# Diagnosis of Pneumonia using Chest X-Ray Images based on Deep Learning Models

**Prashant Uttam Wakchaure**
A1 - Deep Learning Project
Student No. 20200126
prashant.wakchaure@ucdconnect.ie

## Abstract

This paper demonstrates the task of classifying Chest X-Ray images as either *Normal* or having *Pneumonia*. This task is avidly famous on the Kaggle platform as Chest X-Ray Images (Pneumonia). The manuscript proposes 2 Deep Learning Architectural Neural Network Models, one; is a CNN Model, which I call the X-Ray CNN; which is tuned and trained from scratch, while the other is the famous ResNet50 Transfer Learning Model. Even though the ResNet50 has exceptional results on the unseen data, the X-Ray CNN model which I trained is giving similar competing evaluation metrics. There are nearly 1000+ notebooks on Kaggle, but hardly any of them are appealing as this dataset has inconsistencies and noise in the images, also the dataset split is very biased and imbalanced. With out-of-the box data augmentations, device loaders and weight balancing techniques, I have managed to achieve an AUC-ROC score of 0.80-0.90. Taking just the validation loss and AUC-ROC into account for major evaluation for this dataset will be hardly arguable, as we are indeed getting less than 0.1 validation loss with the transfer learning model used, but the results on the unseen images satisfy the aim of recognizing correct predictions of having Pneumonia and less False Negative Error Rate of just 1.28%, which is an exceptional result for a model in the medical domain. I have used PyTorch DL Library for this comprehensive task.

## 1 Introduction

Due to elemental limitations in identifying the interstitial patterns in the chest X-Ray technique, high numbers of false-negative cases are possible and thus the intervention of state of the art - neural network model to identify the patterns is very essential in the medical domain and is also gaining more popularity[1]. Pneumonia is the second leading reason for more no. of hospitalizations. The mortality rate is high, especially in the elderly[2]. We all are aware of the fact that since the inception of the COVID-19 outbreak and its varying mutations which are still ongoing and evolving, one of the prime factors to determine the positivity of the virus is by analyzing the X-Ray images. Hence, we cannot afford to have a large no. of false-negative cases because of medical errors or human errors. Also, the X-Ray Images are somewhat sufficient in many such evident cases where the virus is predicted even before conducting the RT-PCR tests, which consumes a lot of time[3].

For this project, the dataset has predefined train, validation and test sets. But as there are only 16 images in the validation set, I also implemented the models over combining the train and validation set and re-splitting them accordingly. So, I have implemented the models 2 times each to have a look at the learning trends of the model and the final AUC-ROC score along with the confusion matrix metrics. With that said, my model is giving befitting results, although less than but similar to the competent ResNet50 which has a dynamic 50-layer network which learns residuals at the end of its layers. We'll have a look at this in depth later ahead in Section 3.4.

## 2 Related Work

For implementing such a comprehensive project, I needed to gather essential and potential information related to the previous implementations, innovations, improvements and modelling done by researchers and manuscript holders in the past in the same domain, and I found out that there are multiple previous works done in the field of diagnosing a particular disease based on the X-Ray Images using state of the art Deep Learning models. I will discuss the same in this section.

Dingding Wang *et al.* proposed an efficient diagnostic algorithm which combines deep features and machine learning classification techniques to implement an end-to-end diagnostic model. They initially tested the model on 1102 chest X-ray images, in which the experimental results demonstrated that the diagnostic accuracy of Xception + SVM is as high as 99.33%[3]. Rajpurkar, Pranav *et al.* proposed an algorithm, CheXNet, which is a 121-layer CNN trained on ChestX-ray14; as of now it is the largest publicly available chest X-ray dataset, containing over 100,000 frontal-view X-ray images with 14 diseases[5].

Abdullahi Umar Ibrahim *et al.* proposes the use of a deep learning approach based on pretrained AlexNet model for the classification of COVID-19, non-COVID-19 viral pneumonia, bacterial pneumonia, and normal CXR scans obtained from different public databases, in all of which they achieved excellent sensitivity and specificity scores[6]. Dimpy Varshni; Kartik Thakral *et al.* proposed the functionality of pre-trained CNN models utilized as feature-extractors followed by different classifiers for the classification of abnormal and normal chest X-Rays. They also state that the statistical results obtained from their implementations demonstrates that pretrained CNN models employed along with supervised classifier algorithms can be very beneficial in analyzing chest X-ray images, specifically to detect Pneumonia[7].

