуть створювати мітки з зазначенням того, які об'єкти присутні на зображеннях. Деякі з кращих існуючих методів комп'ютерного зору були випробувані на цьому наборі даних провідними групами комп'ютерного зору з Оксфорда, INPIA і XRCE. Як правило системи комп'ютерного зору використовують складні багатоступінчасті конвеєри обробки зображень, де окремі етапи обробки настроюються вручну або шляхом оптимізації ряду параметрів.

Метод паралелепіпедів використовується для класифікації всіх пікселів зображення. Заключається цей метод в розпределенні пікселів зображення до еталонних класів. В якості еталонних характристик класів задаються деякі інтервали значень яскравості. Межі паралелепіпеда для класів визначаються за мінімальною і максимальною кількістю пікселів в конкретному класі. Ці межі допомагають привласнити піксель конкретного класу. Класифікатор навчається

з аналізу гістограм окремих спектральних складових навчальних зразків. Ця техніка має багато переваг:

- простота в розумінні і реалізації;

- висока швидкість функціонування цього класифікатора.

Незважаючи на значні переваги, у нього є багато недоліків, через які він практично не використовується. Цими недоліками є:

- можуть існувати значні прогалини між паралелепіпедами, і пікселі всередині цього регіону залишаться некласифікованими;

- іншим недоліком є те, що не враховуються попередні ймовірності членства в класі.

Також використовується класифікатор мінімальної відстані, який являє собою контрольовану техніку класифікації зображень, при якій пікселі класифікуються на основі їх відстані від середніх спектрів визначених класів [9]. У цьому методі спочатку розраховується середній вектор для кожного класу на основі навчального набору даних. Потім за допомогою алгоритму "Мінімальна відстань" обчислюється евклідова відстань кожного з некласифікованих пікселів від середнього вектора. Потім пікселю присвоюється клас з мінімальною відстанню. Відстань ($d_{cx}$) конкретного пікселя від різних середніх векторів класу (x) зазвичай обчислюється за допомогою евклідової відстані

$$d_{cx} = \|x_i - c_i\|.$$

Цей тип класифікатора математично простий і тому обчислювально менш складний. Він вимагає найменшого часу для обчислень серед всіх інших контрольованих методів класифікації. Недоліком цієї методики є те, що вона враховує тільки середнє значення, і тому вона менш ефективна, ніж техніка максимальної ймовірності.

Максимальна ймовірність — це контрольована методика класифікації зображень, при якій значення ймовірності пікселів враховується при класифікації пікселів. У цьому методі обчислюється ймовірність кожного

пікселя, що належить до класу. Потім ці значення порівнюються. Піксель присвоюється класу, в якому значення ймовірності є найбільшим. В даному методі передбачається, що всі вхідні смуги мають нормальний розподіл

$$L_k(x) = (e^{-(x-uk)^2}) \sum k^{-1} ((x - u_{k)}/2)/(2\pi^{n/2}) \left|\sum k\right|$$

де, функція ймовірності членства X належить до класу k, $x = (x_1, x_2 \dots x_n)^T$ є вектором пікселів з n групами, $uk = (u_1, u_2 \dots u_n)^T$ — середнє з k-го класу. $\sum k$ — матриця дисперсії-коваріацій. Цей метод досить ефективний, коли мова йде про класифікацію знімків мікроскопу, особливо багатоспектральних. Незважаючи на те, що це ефективний метод, він вимагає великого обчислювального часу.

Support Vector Machine, також відомий як SVM є непараметричним класифікатором. Метод опорних векторів є двійковим класифікатором і розділяє класи, використовуючи лінійну границю. Цей класифікатор передбачає, що немає ніякої попередньої інформації про те як класифікувати дані. Це оптимізує використання навчання даних, що є найбільшою перевагою цього класифікатора перед іншими класифікаторами, такими як наприклад класифікатор максимальної ймовірності. Реальна сила SVM полягає в ілюстрації ядра, так як це полегшує нелінійне картування вхідного простору в простір ознак. Отже, вибір функції ядра — це найзначніший крок в Support Vector Machine. Деякі з часто використовуваних функції ядра:

- лінійне ядро ;
- поліноміальне ядро;
- гауссівське ядро;

Так як SVM оптимізує використання тренувальних даних, то це збільшує швидкість класифікатора в значній мірі. Це також мінімізує помилки

класифікації, які можуть виникнути через попередні припущення щодо неконтрольованих даних.

Основні переваги SVM:

- вона має відмінну узагальнюючу здатність;

- вона не стикається з проблемою перенасищення;

- використовує нелінійне перетворення.

Недоліки цього класифікатора:

- структура алгоритму складна і тому її важко зрозуміти;

- оптимальні параметри не можуть бути визначені легко.

Одними з найбільш ефективних технологій є штучні нейронні мережі. Вони є непараметричними класифікаторами. Структура штучних нейронних мереж черпає натхнення в нервовій системі людини. Основною одиницею цього типу мереж є уніфікований рудимент обробки, відомий як нейрон. Кожен нейрон має два етапи — навчання і використання фази [10]. На етапі навчання нейрон вчиться виконувати операцію, а на етапі тестування він використовує тренувальну інформацію для передбачення виходу. Як правило, ці нейронні мережі використовуються для того, щоб виявити об'єктивні тенденції або закономірності в даних. Штучні нейронні мережі мають ряд переваг:

- вона має високу швидкість обчислень;

- вона ефективно справляється з зашемленими входами;

- ця техніка управляється даними у міру того, як вона вчиться на тренувальних даних.

Хоча ці штучні нейронні мережі популярні, у них є деякі недоліки:

- так як вона вимагає попереднього навчання, вона забирає багато часу;

- вважається, що вона семантично погана;

- вона стикається з проблемою переоснащення.

Згорткові нейронні мережі можуть бути використані для класифікації пошкоджень шкіри двома принципово різними способами. З одного боку,

згорткова нейронна мережа, заздалегідь підготовлена на іншому великому наборі даних, наприклад, ImageNet, може бути використана в якості екстрактора ознак. У цьому випадку класифікація виконується іншим класифікатором, наприклад, k-найближчих сусідів, допоміжними векторними машинами або штучними нейронними мережами. З іншого боку, згорткова нейронна мережа може безпосередньо вивчати взаємозв'язок між необробленими пиксельними даними і мітками класу шляхом наскрізного навчання. На відміну від класичного робочого процесу, зазвичай застосовується в машинному навчанні, витяг ознак стає невід'ємною частиною класифікації і більше не розглядається як окремий, незалежний етап обробки. Якщо згорткова нейронна мережа наскрізного навчання, то дослідження можна додатково розділити на два різних підходи: вивчення моделі з нуля або трансфертне навчання.

Основною вимогою для успішного навчання глибоких моделей згорткових нейронних мереж є наявність достатньої кількості навчальних даних, позначених разом з класами. В іншому випадку існує ризик переповнення нейромережі і, як наслідок, неадекватне узагальнююче властивість мережі для невідомих вхідних даних. Існує дуже обмежена кількість загальнодоступних даних для класифікації пошкоджень шкіри. Майже у всіх опублікованих методах використовуються набори даних, які містять набагато менше 1000 навчальних точок даних на кожен навчальний клас. Для порівняння, відомі моделі згорткових нейронних мереж для класифікації зображень, такі як AlexNet, VGG, GoogLeNet або ResNet, навчаються через велику базу даних зображень ImageNet і мають понад 1000 навчальних зображень на кожен навчальний клас.

Проте, за допомогою спеціальної процедури навчання, званої трансферним навчанням, для класифікації можна використовувати потужні моделі згорткових нейронних мереж з декількома мільйонами безкоштовних параметрів, навіть якщо для навчання є лише невелика кількість даних. В цьому випадку згорткова нейронна мережа попередньо підготовлена з використанням дуже великого

набору даних, наприклад, ImageNet; потім вона використовується в якості ініціалізації згорткової нейронної мережі для відповідного завдання. Зокрема, останній повністю пов'язаний шар попередньо підготовленої моделі згорткової нейронної мережі, яка модифікується відповідно до кількості навчальних класів в актуальній класифікаційній задачі. Потім існує два варіанти вагів попередньо підготовленої згорткової нейронної мережі: тонке налаштування всіх фіч мережі або заморожування деяких з передніх шарів через проблеми з підгонкою і тонке налаштування тільки деяких задніх шарів мережі. Ідея цієї методики полягає в тому, що передні шари мережі містять більш загальні функції (наприклад, детектори країв або кольорових плям), які корисні для багатьох завдань, але задні шари згорткової нейронної мережі стають все більш специфічними для деталей класів, що містяться в вихідному наборі даних.

### 1.4 Постановка задачі

Нехай $D_{train}^{cls} = \{ x^{(j)}, y^{(j)} \mid j = \overline{1, n_1} \}$ та $D_{test}^{cls} = \{ x^{(j)}, y^{(j)} \mid j = \overline{1, n_2} \}$ є наборами навчальних та тестових даних, де $x^{(j)}$ – $j$-те N-вимірне спостереження у вигляді зображення 100x100 пікселів, $y^{(j)}$ – мітка класу $j$ –го спостереження, $n_1$, $n_2$ – обсяги навчального та тестового наборів даних відповідно. При цьому $y^{(j)} \in \{X_z^o \mid z = \overline{1, Z}\}$, де $X_z^o$ – z-й клас з заданого алфавіту класів розпізнавання $\{X_z^o \mid z = \overline{1, Z}\}$, що характеризує функціональний стан шкіри.

Необхідно:

1. На етапі машинного навчання системи розпізнавання визначити оптимальний вектор параметрів $g$ (1.4.1), що забезпечує на етапі екзамену максимум критерію ефективності

$$g^* = \arg\max_G \{ J(g) \} \qquad (1.4.1)$$

де $J_{Cls}$ — критерій функціональної ефективності класифікаційних вирішальних правил.

$G$ – допустима область значень параметрів, які впливають на екстракцію ознак і прийняття рішень.

2. На етапі екзамену, тобто безпосередньо у робочому режимі, необхідно прийняти рішення про належність вхідного зображення $x$ до одного з класів функціонального стану шкіри $\{X_z^o \mid z = \overline{1,Z}\}$.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) виконати аналіз існуючих рішень;
2) розробити математичну модель і алгоритм процесу класифікаційного аналізу зображень;
3) виконати попередню обробку навчальних даних для їх адаптації до моделі аналізу даних;
4) розробити програмну реалізацію запропонованої моделі і алгоритмів;
5) здійснити навчання і валідацію результатів на тестовій вибірці.

# 2 ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ КЛАСИФІКАЦІЙНОГО АНАЛІЗУ ДЕРМОСКОПІЧНИХ ЗОБРАЖЕНЬ

## 2.1 Архітектура екстрактора ознак

Згорткові нейронні мережі автоматично проводять екстрацію ознак розпізнавання, без необхідності ручного втручання. У медичній візуалізації існує високий попит на методи глибокого вивчення.

VGG-16 має п'ять блоків згорткових шарів з однаковою кількістю параметрів для розширення контекстного уявлення фільтрів, в яких ми змінюємо швидкість розведення цих шарів. Вихідна карта ознак буде стискатися (вихідний крок збільшується) для будь-якої стандартної згортки і субдескритизації, якщо ми заглибимося в будь-яку модель. При розширеній згортці без збільшення обчислювальної складності ми зможемо отримати більшу вихідну карту ознак, яка, підходить для класифікації ураження шкіри з точки зору точності.

Модель має розмір ядра 3 × 3 в кожному шарі, і без збільшення розміру ядра ми можемо збільшити розмір рецептивного поля, додавши різні швидкості розширення в існуючому шарі архітектури. Розмір вхідного шару становить 192 × 256 × 3 (висота зображення × ширина зображення × RGB). Початковий блок мережі має швидкість розширення = 1, потім від другого до п'ятого блоку має швидкість розширення 2, 4, 8 і 16 відповідно. Для реалізації даної методики, в основному натхнення прийшло від різних мульти-мережевих моделей, в яких було знайдено ієрархію з декількох різних розмірів сітки [14], з багатьма семантичними моделями сегментації зображення.

У всьому згортковому шарі використовуватимо випрямляючий лінійний блок (ReLU) як функцію активації і макс-поляризації, використовуваного для створення виборки між кожним згортковим блоком. Після останнього згорткового і макс-полярного шару ми виконуємо глобальну операцію макс-поляризації, яка приймає тензор форми h × w × d (h × w = просторові розміри,

d = кількість карт характеристик) і забезпечує вихідний тензор форми 1 × 1 × d. Потім додаємо два повністю пов'язаних шару в цих моделях з фільтрами 512, 7 (набір даних має сім класів), відповідно, з випадаючим шаром (швидкість випадання = 0.50), між якими в якості функції регулятора використовується істотне ослаблення швидкості перепідгонки і одночасно обчислювально-розумне. RELU є функцією активації для першого щільного шару, а для останнього — SoftMax. З рисунку 2.2.1 можна помітити детальну візуалізацію розширеної VGG-16.

