SAPIENZA
UNIVERSITÀ DI ROMA

Vision & Perception Final Project

GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in COVID-19 Classification

Members:

Akash Garg
189271

Simone Rossetti
1900592

Supervisor:
Prof. Paolo Russo

Examiner:
Prof. Fiora Pirri

ALCOR Laboratory
SAPIENZA UNIVERSITY OF ROME
Rome, Italy 2020

Vision & Perception Final Project
GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in COVID-19 Classification
Akash Garg - Simone Rossetti
ALCOR Laboratory
SAPIENZA UNIVERSITY OF ROME
Rome, Italy 2020

Abstract

Large-scale annotated datasets have proven to be extremely important part that powers the convolutional neural networks (CNNs) and other deep learning methods. However, it has been observed that obtaining such an enormous dataset is quite challenging in certain domains.
Medical domain is one such area where obtaining an enormous number of images in order to train a classifier is highly challenging and practically inefficient. In this project we try to address the stated critical issue with the help of Generative Adversarial Networks (GANs). We demonstrate that GAN can be used to generate synthetic medical images when trained with even small number of available real images. Not only this the quality of synthetic images is quite high and thus can be used to augment the original training dataset for further Machine Learning applications. In order to validate the quality of generated images we train a classifier firstly using classical augmentation techniques (rotation, translation, flip and scale) on smaller dataset of available real images. After that another classifier is trained but using real images, images coming from classical augmentation as well as synthetic images generated by the GAN as training dataset. In the latter case we observed an increase in sensitivity and in specificity. Adding to this the total accuracy of classification also increased significantly.
In this project we took inspiration from the work done by Maayan Frid-Adar et. al. in this paper: *GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classification*, `https://arxiv.org/pdf/1803.01229.pdf`, and by Mohamed Loey et. al. in this paper *Within the Lack of Chest COVID-19 X-ray Dataset: A Novel Detection Model Based on GAN and Deep Transfer Learning*, `https://www.mdpi.com/2073-8994/12/4/651/pdf`

Keywords: Image synthesis, Convolutional Neural Network, Generative Adversarial Network, Data Augmentation.

Acknowledgements

The authors would like to thank Prof. Fiora Pirri whose lectures aroused our interest in the subject, and Prof. Paolo Russo, who supported, guided and directed us in this work.

Akash Garg, New Delhi, September 2020
Simone Rossetti, Roma, September 2020

Contents

List of Figures **vi**

List of Tables **ix**

1 Introduction **1**
1.1 Motivations and Goals 1
1.2 Overview 2

2 Resources Used **3**
2.1 Platform Used 3
2.2 Dataset 3
2.3 Framework used 4

3 Architecture **5**
3.1 GAN 5
3.1.1 Generator Architecture 6
3.1.2 Discriminator Architecture 6
3.2 Classifier 6

4 Training **9**
4.1 GAN 9
4.1.1 Batch Size 9
4.1.2 Transforms of Training Samples 9
4.1.3 Hyperparameters 10
4.2 Classifier 10
4.2.1 Batch Size 10
4.2.2 Transforms of Training Samples 11
4.2.3 Hyperparameters 12

5 Evaluation Metrics **13**
5.1 GAN 13
5.1.1 Convergence Failure 13
5.1.1.1 Discriminator Dominates 13
5.1.1.2 Generator Dominates 14
5.2 Classifier 14

6 Results **16**

6.1 Synthetic Images Generated from GAN 16
6.1.1 Comparison at different learning rates 16
6.1.2 Comparison at different epochs 18
6.1.3 Comparison at different channels of the training images 19
6.1.3.1 Consistency in results w.r.t. those reported in the original paper[1] 19
6.2 Classification 20

7 Conclusion **32**

List of Figures

2.1 Samples taken from 'COVID-19 image data collection' test set. 4

3.1 GAN architecture 7
3.2 CNN COVID-19 X-ray classifier architecture. 8

4.1 64 random elements batch from (a) original dataset, (b) classic augmented dataset, (c) synthetic augmented dataset 11

5.1 Discriminator overpowering the generator. 14
5.2 Generator overpowering the discriminator. 14
5.3 Multi-class confusion matrix notation and relative indices. 14

6.1 A pair of real (left) and fake X-ray images generated from the GAN (right). (a-b) for Covid ; (c-d) for Normal; (e-f) for Pneumonia bacteria; (g-h) for Pneumonia virus. 17
6.2 Graph of generator loss vs epoch (top) , discriminator loss vs epoch (center) and the stack of fake images (bottom). (a-c) for generator network with lr = 0.0002; (d-f) for generator network with lr = 0.002 18
6.3 Comparison at different epochs (a) Original pneumonia virus X-ray image; (b) Generated image at 50 epoch; (c) Generated image at 100 epoch; (d) Generated image at 200 epoch 19
6.4 Graph of D_x vs epoch (top), D_G_z1 vs epoch (center1), D_G_z2 vs epoch (center2), & generated images (bottom) for GAN trained on pneumonia bacteria X-ray images. (a-d) 1-channel (1-C) images; (e-h) 3-channel (3-C) images 22
6.5 Experiment flowchart for evaluating synthetic data augmentation in the task of classifying COVID-19 and pneumonia infection. 23
6.6 CNN trained on default trainset (O1+C0+G0), 750 epochs, 0.769 total accuracy. 24
6.7 CNN trained on classic augmented trainset (O1+C1+G0), 750 epochs, 0.772 total accuracy at 300 epochs. 25
6.8 CNN trained on classic augmented trainset (O1+C2+G0), 750 epochs, 0.828 total accuracy at 300 epochs. 26
6.9 CNN trained on classic augmented trainset (O1+C3+G0), 750 epochs, 0.827 total accuracy at 750 epochs. 27
6.10 CNN trained on both classic and synthetic augmented trainset (O1+C3+G1), 750 epochs, 0.874 total accuracy at 750 epochs. 28

6.11 CNN trained on both classic and synthetic augmented trainset (O1+C3+G2), 750 epochs, 0.869 total accuracy at 300 epochs. 29
6.12 CNN trained on both classic and synthetic augmented trainset (O1+C3+G3), 750 epochs, 0.907 total accuracy at 720 epochs. 30
6.13 Total accuracy results for pneumonia infection classification of covid, normal, pneumonia bacteria and pneumonia virus with the increase of training set size. The blue line shows the effect of adding classic data augmentation and the red line shows the effect of adding synthetic data augmentation. 31
6.14 CNN predictions improvement given by GAN data augmentation. . . 31

List of Tables

2.1 Number of images for each class in the COVID-19 dataset. 3

1
Introduction

In the context of a COVID-19 pandemic, we want to contribute to improve patient healthcare prognostic predictions. Data is the first step to developing any diagnostic/prognostic tool. While there exist large public datasets of more typical chest X-rays, there are no much collection of COVID-19 chest X-rays designed to be used for computational analysis.
Deal with the small datasets and limited amount of annotated samples is a recurrent problem in the medical imaging domain, especially when employing supervised machine learning algorithms that require labeled data and larger training examples. Although public medical datasets are available online, most datasets are still limited in size and only applicable to specific medical problems, the reason is that collecting medical data is a complex and expensive procedure and researchers use to attempt to overcome this challenge by using data augmentation. The most classical data augmentation methods include simple modifications of dataset images such as translation, rotation, flip and scale, which is a standard procedure in computer vision tasks. This is because little additional information can be gained from small modifications to the images (i.e. the translation of the image a few pixels to the right).
Synthetic data augmentation of high quality examples is new, sophisticated type of data augmentation. Synthetic data examples learned using a generative model enable more variability and enrich the dataset to further improve the system training process. One such promising approach inspired by game theory for training a model that syntheses images is known as Generative Adversarial Networks (GANs). The model consists of two networks that are trained in an adversarial process where one network generates fake images and the other network discriminates between real and fake images repeatedly.
In the current work we used deep learning methodology to classify pneumonia infection origin.

1.1 Motivations and Goals

We conducted different experiments in order to evaluate the potentiality and performances that synthetic augmentation produces in image supervised learning algorithms.

We started studying the method presented in the paper *GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classifi-*

cation by Maayan Frid-Adar et al., 2018. The paper conceived a strong interest in us about how GANs can be used to solve narrow datasets problems and improve supervised learning methods.

We implemented the code following the method presented in the above research and we conducted several experiments on a different dataset named *COVID-19 image data collection* by Joseph Paul Cohen, Paul Morrison and Lan Dao, 2020.

We definitely achieved the high quality synthesis of chest X-ray scans using generative adversarial networks (GANs).

We designed a CNN-based solution for pneumonia infection classification task, with comparable results to state-of-the-art methods.

We demonstrated how synthetic augmentation of the CNN training set, using the synthetic data generated by GANs, highly improved classification results: the classification performance using only classic data augmentation yielded 82.7% sensitivity and 90.4% specificity (average). By adding the synthetic data augmentation the results increased to 90.6% sensitivity and 97.2% specificity (average).

1.2 Overview

The work is organized as follows:
In Chapter 2 we are going to present the background of our work: the framework, the libraries and the datasets used. In Chapter 3 we will focus on the particular networks architectures we adopted. Chapter 4 contains the technical description of the training process we defined for each network. In Chapter 5 we will discuss about some basic knowledge to evaluate the performances of GAN and CNN. Chapter 6 describes in detail the experiments we conducted and the results obtained. Chapter 7 contains the conclusions, the possible future applications and summarizes our work.

2

Resources Used

2.1 Platform Used

The task at hand required us to train GAN model as well as Classifier at various epochs and learning rates for experimentation and analysis. Thanks to Google Colab we could use the Google servers and GPU to efficiently train our models.

2.2 Dataset

The authors in the paper[1] we referred to have used Liver Lesions collected from Sheba Medical Center. They considered three classes of the lesion (cysts, metastases and hemangiomas). However due to unavailability of the dataset to us we decided to search for another dataset relevant to our application.

We trained all our models on the covid chest Xray dataset which we found at the Github link https://github.com/vj2050/Transfer-Learning-COVID-19. The dataset is well structured and contains 4 classes namely, covid, normal, pneumonia bacteria and pneumonia viral. The dataset has a total of 270 training images (60,70,70,70) and 36 test images (9,9,9,9). We used the default train/test split proposed by the creators of the dataset.

