



Machine Learning Engineer Nanodegree

Capstone Project

Convolutional Neural Network Model for Predicting COVID-19 Presence in Chest X-Rays

Eugene Medina

eu63n3m@outlook.com

September 2020

1 Definition

1.1 Project Overview

Machine learning as a subset of Artificial Intelligence has proven its usefulness in the medical field, and as such, gained traction over the years. Although not yet fully mature, it is definitely making healthcare smarter. Machine learning techniques are frequently used for medical applications using feature extraction and image analysis. It is used in the diagnosis of viral pneumonia using X-rays and CT scans, tumor diagnosis, and cystoscopic analysis. Viral pathogenic patterns contain several features in X-ray images. COVID-19 shows an irregular distribution and shading within the image. In this project, the objective is to show that it is possible to utilize machine learning, particularly in image recognition and classification, to determine if a given chest X-ray is indicative of the presence of COVID-19.

When the COVID-19 pandemic hit early this year, scientists, and doctors all over the world were given the mandate to put an end to this disease. As more people became infected, public data became more available and accessible to the doctors, scientists, and even researchers from the scientific and engineering communities. Medical institutions, hospitals, and research agencies all over the world recognized the importance of chest X-rays and their role in the diagnosis and the treatment planning for patients with suspected or confirmed COVID-19 infections. It is vital that physicians receive clear information from their radiologists to assist in diagnosing, treating, and managing their patients. Chest X-ray (CXR) interpretations have been very manual in the past, with specialist physicians having to look at X-ray pictures of patients as they are presented. Doctors have used the ABCDE approach to carry out structured interpretation of a chest X-ray: **A**irway, **B**reathing, **C**ardiac, **D**iaphragm, and **E**verything else. Although the procedure has been deemed accurate by the medical community, resources have recently been overwhelmed when the COVID-19 pandemic struck. Doctors need tools to accurately, and with equally important mandate, quickly diagnose patients who are suspected to have COVID-19 to provide appropriate healthcare and treatment. Test kits have been in limited supply and new studies suggest that chest X-rays and chest CT scans can help diagnose the disease.

Machine learning has been proven to have life-impacting potential in healthcare – particularly in the field of medical diagnosis. Please note that this is just a proof of concept and by no means considered as a diagnostic tool but I think that the basic foundation built is that machine learning in healthcare can have a significant impact on the battle versus COVID-19. It is my hope that this project will eventually provide some benefit to the doctors, scientists, and health professionals in the global fight against COVID-19.

The primary dataset I will be working on is found [here](#) [1]. This is a database of chest X-ray images for COVID-19-positive cases along with normal and viral pneumonia images. This is the work of a team of researchers from Qatar University in Doha, Qatar and the University of Dhaka, Bangladesh along with their collaborators from Pakistan and

Malaysia in cooperation with medical doctors. There are 219 COVID-19-positive images, 1,341 normal images and 1,345 viral pneumonia images.

I also plan to use the Mila COVID -19 image dataset found [here](#) [2] to augment the above database. Patients who have been confirmed with COVID -19 through polymerase chain reaction (PCR) test with pneumonia may present X-ray images with a pattern that is only moderately characteristic for the human eye.

Another useful dataset can also be found [here](#) [3]. It is intended to be used for research purposes only but will also use it to augment our primary dataset.

1.2 Problem Statement

Despite the success and promising efforts in applying machine learning in the diagnosis of COVID-19 through CT scans and other means of imaging detection, I believe the fact remains that COVID -19 is an infection of a global scale and is likely to be experienced in communities of varying sizes, including those that are considered remote. Since X-rays are more accessible in these communities because it remains inexpensive and quick to perform, they are generally more accessible to healthcare providers in most regions. This is not intended to replace existing means of testing for COVID-19 such as molecular (polymerase chain reaction or PCR test) and serological (testing for antibodies in blood and tissues) tests.

This project aims to leverage the use of a supervised machine learning technique, particularly classification, where the output will have a defined label. In this project, we will be utilizing a well-known Convolutional Neural Network – PyTorch, and one of the computer vision models, Resnet18. This framework is particularly useful in computer vision, and with this project, we will train a model to enable us to classify X-ray images and determine if a particular X-ray has COVID-19 in it, or does it belong to two other classes, normal and viral pneumonia. Originally, I was going for a simple binary classification but since the primary dataset already contains classes for COVID-19, normal, and viral pneumonia images, I thought of just going with a multi-class classification – since the classes are distinct from each other.

1.3 Metrics

I will be implementing the evaluation metric classification accuracy in this project. Simply stated, accuracy is the number of correct predictions divided by the total number of predictions:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

In addition to accuracy, we will also indicate the validation loss and the training loss. Loss is often used in the training process to find the best parameter values in the model.

2 Analysis

2.1 Data Exploration and Visualization

In this project, we will be using the dataset found in this [link](#) [1]. We will be downloading and using the file “576013_1042828_bundle_archive.zip”. This is our primary dataset. This dataset has 2,905 chest X-ray scans: 1,341 normal, 1,345 viral pneumonia and 219 COVID-19. The images are distributed in 3 folders and labeled in 3 classes namely COVID-19, NORMAL, and Viral Pneumonia. By default, these images are 1024 x 1024 in size and are in .PNG file format. This dataset is the work of a team of researchers from Qatar University in Doha, Qatar and the University of Dhaka, Bangladesh along medical professionals from Pakistan and Malaysia. In this dataset, we are particularly interested in the features “filename” and “format”.

2.1.1 Primary Dataset - COVID-19 Metadata

The COVID-19 folder contains 219 images. Compared against the two other classes, NORMAL and Viral Pneumonia, the COVID-19 folder contains a notably lesser number of files and thus we will be presented with a class imbalance. To mitigate this problem, I am going to add more COVID-19 images from two other sources and will be discussed later. A snapshot of the COVID-19 metadata is seen below.

	A	B	C	D
1	FILE NAME	FORMAT	SIZE	URL
2	COVID-19(1)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S0140673620303706
3	COVID-19(2)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S0929664620300449
4	COVID-19(3)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S0929664620300449
5	COVID-19(4)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S0929664620300449
6	COVID-19(5)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S0929664620300449
7	COVID-19(6)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S1684118220300608
8	COVID-19(7)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S1684118220300608
9	COVID-19(8)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S1684118220300682
10	COVID-19(9)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S1684118220300682
11	COVID-19(10)	PNG	1024*1024	https://www.sciencedirect.com/science/article/pii/S1684118220300682

Figure 1: Snapshot of COVID-19 metadata spreadsheet

Below are some of the images as indicated in the above snapshot. I like the file-naming convention and the same convention will be used when adding in images from other source datasets.

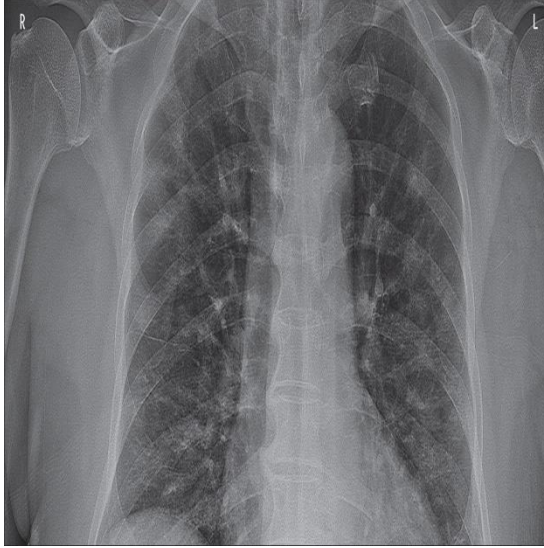


Figure 2: COVID-19(1).png

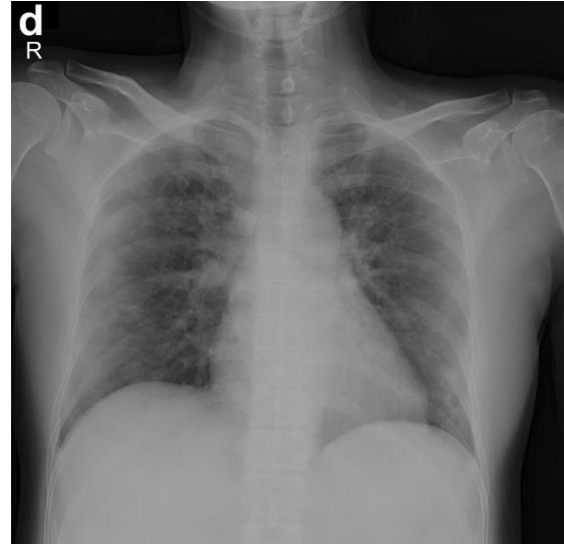


Figure 3: COVID-19(5).png

2.1.2 Primary Dataset - NORMAL Metadata

The NORMAL folder contains 1,341 images. Given the scope of this project, I think that the number of files for this class is sufficient so we will not be adding any more normal images from other datasets. A snapshot of the normal metadata can be seen below, followed by samples of the image files.