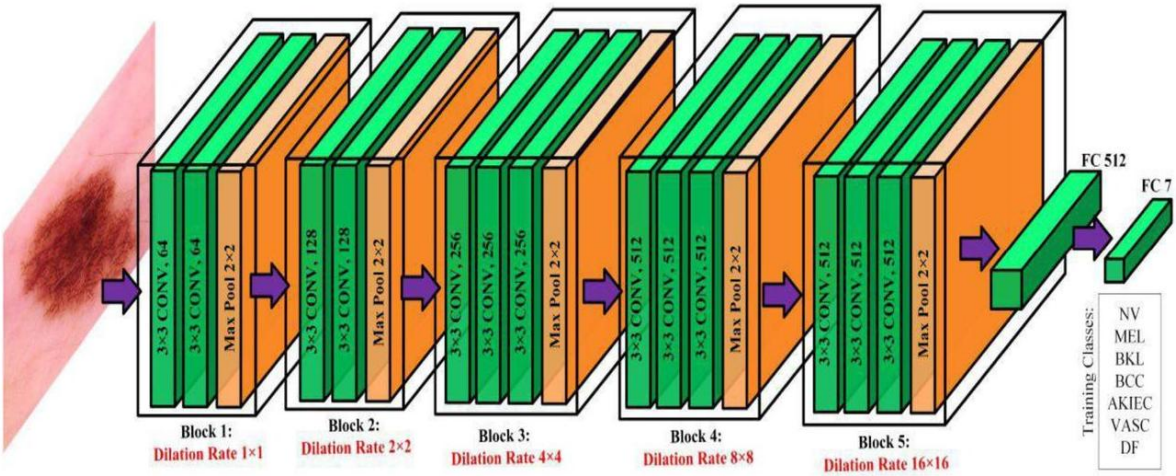


Рисунок 2.2.1 — Розширена модель VGG16 з різною швидкістю розширення на кожному блоці

## 2.2 Архітектура детектора

Алгоритми класифікації намагаються знайти присутність на зображенні раніше визначеного об'єкта, в той час як алгоритми виявлення об'єктів намагаються описати граничне поле навколо об'єкта, щоб знайти його всередині зображення. Існують різні алгоритми виявлення і локалізації об'єктів, засновані на глибокому навчанні. Ці алгоритми можна розділити на дві групи. Перша група алгоритмів складається з двох етапів. На першому етапі на зображенні створюється певна кількість потенційних обмежувальних прямокутників, потім запускаються класифікатори на основі згорткової нейронної мережі для

виявлення об'єктів в цих раніше визначених прямокутниках. Після процесу класифікації на етапі постобробки на виявлених обмежувальних прямокутниках вносяться деякі поліпшення, такі як уточнення обмежувальних прямокутників, усунення дублюючих детекцій, переупорядкування боксів відповідно до інших певних об'єктів на сцені. Ці складні процеси протікають досить повільно, і оптимізувати кожну сингулярну складову дуже складно без окремої підготовки, необхідної для кожної зі складових. Найбільш поширеними прикладами цих алгоритмів є основані на регіоні згорткові нейронні мережі і їх більш просунуті версії Fast-RCNN і Faster-RCNN. Алгоритми другої групи засновані на проблемі регресії. Замість того, щоб вибирати привабливі частини всередині зображення, вони намагаються передбачити обмежуючі поля і класи у всьому зображенні за один прогін.

Yolo є одним з найвідоміших прикладів цієї групи алгоритмів і одним з найпотужніших і швидких алгоритмів виявлення об'єктів з використанням методів глибокого вивчення. Перша версія Yolo була представлена в 2016 р. Він здатний виявляти і класифікувати кілька об'єктів по зображеннях в реальному часі зі швидкістю 45 кадрів в секунду. На відміну від інших методик, які посилають до класифікаторів кілька плям зображень, він посилає все зображення на одну згорткове нейронне мереже. Принцип роботи Yolo досить простий. Одна згорткова нейронна мережа пророкує кілька обмежувальних блоків разом з усіма можливостями класу одночасно (див. рис. 2.3.1). Тому Yolo більш здатний вивчати узагальнююче уявлення об'єктів, ніж альтернативи. Yolo перетворює проблему виявлення в проблему регресії. Прогнози представлені у вигляді (S × S) * B * (5 + C) тензора. Щоб коротко пояснити основну ідею Yolo, необхідно почати з розуміння його тензора пророкувань. Yolo розбиває вхідний зображення на непересічні (S × S) осередки сітки. Якщо середня точка об'єкта потрапляє в осередок сітки, то цей самий осередок сітки контролює виявлення даного об'єкта. Кожен осередок сітки відповідає за пророкування в можливих обмежувальних

прямокутниках і довірчих оцінок для них. Довірча оцінка — це вираз існування або відсутність будь-якого об'єкта всередині обмежуючого поля.

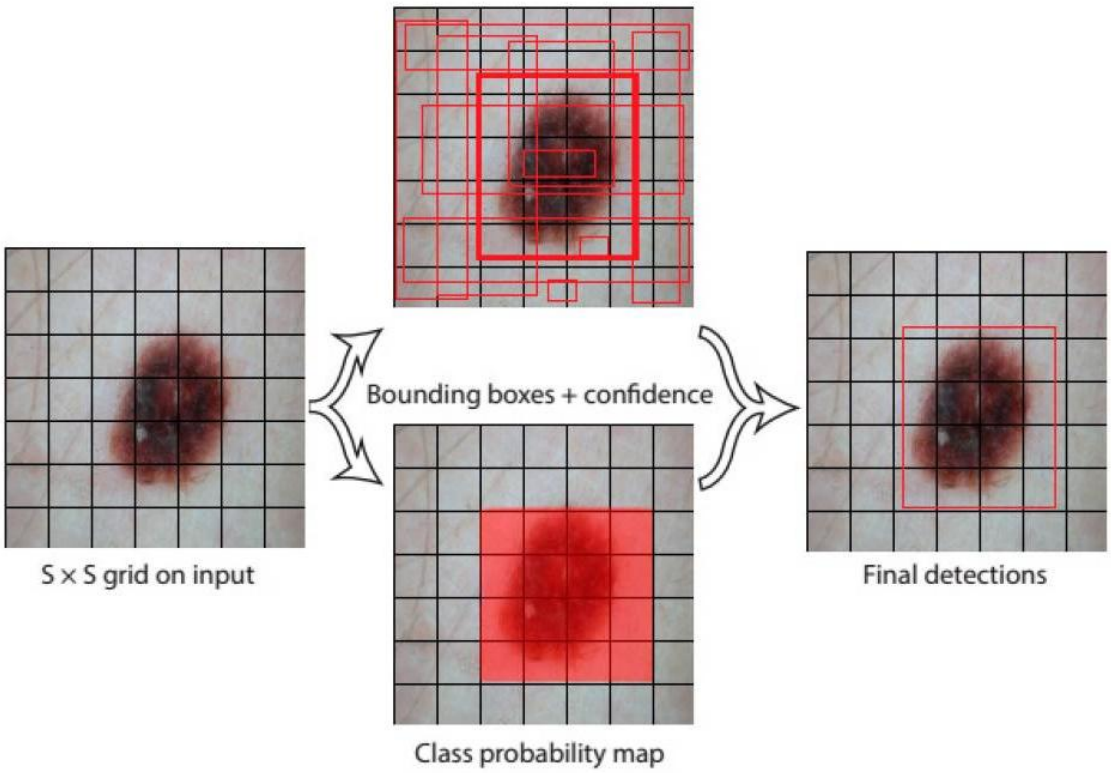


Рисунок 2.3.1 — Спрощена схема розпізнавання ураження шкіри з використанням архітектури детектора Yolo

Аналіз рис. 2.3.1 показує, що у Yolo детекторі відбувається поділ зображення сіткою розміром (S × S) комірок, кожна з яких відповідає за створення B можливих обмежувальних рамок з оцінкою ймовірності їх належності до класу C.

Впевненість розпізнавання (confidence) — це множення ймовірності належності до об'єкта інтересу (Pr) і частки перетину прогнозованою і розмітковою обмежувальними рамками (Intersection over Union, IOU) (див. рис. 2.3.2), як в даній формулі

$$Confidence = \Pr(Object) * IOU_{predicted}^{ground\ truth}. \quad (2.3.1)$$

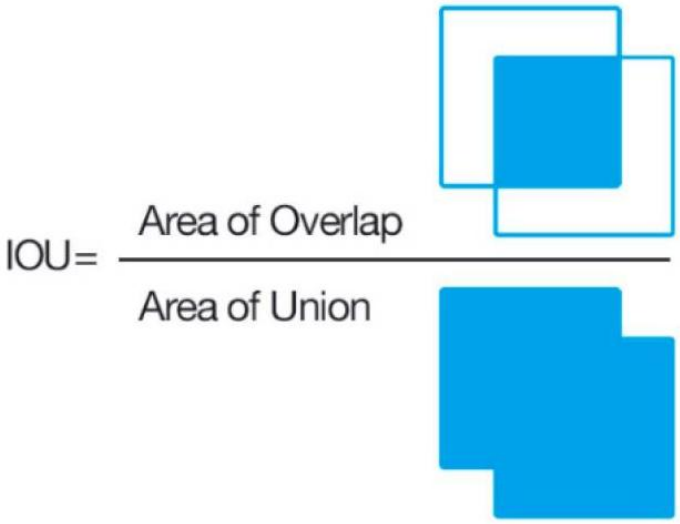


Рисунок 2.3.2 — Ілюстрація функції обчислення ступеню перетину між прогнозованою і розмітковою обмежувальними рамками (IOU), що використовується для валідації продуктивності моделі детектора

Якщо у відповідній клітинці немає об'єкта, то оцінка достовірності повинна бути нульовою. В іншому випадку бал довіри дорівнює IOU. Кожне обмежуюче поле має 5 змінних, таких як x, y, w, h і довірча оцінка. x і y вказують координати центральної точки обмежуючого поля, в той час як w і h представляють значення ширини і висоти (див. рис. 2.3.3). Також існує додаткова змінна C для оцінки класу. Кожен осередок сітки пророкує ймовірності умовного класу C. Yolo обчислює для кожного осередку класу довірчі ймовірності шляхом множення ймовірностей умовного класу та індивідуальних прогнозів довірчих ймовірностей осередку під час тестування, як показано у формулі 2.3.2. Ці оцінки демонструють ймовірність появи цього класу в осередку і те, наскільки добре передбачений бокс підходить для даного об'єкта

$$\Pr(Class_i \,|Object) * \Pr(Object) * IOU_{predicted}^{ground\ truth} = \Pr(Class_i) * IOU_{predicted}^{ground\ truth}. \quad (2.3.2)$$

