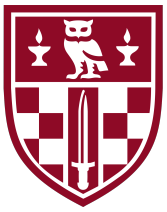
Birkbeck, University of London

Pneumonia Detection from Chest X-Ray Images

Author:
Baran Buluttekin

Supervisor:
Prof. George Magoulas



A project report submitted in fulfillment of the requirements for the degree of MSc Data Science

in the

Department of Computer Science and Information Systems

September 28, 2020

Declaration

I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta- searching software.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Professor George Magoulas for advice, encouragement and guidance he provided throughout this project. Finally, I would like to thank my wife Inga for supporting me during the compilation of this project.

Abstract

This project implements a working CI/CD pipeline for detecting presence or absence of pneumonia from X-ray images. Predictions part of the pipeline first sets up a benchmark with well known neural network architectures then using techniques such as transfer learning and designing a custom architecture to exceed the benchmark results. In addition to transfer learning and custom model design, custom data pipeline is implemented to create a balanced dataset from imbalance data with data augmentation. Whenever the new model with a better performance found or any changes added to the code base, the automated pipeline runs integration tests and determines the compatibility of the new changes with the rest of the implementations. Upon successful passing of the tests, the model can be inspected with the model visualization technique GradCAM [38]. If model interpretability requirements are also met the model then deployed to live static website where users can input X-ray images to receive predictions.

Contents

1 Introduction

Medical diagnosis and specifically computer-aided diagnosis (CAD) is a hot topic in the field of technology. One of the main reasons for becoming a hot topic is the recent innovation and breakthroughs achieved by computer vision research. Combined with poor healthcare coverage around the globe, CAD systems offer a promising solution to mitigate the devastating impact of fatal diseases such as pneumonia. Achieving human-level accuracy in computer vision task in a wide array of classification task such as ImageNet large scale visual recognition challenge (ILSVRC) [5] sparked the debates about whether these CAD systems can reduce or altogether replace the jobs such as radiologist in the future. Controversial topics such as whether or not artificial intelligence will replace the radiologist in the future aside, these automated systems can offer answers for patient's questions in absence of medical help or to very least offer much needed second opinion in the face of unsatisfied diagnoses. Given all the mentioned possible benefits of the CAD systems, this project is focused on building classification CAD systems for diagnosing pneumonia from the chest X-ray images together with implementing CI/CD pipeline for automation.

1.1 Aims and Objectives

The aim of this project is to build a fully functional chest X-ray image classification pipeline that implements CI/CD principals for testing and deployment. These pipelines also referred to as MLOps where the part of the machine learning workflow is automated.

1.1.1 Objectives

The project will be implemented with the execution of the following objectives:

1. **Data pre-processing and data exploration:** Preparing the data for model ready state and general data exploration.

2. **Building baseline model with well known neural network architectures:** This step involves setting additional benchmarks with out of the box models.

3. **Increasing model performance:** Using custom architecture and techniques such as transfer learning to increase model performance beyond bench-

mark levels.

4. **Ensuring model interpretability with visualization:** For making sure model learning as intended and extracting useful informations out of the image.

5. **Applying different deployment options:** Implementation of model development. Based on the best choice for project specification.

It's worth emphasizing that the objective of this project is not to achieve the state of the art result in pneumonia detection but to offers a preferred method for improving and enhancing the existing models. The intuition behind choosing the above objectives instead of attempting to build a novel state of the art architecture from scratch is the process of building such architecture has a very large search space and requires a lot of iteration and experimentation. Due to the limited time frame of this project attempting to obtain the state of the art model is not feasible.

1.2 CI/CD Pipeline

In this section, I will give a brief introduction to the CI/CD pipeline to explain what CI/CD is and why it is chosen as a preferred way to build this project.

Continuous integration (CI) is a process design to help software teams to develop projects with confidence. This allows members of the team to find incompatible code, merge conflicts, and increase the overall reliability of the software. Often, a team will apply CI with automated testing using a server or tools such as Jenkins to assess the integration of the development code. When a new code added to a repo, build and the testing on this commit automatically starts. If the tests are passed that would indicate that the code can be merge to the rest of the project. The CI server will send back output containing the results of the build and a sign of whether or not the branch passes the tests for integration into the main branch. With introducing build and test information to commits on all of the branches, CI makes continuous delivery possible along with the related process called continuous deployment. Difference between continuous delivery and continuous deployment is that continuous delivery lets you develop code with automated release integration. Combined with CI, continuous delivery lets you develop projects with a modular code base that can be integrated into the rest of the project with ease. Even though the benefits of using CI/CD pipelines are more prominent in the software teams, integrating automated testing will help even individual projects such as this by reducing technical debt.

In more granular detail, this system works with central version control services and for this project central version control service used is Github. GitHub uses a communication tool called *webhooks* to send messages to external systems about activities and events that occurred in the project. For each event type, subscribers will receive messages related to the event. Generally, events refer to actions involving the software development such as new commit push, pull (merge) request, or other software related actions. In this case, whenever a new commit is pushed

to any branch of the project, a message from Github will be sent out to a third party system called *travis.* [1] Travis is a hosted CI service that allows building and testing software hosted in version control services. When travis receives the webhook call, it will fetch the most recent version of the project and run the tests associated with it. When the test runs completed with the latest version of the software, test results will be sent back to relevant commit as status update using GitHub API. This information can either be used by developers for making decisions such as whether to accept the pull request or reject it. If applicable, the update can be used by service to initiate the deployment process for the software. In all cases, CI/CD is an automation tool for software quality assurance process to speed up the development and improve the overall reliability of the software.

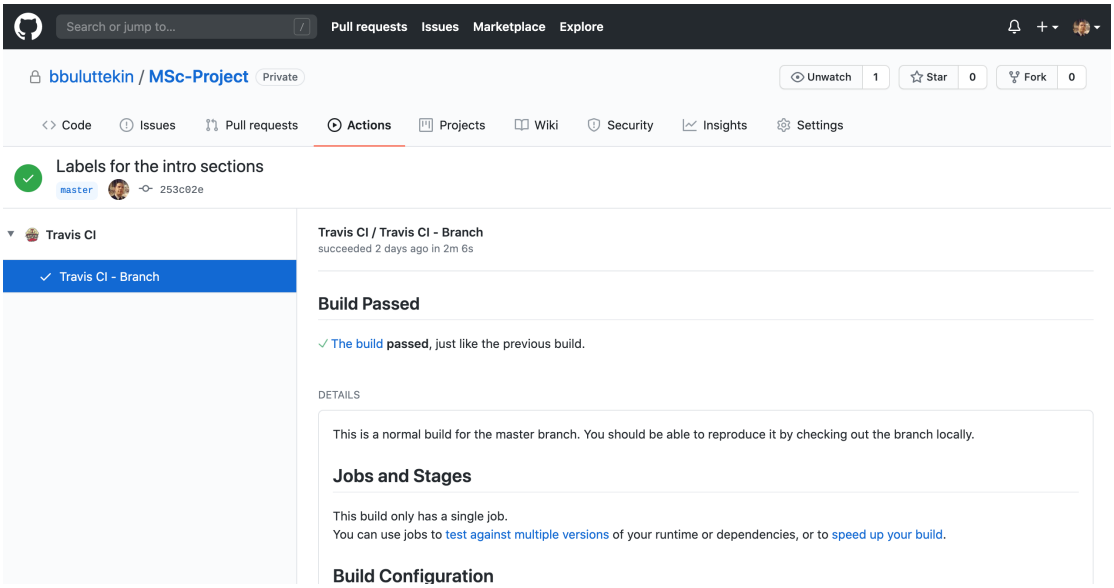


Figure 1.1: CI feedback received from Travis.

1.3 Project specification and design

This project I aimed to keep code and reporting together to provide easy reproduction. Codebase design to be extendable and modular. Therefore, I assign a sub-folder for all the project-specific code under the name *src*. Having a module in the same directory level with the other components allows the ability to use the code in notebook experiment as well as with the tests in the CI integration. Both project proposal and report developed using the LaTeX typesetting system and documents kept in the version controlling to allow easy changes and rolling back to the desired version. Finally, root-level files such as *.travis.yml* and *requirements.txt* is instrumental in defining which steps to take in CI runs and constructing a near-identical environment for software dependencies. Below, I added a directory tree to serve as a guide for navigating and finding project files.

[1] https://travis-ci.org/

```
├── Proposal
│   └── Proposal files
├── Report
│   ├── chapters
│   │   └── Chapter files
│   ├── img
│   │   └── Images
│   └── Main project latex files
├── scripts
│   └── Utility scripts
├── models
│   └── Deployment model
├── notebooks
│   └── Experiment notebooks
├── src
│   └── Python library files
├── static
│   └── Files for static website
├── tests
│   └── Test files
├── .gitignore
├── .travis.yml
├── index.html
├── README.md
└── requirements.txt
```

1.4 Reproducibility Guidance

As a scientific project, it is very important that anyone can reproduce the experiments and findings in this project to verify the conclusions that reached are accurate. Main components of reproducible research are open code, open data and repeatable software runtime specification. Open code component is the most straightforward among the other components as the source-code produced part of this project will be shared with the project reviewers [2] and will be made public in GitHub [3] once the assessment of the project is completed. Dataset [16] used in this project is also available through the website URL cited and hosted in online data science community called Kaggle [4]. I have used Kaggle as the main source of accessing this data for two reasons, firstly for its functionality of allowing API calls to retrieve data and secondly for managing the data versioning for the user. Data versioning is an integral component of the reproduction of machine learning projects because the model produced by the training will heavily depend on the data it is trained on. The current version of the dataset as of the writing of this

[2] https://github.com/Birkbeck/msc-data-science-project-2019-20-files-bbuluttekin
[3] https://github.com/bbuluttekin/MSc-Project
[4] https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

project is version 2. To enable easier runtime replication and to leverage computational power I have chosen to use an online service called Google Colaboratory. [5] Colaboratory or *"Colab"* for short, is a free service provided by Google Research. It will allow running Python code through the browser that connected to remote compute resources. Considering that Colab is a remote compute resource, I have created starter utility scripts to automate data acquisition. These files can be found inside the *scripts* folder. Please note that using these script will require obtaining API key from Kaggle platform and this API key file should be in the path specified in the scripts. However, reproducing in the Colab is optional and software dependencies required to produce local development environment is provided with the "requirements.txt" file. Lastly, custom software components for this project resides in the "src" folder and this folder must be placed in a location available to the scope of the python runtime.

[5] https://colab.research.google.com/

2 | Background

Generally, the first part of every machine learning project is to choosing the algorithm to tackle the problem in hand. As I stated in the proposal of this project, I choose to apply specific machine learning algorithms called Artificial Neural Networks (ANNs) to tackle the classification challenge of detecting pneumonia in X-ray images. The first part of this chapter I would like to provide some information to justify that decision.

The objective of the algorithm in this report is to classify X-ray images with or without pneumonia. Classification is a task of determining what is the defined class of an example given the data associated with it. In our case, we defined our classes for prediction to the person in the example image having pneumonia or not. In order to achieve that goal, machine learning algorithm must produce a function that outputs the class within defined finite possible classes such as $f : \mathbb{R}^n \rightarrow \{1, \ldots, k\}$ which in this case k is equal to 2. In essence all machine learning algorithms will map input representation of the data $\boldsymbol{x}$ to prediction output $\hat{y} = f(x)$. The only difference between them is how each algorithm is representing the data distribution with a model f. Which consequently leads to the question of which algorithm is the best algorithm for machine learning or which algorithm to choose for finding the best model representation. According to **no free lunch theorem** [36] there is no such algorithm exist that will consistently achieve low error rate averaged over all possible distributions. In other words, no model is universally any better than any other machine learning model. Luckily, the objective in this project is not to find the universally best algorithm but rather to find the algorithm that will find the best representation for the data distribution of this dataset. Historically, traditional machine learning algorithms often performed poorly on tasks such as computer vision, detecting objects or speech recognition. Part of the reason is, traditional algorithms rely on the similarity of data points and these tasks usually have high-dimensional data, yet the notion of nearness gets weaker with the high dimensionality. An effect is commonly known as *curse of dimensionality*. Because of the weakness described earlier, Artificial Neural Networks emerges as a clear choice for image classification and object detection task which became evident with the performance of AlexNet [19], VGGNet [30] and ResNet [13] in the ILSVRC [5].

2.1 Building Blocks of ANN

The idea of Artificial Neural Networks inspired by the neural cells of the human brain. Earliest known research for ANNs dates back to 1943 as a multidisciplinary work of psychology and mathematics by Warren McCulloch and Walter Pitts [22]. Their work covered how computational logic could model complex neural cells activities. However, first development that reshaped the way for current generally accepted practices of ANNs was the work of Frank Rosenblatt's *"The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain"* [27]. Perceptron is the simplest linear ANN that always converges to hyper-plane when there are two sets of classes that can be separated linearly. Like the current single neurons generally used in modern ANNs, for producing an output it calculates the sum of weighted inputs and applies a *step function* to that sum. For example, let the input $\boldsymbol{x}$ be n-dimensional input vector, Perceptron first calculates $z = w_1x_1 + w_2x_2 + \ldots + w_nx_n = x^Tw$ then pass this weighted sum of inputs to step function $h(z)$ below to calculate final output.

$$h(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \tag{2.1}$$

Current ANN units also have the same properties except for the use of step function. Instead, current ANN units use a set of non-linear functions that generally called as *activation functions*.