	A	B	C	D
1	FILE NAME	FORMAT	SIZE	URL
2	NORMAL-1	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
3	NORMAL-2	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
4	NORMAL-3	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
5	NORMAL-4	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
6	NORMAL-5	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
7	NORMAL-6	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
8	NORMAL-7	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
9	NORMAL-8	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
10	NORMAL-9	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
11	NORMAL-10	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Figure 3: Snapshot of NORMAL metadata spreadsheet



Figure 4: NORMAL (4).png

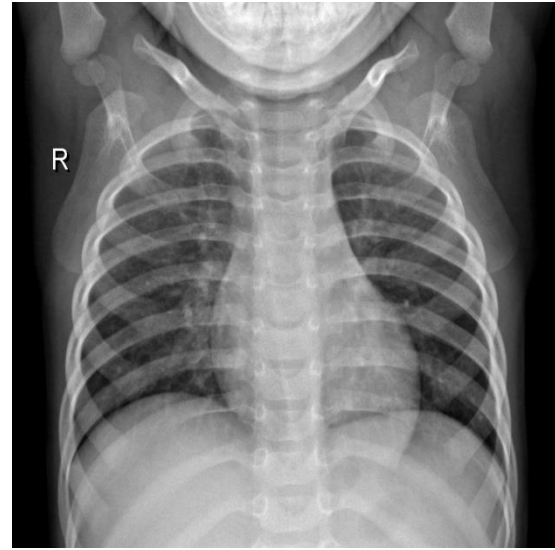


Figure 5: NORMAL (8).png

2.1.3 Primary Dataset - Viral Pneumonia Metadata

The Viral Pneumonia folder contains 1,345 images. Again, I think that the number of files for this class is sufficient so we will not be adding any more viral pneumonia images from other source datasets. A snapshot of the viral pneumonia metadata can be seen below, followed by samples of the image files.

	A	B	C	D
1	FILE NAME	FORMAT	SIZE	URL
2	Viral Pneumonia-1	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
3	Viral Pneumonia-2	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
4	Viral Pneumonia-3	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
5	Viral Pneumonia-4	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
6	Viral Pneumonia-5	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
7	Viral Pneumonia-6	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
8	Viral Pneumonia-7	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
9	Viral Pneumonia-8	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
10	Viral Pneumonia-9	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
11	Viral Pneumonia-10	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
12	Viral Pneumonia-11	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
13	Viral Pneumonia-12	PNG	1024*1024	https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Figure 6: Snapshot of Viral Pneumonia metadata spreadsheet



Figure 7: Viral Pneumonia (3).png

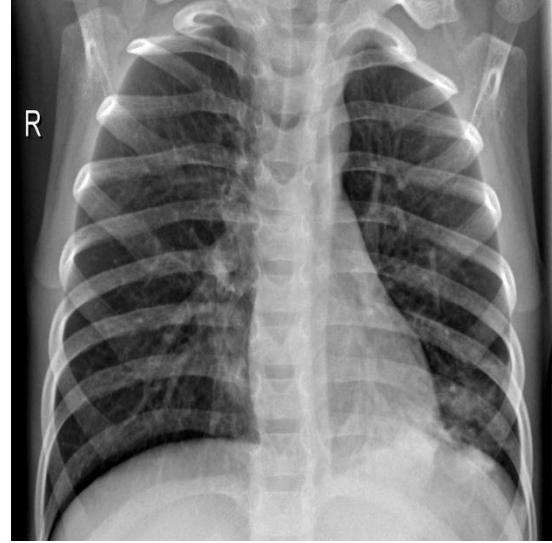


Figure 8: Viral Pneumonia (9).png

2.1.4 Secondary Dataset - MILA Metadata

Since the number of COVID-19-positive images are small compared to the other two classes, I have added another dataset to help mitigate class imbalance. In the end there will still be fewer images for COVID-19 compared to either the NORMAL or Viral Pneumonia classes but the number of COVID-19 images from where we can extract the test and train dataset significantly increased. The secondary dataset can be found [here](#) [2]. For this source dataset, we will be downloading and using the file "covid-chestxray-dataset-master.zip". This dataset is a result of the project of John Paul Cohen from the University of Montreal. The project is a collection of images to build a public open dataset of chest X-ray and CT images of patients which are positive or suspected of COVID-19 or other viral and bacterial pneumonias. Data is collected from public sources as well as through indirect collection from hospitals and physicians. There are 884 images in this dataset where COVID-19 images were properly identified along with other findings such as SARS, Pneumocystis, Klebsiella, Bacterial Pneumonia, and others. Images that are identified as COVID-19 will be added into our COVID-19 folder for processing. Please note that only the images identified as positive for COVID-19 will be added to the primary dataset described above. A snapshot of the metadata spreadsheet is shown below. Features extracted for this dataset are "finding" (only COVID-19 are considered), "view" (PA or AP), and "filename".

	A	B	C	D	E	F	G	H	I	J	K	L	S	T	U	V	W	X	
1	patientid	offset	sex	age	finding	RT_PCR_positive	survival	intubated	intubation	went_icu	in_icu	needed_su	view	modality	date	location	folder	filename	doi
2	2	0	M	65	COVID-19	Y	Y	N	N	N	N	Y	PA	X-ray	22-Jan-20	Cho Ray Hospital, Hc	images	auntminnie-	10.
3	2	3	M	65	COVID-19	Y	Y	N	N	N	N	Y	PA	X-ray	25-Jan-20	Cho Ray Hospital, Hc	images	auntminnie-	10.
4	2	5	M	65	COVID-19	Y	Y	N	N	N	N	Y	PA	X-ray	27-Jan-20	Cho Ray Hospital, Hc	images	auntminnie-	10.
5	2	6	M	65	COVID-19	Y	Y	N	N	N	N	Y	PA	X-ray	28-Jan-20	Cho Ray Hospital, Hc	images	auntminnie-	10.
6	4	0	F	52	COVID-19	Y		N	N	N	N	N	PA	X-ray	25-Jan-20	Changhua Christian H	images	nejmc20015	10.
7	4	5	F	52	COVID-19	Y		N	N	N	N	N	PA	X-ray	30-Jan-20	Changhua Christian H	images	nejmc20015	10.
8	5				ARDS			Y	Y	Y	Y		PA	X-ray	2017		images	ARDSSevere.png	
9	6	0			COVID-19	Y		Y	Y	Y	Y		PA	X-ray	06-Jan-20	Wuhan Jinyintan Hos	images	lancet-case2	10.
10	6	4			COVID-19	Y		Y	Y	Y	Y		PA	X-ray	10-Jan-20	Wuhan Jinyintan Hos	images	lancet-case2	10.
11	3	4	M	74	SARS		N						AP	X-ray	2004	Mount Sinai Hospital	images	SARS-10.114	10.
12	3	9	M	74	SARS		N						AP	X-ray	2004	Mount Sinai Hospital	images	SARS-10.114	10.
13	3	10	M	74	SARS		N						AP	X-ray	2004	Mount Sinai Hospital	images	SARS-10.114	10.
14	7	7	F	29	SARS		Y						PA	X-ray	2004	Mount Sinai Hospital	images	SARS-10.114	10.

Figure 9: Snapshot of the MILA COVID-19 metadata spreadsheet

The images below are some of the sample files found in the dataset.

