**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Кафедра системного аналізу та теорії прийняття рішень**

**Звіт з виробничої практики на тему:**

**Порівняльний аналіз засобів класифікації зображень**

**студента 4-го курсу**

**Марціленко Івана Володимировича**

**Київ - 2020**

# Зміст

Contents

[Зміст 1](#_Toc51548849)

[Вступ 2](#_Toc51548850)

[Огляд літератури 3](#_Toc51548851)

[Набір даних для навчання та тестування 3](#_Toc51548852)

[Попередня обробка даних 3](#_Toc51548853)

[Критерії оцінки 4](#_Toc51548854)

[Розгляд нейронних мереж 4](#_Toc51548855)

[Згорткові мережі VGG типу 4](#_Toc51548856)

[Архітектура на прикладі VGG - 16 5](#_Toc51548857)

[Варіанти конфігурацій VGG мереж 6](#_Toc51548858)

[Реалізація мережі VGG 7](#_Toc51548859)

[Програмна реалізація за допомогою Keras 8](#_Toc51548860)

[ResNet (Residual Network) - Залишкові нейронні мережі 8](#_Toc51548861)

[Архітектура ResNet 8](#_Toc51548862)

[Реалізація ResNet 12](#_Toc51548863)

[Реалізація простої CNN мережі 13](#_Toc51548864)

[Архітектура мережі 13](#_Toc51548865)

[Програмна реалізація 13](#_Toc51548866)

[Процес навчання 14](#_Toc51548867)

[VGG 15](#_Toc51548868)

[ResNet 50 15](#_Toc51548869)

[Приклад звичайної CNN 16](#_Toc51548870)

[Висновок 18](#_Toc51548871)

[Використана література: 18](#_Toc51548872)

[Додатки з кодом 18](#_Toc51548873)

[Розвідувальний аналіз EDA.ipynb 19](#_Toc51548874)

[Мережа VGG VGGtypeCNN.ipynb 28](#_Toc51548875)

[CustomCNN.py 34](#_Toc51548876)

[ResNet CustomResNet50.ipynb 36](#_Toc51548877)

# Вступ

У роботі розглянуті методи класифікації зображень на прикладі з пневмонією. Були розглянуті кілька видів згорткових мереж, а саме :  
VGG[1][3], ResNet та одна з реалізацій звичайної згорткової мережі, як приклад.  
Розглянуті архітектура, загальні відомості даних типів мереж. Описаний процес навчання. Мною виконана програмна реалізація прикладів, то за допомогою IBM Watson Studio проведено їх навчання.

Також було проведено обробку данних.

# Огляд літератури

Роботу було розпочато з огляду відповідної літератури на цю тему.  
Для початку розглянув інформацію про змагання ImageNet[2] . Та як статтю для перевірки результатів було взято Automated Methods for Detection and Classification Pneumonia based on X-Ray Images Using Deep Learning by Khalid EL ASNAOUI, Youness CHAWKI, Ali IDRI [4]

# Набір даних для навчання та тестування

Для своєї роботи я обрав загальнодоступний набір даних зображень[6], який містить рентгенівські знімки і зображення комп'ютерної томографії (КТ). Цей набір даних, названий набором даних рентгена і КТ грудної клітини, доступний за цим посиланням, складається з 5856 зображень (формат jpeg) і має дві категорії (4273 пневмонії і 1583 нормальних).

# Попередня обробка даних

З розглянутим вище набором данних я провів попередню обробку. Детально описано цей розділ у Додатку 1 EDA.ipynb.  
Я завантажив дані, побачив, що вибірка незбалансована і це треба враховувати у побудові моделі та аналізі результати. Додатково проводив тестові маніпуляції, можна назвати це розвідувальним аналізом. Користувався модулем OpenCV.   
Розглянув методи:

1. Бена Грема
2. Фонового віднімання Background Subtraction.
3. Метод Фур’є
4. Image erosion
5. Image Dilation
6. Canny Edge Detection

# Критерії оцінки

Оскільки на виході роботи методів ми отримаємо класифікації зображень, то усі ситуації можемо описати як:   
Істинно-позитивні TP: True Positive.   
Хибно-позитивні FP: False Positive.  
Істинно-негативні TN: True Negative.

Хибно-негативні FN: False Negative

Після попередньої обробки даних ми маємо збалансовані вибірки. Оскільки однією з важливих частин роботи є порівняння з попередніми результатами у інших роботах, то я використовуватиму класичні метрики якості роботи методів класифікації.

Також у статті[4] використана метрика F1[5]

Використовуючи ці показники й будемо порівнювати наведені нижче методи.

# Розгляд нейронних мереж

## Згорткові мережі VGG типу

[VGG](#_4mk8r3q202d)[1] - Згорткова нейронна мережа запропонована K. Simonyan та A. Zisserman з Оксфордського університету. Модель досягла точності 92.7 % на топ 5, при тестуванні на наборі зображень конкурсу ImageNet у задачі розпізнавання об’єктів на зображення. Одна з самих популярних реалізацій цього типу нейронних мереж це VGG-16 представлена на конкурсі ILSVRC 2014

### Архітектура на прикладі VGG - 16

Використані блоки:

Conv, convolution - процес згортки

Padding - Просторове доповнення

Max polling - Поєднання, просторовий пулінг, інший тип віконного фільтру, у даному випадку обирається найбільший елемент з комірки фільтру і передається далі. Відбувається dawnsampling зменшення кількості параметрів для обробки, за рахунок зменшення зображення

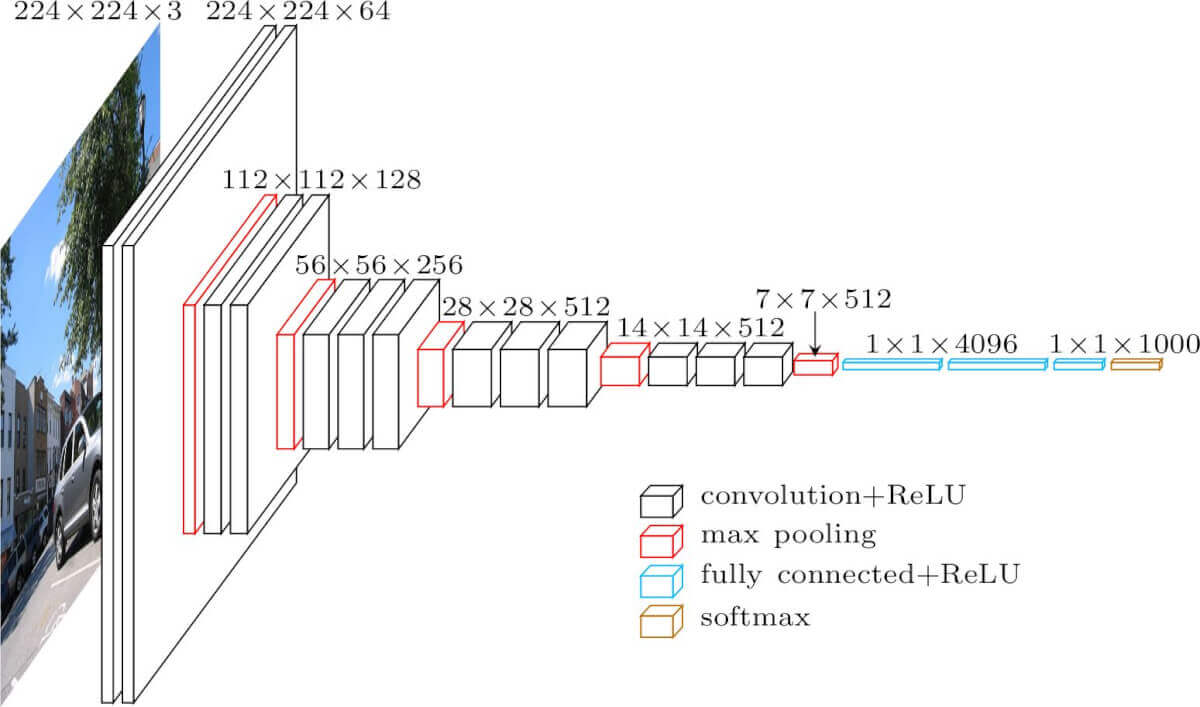
Dense, fully connected -повнозв’язний шар