Despite its robust nature Perceptron is ultimately a linear model which implies that it can only be effective for the linearly separable data distributions. The popularity of the algorithm is faded as the limitations such as not being able to separate logical operation exclusive OR (also know as XOR) is discovered [21]. However, later on, it has found that these limitations can be eliminated by just using layers of many Perceptrons together as a single model and the resulting model is named *Multilayer Perceptron* (MLP). (*Feedforward Network* is another term that usually used interchangeably with Multilayer Perceptron.) MLPs uses a concept called layer which is useful for calculation and managing the architecture of the model. In a nutshell, a layer is the combination of neurons with biases stack together (with exception of output layer) that have the same input source. Concept of a layer is used in many different architectures, for example, a layer with each neuron that connected to each unit in previous and next later is known as *Fully connected layer* or *Dense layer* whereas layer with pre-defined convolution operation is called *Convolutional layer*.

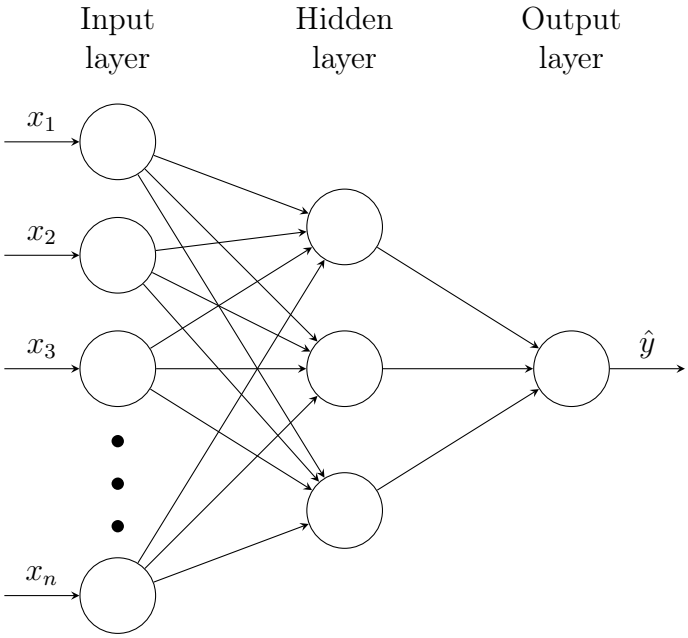


Figure 2.1: Illustration of a MLP model.

2.2 Exploding and Vanishing Gradients

Having the sequential architecture described in section 2.1 MLPs faces an additional implementation challenge that is not common in traditional machine learning, the difficulty of training. The typical training process for the machine learning algorithm is at a very high level is a standard procedure. The first step is to feeding input data to model and produce an output, then based on desired output for the input, loss value can be calculated. Loss value is a calculation of how much the output is further away from the desired output. Using this loss value we can approximate weight updates with the help of the optimization algorithm until the model weights converge. Even though the training process is also the same for the MLPs there are more complexities involved in MLPs training. Given that instead of dealing with one model we have layers of neurons chained together that each has its own weights. For those reasons training MLPs was a challenge until the influential *backpropagation* paper [28] is published. In this paper efficient technique of calculating the gradients using forward and backwards passes demonstrated. Utilizing the chain rule of calculus, the backpropagation algorithm was able to calculate the update for each weight in every neuron.

With the help of the backpropagation algorithm, need for training deeper networks increased, but the difficulty of training such networks remained a problem. Part of the problem was, the gradients get smaller and smaller as the gradient flowed down to lower layers (layers close to the input layer) of the network. When the gradient updates get close to very small values some of the lower layer weights do not update enough and consequently not converging model to appropriate representation. This phenomenon is usually referred to as *vanishing gradients* problem. Similarly, some network such as recurrent neural networks can have an

opposite problem that resulting in gradients getting larger and larger which pushes weights to be a very large number. Similarly, this phenomenon referred to as the *exploding gradients* problem. Later on, this unstable gradient flow problems are shown to be the combination of choice for weight initialization and the characteristic of the activation functions that widely used [9]. Previously, activation function often get used was the sigmoid (also known as logistic function) $\sigma(x) = \frac{1}{1+e^{-x}}$ and the hyperbolic tangent function $(tanh(x) = 2\sigma(2x) - 1)$ have a flatten tails where $\boldsymbol{x}$ get very large or very small. Giving that the chain rule states that the derivative of a variable is equal to the product of the partial derivatives of the composite function with respect to output, neurons that produce an output very large and very small will not likely to be updated. For example lets suppose $\hat{y} = \sigma(z(w, b))$ and $z(w, b) = wx + b$. Then chain rule states that

$$\frac{\partial \hat{y}}{\partial w} = \frac{\partial \hat{y}}{\partial \sigma} \frac{\partial \sigma}{\partial w} \tag{2.2}$$

It is clear that the healthy gradient update of $\boldsymbol{w}$ is predicated on the gradient of the σ is not being zero or too close to zero. Sigmoid and hyperbolic tangent functions produce derivatives that are very close to zero when the function inputs are on the saturating region. To eliminate this effect, *Rectified Linear Unit* (ReLU) activation function applied as a good alternative to training deeper neural networks. ReLU behaves like a linear function for the non-negative input values $(f(x) = max(0, x))$ and outputs zero for the negative input values. It's also implied that the derivative of the ReLu is either zero for the neuron outputs below zero or one for neuron outputs that are positive.

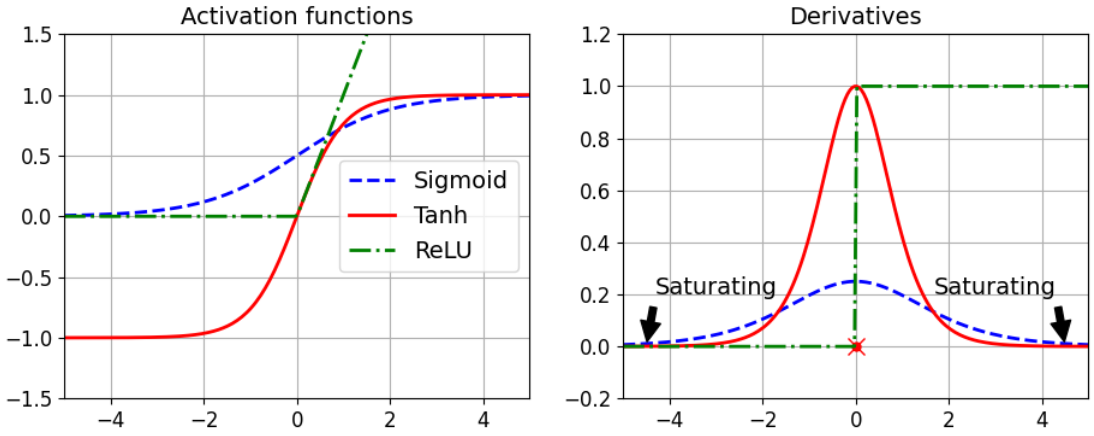


Figure 2.2: Saturation points in activation functions.

Because of the consistent gradient flow generated by ReLu. It is usually a better choice for starting an experiment with such activation functions for hidden layers to ensure the gradient flow is faster and Network will converge faster as a result.

2.3 Optimization

Most machine learning algorithms utilize some sort of optimization. The main objective of the optimization algorithms is to find the maximum or the minimum point for the function in hand with respect to one or more variable. For the machine learning domain, we would like to optimize the loss function (also known as a cost function) to the global minimum for having a model that is most representative of the data. The task of searching minimum or maximum can be used interchangeably as finding the minimum point of the function f may also be found by getting maximum of the inverse function of the same function (f^{-1}). In essence, the process is achieved by taking derivatives of the function which will give us a slope for the given point, the moving opposite to the slope with the small increments until we reach to the minimum point where the gradient is zero. The process is also known as *gradient descent*. Gradient descent used in machine learning field for a long time, however depending on the complexity of the function space this process could take a significantly long time. For mitigating this problem new set of algorithms called *momentum optimizers* created. The difference between these algorithms and gradient descent is simply, gradient descent will take small consistent steps regularly throughout the optimization process. These momentum-based algorithms work by paying attention to the previous gradients and accelerate the updates based on gradients and the slope at the point of the gradient. This will help the optimization to converge to global minimum faster when there is a lot of plateau areas in the function surface. Later on, momentum algorithms expanded to include taking the steepest dimension of the gradient to point updates more toward to the global minimum, a new family of optimizers named as *adaptive* optimizers. Most well know algorithms of this category is AdaGrad [7], RMSProp [33], Adam [17] and Nadam [6]. Despite the fact, these adaptive optimizers usually converge to good solutions faster, research by Ashia C. Wilson et al [35] showed that in some cases these optimizers can lead to poor generalization result depending on the dataset. It might be good practice to try adaptive and momentum-based optimizers in the training job to observe which is the most suitable optimization algorithm for that dataset.

2.4 Regularization and Over-fitting

Generally, the fundamental problem in machine learning is that our model should perform well on previously unseen data points which is also known as generalization. If the scope of the model is sufficiently large (model with a large number of parameters) optimization techniques described previously could enable the model to capture noise that specific to training records. In other words model can *overfit* to the dataset. Overfitting is where the model will focus on individual nuances of data records instead of focusing on general pattern overall. There are numerous regularization techniques available to mitigate the overfitting in ANNs. Just like traditional machine learning l regularization (norm) can be applied to the artificial neural networks. In general form l_p norm is given by

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}} \tag{2.3}$$

for vector $\boldsymbol{x}$ and $p \in \mathbb{R}$, $p \geq 1$.

Most frequently used l norms are l_1 and l_2 regularization. The l_1 norm usually used when the difference between elements are important. This regularization seldom referred to as *Manhatten distance* in the literature. Similarly, another norm get used often is l_2 norm, also known as *Euclidean distance*, is the geometric distance between points and could be calculated simply as $x^T x$

In addition to l_p norms *dropout* [32] emerges as a popular regularization technique specific to the ANNs. Dropout is a process of removing a fraction of the units from a neural network layer by setting the weight of the unit to zero during training. This technique reduces the reliance of each unit for the prediction. In each training iteration fraction of the neurons dropped randomly so the remain neurons are forced to train in the absence of the neurons picked. Although dropout has proven to improve generalization in many machine learning tasks, due to its unstable nature time it takes for the networks to converge usually fluctuates.

2.5 Convolutional Networks

In the proposal of this project, I used the definition from Deep Learning book [10] for the definition of Convolutional Neural Networks (CNNs) that I believe summarize the concept very well. To quote that definition again CNNs are:

> "Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers."

In CNN's neurons replaced by units usually called kernel or filter. These units are rectangular matrix span over the layers of the input data. The way this unit calculates its output by positioning each kernel item to pixels on the image and calculate the element-wise multiplication between them. The kernel then slides over the rest of the image until all image covered. The area that processed by the image is called *receptive field* and operating this way by calculating each area in image CNNs can capture information about the locality of the image as opposed to flattening. The output of the operation depends on the design of the convolution operation. There are two main concept defines the operation, one is stride or how many pixels the kernel will be moved to complete the operation, and the size of the padding. Padding is zero value pixels that get placed around the image to centralize the items at the edge of the images.

The output of the convolutional network can be calculated easily to ensure no bug is introduced to the model during implementation. Given the volume size of the input image ($\boldsymbol{I}$), kernel size of CNN ($\boldsymbol{K}$), stride which is applied ($\boldsymbol{S}$), and the amount of zero padding applied ($\boldsymbol{P}$). We can calculate the output as:

$$output = \frac{(I - K + 2P)}{S} + 1 \tag{2.4}$$

The generally accepted way of padding in convolution operations are, the SAME and the VALID padding. SAME padding is using zero paddings around the image to output *same* size output from the convolution when the stride is one. VALID padding, however, is a practice of not applying any zero paddings and only uses *valid* pixels from the image.

Conventionally, convolutional layers are used together with the pooling layer. The main contribution of the pooling layer is to under-sample the output data. Doing so they will reduce the memory usage and the computational complexity of the network. Additionally reducing the size of the output will reduce the chance of overfitting. Most frequently used versions of the pooling layer are max pooling and average pooling. Regardless of the operation, the pooling layer works by choosing the pool size and applying the desired operation to that field in the input. For example, if it is a max-pooling layer, the maximum pixel value in the polling size is passed on to the next layer.

2.6 Software Challenges Specific to Machine Learning Systems

It is clear from some of the problems mentioned earlier, training deep neural networks is a difficult task. One particular reason for difficulty when applying ANNs to new fields is identifying the problem when getting bad scores from the model. Because unlike tradition software development it is difficult to know whether there is a bug in the implementation of the software or the model itself is not suitable to represent the underlying data distribution. For overcoming this challenge I am employing a diagnosis and debugging checklist that will help identify and eliminate any bugs that can occur in the implementation phase. Where applicable checklist points are:

- Overfit to a sample data to ensure no optimization mistakes made.
- Monitor training loss during training and make sure it is declining throughout the training.
- Check the input and output shapes are correct.
- Make minor changes to models and only add additional changes after ensuring there is no bug in the system.