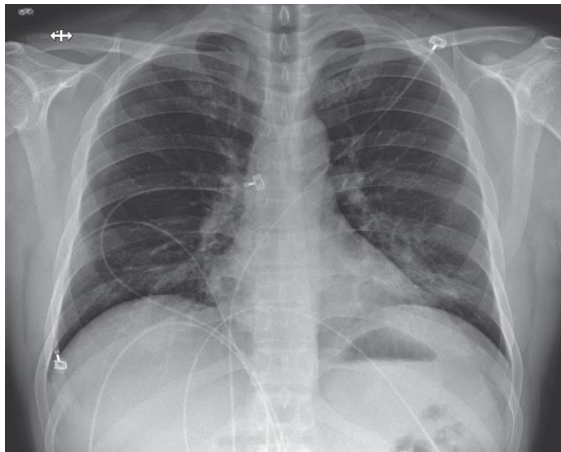


Figure 10: Identified as COVID-19 - nejmoa2001191_f4.jpeg



Figure 11: Identified as SARS - wong-0002.jpg

2.1.5 Tertiary Dataset – Figure-1 Metadata

We will download [this](#) [3] dataset and use the file “Figure1-COVID-chestxray-dataset-master.zip”. This dataset is the progressive work of the Core Covid-Net Team mainly composed of the following: DarwinAI Corp., Canada, Vision and Image Processing Research Group, University of Waterloo, Canada and many others. They have built this dataset as a part of the COVIDx dataset to enhance the models for COVID-19 detection (COVID-Net) and COVID-19 risk stratification (COVID-RiskNet). There are 55 images in this dataset and all of them identified as COVID-19-positive. These images will be added to our primary dataset described above. A snapshot of the metadata can be seen below, followed by samples of the image files. Features extracted for this dataset are “finding” (only COVID-19 are considered), “view” (PA or AP), and “patientid” which will be treated as the filename of the image.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	patientid	offset	sex	age	finding	survival	temperatu	pO2 satur	view	modality	artifacts/d	notes					
2	COVID-00001	13	M	33	COVID-19			58	AP erect	X-ray		O2 saturation was initially 58% on room air and 89% on 15 litres of O					
3	COVID-00002		M	50+					AP erect	X-ray		50+ male patient, asthmatic, returning from a COVID-19-affected Euro					
4	COVID-00003a	8	M	28			39.1	90	AP erect	X-ray		28M previously fit and well, not on any regular medications, presented					
5	COVID-00003b	8	M	28			39.1	90	AP erect	X-ray		Post-intubation in the critical care setting					
6	COVID-00004	2	M	42	COVID-19			91-92	PA	X-ray		42 year old male patient presented to ED with two day history of fever					
7	COVID-00005									X-ray		Slight from N/A					
8	COVID-00006	5	F	57	COVID-19					X-ray		Some from 57 year old female. Returning from Texas March 13th, unremarkable P					
9	COVID-00007			73			38+			X-ray		Some from 73 yo. No antecedents. Fever 38° + cough. Severe respiratory insufficie					
10	COVID-00008	4	M	47	COVID-19				92	AP erect	X-ray	Some from 47 Male, BMI 34, sleep apnea and childhood asthma, return from trav					
11	COVID-00009				COVID-19					X-ray		Some from CXray during worsening corona infection					
12	COVID-00010	5	M	50+					AP erect	X-ray		Male in his 50s. Presenting to ER 5 days after developing a fever and cc					

Figure 12: Snapshot of the FIG1 COVID-19 metadata spreadsheet



Figure 13: COVID-00042.jpg

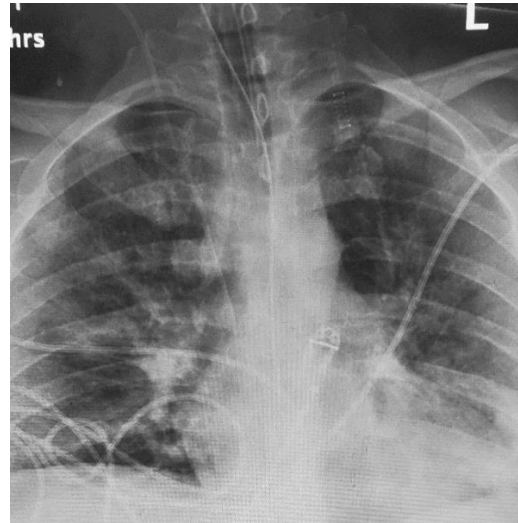


Figure 14: COVID-00008.jpg

2.2 Algorithms and Techniques

The proposed solution to this given problem is to apply deep learning techniques that have been proven to be extraordinarily successful in image classification. The solution is similar to the open-source machine learning model, Covid-cxr – the result of the work of the Artificial Intelligence Research and Innovation Labs at the city of London, Ontario, Canada – which successfully predicts the COVID-19 presence from chest X-rays. An article describing this model can be found [here](#) [4].

We will use the primary dataset described above as a starting point to gather data. Two additional datasets are also used to add to our COVID-19 images for test and training. There are three classes that will be defined – normal, viral pneumonia, and COVID-19. The number of COVID-19-positive images are small compared to the other two classes. To help mitigate class imbalance, the metadata of the other two datasets are read; zip files extracted to a temporary directory. All COVID-19-positive images that were identified in the metadata are copied to the directory of the COVID-19-positive images generated by the primary dataset, renaming them to .PNG format for uniformity.

There is no training or test set pre-defined separately, so we are going to create them in this project. We will be using a training set and test set for validation. Usually, we have a separate validation set as well, and we will reserve the test set for testing but for simplicity we are going to use just the two sets. In the code, we will randomly select 50 images for each class of normal, viral pneumonia, and COVID-19 for our test set.

Next, we will transform the images picked up to the required size of 224 x 224 pixels. We will also process the filename to be consistent throughout. The selected file format is that of Portable Network Graphics or .PNG. To be consistent, any image file that is native .jpeg or .jpg will be converted to .PNG to be consistent. We will then feed the pre-processed data to a Residual Neural Network ResNet18 using PyTorch. ResNet18 is a convolutional neural network pre-trained on more than one million images from the [ImageNet](#) [5] database. This database is a dataset of millions of high-resolution images, labelled by humans, and belong roughly to 22,000 categories. There are 18 layers present in its architecture. The core idea of ResNet is the so-called *identity shortcut connection* that skips one or more layers, making the gradient update for those layers much easier. This makes the model one of the most popular CNN architectures, which provides for more efficient training.

ResNet18 is very efficient in image classification and can classify images into 1000 classes or object categories. Input image size required is 224 x 224 pixels. I have chosen ResNet18 because networks with large numbers of layers can be trained easily without increasing the training error percentage. This is more than sufficient as we are only identifying images and classifying them on three categories.

2.3 Benchmark

I have chosen Covid-cxr as a good reference model to be used as benchmark for the proposed project. This existing model allows both binary and multi-class classification. It has also provided results that are remarkably accurate. Although this work is not intended to be used clinically by health professionals, its use is purely exploratory. The classifier model's performance on the test set had an AUC of 0.9633, a sensitivity (recall) of 0.875, and accuracy of 0.92. Conclusions and overall performance of the proposed project will be drawn from the results and compared against the existing model. Here is a table of the metrics of the Covid-cxr model and its corresponding values:

Metric	Value
loss	3.206988981791905
accuracy	0.92
precision	0.5
recall	0.875
auc	0.9633001
f1score	0.6363636

Figure 15: Covid-cxr sample metrics

As mentioned earlier, I will be implementing the evaluation metric classification accuracy in this project, as well as validation loss, and training loss.

3 Methodology

3.1 Data Preprocessing and Augmentation

3.1.1 Importing the Required Libraries

PyTorch ResNet18 is the model proposed to be used for the solution. The library will thus be imported along with other required libraries indicated by the code below:

```

%matplotlib inline

import os
import sys
import datetime
import shutil
import random
import torch
import torchvision
import numpy as np
import pandas as pd

from PIL import Image
from matplotlib import pyplot as plt
from zipfile import ZipFile

torch.manual_seed(0)
dt = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

print('Capstone Project - Determining Presence of COVID-19 Through Chest X-rays')
print('Using PyTorch version', torch.__version__)
print('by Eugene Medina')
print('September 2020')
print('=' * 80)

```

I have also included a code for displaying the time and date of the execution. This may not be necessary in production code, but I thought of using it here to keep track of the code execution.