Okeke Stephen , Mangal Sain *et al.* says that unlike other deep learning classification tasks with sufficient image repository, it is difficult to obtain a large amount of pneumonia dataset for this classification task; therefore, they deployed several data augmentation algorithms to improve the validation and classification accuracy of the CNN model and achieved remarkable validation accuracy[4]. With that said, my CNN model is somewhat inspired from this paper, although I have done heavy fine tuning and trial and errors of adding and removing Conv2d layers, BatchNorm2d and Dropout layers throughout the modelling. The only thing missing in their paper is the testing results, also they combined the validation and test dataset, which gave them more density in the validation set and the train set already has more number of images, whereas in my case I combined the train and valid sets and further splitted it, by doing this, I lost potential training data. However, I got significant results pertaining to the False Negative Error Rate.

Mohammad Farukh Hashmi, Satyarth Katiyar *et al.* proposed an efficient model for the detection of pneumonia, which could aid the radiologists in their decision making process. Their implementation is based on a weighted classifier, which combines the weighted predictions from the state-of-the-art deep learning models such as ResNet18, Xception, InceptionV3, DenseNet121, and MobileNetV3 in an optimal way[8]. Zhichao Lu, Ian Whalen *et al.* proposed an evolutionary algorithm for searching neural architectures under multiple objectives, such as classification performance and floating-point operations (FLOPs).This improves computational efficiency by carefully down-scaling the architectures during the search as well as reinforcing the patterns commonly shared among past successful architectures through Bayesian model learning[9].

Samir S. Yadav & Shivajirao M. Jadhav proposed a linear support vector machine classifier with local rotation and orientation free features, transfer learning on two CNNs, VGG16 and InceptionV3, and a capsule network training from scratch[10]. They also talk about how transfer learning emphasizes on retraining specific features on a new target dataset to improve performance & how it is a proper network complexity that matches the scale of the given dataset, with which I totally agree and that's the reason why we should not shy upon using these transfer learning models once we have laid off with our own from scratch model.

The most cumbersome task which I faced was the hyperparameter tuning of the model & how I can implement a single epoch in as less time as possible, along with the proper functioning of the python code which uses PyTorch, which is an ML library used for extensive Deep Learning modelling tasks; which is more complex and non-direct compared to Keras and Tensorflow. The other hurdle which I faced was in realizing that the X-Ray images have a lot of noise in them[20], making the normal and pneumonia images look alike, which is evident by the fluctuations in the val loss.

## 3 Experimental Setup

For this task, I have used PyTorch v1.7.1 Deep Learning library. I chose Pytorch because of its quick results and good debugging capabilities. However, one of its potential issue is with reproducibility[21], which is not at all guaranteed by PyTorch. In fact, setting `cudnn.deterministic` as true will only gives us the same results on the CPU or the GPU on the same system when feeding the same inputs. But it still cannot guarantee same results on different machines[21]. Their manual seed is such that we have to use it every time before a line of random shuffle or weight initialization.

There are some potential considerations which I want to lay out before starting with the explanation of the setup. As the class is imbalanced, we will take the test AUC-ROC score into consideration rather than the test accuracy. Also, the False Negative rate of this model is very important as this is a medical domain, we need as much low FNR as possible. The validation set is so biased that I have seen validation accuracy which is divisible just by 5, this is just because of the 16 images. But when we combine the train and validation set and re-split, we get unbiased evaluation metrics. I am not comparing the with and without re-splitting nature, whereas I'm just demonstrating the nature of val loss graph, wherein we can see how much noise is in the val loss graph when we use the validation set as it is. I have tried to made the code as modular, reproducible and fast as possible, by defining various .py script which can be useful for any other image classification task in PyTorch along with the ease of just running the 2 notebooks to see the implementation and the testing results which will be discussed further in Section 3.5.

### 3.1 Dataset

The dataset contains anterior-posterior Chest X-Ray Images and it is from Kaggle . This dataset is quite popular and has over 1000+ notebooks. It contains predefined train, validation and test sets. The train set contains 3875 Pneumonia and 1341 Normal Images, while the Test set contains 390 Pneumonia and 234 Normal Images, on the other hand the Validation set contains just 8 images each for Normal and Pneumonia. The prediction is evaluated by AUC-ROC score, Weighted F1 score and Low False Negative Rate. After having a look at most of the notebooks on Kaggle, I noticed that there are many notebooks which have used Keras/Tensorflow quite efficiently and wholesomely, but there are a very few appealing PyTorch notebooks which have evident non-biased results and have conspicuously addressed the issues of noise in the images. Some discussions on Kaggle, also say that the Testing set may have mislabelled disease prediction[20].