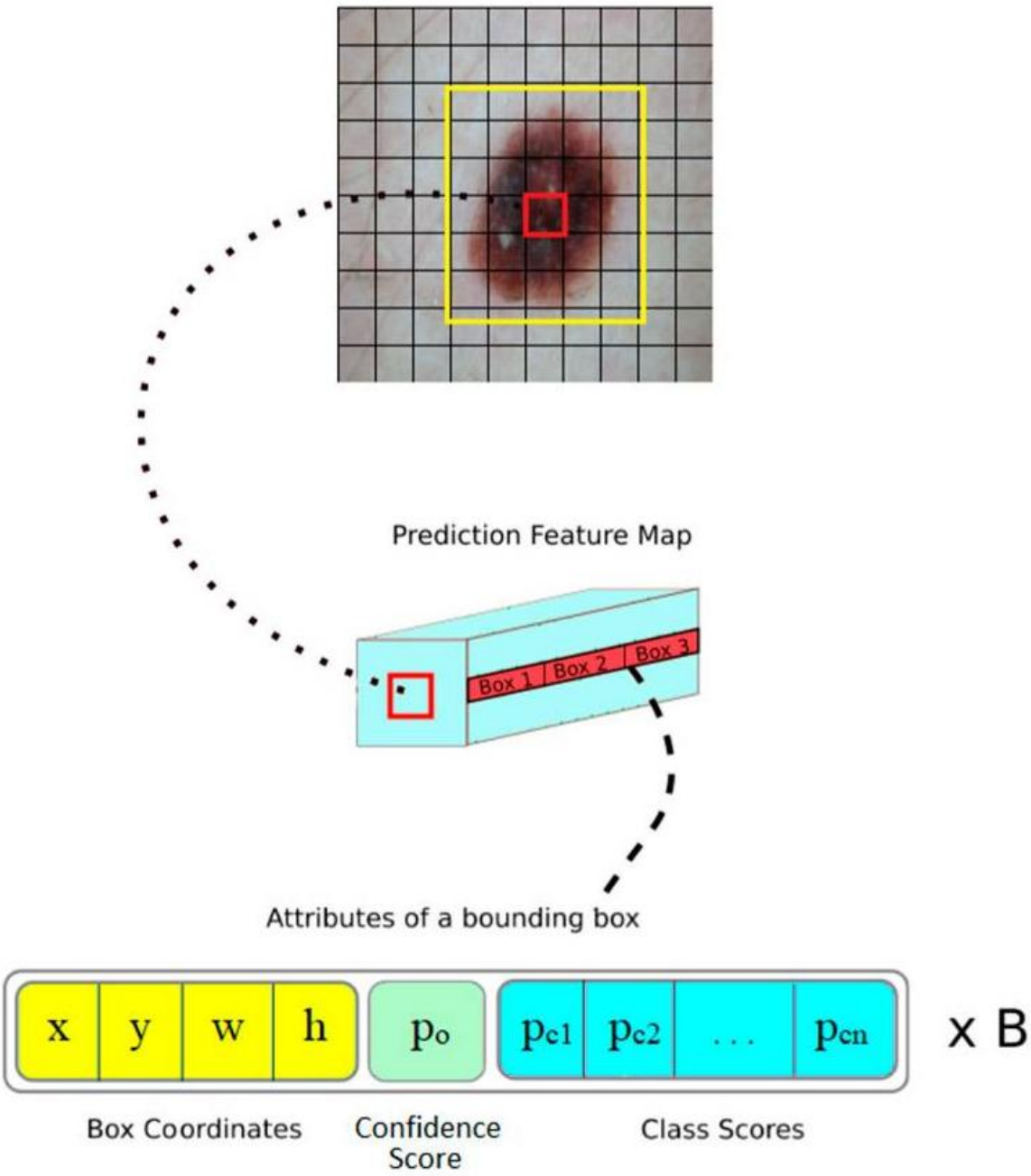


Рисунок 2.3.3 — Зв’язок виходів детектора з пікселем карти ознак

Показана на рис.2.3.3 жовта рамка є обмежуючою рамкою, що побудована на сітці і відповідає за виявлення пошкодження. Кожна обмежуюча рамка має такі параметри, як x і y, центральні координати обмежуючого поля, w і h, ширина і висота обмежуючого поля, оцінка достовірності Po і оцінка ймовірності класу Pcn.

Yolo надзвичайно швидкий в порівнянні з іншими алгоритмами виявлення і класифікації об'єктів, так як він дивиться на зображення один раз і не вимагає складних процесів. Крім того, він робить більш точні прогнози під час сканування всього зображення.

В даному дослідженні використовувалася модель Yolov3 [15], остання і поліпшена версія попередніх мереж Yolo. У відповідності з останніми досягненнями в області алгоритмів виявлення об'єктів, деякі ефективні розробки в попередніх версіях Yolo поступилися місцем Yolov3. Основні зміни в Yolov3 перераховані нижче:

- yolov3 використовує логістичну регресію як функцію втрат при прогнозі довірчих значень обмежуючих полів;
- yolov3 використовує кілька незалежних логістичних класифікаторів замість софтмакс-функції для прогнозування довірчих ймовірностей класу. Це поліпшення дуже важливо, коли на зображенні присутні кілька об'єктів;
- у yolov3 замість макс-полярної вибірки використовується зрушена конвекція для низхідної вибірки;
- yolov3 складається з декількох згорткових шарів на додаток до екстрактора базової характеристики, що розширює його можливості багатомасштабного прогнозування при трьох різних розмірах. Це дозволяє системі виробляти більш точні детекції на маленьких об'єктах на зображенні;
- останнім удосконаленням в Yolov3 є міжшарові зв'язку між шарами передбачення. Карти характеристик, отримані в результаті операції висхідної вибірки, були об'єднані з картами характеристик попередніх шарів за допомогою операції конкатенації. Ця комбінація забезпечила більш точне виявлення на невеликих об'єктах.

## 2.3 Метрики оцінки ефективності класифікатора

Для вимірювання продуктивності, пропонована робота оцінюється з використанням точності, чутливості, достовірності та f1-оцінки. Ці метрики визначаються з точки зору числа справжніх позитивних (TP), справжніх негативних (TN), помилкових негативних (FN) і помилкових позитивних (FP).

Чутливість допомагає, коли вартість помилково-негативного висока

$$\text{Чутливість} = \frac{TP}{TP+FN}.$$

Точність допомагає при високій вартості хибнопозитивних спрацьовувань. Так що давайте припустимо, що проблема полягає в виявленні раку шкіри. Якщо у нас є модель з дуже низькою точністю, то багатьом пацієнтам скажуть, що у них меланома, і це буде включати в себе деякі помилкові діагнози. На карту поставлено багато додаткових тестів і стресів. Коли хибнопозитивні результати занадто високі, ті, хто відстежує результати, навчаться ігнорувати їх після того, як на них почнуть подавати помилкові сигнали тривоги

$$\text{Точність } = \frac{TP}{TP+FP}.$$

Достовірність показує можливість правильного визначення, тобто відсоток пацієнтів і здорових людей, яким поставлений правильний діагноз

$$\text{Достовірність} = \frac{TP+TN}{TP+TN+FP+FN}.$$

F1 — це загальний показник точності моделі, що поєднує в собі точність і нагадування. Тобто, хороша оцінка F1 означає, що у вас низький рівень FP і низький рівень FN, тому ви правильно ідентифікуєте реальні загрози, і вас не турбують помилкові спрацьовування. Оцінка F1 вважається ідеальною, коли вона дорівнює 1, в той час як модель є повною невдачею, коли вона дорівнює 0

$$F1 = 2 \times \frac{\text{Чутливість}*\text{Точність}}{\text{Чутливість}+\text{Точність}}.$$

У нижчеприведеній формулі представлено функцію втрат для Yolov3

$$\begin{aligned}
&\boldsymbol{\lambda_{coord}} \sum_{\boldsymbol{i=0}}^{\boldsymbol{S^2}} \sum_{\boldsymbol{j=0}}^{\boldsymbol{B}} \mathbb{1}_{\boldsymbol{ij}}^{\boldsymbol{obj}} (\boldsymbol{x_i} - \boldsymbol{\hat{x}_i})^2 + (\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^2 \\
&+ \boldsymbol{\lambda_{coord}} \sum_{\boldsymbol{i=0}}^{\boldsymbol{S^2}} \sum_{\boldsymbol{j=0}}^{\boldsymbol{B}} \mathbb{1}_{\boldsymbol{ij}}^{\boldsymbol{obj}} \left(\sqrt{\boldsymbol{w}_i} - \sqrt{\boldsymbol{\hat{w}}_i}\right)^2 + \left(\sqrt{\boldsymbol{h}_i} - \sqrt{\boldsymbol{\hat{h}}_i}\right)^2 \\
&+ \sum_{\boldsymbol{i=0}}^{\boldsymbol{S^2}} \sum_{\boldsymbol{j=0}}^{\boldsymbol{B}} \mathbb{1}_{\boldsymbol{ij}}^{\boldsymbol{obj}} (\boldsymbol{C_i} - \boldsymbol{C_i})^2 \\
&+ \boldsymbol{\lambda_{noobj}} \sum_{\boldsymbol{i=0}}^{\boldsymbol{S^2}} \sum_{\boldsymbol{j=0}}^{\boldsymbol{B}} \mathbb{1}_{\boldsymbol{ij}}^{\boldsymbol{noobj}} \left(\boldsymbol{C_i} - \boldsymbol{\hat{C}_i}\right)^2 \\
&+ \sum_{\boldsymbol{i=0}}^{\boldsymbol{S^2}} \mathbb{1}_{\boldsymbol{i}}^{\boldsymbol{obj}} \sum_{\boldsymbol{c \in classes}} (\boldsymbol{p_i(c)} - \boldsymbol{\hat{p}_i(c)})^2.
\end{aligned} \tag{2.3.3}$$

де:

- $\boldsymbol{\lambda}$ — константа, для обліку більш ніж одного аспекту функції втрат;
- C — кількість класів;
- S — розмірність сітки;
- B — параметр описучий кількість обмежуючих прямокутників;
- $x, y, h, w$ — координати центра, висоти та ширини(відповідно) обмежувального прямокутника;
- $\boldsymbol{C_i}$ — показник впевненості в том, чи існує який-небудь об’єкт, чи ні;
- $\boldsymbol{p_i(c)}$ — похибка класифікації;
- Усі позначення з «^» — реальні значення взяті з фінального обмежуючого прямокутника, в той час я к позначення без , «^» — прогнозовані занчення. Усі похибки являють собою середньоквадратичні похибки;

- $\mathbb{1}_i^{obj}$ — значення 1 коли об’єкт знаходиться в осеердку *i* та 0 в протилежному випадку;
- $\mathbb{1}_{ij}^{obj}$ — значення 1 коли в осередку знаходитсья об’єкт та j обмежуючий прямокутник має найвищу впевненість серед усіх інших в даному осередку.
- $\mathbb{1}_{ij}^{noobj}$ — значення 1 коли об’єкт не знаходиться в осеердку *i* та 0 в протилежному випадку;

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

## 3.1 Формування вхідного математичного опису

Це дослідження було оцінено за двома загальнодоступними наборами даних, ISBI 2017 і PH2. Набір даних ISBI 2017 був створений для завдання під назвою "Аналіз ураження шкіри з метою виявлення меланоми". Цей набір даних є невеликою частиною архіву Міжнародного співробітництва по візуалізації шкіри (ISIC), який складається з 43906 дермоскопічних зображень [16] і є загальнодоступним для дослідників. Набір даних складається з 8-розрядних RGB дермоскопічних зображень з роздільною здатністю від 540 × 722 до 4499 × 6748. Зображення в наборі даних позначені фахівцями як себорейний кератоз, меланома і доброякісні. Інший набір даних PH2 було запозичено у дослідницької групи Університету Порту, зібраний в дерматологічній службі лікарні Педру Хіспано, Португалія [17]. Набір даних PH2 складається з 800 атипових невусів, 800 поширених невусів і 400 випадків меланоми. Загальний розмір набору даних становить 2000 зображень ураження шкіри. Як і набір даних ISBI 2017, набір даних PH2 був отриманий при тих же умовах. Всі зображення є 8-бітові RGB-зображення з роздільною здатністю 768 × 560 пікселів і були зроблені за допомогою об'єктива з 20-кратним збільшенням. Крім того, обидва набори даних містять зображення ушкоджень з межами сегментації, анотовані експертним дерматологом.

На рис. 3.1.1 можна спостерігати розпределення датасету ISBI 2017 за статтю та віковою групою. Можна помітити, більшість пацієнтів перебувають у віковій групі від 35 до 65 років, а більшість — в діапазоні 40-50 років:

- чоловіки в основному знаходяться в діапазоні 40-70;
- жінки знаходяться в діапазоні 30-60.

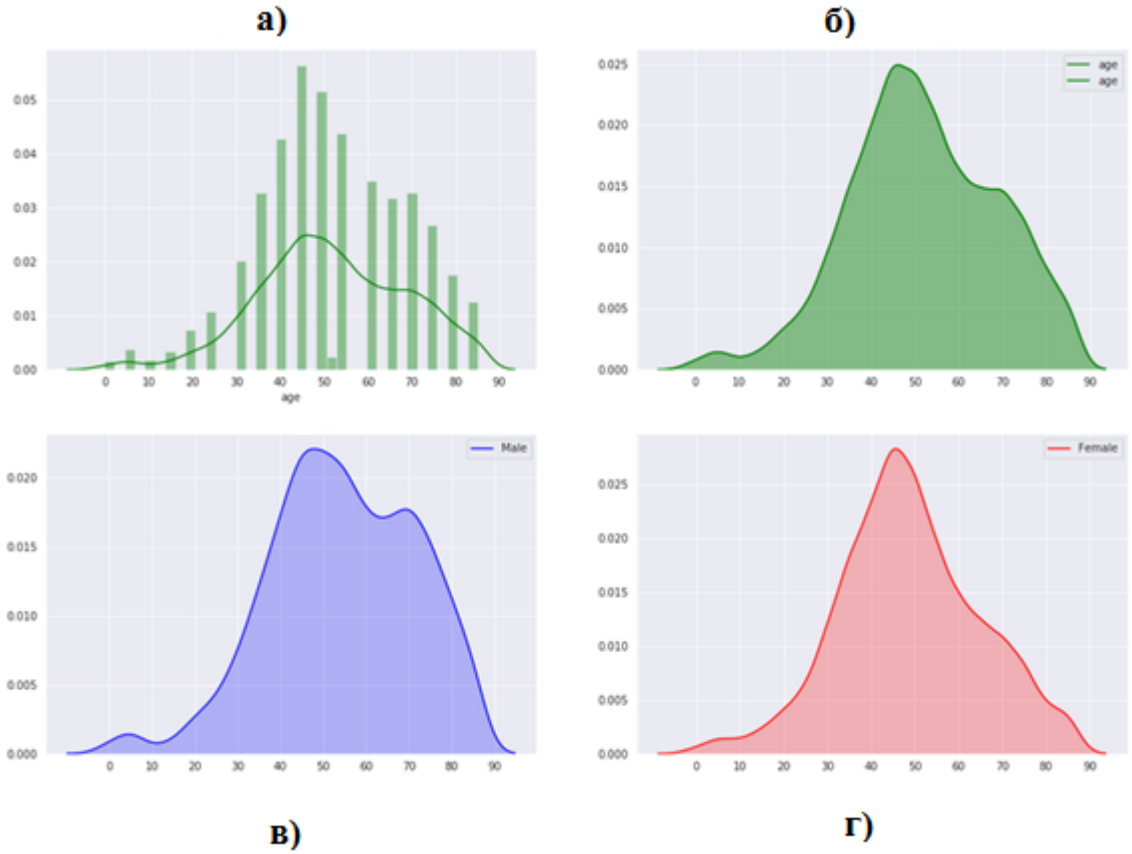


Рисунок 3.1.1 — Графіки розпределення даних датасету за статтю та віковою групою: а — загальний розподіл; б — загальний віковий розподіл; в — розподіл серед чоловіків; г — розподіл серед жінок

На рис. 3.1.2 показані деякі зразки з ISBI 2017 і наборів даних PH2 у вигляді зображень пошкоджень і їх істин.

