



## Розширенна класифікація з використанням машинного навчання

Орієнтовний час, необхідний для виконання: **2** години

### Анотація

Ця лабораторна робота присвячена застосуванню прогресивних методів аналізу великих даних для дослідження поширення COVID-19 у світі. Розглядається метод розпізнавання зображення на рентгенівському знімку грудної клітки для прогнозування діагнозу.

### Вступ

Рентгенівське випромінювання широко використовується в медичній практиці.

Рентгенівські промені можна використовувати для діагностики різних захворювань.

Проте, діагноз також залежить від досвіду лікаря, що може привести до неправильного лікування. Сучасні методи штучного інтелекту та розпізнавання образів дозволяють створювати експертні системи, які здатні автоматично встановлювати діагноз.

В лабораторній роботі буде показано як завантажити зображення, трансформувати їх і визначити основні ознаки, які визначають класифікації захворювань.

Буде розглянуто два різних підходи до класифікації зображень (захворювань):

1. Різні класичні методи та їх порівняння;
2. Згорткові нейронні мережі.

### Матеріали та методи

У цій лабораторній роботі розглянуто основні методи класифікації зображень.

Лабораторна робота складається з чотирьох етапів:

- Завантаження та попередня обробка зображень
- Встановлення ознак зображення
- Порівняння різних класичних методів класифікації

- Побудова та підгонка згорткової нейронної мережі

Статистичні дані отримані з <https://www.kaggle.com/pranavraikokte/covid19-image-dataset> на основі [CC BY-SA 4.0](#) ліцензії.

## Середовища виконання

- Python
- os
- numpy
- glob
- SeABorn
- Matplotlib
- mahotas
- keras
- scikit-learn
- pandas

## Цілі

Після виконання лабораторної роботи ви зможете:

- Завантажувати та обробляти зображення.
- Визначати ознаки зображень.
- Створювати різні моделі класифікації.
- Будувати моделі згорткових нейронних мереж.
- Визначати діагноз на основі рентгенівських зображень.

## Завантаження та попередня обробка зображень

## Встановлення необхідних бібліотек

Для виконання лабораторної роботи потрібно встановити додаткові бібліотеки або оновити встановлені.

```
In [ ]: # conda install matplotlib >= 3.4
```

```
In [ ]: # conda config --add channels conda-forge
```

```
In [1]: pip install mahotas
```

```
Collecting mahotas
  Downloading mahotas-1.4.13-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (5.7 MB)
  ━━━━━━━━━━━━━━━━ 5.7/5.7 MB 58.0 MB/s eta 0:00:000:010
0:01
Requirement already satisfied: numpy in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from mahotas) (1.21.6)
Installing collected packages: mahotas
Successfully installed mahotas-1.4.13
Note: you may need to restart the kernel to use updated packages.
```

```
In [2]: conda install scikit-learn
```

```
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done
```

```
## Package Plan ##
```

```
environment location: /home/jupyterlab/conda/envs/python
```

```
added / updated specs:
- scikit-learn
```

The following packages will be downloaded:

package	build	
ca-certificates-2023.08.22	h06a4308_0	123 KB
certifi-2022.12.7	py37h06a4308_0	150 KB
joblib-1.1.1	py37h06a4308_0	379 KB
openssl-1.1.1w	h7f8727e_0	3.7 MB
scikit-learn-1.0.2	py37h51133e4_1	5.5 MB
threadpoolctl-2.2.0	pyh0d69192_0	16 KB
Total:		9.8 MB

The following NEW packages will be INSTALLED:

```
joblib      pkgs/main/linux-64::joblib-1.1.1-py37h06a4308_0
scikit-learn pkgs/main/linux-64::scikit-learn-1.0.2-py37h51133e4_1
threadpoolctl pkgs/main/noarch::threadpoolctl-2.2.0-pyh0d69192_0
```

The following packages will be UPDATED:

```
ca-certificates    conda-forge::ca-certificates-2023.5.7~ --> pkgs/main::ca-certificates-2023.08.22-h06a4308_0
openssl           conda-forge::openssl-1.1.1t-h0b41bf4_0 --> pkgs/main::openssl-1.1.1w-h7f8727e_0
```

The following packages will be SUPERSEDED by a higher-priority channel:

```
certifi          conda-forge/noarch::certifi-2023.5.7~ --> pkgs/main/linux-64::certifi-2022.12.7-py37h06a4308_0
```

#### Downloading and Extracting Packages

threadpoolctl-2.2.0	16 KB	0%
certifi-2022.12.7	150 KB	0%
ca-certificates-2023	123 KB	0%

scikit-learn-1.0.2	5.5 MB		0%
joblib-1.1.1	379 KB		0%
threadpoolctl-2.2.0	16 KB	#####2	98%
joblib-1.1.1	379 KB	#5	4%
openssl-1.1.1w	3.7 MB	1	0%
certifi-2022.12.7	150 KB	###9	11%
ca-certificates-2023	123 KB	####8	13%
threadpoolctl-2.2.0	16 KB	#####	100%
openssl-1.1.1w	3.7 MB	#####	16%
scikit-learn-1.0.2	5.5 MB	####4	12%
openssl-1.1.1w	3.7 MB	#####2	39%
ca-certificates-2023	123 KB	#####	100%
scikit-learn-1.0.2	5.5 MB	#####5	26%
ca-certificates-2023	123 KB	#####	100%
certifi-2022.12.7	150 KB	#####	100%
certifi-2022.12.7	150 KB	#####	100%
openssl-1.1.1w	3.7 MB	#####4	66%
scikit-learn-1.0.2	5.5 MB	#####1	46%

```
scikit-learn-1.0.2 | 5.5 MB | #####1 | 68%
```

```
scikit-learn-1.0.2 | 5.5 MB | #####7 | 99%
```

```
joblib-1.1.1 | 379 KB | ##### | 100%
```

```
joblib-1.1.1 | 379 KB | ##### | 100%
```

```
openssl-1.1.1w | 3.7 MB | ##### | 100%
```

```
openssl-1.1.1w | 3.7 MB | ##### | 100%
```

```
Preparing transaction: done  
Verifying transaction: done  
Executing transaction: /
```

```
Installed package of scikit-learn can be accelerated using scikit-learn-intelex.  
More details are available here: https://intel.github.io/scikit-learn-intelex
```

For example:

```
$ conda install scikit-learn-intelex  
$ python -m sklearnex my_application.py
```

done

Note: you may need to restart the kernel to use updated packages.

In [3]: `conda install -c conda-forge keras --yes`

```
Collecting package metadata (current_repodata.json): / WARNING conda.models.version:
get_matcher(546): Using .* with relational operator is superfluous and deprecated and will be removed in a future version of conda. Your spec was 1.7.1.*, but conda is ignoring the .* and treating it as 1.7.1
done
Solving environment: done
```

```
==> WARNING: A newer version of conda exists. <==
  current version: 23.3.1
  latest version: 23.10.0
```

Please update conda by running

```
$ conda update -n base -c conda-forge conda
```

Or to minimize the number of packages updated during conda update use

```
conda install conda=23.10.0
```

```
## Package Plan ##
```

```
environment location: /home/jupyterlab/conda/envs/python
```

```
added / updated specs:
  - keras
```

The following packages will be downloaded:

package		build		
ca-certificates-2023.7.22		hbcca054_0	146 KB	conda-forge
certifi-2023.7.22		pyhd8ed1ab_0	150 KB	conda-forge
keras-2.3.1		py37_0	591 KB	conda-forge
libgpuarray-0.7.6		h7f98852_1003	245 KB	conda-forge
mako-1.3.0		pyhd8ed1ab_0	65 KB	conda-forge
openssl-1.1.1w		hd590300_0	1.9 MB	conda-forge
pygpu-0.7.6		py37hb1e94ed_1003	664 KB	conda-forge
theano-1.0.5		py37h745909e_1	3.7 MB	conda-forge
<hr/>				
				Total: 7.3 MB

The following NEW packages will be INSTALLED:

keras	conda-forge/linux-64::keras-2.3.1-py37_0
libgpuarray	conda-forge/linux-64::libgpuarray-0.7.6-h7f98852_1003
mako	conda-forge/noarch::mako-1.3.0-pyhd8ed1ab_0
pygpu	conda-forge/linux-64::pygpu-0.7.6-py37hb1e94ed_1003
theano	conda-forge/linux-64::theano-1.0.5-py37h745909e_1

The following packages will be UPDATED:

certifi	pkgs/main/linux-64::certifi-2022.12.7~ --> conda-forge/noarch::
---------	---

certifi-2023.7.22-pyhd8ed1ab\_0

The following packages will be SUPERSEDED by a higher-priority channel:

ca-certificates pkgs/main::ca-certificates-2023.08.22~ --> conda-forge::ca-certificates-2023.7.22-hbcc054\_0  
openssl pkgs/main::openssl-1.1.1w-h7f8727e\_0 --> conda-forge::openssl-1.1.1w-hd590300\_0