### 3.2 Preprocessing & Image Augmentations

Table 1: Applied Augmentations on the Train set.

| **Transform** | **Setting** |
|---|---|
| Resize | 255 |
| CenterCrop | 224 |
| RandomHorizontalFlip | 0.5 |
| RandomRotation | 10 |
| RandomGrayscale | 0.1 |
| RandomAffine | translate (0.05,0.05) |

As all the sets contain different image sizes, it was essential to resize the images, so I resized them to 255 and center cropped to 224. The Training images were applied some augmentations; which can be seen in the above Table 1. I invested a hefty amount of time doing these augmentations and finally reaching to this setting. These augmentations are done in order to introduce diversity of data to the training model without the need of over or under sampling also to avoid minimal overfitting. One thing to note here is that this does not increase the number of images for training. Flipping, Grayscaling, translating are some of the augmentations which I did. I also tried normalizing the images using mean and standard deviation, but the performance was decreasing, so I did not use it. The RandomHorizontalFlip flips the image horizontally by a probability of 0.5, in the similar way the RandomRotation rotates the image with 10 degrees & further translating it by 0.05,0.05.

### 3.3 Model Architecture

Table 2: X-Ray CNN Model Architecture

| Layer (type) | Output Shape | Param |
|---|---|---|
| Conv2d-1 | [-1, 32, 224, 224] | 896 |
| ReLU-2 | [-1, 32, 224, 224] | 0 |
| MaxPool2d-3 | [-1, 32, 112, 112] | 0 |
| Conv2d-4 | [-1, 64, 112, 112] | 18,496 |
| ReLU-5 | [-1, 64, 112, 112] | 0 |
| MaxPool2d-6 | [-1, 64, 56, 56] | 0 |
| Conv2d-7 | [-1, 128, 56, 56] | 73,728 |
| ReLU-8 | [-1, 128, 56, 56] | 0 |
| MaxPool2d-9 | [-1, 128, 28, 28] | 0 |
| Conv2d-10 | [-1, 256, 28, 28] | 294,912 |
| BatchNorm2d-11 | [-1, 256, 28, 28] | 512 |
| ReLU-12 | [-1, 256, 28, 28] | 0 |
| MaxPool2d-13 | [-1, 256, 14, 14] | 0 |
| Conv2d-14 | [-1, 512, 14, 14] | 1,179,648 |
| BatchNorm2d-15 | [-1, 512, 14, 14] | 1,024 |
| ReLU-16 | [-1, 512, 14, 14] | 0 |
| MaxPool2d-17 | [-1, 512, 7, 7] | 0 |
| Flatten-18 | [-1, 25088] | 0 |
| Dropout-19 | [-1, 25088] | 0 |
| Linear-20 | [-1, 512] | 12,845,568 |
| ReLU-21 | [-1, 512] | 0 |
| Dropout-22 | [-1, 512] | 0 |
| Linear-23 | [-1, 1024] | 525,312 |
| ReLU-24 | [-1, 1024] | 0 |
| Dropout-25 | [-1, 1024] | 0 |
| Linear-26 | [-1, 2] | 2,050 |

The Table 2 demonstrates the architecture of the proposed model which is nothing but what I have named as X-Ray CNN. I have used 5 Convolutional Layers with ReLU activation function; I have used ReLU because it avoids the exponential growth in the computation required to operate the neural network[11]. I have included 2 `BatchNorm2d` Layers after the 4th and 5th `Conv2d` layers to avoid overfitting and minimal normalization of the images during training. There is a big argument over the placement of `BatchNorm2d` layer on various forums [22], wherein Piotr Bialecki - PyTorch Software Engineer at Nvdia, encourages that we can place the BatchNorm2d according to the performance results of the model [22]. I did not use Transform.normalize for Data Augmentation as I used `BatchNorm2d`. I also used Max-Pooling layer after each of the ReLU activations. I used maxpooling layers to reduce the dimensionality of the input images, we can also see the same from Table 2 above that the size of the image is decreasing by a factor of 2 in accordance to the Max-Pooling layer used. I also used a dropout after the flattened layer and each dense layers to avoid overfitting of the model[11], I was very specific because I tried multiple permuatations and combinations. In the end, we flatten the output from the convolution networks to generate one hightened feature vector which is used by the classifier which is a Dense Layer. So, the classification layer contains a flattened layer, along with 3 dense layers and a ReLU Activation function between them.