Перед початком тренування Yolov3 всі дані в навчальному наборі були змінені до дозволу 512 × 512. Потім кожне зображення в тренувальній установці було марковано відповідно до потреб тренування Yolov3. Yolov3 потребує деякої інформації про зображення, а також в зображеннях під час навчання. Ця інформація включає в себе координати середньої точки (x, y), ширину (w) і висоту (h), значення граничного поля і визначення його класу визначається об'єкта.

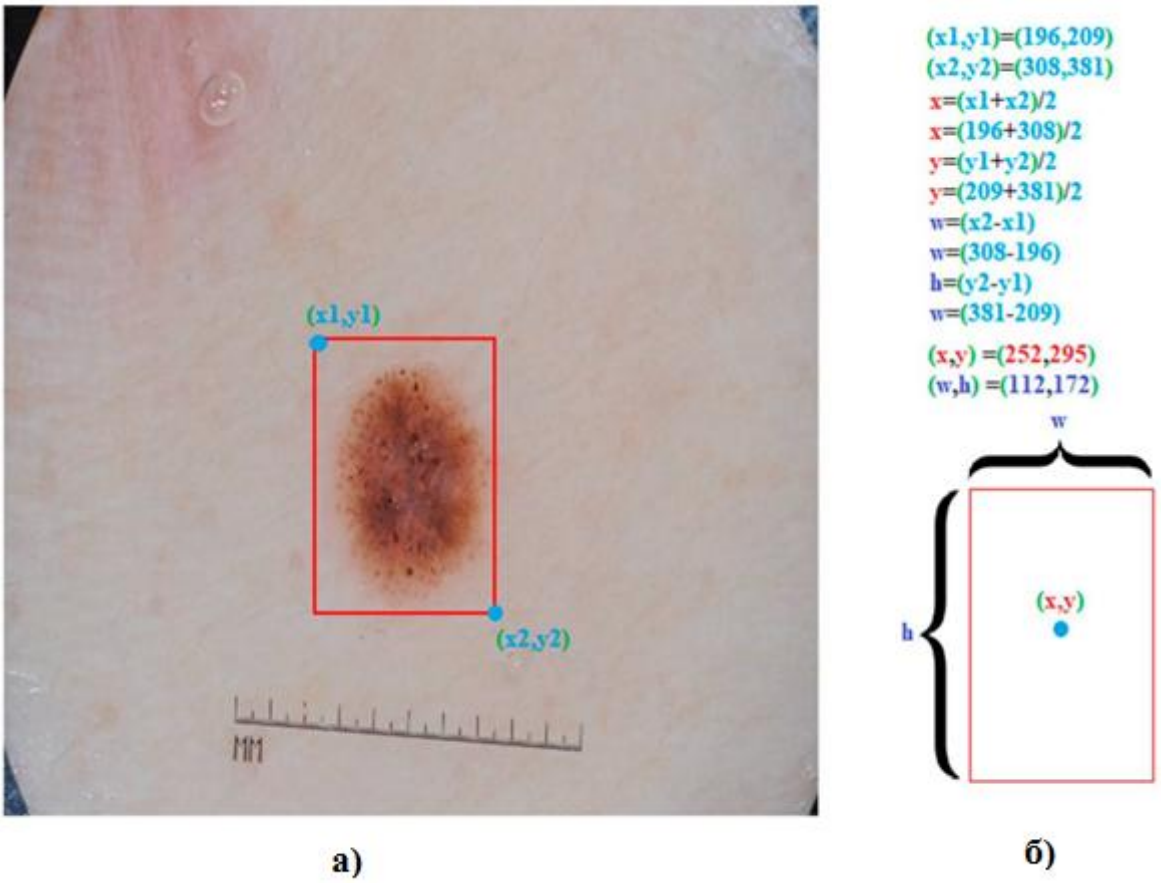


Рисунок 3.1.2 — Приклад маркування зображення ураження шкіри для тренування Yolov3: а — обмежувальна рамка розмітки; б — координати і параметри обмежувальної рамки

Як видно з рис. 3.1.2, координати верхнього лівого кута (x1, y1) і нижнього правого кута (x2, y2) обмежувальної рамки використовуються для визначення параметрів області ітересу x, y, w і h.

Для порівняння характеристик згорткової нейронної мережі, було використано датасет з 7-ма класами різних захворювань шкіри.

Класи 7 хвороб шкіри:

1) базальноклітинна карцинома;
2) актинічний кератоз;
3) меланоцитарний невус;
4) доброякісний кератоз;

5) дерматофіброма;
6) меланома;
7) ураження судин.

### 3.2 Короткий опис програмної реалізації системи

Система розпізнавання шкіряних захворювань за зображенням була розроблена за допомогою мови програмування Python 3, на базі PyCharm Community Edition 2019.1.2. Програмний код системи приведено в додатку А. Також в ході розробки використовувались додаткові модулі з опис яких можно ознайомитись в табл. 3.2.1.

Таблиця 3.2.1 — Додаткові модулі для Python, що використовувалися в процесі розробки

| Назва модулю | Огляд роботи та особливостей |
|---|---|
| NumPy | Пакет загального призначення для обробки масивів. Він надає високопродуктивний багатовимірний об'єкт масиву і інструменти для роботи з цими масивами.<br>Це фундаментальний пакет для наукових обчислень за допомогою Python. Він містить різні можливості, в тому числі і ці важливі:<br>- потужний об'єкт N-мірного масиву;<br>- складні (мовні) функції;<br>- корисні можливості лінійної алгебри.<br>Крім очевидного наукового використання, NumPy також може бути використаний в якості ефективного багатовимірного контейнера загальних даних. Довільні типи даних можуть бути визначені за допомогою Numpy. |

Продовження таблиці 3.2.1

| | |
|---|---|
| Scikit-learn | Бібліотека на Пайтон, яка надає безліч алгоритмів навчання. Вона побудована на основі деяких технологій, з якими ви, можливо, вже знайомі, таких як NumPy, pandas і Matplotlib.<br>Функції, які надає Scikit-learn, включають в себе:<br>- регресія, включаючи лінійну і логістичну регресію;<br>- класифікація, включаючи K-найближчих сусідів;<br>- кластеризація, включаючи K-means і K-means ++;<br>- попередня обробка, включаючи нормалізацію minmax. |
| Pandas | Пакет Python, що надає швидкі, гнучкі та виразні структури даних, призначені для того, щоб зробити роботу з "реляційними" або "поміченими" даними простішою і інтуїтивно зрозумілою. Його мета — стати фундаментальним будівельним блоком високого рівня для проведення практичного аналізу даних в реальному світі на Python. Крім того, його ширша мета — стати найбільш потужним і гнучким інструментом аналізу даних з відкритим вихідним кодом і маніпулювання ними на будь-якій мові. Він уже знаходиться на шляху до цієї мети. |

Кінець таблиці 3.2.1

| | |
|---|---|
| TensorFlow | Децентралізована платформа з відкритим вихідним кодом для машинного навчання. Вона має велику гнучку екосистему інструментів, бібліотек і ресурсів спільноти, яка дозволяє дослідникам просувати новітні досягнення в області ML, а розробникам легко створювати і впроваджувати програми на базі ML.<br>Спочатку TensorFlow був розроблений дослідниками і інженерами, що працюють в команді Google Brain в рамках дослідницької організації Google Machine Intelligence Research для проведення машинного навчання і досліджень глибоких нейронних мереж. Система досить загальна, щоб бути придатною і в цілому ряді інших областей. |

За табл. 3.2.2, можна ознайомитися з переліком функцій програмного додатку та їх описом.

Таблиця 3.2.2 — Опис функій програмного додатку

| Функції | Опис |
|---|---|
| findScan(data, name, key) | Функція пошуку відповідного скану за ім’ям. |
| test_io(path, mode) | Метод для завантаження зображень.<br>Параметри:<br>- path — місцезнаходження зображень;<br>- mode — режим роботи системи. |
| process_path(file_path) | Метод для попереднього упорядкування файлів та формування відповідної пари зображення з обмежувальним прямокутником. File_path — шлях до місцезнаходження вхідних даних. |

Кінець таблиці 3.2.2

| | |
|---|---|
| train_io(file_path) | Формування необхідної пари зображення та обмежувального прямокутника. Їх необхідна попередня обробка для тренувальних даних. File_path — шлях до місцезнаходження вхідних даних. |
| val_io(file_path) | Формування необхідної пари зображення та обмежувального прямокутника. Їх необхідна попередня обробка для валідаційних даних. File_path — шлях до місцезнаходження вхідних даних. |
| prepare_for_training(ds, cache, shuffle_buffer_size) | Метод для загрузки датасету та формування порції даних для тренування.<br>Параметри:<br>- ds — датасет;<br>- cache — параметр для використання кешу;.<br>- shuffle_buffer_size — розмір буферу. |

Також є додатковий клас CyclicLR, який імлементує політику циклічної швидкості навчання. Цей метод циклічно змінює швидкість навчання між двома граниями з деякою постійною частотою. Амлітуда циклу може бути масштабована по кожній ітерації чи по циклу. З методати даного класу можна ознайомитись в  табл. 3.2.3.

Список аргументів класу:

- new_base_lr — початковий рівень навчання, котрий являється нижньою границею циклу;
- new_max_lr — верхня границя циклу. Функціонально вона визначає амплітуду циклу (new_max_lr – new_base_lr). lr в будь-якому циклі,

являється сумою new_base_lr і деякого масштабування амплітуди; тому new_max_lr в залежності від функції масштабування може бути фактично не досягнуто;

- new_step_size — кількість навчальний ітерацій за цикл;
- logs — параметр для логування.

Таблиця 3.2.3 — Опис фунцій класу CyclicLR

| Функції | Опис |
|---|---|
| _reset(self, new_base_lr, new_max_lr, new_step_size) | Скидання циклових ітерацій. Опціональне регулювання розміру границі/кроку. |
| clr(self) | Допоміжна функція для скидання циклових ітерацій. |
| on_train_begin(self, logs) | Ініціює параметри тренування для n_epochs. |
| on_batch_end(self, epoch, logs) | Готує гіперпараметри до наступного циклу. |

### 3.3 Аналіз результатів моделювання

Під час експериментів навчання пропонується здійснювати починаючи з відносно великих значень коефіцієнту оновлення, який ще називають швидкістю навчання (learning rate), а потім поступово знижувати ого значення. Основна ідея полягає в тому, щоб як найшвидше перейти від початкових параметрів моделі до квазіоптимальних. При цьому наступні зменшення швидкості навчання забезпечують дослідження більш глибоких, але більш вузьких частини функції втрат.

Були проведені відповідні експерименти, з метою оцінки системи ідентифікації ураження шкіри з метою виявлення захворювання шкіри, з використанням датасетів ISBI та PH2, про які згадувалось раніше. Цей набір даних випущений Міжнародною організацією зі співробітництва в області візуалізації шкіри (ISIC) і складається з трьох частин: 1, 2 і 3. 90% використовуються в якості навчальних даних для класифікації. Решта 10%

зображень використовуються в якості тестових даних для прогнозування результату.