Downloading and Extracting Packages

theano-1.0.5 | 3.7 MB | 0%  
certifi-2023.7.22 | 150 KB | 0%

libgpuarray-0.7.6 | 245 KB | 0%

openssl-1.1.1w | 1.9 MB | 0%

pygpu-0.7.6 | 664 KB | 0%

ca-certificates-2023 | 146 KB | 0%

keras-2.3.1 | 591 KB | 0%

mako-1.3.0 | 65 KB | 0%

libgpuarray-0.7.6 | 245 KB | ##4 | 7%

ca-certificates-2023 | 146 KB | ##### | 11%

openssl-1.1.1w | 1.9 MB | 3 | 1%

pygpu-0.7.6 | 664 KB | 8 | 2%  
certifi-2023.7.22 | 150 KB | #####9 | 11%

keras-2.3.1	591 KB	#	3%
mako-1.3.0	65 KB	#####	25%
openssl-1.1.1w	1.9 MB	#####5	28%
pygpu-0.7.6	664 KB	#####7	72%
ca-certificates-2023	146 KB	#####	100%
theano-1.0.5	3.7 MB	1	0%
theano-1.0.5	3.7 MB	#####3	15%
certifi-2023.7.22	150 KB	#####	100%
theano-1.0.5	3.7 MB	#####5	42%
libgpuarray-0.7.6	245 KB	#####	100%
libgpuarray-0.7.6	245 KB	#####	100%
mako-1.3.0	65 KB	#####	100%
theano-1.0.5	3.7 MB	#####6	75%
openssl-1.1.1w	1.9 MB	#####	100%

```
openssl-1.1.1w      | 1.9 MB    | #####| 100%
```

```
keras-2.3.1          | 591 KB    | #####| 100%
```

```
keras-2.3.1          | 591 KB    | #####| 100%
```

```
Preparing transaction: done  
Verifying transaction: done  
Executing transaction: done
```

Note: you may need to restart the kernel to use updated packages.

```
In [4]: conda install -c conda-forge tensorflow --yes
```

```
Collecting package metadata (current_repodata.json): / WARNING conda.models.version:  
get_matcher(546): Using .* with relational operator is superfluous and deprecated an  
d will be removed in a future version of conda. Your spec was 1.7.1.*, but conda is  
ignoring the .* and treating it as 1.7.1  
done  
Solving environment: done
```

```
==> WARNING: A newer version of conda exists. <==  
    current version: 23.3.1  
    latest version: 23.10.0
```

Please update conda by running

```
$ conda update -n base -c conda-forge conda
```

Or to minimize the number of packages updated during conda update use

```
conda install conda=23.10.0
```

```
# All requested packages already installed.
```

Note: you may need to restart the kernel to use updated packages.

## Імпорт необхідних бібліотек

Для обробки зображень ми будемо використовувати бібліотеку Mahotas і Keras для створення моделі ЗНМ та її навчання. Також буде використовуватись Matplotlib і Seaborn для візуалізації набору даних, щоб краще зрозуміти зображення, які ми збираємося обробляти. Бібліотеки os і glob потрібні нам для роботи з файлами і папками. NumPy буде застосовано до масивів зображень. Scikit-Learn буде використовуватися для класичних моделей класифікації. А для порівняння класифікаторів візьмемо Pandas.

In [5]:

```
import mahotas as mh  
import seaborn as sns  
from matplotlib import pyplot as plt  
from glob import glob  
import os  
import numpy as np  
import pandas as pd  
  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import StandardScaler  
  
#Classifiers  
from sklearn.linear_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC
```

```
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import plot_confusion_matrix
```

## Завантаження даних

```
In [6]: import skillsnetwork

await skillsnetwork.prepare("https://cf-courses-data.s3.us.cloud-object-storage.app"

Downloading Covid19-dataset.zip:  0%|          | 0/165693838 [00:00<?, ?it/s]
  0%|          | 0/672 [00:00<?, ?it/s]
Saved to '.'
```

Для класифікації зображення мають бути однакового розміру. Щоб досягти цього, створимо глобальну змінну, яка визначатиме розмір (висоту та ширину) зображення та будемо використовувати її для зміни розмірів. В нашому випадку обидва 224.

```
In [7]: IMM_SIZE = 224
```

Для зручності створюємо функцію, яка завантажує та відображає всі зображення із зазначеного каталогу. Для того, щоб класифікувати зображення, всі вони повинні бути розміщені в підкаталогах. Назви цих підкаталогів насправді є іменами класів. У нашому випадку зображення є рентгенівськими знімками, які потрібно помістити у підкаталоги з назвами діагнозів. Наприклад, підкаталог COVID має містити рентгенівські знімки людей із цією хворобою. Перш за все, створимо список підкаталогів, який містить можливі класи захворювань. У нашему випадку це буде: Норма, COVID, Вірусна пневмонія.

Далі нам потрібно створити список усіх зображень. Після цього завантажимо та обробимо всі зображення:

1. Завантажуємо за допомогою `mahotas.imread()`
2. Необхідно змінити розмір зображень до (IMM\_SIZE x IMM\_SIZE). Якщо зображення сірого кольору, воно подається у вигляді 2D-матриці: (висота, ширина). jpg та png зображення мають 3D формат (висота, ширина, 3 або 4). Щоб це зробити треба використати: `mahotas.resize_to()`
3. Якщо третій параметр формату зображення дорівнює 4, це означає альфа-канал. Ми можемо видалити його за допомогою зрізу зображення `image[:, :, :3]`.
4. Потім нам потрібно перетворити всі зображення в сірий 2D формат за допомогою: `mahotas.colors.rgb2grey()`

Функція повертає масив кортежів [image, class name].

```
In [8]: def get_data(folder):
    class_names = [f for f in os.listdir(folder) if not f.startswith('.')] # create
    data = []
    print(class_names)
    for t, f in enumerate(class_names):
        images = glob(folder + "/" + f + "/*") # create a list of files
        print("Downloading: ", f)
        fig = plt.figure(figsize = (50,50))
        for im_n, im in enumerate(images):
            plt.gray() # set grey colormap of images
            image = mh.imread(im)
            if len(image.shape) > 2:
                image = mh.resize_to(image, [IMM_SIZE, IMM_SIZE, image.shape[2]]) #
            else:
                image = mh.resize_to(image, [IMM_SIZE, IMM_SIZE]) # resize of grey
            if len(image.shape) > 2:
                image = mh.colors.rgb2grey(image[:, :, :3], dtype = np.uint8) # chan
            plt.subplot(int(len(images)/5)+1, 5, im_n+1) # create a table of images
            plt.imshow(image)
            data.append([image, f])
    plt.show()

    return np.array(data)
```

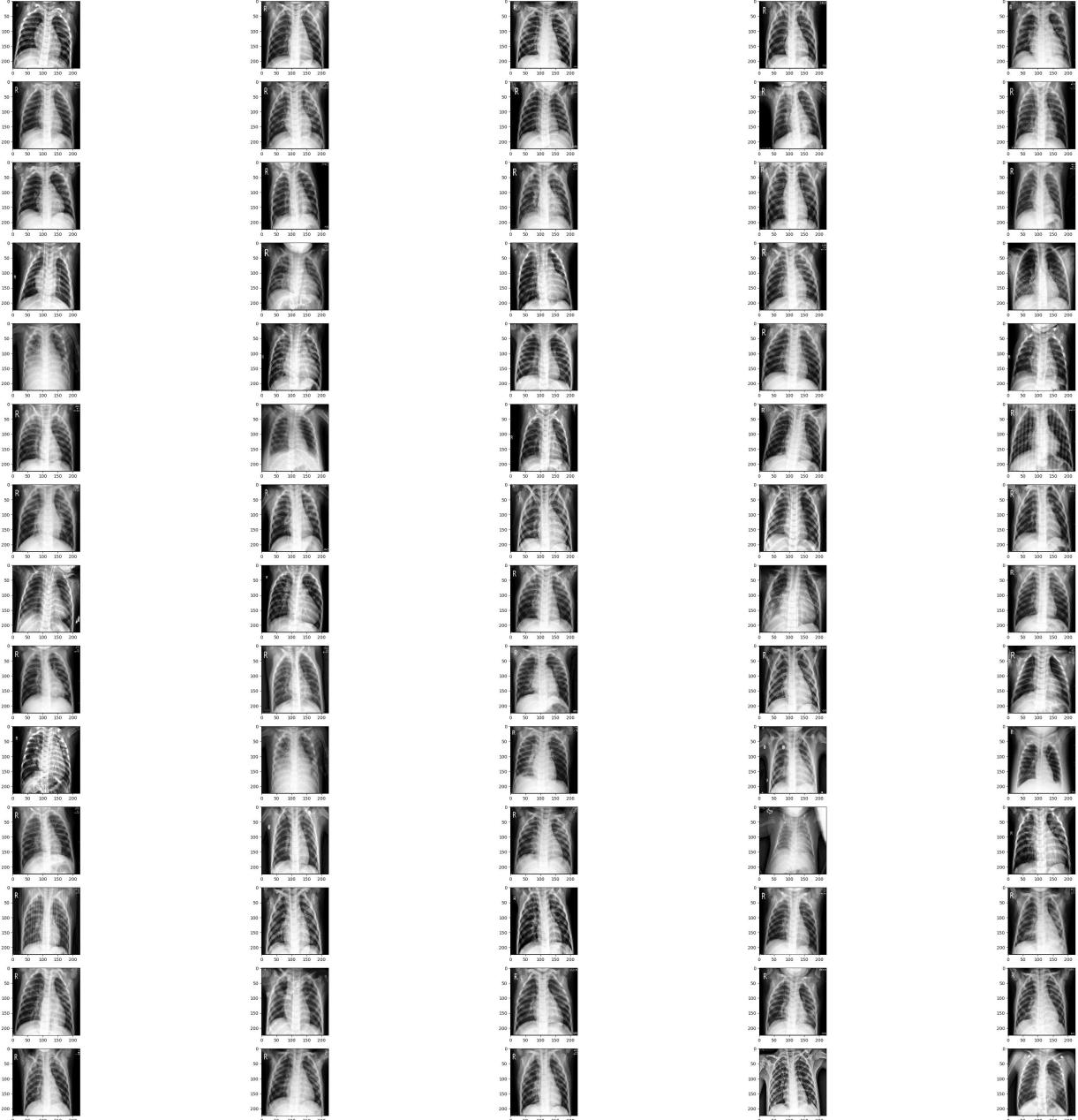
Для навчання та тестування всі зображення слід розділити на навчальні та тестові групи та розмістити в окремих папках. Завантажимо всі зображення у відповідні масиви.

```
In [9]: d = "Covid19-dataset/train"
train = get_data(d)
```