3.1.2 Preparing the Source Directory and Downloading the Files

Before executing the code, a required directory must first be created. This directory must be created in the same directory as the source code and must be named “Source CXR Database”. Download the primary dataset from this link: <https://www.kaggle.com/tawsifurrahman/covid19-radiography-database/download> and save the file to the “Source CXR Database” directory. Next, download the file from this link: <https://github.com/ieee8023/covid-chestxray-dataset/archive/master.zip> then save the file to the same directory above. Lastly, download the file from the following link: <https://github.com/agchung/Figure1-COVID-chestxray-dataset/archive/master.zip> and again, save the file to the directory indicated above. Do not extract the zip files as they will be extracted during preprocessing. The following section of the code sets the names of the required dataset directory along with the names of the required files, “testset” directory for containing our test set images, “trainset” directory for containing our training set images, class names “normal”, “viral”, and “covid” and their associated source directories, as well as the working directory “COVID-19 Radiography Database”. For this project I have set the number of random image sample to 50 for

each of the classes for simplicity and can be arbitrarily increased or decreased. Since there are only 219 images for COVID-19-positive in our primary dataset, I have set the “idx” value to 219 so that any files added to this directory will continue to increment starting at “idx” plus 1. The indicator “skipthis = ‘false’” is just to indicate if the source dataset directory already exists and all the files required are present to avoid re-extracting the files if they have already been extracted previously.

```
%%time

# set the names of the required directories, files and class names for our dataset
required_dir = 'Source CXR Database'
required_file1 = '576013_1042828_bundle_archive.zip'
required_file2 = 'covid-chestxray-dataset-master.zip'
required_file3 = 'Figure1-COVID-chestxray-dataset-master.zip'
class_names = ['normal', 'viral', 'covid']
work_dir = 'COVID-19 Radiography Database'
testset_dir = 'testset'
trainset_dir = 'trainset'
source_dirs = ['NORMAL', 'Viral Pneumonia', 'COVID-19']

# set the number of samples for each class for test dataset
#                               test dataset
# normal                        50 images
# viral pneumonia              50 images
# covid-19                     50 images
num_random_samples = 50

# there are 219 COVID-19-positive images in our primary dataset
# idx will be used to organize and continue the file-naming convention used when adding
# more COVID-19-positive images to our primary dataset from the other two datasets
idx = 219
skipthis = 'false'
```

3.1.3 Primary Dataset Feature Extraction

When the code is initially run, it will create a working directory “COVID-19 Radiography Database” when preprocessing the primary dataset. The contents of this directory will be our starting point and will not do any feature engineering as all the necessary files are already arranged according to their classes – NORMAL, Viral Pneumonia and COVID-19. It is important to note, however, that the features we are interested in are the “filename” and “format”. The file-naming convention under the directory COVID-19 will be followed and adhered to when adding and augmenting the list of files in this directory. There are 219 images in this directory, the last being “COVID-19(219).png”. Additional image files copied into this directory will start its filename with “COVID-19(220).png”. The following code shows the process.

```

# Check if COVID-19 Radiography Database/COVID-19 and its files already exist, if so skip extracting the file.
if os.path.isfile(os.path.join(work_dir, source_dirs[2], 'COVID-19(681).png')):
    print(str(dt), ': Required dataset already exists.')
    skipthis = 'true'

if skipthis == 'false':
    # Create a ZipFile object for the first required file as read only then
    # extract all the contents of zip file in current directory;
    # work_dir will be our working directory
    with ZipFile(os.path.join(required_dir, required_file1), 'r') as zipObj:
        print(str(dt), ': Extracting first required dataset, please wait...')
        zipObj.extractall()

```

3.1.4 Secondary Dataset Feature Extraction

The secondary dataset used here is the MILA dataset, from the University of Montreal, Canada. The zip file is extracted to a temporary directory. Files that are not relevant are then deleted and the directory “images” and the file “metadata.csv” are retained, as shown below. The file “metadata.csv” is then parsed and a dataframe constructed with all the image files that have the features “view” as either “PA” or “AP”, and “finding” equals to “COVID-19”. The code then iterates through the dataframe, converting the image files to .PNG format and renaming them with the convention used in our primary dataset. The files are then added to the directory “COVID-19” created above when processing the primary dataset. Here is the code.


```

# Create a ZipFile object for the second required file as read only then
# extract all contents of zip file in temp_2
with ZipFile(os.path.join(required_dir, required_file2), 'r') as zipObj:
    print(str(dt), ': Extracting second required dataset, please wait...')
    zipObj.extractall('temp_2')

# Retain only the necessary files, discard the rest
shutil.move(os.path.join('temp_2', 'covid-chestxray-dataset-master', 'images'), 'temp_2')
shutil.move(os.path.join('temp_2', 'covid-chestxray-dataset-master', 'metadata.csv'), 'temp_2')
shutil.rmtree(os.path.join('temp_2', 'covid-chestxray-dataset-master'))

# Read metadata.csv from MILA data and construct dataframe
mila_df = pd.read_csv(os.path.join('temp_2', 'metadata.csv'))
mila_df['filename'] = mila_df['filename'].astype(str)
# Select desired X-ray views, PA or AP
mila_views_cxr_df = (mila_df['view'].str.contains('|'.join(['PA', 'AP'])))
mila_covid_fdg_df = (mila_df['finding'] == 'COVID-19')
# Images for patients diagnosed with COVID-19
mila_covid_views_df = mila_df[mila_covid_fdg_df & mila_views_cxr_df]
# Convert the filenames to pandas dataframe
mila_covid_fn_df = pd.DataFrame(mila_covid_views_df)
print(str(dt), ': There are ' + str(len(mila_covid_fn_df)) + ' additional images for COVID-19 to be saved.')
print(str(dt), ': Saving additional files. This will take quite a while. Time for coffee...')

# Iterate through the list of covid-validated filenames,
for index, row in mila_covid_fn_df.iterrows():
    idx = idx + 1
    new_fn = 'COVID-19(' + str(idx) + ').png'
    if row['filename'].find('.jpeg') or row['filename'].find('.jpg') or row['filename'].find('.png'):
        # Save file using new name and as png
        img = Image.open(os.path.join('temp_2', 'images', row['filename']))
        img.save(os.path.join(work_dir, source_dirs[2], new_fn))
    print('#', end = '')
print('')

```

3.1.5 Tertiary Dataset Feature Extraction

The tertiary dataset used here is the result of the work of the Core COVID-Net team, Figure 1 COVID chest X-ray dataset. Similar to the secondary dataset, the zip file is extracted to a temporary directory, files that are not relevant deleted and the directory “images” and the file “metadata.csv” are retained. The code snippet is shown below.

```

# Retain only the necessary files, discard the rest
shutil.move(os.path.join('temp_3', 'Figure1-COVID-chestxray-dataset-master', 'images'), 'temp_3')
shutil.move(os.path.join('temp_3', 'Figure1-COVID-chestxray-dataset-master', 'metadata.csv'), 'temp_3')
shutil.rmtree(os.path.join('temp_3', 'Figure1-COVID-chestxray-dataset-master'))

# Read metadata.csv from FIG1 data and construct dataframe
fig1_df = pd.read_csv(os.path.join('temp_3', 'metadata.csv'), encoding = 'ISO-8859-1')
fig1_df['filename'] = ''
for i, row in fig1_df.iterrows():
    if os.path.exists(os.path.join('temp_3', 'images', fig1_df.loc[i, 'patientid'] + '.jpg')):
        fig1_df.loc[i, 'filename'] = os.path.join('temp_3', 'images', fig1_df.loc[i, 'patientid'] + '.jpg')
    else:
        fig1_df.loc[i, 'filename'] = os.path.join('temp_3', 'images', fig1_df.loc[i, 'patientid'] + '.png')

# Select desired X-ray views, PA or AP
fig1_views_cxr_df = (fig1_df['view'].str.contains('|'.join(['PA', 'AP', 'AP erect'])))
fig1_covid_fdg_df = (fig1_df['finding'] == 'COVID-19')
# Images for patients diagnosed with COVID-19
fig1_covid_views_df = fig1_df[fig1_covid_fdg_df & fig1_views_cxr_df]
# Convert the filenames to pandas dataframe
fig1_covid_fn_df = pd.DataFrame(fig1_covid_views_df)
print(str(dt), ': There are ' + str(len(fig1_covid_fn_df)) + ' additional images for COVID-19 to be saved.')
print(str(dt), ': Saving additional files. Please wait...')

# Iterate through the list of covid-validated filenames,
for index, row in fig1_covid_fn_df.iterrows():
    idx = idx + 1
    new_fn = 'COVID-19(' + str(idx) + ').png'
    if row['filename'].find('.jpeg') or row['filename'].find('.jpg') or row['filename'].find('.png'):
        # Save file using new name and as png
        img = Image.open(row['filename'])
        img.save(os.path.join(work_dir, source_dirs[2], new_fn))
    print('#', end = '')
print('')
# Cleanup temporary directories
shutil.rmtree('temp_2')
shutil.rmtree('temp_3')

```

The file “metadata.csv” is then parsed and a dataframe constructed with all the image files that have the features “view” as either “PA”, “AP” or “AP erect”, and “finding” equals to “COVID-19”. The code then iterates through the dataframe, converting the image files to .PNG format and renaming them with the convention used in our primary dataset. The files are then added to the directory “COVID-19” created above when processing the primary dataset. Lastly, the temporary directories are deleted.