Another challenge is the computationally expensive nature of the training. Due to the iterative nature of the machine learning, it is vital that the experimentation for the task will run sufficiently quick that the improvement for the model can be identified quickly and progressed to further stages. However, standard ANNs usually have millions of parameters that need to be updated while training with hundreds of thousands if not millions of training examples to train on. Considering the average training process that will need multiple passes on the entire dataset for converge to solution the need for high computation power becomes evident. That's

why most training is done with the help of hardware accelerators such as GPUs or TPUs. Choice of computation environment for this project luckily includes such hardware accelerators but these special hardwares must be detected and enabled so the calculation can take advantage of the computing power. To enable the hardware related capabilities and to reduce training management overhead I have written a wrapper module in the custom code library. Some of the other features that made possible with this module are:

- Ability to detect and set the hardware accelerators.
- Choosing training strategy suitable for the hardware accelerator.
- Logging performance metrics persistently.
- Saving model persistently in case of computing failure and continue where the training left previously.
- Extract final model with runtime data.

Possibly one of the most important features of the helper library is providing resilient training. Some large models take hours or days to train and my choice of computation environment allows only twelve-hour training before terminating. Not to mention virtual machines power that environment might disconnect or terminate much earlier. In the case of such an event, if all the previous training is lost, time spent on training will be wasted. If similar events happen many more times it could even risk not completing all the experimentations on time for this project deadline. To eliminate that risk, I have implemented a process that will save model weight to storage outside the compute environment regularly. If such failure happens, the module will pick up the last saved weights and continue to training where it left of previously. In addition to starting the training with the previous model, by using naming conventions, the module can initialize the epoch number for the training accordingly and allow accurate comparison between other models. For instance, if the first training attempt is failed in epoch number 23 restarting training will start with the epoch number 23 together with the weights from that epoch.

2.7 Literature Review

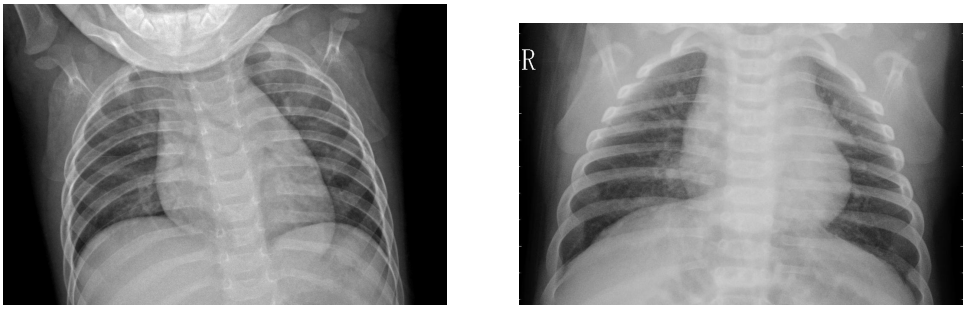
Applying machine learning to medical imaging has seen an increase in popularity with the emergence of the popularity of neural networks. Such algorithms applied to many medical diagnosis problems such as Gulshan et al [11] diabetic retinopathy detection or Estava et al [8] cancer classification etc. More specifically performance of convolutional neural networks and localization studied in Islam et al [15] by using OpenI [24] dataset.

Because of the limited number of examples in the dataset for this project, the most impactful part of the literature was the application of transfer learning in the medical imaging field. Use of transfer learning becomes ubiquitous when it

comes to eliminating the learning problems of the small datasets. But the use of transfer learnings effectiveness is an open question in this field because of the dissimilarity between the source dataset of learned features such as imagenet versus the datasets of medical imagery. Performance of the pre-trained networks in such cases has found to be lower compared to similar tasks (Yosinski et al [37]). Another technique that found to be successful is initializing models with the pre-trained weights rather than random initialization [25, 18, 12]. Evidence of effectiveness of this method is also proven by the results of CheXNet [26] which have fallowed the initialization of the Dense convolution networks this way and results of this network surpassed practitioner level accuracy in X-ray images. Considering all these encouraging results, it is a worthwhile option to explore this technique as part of this projects as well.

3 | Data

Dataset [16] for the project is X-ray images collected at Guangzhou Women and Children's Medical Center, from pediatric patients aged between one to five. All X-ray images are in JPEG format and organized into a folder-based structure. Main sub-structures in the dataset are train, validation and the test folders. All main folders are also broken into sub-folders that named *Normal* and *Pneumonia* to indicate the classification of the images they contain. All X-rays have classified by expert physicians and check individually, damaged or un-readable X-rays discarded. Below are the illustrative examples from both classes of the X-ray.



(a) X-ray without Pnuemonia (b) X-ray with Pnuemonia

Figure 3.1: Two sample X-ray Chest images with and without pneumonia.

For a more detailed understanding of the data, I have checked the content of each train, validation and test datasets. There is a significant imbalance in between Pneumonia and Normal classes for train and the test datasets. A detailed breakdown of each dataset is given in table 3.1.

Dataset Name	No of images	Pneumonia	Normal
Train	5216	3875	1341
Validation	16	8	8
Test	624	390	234

Table 3.1: Breakdown of images for each classification folder

3.1 Data Augmentation

Data augmentation is a process of producing additional training instances by modifying existing training data points. Process of data argumentation is very common within the computer vision field. Generally, because it is well understood that the images could be modified some way without having to lose the classification of the image. For example image of a cat on the table can be cropped such a way that will still contain the cat on the table but without the background that is not relevant. The resulting image from this cropping will also be an image classified as a cat image.

There are several sets of transformation that could be applied to images to create labelled data instances. It is import to choose which augmentation to apply and which one to abandon. Some of these augmentations would be appropriate for the problem set some are not. Example of highlighting some argumentation that is not appropriate is flipping the image upside down. Even though flipping upside down is acceptable some cases like the cat image example we considered earlier. It would not be useful in the setting of Pneumonia classification because considering upside down X-rays is not a natural occurrence in the field of medical diagnosis. Perhaps the most striking example of an area that this argumentation technique should not be used is handwritten digits classification. Reason for that is although upside number one is still an image of number one, in the case for number six it will result in images labelled as six but in content, they will appear as number nine. Train machine learning model with samples of number nine labelled as six will indeed hinder the accuracy of the model instead of helping it.

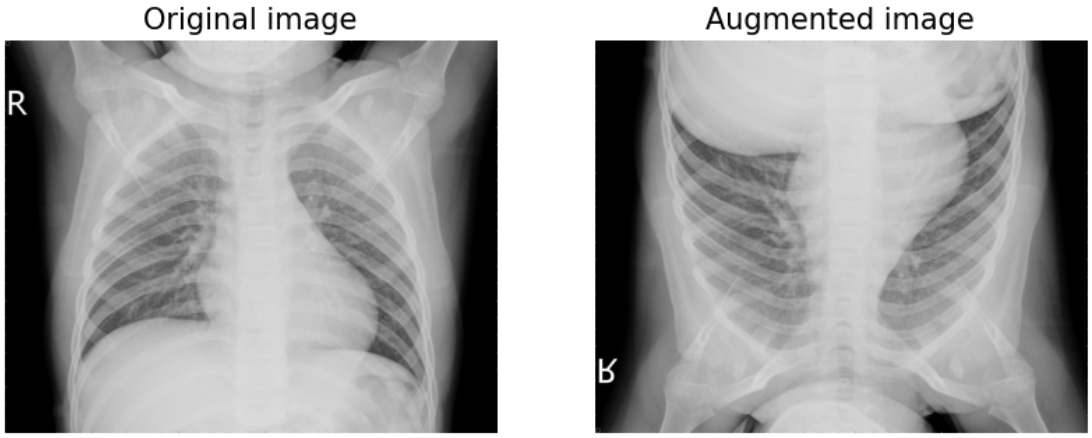


Figure 3.2: Flipping an image upside down is not applicable in CAD context

3.1.1 Horizontal flip

This data augmentation is achieved by reversing the pixel values horizontally. The resulting image would look like a mirror flip from left to right of the same image.

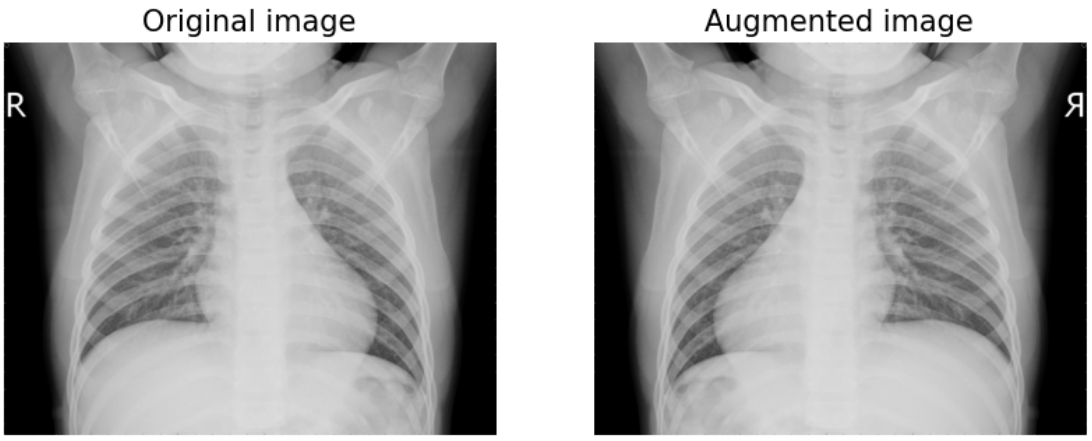


Figure 3.3: Horizontal flip augmentation

3.1.2 Random zoom augmentation

Also knows as random cropping augmentation. This argumentation applied by randomly sampling a certain percentage of the original image and either adding padding to a sampled area or rescale the sampled area with interpolation to produce an image of the same shape as the original image. It is important to choose an appropriate percentage for this argumentation. The percentage should be chosen in a way that it would not omit the area of interest in the image. For the purpose of my project, this argumentation must be applied such a way that it would not leave any part of the lungs outside of the image. As an example figure 3.4, is applied with 10% of random zoom and it is a considerable level for this dataset.

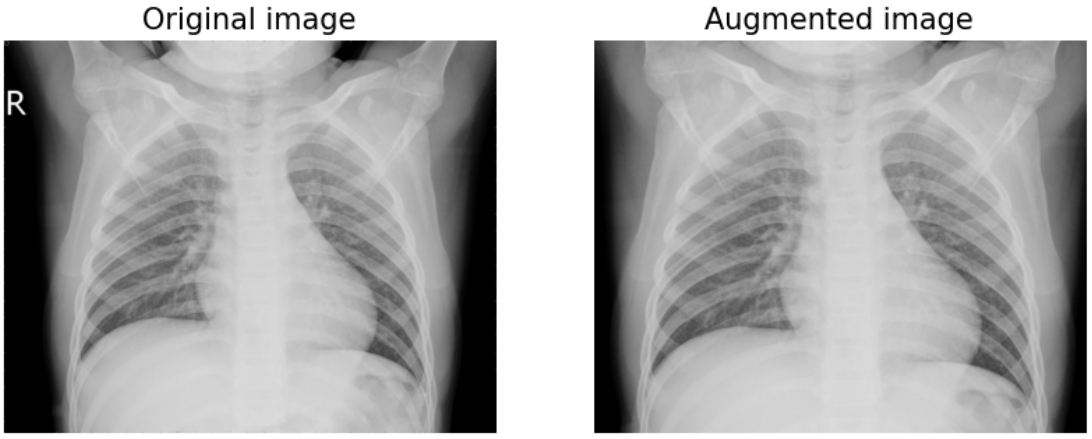


Figure 3.4: Random zoom augmentation

3.1.3 Changing brightness or saturation

These argumentations either change the brightness or saturation of the image to create an augmented image. Saturation is done by converting RGB images to

HSV (Hue Saturation Value) then multiplying saturation channel with saturation factor chosen for the augmentation.

Brightness augmentation also works a similar way. RGB values of the image are converted to float representation then applied brightness factor to the values.