Валідація моделі була проведена на 938 зразках зображень. Було оцінено мікро і середньозважене значення для точності, чутливості і f1-оцінки для розуміння узагальненої продуктивності моделі, середньозважене і мікро середнє для точності, чутливості і f1-оцінки були оцінені для семи класів .

Середньозважене значення 0.89, 0.83, 0.83 і мікро-середнє значення 0.83, 0.83, 0.83 були зафіксовані для точності, чутливості і f1-оцінки. Модель показує кращу точність, чутливість і f1-оцінку для меланоцитарного невуса. Звіт про класифікацію за багатьма класами, що показує мікро-середню і середньозважену величину для точності, чутливості і f1-оцінки, представлений у таблиці 3.3.1.

Таблиця 3.3.1 — Точнісні характеристики отриманої моделі для заданого алфавіту класів

| Класи | Точність | Чутливість | F1-оцінка |
|---|---|---|---|
| Актинічний кератоз | 0.36 | 0.38 | 0.37 |
| Базальноклітинна карцинома | 0.55 | 0.87 | 0.68 |
| Доброякісний кератоз | 1.00 | 0.13 | 0.24 |
| Дерматофіброма | 0.21 | 0.50 | 0.30 |
| Меланома | 0.28 | 0.69 | 0.40 |
| Меланоцитарний невус | 0.95 | 0.93 | 0.94 |
| Ураження судин | 0.73 | 0.73 | 0.73 |
| Мікро-середнє | 0.83 | 0.83 | 0.83 |
| Середньозважене | 0.89 | 0.83 | 0.83 |

На рис.3.3.1 та рис.3.3.2 показано графіки зміни протягом навчання усереднених значень функції втрат та точності відповідно на навчальній та тестовій множині. Аналіз рис.3.3.1 та рис.3.3.2. показує, що збіжність до оптимальних параметрів моделі досягається вже з 20-ї епохи навчання. При цьому не спостерігається значного розриву між навчальними та валідаційними кривими, що свідчить про гарну узагальнюючу здатність моделі і відсутність прояву ефекту підгонки під тренувальні дані.

Для порівняння, в табл. 3.3.2, наведено класифікаційні характеристики деяких існуючих моделей.

Таблиця 3.3.2 — Порівняння результатів поточного дослідження з деякими суміжними попередніми

| Джерело | Рік | Класифікатор | № класів | Точність % |
|---|---|---|---|---|
| [18] | 2016 | VGGNet | 4 | 79.3 |
| | | AlexNet&VGGNet | | 79.9 |
| | | GoogleNet&VGGNet | | 81.2 |
| | | GoogleNet&AlexNet | | 80.7 |
| [19] | 2016 | Multi-tract CNN | 10 | 75.1 |
| [20] | 2017 | CNN | 3 | 69.4 |
| | | CNN-PA | | 72.1 |
| | | CNN | 9 | 48.9 |
| | | CNN-PA | | 55.5 |
| | 2020 | Поточне дослідження | 7 | 83.15 |

З табл. 3.3.2, легко бачити, що у порівнянні з суміжними моделями, запропонована у даному дослідженні модель показала більш точну продуктивність. У додаток до цього, система має більш швидку та легку архітектуру.

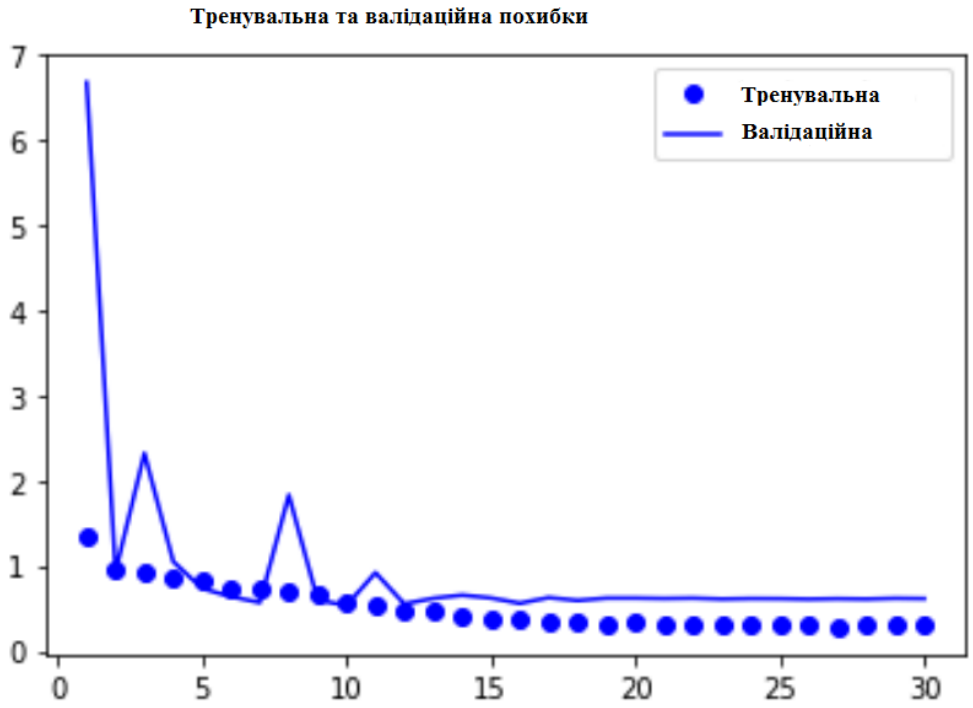


Рисунок 3.3.1 — Графік зміни похибки моделі протягом процесу тренування та валідації

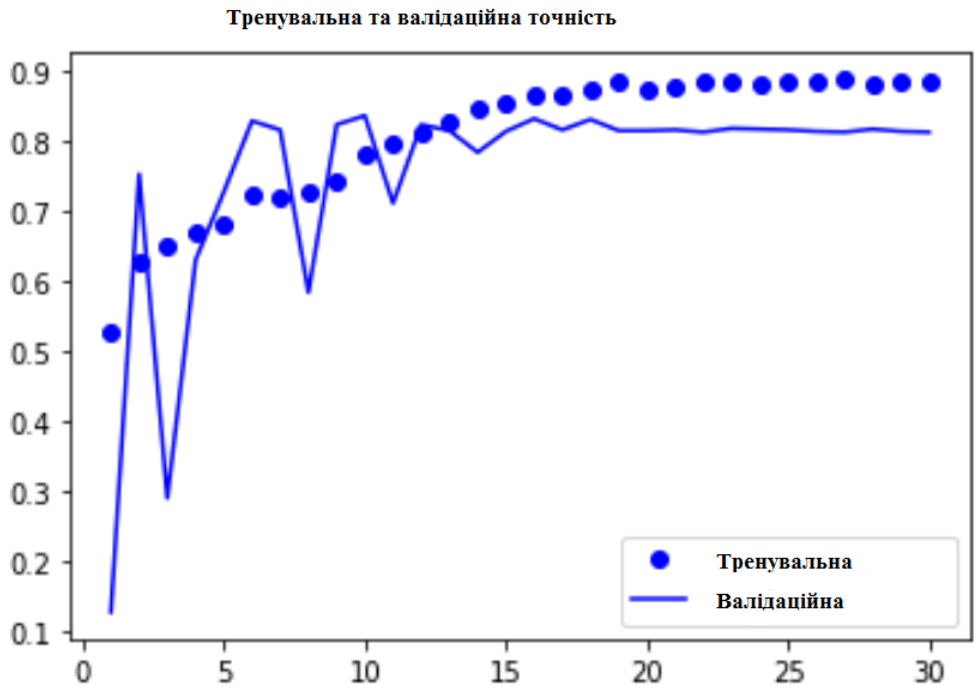


Рисунок 3.3.2 — Графік зміни точності моделі протягом процесу тренування та валідації

Було розроблено систему для забезпечення ефективної автоматизованої багатокласової класифікації зображення ураження шкіри. Модель на тестовій вибірці забезпечує усереднене за класами розпізнавання значення точності, що становить 83.15%, що перевищує отримані іншими дослідниками результати (табл.3.3.2). При цьому, отримана точність є прийнятною для практичного використання в системах інтелектуальної підтримки прийняття рішень. Таким чином, було розроблено систему для забезпечення детектування та класифікації шкіряних захворювань за зображенням.

# ВИСНОВКИ

За останні десятиліття почастішали випадки захворювання на рак шкіри і протягом найближчого часу необхідно перейти до ефективної і надійної автоматизованої системи класифікації раку шкіри, яка може забезпечити високоточні і швидкі прогнози. Діагностика раку шкіри є складним завданням навіть для досвідчених лікарів-дерматологів, що розглядають найменшу варіативність появи пошкоджень шкіри.

У даній роботі була реалізована система класифікації шкіряних захворювань за зображенням. Для вирішення завдань автоматизованої класифікації меланоми на дермоскопічних зображеннях як екстрактор ознак було використано згорткову мережу VGG-16. При цьому детектуючий шар було побудовано за архітектурою Yolo v3, що дозволяла уточнити локалізацію та класифікацію ураження шкіри. Проведено експерименти на відкритому наборі даних для аналізу ураження шкіри з метою виявлення меланоми на ISBI 2017 та PH2, заснованому на точності, чутливості і специфічності.

Модель розпізнавання уражень шкіри була навчена на 38 569 зразках зображень дермаскопії з наборів даних ISBI та PH2. При цьому алфавіт класів розпізнавання включає сім класів хвороб: актинічний кератоз, базальноклітинна карцинома, доброякісний кератоз, дерматофіброма, меланома, меланоцитарний невус і судинні ураження. Навчена модель має точність 83,15 %.

Вважається, що запропонована система класифікації меланоми, заснована на глибокому навчанні, може бути використана як частина більш складної системи для аналізу пошкоджень шкіри. В майбутньому ця концепція може бути розширена за рахунок включення в цю мережу імовірнісних графічних моделей для покращення функціональної ефективності системи.

## СПИСОК ЛІТЕРАТУРИ


1. Carli P, De Giorgi V, Crocetti E, et al. Improvement of malignant/benign ratio in excised melanocytic lesions in the ‘dermoscopy era’: a retrospective study 1997–2001. Br J Dermatol. 2004;150(4):687–692.
2. Vestergaard M, Macaskill P, Holt P, Menzies S. Dermoscopy compared with naked eye examination for the diagnosis of primary melanoma: a meta-analysis of studies performed in a clinical setting. Br J Dermatol. 2008;159(3):669–676.
3. Salerni G, Teran T, Puig S, et al. Meta-analysis of digital dermoscopy follow-up of melanocytic skin lesions: a study on behalf of the International Dermoscopy Society. J Eur Acad Dermatol Venereol. 2013 Jul;27(7):805–814.
4. Jemal, A.; Siegel, R.; Ward, E.; Hao, Y.; Xu, J.; Thun, M.J. Cancer statistics, 2019. CA Cancer J. Clin. 2019, 69, 7–34.
5. Terushkin V, Oliveria SA, Marghoob AA, Halpern AC. Use of and beliefs about total body photography and dermatoscopy among US dermatology training programs: an update. J Am Acad Dermatol. 2010 May;62(5):794–803.
6. Kohler R. A segmentation system based on thresholding[J]. Computer Graphics and Image Processing, 1981, 15(4): 319-338.
7. Sulaiman S N, Isa N A M. Adaptive fuzzy-Kmeans clustering algorithm for image segmentation[J]. IEEE Transactions on Consumer Electronics, 2010, 56(4).
8. Chen L C, Papandreou G, Kokkinos I, et al. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs[J].

9. Sunitha Abburu and Suresh Babu Golla, “Satellite Image Classification Methods and Techniques: A Review”, International Journal of Computer Applications (0975 – 8887) Volume 119 – No.8, June 2015
10. Rajesh Sharma R Beaula A, Marikkannu P, Akey Sungheeth, C. Sahana, “Comparative Study of Distinctive Image Classification Techniques”, 10th International Conference on Intelligent Systems and Control (ISCO), 2016.
11. Ronneberger O, Fischer P, Brox T., “ U-Net Convolutional networks for Biomedical image segmentation”, International Conference on Medical Image Computing and Computer Assisted Intervention(MICCAI), Springer, LNCS, vol 9351: pp.234-241, 2015.
12. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in Proc. of International Conference on Machine Learning, pp. 448-456, 2015.
13. A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," Computer Science, 2015.
14. Papandreou, G., & Maragos, P. (2006). Multigrid geometric active contour models. IEEE transactions on image processing, 16(1), 229–240.
15. Redmon J., Farhadi A. Yolov3: An incremental improvement. arXiv. 20181804.02767.
16. Codella N.C., Gutman D., Celebi M.E., Helba B., Marchetti M.A., Dusza S.W., Kalloo A., Liopyris K., Mishra N., Kittler H., et al. Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (ISBI), hosted by the international skin imaging collaboration (ISIC); Proceedings of the 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018); Washington, DC, USA. 4–7 April 2018.