The above dataset works as the real images input to the Generative Adversarial Network which produces synthetic images for each of the four classes. Moreover the same dataset is also used to train the classifier model.

Set/Class	**Covid**	**Normal**	**Pneumonia_bac**	**Pneumonia_vir**	**Total**
Train	60	70	70	70	270
Test	9	9	9	9	36
Total	69	79	79	79	306

Table 2.1: Number of images for each class in the COVID-19 dataset.

[1] https://arxiv.org/pdf/1803.01229.pdf

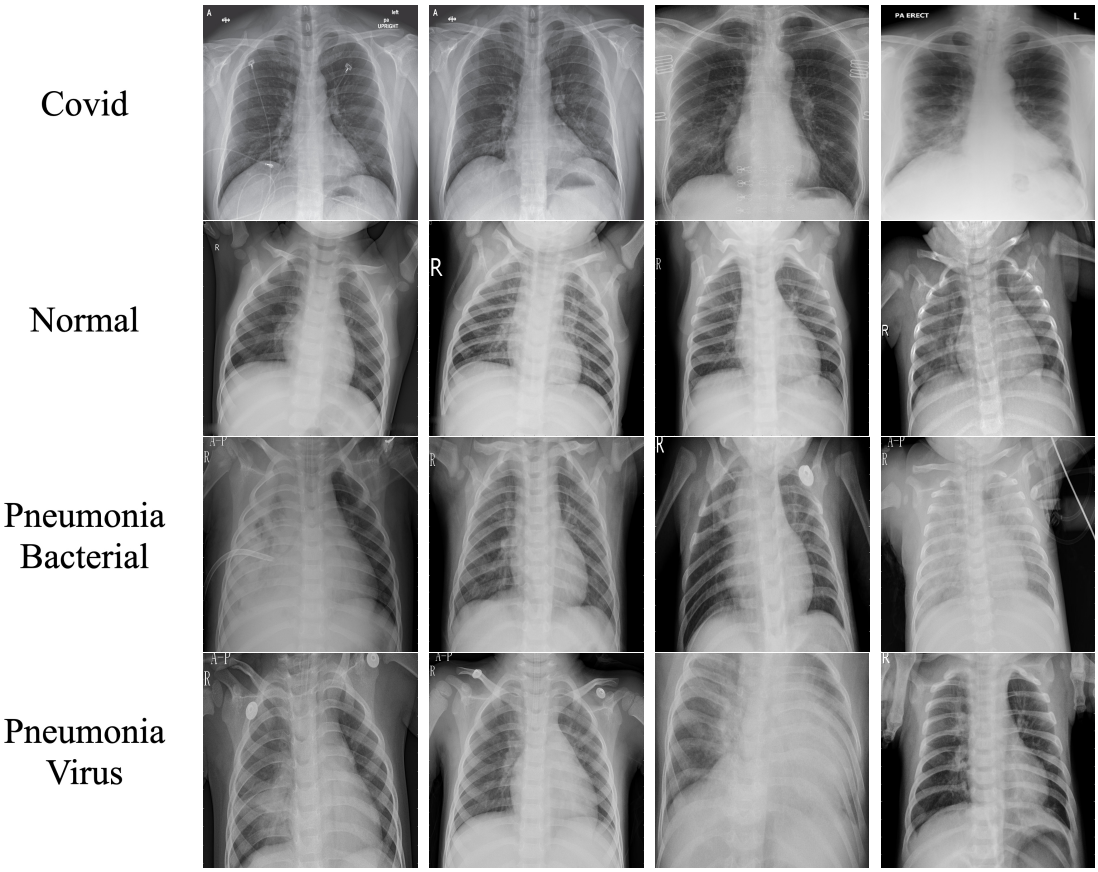


Figure 2.1: Samples taken from 'COVID-19 image data collection' test set.

2.3 Framework used

- **PyTorch:** We used PyTorch, the open source machine learning library, throughout the project. The library is primarily based and developed upon the Torch library. It provides easy to use functions and high level APIs much required for Machine learning and Deep Neural Network projects.

3
Architecture

As listed in the previous chapter the dataset of covid chest xray is quite small. In order to augment the dataset we generate synthetic images for each of the three classes in the original dataset. For this purpose we used Generative Adversarial Networks (GANs). This required us to train a separate GAN for each class. Further, a convolutional neural network (CNN) classifier is built to evaluate the effect of synthetic images on the classification of the three classes of chest Xray dataset. A detailed explanation of the GAN and the classifier architecture has been provided below.

3.1 GAN

GANs are specific framework of a generative model. A generative model aims to implicitly learn the data distribution p_{data} from a set of samples $x^{(1)}, ..., x^{(m)}$ (e.g. images) to further generate new samples drawn from the learned distribution. We used Deep Convolutional GAN (DCGAN) where both the Generator (G) and Discriminator (D) networks are deep CNNs.
The model consists of two neural networks that are trained simultaneously. The first network is termed the discriminator and is denoted D. The role of the discriminator is to discriminate between the real and fake samples. It is inputted a sample x and outputs D(x), its probability of being a real sample. The second network is termed the generator and is denoted G. The generator synthesizes samples that D will consider to be real samples with high probability. G gets input samples $z^{(1)}, ..., z^{(m)}$ from a known simple distribution p_z, usually a uniform distribution, and maps G(z) to the image space of distribution p_g. The goal of G is to achieve $p_g = p_{data}$.
Adversarial networks are trained by optimizing the following loss function of a two-player minimax game:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}} log D(x) + \mathbb{E}_{x \sim p_x}[log(1 - D(G(z)))] \tag{3.1}$$

The discriminator is trained to maximize D(x) for images with $x \sim p_{data}$ and to minimize D(x) for images with $x \not\sim p_{data}$. The generator produces images G(z) to fool D during training such that $D(G(z)) \sim p_{data}$. Therefore, the generator is trained to maximize $D(G(z))$, or equivalently minimize $1 - D(G(z))$. During training the generator improves in its ability to synthesize more realistic images while the discriminator improves in its ability to distinguish the real from the synthesized images. Hence the moniker of adversarial training.

3.1.1 Generator Architecture

The generator network takes a vector of 100 random numbers drawn from a uniform distribution as input and outputs a chest X-ray image of size 64 × 64 × 1. The network architecture consists of a fully connected layer reshaped to size 4 × 4 × 512 and four fractionally-strided convolutional layers to up-sample the image with a 4 × 4 kernel size. A fractionally-strided convolution can be interpreted as expanding the pixels by inserting zeros in between them. Batch-normalization is applied to each layer of the network, except for the output layer. Normalizing responses to have zero mean and unit variance over the entire mini-batch stabilizes the GAN learning process and prevents the generator from collapsing all samples to a single point. ReLU activation functions are applied to all layers except the output layer which uses a tanh activation function. Figure 3.1(a) shows the generator network architecture along with the trainable parameters.

3.1.2 Discriminator Architecture

The discriminator network takes the input image (chest X-ray) of size 64 × 64 × 1, and outputs one decision: is this X-ray image real or fake? The architecture of the discriminator network is shown in Figure 3.1(b). The network consists of four convolution layers with a kernel size of 4 × 4 and a fully connected layer. Strided convolutions are applied to each convolution layer to reduce spatial dimensionality. Batch-normalization is applied to each layer of the network, except for the input and output layers. Leaky ReLU activation functions f(x) = max (x, leak × x) are applied to all layers except the output layer which uses the Sigmoid function for the likelihood probability (0, 1) score of the image. Also dropout is applied to each layer of the network, expect for the second and the output layers.

3.2 Classifier

The architecture of the classifier we propose and the relative trainable parameters quantities are shown in the *Figure* 3.2. CNN architectures for medical imaging usually contain fewer convolutional layers because of the small datasets and smaller input size. Our classification CNN gets fixed size grayscale input X-rays of 64×64 normalized in the range $(0, 1)$.
The architecture consists in three convolutional layers with kernel 3 × 3, ReLU activation functions, batch normalization and max pooling. The output layer is made of two fully connected layers with Dropout 0.5 and Softmax output function over the 4 classes. Batch normalization and dropout are added in order to avoid overfit of the trainset, which is very likely in small size datasets.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 1, 64, 64]           --
│    └─ConvTranspose2d: 2-1              [-1, 512, 4, 4]           819,200
│    └─BatchNorm2d: 2-2                  [-1, 512, 4, 4]           1,024
│    └─ReLU: 2-3                         [-1, 512, 4, 4]           --
│    └─ConvTranspose2d: 2-4              [-1, 256, 8, 8]           2,097,152
│    └─BatchNorm2d: 2-5                  [-1, 256, 8, 8]           512
│    └─ReLU: 2-6                         [-1, 256, 8, 8]           --
│    └─ConvTranspose2d: 2-7              [-1, 128, 16, 16]         524,288
│    └─BatchNorm2d: 2-8                  [-1, 128, 16, 16]         256
│    └─ReLU: 2-9                         [-1, 128, 16, 16]         --
│    └─ConvTranspose2d: 2-10             [-1, 64, 32, 32]          131,072
│    └─BatchNorm2d: 2-11                 [-1, 64, 32, 32]          128
│    └─ReLU: 2-12                        [-1, 64, 32, 32]          --
│    └─ConvTranspose2d: 2-13             [-1, 1, 64, 64]           1,024
│    └─Tanh: 2-14                        [-1, 1, 64, 64]           --
==========================================================================================
Total params: 3,574,656
Trainable params: 3,574,656
Non-trainable params: 0
Total mult-adds (M): 423.53
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 1.91
Params size (MB): 13.64
Estimated Total Size (MB): 15.54
==========================================================================================
```

(a) GAN COVID-19 X-ray generator architecture.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 1, 1, 1]             --
│    └─Conv2d: 2-1                       [-1, 64, 32, 32]          1,024
│    └─LeakyReLU: 2-2                    [-1, 64, 32, 32]          --
│    └─Dropout: 2-3                      [-1, 64, 32, 32]          --
│    └─Conv2d: 2-4                       [-1, 128, 16, 16]         131,072
│    └─BatchNorm2d: 2-5                  [-1, 128, 16, 16]         256
│    └─LeakyReLU: 2-6                    [-1, 128, 16, 16]         --
│    └─Conv2d: 2-7                       [-1, 256, 8, 8]           524,288
│    └─BatchNorm2d: 2-8                  [-1, 256, 8, 8]           512
│    └─LeakyReLU: 2-9                    [-1, 256, 8, 8]           --
│    └─Dropout: 2-10                     [-1, 256, 8, 8]           --
│    └─Conv2d: 2-11                      [-1, 512, 4, 4]           2,097,152
│    └─BatchNorm2d: 2-12                 [-1, 512, 4, 4]           1,024
│    └─Dropout: 2-13                     [-1, 512, 4, 4]           --
│    └─LeakyReLU: 2-14                   [-1, 512, 4, 4]           --
│    └─Conv2d: 2-15                      [-1, 1, 1, 1]             8,192
│    └─Sigmoid: 2-16                     [-1, 1, 1, 1]             --
==========================================================================================
Total params: 2,763,520
Trainable params: 2,763,520
Non-trainable params: 0
Total mult-adds (M): 104.48
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 1.38
Params size (MB): 10.54
Estimated Total Size (MB): 11.93
==========================================================================================
```

(b) GAN COVID-19 X-ray discriminator architecture.