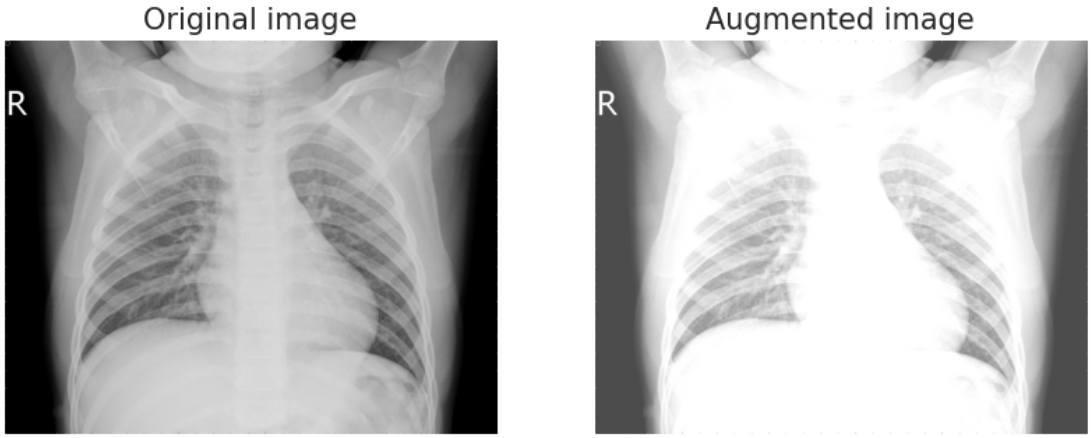


Figure 3.5: Brightness augmentation example

3.2 Data Representation

Original and augmented images need to be turned into data representations that machine learning algorithms can process. Normally, colored images represented with pixel values for red, green and blue (RGB) color channels and values for each pixel in any given channel ranges between 0 to 255. For the traditional machine learning techniques, each data point should be represented in one vector. Additionally, pixel values for each color channel need to be turned into a vector by flattening each row of pixel values for a particular color channel. After applying the same step to each channel flattened, channel vectors should be appended one after another to get the final vector for the image. Given that the model size cannot change during training, each image should be resized to the same shape before representational vector is created (e.g. $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$).

Representation for CNNs, however, require different processing because the convolution operation requires locality of the data points to be taken into consideration. Therefore the most common approach is to represent each colour channel of the image as an array of vectors which commonly know as *matrix*. Because the coloured image has three colour channels each matrix for the channels should be combined as a multi-dimension matrix. This multi-dimensional matrix representation usually referred to as *tensor*.

Much like input data, the label for each image should also be represented in appropriate mathematical entity to be able to process in training. Because this problem has only two classifications most used method is to encode each class in binary (e.g. 1 for pneumonia and 0 for normal). However it can also be represented in one-hot encoding, which will return a vector for each label includes binary values for each class (e.g. $[0, 1]$ for pneumonia and $[1, 0]$ for normal).

3.3 Limitations of the Dataset

Although dataset I have chosen is well designed and validated, some aspects of data have undesired properties. First of such property is the size of the validation data. As I mentioned earlier dataset has a folder where it hosts validation files. Within this folder, there are sixteen images where each class have eight images each. This is significantly small validation dataset and would reduce the bins for each available accuracy step significantly. For example, making one error in the validation data will result in accuracy declining from 100% to 93.75%.

Second undesired property is the high variance of the image resolutions. Having such a high variance in image resolution makes it difficult to choose the right resolution to resize all the images. To show the variance within the data I have added a density plot that illustrates the height and width of the images and their distribution in Figure 3.6.

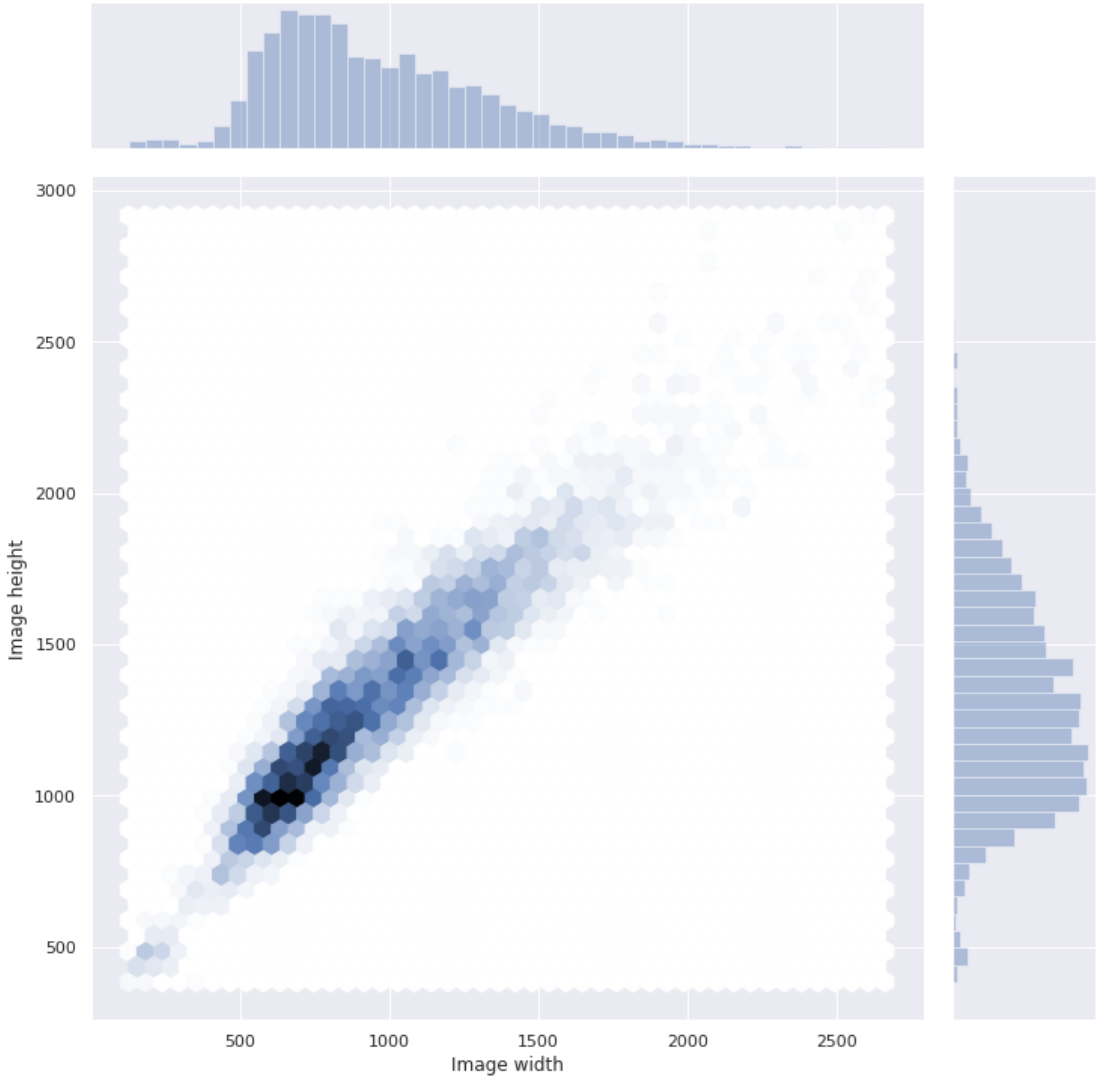


Figure 3.6: Density plot of image dimensions

This graph gives an insight into what resolution for the image should be chosen to have minimal informal loss and distortion in the final image. As we can see most of the images have shape above 500×500, therefore any resolution below this shape would be an appropriate choice for the training.

3.4 Data Processing

Inline with the information provided in section 3.2, I have designed two separate data processing pipeline. First pipeline created for data processing for so-called traditional machine learning algorithms such as Random forest or Support vector machine (SVM) that will be used in performance benchmarking. This pipeline does the flattening of the X-ray images discussed and depends on open-source software libraries keras [4] and numpy [23].

The second pipeline is created for pre-processing of CNN models which would be used in the majority of the time in training and testing. The module is designed to be lightweight with minimal dependencies. Only third party open source libraries used are, Numpy and the TensorFlow [1]. Although TensorFlow has high-level Keras module that allows easier implementation for data pipeline, I have decided on building data processing pipeline in tf.data module of the TensorFlow library. Building my pipeline in tf.data meant that I have to build all of the implementations from scratch myself but it also allowed me to build more flexible and more performant pipeline I could have had than the other alternative. The goal of having faster training runs and ultimately having shorter prototyping time only achievable if the training is fast enough to feed data into powerful hardware accelerators. Otherwise, the process will become a bottleneck in the training because the processing unit whether it is GPU or any other device will remain idle until the data fed into the system. Tf.data designed with parallelization in mind that carries out the process of fetching next data batch while the training for the current batch is in progress so the next batch can be ready as soon as the training is finished. Providing that all the augmentations and pre-processing will be done in CPU while GPU is training on the previous batch ultimately means all the augmentation calculations are computationally free.

At the beginning of this chapter, I point out that the dataset for this project has an imbalance problem. Building the data pipeline with the tf.data also meant that I am able to eliminate this issue by making a custom pipeline operation that creates a balanced training set. With the flexibility provided, I was able to apply the augmentations to only minority class of the dataset and add this new training example to training set until I have the same number of examples for both classes.

4 | Methodology

This chapter lays out the methodical steps I took while implementing the project. Main steps for implementing this project fall into four general categories. These four categories are:

- Establishing a baseline benchmark
- Improving metrics beyond baseline benchmark
- Ensuring model interpretability is maintained
- Deploying the final model to production

Further sections will provide detail for these four main categories.

4.1 Establishing a benchmark

The first step to every machine learning project is that model produced is performs better than any random or naive solution. Although random guess in any binary classification suggests that the minimum accuracy should be greater or equal than 50%, because of the imbalance in the test dataset, minimum accuracy requirement for this dataset is even higher. I have 390 pneumonia images in the 624 total images in the test dataset, dummy classifier that predict pneumonia for any given image will achieve 62.5% accuracy in this test dataset. Because of the imbalance in the dataset, accuracy would not be a good metric choice for assessing the performance. Precision (specificity) and recall (sensitivity) are a better choice of performance metrics in the field of medical research.

If I introduce these metrics in more detail, precision is calculated by dividing true positives (tp) in predictions to true positives and false positives (fp).

$$Precision = \frac{tp}{tp + fp} \tag{4.1}$$

The recall is calculated by dividing true positives to true positives and false negatives (fn).

$$Recall = \frac{tp}{tp + fn} \tag{4.2}$$

To capture the correct performance of the classifier we need to consider both precision and recall and F1 metrics allow us to capture that in a single metric.

F1 score provides the ability to consider both precision and recall for the same classification problem because it is calculated by getting a harmonic mean of the precision and recall.

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{precision \times recall}{precision + recall} \tag{4.3}$$

Remaining part for the benchmarking involves choosing which algorithms to train on the dataset. Given we don't know the distribution of the data, training fundamental machine learning algorithms together with the neural network algorithms is a cautious step to take. For that reason, I have chosen to train two fundamental machine learning algorithm, namely the Random Forest classifier and the Support vector classifier (SVC) for establishing a benchmark.

Previously I have mentioned in section 3.3 that the validation data for this dataset is very small. Having only 16 records in validation dataset prevent reliable feedback from the validation metrics. For that reason, I had opted in for using the training dataset as a validation dataset and evaluated the model performance in both the test and the validation dataset at the end of the report. Please note that the figures that will be reported in the next chapter are based on the validation data of a single model.

4.1.1 Random Forest Classifier

For the past years, tree-based algorithms have been very popular in the academic community as well as in the industry with numerous papers demonstrated in ICML, NIPS and JMLR. Random forest emerges in this category as a very strong algorithm by Breiman [3] that achieves remarkable performance in small to a medium dataset. Application of random forest involves choosing many hyper-parameters of this algorithm and the final baseline specification will be determined by finding an optimal number of estimators and maximum features using cross-validation.

4.1.2 SVM Classifier

In addition to Random Forest, the secondary mainstream machine learning algorithm is Support Victor Machine classifier [14]. The biggest factor for choosing this algorithm was its robustness in detecting non-linear features in the data using kernel trick. Similarly to Random Forest hyper-parameters of this model will be determined with cross-validation.

4.1.3 LeNet-5

LeNet-5 [20] is the first one of the well-known CNNs that I will apply to this problem to consider in baseline benchmark. I kept the attributes of the network as close to the origin network as possible but some characteristic of the network had to be changed when it is trained on this classification problem. The first difference in this project needed in input and output layers, MNIST dataset have

ten classes therefore required ten dense neurons at the output layer in the original network but in this project, there are two classes to predict which can be achieved by having one dense output neuron instead. Input layer also changed because 32×32 is not a suitable resolution for the pneumonia detection and changed to 224×224 for better representation. Another change made to this model is using relu activation function rather than hyperbolic tangent function (tanh) because of the less than ideal gradient flow in saturated tanh function prevented model to converge to a good solution. Details of the model summary given below for reference.

```
    Model: "LeNet-5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 220, 220, 6)       456
_________________________________________________________________
average_pooling2d_4 (Average (None, 110, 110, 6)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 106, 106, 16)      2416
_________________________________________________________________
average_pooling2d_5 (Average (None, 53, 53, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 49, 49, 120)       48120
_________________________________________________________________
flatten_2 (Flatten)          (None, 288120)            0
_________________________________________________________________
dense_6 (Dense)              (None, 120)               34574520
_________________________________________________________________
dense_7 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 85
=================================================================
Total params: 34,635,761
Trainable params: 34,635,761
Non-trainable params: 0
_________________________________________________________________
```

4.1.4 AlexNet

Similar to LeNet-5, AlexNet [19] also aimed to the kept original structure as much as possible. Original AlexNet comprises of eight layers, five of those were convolution layers where some of them connected to max-pooling layer. Despite the fact, the architect is preserved almost same as the original implementation additional steps such as adding local response normalization or PCA augmentation is not applied because of the ad hoc nature of the process and limited effect of this processes in the final performance. AlexNet is ultimately a very influential paper that steers the direction for how the CNNs are designed and fueled the adoption

of the use of neural networks in many fields. Fallowing quote from the paper also explains the reason why neural networks gain so much popularity and pushing the state of the art results year after year.