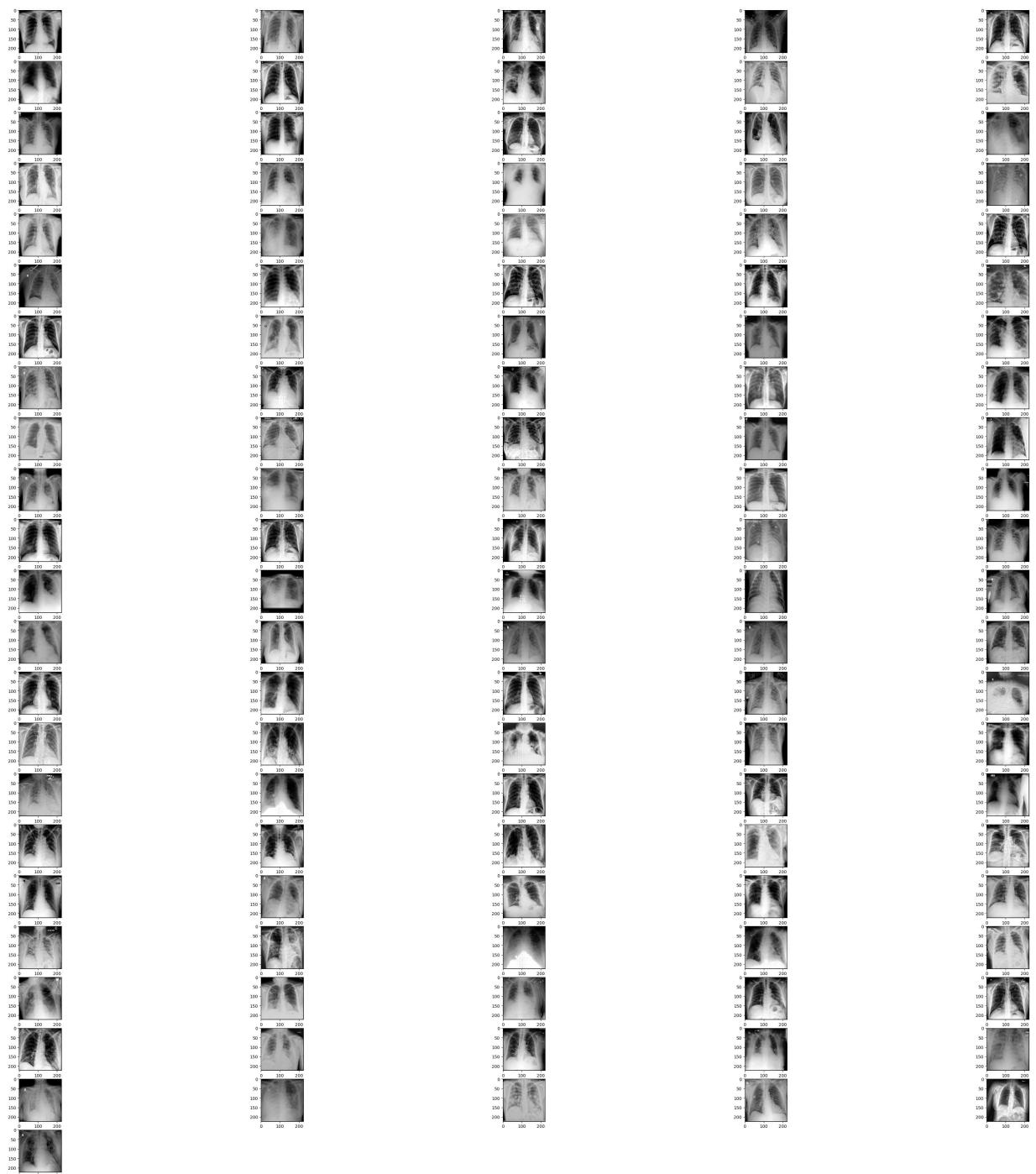
```
d = "Covid19-dataset/test"
val = get_data(d)
```

[ 'Viral Pneumonia', 'Covid', 'Normal' ]