Figure 3.1: GAN architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 32, 32, 32]          --
│    └─Conv2d: 2-1                       [-1, 32, 64, 64]          320
│    └─ReLU: 2-2                         [-1, 32, 64, 64]          --
│    └─BatchNorm2d: 2-3                  [-1, 32, 64, 64]          64
│    └─MaxPool2d: 2-4                    [-1, 32, 32, 32]          --
├─Sequential: 1-2                        [-1, 64, 16, 16]          --
│    └─Conv2d: 2-5                       [-1, 64, 32, 32]          18,496
│    └─ReLU: 2-6                         [-1, 64, 32, 32]          --
│    └─BatchNorm2d: 2-7                  [-1, 64, 32, 32]          128
│    └─MaxPool2d: 2-8                    [-1, 64, 16, 16]          --
├─Sequential: 1-3                        [-1, 128, 8, 8]           --
│    └─Conv2d: 2-9                       [-1, 128, 16, 16]         73,856
│    └─ReLU: 2-10                        [-1, 128, 16, 16]         --
│    └─BatchNorm2d: 2-11                 [-1, 128, 16, 16]         256
│    └─MaxPool2d: 2-12                   [-1, 128, 8, 8]           --
├─Sequential: 1-4                        [-1, 4]                   --
│    └─Flatten: 2-13                     [-1, 8192]                --
│    └─Linear: 2-14                      [-1, 256]                 2,097,408
│    └─ReLU: 2-15                        [-1, 256]                 --
│    └─Dropout: 2-16                     [-1, 256]                 --
│    └─Linear: 2-17                      [-1, 4]                   1,028
│    └─Softmax: 2-18                     [-1, 4]                   --
==========================================================================================
Total params: 2,191,556
Trainable params: 2,191,556
Non-trainable params: 0
Total mult-adds (M): 43.22
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 3.50
Params size (MB): 8.36
Estimated Total Size (MB): 11.88
==========================================================================================
```

Figure 3.2: CNN COVID-19 X-ray classifier architecture.

4
Training

After the network architectures have been defined the next most crucial step is to train the networks. In this chapter we talk about the extensive training carried out for the GAN and the Classifier. This involves defining an appropriate loss function, picking different epoch numbers and trying various learning rates for the optimizers.

4.1 GAN

Generative Adversarial Networks (GANs) are capable of producing extremely good results, but training a GAN is quite challenging. The fact that both the sub-networks involved, Generator and Discriminator, are trained simultaneously and training of one affects the other. It is often the case that improvements to one sub-network come at the expense of the other sub-network.

4.1.1 Batch Size

Generally it is recommended to not opt for a large batch size for training a GAN. The reason being that in the initial phases of training the discriminator gets a lot of examples to train on because of a larger batch size which can overpower the generator. This can lead to an unstable network and the network as a whole might not converge. Considering the size of the dataset we chose to train our model on we took a small batch size of 32. The batch size was quite suitable for our case and we observed that both the discriminator as well as the generator could train with stability.

4.1.2 Transforms of Training Samples

Thanks to *torchvision.transforms* we could chain together a number of image transforms before feeding the training samples to the network for training. Below we list the transforms used:

- **Resize:** This helps to resize the input image to the given size. Since our GAN takes images of size 64 x 64 we resize the input images to the desired image size before providing to the network.

- **Random horizontal flip:** This flips the given image horizontally.

- **Grayscale:** We make use of this function to convert the input image to a single channel image which can be later fed to the network.

- **Normalize:** Using this function we normalize the given image to have unit mean as well as unit standard deviation.

4.1.3 Hyperparameters

Four separate DCGANs were trained to synthesize chest X-ray images for each of the four classes of chest X-ray images namely, covid, normal, pneumonia bacteria, & pneumonia virus. The training process was done iteratively for the generator and the discriminator. As mentioned earlier we used mini-batches of m=32 chest x-ray examples $x_l^{(1)}, ..., x_l^{(m)}$ for each x-ray type l $\in$ (Covid, Normal, Pneumonia bacteria, Pneumonia virus) and $m = 32$ noise samples $z^{(1)}, ..., z^{(m)}$ drawn from uniform distribution between [-1, 1]. In the Leaky ReLU (Rectified Linear Unit), the slope of the leak was set to leak = 0.2. Weights were initialized to a zero-centered normal distribution with standard deviation of 0.02. We applied stochastic gradient descent with the Adam optimizer, an adaptive moment estimation that incorporates the first and second moments of the gradients, controlled by parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$ respectively. Learning rate of 0.0002 was kept for both the generator and the discriminator networks to start with. Later different combinations of learning rates for generator and discriminator networks were used to analyse the convergence of the overall GAN. Finally we increased the learning rate of the generator network to 0.002. The reason for the same is explained in the results section.