### 3.4 Baseline Approach and Transfer Learning

The above 22-layer CNN model is somewhat giving similar results when we compare it with the State of the Art ResNet50 Model, which is evident from the metrics in the Section 4. Before coming up with this model architecture, I tried using a basic CNN Baseline Model without any Drop-Out or Activation Layers, and I got an accuracy of just 23.78%. From this initial stage, I started tuning the model and came up with the X-Ray CNN Model. Furthermore, I also implemented a Transfer Learning model, which is none other than the ResNet50 model. The residual network is just like any other CNN network architecture, except that they use a residual block[12]. This adds the original

input back to the output feature map obtained by passing the input through one or more convolutional layers, this also allows deeper networks to work better since ResNet50 uses skip-connections blocks. I also freezed the starting layers of the network (except the fully connected layer) and replaced the last layer (fc) with 2 classes. The overall architecture of the model can be seen in Figure 1.

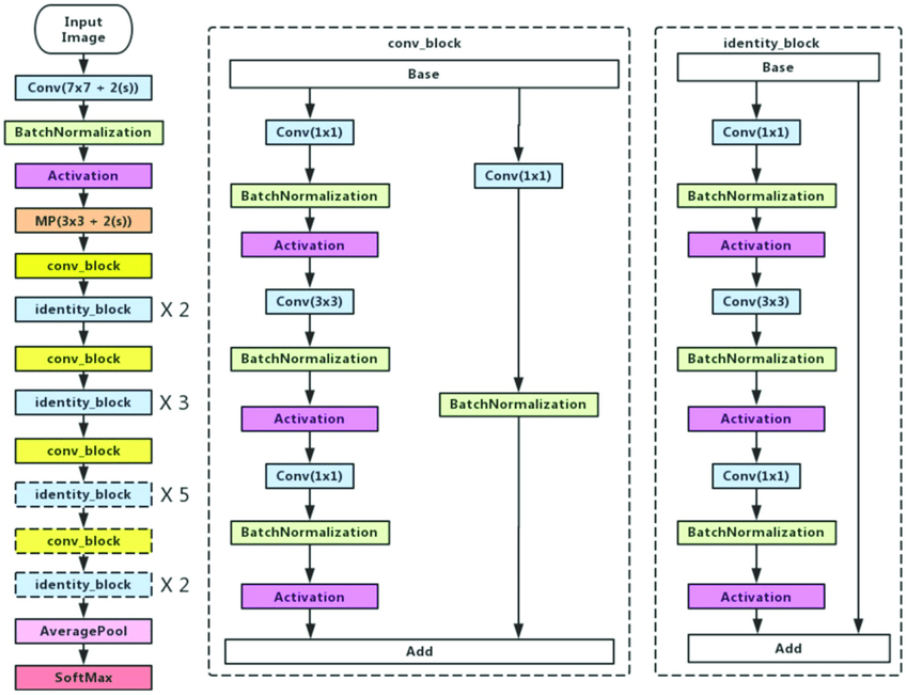



Figure 1: ResNet50 Architecture [13]

### 3.5 Modularity and Reproducibility

My approach for writing the code was very clean, modular, and fast; which in turn makes my notebooks readable and abstract to the person who is executing the project. I have also made full usage of the markdown comments with clear instructions and references of the code if any. To achieve this, I created 2 files **utils.py** and **models_and_metrics.py**. The **utils.py** file contains the methods `data_transforms()` for data augmentations, `get_class_distribution()` for getting the class distribution of the WeightedRandomSampler[14], `without_split_dataset()` containing the original dataset without any splitting, `with_split_dataset()` for combining the train and validation set and re-splitting it into 90% and 10% respectively, `get_default_device()` for checking whether the GPU device is available or not, `to_device()` for getting the Device drivers for faster training[15], `DeviceDataLoader()` for loading the Data Loaders with GPU[15]. Whereas the **models_and_metrics.py** file contains multiple functions for calling all the necessary models, metrics and distinctive fit,test,validate functions, from which the `set_seed()` method handles all the random weight and shuffle initializations and controls the reproducible results, however as discussed in the Section 3, PyTorch does not guarantee reproducibility, you will only see reproducible results everytime you run the notebook on your machine, but if you change the machine the results have minor changes in the metrics [21], this is because of the weights and random seeds initialized by the GPUs. I also used a Early Stopping[16] Class for stopping the training if the validation loss does not improve after a given patience. I have also made 2 notebooks in disguise of **README** files; the **DL_Prashant20200126-README.ipynb** is the main file which consists whole implementation of the project and the **Testing-Submitted-Models_Prashant20200126.ipynb** is used to run the model checkpoints which will demonstrate the metrics which we will see in Section 4. I am also saving the figures, metric logs and model checkpoints in appropriate folders, whenever we run the project implementation in the **DL_Prashant20200126-README.ipynb** file.