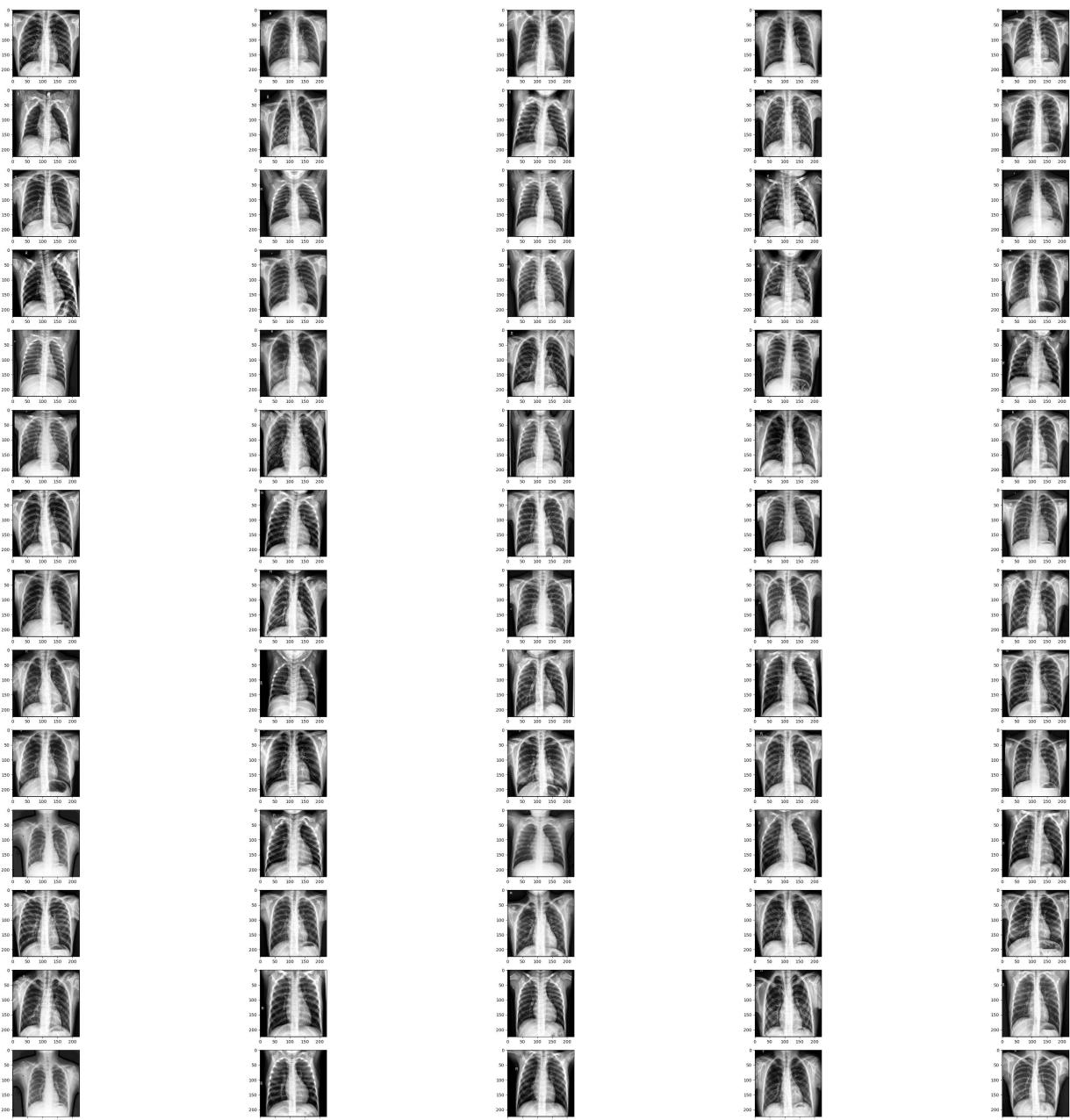
Downloading: Viral Pneumonia



Downloading: Covid



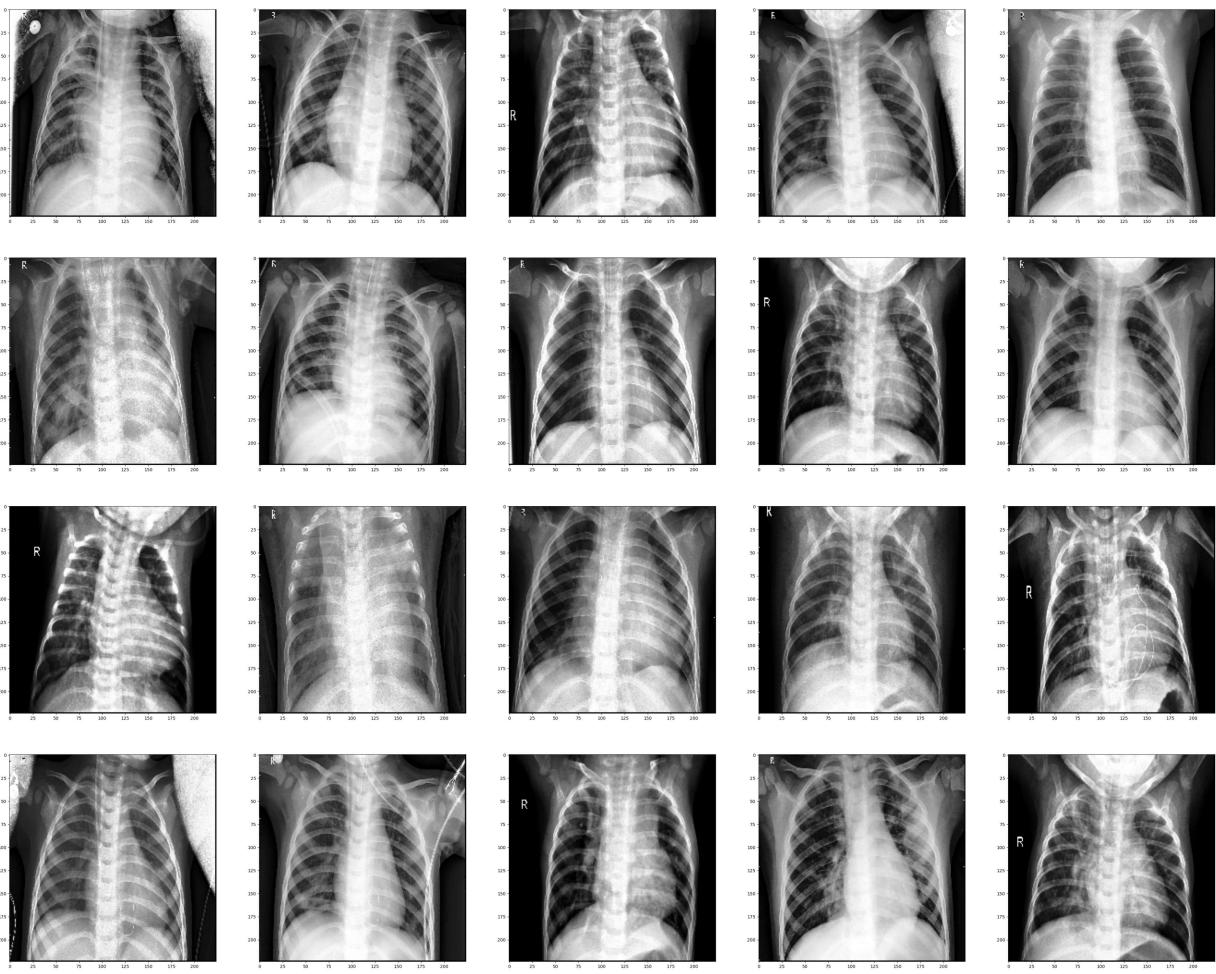
Downloading: Normal



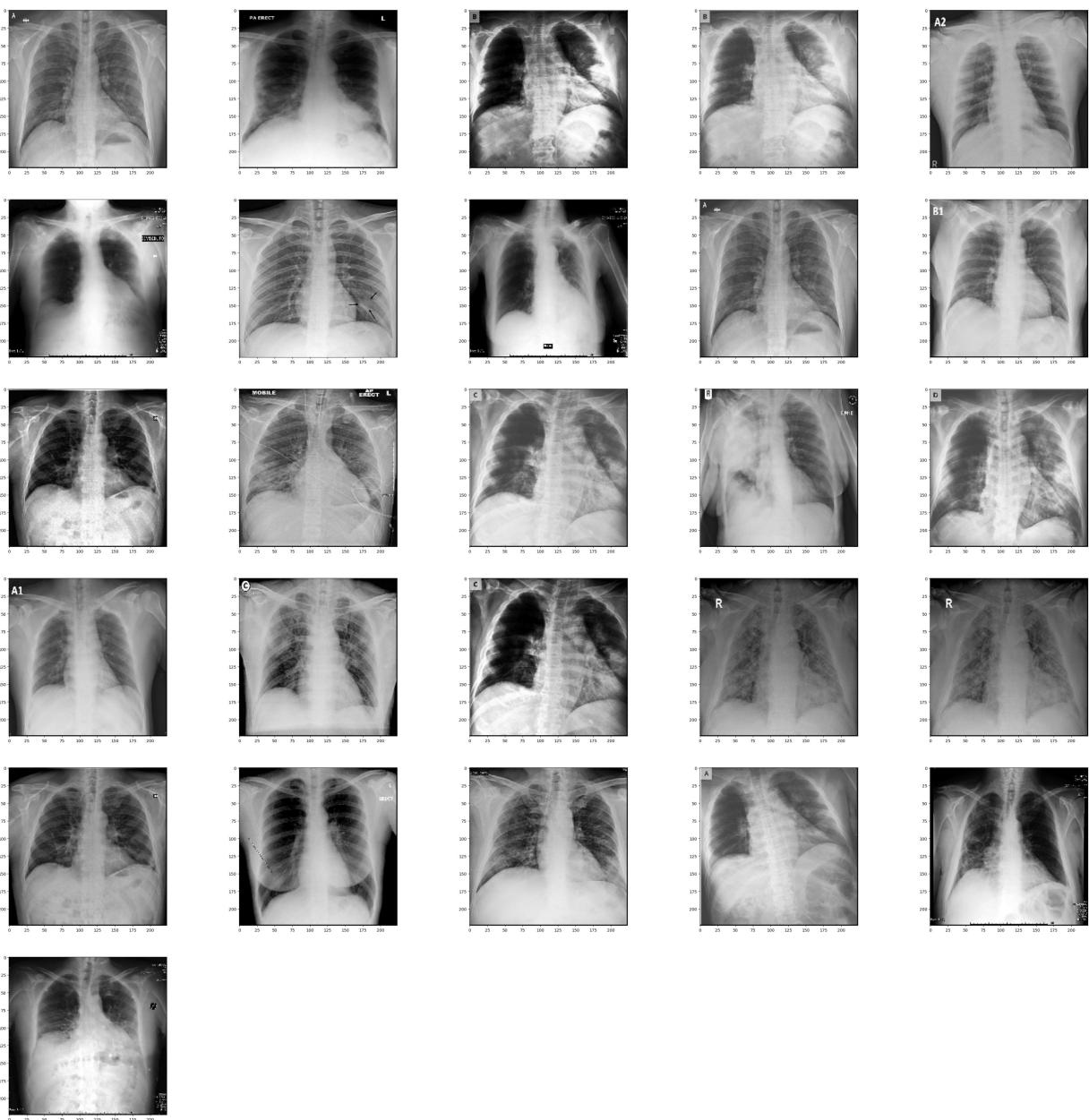
```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launcher.py:23: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
```

['Viral Pneumonia', 'Covid', 'Normal']