4.2 Classifier

CNN architectures for medical imaging usually contain fewer convolutional layers because of the small datasets and smaller input size, but in our particular case also the small set of classes. Differently from the paper we refer to, we decided to extend the training process until the epoch in which the CNN loss curve stabilizes around it's minimum axis. This in general depends on the size of the dataset, thus, because we created different lenght datasets, we established a max training epochs of 1500 on the longer datasets. In order to prevent overfitting and performance dropping, we decided to save, record and consider during experiments the results of the processes each 150 epochs.

4.2.1 Batch Size

The strategies we used operate in a small-batch regime where in the set of the training data, a subset, namely 256 elements, is randomly sampled to compute an approximation to the gradient. This is because it has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize. This and others batch size strategies have been tested and evaluated and this seems to be the best suited for the architecture and datasets chosen.

4.2.2 Transforms of Training Samples

In order to train the CNN and to carry out our experiments we had to create separate sets of training data, some for classical augmented and the others for synthetic augmented. We will talk about it in more detail in *Chapter* 6.

For the classic data augmentation, using again *torchvision.transforms* library, we choose to apply the following transformations:

- **Resize:** This helps to resize the input image to the given size. Since our CNN takes images of size 64 x 64 we resize the input images to the desired image size before providing to the network.

- **Color Jitter:** This randomly change the brightness, contrast and saturation of the image.

- **Random Rotation:** This randomly rotate the image in a chose range of degrees.

- **Random Horizontal Flip:** This flips the given image horizontally.

- **Random Affine:** This randomly apply translation and scaling to the image in the ranges chosen.

- **Grayscale:** We make use of this function to convert the input image to a single channel image which can be later fed to the network.

- **Normalize:** Using this function we normalize the given image to have unit mean as well as unit standard deviation.

(a) **(b)** **(c)**

Figure 4.1: 64 random elements batch from (a) original dataset, (b) classic augmented dataset, (c) synthetic augmented dataset

4.2.3 Hyperparameters

For training we used a batch size of 256. We used stochastic gradient descent optimization with high learning rate ($lr = 0.01$), which performed very good for this specific task, and Nesterov momentum updates ($m = 0.9$), where instead of evaluating the gradient at the current position we evaluated it at the "lookahead" position which improves the optimization and speed up the process. Separates CNN were trained on different augmented chest X-ray scan types for each of available datasets for variable epochs quantities depending on the dataset dimensions, between 150 epochs for small datasets to 750 epochs for larger ones.

5

Evaluation Metrics

5.1 GAN

In order to evaluate the performance of a GAN we record the following training statistics:

- **Loss_D :** This represents the discriminator loss calculated as the sum of losses for all the real and all the fake batches ($log(D(x)) + log(D(G(z))))$.

- **Loss_G :** It is the generator loss calculated as $log(D(G(z)))$.

- **D(x) :** It represents the average output of the discriminator for all the all real batch. This should start close to 1 then theoretically converge to 0.5 when G gets better.

- **D(G(z)) :** This is the average discriminator outputs for the all fake batch. The first number is before D is updated (D_G_z1) and the second number (D_G_z2) is after D is updated. These numbers should start near 0 and converge to 0.5 as G gets better.

When a discriminator is updated it tries to push $D(x)$ close to 1 and at the same time pushing $D(G(z))$ close to 0. Contrary to this an update of a generator tries to increase $D(G(z))$ that is, it tries to fool the discriminator to make it believe that the images generated from noise are the real ones. In an ideal case the discriminator cannot tell the difference between real and fake images. However, in practice this scenario is not easily achieved.

5.1.1 Convergence Failure

Convergence failure happens when the generator and discriminator do not reach a balance during training.