> *"All of our experiments suggest that our results can be improved simply by waiting for faster GPUs and bigger datasets to become available."*

```
Model: "AlexNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 54, 54, 96)        34944
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 26, 26, 96)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 26, 26, 256)       614656
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 12, 12, 384)       885120
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 12, 12, 384)       1327488
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 12, 12, 256)       884992
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_6 (Dense)              (None, 4096)              26218496
_________________________________________________________________
dropout_4 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_7 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_5 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 4097
=================================================================
Total params: 46,751,105
Trainable params: 46,751,105
Non-trainable params: 0
_________________________________________________________________
```

4.1.5 VGGNet

VGGNet is one of the best performant in 2014 ILSVRC competition that gets the best performance in classification and localization task. There are two different variations of this network is available namely the VGGNet 19 and VGGNet 16. For this project, 16 layered VGGNet 16 appears as a good choice because it has a sufficient capacity given that the dataset is relatively small and larger model likely to overfit to my project dataset. The main contribution of the VGGNet is that it demonstrated the importance of the network depth to good performance. The layer structure of the architecture is straightforward, only performs 3×3 convolution with 2×2 pooling. Similarly, for the purpose of the benchmarking architecture kept as close to original as possible with only change is made to final layer replaced with one neuron dense layer to accommodate the binary classification task. Advantage of using this network is that it is implemented in most of the modern software packages and is also available to initialize with the weights of imagenet. That property will be very useful for comparing the performance when transfer learning is explored in subsection 4.2.1. Detail breakdown of the network listed below.

```
    Model: "VGGNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
vgg16 (Functional)           (None, 512)               14714688
_________________________________________________________________
dense_3 (Dense)              (None, 4096)              2101248
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_4 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_3 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 4097
=================================================================
Total params: 33,601,345
Trainable params: 33,601,345
Non-trainable params: 0
```

4.2 Improving Performance

After establishing a benchmark for minimum acceptable performance, this step will focus on continuously improving the performance on the problem. Materialize that

goal I have chosen two methods, applying transfer learning and building custom neural network.

4.2.1 Transfer Learning

Transfer learning is fundamentally taking weights (or features) learned on one problem, and applying them to a similar or new problem that could benefit from it. Transfer learning usually considered when the training data is too small to train a deep neural network from a scratch. Process for transfer learning is simple, architecture is initialized with the weights learned from another problem without the top part of the model. Here the top part is referred to the layer(s) at the end of the model which has got the high level features specific to the problem. And given that the lower layers of the model contribute to simple features like vertical and horizontal lines or edges, weights in these layers held frozen to allow feature extraction from the images in new problem. After that step, the top part of the model initialized with the random weights to be trained on the new dataset and start learning higher-level features from that data. Transfer learning sometimes applied with the concept of *fine-tuning*, which is a process of unfreezing the part of the convolution layers or the entire model and re-training it with the new data.

Because the dataset of this project is suffering from a small data problem, transfer learning is utilized to discover if generic datasets like imagenet [5] can be helpful to improve the existing model. Effectiveness of the transfer learning will be measured by training the VGG16 network with the imagenet weights and comparing the performance against the same network with a randomly initialized version from the benchmarking experiments.

4.2.2 Custom Model Architecture

Another method available to get higher performance in machine learning is to designing custom architecture that better represent the data. Despite the great potential of better performance, this method is usually more complex in nature because of the challenges in hyper-parameter search. Modern CNNs have significantly more hyper-parameters compare to traditional machine learning algorithms. For instance, below items are a non-exhaustive list of hyper-parameters to consider when designing custom architecture.

- Learning rate α
- Momentum term β
- Number of layers
- Number of units in a layer
- Kernel size for convolution layers
- Type of the convolution layer
- Learning rate decay

- Type of optimization algorithm
- Regularization type and quantity
- Mini batch size

The complexity of the task gets significantly higher when the attributes such as convolution layers padding and strides included in the number of layer and kernel size. When used with valid padding convolutional layer have addition constrain that the further layers have to be compatible with the output tensor shapes from previous layers. Given these complexities and limited time frame of this project, some aspects of the hyper-parameter search will be constant to allow experimentation on other parameters. Grid search and random search are two of the widely used method when determining the right hyper-parameters. Studies suggest that random search is more efficient and computationally inexpensive than the grid search in the context of neural network [2].

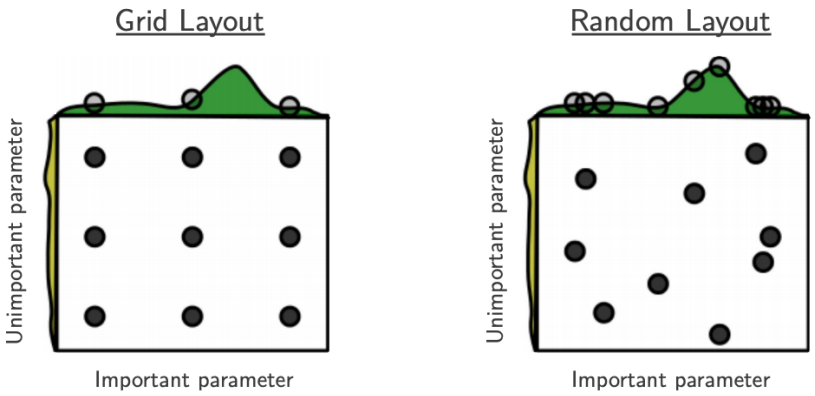


Figure 4.1: Random search versus grid search [2]

For the custom architecture, I will be running experiments with the random search over the following hyper-parameters.

- Type of convolutional layer. Conv2D versus Depthwise convolution
- Number of layers
- Different convolution types (e.g. Depthwise convolution)
- Type of regularization (e.g. Batch normalization, Dropout)

4.3 Model Interpretability

Machine learning models and a more specifically neural network usually regarded as black-box machines. Meaning that reason for their predictions are unknown.

However, there are a variety of techniques that design to elaborate the machine learning models emphasis points. Before any model put into production it should also be examined for their predictive reasons and because of that, I added model interpretability rules before making a decision to promote any model to deployment stage. One such technique call GradCAM [29] is designed to highlight the areas in the image that contributes the most for the prediction machine learning outputs.

4.3.1 GradCAM

GradCAM [29] is part of the general visualization technique called *class activation map* (CAM). It consists of creating a heatmap of activations over the input image space. A class activation heatmap is a two dimensional surface of scores for individual class mapped into the corresponding location. For instance, if the input image is with pneumonia, heatmap indicates the areas that most represent pneumonia-like features found and vice versa for images that absent from pneumonia. This technique originally designed for softmax output with different classes and to adapt it to binary classification with sigmoid output, but the integration to the sigmoid output is also possible because ultimately what algorithm needs is the probability of the certain class decision.

4.4 Deployments with CI/CD

Upon discovery of the best performer model up to this point, if the model satisfies the interpretation stage, can be added to model folders as a candidate for production release. Please note that due to the manual nature of the model interpretation stage this project utilizes continuous development rather than continuous deployment. Because it is not possible to assess this step without having a human intuition in the loop. After the insertion of the model to models folder, the test suite for the project will run and if the tests are successful, deployment script will automatically run to produce the model artificial needed for the deployment. The final step in this process is to commit the most recent artifact to the central code repository for deployment. This process repeats ever a time when there is a better performer model found. Detailed information about the deployment choice and the assessment of the deployment is covered in chapter 6.

5 | Design and Experiments

Summary of the experiment results and their details laid out in this section. Neural network experiment logs are published online for more detailed examination, they are accessible by the URL provided in the corresponding footnotes.

5.1 Training Specifications

Training for traditional machine learning used in this project is fallowing two-step approach. At the first part hyper-parameters for the model is selected by cross-validation score then in the second part model is trained on the entire dataset with the selected hyper-parameters. The final model then evaluated on the test set.

There are many artificial neural networks used in both in benchmarking and the model selection. Weights of the networks initialized by random except for the transfer learning models. The model used in the transfer learning experiments initialized with the weighs pre-trained on imagenet [5] dataset. Each architecture is trained using Adam [17] optimizer with the default parameters ($\beta_1 = 0.9$ & $\beta_2 = 0.999$). I used a batch size of 64 for the training, and default learning rate of 0.001 that decayed by the factor of ten every time validation loss stagnates at the end of an epoch. Each image in the dataset is downscaled to 224×224 shape and normalized to have values between 0 to 1. Data augmentation is applied to minority class in the dataset to create a balanced dataset as explained in section 3.4. Random cropping, changing saturation and horizontal flipping are set of augmentations chosen to create a balanced dataset.

5.2 Benchmark Experiments

5.2.1 Random Forest Classifier and SVC

Cross-validation results of the random forest classifier found the fallowing hyper-parameters achieves the best performance:

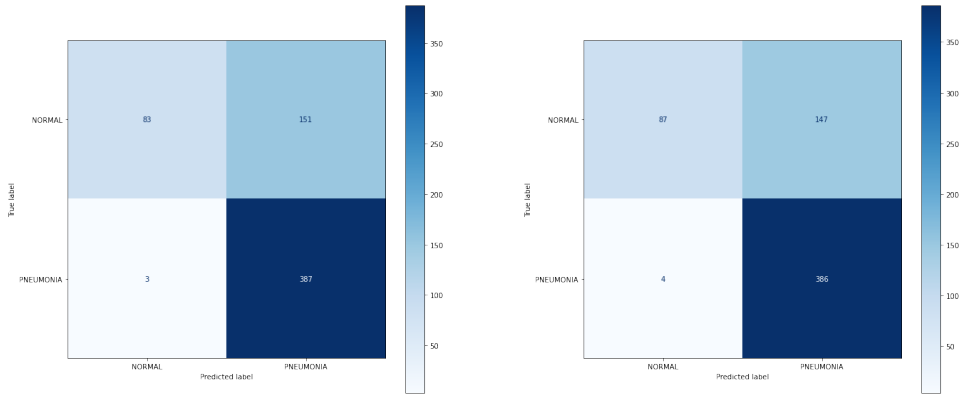
- Number of trees: 500
- Information gain criteria: Gini
- Number of max features: Square root

- Bootstrap: allowed

Cross validation for the support vector machine classifier runs suggested the fallowing hyper-parameters will achieve the best metrics:

- Regularization parameter: 10
- Kernel: rbf
- Polynomial degree: 3
- Kernel coefficient γ: scale
- Tolerance: 0.001

Confusion matrix of these classifiers evaluated on test data generated following results.



(a) Random forest (b) SVC

Figure 5.1: Confusion matrix of the classifiers.

It is clear from the amount of wrongly classified healthy patience X-ray, both of the models suffer from the imbalance effect. This effect is slightly more dominant in the random forest classifier than the SVC, albeit performance of both models are similar.

Classifier	Accuracy	Precision	Recall	f1
Random Forest	0.7532	0.3547	0.9650	0.5187
SVC	0.7580	0.3718	0.9560	0.5354

Table 5.1: Performance metrics for traditional machine learning algorithms.

5.2.2 AlexNet

During the experiments for AlexNet and LeNet5 effect of the balanced dataset [1] is also compared to base dataset [2] to measure the effectiveness of the data augmen-

[1] AlexNet Balanced: https://tensorboard.dev/experiment/7b6W0tdHQQ2RINGt6cYbrA/
[2] AlexNet Base: https://tensorboard.dev/experiment/PaBawCErSqG6zIef7JPq8g/

tation. Figures used in AlexNet and LeNet5 annotated with balanced training/-validation to reflect that balanced dataset is created with the data augmentation. Base training/validation is used to refer base image data is used without any data augmentation. Results from AlexNet was acceptable but given the loss values did not declined in the validation dataset suggest that ALexNet may have over-fitted the data.

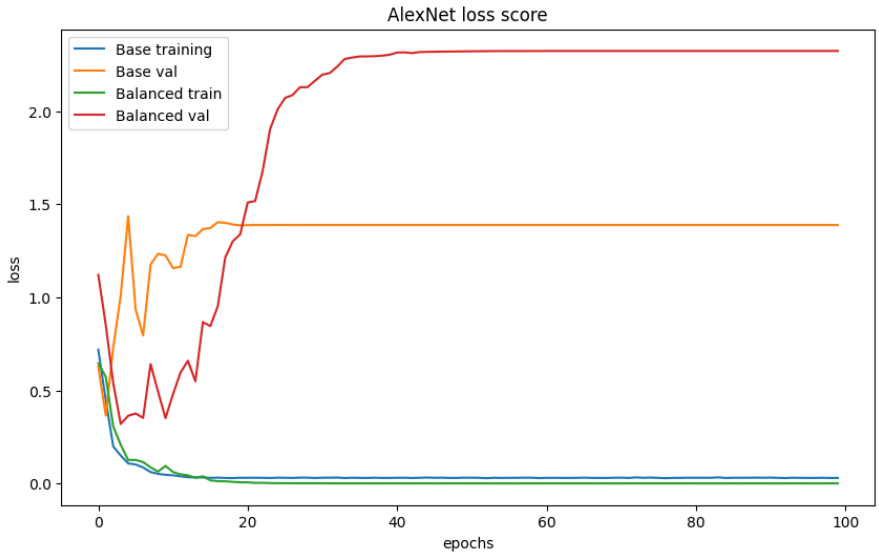


Figure 5.2

Another important point is that convergence was very quick for this dataset. It is clearly visible in the charts that model converges in around first 30 epochs and remained unchanged for the rest of the training.