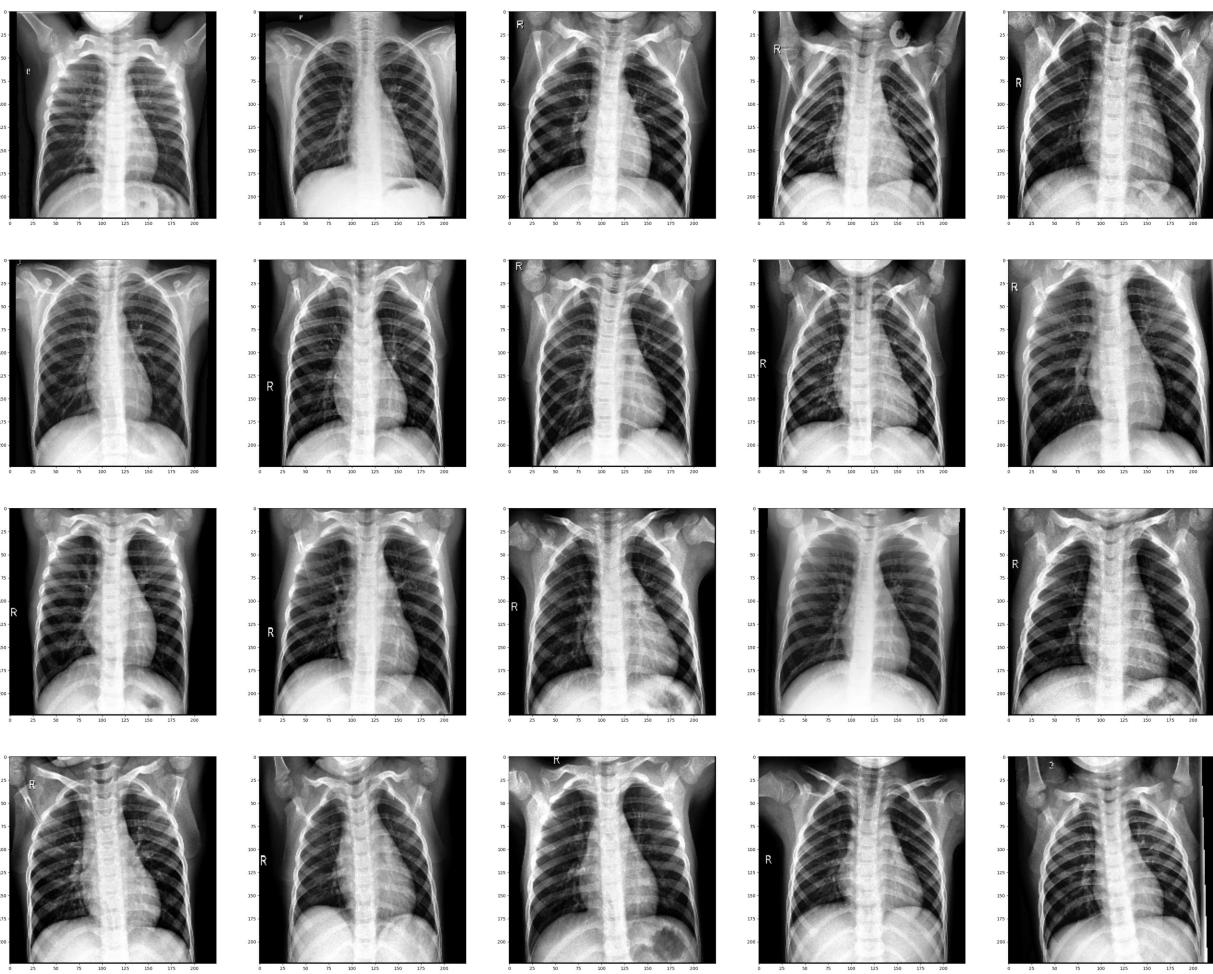
Downloading: Viral Pneumonia



Downloading: Covid



Downloading: Normal



```
In [10]: print("Train shape", train.shape) # Size of the training DataSet
print("Test shape", val.shape) # Size of the test DataSet
print("Image size", train[0][0].shape) # Size of image
```

Train shape (251, 2)

Test shape (66, 2)

Image size (224, 224)

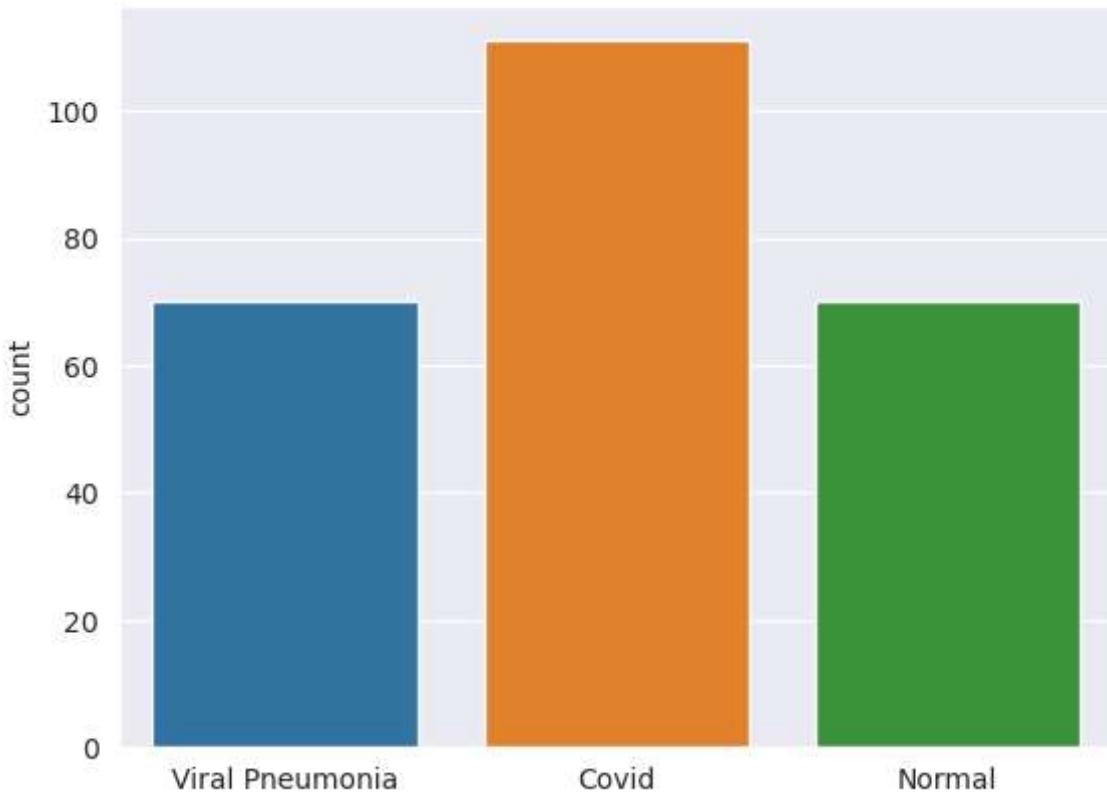
Як бачимо, навчальний Набір даних складається з 251 зображення, а тестовий – із 66 зображень. Усі зображення у сірому 2D форматі (224x224).

## Візуалізація даних

Візуалізуємо дані і подивимося, з чим саме ми працюємо. Використовуємо Seaborn для побудови графіка кількості зображень у всіх класах. Ви можете побачити, як виглядає результат.

```
In [11]: l = []
for i in train:
    l.append(i[1])
sns.set_style('darkgrid')
sns.countplot(l)
```

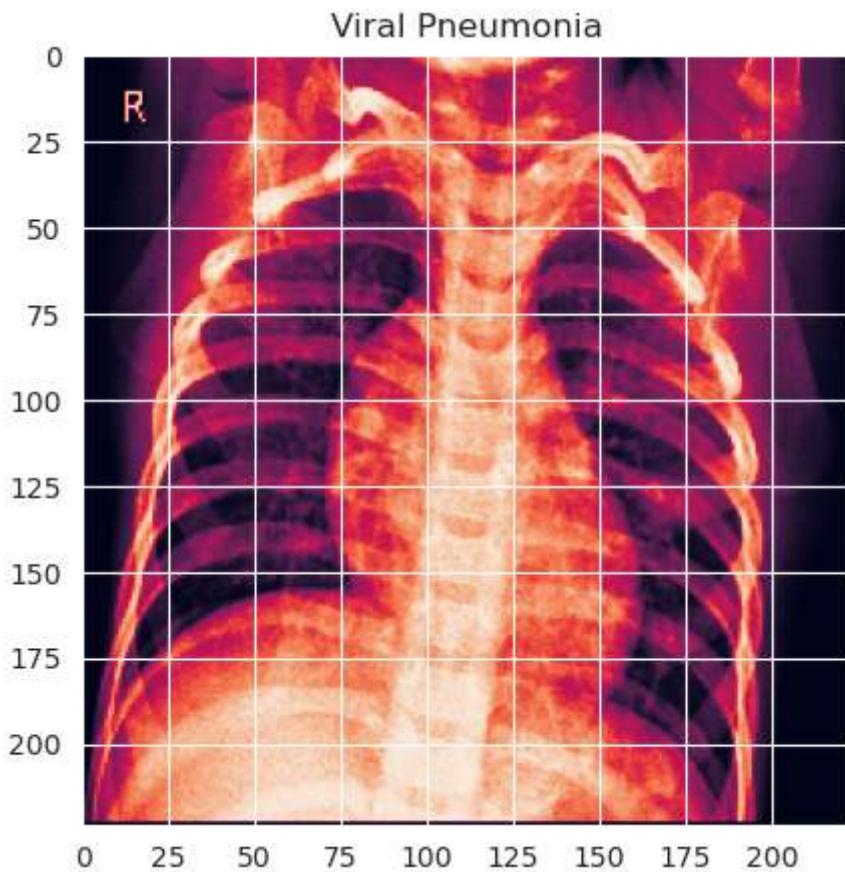
Out[11]: <AxesSubplot:ylabel='count'>



Також візуалізуємо перше зображення з класів Вірусна пневмонія та Covid у навчальному Наборі даних:

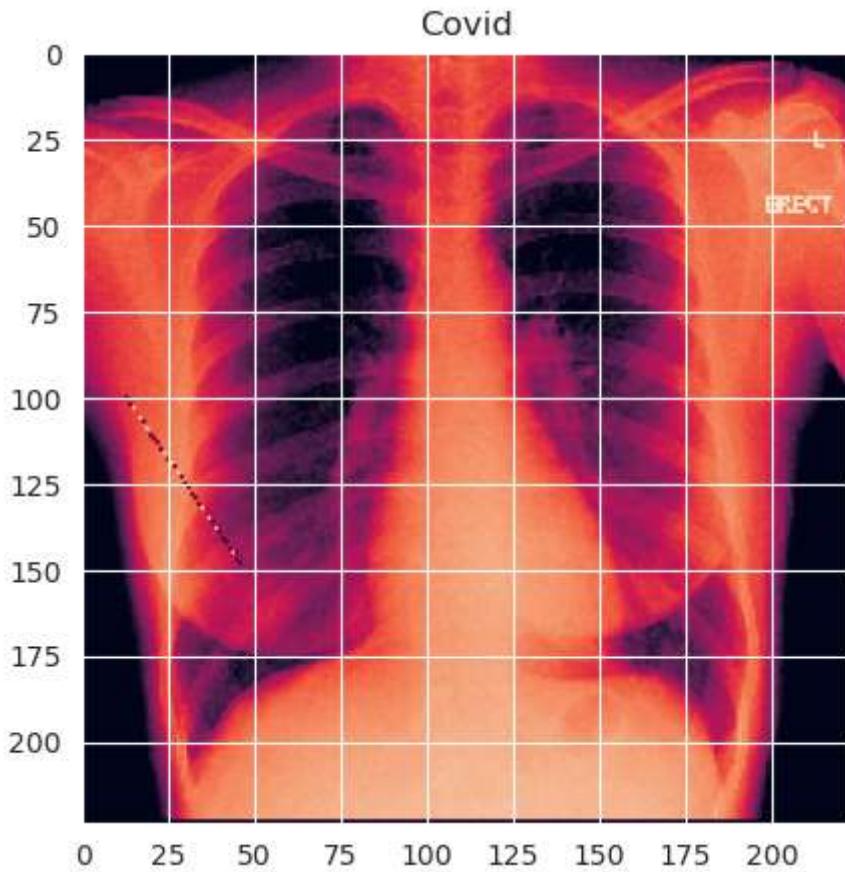
```
In [12]: plt.figure(figsize = (5,5))
plt.imshow(train[np.where(train[:,1] == 'Viral Pneumonia')[0][0]][0])
plt.title('Viral Pneumonia')
```

```
Out[12]: Text(0.5, 1.0, 'Viral Pneumonia')
```



```
In [13]: plt.figure(figsize = (5,5))
plt.imshow(train[np.where(train[:,1] == 'Covid')[0][0]][0])
plt.title('Covid')
```

```
Out[13]: Text(0.5, 1.0, 'Covid')
```