5.1.1.1 Discriminator Dominates

This scenario happens when the generator score reaches zero or near zero and the discriminator score reaches one or near one.
Figure 5.1 shows an example of the discriminator overpowering the generator.

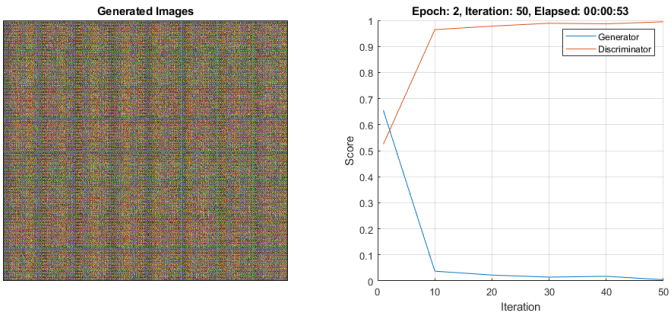


Figure 5.1: Discriminator overpowering the generator.

5.1.1.2 Generator Dominates

This scenario happens when the generator score reaches one or near one. Figure 5.2 shows an example of the generator overpowering the discriminator. Here the generator score goes to one for a many iterations. In this case, the generator learns how to fool the discriminator almost always.

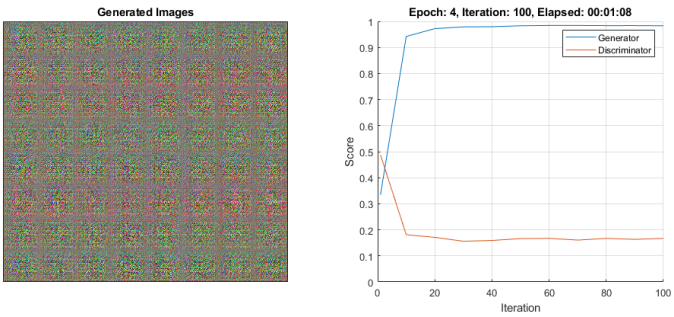


Figure 5.2: Generator overpowering the discriminator.

5.2 Classifier

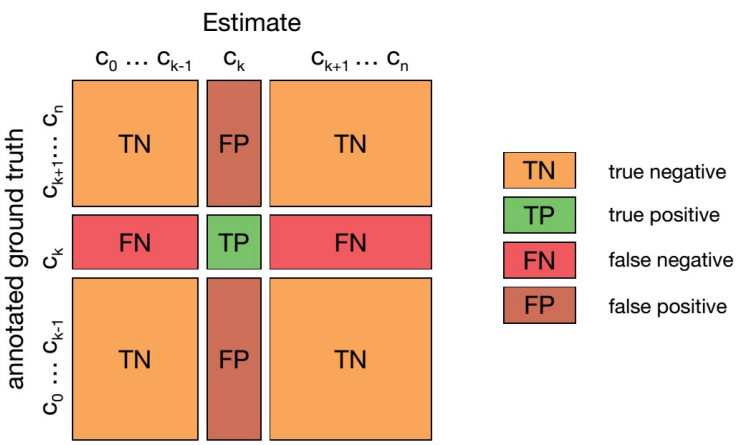


Figure 5.3: Multi-class confusion matrix notation and relative indices.

The number of examples in original test set was (9,9,9,9), thus we adopted a batch of 36 containing a balanced number of chest X-rays images for each class. To get a more precise estimation of the performances, we decided to average each test made over 500 iterations. We evaluated the classification performance using a classification total accuracy measure. Additionally, we calculated confusion matrices, sensitivity, specificity, accuracy and f1score measures for each lesion category. All these measures are presented in the following equations:

- *TotalAccuracy*: is the overall closeness of the classification to the ground truth.

$$TotalAccuracy = \frac{\sum TP}{NumberOfSamples} \tag{5.1}$$

- *Accuracy*: is closeness of the classification to the correct class.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

- *Sensitivity*: also known as *Recall*, measures the reliability of the prediction of a class (i.e. the percentage of COVID-19 affected who are correctly identified as having some illness) and is given by the correctly predicted samples over all the samples of that class.

$$Sensitivity = \frac{TP}{TP + FN} \tag{5.3}$$

- *Specificity*: or *Precision*, measures the proportion of false that are correctly identified (i.e. the percentage of healthy people who are correctly identified as not having COVID-19 and pneumonia).

$$Specificity = \frac{TN}{TN + FP} \tag{5.4}$$

- *F1Score*: is a measure of a test's accuracy. It is calculated from the sensitivity and the specificity.

$$F1Score = \frac{2 * Specificity * Sensitivity}{Specificity + Sensitivity} \tag{5.5}$$

Where for each category, positives (P) are examples from this category and negatives (N) are examples from the other three categories.

6
Results

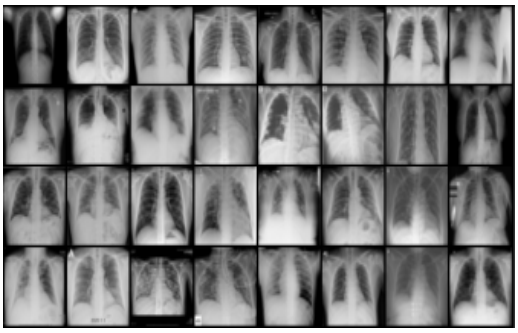
In this section we discuss about the various experiments conducted with different models that we trained for the completion of the task. Below in section:6.1 we analyse the results obtained while generating fake images from GAN for data augmentation, whereas in section:6.2 results are presented for the classification of the four classes of chest X-ray images present namely, covid, normal, pneumonia bacteria, & pneumonia virus. We also compare the CNN performance when classical data augmentation technique is followed and when augmentation is done using the fake images generated from the GAN.

6.1 Synthetic Images Generated from GAN

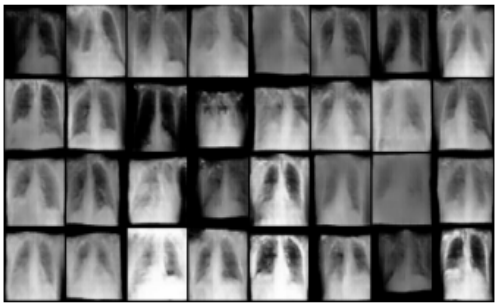
As discussed in section:2.2 we used COVID-19 chest X-ray dataset for the training of the GAN. For every trained network the relevant parameters namely, *Loss_D. Loss_G, D(x), D(G(z1)) & D(G(z2))* were recorder and plotted against the epoch number for analysis. This helped in tracking the stability of the generator as well the discriminator networks as the training epochs increases. Figure 6.1 represents a stack of real X-ray images coming from the dataset and fake X-ray images generated from GAN trained for each of the four classes, *covid, normal, pneumonia bacteria, & pneumonia virus*, under consideration.
The fake images shown in Figure 6.1 are singel channel images obtained from an individual GAN trained at 512 epochs for each of the four classes. In the sections below we provide a description of various experiments and analysis conducted while training GANs at different settings.

6.1.1 Comparison at different learning rates

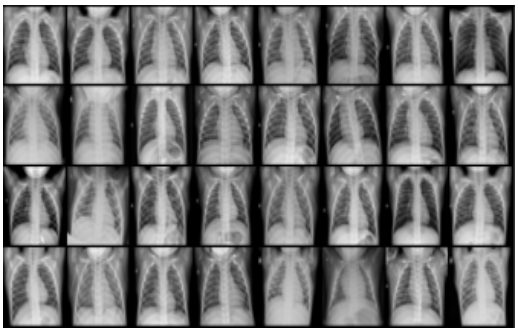
We trained four GANs, one for each of the four classes, *covid, normal, pneumonia bacteria, & pneumonia virus.* Initially at low learning rate of 0.0002 for the generator network resulted in poor quality of synthetic images. The major cause was the discriminator overpowering the generator network. The output synthetic images from the network were improved in quality when the learning rate was increased to 0.002. This helped the generator network to learn more repressively. As shown in Figure 6.2 for a GAN trained on *normal* X-ray images, the discriminator dominates the generator network at $lr = 0.0002$ whereas with $lr = 0.002$ the generator network converges as its loss decreases much more smoothly with the training epochs.