Softmax - функції активації, по суті це відображення Х-> Y , де Х - останній прихований шар, а Y - ймовірнісний розподіл по класам, себто вихідний клас.  
ReLU - функції активації max(0, x)



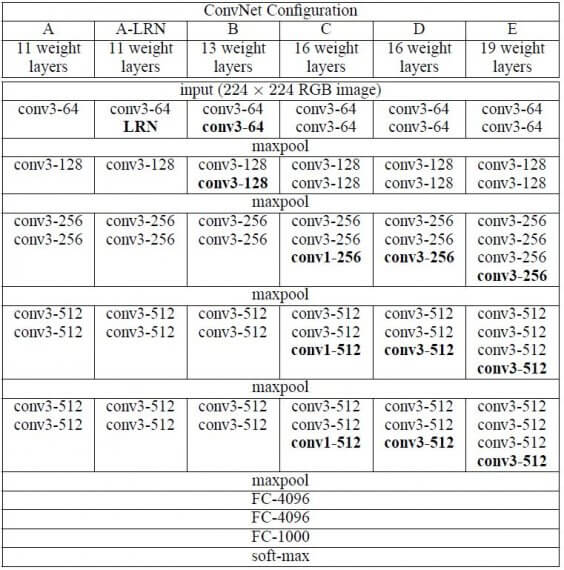
На вхід у шар conv1 подаються RGB зображення розміром 224х224. Далі зображення проходят через набір згорткових шарів, у яких використовуються фільтри за маленьким рецептивним полем розміру 3х3, по суті мінімальний розмір для розуміння де право-ліво, вгору вниз. Просторове доповнення (padding) обирається таким чином, щоб просторовий розмір зберігався після згортки, для шарів 3х3 це дорівнюватиме 1 . Просторовий пулінг (Pooling) відбувається за допомогою п’яти max-pooling шарів. При цьому не за усіма згортковими шарами йде просторовий Pooling. Операція max-pooling відбувається з вікном розміром 2х2 пікселів та кроком 2.

Після набору згорткових слоїв, що може видозмінюватись відповідно до конкретної реалізації, йдуть три повнозв’язних (Dense) шара, перші два мають по 4096 каналів, а третій - 100, оскільки для ILSVRC необхідно класифікувати об’єкти на 100 категорій, себто кожному класу відповідає один канал. Останнім йде soft-msx шар. Конфігурація повнозв’язних шарів однакова для всіх варіантів реалізації VGG. Також усі приховані шари мають RelU функцію активації. Та усі варіанти, крім A-LRN, не мають шару нормалізації (Local Response Normalisation).



### Варіанти конфігурацій VGG мереж

Кожна мережа має свою назву ( А - Е). Усі конфігурації мають спільну конструкцію і відрізняються лише глибиною(від 11 у А, до 19 у Е), при цьому мають невелику ширину ( від 64 у першому шарі до 512 в останньому)

[[3]](#_5s8l7kb45ivo)

### Реалізація мережі VGG

Данний тип мереж має два серйозних недоліка:

1. Дуже повільна швидкість навчання
2. Сама архітектура займає багато пам’яті ( більше 533 МБ для VGG16)

Це робить процес відладки та навчання мереж VGG типу доволі важкою задачею, при цьому вона доволі прямолінійна і її не важко програмно реалізувати.

### Програмна реалізація за допомогою Keras

Мною була програмно реалізована дана відома мережа за допомогою python 3 та tensorflow.keras

## ResNet (Residual Network) - Залишкові нейронні мережі

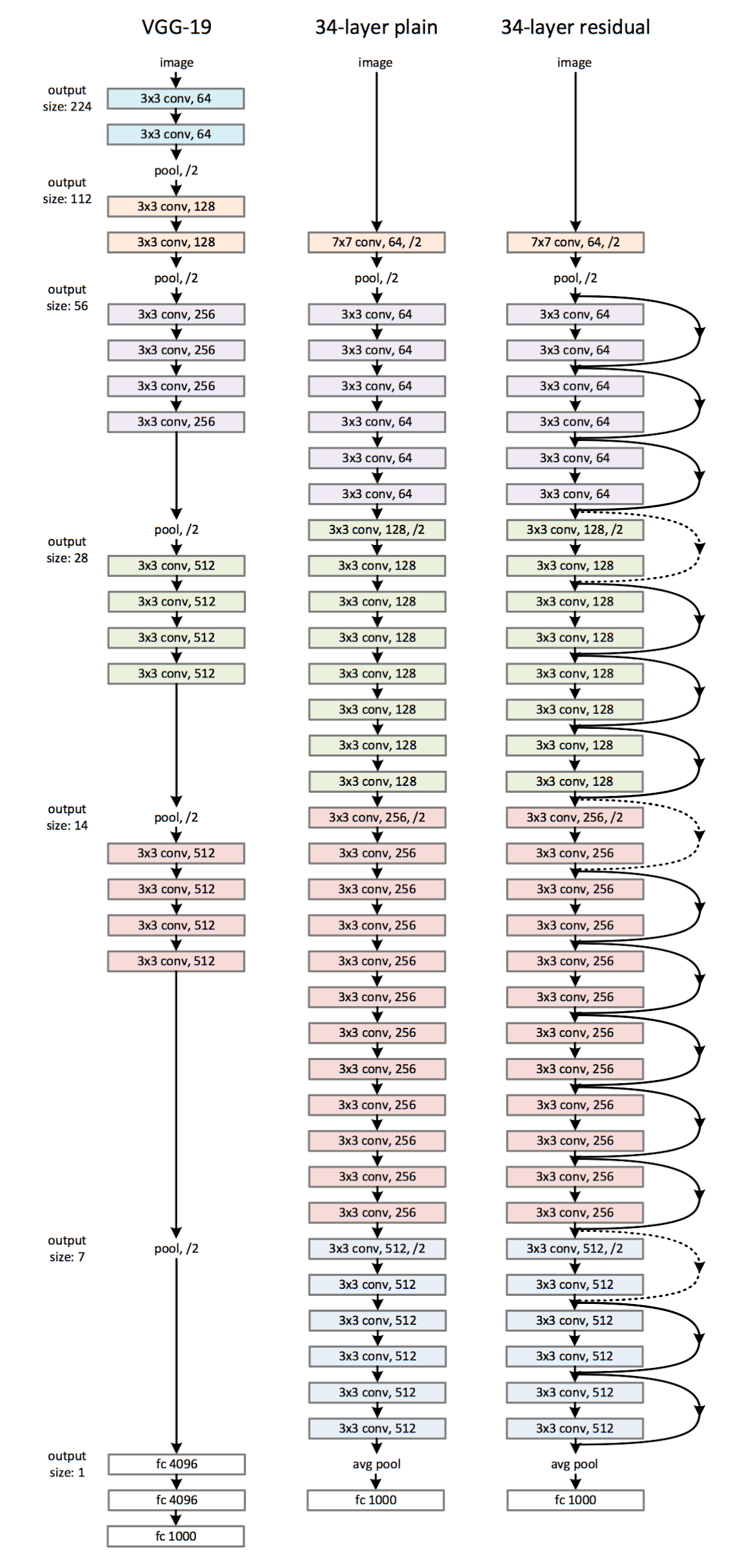
Це класс нейронних мереж, що у своїй структурі використовує швидке з’єднання (shortcut connections), таким чином деякі шари можуть пропускатись і не виконуватимуться обчислення для них. Цей трюк довзволяє збільшувати точність збільшуючи глибину нейронної мережі, обходячи проблему затухаючого градіенту при зворотньому розповсюдженню похибки.