## Встановлення ознак зображення

Щоб класифікувати об'єкти, потрібно перетворити набір даних таким чином, щоб вхідними даними був набір ознак, а виходом — клас об'єктів. Зображення - це матриця пікселів. Кожен піксель є кольором. Тому неможливо подати безпосередньо зображення на вхід класичного класифікатора. Необхідно перетворити кожне зображення в набір певних ознак.

Mahotas дозволяє легко розрахувати особливості зображення. Відповідні функції можна знайти в підмодулі `mahotas.features`. Набір текстурних ознак Haralick добре відомий. Як і багато алгоритмів обробки зображень, він названий на честь свого винахідника. Ознаки визначаються на основі текстур, тобто розрізняють структуровані і неструктуровані зображення, а також різноманітні повторювані структури. За допомогою Mahotas ці ознаки дуже легко розраховуються: `haralick_features = mh.features.haralick (image)` `haralick_features_mean = np.mean (haralick_features, axis = 0)` `haralick_features_all = np.ravel (haralick_features)` Функція `mh.features.haralick` повертає масив  $4 \times 13$ . Перший вимір — це чотири можливі напрямки, в яких обчислюються ознаки (вертикаль, горизонталь і дві діагоналі). Якщо якийсь конкретний напрямок не важливий для моделі, ми можемо усереднювати ознаки в усіх напрямках (у коді вище ця змінна називається `haralick_features_mean`). Крім того, можна використовувати всі ознаки окремо (змінна `haralick_features_all`). Вибір залежить від властивостей

конкретного набору даних. В нашому випадку вертикальні та горизонтальні об'єкти будуть зберігатися окремо, тому використовуємо haralick\_features\_all. Далі цей масив необхідно перетворити в одномірний за допомогою NumPy.ravel().

Створимо функцію для виділення ознак Набору даних.

```
In [14]: def create_features(data):
    features = []
    labels = []
    for image, label in data:
        features.append(mh.features.haralick(image).ravel())
        labels.append(label)
    features = np.array(features)
    labels = np.array(labels)
    return (features, labels)
```

```
In [15]: features_train, labels_train = create_features(train)
features_test, labels_test = create_features(val)
```

## Порівняння різних класичних методів класифікації

Для порівняння кількох класифікаторів зручно використовувати конвеєр. Конвеєр допомагає об'єднати кілька оціночних функцій в одну. Це зручно, оскільки в обробці даних часто є фіксована кількість кроків, наприклад, вибір ознак, нормалізація та класифікація. Конвеєр тут виконує кілька цілей:

Потрібно лише один раз викликати fit(), щоб оцінити цілу послідовність оціночних функцій.

Можна здійснювати пошук у сітці за параметрами всіх оціночних функцій у конвеєрі одночасно.

Конвеєри допомагають уникнути витоку статистичних даних із ваших тестових даних у навчену модель під час перехресної перевірки, гарантуючи, що ті самі вибірки використовуються для навчання конверторів і предикторів. Усі оціночні функції в конвеєрі, крім останньої, повинні бути конверторами (тобто повинні мати метод перетворення даних). Остання оціночна функція може бути будь-якого типу (конвертор, класифікатор тощо).

Модуль `sklearn.pipeline` реалізує утиліти для побудови складеної оціночної функції, як ланцюга перетворень і оцінок.

Щоб перевірити, як це працює, будемо використовувати LogisticRegression.

```
In [ ]: clf = Pipeline([('preproc', StandardScaler()), ('classifier', LogisticRegression())])
clf.fit(features_train, labels_train)
```

```

scores_train = clf.score(features_train, labels_train)
scores_test = clf.score(features_test, labels_test)
print('Training DataSet accuracy: {:.1%}'.format(scores_train), 'Test DataSet accu
plot_confusion_matrix(clf, features_test, labels_test)
plt.show()

```

Як бачимо, результати непогані. Матриця помилок показує, скільки помилкових прогнозів ми отримали. Це дозволяє перевірити інші класифікатори та порівняти результати. Перевіримо:

- Logistic Regression (Логістичну регресію)
- Nearest Neighbors (Метод найближчих сусідів)
- Linear SVM (Метод опорних векторів лінійної регресії)
- RBF SVM (Радіально-базисну функцію)
- Gaussian Process (Гауссівський процес)
- Decision Tree (Дерево рішень)
- Random Forest (Метод випадкових лісів)
- Multi-layer Perceptron classifier (Багатошаровий персептрон)
- Ada Boost (Адаптивний бустинг)
- Naive Bayes (Найвій класифікатор Байеса)
- Quadratic Discriminant Analysis (Квадратичний дискримінантний аналіз)

```

In [16]: names = ["Logistic Regression", "Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaus
          "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
          "Naive Bayes", "QDA"]

classifiers = [
    LogisticRegression(),
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]
scores_train = []
scores_test = []
for name, clf in zip(names, classifiers):
    print("Fitting:", name)
    clf = Pipeline([('preproc', StandardScaler()), ('classifier', clf)])
    clf.fit(features_train, labels_train)
    score_train = clf.score(features_train, labels_train)
    score_test = clf.score(features_test, labels_test)
    scores_train.append(score_train)
    scores_test.append(score_test)

```

```
Fitting: Logistic Regression
Fitting: Nearest Neighbors
Fitting: Linear SVM
Fitting: RBF SVM
Fitting: Gaussian Process
Fitting: Decision Tree
Fitting: Random Forest
Fitting: Neural Net
Fitting: AdaBoost
Fitting: Naive Bayes
Fitting: QDA
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/discriminant_
analysis.py:878: UserWarning: Variables are collinear
    warnings.warn("Variables are collinear")
```

Виведемо результати у таблицю.

```
In [17]: res = pd.DataFrame(index = names)
res['scores_train'] = scores_train
res['scores_test'] = scores_test
res.columns = ['Test', 'Train']
res.index.name = "Classifier accuracy"
pd.options.display.float_format = '{:, .2f}'.format
print(res)
```

	Test	Train
Classifier accuracy		
Logistic Regression	0.90	0.85
Nearest Neighbors	0.87	0.70
Linear SVM	0.79	0.71
RBF SVM	1.00	0.48
Gaussian Process	0.79	0.67
Decision Tree	0.90	0.62
Random Forest	0.85	0.53
Neural Net	0.92	0.83
AdaBoost	0.86	0.58
Naive Bayes	0.67	0.59
QDA	1.00	0.58

Як можна побачити, обчислення дуже швидкі логістична регресія та нейронна мережа показують найкращий результат для тестового набору даних.

Порівняємо результати на графіку.

```
In [18]: x = np.arange(len(names)) # the label locations
width = 0.35 # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, scores_train, width, label='Train')
rects2 = ax.bar(x + width/2, scores_test, width, label='Test')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy of classifiers')
ax.set_xticks(x)
```