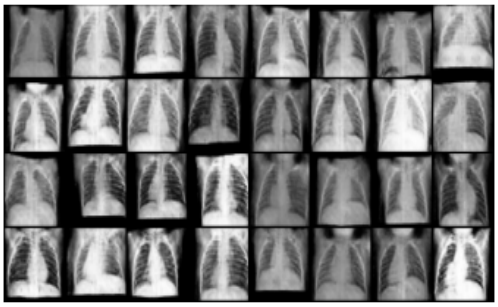
(a) covid real



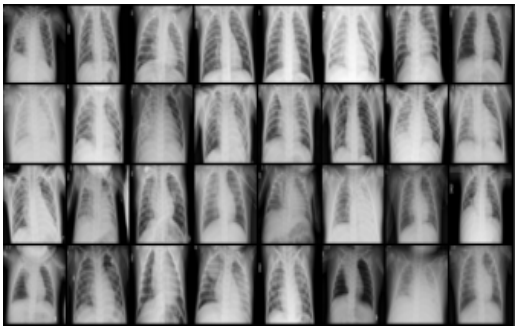
(b) covid fake



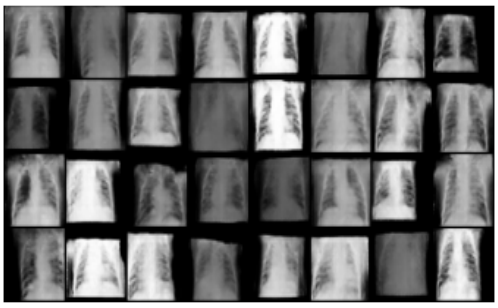
(c) normal real



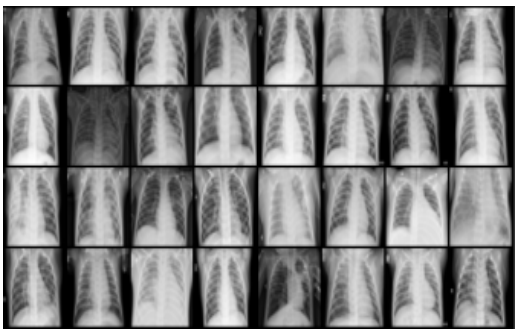
(d) normal fake



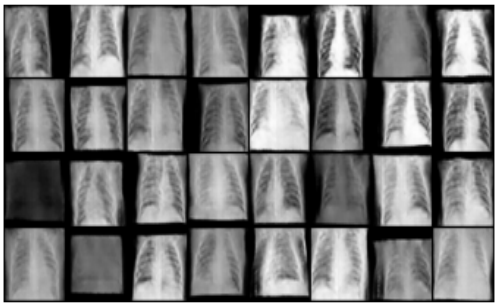
(e) pneumonia bacteria real



(f) pneumonia bacteria fake

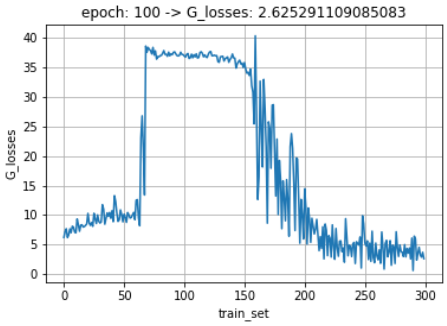


(g) pneumonia virus real

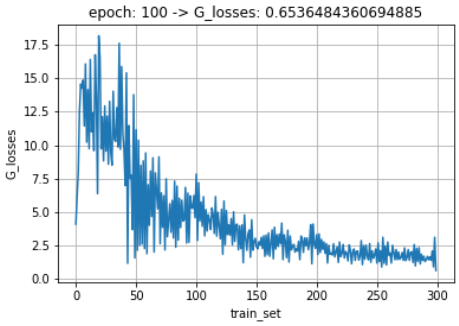


(h) pneumonia virus fake

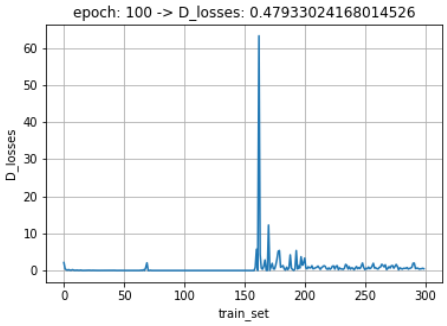
Figure 6.1: A pair of real (left) and fake X-ray images generated from the GAN (right). (a-b) for Covid ; (c-d) for Normal; (e-f) for Pneumonia bacteria; (g-h) for Pneumonia virus.



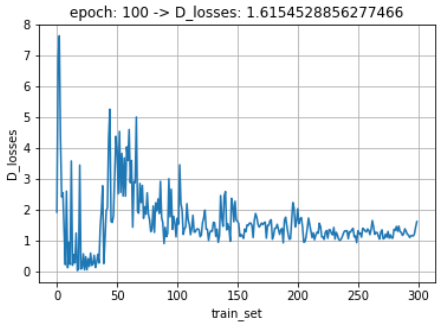
(a) G_loss
(lr_g=0.0002)



(d) G_loss
(lr_g=0.002)

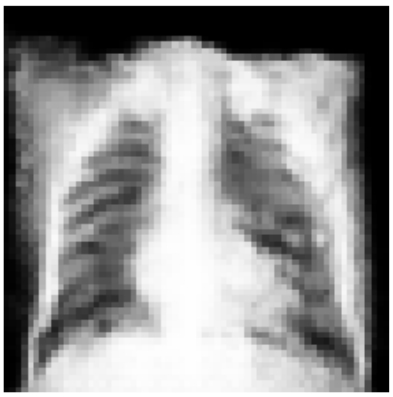


(b) D_loss
(lr_g=0.0002)



(e) D_loss
(lr_g=0.002)

(c) Generated Image
(lr_g=0.0002)



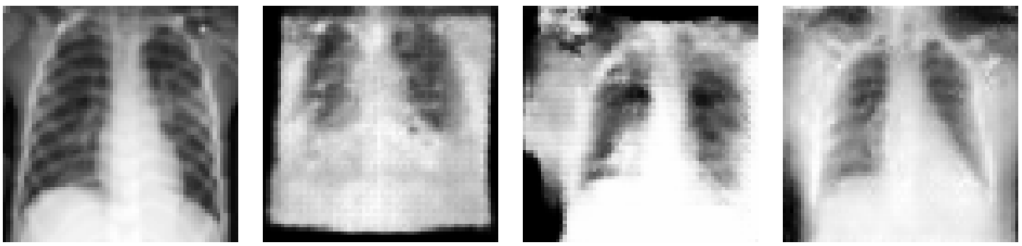
(g) Generated Image
(lr_g=0.002)

Figure 6.2: Graph of generator loss vs epoch (top) , discriminator loss vs epoch (center) and the stack of fake images (bottom). (a-c) for generator network with lr = 0.0002; (d-f) for generator network with lr = 0.002

6.1.2 Comparison at different epochs

After the analysis that a generator network trained at learning rate equal to 0.002 performs better than other models, we did an evaluation of training GAN at different epochs. We trained a GAN on *pneumonia virus* X-ray images at three different values of epochs, 128, 256 & 512, for comparison. Figure:6.3 displays the generated fake

images from the three networks. It can be seen that the quality of images increased as the epochs are increased. With the increasing epochs the details contained in the generated images increases that also suggests the increasing ability of the generator network to produce fake images that are close to real images.



(a) (b) (c) (d)

Figure 6.3: Comparison at different epochs (a) Original pneumonia virus X-ray image; (b) Generated image at 50 epoch; (c) Generated image at 100 epoch; (d) Generated image at 200 epoch

6.1.3 Comparison at different channels of the training images

Next, we compare the results of two different GANs both trained on *pneumonia bacteria* X-ray images and for 200 epochs. The first GAN is trained on single channel X-ray images and the second is trained on 3 channel images. It was observed that the former GAN performed better. Figure 6.4 illustrates that the GAN trained on single channel images results in *D_x, D_G_z1 & D_G_z2* attaining values close to an ideal GAN, that is 0.5. In comparison the GAN trained on 3 channel images results in *D_x* which is close to 1 and *D_G_z1 & D_G_z2* close to 0. This suggests the GAN trained on single channel images was quite stable as well as balanced. On the other hand the GAN trained on 3 channel images was not balance as the discriminator overpowers the generator network. This leads to the generator network not able to produce 'real' fake images.

6.1.3.1 Consistency in results w.r.t. those reported in the original paper[1]

From the discussion above it is evident that the synthetic images we obtained from DCGAN are quite high in quality. This can help to improve the CNN performance in chest X-ray classification. The following results presented in the next section corroborate the claim. Thus the experiments and the results obtained during our study are coherent with that reported by authors in the original paper.

[1]GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classification

6.2 Classification

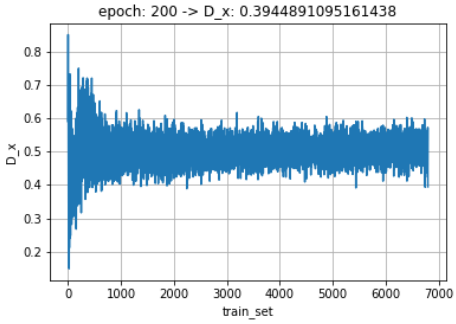
The flowchart of the classification experiment we conducted, similiar to the paper we refer to, is presented in the *Figure* 6.5. We started by examinating the effects that classic data augmentation produces on pneumonia infection classification task. We then synthesized pneumonia infection chest X-rays scans using GAN and examined the classification results after adding the synthesized lesion X-rays scans to the training set. A detailed description of each step is provided next.

- *Classic Augmented Datasets*: We recorded the classification results for the pneumonia infection classification CNN for increasing amounts of data augmentation over the original training set. We trained the network and evaluated the results separately for each set of data images (that included the original images and additional classic augmented images), as follows: Let $\{D_{classic}\}_{i=1}^{5}$ be the data groups that include increasing amounts of augmented examples for each training. During testing time, we used the same data examples for all evaluations. In order to examine the effect of adding increasing numbers of examples, we formed the data groups additively such that $D_{classic}^{1} \subset D_{classic}^{2} \subset \cdots \subset D_{classic}^{4}$. The first data group was only made up of the original COVID-19 trainset. For each original X-ray scan, we produced a large number of augmentations ($N_{rot} = 2$, $N_{color} = 2$, $N_{flip} = 2$, $N_{trans} = 4$ and $N_{scale} = 4$), resulting in $N = 128$ augmented images per X-ray scan and overall ~ 8000 examples per class. Then, we selected the images for the data groups by sampling randomly augmented examples such that for each original scan we sampled the same augmentation volume. To summarize the augmentation data group preparation process, the number of samples added to each fold (in our 4-folds) was $\{500, 1000, 2000, 3000\}$. The training process was conducted by cross-validation over 4 different folds.