Швидке з'єднання



### Архітектура ResNet

На схемі показано приклад структури мережі дляImageNet. Зліва: модель VGG-19 (19,6 млрд. FLOP) як еталон. Посередині: проста мережа з 34 шарами (3,6 млрд. FLOP). Справа: ResNet з 34 шарами (3,6 миллиарда FLOP). Пунктиром позначені швидкі з’єднання, що збільшують розмірність.



Проста мережа: Прості базові лінії (рис. 2, в центрі) в основному натхненні філософією мереж VGG (рис. 2, зліва). Згорткові шари в основному мають фільтри 3 × 3 і слідують двох простих правил:

Для однієї і тієї ж вихідний карти об'єктів шари мають однакову кількість фільтрів;

Якщо розмір карти об'єктів зменшується вдвічі, число фільтрів подвоюється, щоб зберегти тимчасову складність кожного шару.

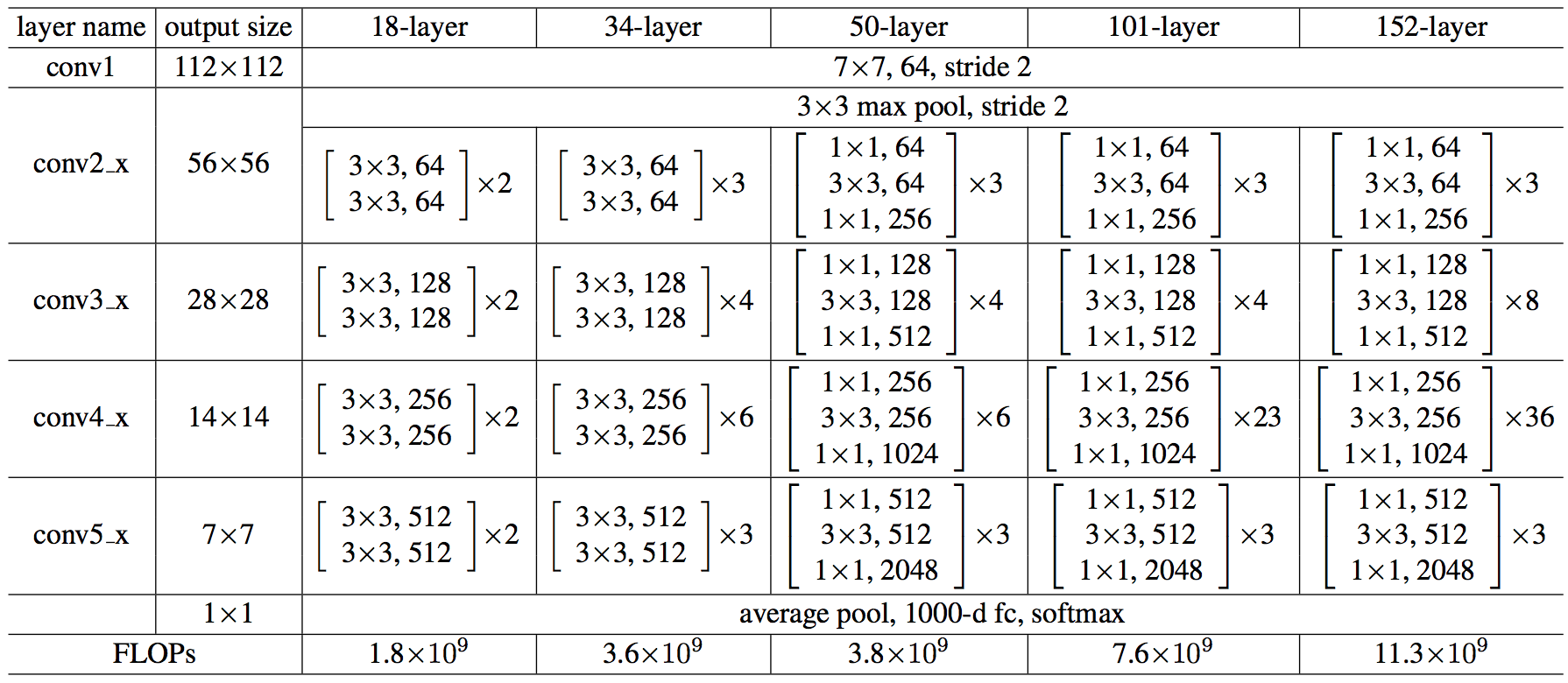
Варто відзначити, що модель ResNet має менше фільтрів і складність менше, ніж мережі VGG.

ResNet: на основі описаної вище простий мережі додано швидке з'єднання (рис. 2, праворуч), яке перетворює мережу в її залишкову версію. Ідентифікаційні швидкі з'єднання F (x {W} + x) можуть використовуватися безпосередньо, коли вхід і вихід мають однакові розмірності (швидкі з'єднання суцільної лінії на рис. 2). Коли розмірності збільшуються (пунктирні лінії на рис. 2), він розглядає два варіанти:

Швидке з'єднання виконує зіставлення ідентифікаторів з додатковими нулями, доданими для збільшення розмірності. Ця опція не вводить ніяких додаткових параметрів.

Проекція швидкого з'єднання в F (x {W} + x) використовується для зіставлення розмірності (виконано за допомогою 1 × 1 згорток).

Для будь-якої з опцій, якщо швидкі з'єднання йдуть по картах об'єктів двох розмірностей, вони виконуються з кроком 2.



Кожен блок ResNet має два рівня глибини (використовується в невеликих мережах, таких як ResNet 18, 34) або 3 рівня (ResNet 50, 101, 152).

50-шарова ResNet: кожен 3-шаровий блок замінюється в 34-шарової мережі цим 3-шаровим вузьким місцем, в результаті виходить 50-шарова ResNet (див. Таблицю вище). Вони використовують варіант 2 для збільшення розмірності. Ця модель має 3,8 мільярда FLOPs.

ResNet з 101 і 152 шарами: вони створюють ResNet з 101 і 152 шарами, використовуючи більше 3-шарових блоків (див. Таблицю вище). Навіть після збільшення глибини 152-шарова ResNet (11,3 мільярда FLOP) має меншу складність, ніж мережі VGG-16/19 (15,3 / 19,6 мільярда FLOPs).