### 3.6 Hyper-parameter Tuning

As discussed in Section 3.4, before coming up with the X-Ray CNN Model, I built a baseline model without any activation and dropout layers, and I saw evident poor performance by the CNN Model, then I tried various combinations of adding and removing layers from the model, along with various data augmentations. But a huge amount of time went for the hyper-parameter tuning of the model in PyTorch:

**Epochs** = I tried training(fitting) the models with **[10, 15, 30]** epochs, in the notebook I have demonstrated 15 epochs, due to time constraints, but because of the usage of special DeviceLoaders[15] function, I was running a single epoch in around 1-2 minutes. I trained the models on NVIDIA GeForce GTX 1050Ti 4GB Graphic Card. I also handled the earling stopping epoch phenomenon[16] wherein if the validation loss does not improve after a certain number the training ceases. The same can be seen in the loss graphs in the given README files.

**Learning Rate** = I tried training(fitting) the models with high and low learning rates like **[0.1, 0.001, 0.0001, 0.00001]**. I also tried using the "One Cycle Learning Rate Policy"[17], in which we start with a low learning rate, & gradually increase it batch-by-batch to a high learning rate for about 30% of epochs, then gradually decrease it to a very low value for the remaining epochs. But, I did not see much improvement. This could be more efficiently used when we run the algrithm for over 100 epochs. I will surely try running this project for 100 epochs in the near future.

**Optimizer** = I used the **Adam optimizer**[18] which is an amalgamation of the AdaGrad, RMSProp algorithms to provide more confidence towards the global minimum. It is also a replacement for the stochastic gradient descent algorithm.

**Patience** = Patience[16] is used for giving a certain number to the fit method so that the algorithm stops training if the validation loss is not improved for that certain number of epochs. If we are using 10 epochs, then a patience of **[5]** is good and in case of 15-30 epochs, a patience of **[10-12]** is good.

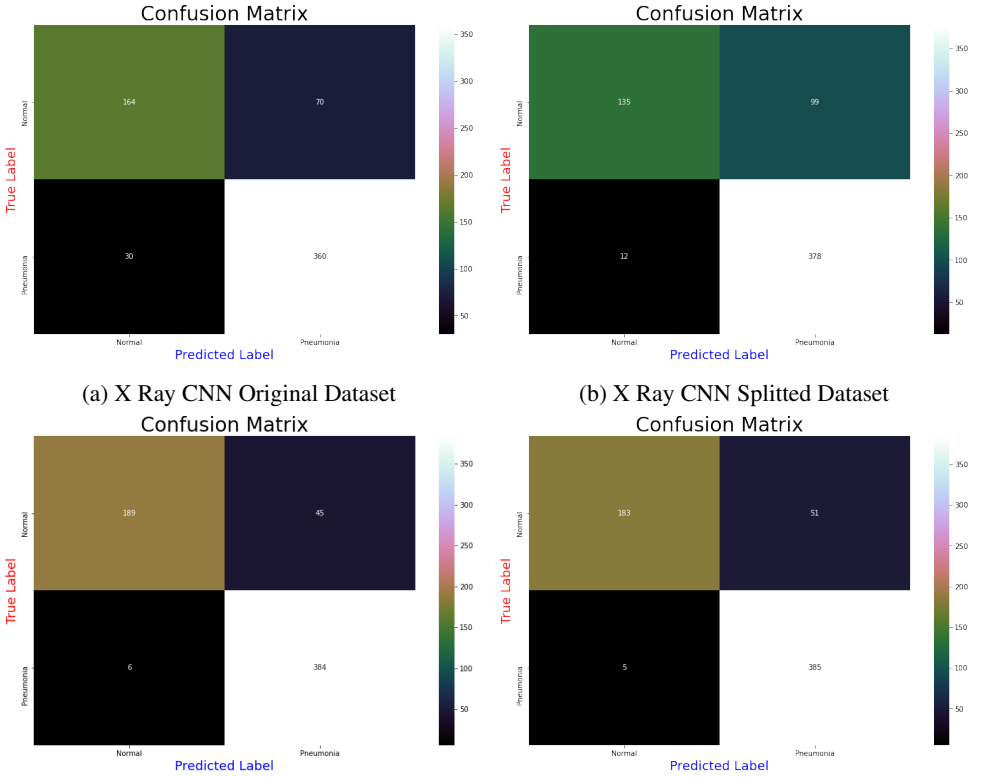
**Weight** = Weight is used for giving an **appropriate weight to a particular class**. Generally, for classes with small number of training images, you give it more weight so that the network will be punished more if it makes mistakes predicting the label of these classes. For classes with large numbers of images, you give it small weight[19]. So the weights given for the original and the later transformed train and validation set is different, and due to the appropriate weights given to the transformed training and validation set, we can clearly see some over-estimation of test accuracy from the original 16 images validation dataset CNN Models. For giving weights for fitting the model, we can even use WeightedRandomSampler[14], which I implemented for the Training DataLoader.