- *Synthetic Augmentated Datasets*: The second step of the experiment consisted of generating synthetic pneumonia infection chest X-rays scans for data augmentation using GAN. We took the optimal point for the classic augmentation $D_{classic}^{optimal}$ and used this group of augmented data to train the GAN. Since our dataset was too small for effective training, we incorporated classic augmentation for the training process. We employed the DCGAN architecture to train each class separately, using the same data partition. After the generator had learned each class data distribution separately, it was able to synthesize new examples by using an input vector of normal distributed samples ("noise"). *Figure* 6.4 presents examples of synthesized pneumonia infection X-ray scans from each class. The same approach that was applied in step one of the experiment when constructing the data groups was also applied in step two: We collected large numbers of synthetic lesions for all four classes, and constructed data groups $\{D_{synthetic}\}_{j=1}^{4}$ of synthetic examples additively. To keep the classes balanced, we sampled the same number of synthetic scans for each class. To summarize the synthetic augmentation data group preparation process, the number of samples added to each fold (in our 4-folds) was $\{100, 500, 1000, 2000\}$.

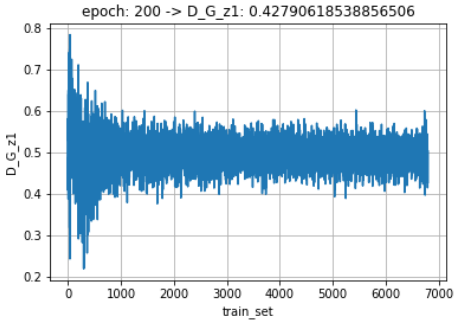
Results of the GAN-based synthetic augmentation experiment are shown in *Figure*

6.13 . The baseline results (classical augmentation) are shown in red. We see the total accuracy results for the pneumonia infection classification task, for each group of data. All test results are averaged on 500 repetitions of CNN predictions.
When no augmentations were applied, a result of 76.9% was achieved; this may be due to overfitting over the small number of training examples (60/70 samples per class), the confusion matrix for this point appears in *Figure* 6.6. The results improved as the number of training examples increased, up to saturation around 82.7% where adding more augmented data examples failed to improve the classification results. We note that the saturation starts with $D^3_{classic} = 2000$ samples per fold. We define this point as $i = optimal$ where the smallest number of augmented samples were used. The confusion matrix for the optimal point appears in *Figure* 6.9. The blue line in *Figure* 6.13 shows the total accuracy results for the pneumonia infection classification task for the synthetic data augmentation scenario. The classification results improved from 82.7% with no synthesized lesions to 90.7% for $D^{optimal}_{classic} + D^3_{synthetic} = 2000 + 1000 = 3000$ samples per class. The confusion matrix for the best classification results using synthetic data augmentation is presented in *Figure* 6.13 which shows these results to be consistent with results in the paper we refer to.
Legend:

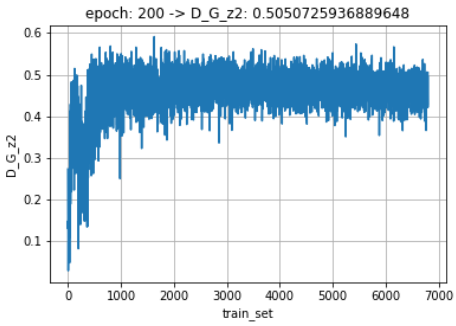
- **O**: *Default trainset*, no data augmentation, set $\{O1\}$ correspond to $\sim \{65\}$ per class.
- **C**: *Classic augmented trainset*, sets $\{C1, C2, C3, C4\}$ corresponds to $\{500, 1000, 2000, 3000\}$ samples per class respectively.
- **G**: *Synthetic augmented trainset*, sets $\{G1, G2, G3, G4\}$ corresponds to $\{100, 500, 1000, 2000\}$ samples per class respectively.



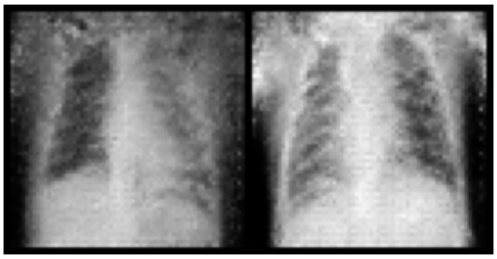
(a) D_x (1-C)



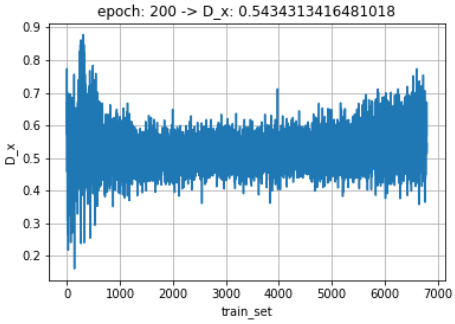
(b) D_G_z1 (1-C)



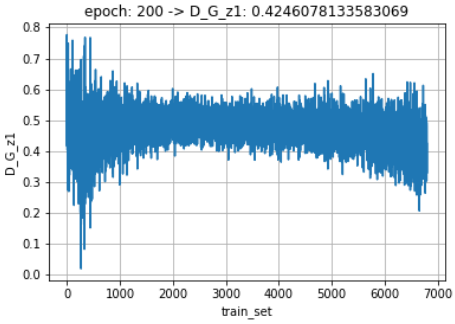
(c) D_G_z2 (1-C)



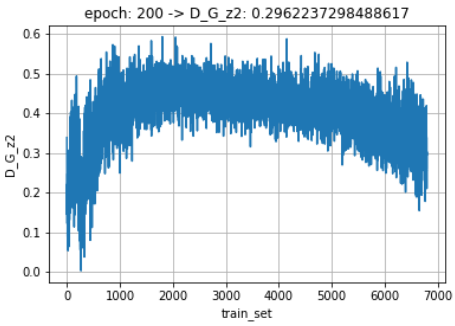
(d) Generated images (1-C)



(e) D_x (3-C)



(f) D_G_z1 (3-C)



(g) D_G_z2 (3-C)

(h) Generated images (3-C)

Figure 6.4: Graph of D_x vs epoch (top), D_G_z1 vs epoch (center1), D_G_z2 vs epoch (center2), & generated images (bottom) for GAN trained on pneumonia bacteria X-ray images. (a-d) 1-channel (1-C) images; (e-h) 3-channel (3-C) images

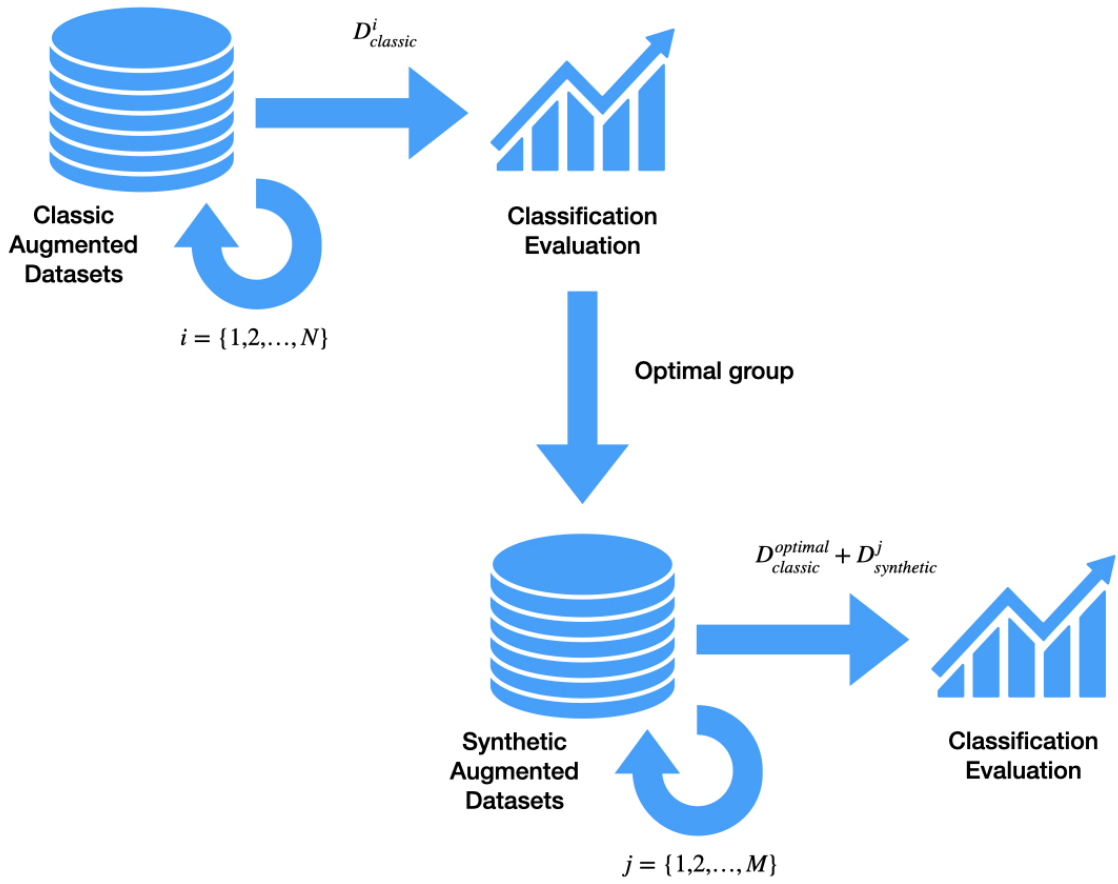
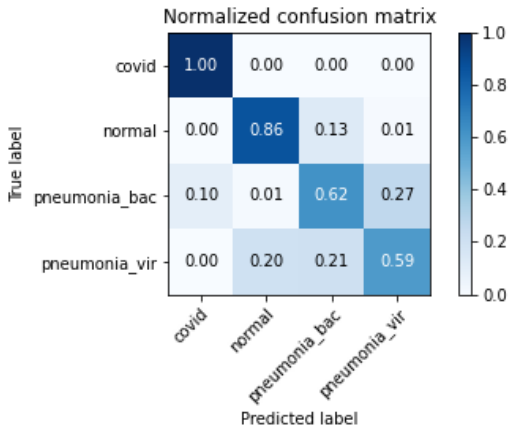


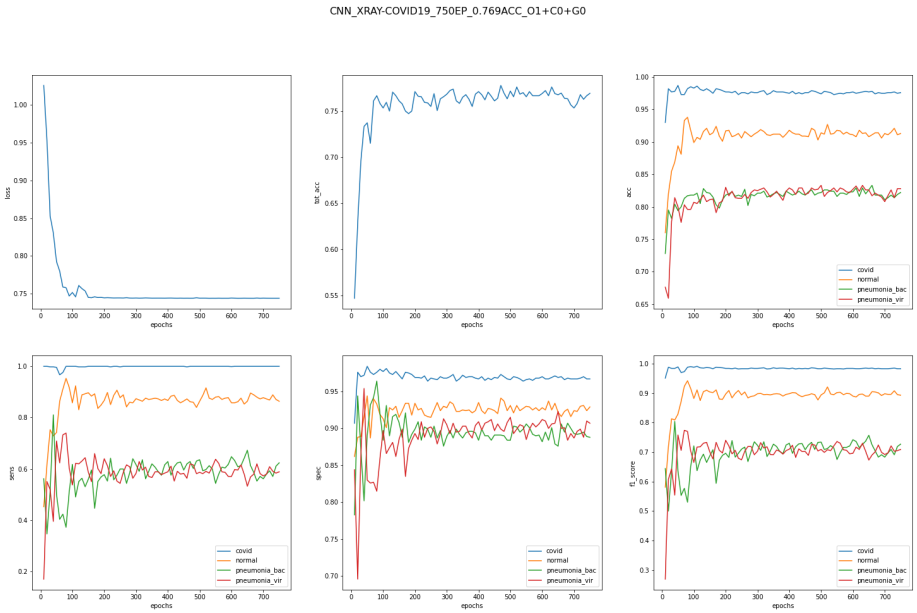
Figure 6.5: Experiment flowchart for evaluating synthetic data augmentation in the task of classifying COVID-19 and pneumonia infection.

4-fold_750_ep_0.769_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	0.976	1	0.967	0.983
normal	0.913	0.864	0.929	0.893
pneumonia_bac	0.822	0.624	0.888	0.727