### Реалізація ResNet

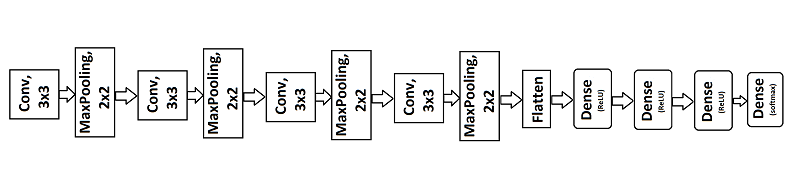
Розмір зображення змінюється за допомогою випадкової вибірки його короткої сторони в [256,480] для збільшення масштабу. Кадрування 224 × 224 вибирається випадковим чином з зображення або його горизонтального зміщення з вирахуванням середнього значення для кожного пікселя. Швидкість навчання стартує з 0,1 і ділиться на 10, коли зміна помилок виходить на плато, моделі навчаються аж до 60 × 10000 ітерацій. Вони використовують зниження ваги 0,0001 та імпульс 0,9.

## Реалізація простої CNN мережі

Як приклад особистої роботи за допомогою пакету Keras реалізував приклад згорткової мережі.

### Архітектура мережі

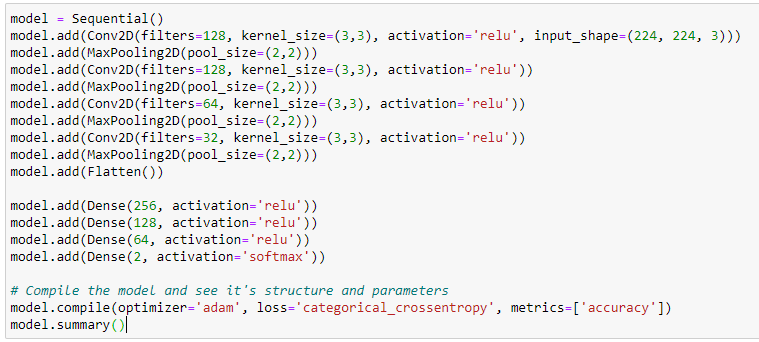
Користуючись блоками згортки (Conv), пулінгу MaxPooling, сплющюючого Flatten шару , та повнозв’язними шара( Dense) з функціями ReLU та Softmax побудував от таку нейронну мережу.



Для початку йдуть 4 блоки що складаются з пари шарів згортки (з ReLU функцією) та пулінгу, після цього йде сплющуючий шар, 3 повнозв’язних шари з ReLU функцією та останній повнозв’язний шар з функцією Softmax.

### Програмна реалізація

Повний код програмної реалізації наведено у додатку 4, нище наведено основна частина з реалізацією архітектури мережі.

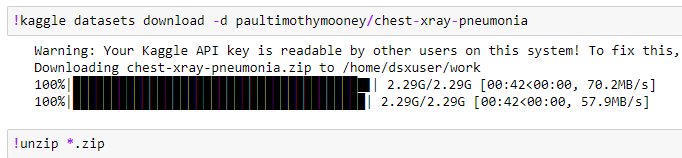


# Процес навчання

Для навчання нейронних мереж я використовував хмарний сервіс IBM Watson Studio. Перевагою такого режиму було те, що я не використовував обчислювальні ресурси власних пристроїв, дані також знаходяться на сервері.   
Для завантаження даних я підключив Kaggle.Api та у кожному проекті ввів необхідні команди



Де замість UserName та Key були введені мої особисті дані.

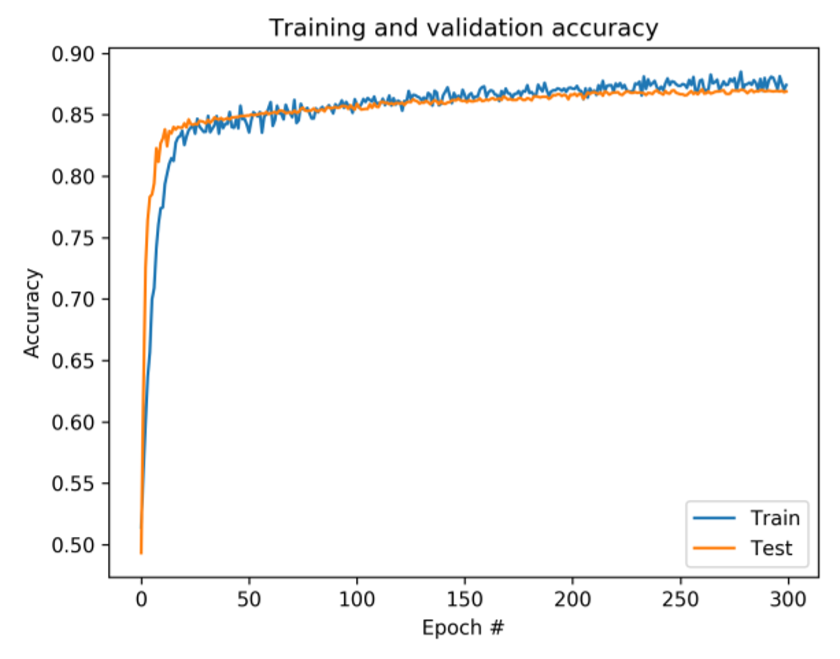
Далі я завантажив дані на свою хмару, та розархівував їх.  


На цьому загальна частина завершилась вже починалась реалізація кожного конкретного методу.

### VGG

Даний тип мереж характерний величезною кількістю параметрів необхідних для навчання, через що сам процес займає тривалий час.

Реалізація наведена у додатку 1.

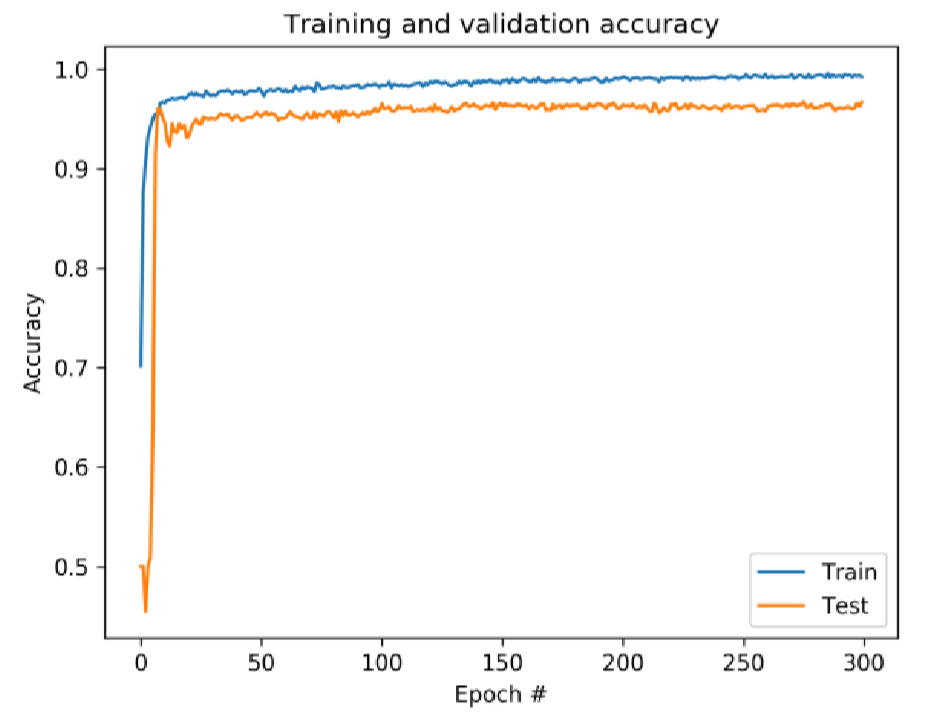


Точність роботи VGG-19 відповідно до [4]

### ResNet 50

Кінцева програмна реалізація наведена у додатку.