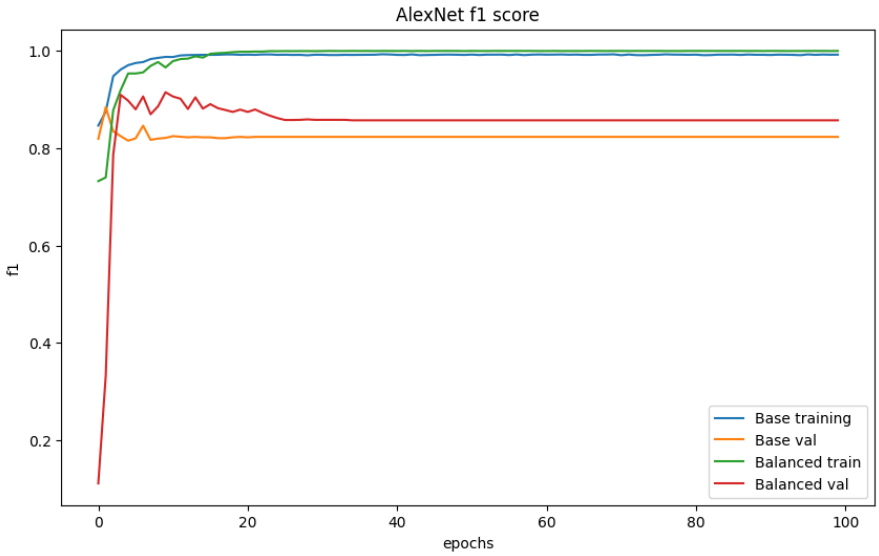


Figure 5.3

5.2.3 LeNet-5

I was not able to train the LeNet5 model with the original model specification that utilizes hyperbolic tangent function (tanh). This was partially expected as tanh activation function is not able to pass the gradient flow very well when the initialization is not favourable. This effect is mentioned in section 2.2 and as such, I have replaced the activation function with the ReLu activation function for my implementation. Similar to AlexNet, LeNet5 also displayed a possible sign of over-fitting when loss did not decline on the validation dataset.

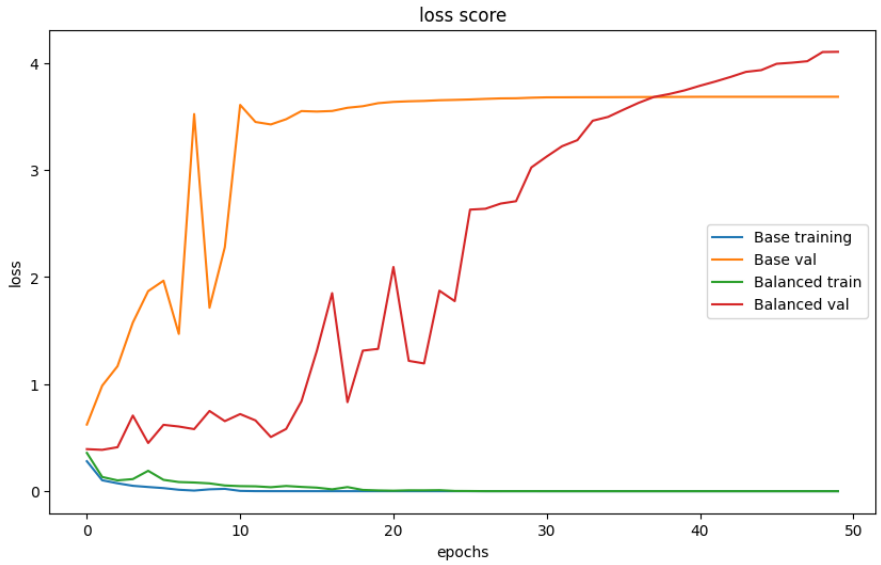


Figure 5.4: LeNet5 loss metrics

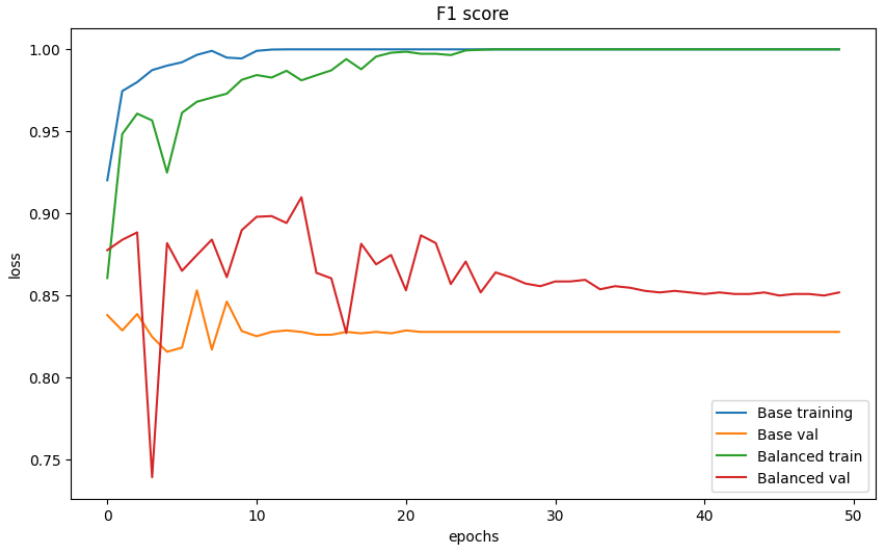


Figure 5.5: LeNet5 f1 metrics

LeNet5 performance was slightly below AlexNet's on the holdout data, yet it

is still a considerable performance given the algorithm is the oldest one among the neural networks.

5.2.4 VGGNet

Performance of this model [3] [4] is the worst among all of the model I tried. Disappointing performance didn't change with the different random initialization. However, training and validation loss suggested good training by declining in both training and validation data, accuracy metric was below random guess.

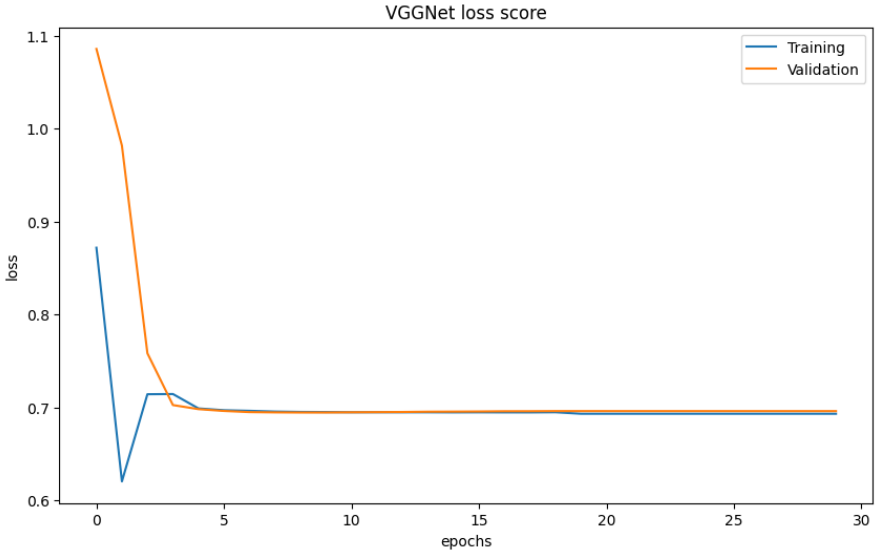


Figure 5.6

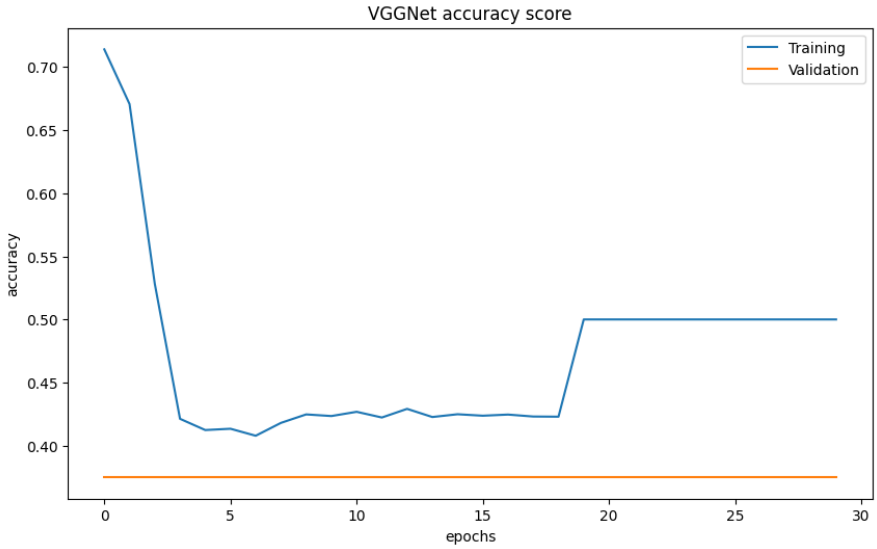


Figure 5.7

[3] VGGNet Base: https://tensorboard.dev/experiment/IbsPnocxSMKqNjmdJ3CdpA/
[4] VGG Balanced: https://tensorboard.dev/experiment/ABd8GrKdSXyzHgNPDngeEw/

5.3 Transfer Learning

I was very skeptical when adopting transfer learning to pneumonia detection because of the fundamental difference in the type of images in the imagenet compare to the project images. In this implementation weights in the convolutional part of the model initialized with the pre-trained VGGNet model from imagenet dataset. Convolution layer weights in this network are kept frozen during the training and allowed only fully connected layers at the end of the model to be trained. To my surprise, this approach worked considerably well. Achieving $\approx 77\%$ in accuracy and $\approx 87\%$ f1 on validation data.

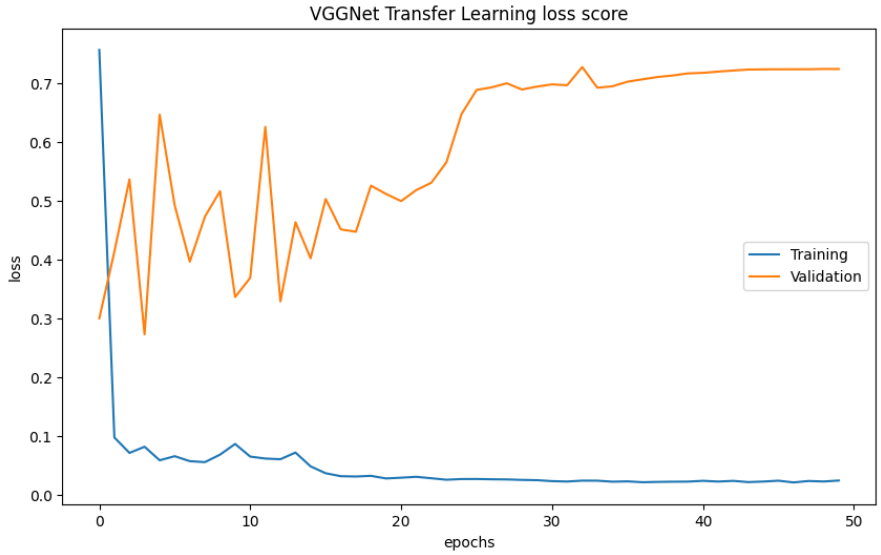


Figure 5.8

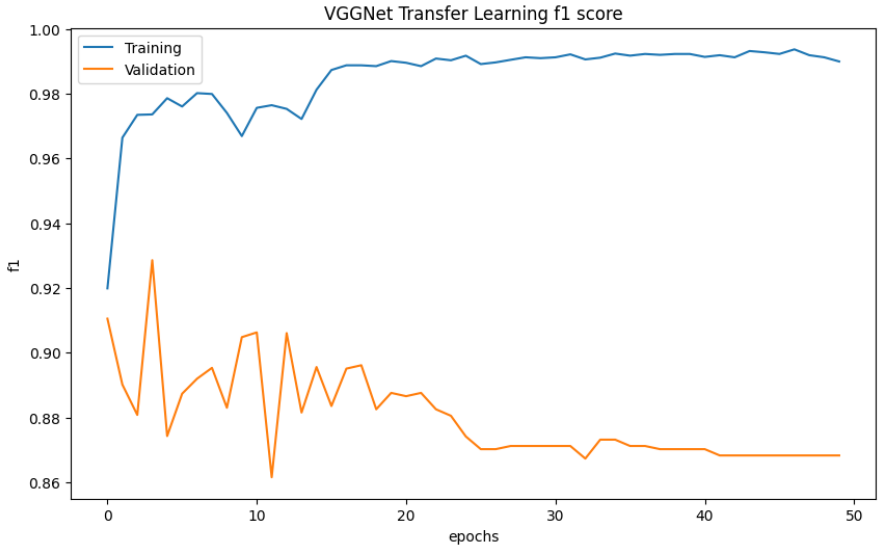


Figure 5.9

Although transfer learning has proven to be a viable method to increase model performance, fine-tuning attempts on transfer learning was unsuccessful [5]. After unfreezing top convolutional block layers and training the model, performance decreased to below 0.4 in the validation accuracy with a f1 score of zero.