3.1.6 Splitting the Datasets

The preprocessed dataset is then split into a test set and a training set. Every time the code is run, previous test and train directories are removed to ensure that random images are picked every time. The code below performs that:

```

# skipthis == 'true' here
if os.path.isdir(os.path.join(work_dir, testset_dir)):
    # remove previous test directory to start anew
    shutil.rmtree(os.path.join(work_dir, testset_dir))
    print(str(dt), ': Removing previous test directory...')

if os.path.isdir(os.path.join(work_dir, trainset_dir)):
    # remove previous train directory to start anew
    shutil.rmtree(os.path.join(work_dir, trainset_dir))
    print(str(dt), ': Removing previous train directory...')

if os.path.isdir(os.path.join(work_dir, source_dirs[1])):
    os.mkdir(os.path.join(work_dir, testset_dir))
    print(str(dt), ': Creating new test directory...')
    os.mkdir(os.path.join(work_dir, trainset_dir))
    print(str(dt), ': Creating new train directory...')

```

After the test and train directories are created anew, files from our source dataset directories are copied to the “trainset” directory. This directory contains the training images composed of “COVID-19”, “NORMAL”, and “Viral Pneumonia” directories, as shown in the following code:

```

# Create directory for our training set:
# "trainset" directory contains files from our "COVID-19", "NORMAL",
# and "Viral Pneumonia" directories
for t, sd in enumerate(source_dirs):
    shutil.copytree(os.path.join(work_dir, sd), os.path.join(work_dir, trainset_dir, sd))
print(str(dt), ': Images for the trainset directories copied.')

```

Next, we will derive the images for our test dataset from our trainset directories. After the “testset” directory has been created, *num_random_samples* (by default it is 50) images from each class of trainset directories in “COVID-19”, “NORMAL”, and “Viral Pneumonia”, will be moved to its corresponding “testset” directories “covid”, “normal”, and “viral” respectively. The following code indicates this.

```

# Create directory for our test set: "testset"
for cn in class_names:
    os.mkdir(os.path.join(work_dir, testset_dir, cn))

# The "testset" directory will contain the following
# "covid" directory: num_random_samples images from training set "COVID-19"
# "normal" directory: num_random_samples images from training set "NORMAL"
# "viral" directory: num_random_samples images from training set "Viral Pneumonia"
# the num_random_samples images will be moved from the above trainset directories to
# the corresponding testset directories
for i, s in enumerate(source_dirs):
    images = [x for x in os.listdir(os.path.join(work_dir, trainset_dir, s)) if x.lower().endswith('png')]
    selected_images = random.sample(images, num_random_samples)
    for image in selected_images:
        source_path = os.path.join(work_dir, trainset_dir, s, image)
        target_path = os.path.join(work_dir, testset_dir, class_names[i], image)
        shutil.move(source_path, target_path)
        print('#', end = '')
print('')
print(str(dt), ': Images for the testset directories moved successfully.')

```

3.1.7 Example Output of the Data Preprocessing

Here is a snapshot of the example output of our data preprocessing and augmentation when the code has been run for the first time.

```

Current working directory is: G:\Documents\Learning\Udacity\capstone-project\cxr-covid
=====
2020-09-05 12:18:48 : Checking required source directory and files...
2020-09-05 12:18:48 : Source dataset working directory and files already exist.
2020-09-05 12:18:48 : Extracting first required dataset, please wait...
2020-09-05 12:18:48 : Extracting second required dataset, please wait...
2020-09-05 12:18:48 : There are 454 additional images for COVID-19 to be saved.
2020-09-05 12:18:48 : Saving additional files. This will take quite a while. Time for coffee...
#####
#####
#####
2020-09-05 12:18:48 : Extracting third required dataset, please wait...
2020-09-05 12:18:48 : There are 8 additional images for COVID-19 to be saved.
2020-09-05 12:18:48 : Saving additional files. Please wait...
#####
2020-09-05 12:18:48 : Creating new test directory...
2020-09-05 12:18:48 : Creating new train directory...
2020-09-05 12:18:48 : Images for the trainset directories copied.
#####
2020-09-05 12:18:48 : Images for the testset directories moved successfully.
Wall time: 4min 10s

```

A slightly different output is shown below if the code has been run for the second time, third time, and so on. Please note that the current working directory will be different when run on another machine.

```

Current working directory is: G:\Documents\Learning\Udacity\capstone-project\cxr-covid
=====
2020-09-05 12:18:48 : Checking required source directory and files...
2020-09-05 12:18:48 : Source dataset working directory and files already exist.
2020-09-05 12:18:48 : Required dataset already exists.
2020-09-05 12:18:48 : Removing previous test directory...
2020-09-05 12:18:48 : Removing previous train directory...
2020-09-05 12:18:48 : Creating new test directory...
2020-09-05 12:18:48 : Creating new train directory...
2020-09-05 12:18:48 : Images for the trainset directories copied.
#####
2020-09-05 12:18:48 : Images for the testset directories moved successfully.
Wall time: 22.7 s

```

3.2 Implementation

3.2.1 Custom Dataset Class

This section details the use of a custom dataset class to create both test and training dataset instances through the `torch.utils.data` module, and implements the `getitem` function which returns the image and its corresponding class label. The code is shown below.

```

class CXR_Dataset(torch.utils.data.Dataset):
    def __init__(self, image_dirs, transform):
        def get_images(class_name):
            # To simplify list of images, only .png files will be preprocessed
            images = [x for x in os.listdir(image_dirs[class_name]) if x.lower().endswith('png')]
            print(str(dt), f': Found {len(images)} {class_name} examples')
            return images

        self.images = {}
        self.class_names = ['normal', 'viral', 'covid']

        for class_name in self.class_names:
            self.images[class_name] = get_images(class_name)

        self.image_dirs = image_dirs
        self.transform = transform

    def __len__(self):
        return sum([len(self.images[class_name]) for class_name in self.class_names])

    def __getitem__(self, index):
        class_name = random.choice(self.class_names)
        index = index % len(self.images[class_name])
        image_name = self.images[class_name][index]
        image_path = os.path.join(self.image_dirs[class_name], image_name)
        image = Image.open(image_path).convert('RGB')
        return self.transform(image), self.class_names.index(class_name), self.images[class_name][index]

```

The initializer function passes one of the image directories “covid”, “normal”, and “viral” where we will store the corresponding image files. A transform object is also passed, used to do data augmentation for the training dataset transformation. The test set does not need transformation but will still be converted to tensors as required by our model and normalize the values in the input images.

We also have a function called *get_images* where we pass the *class_name* as an argument. This enables to get a list of images and its image directory is a dictionary with the location of the different classes. All the required images have already been converted to .PNG format. Nonetheless, I implemented a check that only .PNG files are considered. The number of images found within each class names will then be printed out.

Another function returns the length of the dataset – the number of images of all the three classes combined.

Finally, the main function of this class *__getitem__* returns the transformed image under a specific class name, including the filename of the image. One of the problems that may be encountered is the number of images for our training set for the covid is a lot lower than, say, the normal or the viral. To help address class imbalance during training, given any index we can randomly select one of the three classes. The code is used to avoid any out-of-bounds index value. The selected image will then be opened and converted through the pillow library to an RGB color space as required by ResNet18.

3.2.2 Image Transformation

The CNN model ResNet18, which is a pre-trained model, does not understand raw images as input. The images will need to be transformed to tensor objects. Two transform objects need to be created – one for the training set and one for the test set.


```

# Train uses the RandomHorizontalFlip for redundancy and accuracy
# Images will be resized to 224 x 224 for manageability
train_transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize(size = (224, 224)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
])

# Test does not need the RandomHorizontalFlip and is not needed here
# Images will be resized to 224 x 224 for manageability
test_transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize(size = (224, 224)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
])

```

The first transform will be for our training set, as indicated in the code shown above. The image is first resized to 224 x 224 as required. A *RandomHorizontalFlip* is next applied. This will randomly flip the image horizontally, where the image is expected to have dimensions such as height, and width. This is used for data augmentation. The image is then converted to tensor for PyTorch to be able to implement calculations. Since we are using a pre-trained model, we then normalize the object using the recommended weights for mean and standard deviation from Imagenet. This is common practice. Also, normalizing an image makes model training stable and fast.

The second transform will be for our test set. Everything else is the same as that of our training set except that for our test set, we do not need to perform data augmentation. As such, the *RandomHorizontalFlip* method is not applied.