**Loss Function** = I used the **CrossEntropyLoss** as the loss function, and in PyTorch, it is already implemented with Softmax activation. Sigmoid with BCELogitswithLoss is implemented swiftly in Keras with Sigmoid activation, but in PyTorch, I was not able to find good flexibility of implementing it, also; there are almost nearly no Kaggle notebooks which uses BCELogitswithLoss with Sigmoid for this dataset in PyTorch, but I believe it to be more essential and power enhancer for a binary classification problem. Somehow, during the closure of winding up the project, I loosely implemented it, but it was giving poor performance. So, my conclusion is; CrossEntropyLoss with weights for training is best, even for binary classification as it becomes more feasible and a small reduction of that extra Softmax Activation layer as the classifier makes the model more flexible. If we use Sigmoid with BCEWithLogitsLoss we should mention 1 in place of 2 for the number of classes.

**Batch Size** = The most vital hyperparameter which I tuned was the Batch Size. I tried batch sizes of as low as 2 and I could only try the batch size until 32, but after a few epochs I would get CUDA OOM error. In the implementation, I have used a **batch size of 16**. Although, it is known that too large of a batch size will lead to poor generalization of the convergence to global optima function[23]. On the other hand, using smaller batch sizes is beneficial as it allows the model to initiate the learning process in the early epochs. However, the main disadvantage of using smaller batch sizes is that, we cannot guarantee that the model will converge to global optima, in fact it will bounce around the global optima, obviously in accordance to other hyper parameter tuning procedures, as we see for the validation loss of the model containing the original dataset of 16 validation set images. In case of our dataset, I have seen some notebooks where a high batch size have given immensely great scores with a good fit training and validation loss. But unfortunately I couldn't try more than 16 epochs due to low computation power; also Kaggle and Colab are too slow for some reason even with GPU runtime.

# 4 Results

As discussed in Section 3, I ran the 2 models twice; once on the original dataset and the next time combining and re-splitting the validation set into 90%-10% respectively. The crux of the results are not to compare these two ways of evaluating the model, but to just have a comprehensive look at the fine-tuning, originality, speed, and modularity of the setting which I have done to train the models. As the **Testing-Submitted-Models_Prashant20200126.ipynb** file contains all the results of all the models and techniques applied, I am including here the confusion matrices of the 4 runs, because as said earlier; the Low False Negative Rate was the goal.




(a) X Ray CNN Original Dataset

(b) X Ray CNN Splitted Dataset

(c) ResNet50 Original Dataset

(d) ResNet50 Splitted Dataset

Figure 2: Confusion Matrices

As we can see from the Figure 2 (please zoom in), the original dataset's results are very promising, but are they? The validation set is used to estimate how well your model has been trained, but are 16 images enough? That's the reason why I calculated the False Negative Rate, it means that the how many true values are pneumonia and were predicted as Normal. This value is low across all the 4 models. One thing to notice is that the models with re-splitted datasets have done well than the models with original dataset. The training and validation loss of the original dataset's models also shows evident noise in the graph, which leads to overestimating and untrue metrics, which are hard to believe. That's the reason why, accumulating more data into the validation set matters. As we can see the False Positive, which is the Type I error is also having high number in each of the models confusion matrix, but it is more ok to wrongly predict an image under pneumonia category than wrongly predicting normal, because a person can take more assuring medical tests if he/she is falsely predicted having pneumonia than vice-versa.