pneumonia_vir	0.828	0.589	0.907	0.709

(a) Performance results



(b) Confusion matrix

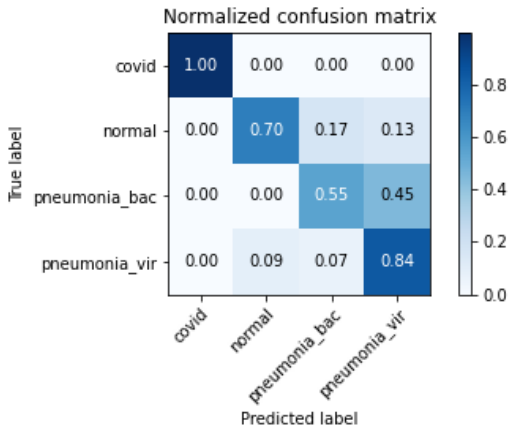


(c) Training history

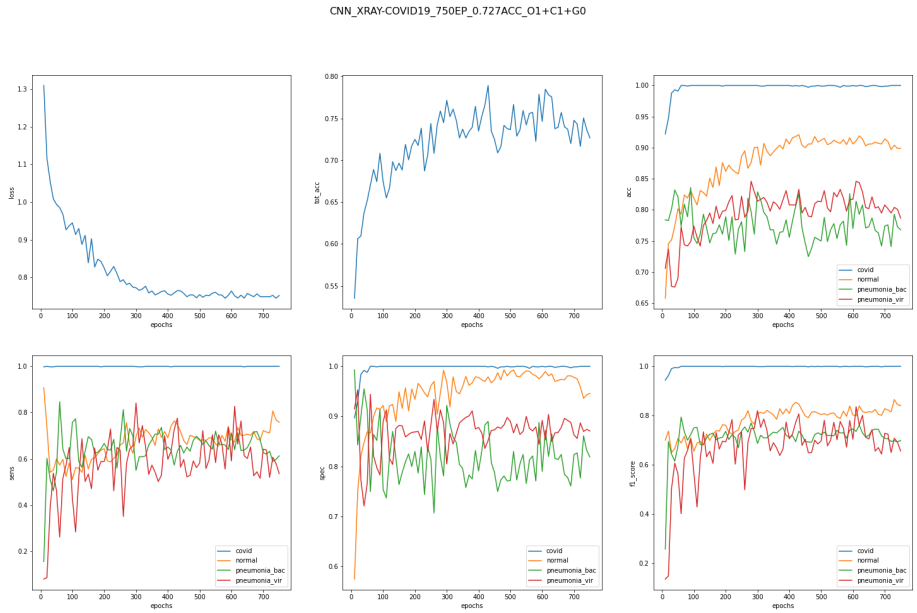
Figure 6.6: CNN trained on default trainset (O1+C0+G0), 750 epochs, 0.769 total accuracy.

4-fold_300_ep_0.772_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	0.999	1	0.999
normal	0.901	0.697	0.969	0.809
pneumonia_bac	0.829	0.55	0.922	0.687
pneumonia_vir	0.814	0.841	0.805	0.82

(a) Performance results



(b) Confusion matrix

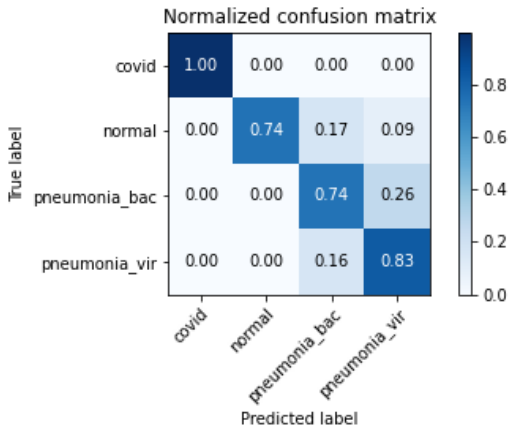


(c) Training history

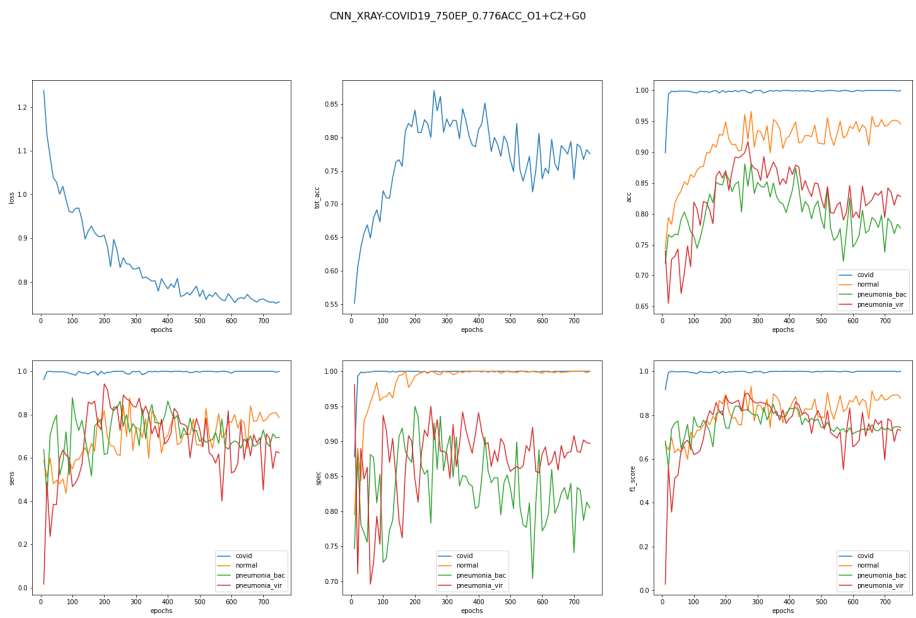
Figure 6.7: CNN trained on classic augmented trainset (O1+C1+G0), 750 epochs, 0.772 total accuracy at 300 epochs.

4-fold_300_ep_0.828_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	0.998	1	0.999
normal	0.935	0.741	0.999	0.847
pneumonia_bac	0.851	0.737	0.889	0.802
pneumonia_vir	0.871	0.834	0.883	0.855

(a) Performance results



(b) Confusion matrix

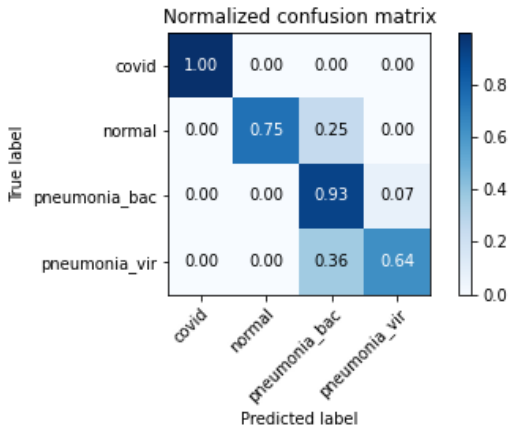


(c) Training history

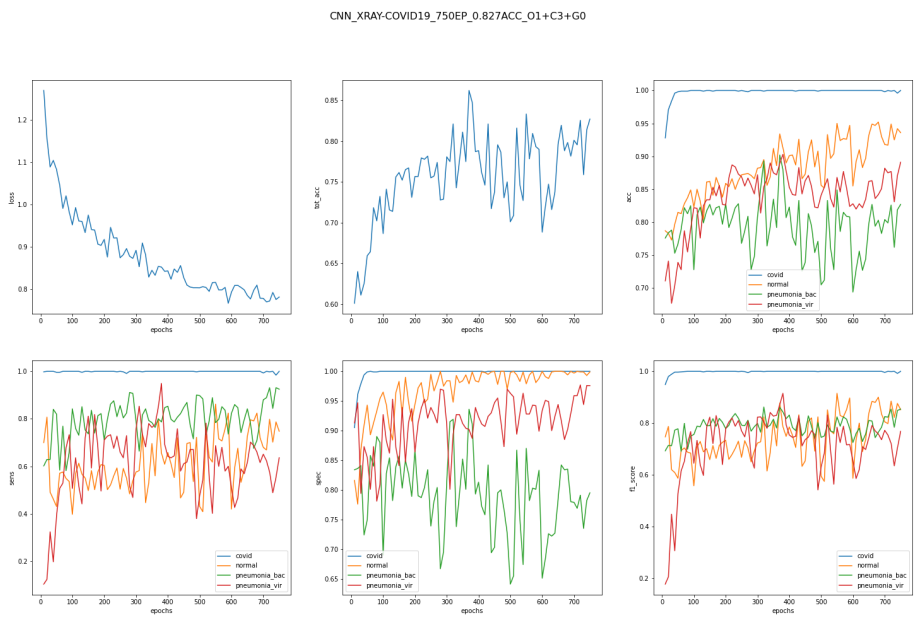
Figure 6.8: CNN trained on classic augmented trainset (O1+C2+G0), 750 epochs, 0.828 total accuracy at 300 epochs.

4-fold_750_ep_0.827_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	1	1	1
normal	0.936	0.748	0.999	0.852
pneumonia_bac	0.827	0.925	0.795	0.854
pneumonia_vir	0.891	0.636	0.976	0.768

(a) Performance results



(b) Confusion matrix

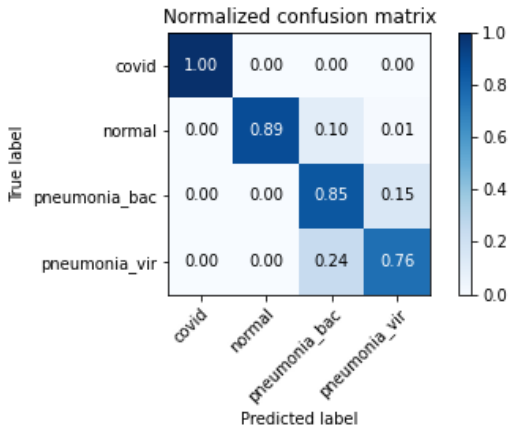


(c) Training history

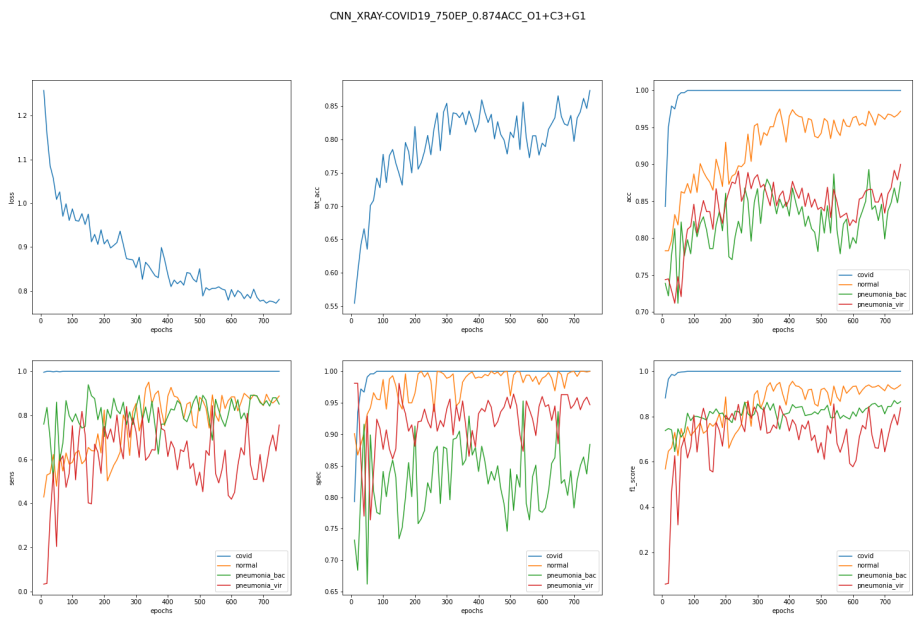
Figure 6.9: CNN trained on classic augmented trainset (O1+C3+G0), 750 epochs, 0.827 total accuracy at 750 epochs.

4-fold_750_ep_0.874_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	1	1	1
normal	0.972	0.888	1	0.94
pneumonia_bac	0.876	0.851	0.884	0.866
pneumonia_vir	0.9	0.756	0.947	0.839

(a) Performance results



(b) Confusion matrix

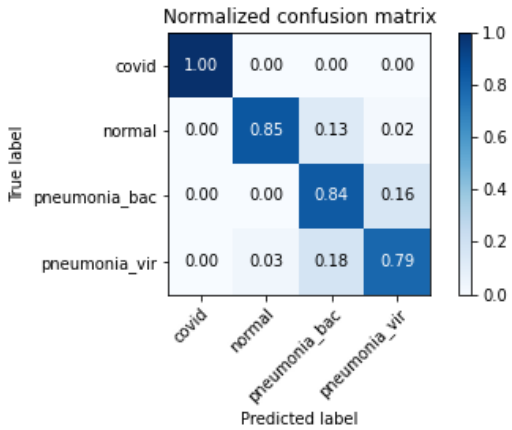


(c) Training history

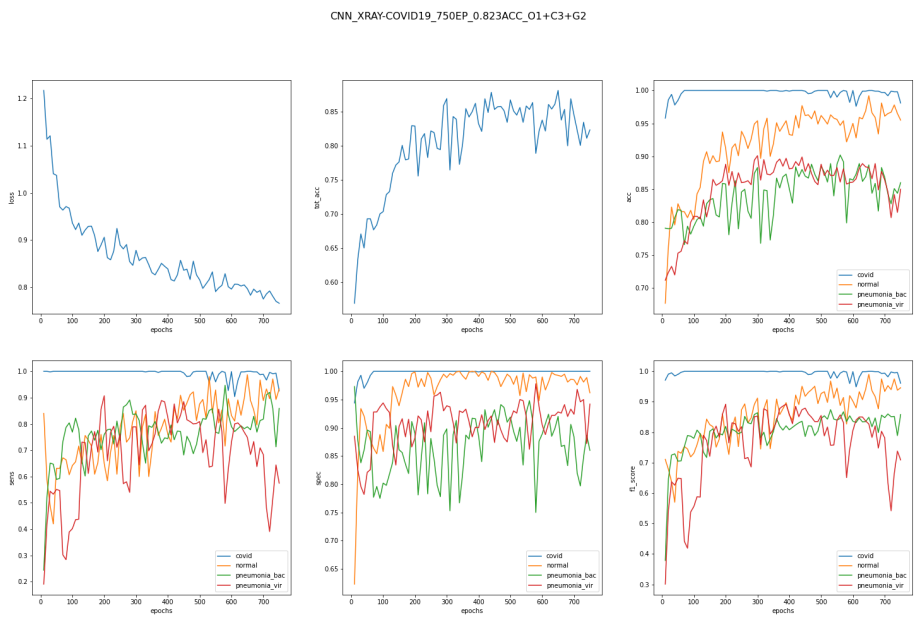
Figure 6.10: CNN trained on both classic and synthetic augmented trainset (O1+C3+G1), 750 epochs, 0.874 total accuracy at 750 epochs.

4-fold_300_ep_0.869_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	1	1	1
normal	0.954	0.85	0.989	0.912
pneumonia_bac	0.883	0.838	0.898	0.864
pneumonia_vir	0.901	0.789	0.939	0.853

(a) Performance results



(b) Confusion matrix

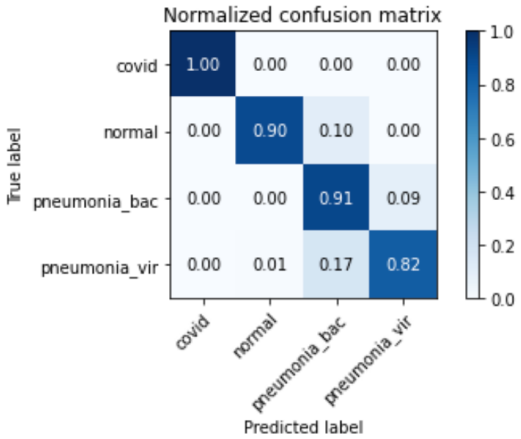


(c) Training history

Figure 6.11: CNN trained on both classic and synthetic augmented trainset (O1+C3+G2), 750 epochs, 0.869 total accuracy at 300 epochs.

4-fold_720_ep_0.907_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	1	1	1
normal	0.972	0.896	0.997	0.943
pneumonia_bac	0.908	0.907	0.909	0.907
pneumonia_vir	0.933	0.824	0.97	0.888

(a) Performance results



(b) Confusion matrix

(c) Training history

Figure 6.12: CNN trained on both classic and synthetic augmented trainset (O1+C3+G3), 750 epochs, 0.907 total accuracy at 720 epochs.

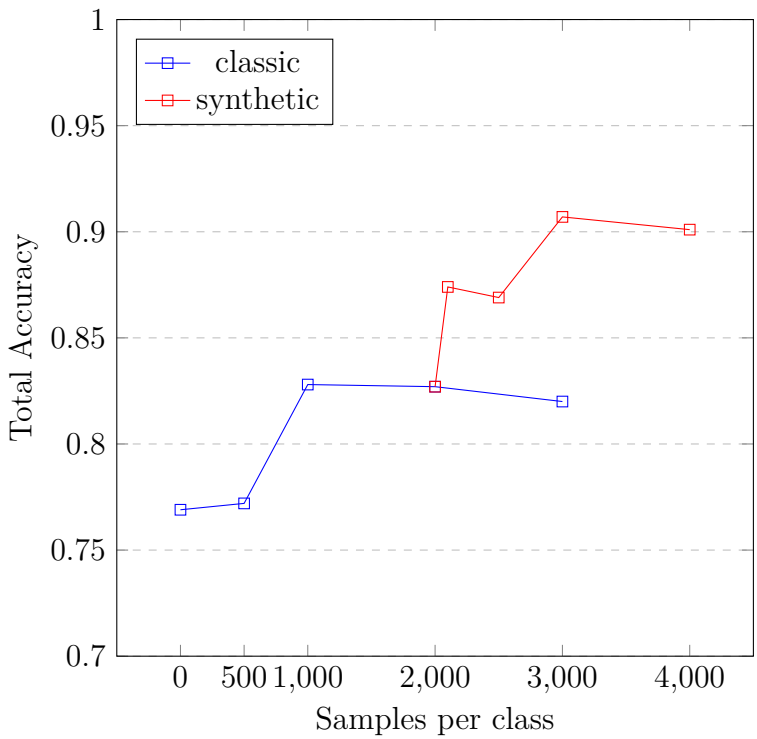


Figure 6.13: Total accuracy results for pneumonia infection classification of covid, normal, pneumonia bacteria and pneumonia virus with the increase of training set size. The blue line shows the effect of adding classic data augmentation and the red line shows the effect of adding synthetic data augmentation.

4-fold_750_ep_0.827_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	1	1	1
normal	0.936	0.748	0.999	0.852
pneumonia_bac	0.827	0.925	0.795	0.854
pneumonia_vir	0.891	0.636	0.976	0.768

(a) Classic augmented trainset, 750 epochs, 0.827 total accuracy.

4-fold_720_ep_0.907_acc	Accuracy	Sensitivity	Specificity	F1Score
covid	1	1	1	1
normal	0.972	0.896	0.997	0.943
pneumonia_bac	0.908	0.907	0.909	0.907
pneumonia_vir	0.933	0.824	0.97	0.888

(b) Synthetic augmented trainset, 720 epochs, 0.907 total accuracy.

Figure 6.14: CNN predictions improvement given by GAN data augmentation.

7
Conclusion

In this project, a GAN learning for COVID-19 detection in limited chest X-ray images is presented. The lack of benchmark datasets for COVID-19 especially in chest X-rays images was the main motivation of this project. The main idea is to collect all the possible images for COVID-19 and use the GAN network to generate more images to help in the detection of the virus from the available X-ray's images.

The number of images of the collected dataset was 200 images for three types of classes. The classes are the covid, normal, and pneumonia bacterial. The task of classification and detection of the three classes is quite challenging in the absence of a large dataset. It was shown how the generated synthetic images could be used to augment the original dataset and eventually lead to a higher classification and detection accuracy of the chest X-ray images.

We would like to further investigate generative adversarial architectures that generate multi-class samples together. Also an extension of the presented work in domains different from medical tasks could be a priority to investigate in future.