5.4 Custom Neural Network Architecture

After experimenting [6] with a randomly chosen number of layers best combination I discovered is an architecture with ten convolution layers followed by batch normalization and max-pooling layer. In final experiments showed using two convolutional layers fallowed by batch normalization layer and max-pooling layer like a convolutional block is the best optimal choice for this architecture.

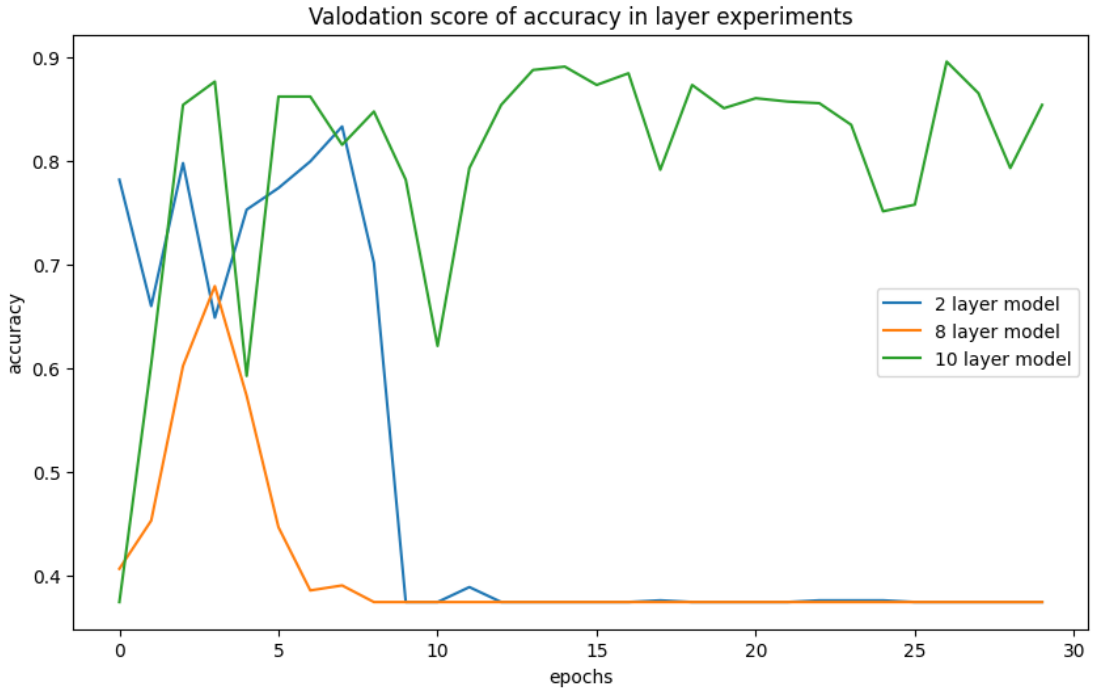


Figure 5.10

Immediately after this experiments, the same number of layers are maintained to determine the performance of the type of the convolution layer on the dataset. For the same number of layers, I replaced the convolutional layers with the depthwise separable convolutional layers [7]. Depthwise separable convolutional layers work by performing a depthwise convolution first then mixing that with the pointwise convolution. The result shows that convolutional layers perform better than depthwise convolutional layers whit the same layer numbers and other convolution properties set equally.

[5]https://tensorboard.dev/experiment/QRbFzlgwSGaf0lu5LLRSzA/
[6]https://tensorboard.dev/experiment/BDwUXqGgSVWA8Ieawcb3Rg/
[7]https://tensorboard.dev/experiment/aA2SgzCNTXWXK7kWMhyJNg/

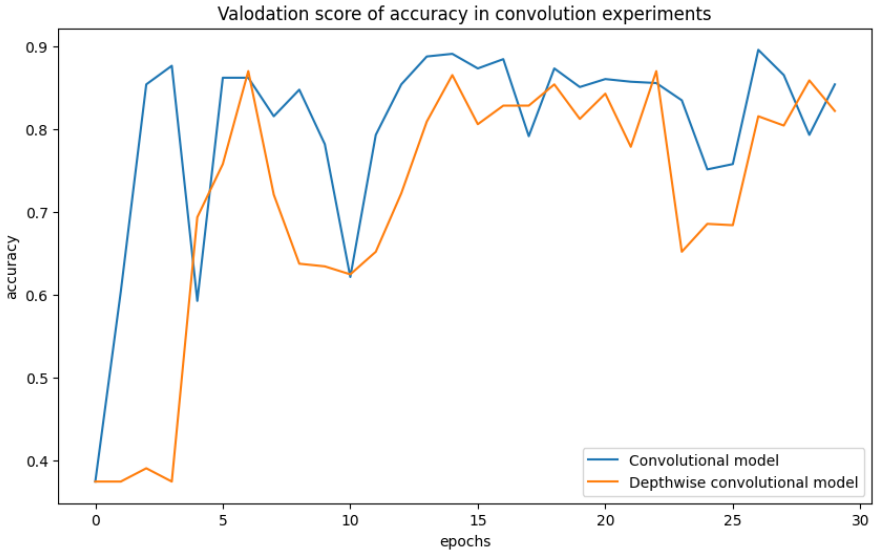


Figure 5.11

Lastly, I have started experiments [8] to observe the effect of regularization at the fully connected layers. Varying levels of dropout rate applied to dense layers to fine-tune the regularization level that best suits this architecture. Results show that the best level of dropout rate as 30%, albeit dropout rates in the effect of the dense layers change very little the performance scores of the model.

5.5 Interpreting Model Decisions

Application of GradCAM [38] provided valuable insight into understanding the features and parts of the images that lead to a certain type of prediction. In the case of images with the pneumonia presence models usually focused on the lung or lung segments which were expected. However, occasionally some predictions were based on regions that consist of bones that suggested that search for a better model should continue.

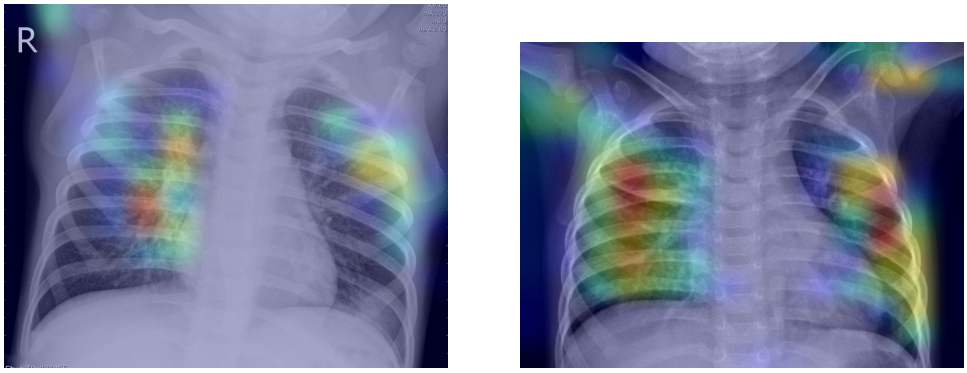
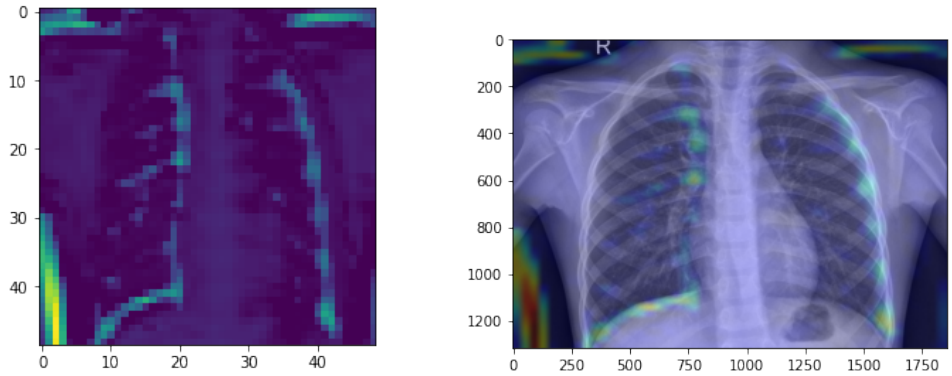


Figure 5.12: GradCAM result of the images with pneumonia diagnosis.

[8] https://tensorboard.dev/experiment/x8d0woI6Q7SpKo8zc0ZbSQ/

It was also noteworthy that in the case of absence of pneumonia X-rays models tend to focus on features like background or bone structures. One such example is the figure 5.13, that model gives emphasis on the background above shoulders and curved parts of the ribs of the patient.



(a) Stand alone heatmap

(b) Overlayed heatmap on image

Figure 5.13: GradCAM result and heatmap of the image without pneumonia diagnosis.

Model interpretability tool, in this case, is valuable for the decision making in which models to promote to production, especially in the case of selecting between models that have very close performance metrics.

6 | Model Deployments

Deploying machine learning model is integrating a machine learning model to an existing or new product to enable end-users to benefit from it. Although it ofter overlooked deployment is the most important part of the machine learning which without it artifacts only sits in a machine and occupy storage space. Only by deploying models, machine learning can create value to solve a specific problem. One of the reasons behind deployments being neglected is the complexities surrounding the deployment process. These difficulties include data collection, dependency management, feature engineering, versioning and etc. Another difficulty is choosing the right deployment method for the problem. Most popular ones among this large possibility of choices are maintaining a server to serve any prediction request or utilize serverless infrastructure serve prediction as they arrive. However, the options for deploying models have exploded in recent times with the increase in demand for intelligent systems. For this project, I have chosen to use a relatively new deployment option by serving the model via a static website. Serving the model this way is only possible because of the rich ecosystem of the TensorFlow [1]. Deployment relies on the JavaScript language and the specific framework called TensorFlow.js [31] to retrieve the model file and enable users input to make predictions on that model. A static website that host the model in this project is currently online and accessible via the url [1] of the website.

6.1 Why this deployment choice

Main reasons behind the decision of deploying machine learning model via a static webpage is two-fold. The first and most important reason is the financial cost of maintaining the model deployment. From the beginning, I wanted this project to be zero cost model to demonstrate that similar project that aims at social good and non-profit initiatives can achieve and maintain their goal with limited resources. This attribute also enables me to maintain deployment for a very long time. In fact, I am not planning to disconnect it in foreseeable future. The secondary reason is the transparency that comes from this deployment method. Deploying as a static webpage means that the user can see and read the source code of the page and explore which model makes the prediction and how it makes it.

[1] https://bbuluttekin.github.io/MSc-Project/

6.2 Critical evaluation of the deployment model

Every deployment type has strengths and weaknesses when compared to the other available choices. Therefore, it is imperative for me to discuss the pros and cons associated with the static website based deployments. I have previous mentioned some of the qualities that made this deployment model attractive in the previous section, however for the interest of full disclosure I have also listed them below. These qualities are:

- Very little or no cost of deployment.
- Deployment details are transparent
- Privacy, user data does not leave the users computer

First two item in this list explained in section 6.1, but it is also important to touch on the last item which is the privacy element of this deployment. Static website deployment works by bringing machine learning model to users browser and does not require connection to a third party server to receive the data. This will imply that the user data does not leave the users browser and their privacy is protected along with their data.

As oppose to these positive attributes there are some downsides to deploying machine learning model with this method. Some of these downsides are:

- Model is available to anyone for downloading
- Loading the model takes longer than other methods

There is no doubt that the transparency for a user to inspecting the model comes with a challenge for the companies or profit-driven initiatives about protecting the propriety software. Anybody that has access to the website can download and use the trained model for themselves. This issue becomes more apparent when considering that even competitors of an enterprise can access and used model for their own benefit. Considering all the above, this deployment model is not a good option for models that contain intellectual property. Another downside of this deployment is that its reliance on the resource of the user. Very complex machine learning models have millions to billions of parameters with many layers that contain them. All this complexity translates into bigger and bigger sized models that need to be deployed. Considering the model size for this project is around 200 to 600 MB implies there is a need to wait for user browser to load the entire model. Waiting time on this step can vary depending on the user's internet connection speed. This poses a significant challenge because most users would be unwilling to wait if they are not motivated to use the website. If the project is not important to the user this deployment option should be reconsidered by the developer.

6.3 Implementation steps

Implementation of this deployment can be broken down to three steps.

- Converting the saved model to applicable format
- Creating HTML file that will be served to users
- Building JavaScript file for user interactions

Model conversion step is the easiest step among other steps. Converting TensorFlow or Keras model to compatible format is provided by TensorFlow as a command-line tool and a straightforward process that can be completed at the start. Typical scenario for the usage of this model include, user visiting the website, user choosing and loading the image they would like to get predictions and finally clicking prediction button to receive the relevant prediction. Considering the scenario above HTML file is created with the input field, predict button, instructions and the spinner to inform the user that the model loading process is initiated.