Other important thing, which is the ying and yang of the results is sensitivity and specificity. All the models have a sensitivity higher than 95%, except the X-Ray Original Dataset Model. It means that the model will return a positive result for more than 95% of the images which have pneumonia,

Table 3: Model Comparisons

| Models | AUC-ROC Score | FNR | Sensitivity(Recall) | Specificity(TNR) | Precision |
|---|---|---|---|---|---|
| ResNet50 $SplittedDataset$ | 88.46 | 1.28 | 98.72 | 78.21 | 88.30 |
| ResNet50 $OriginalDataset$ | 89.62 | 1.54 | 98.46 | 80.77 | 89.51 |
| X Ray CNN $SplittedDataset$ | 77.31 | 3.08 | 96.92 | 57.69 | 79.25 |
| X Ray CNN $OriginalDataset$ | 81.20 | 7.69 | 92.31 | 70.09 | 83.72 |

on the other hand it will return a normal result for about 5% of the images. Specificity, also known as the true negative rate is low across the model, nearly around 70-80%, which is not too bad. But we generally see a tradeoff between specificity-sensitivity. The accuracy in this case will be very biased even though high, because of the imbalance, but as I have used weights and random samplers, I have reported it duly in the notebook. It is thus evident that the ResNet50 is very powerful than my model by not a vast margin, but talking about X-Ray CNN model, the original dataset has overly estimated the performance which is evident from the high AUC-ROC score, so taking the splitted datasets models AUC-ROC score of 77.32% is appropriate which also has a low FNR of just 3.08% as compared to the original dataset models 7.69% FNR. On the other hand, the ResNet50 has an FNR of just 1.28% on the splitted dataset, which outperforms all the other models. The Table 3 above conspicuously tells us more about the results.

## 5 Future Work

Even though I have used ResNet and it is giving us exceptional results, one can even apply multiple models and compare them thoroughly using the given metrics. Models like DenseNet, VGG19, MobileNet have also shown better performances on this dataset[24]. When it comes to augmentations, we can also try shearing or rescaling the images as we do in keras. I also feel that Keras has a better way of Image Augmentation than PyTorch. While training the model, we can use more things like Weight decay, which is yet another regularization technique which avoids the weights given to the class from developing into something larger than the model could afford, it does this decay by adding an additional term to the loss function [25]. Apart from the class labels weights', we can also limit the values of the gradients to a very small range in order to avoid unwanted changes in the parameters due to very big gradient values, this technique is very famously known as gradient clipping [26]. All this and more is valid only when we consider adding more images to the dataset, because heavier hyperparameter tuning and adding more layers to the model is of no use on a minimalistic demo/challenge dataset. All these things which I mentioned are best to be done on real-world dataset. With that said, the application of clinical image diagnosis of pneumonia X-rays can reduce the workload of clinicians and enable patients to obtain early diagnosis and timely treatment, thereby reducing the mortality rate of pneumonia [26]. If I had more time, I would've enhanced the model to another level, which I will surely do by applying more permutations and combinations of the model layers. I will personally implement this project on Keras in the near future. The most primal thing which any Deep Learning practitioner should take into consideration in the near future is the ethical concerns of the model before deploying it in the real world. There cannot be misuse of such technology which is a boon in disguise for the ones in need. But, we should also not forget that deep learning models can be risky, so more cross-evaluations and demo practicing should be done before using it in critical conditions and arriving at conclusions!

## 6 Conclusion

In this paper, I have holistically elaborated on the deep learning models used to diagnose an X-Ray Image as Normal or Pneumonia based on its patterns. By performing this wholesome task, I'm able to differentiate the normal/pneumonia chest X-Ray images with a low false negative rate of nearly 1.28%. Because of the residual blocks of ResNet50, even a 50-layer model was able to outperform my own CNN model from scratch. But with nearly half of layers as ResNet50, X-Ray CNN model has given promising results as evident from Section 4. All this was possible because of the unique data augmentations, out-of-the box device loaders for fast processing and results retrieval, which inturn made the hyper parameter tuning furious and worthy. Also, the weights, balancers, and early stoppers applied during the training made the training more unbiased and reliable. Subsequently,

even though PyTorch has loose reproducibility, I wrote a function to achieve it, because of which one can get same results on their own PC on multiple runs of the project implementation. Modular python coding, easy retrieval of functions, plots, model performance logs and checkpoints, up to the mark - fit, evaluate and predict function, and a lot more makes my project implementation stand out. As I implemented this project in PyTorch, and nearly read 1000+ discussions; I can now confidently perform any image classification task.