У статті [4] мережу ResNet-50 навчали 300 епох, досягши кінцевої точності Accuracy 96.62% при цьому видно, що десь після 50 епохи покращення від кількості епох майже не змінювалось.



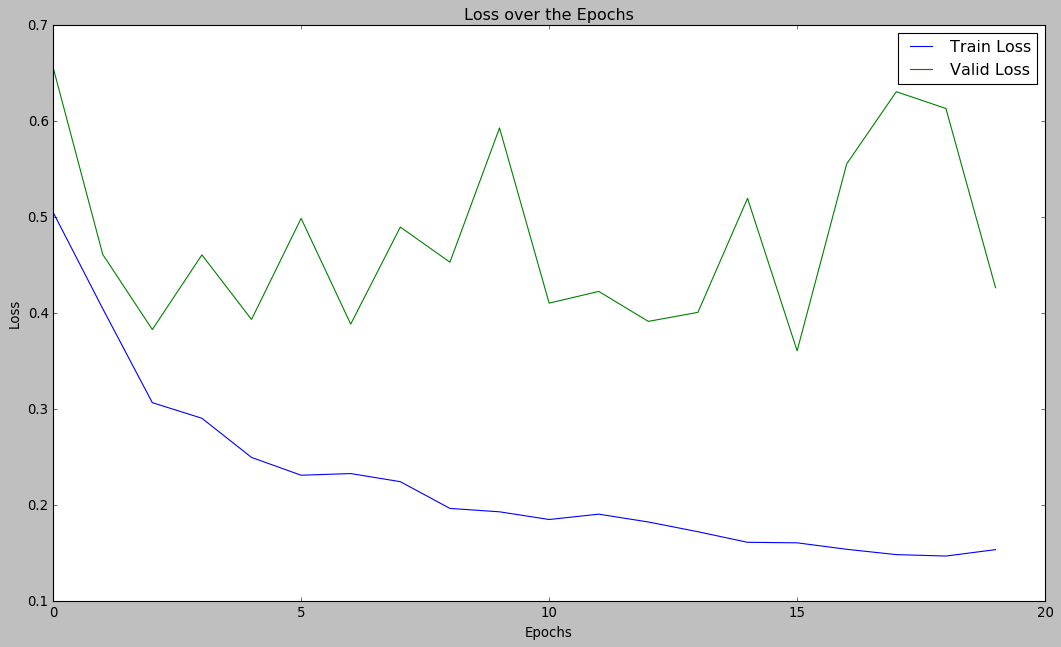
Точність роботи ResNet 50 відповідно до [4]

### Приклад звичайної CNN

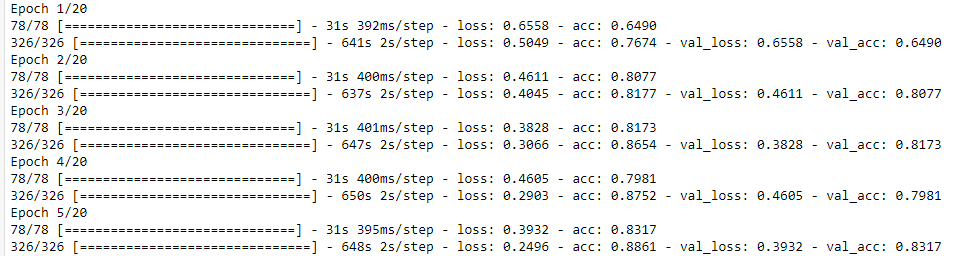
Я так само за допомогою IBM Watson Studio проводив навчання данної мережі.

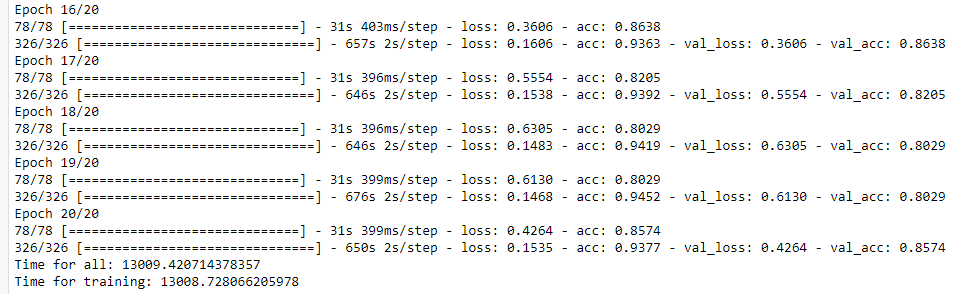
Реалізовану згорткову мережу, код якої наведено у додатку 3 я навчав 20 епох з максимально отриманою точністю 86.38%.

Нище наведені графіки, які я отримав після виконання програми



Графік помилки. Нажаль через помилку у коді графік точності виведено не було, тому я додам скріншоти логів перших та останніх 5 епох.





# Висновок

У процесі виконання мною виробничої практики я навчився користуватись IBM Watson Studio, познайомився з модулем tensorflow.keras, використовуючи який, реалізував мережу VGG типу та приклад згорткової мережі. Провів навчання цих мереж та порівняв результати.  
Мережі ResNet з величезним відривом показує кращу точність, на її фоні різниця між прикладом згорткової мережі та VGG не значний.   
При цьому мережі VGG вимагають значно більше обчислювальних потужностей.

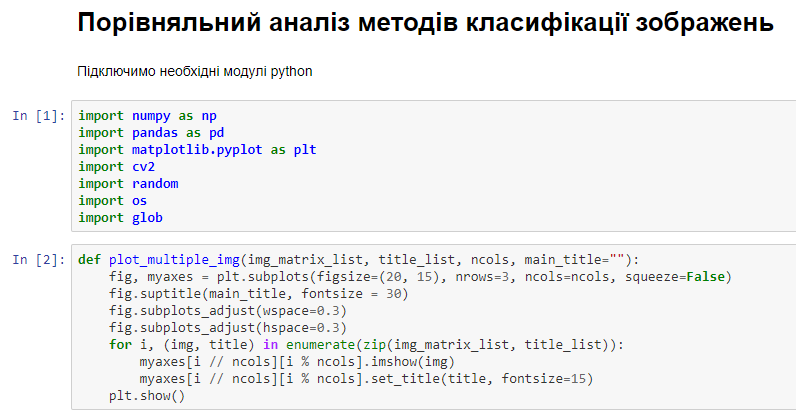
# Використана література:

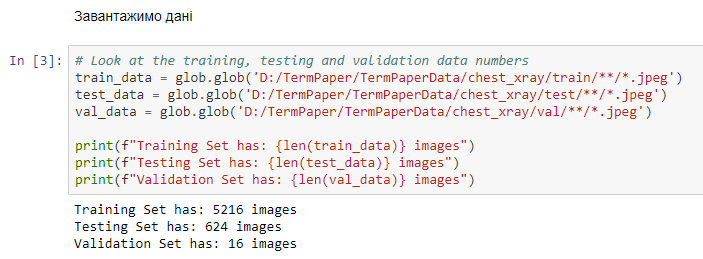
1. [Very Deep Convolutional Networks for Large-Scale Image Recognition](https://arxiv.org/abs/1409.1556) by [Karen Simonyan](https://arxiv.org/search/cs?searchtype=author&query=Simonyan%2C+K), [Andrew Zisserman](https://arxiv.org/search/cs?searchtype=author&query=Zisserman%2C+A)
2. [Results of ILSVRC2014](http://www.image-net.org/challenges/LSVRC/2014/results)
3. <https://neurohive.io/ru/vidy-nejrosetej/vgg16-model/>
4. Automated Methods for Detection and Classification Pneumonia based on X-Ray Images Using Deep Learning by Khalid EL ASNAOUI, Youness CHAWKI, Ali IDRI
5. [The truth of the F-measure by Yutaka Sasaki](https://www.toyota-ti.ac.jp/Lab/Denshi/COIN/people/yutaka.sasaki/F-measure-YS-26Oct07.pdf)
6. <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia>

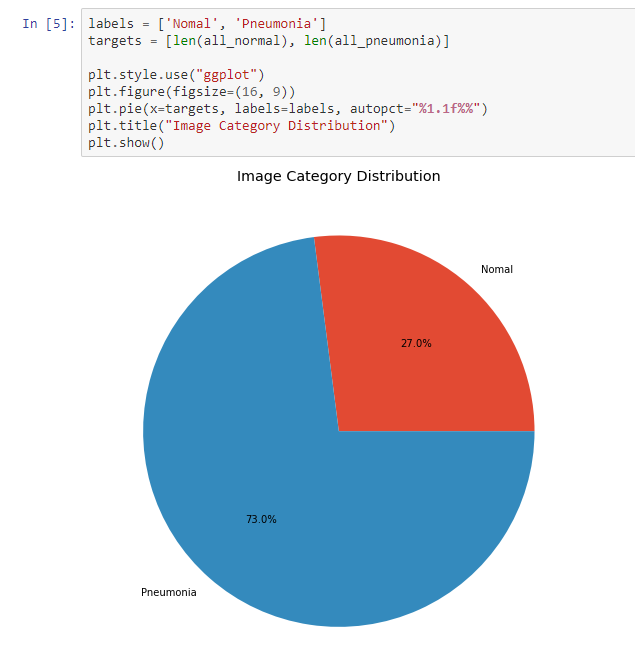
# Додатки з кодом

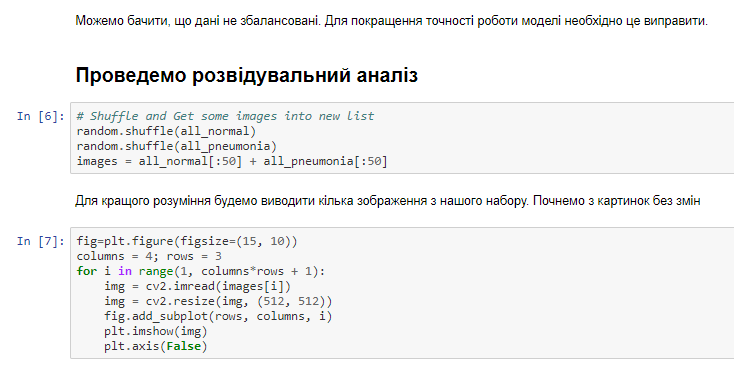
1. Розвідувальний аналіз EDA.ipynb
2. Мережа VGG VGGtypeCNN.ipynb
3. CNN CustomCNN.py
4. ResNet CustomResNet50.ipynb