17. Mendonça T., Ferreira P.M., Marques J.S., Marcal A.R., Rozeira J. PH 2-A dermoscopic image database for research and benchmarking; Proceedings of the 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC); Osaka, Japan. 3–7 July 2013. [PubMed]
18. Harangi B, Baran A, Hajdu A. Classification Of Skin Lesions Using An Ensemble Of Deep Neural Networks.40th Annual International Conference of the IEEE Engineering in Medi-cine and Biology Society (EMBC). 2018; 2018:2575–2578.
19. Kawahara J, HamarnehG. Multi-resolution-Tract CNN with Hybrid Pretrained and Skin-Lesion Trained Layers. In: Wang L, Adeli E, Wang Q, Shi Y, Suk HI (eds) Machine Learn-ing in Medical Imaging. MLMI 2016. Lecture Notes in Computer Science. 2016; 10019.
20. Esteva A et al.Dermatologist-level classification of skin cancer with deep neural networks. Nature. 2017; 542(7639):115–118.

# ДОДАТОК А

```
# Libraries
import glob
import numpy as np
import cv2
from skimage import filters as skifilters
from scipy import ndimage
from skimage import filters
import matplotlib.pyplot as plt
import tqdm
from sklearn.utils import shuffle
import pandas as pd
from __future__ import unicode_literals
from __future__ import print_function
from __future__ import division
from __future__ import absolute_import
import tensorflow as tf
import os
import h5py
import time
import json
import warnings
import dill
from sklearn.metrics import (roc_curve, auc, accuracy_score, f1_score, precision_score,
                             recall_score, classification_report, confusion_matrix)
from scipy import interp
import efficientnet.tfkeras as efn
from classification_models.tfkeras import Classifiers as aux_models

# Neural Network Parameters
NN_COLOR = 'single'  # Input Color Space           ('single'/'multi')
NN_FINETUNE = 'full'  # Trainable Layers            ('full'/'classifier'/Starting Layer Name)
NN_WEIGHTS = None  # Pre-Trained Encoder Weights ('imagenet'/None)
TRAIN_SCHEME = 'clean'  # Training Scheme             ('clean'/'warm'/'freeze'/'resume')
DROPOUT = False  # Dropout Regularization
EPOCHS = 7
AUX_EPOCHS = 1
TTA = True  # Test-Time Augmentation
TTA_MODE = 'mean'  # TTA Aggregation             ('mean'/'maxconf')

# Learning Rate Hyperparameters
BASE_LR = 1e-5
MODE_LR = 'CLR'

# I/O Parameters
CLASS_NAMES = ['nv', 'les']
TRAIN_SAMPLES = 4800
VAL_SAMPLES = 1200
IO_X = 450  # Original Dimensions
IO_Y = 600  # Original Dimensions
CROP_FACTOR = 1.00  # Crop Percentage
```

```
DIM_X = 448  # Cropped Training Dimensions
DIM_Y = 448  # Cropped Training Dimensions
BATCH_SIZE = 4  # Mini-Batch Size
# Testing Dataset Parameters
TEST_DIR = '../data/grayworld/val/'
TEST_MODE = 'val'  # 'val'/'test'


def findScan(data, name, key):
    for i, dic in data.items():
        if dic[key] == name:
            return i
        return -1


def test_io(path, mode):
    """
    Input:  path
    Output: data[p] = {
        'id':
        'image':
        'label': }
    """
    if (mode == 'test'):
        # Importing Images
        target_dir = glob.glob(path + "/les/*.png")
        print("Number of Test Images:", len(target_dir))

        # Creating Dictionary
        data = {}
        for p in range(len(target_dir)):
            scan_id = target_dir[p].replace(".png", "")
            scan_id = scan_id.replace(path + "\\", "")
            # Creating List of Dictionary
            data[p] = {'id': scan_id,
                       'image': target_dir[p]}

    elif (mode == 'val'):
        # Importing Images
        les_dir = glob.glob(path + "/les/*.png")
        nv_dir = glob.glob(path + "/nv/*.png")
        print("Number of LES Images:", len(les_dir))
        print("Number of NV Images:", len(nv_dir))

        # Creating Dictionary (LES Scans)
        data = {}
        for p in range(len(les_dir)):
            scan_id = les_dir[p].replace(".png", "")
            scan_id = scan_id.replace(path + "/les\\", "")
            # Creating List of Dictionary
```