3.2.3 The Dataloaders

The CNN model needs to be fed with data that is in the correct format. The *DataLoader* class within the *torch.utils.data* package is used. The first section of the code as shown below indicates our train directories previously created. We will create a training dataset by passing the train directories to our custom dataset class defined above, together with the argument *train_transform* for image transformations. If you execute this code cell, the number of images for each class will be output.

```

train_dirs = {
    'normal': 'COVID-19 Radiography Database/trainset/NORMAL',
    'viral': 'COVID-19 Radiography Database/trainset/Viral Pneumonia',
    'covid': 'COVID-19 Radiography Database/trainset/COVID-19'
}

train_dataset = CXR_Dataset(train_dirs, train_transform)

2020-09-05 12:18:48 : Found 1291 normal examples
2020-09-05 12:18:48 : Found 1295 viral examples
2020-09-05 12:18:48 : Found 631 covid examples

```

The next section shows the creation of the test dataset by passing the test directories to our custom dataset class, with the argument *test_transform*. Please note that there are only 50 images for each of the classes. This parameter is set by *num_random_samples* and can be arbitrarily set. I found out that the more samples you indicate, such as 100, the slower the overall process will be.

```

test_dirs = {
    'normal': 'COVID-19 Radiography Database/testset/normal',
    'viral': 'COVID-19 Radiography Database/testset/viral',
    'covid': 'COVID-19 Radiography Database/testset/covid'
}

test_dataset = CXR_Dataset(test_dirs, test_transform)

2020-09-05 12:18:48 : Found 50 normal examples
2020-09-05 12:18:48 : Found 50 viral examples
2020-09-05 12:18:48 : Found 50 covid examples

```

To create the data loader, the *torch.utils.data.DataLoader* class is used. For this project I opted to use a batch size of 6, which means that the output produced is 6 images with their corresponding labels as predicted by the model. The argument *shuffle = True* is set so that the images selected for processing are not in a particular order or sequence. The data loader is created for the training and test datasets. The code is shown below. An example output is shown, which indicates the total number of training batches as well as the number of test batches.

```

batch_size = 6

dl_train = torch.utils.data.DataLoader(train_dataset, batch_size = batch_size, shuffle = True)
dl_test = torch.utils.data.DataLoader(test_dataset, batch_size = batch_size, shuffle = True)

print(str(dt), ': Number of training batches', len(dl_train))
print(str(dt), ': Number of test batches', len(dl_test))

2020-09-05 12:18:48 : Number of training batches 537
2020-09-05 12:18:48 : Number of test batches 25

```

3.2.4 Data Processing Visualization

Model training takes the bulk of the overall computing time. I find it useful and beneficial to provide a means to display the status of the training rather than just indicating the progress through status bars. With this in mind, a helper function is always especially useful. The code below shows the creation of a helper function *show_images*. Parameters to be passed are the images, labels, corresponding predictions, and the filename of the image. At this point we do not have predictions yet, but labels can be used in place of them. Using the pyplot module from the library matplotlib, we will create a figure of size 8 x 4. A subplot is next defined using one row and *batch_size* columns. The output predictions will be in one row, and the number of images shown will be what is specified by *batch_size*. By default, I have set this to 6 images. The filenames are also printed above the images in order.

```

class_names = train_dataset.class_names

def show_images(images, labels, preds, fn):
    plt.figure(figsize = (8, 4))
    for i, image in enumerate(images):
        plt.subplot(1, batch_size, i + 1, xticks = [], yticks = [])
        image = image.numpy().transpose((1, 2, 0))
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        image = image * std + mean
        image = np.clip(image, 0., 1.)
        plt.imshow(image)
        col = 'green'
        if preds[i] != labels[i]:
            col = 'red'

        plt.xlabel(f'{class_names[int(labels[i].numpy())]}')
        plt.ylabel(f'{class_names[int(preds[i].numpy())]}', color = col)

    for i in fn:
        print(i)

plt.tight_layout()
plt.show()

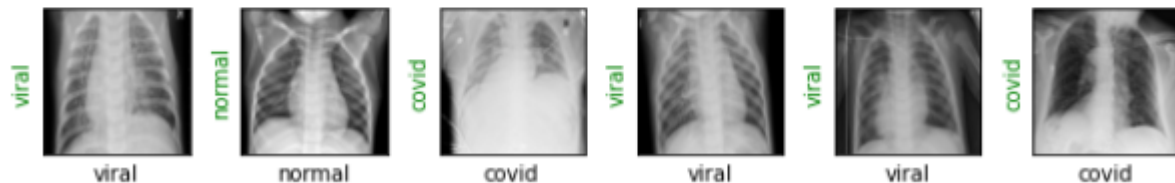
```

Since the labels and images are actually tensors at this point, it is necessary for them to be converted to numpy multi-dimensional array and then execute a transpose. The image needs to be normalized using the recommended mean and standard deviation weights. The image is next displayed, with the x-axis label showing the ground truth, and the y-axis label indicating the prediction. The prediction will be displayed in green if correct and in red otherwise.

The following sample outputs are generated, first for the training set then followed by the test set. Labels are passed as predictions for now.

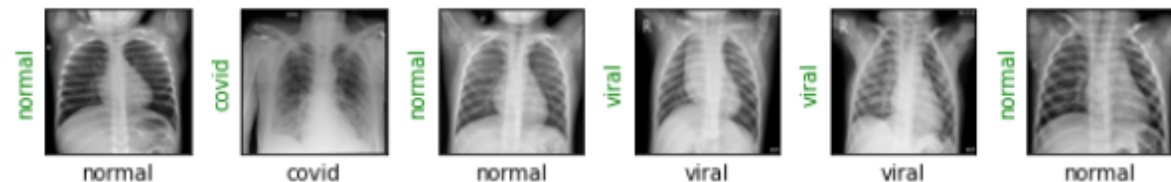
```
images, labels, fn = next(iter(dl_train))
show_images(images, labels, labels, fn)
```

```
Viral Pneumonia (218).png
NORMAL (1259).png
COVID-19(662).png
Viral Pneumonia (837).png
Viral Pneumonia (44).png
COVID-19 (39).png
```



```
images, labels, fn = next(iter(dl_test))
show_images(images, labels, labels, fn)
```

```
NORMAL (422).png
COVID-19(634).png
NORMAL (183).png
Viral Pneumonia (1283).png
Viral Pneumonia (1132).png
NORMAL (1109).png
```



The filenames of the images are printed one after the other. The filenames from top to bottom corresponds to the images from left to right.

3.2.5 Creating the Model

Using the *torchvision* module, we can easily import different model architectures. In this project, ResNet18 is used. While there are other architectures available, I have chosen ResNet18 because it is lighter compared with the others and we can train the model relatively faster while giving us decent results.

```
resnet18 = torchvision.models.resnet18(pretrained = True)

# print(resnet18)|
```

Shown above is the code to construct our model. The parameter *pretrained* is set to *True*. This will leverage the model's pre-training with Imagenet. In PyTorch, you can also print out the model architecture. Just uncomment the *print* line.

ResNet18 is trained in more than a million images in Imagenet having 1000 classes. Since we are classifying images only in three classes, we are going to change the last fully connected layer, the *out_features* parameter, to reflect that. The code is shown below.

```
resnet18.fc = torch.nn.Linear(in_features = 512, out_features = 3)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet18.parameters(), lr = 3e-5)|
```

The loss function chosen for this model is found in the module *torch.nn.CrossEntropyLoss*, because we are using multiple classification. The optimizer used for this model is *Adam* from the *torch.optim* module. The learning rate is rather small, but experimentation with the code yielded optimal results for the value given.

```
def show_preds():
    resnet18.eval()
    images, labels = next(iter(dl_test))
    outputs = resnet18(images)
    _, preds = torch.max(outputs, 1)
    show_images(images, labels, preds)|
```

Next, a method is defined to show the predictions from our ResNet18 model. First, we set the model to evaluation mode, then iterate over the test data loader. We then pass the output to the class we previously defined, *show_images*.