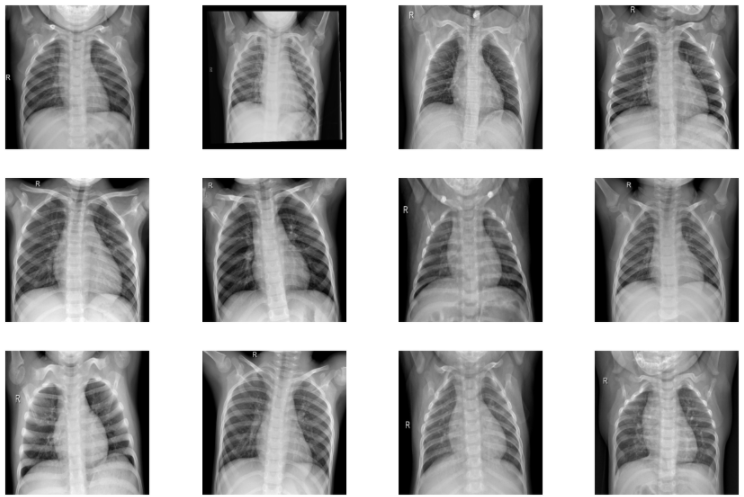
## Розвідувальний аналіз EDA.ipynb

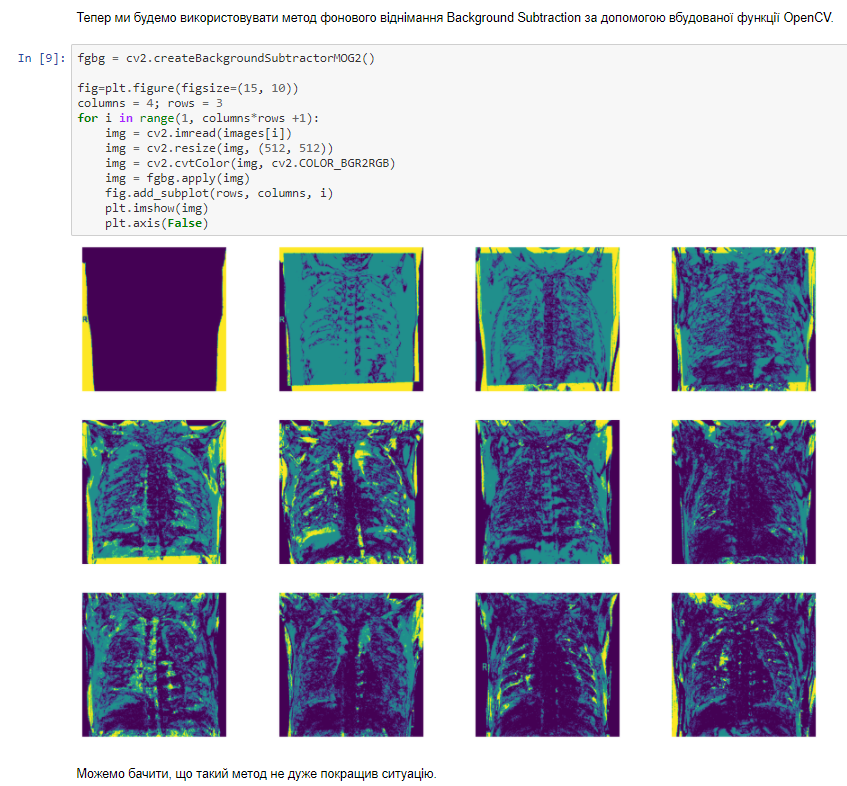


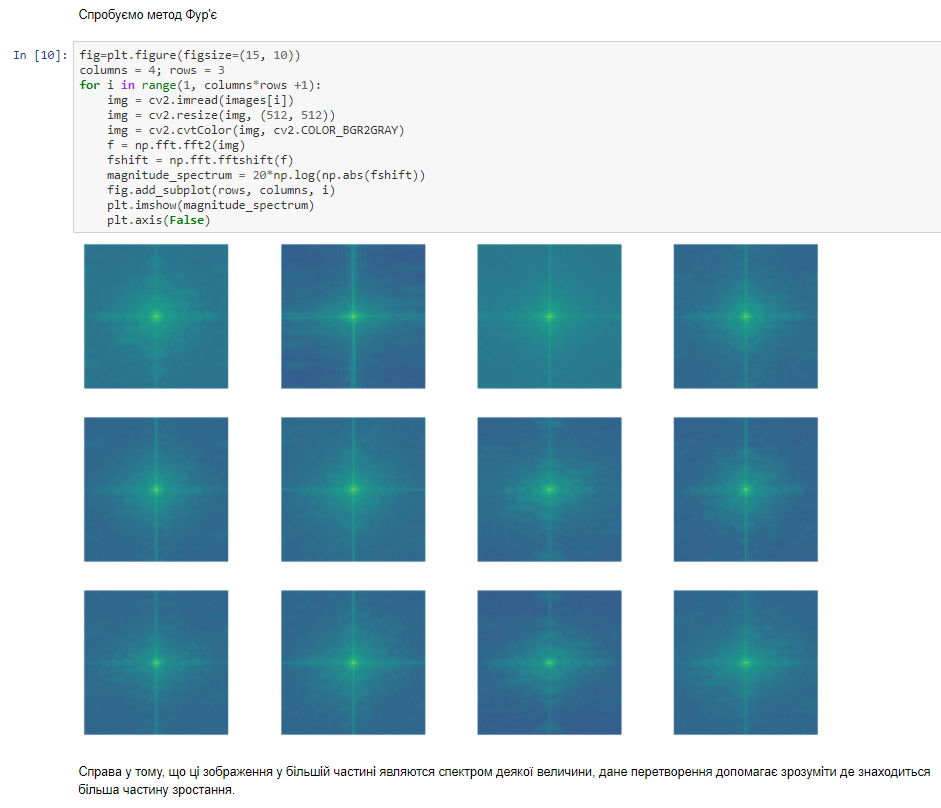


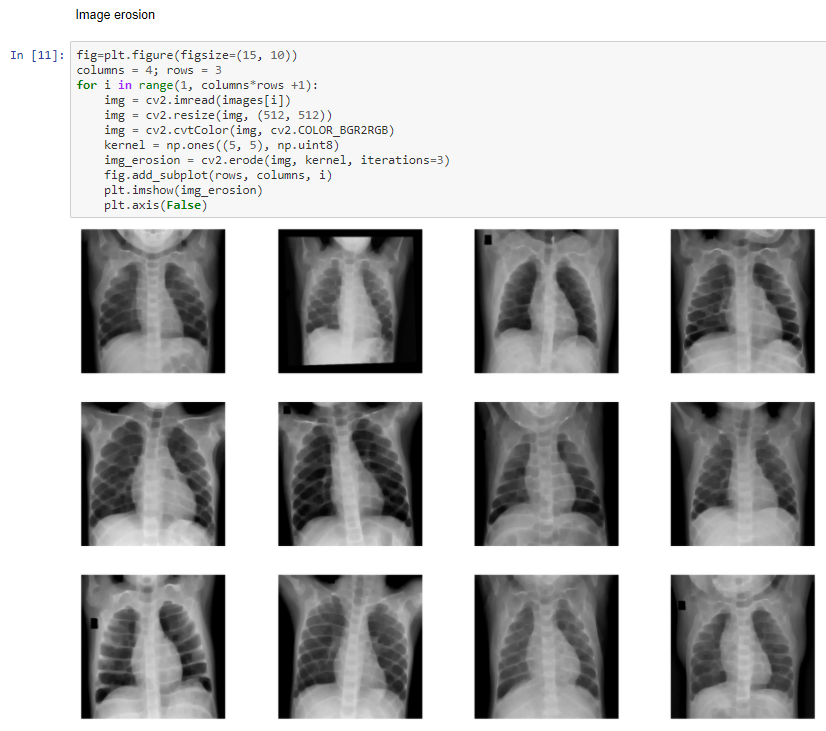


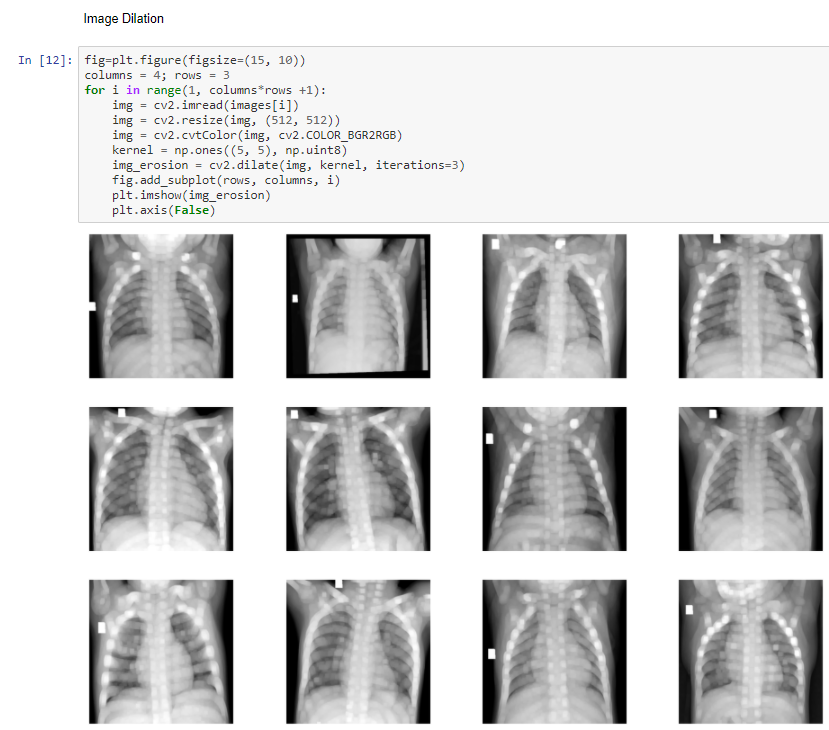


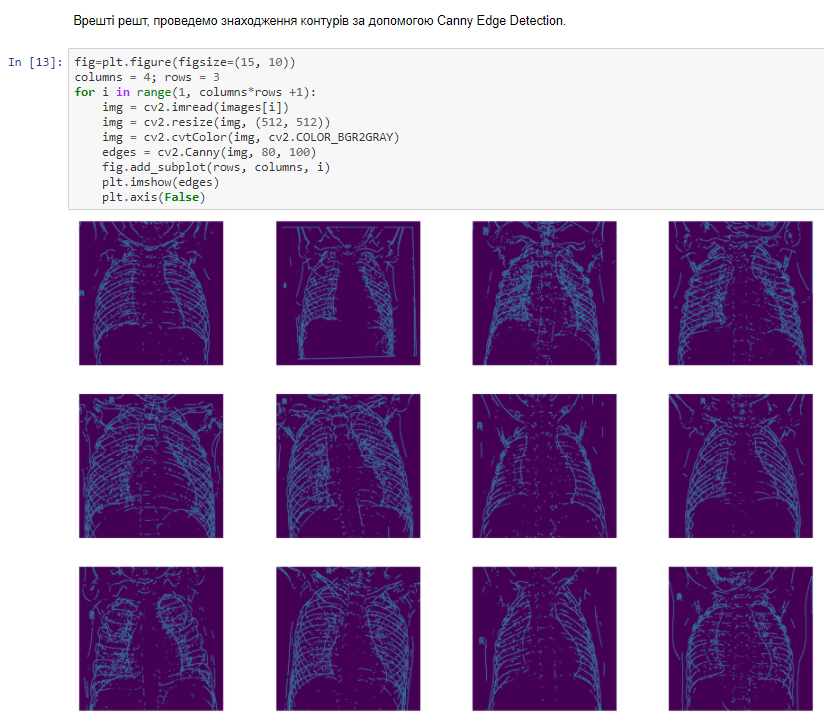






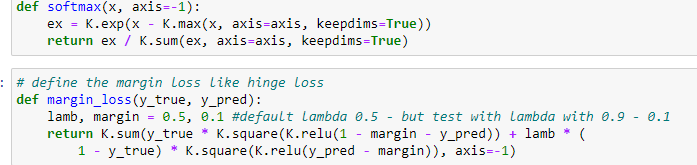




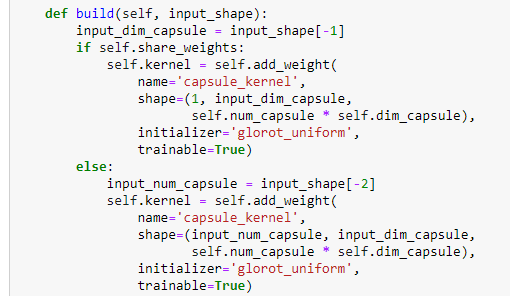


## Мережа VGG VGGtypeCNN.ipynb

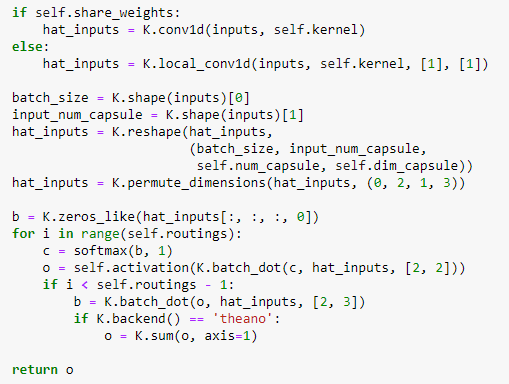