#

```
            data[p] = {'id': scan_id,
                       'image': les_dir[p],
                       'label': 1}  # Label of LES = 1

        # Creating Dictionary (NV Scans)
        for p in range(len(nv_dir)):
            scan_id = nv_dir[p].replace(".png", "")
            scan_id = scan_id.replace(path + "/nv\\", "")
            # Creating List of Dictionary
            data[p + len(les_dir)] = {'id': scan_id,
                                      'image': nv_dir[p],
                                      'label': 0}  # Label of NV = 0
    return data


if (NN_COLOR == 'single'):
    DIM_CHANNELS = 3
    TRAIN_DIR = '../data/grayworld/train/'
    VAL_DIR = '../data/grayworld/val/'

elif (NN_COLOR == 'multi'):
    DIM_CHANNELS = 9
    TRAIN_DIR = '../data/raw/train/'
    VAL_DIR = '../data/raw/val/'
    DATASET_MEAN = [0.610446659205108800, 0.5220753348750297000, 0.5095079890928187000,
                    0.545083657305597700, 0.3857680235700908000, 0.7072208299516552000,
                    1.435617385489288400, 1.1975180678760260000, 1.0724577830134300000]  # [R,G,B,H,S,V,L,a,b]
    DATASET_STD = [0.011219814446842410, 0.0114450729879776450, 0.0104303418370314950,
                   0.008512948157193818, 0.0082573833265490880, 0.0162066173083936020,
                   0.028451386706097037, 0.0018845362785246164, 0.0038278101960121606]  # [R,G,B,H,S,V,L,a,b]


def process_path(file_path):
    # Derive Label
    parts = tf.strings.split(file_path, os.path.sep)  # Split Path into Components
    parts = parts[-2] == CLASS_NAMES  # Generate One-Hot Encoded Label
    label = tf.argmax(tf.dtypes.cast(parts, tf.float32))  # Convert to Single Digit Label

    # Derive Image
    if (NN_COLOR == 'single'):
        img = tf.io.read_file(file_path)  # Load RAW Data from File as String
        img = tf.image.decode_jpeg(img, channels=DIM_CHANNELS)  # Convert RAW Data to a 3D uint8 Tensor
        img = tf.image.convert_image_dtype(img, tf.float32)  # Convert to Floats in Range [0,1]
        img = tf.image.crop_to_bounding_box(img, (IO_X - DIM_X) // 2,  # Crop Image to Central Target
                                            (IO_Y - DIM_Y) // 2,
                                            DIM_X, DIM_Y)
    elif (NN_COLOR == 'multi'):
        img = tf.io.read_file(file_path)  # Load RAW Data from File as String
        img = tf.reshape(tf.io.decode_raw(img, tf.float32),
                         [IO_X, IO_Y, DIM_CHANNELS])  # Convert RAW Data to a 3D float32 Reshaped Tensor
```

```
        img = tf.image.convert_image_dtype(img, tf.float32)  # Convert to Floats in Range [0,1]
        img = tf.image.crop_to_bounding_box(img, (IO_X - DIM_X) // 2,  # Crop Image to Central Target
                                            (IO_Y - DIM_Y) // 2,
                                            DIM_X, DIM_Y)
    return img, label


def train_io(file_path):
    img, label = process_path(file_path)  # Process Filepath to Obtain Image, Label Pair
    # Train-Time Spatial Augmentation
    img = tf.image.random_flip_left_right(img)  # Horizontantal Flip (50% Probability)
    img = tf.image.random_flip_up_down(img)  # Vertical Flip (50% Probability)
    if (NN_COLOR == 'single'):
        # Train-Time Intensity Augmentation
        img = tf.image.random_brightness(img, max_delta=0.25)  # Random Brightness Augmentation
        img = tf.image.random_saturation(img, lower=1.00, upper=1.25)  # Random Saturation Augmentation
        img = tf.image.random_contrast(img, lower=1.00, upper=1.25)  # Random Contrast Augmentation
        img = tf.clip_by_value(img, 0.0, 1.0)  # Ensure intensity range in [0, 1]
    if (NN_COLOR == 'multi'):
        # Normalization (Mean=0, Std=1)
        mean_tensor = tf.constant(np.ones(shape=(DIM_X, DIM_Y, 1)) * DATASET_MEAN, tf.float32)
        std_tensor = tf.constant(np.ones(shape=(DIM_X, DIM_Y, 1)) * DATASET_STD, tf.float32)
        img = (img - mean_tensor) / std_tensor
    return img, label


def val_io(file_path):
    img, label = process_path(file_path)  # Process Filepath to Obtain Image, Label Pair
    if (NN_COLOR == 'multi'):
        # Normalization (Mean=0, Std=1)
        mean_tensor = tf.constant(np.ones(shape=(DIM_X, DIM_Y, 1)) * DATASET_MEAN, tf.float32)
        std_tensor = tf.constant(np.ones(shape=(DIM_X, DIM_Y, 1)) * DATASET_STD, tf.float32)
        img = (img - mean_tensor) / std_tensor
    return img, label


# Generate Dataset of Filepaths
train_list = tf.data.Dataset.list_files(str(TRAIN_DIR + '*/*'))
val_list = tf.data.Dataset.list_files(str(VAL_DIR + '*/*'))

# Multiple Images Loaded/Processed in Parallel from Labeled One-Hot Encoded Dataset
labeled_train_data = train_list.map(train_io, num_parallel_calls=tf.data.experimental.AUTOTUNE)
labeled_val_data = val_list.map(val_io, num_parallel_calls=tf.data.experimental.AUTOTUNE)

for image, label in labeled_train_data.take(1):
    # Verify Example (Redundancy Check)
    print('Data Type/Shape')
    print('image: ', (image.numpy()).dtype, (image.numpy()).shape)
    print('label: ', (label.numpy()).dtype, (label.numpy()).shape)
    print('--------------\nExample Pair\nLabel: ', (label.numpy()))
```

```
    plt.imshow(image.numpy()[:, :, 1], cmap='gray')
    plt.show()


def prepare_for_training(ds, cache=True, shuffle_buffer_size=1000):
    # Small Dataset: Load it only once, keep it in memory.
    # Large Dataset: Use `.cache(filename)` to cache preprocessing.

    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()

    ds = ds.shuffle(buffer_size=shuffle_buffer_size)  # Shuffle Dataset
    ds = ds.repeat()  # Repeat Process
    ds = ds.batch(BATCH_SIZE)  # Load Data in Batches
    ds = ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)  # Prefetch Data in Batches while Model is Training

    return ds


# Final tf.data.Dataset for Training/Validation
train_dataset = prepare_for_training(labeled_train_data)
val_dataset = prepare_for_training(labeled_val_data)

print('Training/Validation tf.data.Dataset Successfully Loaded.')
sample = 0

# Verify Example (Redundancy Check)
image_batch, label_batch = next(iter(train_dataset))
print('Data Type/Shape')
print('image_batch: ', image_batch.dtype, image_batch.shape)
print('label_batch: ', label_batch.dtype, label_batch.shape)
print('--------------\nExample Pair\nLabel: ', label_batch[sample].numpy())
plt.imshow(image_batch.numpy()[sample, :, :, 1], cmap='gray')
plt.show()
# Reduce Learning Rate on Plateau
if (MODE_LR == 'RLRP'):
    COOLDOWN_RLRP = 0
    PATIENCE_RLRP = 3
    FACTOR_RLRP = 0.5
    MIN_RLRP = 1e-8

    RLRP = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', mode='min', verbose=1, cooldown=COOLDOWN_RLRP,
                                                factor=FACTOR_RLRP, patience=PATIENCE_RLRP, min_lr=MIN_RLRP)
    CALLBACKS = [RLRP]


# Exponentially Decaying Learning Rate
```

```
elif (MODE_LR == 'ELR'):
    INIT_LR = BASE_LR
    DECAY_EPOCHS_ELR = 2
    DECAY_RATE_ELR = 0.5
    STAIRCASE = True

    ELR = tf.keras.optimizers.schedules.ExponentialDecay(INIT_LR, decay_steps=(round(
        TRAIN_SAMPLES) // BATCH_SIZE) * DECAY_EPOCHS_ELR,
                                         decay_rate=DECAY_RATE_ELR, staircase=STAIRCASE)
    BASE_LR = ELR
    CALLBACKS = None


# Learning Rate Scheduler
elif (MODE_LR == 'LRS'):
    def scheduler(epoch):
        LRS_LR = BASE_LR
        if (epoch < 2):
            return LRS_LR
        elif (epoch >= 2):  # Decay Every 2 Epochs After Warmup
            LRS_LR = np.power(0.500, (epoch // 2) - 0) * LRS_LR
            return LRS_LR


    CALLBACKS = [tf.keras.callbacks.LearningRateScheduler(scheduler)]


# Cyclic Learning Rate
elif (MODE_LR == 'CLR'):
    CLR_MODE = 'exp_range'
    CLR_MAXLR = 5e-5
    CLR_GAMMA = 1
    CLR_STEPFACTOR = 1.25


    class CyclicLR(tf.keras.callbacks.Callback):
        def __init__(self, base_lr=0.001, max_lr=0.006, step_size=2000.,
                     mode='triangular', gamma=1., scale_fn=None,
                     scale_mode='cycle'):
            super(CyclicLR, self).__init__()
            self.base_lr = base_lr
            self.max_lr = max_lr
            self.step_size = step_size
            self.mode = mode
            self.gamma = gamma
            if scale_fn == None:
                if self.mode == 'triangular':
                    self.scale_fn = lambda x: 1.
                    self.scale_mode = 'cycle'
                elif self.mode == 'triangular2':
```

```
            self.scale_fn = lambda x: 1 / (2. ** (x - 1))
            self.scale_mode = 'cycle'
        elif self.mode == 'exp_range':
            self.scale_fn = lambda x: gamma ** (x)
            self.scale_mode = 'iterations'
    else:
        self.scale_fn = scale_fn
        self.scale_mode = scale_mode
    self.clr_iterations = 0.
    self.trn_iterations = 0.
    self.history = {}
    self._reset()

def _reset(self, new_base_lr=None, new_max_lr=None, new_step_size=None):
    if new_base_lr != None:
        self.base_lr = new_base_lr
    if new_max_lr != None:
        self.max_lr = new_max_lr
    if new_step_size != None:
        self.step_size = new_step_size
    self.clr_iterations = 0.

def clr(self):
    cycle = np.floor(1 + self.clr_iterations / (2 * self.step_size))
    x = np.abs(self.clr_iterations / self.step_size - 2 * cycle + 1)
    if self.scale_mode == 'cycle':
        return self.base_lr + (self.max_lr - self.base_lr) * np.maximum(0, (1 - x)) * self.scale_fn(cycle)
    else:
        return self.base_lr + (self.max_lr - self.base_lr) * np.maximum(0, (1 - x)) * self.scale_fn(
            self.clr_iterations)

def on_train_begin(self, logs={}):
    logs = logs or {}
    if self.clr_iterations == 0:
        tf.keras.backend.set_value(self.model.optimizer.lr, self.base_lr)
    else:
        tf.keras.backend.set_value(self.model.optimizer.lr, self.clr())

def on_batch_end(self, epoch, logs=None):
    logs = logs or {}
    self.trn_iterations += 1
    self.clr_iterations += 1
    self.history.setdefault('lr', []).append(tf.keras.backend.get_value(self.model.optimizer.lr))
    self.history.setdefault('iterations', []).append(self.trn_iterations)
    for k, v in logs.items():
        self.history.setdefault(k, []).append(v)
    tf.keras.backend.set_value(self.model.optimizer.lr, self.clr())


CALLBACKS = [CyclicLR(mode=CLR_MODE, gamma=CLR_GAMMA, base_lr=BASE_LR, max_lr=CLR_MAXLR,
```

```
                    step_size=(round(TRAIN_SAMPLES) // BATCH_SIZE) * CLR_STEPFACTOR)]

else:
    print('Learning Rate Mode not recognized. Falling back to Fixed Base Learning Rate.')
    CALLBACKS = None

print('Hyperparameters Setup Complete.')
DROPOUT = False

# Network Architecture
if (NN_COLOR == 'multi'):

    # Color Space Fusion Layers
    input_layer = tf.keras.layers.Input(shape=(DIM_X, DIM_Y, DIM_CHANNELS))
    conv1x1 = tf.keras.layers.Conv2D(filters=3, kernel_size=1, strides=(1, 1))(input_layer)

    # Feature Extractor Layers
    encoder = efn.EfficientNetB0(input_tensor=conv1x1, include_top=False, weights=None)

    # Classifier Layers
    globavgpool_out = tf.keras.layers.GlobalAveragePooling2D()(encoder.output)
    predictions = tf.keras.layers.Dense(1, kernel_regularizer=tf.keras.regularizers.l2(l=0.01), activation='sigmoid')(
        globavgpool_out)
    model = tf.keras.models.Model(inputs=input_layer, outputs=predictions)

elif (NN_COLOR == 'single'):

    # Feature Extractor Layers
    ## E01
    encoder = efn.EfficientNetB6(input_shape=(DIM_X, DIM_Y, DIM_CHANNELS), include_top=False, weights=NN_WEIGHTS)
    ## E02
    # encoder        = tf.keras.applications.InceptionV3(input_shape=(DIM_X,DIM_Y,DIM_CHANNELS), include_top=False, wei
    ## E03
    # SENet154,_     = aux_models.get('senet154')
    # encoder        = SENet154(input_shape=(DIM_X,DIM_Y,DIM_CHANNELS), include_top=False, weights=NN_WEIGHTS)
    ## E04
    # SEResNeXT101,_ = aux_models.get('seresnext101')
    # encoder        = SEResNeXT101(input_shape=(DIM_X,DIM_Y,DIM_CHANNELS), include_top=False, weights=NN_WEIGHTS)
    ## E05
    # encoder        = tf.keras.applications.DenseNet169(input_shape=(DIM_X,DIM_Y,DIM_CHANNELS), include_top=False, wei

    if (NN_FINETUNE == 'full'):
        # Finetune Full Model
        encoder.Trainable = True
    elif (NN_FINETUNE == 'classifier'):
        # Finetune Classifier Only
        encoder.Trainable = False
    else:
        # Finetune Specific Layers
        layer_names = [layer.name for layer in encoder.layers]
```

```
        fine_tune_at = layer_names.index(NN_FINETUNE)
        for layer in encoder.layers[:fine_tune_at]:
            layer.trainable = False
        print("Finetuning: " + str(len(encoder.trainable_weights)) + "/"
              + str(len(encoder.weights)) + " Encoder Weights Variables")

    # Classifier Layers
    globavgpool_out = tf.keras.layers.GlobalAveragePooling2D()(encoder.output)
    if (DROPOUT == True):
        dense_01 = tf.keras.layers.Dense(512, kernel_regularizer=tf.keras.regularizers.l2(l=0.01), activation='relu')(
            globavgpool_out)
        dropout_01 = tf.keras.layers.Dropout(0.25)(dense_01)
        # dense_02         = tf.keras.layers.Dense(512, kernel_regularizer=tf.keras.regularizers.l2(l=0.01), activatio
        # dropout_02       = tf.keras.layers.Dropout(0.25)(dense_02)
        predictions = tf.keras.layers.Dense(1, kernel_regularizer=tf.keras.regularizers.l2(l=0.01),
                                            activation='sigmoid')(dropout_01)
    else:
        predictions = tf.keras.layers.Dense(1, kernel_regularizer=tf.keras.regularizers.l2(l=0.01),
                                            activation='sigmoid')(globavgpool_out)
    model = tf.keras.models.Model(inputs=encoder.input, outputs=predictions)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=BASE_LR),
              loss='binary_crossentropy',
              metrics=['accuracy'])

print("Training: " + str(len(model.trainable_weights)) + "/"
      + str(len(model.weights)) + " Model Weight Variables")

print('Neural Network Successfully Compiled.')

# Verify Output Shapes (Redundancy Check)
print('Feature Extractor Output: ', encoder(image_batch).shape)
print('Classifier Output: ', model(image_batch).shape)
print('Singular Example Output: ', model(image_batch)[0])
print('Neural Network Successfully Loaded.')
TRAIN_SCHEME = 'resume'
EPOCHS = 2
AUX_EPOCHS = 2

# CLEAN
if (TRAIN_SCHEME == 'clean'):
    # Fit Training Data to Labels
    history = model.fit(train_dataset,
                        steps_per_epoch=round(TRAIN_SAMPLES) // BATCH_SIZE,
                        epochs=EPOCHS,
                        initial_epoch=0,  # Fine-Tuning/Resume Training ("history.epoch[-1]+1")
                        validation_steps=round(VAL_SAMPLES) // BATCH_SIZE,
                        validation_data=val_dataset,
                        callbacks=CALLBACKS)
    # Evaluation Metrics
```

```
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc.insert(0, 0)
    val_acc.insert(0, 0)