## Acknowledgement

I hereby acknowledge that this project has been accomplished just by myself and all the references to research papers have been thoroughly mentioned at the end in the References section 6 as well as in the project implementation code; where ever I have used any methods/functions/classes from online repositories such as kaggle/git, I have provided citation for the same, either in the code or over here. In the end, I would like to thank Prof. Guenole Silvestre, TA Nikita Pavlenko, along with TA Eoghan Cunningham & TA Alessandro Ragano at University College Dublin and anonymous referees for giving me the opportunity to perform this comprehensive task & to the references, which helped me form technical ideas & improve the quality of the project implementation.

## References

[1] Cellina M, Orsi M, Toluian T, Valenti Pittino C, Oliva G. False negative chest X-Rays in patients affected by COVID-19 pneumonia and corresponding chest CT findings. Radiography (Lond). 2020;26(3):e189-e194. doi:10.1016/j.radi.2020.04.017.

[2] Li W, Ding C, Yin S. Severe pneumonia in the elderly: a multivariate analysis of risk factors. Int J Clin Exp Med. 2015;8(8):12463-12475. Published 2015 Aug 15.

[3] Wang D, Mo J, Zhou G, Xu L, Liu Y (2020), An efficient mixture of deep and machine learning models for COVID-19 diagnosis in chest X-ray images. PLOS ONE 15(11): e0242535.

[4] Okeke Stephen, Mangal Sain, Uchenna Joseph Maduh, Do-Un Jeong, An Efficient Deep Learning Approach to Pneumonia Classification in Healthcare., Journal of Healthcare Engineering, vol. 2019, Article ID 4180949, 7 pages, 2019. https://doi.org/10.1155/2019/4180949

[5] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu (2017), CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. PLoS ONE 15(11): e0242535. https://doi.org/10.1371/journal.pone.0242535

[6] Ibrahim, A.U., Ozsoz, M., Serte, S. et al. Pneumonia Classification Using Deep Learning from Chest X-ray Images During COVID-19. Cogn Comput (2021). https://doi.org/10.1007/s12559-020-09787-5

[7] D. Varshni, K. Thakral, L. Agarwal, R. Nijhawan and A. Mittal, Pneumonia Detection Using CNN based Feature Extraction.. 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 2019, pp. 1-7, doi: 10.1109/ICECCT.2019.8869364. Cogn Comput (2021). https://doi.org/10.1007/s12559-020-09787-5

[8] Mohammad Farukh Hashmi, Satyarth Katiyar, Avinash G Keskar, Neeraj Dhanraj Bokde, Zong Woo Geem, Efficient Pneumonia Detection in Chest Xray Images Using Deep Transfer Learning. Diagnostics (Basel) 2020 Jun; 10(6): 417. Published online 2020 Jun 19. doi: 10.3390/diagnostics10060417 PMCID: PMC7345724

[9] Zhichao Lu, Ian Whalen, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, Vishnu Naresh Boddeti, Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification. (2019). Published in IEEE Transactions on Evolutionary Computation. arXiv:1912.01369

[10] Yadav, S.S., Jadhav, S.M. Deep convolutional neural network based medical image classification for disease diagnosis.. J Big Data 6, 113 (2019). https://doi.org/10.1186/s40537-019-0276-2

[11] How ReLU and Dropout Layers Work in CNNs

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. (2015). Computer Vision and Pattern Recognition, arXiv:1512.03385

[13] Qingge Ji, Jie Huang, Wenjie He, Yankui Sun. Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images. (2019). OCT image processing, 10.3390/a12030051

[14] PyTorch Sampling Samplers

[15] Deep Learning with PyTorch: Zero to GANs course

[16] Early Stopping.

[17] One Cycle Learning Rate Policy.

[18] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.

[19] What is the weight values mean in torch.nn.CrossEntropyLoss?

[20] Discussions related to inconsistencies in the Kaggle's Pneumonia Chest X-Ray Dataset.

[21] Issues with Reproducibility in PyTorch. Discussions in PyTorch Forum.

[22] Where should I place the batch normalization layer(s)? Discussions in PyTorch Forum about Batch Normalization of Linear Layers.

[23] Effect of batch size on training dynamics.

[24] Zhenjia Yue, Liangping Ma, and Runfeng Zhang, Comparison and Validation of Deep Learning Models for the Diagnosis of Pneumonia.. Volume 2020 |Article ID 8876798 | https://doi.org/10.1155/2020/8876798

[25] This thing called Weight Decay.

[26] What is Gradient Clipping?