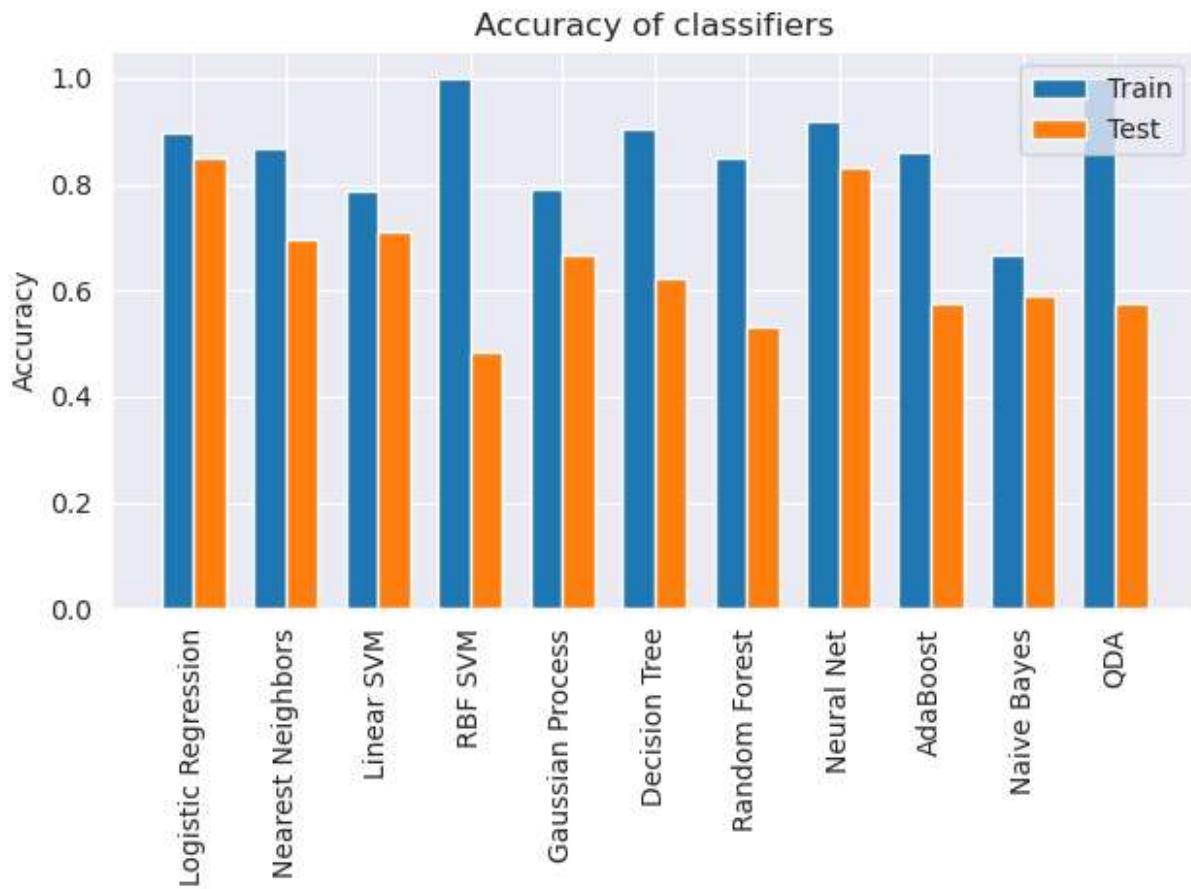
```

plt.xticks(rotation = 90)
ax.set_xticklabels(names)
ax.legend()

fig.tight_layout()

plt.show()

```



## Побудова та підгонка згорткової нейронної мережі

### Імпорт необхідних бібліотек

Для створення та навчання нашої моделі, будемо використовувати бібліотеку Keras.

In [19]:

```

import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
import tensorflow as tf
from sklearn.metrics import classification_report, confusion_matrix

```

```
Using TensorFlow backend.  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/fra  
mework/dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of ty  
pe is deprecated; in a future version of numpy, it will be understood as (type,  
(1,)) / '(1,)type'.  
    _np_qint8 = np.dtype([("qint8", np.int8, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/fra  
mework/dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of ty  
pe is deprecated; in a future version of numpy, it will be understood as (type,  
(1,)) / '(1,)type'.  
    _np_quint8 = np.dtype([("quint8", np.uint8, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/fra  
mework/dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of ty  
pe is deprecated; in a future version of numpy, it will be understood as (type,  
(1,)) / '(1,)type'.  
    _np_qint16 = np.dtype([("qint16", np.int16, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/fra  
mework/dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of ty  
pe is deprecated; in a future version of numpy, it will be understood as (type,  
(1,)) / '(1,)type'.  
    _np_quint16 = np.dtype([("quint16", np.uint16, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/fra  
mework/dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of ty  
pe is deprecated; in a future version of numpy, it will be understood as (type,  
(1,)) / '(1,)type'.  
    _np_qint32 = np.dtype([("qint32", np.int32, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorflow/python/fra  
mework/dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of ty  
pe is deprecated; in a future version of numpy, it will be understood as (type,  
(1,)) / '(1,)type'.  
    np_resource = np.dtype([("resource", np.ubyte, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/te  
nsorflow_stub/dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synony  
m of type is deprecated; in a future version of numpy, it will be understood as (typ  
e, (1,)) / '(1,)type'.  
    _np_qint8 = np.dtype([("qint8", np.int8, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/te  
nsorflow_stub/dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synony  
m of type is deprecated; in a future version of numpy, it will be understood as (typ  
e, (1,)) / '(1,)type'.  
    _np_quint8 = np.dtype([("quint8", np.uint8, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/te  
nsorflow_stub/dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synony  
m of type is deprecated; in a future version of numpy, it will be understood as (typ  
e, (1,)) / '(1,)type'.  
    _np_qint16 = np.dtype([("qint16", np.int16, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/te  
nsorflow_stub/dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synony  
m of type is deprecated; in a future version of numpy, it will be understood as (typ  
e, (1,)) / '(1,)type'.  
    _np_quint16 = np.dtype([("quint16", np.uint16, 1)])  
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/te  
nsorflow_stub/dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synony  
m of type is deprecated; in a future version of numpy, it will be understood as (typ  
e, (1,)) / '(1,)type'.  
    _np_qint32 = np.dtype([("qint32", np.int32, 1)])
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorboard/compat/te
nsorflow_stub/dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synony
m of type is deprecated; in a future version of numpy, it will be understood as (typ
e, (1,)) / '(1,)type'.
    np_resource = np.dtype([("resource", np.ubyte, 1)])
```

## Попередня обробка і збільшення даних

Згорткові нейронні мережі відрізняються тим, що ми можемо подавати зображення безпосередньо на вхід. Однак ці зображення також потребують попередньої обробки.

Зокрема, необхідно нормалізувати колір пікселів, змінивши діапазон їх значень від [0, 255] до [0, 1).

Також потрібно змінити розміри вхідних зображень за допомогою фреймворку Keras.

Класи зображень повинні мати не рядковий, а числовий тип.

Необхідні перетворення здійснює наведений нижче код.

```
In [20]: x_train = []
y_train = []
x_val = []
y_val = []

for feature, label in train:
    x_train.append(feature)
    y_train.append(label)

for feature, label in val:
    x_val.append(feature)
    y_val.append(label)

# Normalize the data
x_train = np.array(x_train) / 255
x_val = np.array(x_val) / 255

# Reshaping input images
x_train = x_train.reshape(-1, IMM_SIZE, IMM_SIZE, 1)
x_val = x_val.reshape(-1, IMM_SIZE, IMM_SIZE, 1)

# Creating a dictionary of classes
lab = {}
for i, l in enumerate(set(y_train)):
    lab[l] = i

y_train = np.array([lab[l] for l in y_train])
y_val = np.array([lab[l] for l in y_val])
```

```
In [21]: print("Shape of the input DataSet:", x_train.shape)
print("Shape of the output DataSet:", y_train.shape)
```

```
print("Dictionary of classes:", lab)

Shape of the input DataSet: (251, 224, 224, 1)
Shape of the output DataSet: (251,)
Dictionary of classes: {'Normal': 0, 'Viral Pneumonia': 1, 'Covid': 2}
```

## Збільшення даних на навчальній вибірці

Варто виконати збільшення даних, для кращого навчання моделі.

```
In [22]: datagen = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False, # apply ZCA whitening
    rotation_range = 30, # randomly rotate images in the range (degrees, 0 to
    zoom_range = 0.2, # Randomly zoom image
    width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
    horizontal_flip = True, # randomly flip images
    vertical_flip=False) # randomly flip images

datagen.fit(x_train)
```

## Визначення моделі

Визначимо просту модель ЗНМ з трьома згортковими шарами, за якими слідують шари максимального об'єднання. Щоб уникнути перенавчання, після 3-ї операції максимального об'єднання додається шар виключення.

```
In [23]: model = Sequential()
model.add(Conv2D(32,1,padding="same", activation="relu", input_shape=(IMM_SIZE, IMM_SIZE, 3))
model.add(MaxPool2D())

model.add(Conv2D(32, 1, padding="same", activation="relu"))
model.add(MaxPool2D())

model.add(Conv2D(64, 1, padding="same", activation="relu"))
model.add(MaxPool2D())
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(128,activation="relu"))
model.add(Dense(3, activation="softmax"))

model.summary()
```

WARNING:tensorflow:From /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/keras/backend/tensorflow\_backend.py:4070: The name `tf.nn.max_pool` is deprecated. Please use `tf.nn.max_pool2d` instead.