3.2.6 Training the Model

A training method is defined as shown in the code snippet below. In this section of the code, I have defined some parameters by which the code will make iterations until the conditions are satisfied. Given the way the model iterates on the processing, it is possible that an accuracy of 100% is achieved. Also, iterative processing could take a very long time. Therefore *target_acc*, *target_vl*, and *ts_tick* parameters have been assigned that corresponds to target accuracy, target validation loss and the number of training steps taken to produce an update, respectively. Training loss as well as validation loss are also initialized.

```
def model_train(epochs):
    # define target accuracy, target validation loss, and test step tick
    target_acc = 0.975
    target_vl = 0.150
    ts_tick = 10

    print(str(dt), ': Starting model training...')
    for e in range(0, epochs):
        print('=' * 70)
        print(str(dt), f': Starting epoch {e + 1} / {epochs}')
        print('=' * 70)
        print(str(dt), f': Updating results every {ts_tick} training steps...')
        print('=' * 70)

        train_loss = 0.
        val_loss = 0.
```

Epoch is set to 1 for now, but you can set that to more than 1. However, since we are already evaluating our model more than once within an epoch, increasing its value may not be necessary. We set our CNN model to training mode with *resnet18.train()* to begin our training.

Next, we iterate on our training set setting the gradient to zero to refresh our optimizer, set our outputs, and set our loss through the *loss_fn* function as defined previously. After getting the loss, we will take a gradient step with *loss.backward()*. This is essentially a back propagation. To complete the gradient step, *optimizer.step()* is executed thus updating all the parameter values. We also update the training loss value.

In every training step reached as indicated by *ts_tick*, we will evaluate our model. We will need to initialize the accuracy, preparing it for calculations later. Set the ResNet18 to evaluation mode, then iterate through the data loader for test for validation step. Please see the snapshot of the code below.


```

# set model to training phase
resnet18.train()

for train_step, (images, labels, fn) in enumerate(dl_train):
    optimizer.zero_grad()
    outputs = resnet18(images)
    loss = loss_fn(outputs, labels)
    loss.backward()
    optimizer.step()
    train_loss += loss.item()
    if train_step % ts_tick == 0:
        print(str(dt), ': Evaluating at step', train_step)

        accuracy = 0

# set model to evaluation phase
resnet18.eval()

for val_step, (images, labels, fn) in enumerate(dl_test):
    outputs = resnet18(images)
    loss = loss_fn(outputs, labels)
    val_loss += loss.item()

    _, preds = torch.max(outputs, 1)
    accuracy += sum((preds == labels).numpy())

val_loss /= (val_step + 1)
accuracy = accuracy / len(test_dataset)
print(str(dt), f': Validation Loss: {val_loss:.4f}, Accuracy: {accuracy:.4f}')

```

Validation loss is then calculated. For the predictions, the indices returned from *torch.max* are used. We can then calculate the accuracy by counting the predictions matching the labels of the images then dividing it by the length of the test dataset. The predictions will then be displayed, the model set back to training and the training loops for the next step. Once accuracy and validation loss targets are satisfied or exceeded, we return the values and indicate the training step reached. We will use the best model for our final predictions. The code is shown below, followed by a sample output.

```

show_preds()

# set model to training phase
resnet18.train()

if accuracy >= target_acc and val_loss <= target_vl:
    print(str(dt), ': Target accuracy of ' + str(target_acc) + ' satisfied or exceeded.')
    print(str(dt), ': Target validation loss of ' + str(target_vl) + ' satisfied or exceeded.')
    print(str(dt), ': Stopping training...')
    break

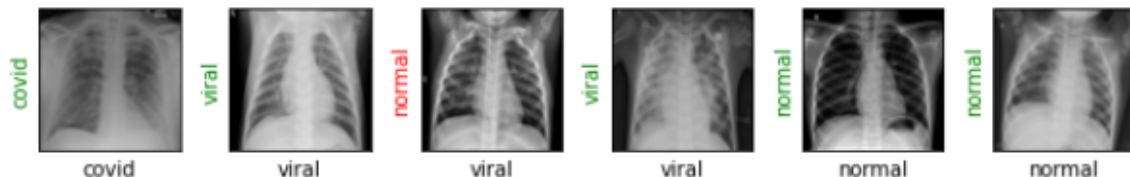
train_loss /= (train_step + 1)
print(str(dt), f': Training Loss: {train_loss:.4f}')
print(str(dt), ': Training complete.')
print(str(dt), ': Using best model trained in step ' + str(train_step) + ', epoch ' + f'{e + 1}' + '.')

```

```

2020-09-06 12:39:01 : Evaluating at step 190
2020-09-06 12:39:01 : Validation Loss: 0.1636, Accuracy: 0.9533
COVID-19(335).png
Viral Pneumonia (870).png
Viral Pneumonia (694).png
Viral Pneumonia (1217).png
NORMAL (596).png
NORMAL (1334).png

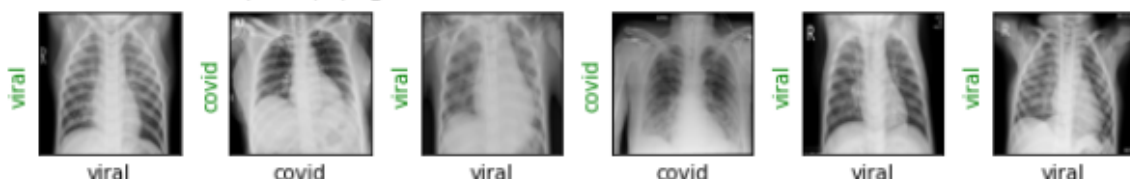
```



```

2020-09-06 12:39:01 : Evaluating at step 200
2020-09-06 12:39:01 : Validation Loss: 0.1103, Accuracy: 0.9800
Viral Pneumonia (588).png
COVID-19(294).png
Viral Pneumonia (937).png
COVID-19(634).png
Viral Pneumonia (1297).png
Viral Pneumonia (1132).png

```



```

2020-09-06 12:39:01 : Target accuracy of 0.975 satisfied or exceeded.
2020-09-06 12:39:01 : Target validation loss of 0.15 satisfied or exceeded.
2020-09-06 12:39:01 : Stopping training...
2020-09-06 12:39:01 : Training Loss: 0.3536
2020-09-06 12:39:01 : Training complete.
2020-09-06 12:39:01 : Using best model trained in step 200, epoch 1.
Wall time: 7min 21s

```

3.2.7 Final Output

Once we have successfully trained our model, we can now use the best model for our final predictions. Simply calling out and executing `show_preds()` will show the predicted classes of the images against their corresponding ground truths. An example output is shown below.

```
show_preds()
```

```
NORMAL (750).png  
Viral Pneumonia (103).png  
COVID-19(502).png  
COVID-19(244).png  
Viral Pneumonia (770).png  
NORMAL (28).png
```



3.3 Refinement

Machine learning, in general, derives its conclusions based on the data supplied to it, how you explore the data and the way feature engineering is done prior to any development. In other words, data preprocessing puts a significantly large amount of work in the hands of a machine learning engineer.

In order to get the best possible solution, we need to experiment on the algorithm previously described. What will happen if I increase the number of random samples to, say, 100? The test dataset will be larger, of course, with 100 image files for each of the classes – normal, covid, and viral. Consequently, I would be expecting a longer training on the dataset as a result of this. I changed the line of the code described in section 3.1.1 above to the following.

```
num_random_samples = 100
```

Since the target accuracy has been defined to be at 0.975, which is already relatively high, I want to push this to 0.980 and see if this can be achieved while using only a single epoch. Validation loss, which was previously defined as 0.150, is now set at 0.100. The following code lines as described in section 3.2.6 have been changed to the following:

```
# define target accuracy, target validation loss, and test step tick

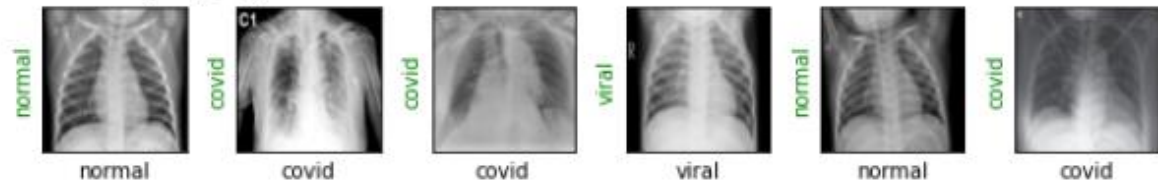
target_acc = 0.980

target_vl = 0.100

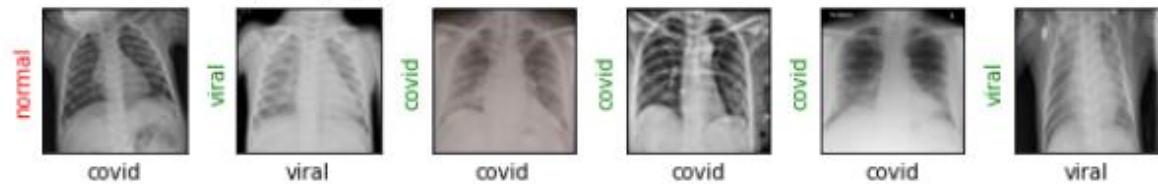
ts_tick = 10
```

The training step, *ts_tick*, has been retained at 10 as I wanted to see the output every 10 training steps. A sample output of the experiment is shown below. It is important to note that the training is significantly longer than before, with training steps reaching to 260 before the model found the best solution at that training step. Also, the target accuracy of 0.9833 has exceeded our defined target accuracy of 0.980. The validation loss of 0.0696 has exceeded the target validation loss defined at 0.100. Training loss is also at a low 0.2813 compared to 0.3536 in section 3.2.6 above. I am drawn into the conclusion that having more images for the test set covid class will make our training longer. However, it will also make our model to be trained deeper and a more accurate result can be achieved.