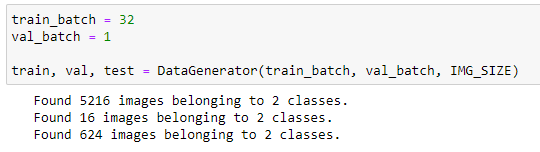


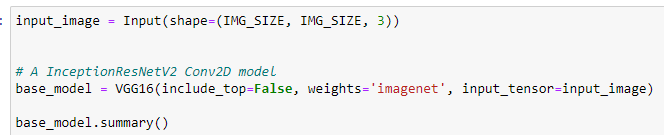


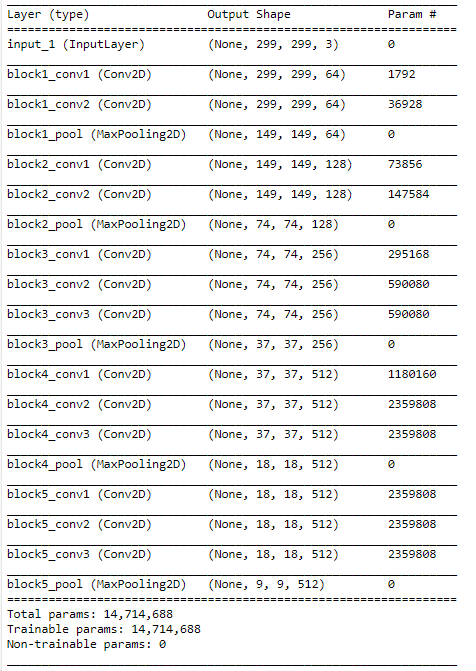




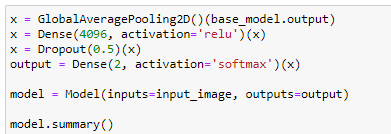


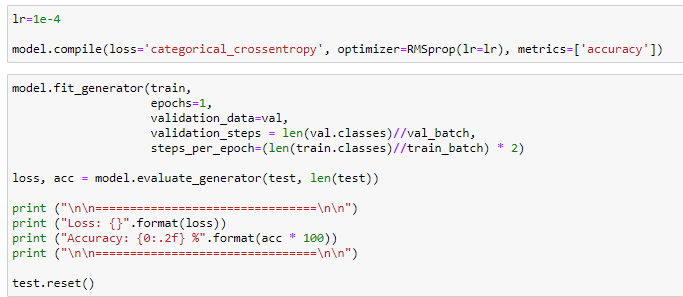


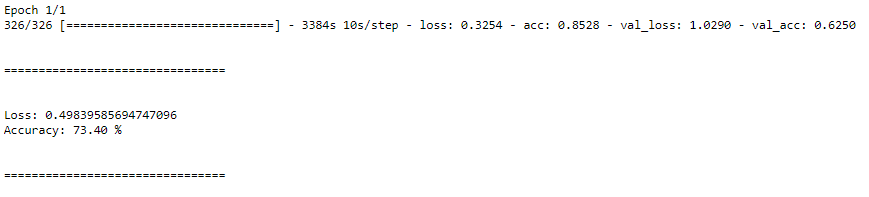


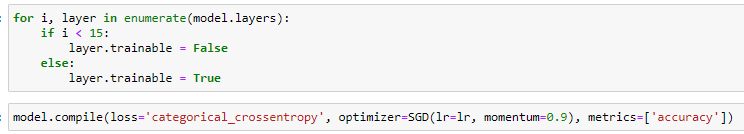


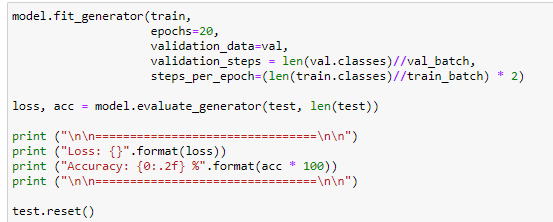


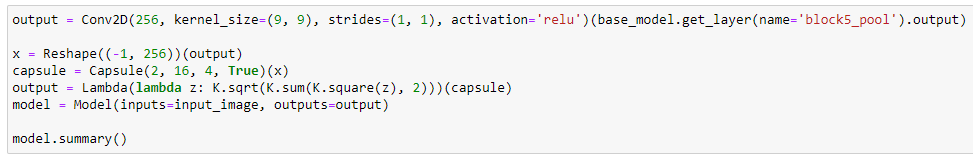












## CustomCNN.py