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 224, 224, 32)	64
<hr/>		
max_pooling2d_1 (MaxPooling2D)	(None, 112, 112, 32)	0
<hr/>		
conv2d_2 (Conv2D)	(None, 112, 112, 32)	1056
<hr/>		
max_pooling2d_2 (MaxPooling2D)	(None, 56, 56, 32)	0
<hr/>		
conv2d_3 (Conv2D)	(None, 56, 56, 64)	2112
<hr/>		
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 64)	0
<hr/>		
dropout_1 (Dropout)	(None, 28, 28, 64)	0
<hr/>		
flatten_1 (Flatten)	(None, 50176)	0
<hr/>		
dense_1 (Dense)	(None, 128)	6422656
<hr/>		
dense_2 (Dense)	(None, 3)	387
<hr/>		
Total params: 6,426,275		
Trainable params: 6,426,275		
Non-trainable params: 0		

Зкомпілюємо модель, використовуючи Adam як наш оптимізатор і SparseCategoricalCrossentropy як функцію втрат. Також, для більш плавної кривої, використаємо нижчу швидкість навчання - 0,000001.

```
In [24]: opt = Adam(lr=0.000001)
model.compile(optimizer = opt , loss = tf.keras.losses.SparseCategoricalCrossentrop
```

Тепер потренуємо нашу модель для **2000** епох. Правда, процес підгонки відбувається дуже повільно. Тому ми зберегли підігнану модель у файл. Щоб заощадити час, завантажимо підігнану модель. За бажання, можна змінити налаштування параметра **fitting** на **True**, щоб оновити модель.

```
In [25]: fitting = False
fitting_save = False
epochs = 2000

import pickle

if fitting:
    history = model.fit(x_train,y_train,epochs = epochs , validation_data = (x_val,
        if fitting_save:
```

```

# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
with open('history.pickle', 'wb') as f:
    pickle.dump(history.history, f)
with open('lab.pickle', 'wb') as f:
    pickle.dump(lab, f)

# Load model
from keras.models import model_from_json
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
# Load weights into a new model
model.load_weights("model.h5")
with open('history.pickle', 'rb') as f:
    history = pickle.load(f)
print("Loaded model from disk")

```

```

-----
FileNotFoundException                                     Traceback (most recent call last)
/tmp/ipykernel_68/2241309740.py in <module>
      21 # load model
      22 from keras.models import model_from_json
---> 23 json_file = open('model.json', 'r')
      24 loaded_model_json = json_file.read()
      25 json_file.close()

FileNotFoundException: [Errno 2] No such file or directory: 'model.json'

```

## Оцінка результатів

Побудуємо графік точності навчання та перевірки разом із втратами під час навчання та перевірки.

```

In [ ]: acc = history['accuracy']
val_acc = history['val_accuracy']
loss = history['loss']
val_loss = history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)

```

```
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Подивимося на вигляд кривої. Можна побачити, що точність наборів навчання та перевірки однакова. Функція втрат валідаційних і навчальних наборів є стабільною. Це означає, що наша ЗНМ добре підігнана і може використовуватися для класифікації.

Ми можемо отримати звіт про класифікацію, щоб побачити точність і її оцінку за допомогою `model.predict_classes()` та `classification_report()`.

Також ми можемо створити матрицю помилок. На жаль, фреймворк Keras не має функції `plot_confusion_matrix()`. Тому ми повинні створити його за допомогою Pandas і `Seaborn.heatmap()`

```
In [ ]: # Classification report
predictions = model.predict_classes(x_val)
predictions = predictions.reshape(1,-1)[0]
print(classification_report(y_val, predictions, target_names = lab.keys()))

# Confusion matrix
cm = pd.DataFrame(confusion_matrix(y_val, predictions))
cm.index = ["Predicted " + s for s in lab.keys()]
cm.columns = ["True " + s for s in lab.keys()]
print(cm)

sns.heatmap(confusion_matrix(y_val, predictions), annot=True,
            xticklabels = list(lab.keys()), yticklabels = list(lab.keys()))
plt.xlabel("True labels")
plt.ylabel("Predicted labels")
plt.show()
```

```
In [ ]: # Accuracy
z = model.predict_classes(x_train) == y_train
scores_train = sum(z+0)/len(z)
z = model.predict_classes(x_val) == y_val
scores_test = sum(z+0)/len(z)
print('Training DataSet accuracy: {:.1%}'.format(scores_train), 'Test DataSet accu
```

Як бачимо, ЗНМ показує кращі результати, ніж класичні моделі. Однак підгонка займає набагато більше часу.

А тепер спробуйте самостійно створити функцію, яка буде встановлювати діагноз на основі CNN. Замініть `##YOUR CODE GOES HERE##` на свій код.

```
In [30]: import mahotas as mh
import matplotlib.pyplot as plt
import numpy as np
import pickle
from keras.models import model_from_json
```

```

IMM_SIZE = 128 # Assuming IMM_SIZE is predefined

def diagnosis(file):
    try:
        image = mh.imread(file)
    except:
        print("Cannot download image: ", file)
        return

    if len(image.shape) > 2:
        image = mh.resize_to(image, [IMM_SIZE, IMM_SIZE, image.shape[2]]) # resize
    else:
        image = mh.resize_to(image, [IMM_SIZE, IMM_SIZE]) # resize grayscale image

    if len(image.shape) > 2:
        image = mh.colors.rgb2grey(image[:, :, :3], dtype=np.uint8) # change color

    plt.gray()
    plt.imshow(image)
    plt.show()

    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    model = model_from_json(loaded_model_json)
    # Load weights into the new model
    model.load_weights("model.h5")

    with open('history.pickle', 'rb') as f:
        history = pickle.load(f)
    with open('lab.pickle', 'rb') as f:
        lab = pickle.load(f)

    image = np.array(image) / 255
    image = image.reshape(-1, IMM_SIZE, IMM_SIZE, 1)

    diag = model.predict_classes(image)
    diag = list(lab.keys())[list(lab.values()).index(diag[0])]

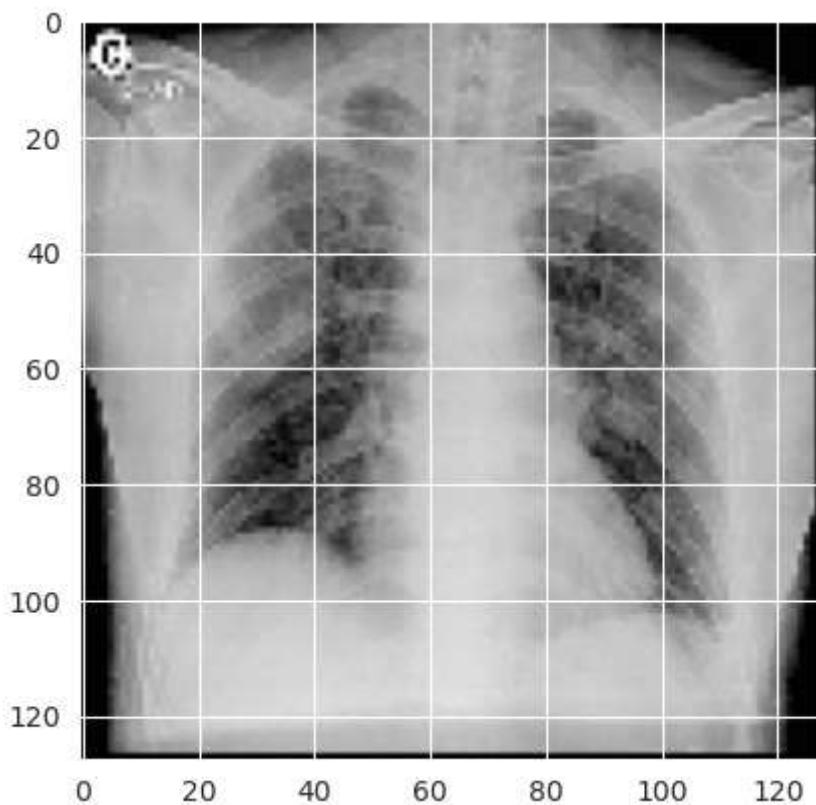
    return diag

```

- ▶ Натисніть **тут**, щоб отримати розв'язок для **Завантаження зображення**
- ▶ Натисніть **тут**, щоб отримати розв'язок для **Підготовка зображення до класифікації**
- ▶ Натисніть **тут**, щоб отримати розв'язок для **Демонстрація зображення**
- ▶ Натисніть **тут**, щоб отримати розв'язок для **Завантаження моделі**
- ▶ Натисніть **тут**, щоб отримати розв'язок для **Нармалізація даних**
- ▶ Натисніть **тут**, щоб отримати розв'язок для **Зміна розміру вхідних зображень**

- Натисніть **тут**, щоб отримати розв'язок для **Прогноз діагнозу**
- Натисніть **тут**, щоб отримати розв'язок для **Отримання назви діагнозу**

```
In [31]: print ("Diagnosis is:", diagnosis("Covid19-dataset/test/Covid/0120.jpg"))
print ("Diagnosis is:", diagnosis("Covid19-dataset/test/Normal/0105.jpeg"))
print ("Diagnosis is:", diagnosis("Covid19-dataset/test/Viral Pneumonia/0111.jpeg"))
```



```
-----
FileNotFoundException                                     Traceback (most recent call last)
/tmp/ipykernel_68/2007669158.py in <module>
----> 1 print ("Diagnosis is:", diagnosis("Covid19-dataset/test/Covid/0120.jpg"))
      2 print ("Diagnosis is:", diagnosis("Covid19-dataset/test/Normal/0105.jpeg"))
      3 print ("Diagnosis is:", diagnosis("Covid19-dataset/test/Viral Pneumonia/0111.jpeg"))

/tmp/ipykernel_68/3400989620.py in diagnosis(file)
    26     plt.show()
    27
--> 28     json_file = open('model.json', 'r')
    29     loaded_model_json = json_file.read()
    30     json_file.close()

FileNotFoundException: [Errno 2] No such file or directory: 'model.json'
```

## Висновки

У даній лабораторній роботі розглянуто, як створити експертну систему, що дає змогу отримувати діагноз на основі рентгенівських зображень, використовуючи різні класифікатори. Ці принципи можна використовувати для будь-якого типу рентгенівських зображень, а не лише для діагностики COVID.

Під час цієї лабораторної роботи реалізовано завантаження та обробка зображень. Ми навчилися отримувати ознаки зображень і створювати/підходити/тестувати/порівнювати набори класифікаторів. Також ми дізналися, як створювати та підлаштовувати згорткові нейронні мережі. Реалізовано порівняння точності різних класифікаторів.

## Автор

[Yaroslav Vyklyuk, prof., PhD., DrSc](#)

Copyright © 2020 IBM Corporation. This notebook and its source code are released under the terms of the [MIT License](#).