    loss.insert(0, 100)
    val_loss.insert(0, 100)

# FREEZE
elif (TRAIN_SCHEME == 'freeze'):
    # Fit Training Data to Labels
    history = model.fit(train_dataset,
                        steps_per_epoch=round(TRAIN_SAMPLES) // BATCH_SIZE,
                        epochs=EPOCHS,
                        initial_epoch=0,  # Fine-Tuning/Resume Training ("history.epoch[-1]+1")
                        validation_steps=round(VAL_SAMPLES) // BATCH_SIZE,
                        validation_data=val_dataset,
                        callbacks=CALLBACKS)

    # Evaluation Metrics
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc.insert(0, 0)
    val_acc.insert(0, 0)
    loss.insert(0, 100)
    val_loss.insert(0, 100)

    for layer in model.layers[:660]:
        layer.trainable = False

    # Fit Training Data to Labels
    history = model.fit(train_dataset,
                        steps_per_epoch=round(TRAIN_SAMPLES) // BATCH_SIZE,
                        epochs=EPOCHS + AUX_EPOCHS,
                        initial_epoch=history.epoch[-1] + 1,  # Fine-Tuning/Resume Training ("history.epoch[-1]+1")
                        validation_steps=round(VAL_SAMPLES) // BATCH_SIZE,
                        validation_data=val_dataset,
                        callbacks=CALLBACKS)

    # Evaluation Metrics
    acc += history.history['accuracy']
    val_acc += history.history['val_accuracy']
    loss += history.history['loss']
    val_loss += history.history['val_loss']

# RESUME
elif (TRAIN_SCHEME == 'resume'):
```

```
    # Fit Training Data to Labels
    history = model.fit(train_dataset,
                        steps_per_epoch=round(TRAIN_SAMPLES) // BATCH_SIZE,
                        epochs=EPOCHS + AUX_EPOCHS,
                        initial_epoch=history.epoch[-1] + 1,  # Fine-Tuning/Resume Training ("history.epoch[-1]+1")
                        validation_steps=round(VAL_SAMPLES) // BATCH_SIZE,
                        validation_data=val_dataset,
                        callbacks=CALLBACKS)

    # Evaluation Metrics
    acc += history.history['accuracy']
    val_acc += history.history['val_accuracy']
    loss += history.history['loss']
    val_loss += history.history['val_loss']
index = 'E01'
export_mode = 'load'  # Load/Save Model             ('save'/'load')

if (export_mode == 'save'):
    # Save Model
    model.save('../models/ensemble/' + index + '_EFN-B6-IMGNET_CLR-5E1E05_448-GW_7E_4mB.h5')
    np.save('../models/ensemble/' + index + '_acc.npy', acc)
    np.save('../models/ensemble/' + index + '_val_acc.npy', val_acc)
    np.save('../models/ensemble/' + index + '_loss.npy', loss)
    np.save('../models/ensemble/' + index + '_val_loss.npy', val_loss)
    print('Trained Model Saved.')

elif (export_mode == 'load'):
    # Recreate Saved Model (architecture, weights, optimizer)
    model = tf.keras.models.load_model('../models/train-val/' + index + '_EFN-B6_RLRP-1E04_CROP-GWHSVLAB_35E_16mB.h5')
    acc = np.load('../models/train-val/ensemble/' + index + '_acc.npy').tolist()
    val_acc = np.load('../models/train-val/ensemble/' + index + '_val_acc.npy').tolist()
    loss = np.load('../models/train-val/ensemble/' + index + '_loss.npy').tolist()
    val_loss = np.load('../models/train-val/ensemble/' + index + '_val_loss.npy').tolist()
    print('Pre-Trained Model Loaded.')
# Load Data
index = 1
image = plt.imread(test_dataset[index]['image'])
scan_id = test_dataset[index]['id']
label = test_dataset[index]['label']

# Crop/Resize to Training Dimensions as Tensors
image_tensor = tf.constant(image, tf.float32)
image_tensor = tf.image.crop_to_bounding_box(image_tensor, (IO_X - 224) // 2,
                                             (IO_Y - 224) // 2,
                                             224, 224)

# Test-Time Augmentation
image01_tensor = tf.clip_by_value(tf.image.flip_left_right(image_tensor), 0.0, 1.0)  # TTA 01: Horizontal Flip
image02_tensor = tf.clip_by_value(tf.image.flip_up_down(image_tensor), 0.0, 1.0)  # TTA 02: Vertical Flip
image03_tensor = tf.clip_by_value(tf.image.adjust_brightness(image_tensor, delta=-16.0 / 255.0), 0.0,
```

```
                                    1.0)  # TTA 03: Brightness Shift
image04_tensor = tf.clip_by_value(tf.image.adjust_saturation(image_tensor, saturation_factor=1.5), 0.0,
                                    1.0)  # TTA 04: Saturation Shift
image05_tensor = tf.clip_by_value(tf.image.adjust_contrast(image_tensor, contrast_factor=1.5), 0.0,
                                    1.0)  # TTA 05: Contrast Boost

plt.figure(figsize=(50, 50))
plt.subplot(161), plt.imshow(image_tensor.numpy()), plt.title('Original'), plt.axis('off')
plt.subplot(162), plt.imshow(image01_tensor.numpy()), plt.title('Flip (H)'), plt.axis('off')
plt.subplot(163), plt.imshow(image02_tensor.numpy()), plt.title('Flip (V)'), plt.axis('off')
plt.subplot(164), plt.imshow(image03_tensor.numpy()), plt.title('Brightness'), plt.axis('off')
plt.subplot(165), plt.imshow(image04_tensor.numpy()), plt.title('Saturation'), plt.axis('off')
plt.subplot(166), plt.imshow(image05_tensor.numpy()), plt.title('Contrast'), plt.axis('off')
plt.show()
# Predictions on Training Dataset
train_predictions = np.zeros((1, 1))
train_labels = np.zeros((1, 1))
for index in tqdm.tqdm(range(0, len(trainX_dataset))):
    # Load Data
    image = plt.imread(trainX_dataset[index]['image'])
    scan_id = trainX_dataset[index]['id']
    label = trainX_dataset[index]['label']
    # Crop/Resize to Training Dimensions as Tensors
    image_tensor = tf.constant(image, tf.float32)
    image_tensor = tf.image.crop_to_bounding_box(image_tensor, (IO_X - DIM_X) // 2,
                                                 (IO_Y - DIM_Y) // 2,
                                                 DIM_X, DIM_Y)

    TTA_prediction = model.predict(np.expand_dims(image_tensor.numpy(), axis=0))

    # Aggregate Predictions
    train_predictions = np.concatenate((train_predictions, TTA_prediction.reshape(-1, 1)), axis=0)
    train_labels = np.concatenate((train_labels, np.array(label).reshape(-1, 1)), axis=0)
train_predictions = train_predictions[1:]
train_labels = train_labels[1:]

TTA = True
TTA_MODE = 'mean'

# Predictions on Validation/Testing Dataset
val_predictions = np.zeros((1, 1))
val_labels = np.zeros((1, 1))
for index in tqdm.tqdm(range(0, len(valX_dataset))):
    # Load Data
    image = plt.imread(valX_dataset[index]['image'])
    scan_id = valX_dataset[index]['id']
    label = valX_dataset[index]['label']
    # Crop/Resize to Training Dimensions as Tensors
    image_tensor = tf.constant(image, tf.float32)
    image_tensor = tf.image.crop_to_bounding_box(image_tensor, (IO_X - DIM_X) // 2,
```

```
                                        (IO_Y - DIM_Y) // 2,
                                        DIM_X, DIM_Y)
    if (TTA == True):
        # Test-Time Augmentation
        image01_tensor = tf.clip_by_value(tf.image.flip_left_right(image_tensor), 0.0, 1.0)  # TTA 01: Horizontal Flip
        image02_tensor = tf.clip_by_value(tf.image.flip_up_down(image_tensor), 0.0, 1.0)  # TTA 02: Vertical Flip
        image03_tensor = tf.clip_by_value(tf.image.adjust_brightness(image_tensor, delta=16.0 / 255.0), 0.0,
                                          1.0)  # TTA 03: Brightness Shift
        image04_tensor = tf.clip_by_value(tf.image.adjust_saturation(image_tensor, saturation_factor=1.25), 0.0,
                                          1.0)  # TTA 04: Saturation Shift
        image05_tensor = tf.clip_by_value(tf.image.adjust_contrast(image_tensor, contrast_factor=1.25), 0.0,
                                          1.0)  # TTA 05: Contrast Boost

        # All Predictions
        all_predictions = np.array((model.predict(np.expand_dims(image_tensor.numpy(), axis=0)),
                                    model.predict(np.expand_dims(image01_tensor.numpy(), axis=0)),
                                    model.predict(np.expand_dims(image02_tensor.numpy(), axis=0)),
                                    model.predict(np.expand_dims(image03_tensor.numpy(), axis=0)),
                                    model.predict(np.expand_dims(image04_tensor.numpy(), axis=0)),
                                    model.predict(np.expand_dims(image05_tensor.numpy(), axis=0))))
        if (TTA_MODE == 'mean'):
            # Average Predictions
            TTA_prediction = np.average(all_predictions)
        elif (TTA_MODE == 'maxconf'):
            # Max Confidence Prediction
            maxconf_arg = np.argmax((abs(0.5 - all_predictions)))
            TTA_prediction = (all_predictions[maxconf_arg]).mean()

    elif (TTA == False):
        TTA_prediction = model.predict(np.expand_dims(image_tensor.numpy(), axis=0))

        # Aggregate Predictions
    val_predictions = np.concatenate((val_predictions, TTA_prediction.reshape(-1, 1)), axis=0)
    val_labels = np.concatenate((val_labels, np.array(label).reshape(-1, 1)), axis=0)
val_predictions = val_predictions[1:]
val_labels = val_labels[1:]
# Accuracy
train_accuracy = sum([np.round(train_predictions)[i] == train_labels[i] for i in range(len(train_labels))]) / len(
    train_labels)
val_accuracy = sum([np.round(val_predictions)[i] == val_labels[i] for i in range(len(val_labels))]) / len(val_labels)

# Sensitivity, Specificity
TN1, FP1, FN1, TP1 = confusion_matrix(val_labels, np.round(val_predictions)).ravel()

# Receiver-Operating Characteristics Curve
M1_TPRS_train = []
M1_AUCS_train = []
M1_mean_FPR_train = np.linspace(0, 0.1, 100)
M1_FPR_train, M1_TPR_train, thresholds = roc_curve(train_labels, train_predictions)
M1_TPRS_train.append(interp(M1_mean_FPR_train, M1_FPR_train, M1_TPR_train))
```

```
M1_TPRS_train[-1][0] = 0.0
M1_ROC_AUC_train = auc(M1_FPR_train, M1_TPR_train)
M1_AUCS_train.append(M1_ROC_AUC_train)

M1_TPRS_test = []
M1_AUCS_test = []
M1_mean_FPR_test = np.linspace(0, 0.1, 100)
M1_FPR_test, M1_TPR_test, thresholds = roc_curve(val_labels, val_predictions)
M1_TPRS_test.append(interp(M1_mean_FPR_test, M1_FPR_test, M1_TPR_test))
M1_TPRS_test[-1][0] = 0.0
M1_ROC_AUC_test = auc(M1_FPR_test, M1_TPR_test)
M1_AUCS_test.append(M1_ROC_AUC_test)

print("Accuracy (Training)   = %0.3f" % train_accuracy)
print("Accuracy (Validation) = %0.3f" % val_accuracy)
print("Sensitivity: {:.3f}".format(TP1 / (TP1 + FN1)))
print("Specificity: {:.3f}".format(TN1 / (TN1 + FP1)))

plt.figure(figsize=[6, 5])
plt.plot(M1_FPR_train, M1_TPR_train, lw=1.5, alpha=1.0, label="AUC (Training) = %0.5f" % M1_ROC_AUC_train,
         color="steelblue")
plt.plot(M1_FPR_test, M1_TPR_test, lw=1.5, alpha=1.0, label="AUC (Validation) = %0.5f" % M1_ROC_AUC_test,
         color="crimson")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc="lower right")
plt.grid(alpha=0.2)
# Pre-Trained ImageNet Encoder for Comparison
imagenet_encoder = efn.EfficientNetB6(input_shape=(DIM_X, DIM_Y, 3), include_top=False, weights='imagenet')
sample = 0

# Target Output Layer of Models (Pre-Trained ImageNet Model, Finetuned Model)
layers_name = ['block2a_expand_activation']
conv1x1op = ['conv2d_2']
imagenet_output = [layer.output for layer in imagenet_encoder.layers
                   if layer.name in layers_name]
finetuned_output = [layer.output for layer in model.layers
                    if layer.name in layers_name]

# Truncated Models for Displaying Feature Maps
imagenet_activation_model = tf.keras.models.Model(imagenet_encoder.inputs, outputs=imagenet_output)
imagenet_activation_model.compile(optimizer='adam', loss='binary_crossentropy')
finetuned_activation_model = tf.keras.models.Model(model.inputs, outputs=finetuned_output)
finetuned_activation_model.compile(optimizer='adam', loss='binary_crossentropy')

# Generate Output Feature Maps
imagenet_feature_map = imagenet_activation_model.predict(image_batch.numpy()[:, :, :, :3] * 255)
plt.figure(figsize=(20, 6))
plt.suptitle("Pre-Trained ImageNet Feature Maps", ha='left', va='bottom', x=0.125, y=0.90, fontsize='xx-large',
             fontweight='regular')
```

```
for i in range(30):
    plt.subplot(3, 10, i + 1), plt.imshow(imagenet_feature_map[sample, :, :, i], cmap='inferno'), plt.axis('off')
plt.show()

finetuned_feature_map = finetuned_activation_model.predict(image_batch.numpy())
plt.figure(figsize=(20, 6))
plt.suptitle("Finetuned Feature Maps", ha='left', va='bottom', x=0.125, y=0.90, fontsize='xx-large',
             fontweight='regular')
for i in range(30):
    plt.subplot(3, 10, i + 1), plt.imshow(finetuned_feature_map[sample, :, :, i], cmap='inferno'), plt.axis('off')
plt.show()
# Pre-Trained ImageNet Encoder for Comparison
imagenet_encoder = efn.EfficientNetB6(input_shape=(DIM_X, DIM_Y, 3), include_top=False, weights='imagenet')
sample = 0

# Target Output Layer of Models (Pre-Trained ImageNet Model, Finetuned Model)
layers_name = ['block2a_expand_activation']
imagenet_output = [layer.output for layer in imagenet_encoder.layers
                   if layer.name in layers_name]
finetuned_output = [layer.output for layer in model.layers
                    if layer.name in layers_name]

# Truncated Models for Displaying Feature Maps
imagenet_activation_model = tf.keras.models.Model(imagenet_encoder.inputs, outputs=imagenet_output)
imagenet_activation_model.compile(optimizer='adam', loss='binary_crossentropy')
finetuned_activation_model = tf.keras.models.Model(model.inputs, outputs=finetuned_output)
finetuned_activation_model.compile(optimizer='adam', loss='binary_crossentropy')

# Truncated Models for Displaying Feature Maps
imagenet_activation_model = tf.keras.models.Model(imagenet_encoder.inputs, outputs=imagenet_output)
imagenet_activation_model.compile(optimizer='adam', loss='binary_crossentropy')
finetuned_activation_model = tf.keras.models.Model(model.inputs, outputs=finetuned_output)
finetuned_activation_model.compile(optimizer='adam', loss='binary_crossentropy')

# Random Noise Image
white_noise0 = (np.random.random((1, 224, 224, 3)) - 0.5) * 20 + 128.0
white_noise1 = (np.random.random((1, 224, 224, DIM_CHANNELS)) - 0.5) * 20 + 128.0

# Cast to Tensor (np.float64 -> tf.float32)
white_noise0 = tf.Variable(tf.cast(white_noise0, tf.float32))
white_noise1 = tf.Variable(tf.cast(white_noise1, tf.float32))

# Iterative Gradient Ascent
epochs = 500
step_size = 1.0
filter_index = 0
for _ in range(epochs):
    with tf.GradientTape() as tape:
        outputs = imagenet_activation_model(white_noise0)
        loss_value = tf.reduce_mean(outputs[:, :, :, filter_index])
    grads = tape.gradient(loss_value, white_noise0)
    normalized_grads = grads / (tf.sqrt(tf.reduce_mean(tf.square(grads))) + 1e-5)
    white_noise0.assign_add(normalized_grads * step_size)
for _ in range(epochs):
    with tf.GradientTape() as tape:
        outputs = finetuned_activation_model(white_noise1)
        loss_value = tf.reduce_mean(outputs[:, :, :, filter_index])
    grads = tape.gradient(loss_value, white_noise1)
    normalized_grads = grads / (tf.sqrt(tf.reduce_mean(tf.square(grads))) + 1e-5)
    white_noise1.assign_add(normalized_grads * step_size)
```