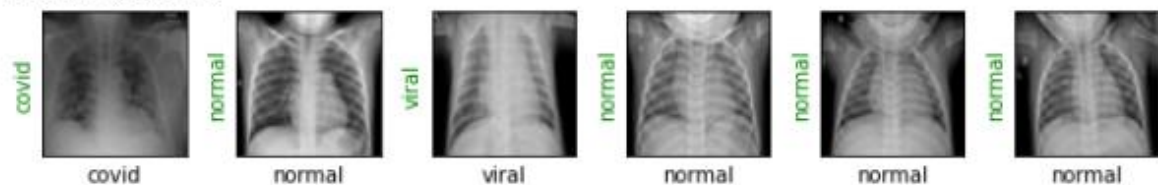
2020-09-06 12:39:01 : Evaluating at step 240
 2020-09-06 12:39:01 : Validation Loss: 0.0895, Accuracy: 0.9767
 NORMAL (236).png
 COVID-19(605).png
 COVID-19(488).png
 Viral Pneumonia (292).png
 NORMAL (974).png
 COVID-19 (54).png



2020-09-06 12:39:01 : Evaluating at step 250
 2020-09-06 12:39:01 : Validation Loss: 0.0880, Accuracy: 0.9733
 COVID-19(194).png
 Viral Pneumonia (763).png
 COVID-19(410).png
 COVID-19(458).png
 COVID-19(235).png
 Viral Pneumonia (1234).png



2020-09-06 12:39:01 : Evaluating at step 260
 2020-09-06 12:39:01 : Validation Loss: 0.0696, Accuracy: 0.9833
 COVID-19(168).png
 NORMAL (1233).png
 Viral Pneumonia (1175).png
 NORMAL (12).png
 NORMAL (388).png
 NORMAL (9).png




2020-09-06 12:39:01 : Target accuracy of 0.98 satisfied or exceeded.
 2020-09-06 12:39:01 : Target validation loss of 0.1 satisfied or exceeded.
 2020-09-06 12:39:01 : Stopping training...
 2020-09-06 12:39:01 : Training Loss: 0.2813
 2020-09-06 12:39:01 : Training complete.
 2020-09-06 12:39:01 : Using best model trained in step 260, epoch 1.
 Wall time: 10min 53s

4 Results

4.1 Model Evaluation and Validation

The development of the project involved creating a custom dataset for our test set and training set. The model was evaluated based on the target accuracy and target validation loss specified. The final model is then chosen based on the best model discovered during training. In typical machine learning, we normally have more than one epoch. In this project, specifying more than one epoch is possible, although a single epoch would be sufficient, as we are evaluating the model over iterations of many training steps in a single epoch. As a result, the specified target accuracy of 0.975 has been exceeded with 0.980 and the target validation loss of 0.150 has been exceeded with 0.1103. Also, the training loss achieved is 0.3536, as shown in the screenshot below.

```
2020-09-06 12:39:01 : Evaluating at step 200
2020-09-06 12:39:01 : Validation Loss: 0.1103, Accuracy: 0.9800
Viral Pneumonia (588).png
COVID-19(294).png
Viral Pneumonia (937).png
COVID-19(634).png
Viral Pneumonia (1297).png
Viral Pneumonia (1132).png
```



```
2020-09-06 12:39:01 : Target accuracy of 0.975 satisfied or exceeded.
2020-09-06 12:39:01 : Target validation loss of 0.15 satisfied or exceeded.
2020-09-06 12:39:01 : Stopping training...
2020-09-06 12:39:01 : Training Loss: 0.3536
2020-09-06 12:39:01 : Training complete.
2020-09-06 12:39:01 : Using best model trained in step 200, epoch 1.
Wall time: 7min 21s
```

The multi-class classifier performs well with the given dataset. Although, increasing the number of random sample images the model takes as inputs, the longer it takes for the best model to be realized.

4.2 Justification

The final model trained in about 200 steps has achieved a remarkable accuracy of 0.980 or 98% compared to the benchmark model Covid-cxr with an accuracy of 0.920 or 92%, as seen on section 2.3 above. This does not in any way conclude that this project is better, but it only shows that ResNet18 can be a particularly useful model architecture in image classification.

Covid-cxr is an on-going and extensive work in image classification of COVID-19-positive chest X-rays. Please note, however, that Covid-cxr is not intended for clinical purposes of diagnosing COVID-19.

While the number of samples are relatively low for each class in our source dataset, the model has achieved a remarkable accuracy and validation loss. Some image classifiers are sometimes trained on tens of thousands of image samples. In this project, I believe that the chosen algorithm solution is sufficient and capable to meet the solution for the given problem.

5 Conclusion

5.1 Summary and Reflection

Any machine learning project or endeavor involves a lot of planning. While there are many algorithms and architectures of choice to tackle a given problem, it is significantly important to put an emphasis is data collection, exploration, and feature engineering. The result of any give model training is heavily influenced by the data you feed to it. As such, data preprocessing and exploration has been the area I have spent my most time on. I initially expected that the training would take the most time but fortunately, we have ResNet18 – a stable, reliable, and efficient image classifier already pre-trained on more than a million images and 1000 classes – more than sufficient for our given problem where we only classify images in three classes.

To summarize what has been done, the (what I call) 4D's of machine learning have been implemented:

1. Define – the problem has been well defined: Classify an X-ray image under three classes – normal, viral, and covid (detect the presence of COVID-19)
2. Discover – Sourced out the dataset – used three different datasets for this project, preprocessed the source data and augmented the list of images, discovered its features, and extracted features that are not readily apparent, and data was explored
3. Develop – PyTorch and ResNet18 model was chosen for the task
4. Deploy – Although we are not deploying the model into production, I consider this to be the final approach in solving the given problem. The model deployed and the final solution presented

Overall, the chosen model performed better than expected, even though the source dataset contained significantly lesser number of COVID-19-positive X-ray images.

5.2 Improvement

Previously mentioned in the capstone proposal for this project is that the images classified will need to be explained using the LIME solution – Local Interpretable Model-Agnostic Explanations – of which an article is found [here](#) [8]. This is useful when we want the explanation to really reflect the behavior of our chosen classifier around the instance being predicted. While this is a powerful tool, it has not been implemented in this project due to time constraints. If this project is to proceed in its maturity, LIME explanations will definitely add value to the solution.

Another area that might be considered to be improved is the presentability of the output. An API would be beneficial – a single X-ray image input is required, and the model would predict the presence of COVID-19, if found positive. Furthermore, a mobile app can be utilized for portability.

6 References

- [1] Kaggle <https://www.kaggle.com/tawsifurrahman/covid19-radiography-database>
- [2] Joseph Paul Cohen. Postdoctoral Fellow, Mila, University of Montreal
<https://github.com/ieee8023/covid-chestxray-dataset>
- [3] Core COVID-Net Team <https://github.com/agchung/Figure1-COVID-chestxray-dataset>
- [4] Towards Data Science <https://towardsdatascience.com/how-to-use-covid-cxr-to-predict-covid-19-on-chest-x-rays-with-explainable-ai-c05cb07d5f0f>
- [5] ImageNet database <http://image-net.org/>
- [6] CodeSpeedy <https://www.codespeedy.com/detection-of-covid-19-from-chest-x-ray-images-using-machine-learning/>
- [7] Towards Data Science <https://towardsdatascience.com/investigation-of-explainable-predictions-of-covid-19-infection-from-chest-x-rays-with-machine-cb370f46af1d>
- [8] Towards Data Science <https://towardsdatascience.com/local-model-interpretation-an-introduction-90d039fbef8d>
- [9] Kaggle <https://www.kaggle.com/pytorch/resnet18>
- [10] MathWorks <https://www.mathworks.com/help/deeplearning/ref/resnet18.html>