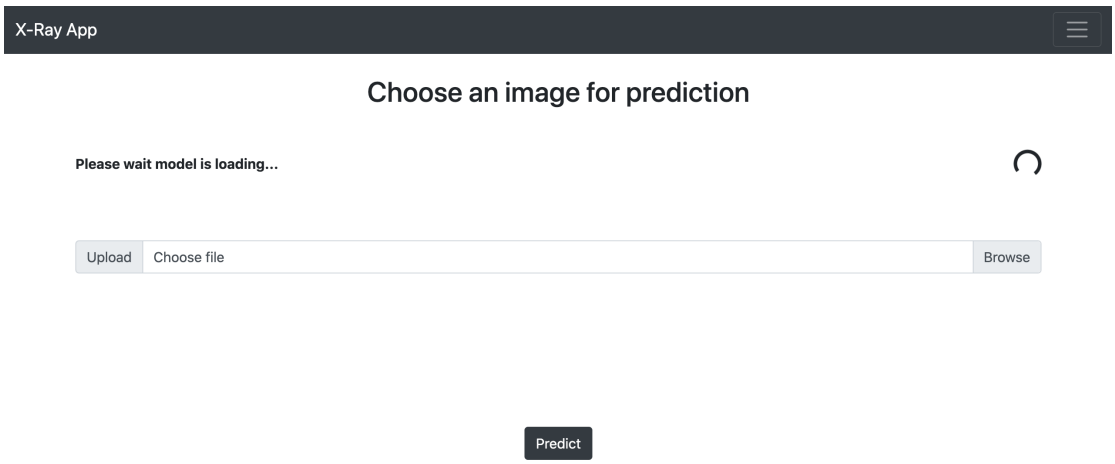


Figure 6.1: Main page for the deployment.

Because loading the model is required step that will take some time, loading of the model is started as soon as the user visiting the website and indicated to the user with spinner containing a message of the load process. The spinner will be hidden upon completion of the model loading, indication that page ready for prediction. At this point, user can select the X-ray image they want to run predictions which will get loaded to browser to insure user can verify the image.

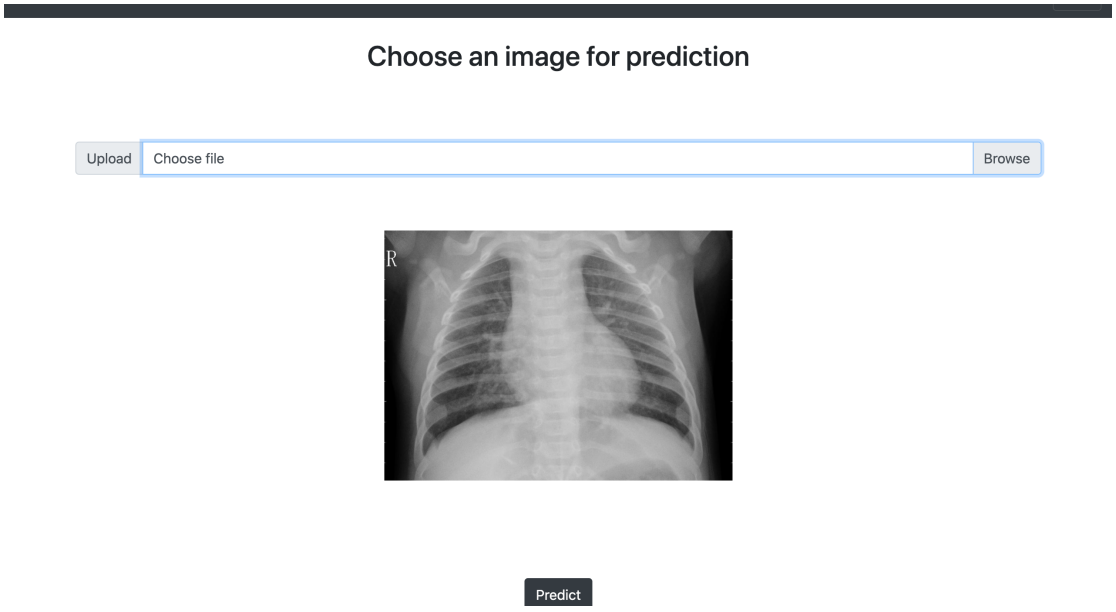


Figure 6.2: X-ray uploaded to website.

At this point, user will click predict button and that action will trigger prediction function in JavaScript. Background JavaScript code fetches the image loaded recently and with the help of build-in function turns the image to rank three tensor with RBG channels. After acquiring the tensor from the image next steps are part of the feature engineering process. Feature engineering in this app consists of resizing the image to expected dimensions by the model or in other words input shape of the model. Immediately after that step reshaped image pixels have to be normalized to a maximum pixel value that puts tensor values between 0 to 1. Final, because the original model is trained with batch data input data has to be a rank four tensor rather than three. Achieving that is possible with a simple step that turning one image to batch data of one incident which commonly referred to as expanding the dimensionality. Now that we have a right shaped tensor, predict method can be called on that image that returns a probability of image having class of *true*, in this case, pneumonia. In the final step, this probability will be translated into prediction by checking if the probability is less than or equal to 0.5 then the absence of pneumonia will be displayed otherwise prediction will be interpreted as the presence of pneumonia. Product of the prediction then displayed on the web page to the end-user along with the probability value as shown below.

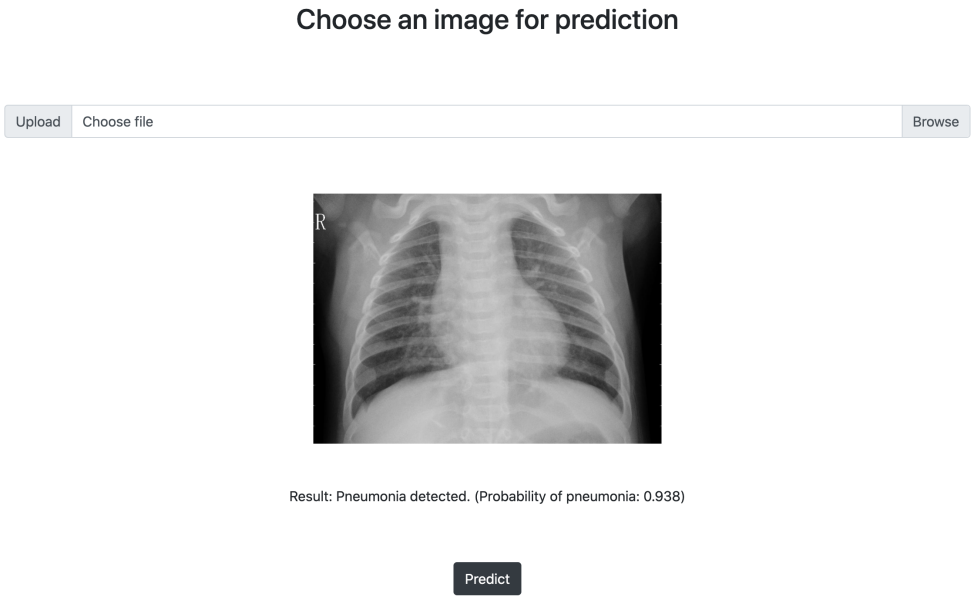


Figure 6.3: Prediction results displayed to user.

7 | Discussion and Conclusions

In summary, I have implemented a functional CI/CD pipeline to predict the presence or absence of pneumonia from X-ray images. Most of the steps I set out to achieve as part of this pipeline were successful with the exception of one. Originally, I have planned to implement ensemble method to increase the performance of the algorithms, but due to the limitations related to deploying multiple models into one static website, I had to abandon that step and decided to create a custom architecture instead. Nevertheless, the decision to building custom architecture also worked as expected and model created as a product of these experiments surpassed the performance of the benchmark and transfer learning experiments and promoted to deployment.

In addition to achieving my aim, I am also proud of some other design implementations that will be useful beyond this project. For example, the solution for resistant training module allows model training that lasts for days in a short term computational environment like Google Colaboratory. This solution can be used in any future project and disruption to training will not affect the progress of the experimentations. Generating a balanced dataset from imbalance data with data augmentation is another side achievement that can be applied to a wide variety of problems that suffering from data imbalance.

Before concluding this chapter, I would like to summarize the model performances. Despite the modest performance goal of this project, the final model performance in the testing was surprisingly good. The F1 score of the custom convolutional neural network model scored better than F1 score reported on CheXNet [26] and ChestX-ray8 [34]. However, this would not be a fair comparison as the dataset used in these papers and the methodology they conducted is different than what I applied in this project. I provided the final performance metrics for all of the benchmark models and best model from transfer learning and custom model architecture. All other models performance below selected models can be found in their corresponding experiment link provided in chapter 5. Metrics in table 7.1 are calculated using test dataset, but please keep in mind that dataset was also used as a validation dataset in training. That decision was based on not having enough data points in validation data which also explained in section 4.1. For the interest of full disclosure, I have tested the models on the very small validation dataset (16 records in total), because it was not used in training this result should act as performance in holdout dataset. Table 7.2 shows the result of that evaluation, please note that these results are very volatile and should be taken with a grain of salt. At the end making this difficult decision to use test data as a validation

taught me a valuable lesson, that I should always examine the dataset thoroughly before committing any work.

Classifier	Accuracy	Precision	Recall	f1
Random Forest	0.7532	0.3547	0.9650	0.5187
SVC	0.7580	0.3718	0.9560	0.5354
LeNet5	0.7869	0.7535	0.9795	0.8517
AlexNet	0.7949	0.7579	0.9872	0.8575
VGGNet	0.3750	0.0000	0.0000	0.0000
VGGNet (Transfer learning)	0.8109	0.7688	**0.9974**	0.8683
Custom ConvNet	**0.8542**	**0.8360**	0.9538	**0.8910**

Table 7.1: Performance metrics of selected algorithms on test data. (Also used during training as validation data.)

Classifier	Accuracy	Precision	Recall	f1
Random Forest	0.625	0.625	1.0000	0.7692
SVC	0.8125	0.2500	1.0000	0.4000
LeNet5	1.0000	1.0000	1.0000	1.0000
AlexNet	0.7500	0.6666	1.0000	0.8000
VGGNet	0.5000	0.0000	0.0000	0.0000
VGGNet (Transfer learning)	0.9375	0.8889	1.000	0.9412
Custom ConvNet	0.8125	0.7273	1.000	0.8421

Table 7.2: Performance metrics of all machine learning with holdout (validation) data.

7.1 Next Steps

This project can be extended with the suggestions from this section to improve many aspects of the project. I listed some steps below to point out what else can be done.

- Addition hyper-parameter tuning to increase model performance.
- Implementing GradCAM visualization to static website deployment so the users can see the reasons for the model predictions together with the prediction.
- Adding more tests to increase test coverage.
- Bigger datasets became publicly available after I started this project. Switching to one of those dataset benefits the performance.

References

[1] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305. URL: http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.html#BergstraB12.

[3] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A%3A1010933404324.

[4] François Chollet et al. *Keras*. https://keras.io. 2015.

[5] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[6] Timothy Dozat. "Incorporating Nesterov Momentum into Adam". In: 2016.

[7] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null (July 2011), pp. 2121–2159. ISSN: 1532-4435.

[8] Andre Esteva et al. "Dermatologist-level classification of skin cancer with deep neural networks". In: *Nature* 542 (Jan. 2017). DOI: 10.1038/nature21056.

[9] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*. 2010.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[11] Varun Gulshan et al. "Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs". In: *JAMA* 316 (Nov. 2016). DOI: 10.1001/jama.2016.17216.

[12] Kaiming He, Ross B. Girshick, and P. Dollár. "Rethinking ImageNet Pre-Training". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 4917–4926.

[13] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[14] Marti A. Hearst. “Support Vector Machines”. In: *IEEE Intelligent Systems* 13.4 (July 1998), pp. 18–28. ISSN: 1541-1672. DOI: 10.1109/5254.708428. URL: https://doi.org/10.1109/5254.708428.

[15] Mohammad Tariqul Islam et al. “Abnormality Detection and Localization in Chest X-Rays using Deep Convolutional Neural Networks”. In: *CoRR* abs/1705.09850 (2017). arXiv: 1705.09850. URL: http://arxiv.org/abs/1705.09850.

[16] Daniel Kermany and Kang Zhang. *“Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification”*. 2018. DOI: http://dx.doi.org/10.17632/rscbjbr9sj.2.

[17] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[18] Simon Kornblith, Jon Shlens, and Quoc V. Le. “Do better ImageNet models transfer better?” In: 2019. URL: https://arxiv.org/pdf/1805.08974.pdf.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[20] Yann Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[21] Minsky Marvin and A Papert Seymour. *Perceptrons*. 1969.

[22] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[23] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[24] *Open Access Biomedical Image Search Engine*. https://openi.nlm.nih.gov/. Accessed: 2019-03-12.

[25] Maithra Raghu et al. *Transfusion: Understanding Transfer Learning for Medical Imaging*. 2019. arXiv: 1902.07208 [cs.CV].

[26] Pranav Rajpurkar et al. “CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning”. In: *CoRR* abs/1711.05225 (2017).

[27] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”. In: *Psychological Review* (1958), pp. 65–386.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362. ISBN: 026268053X.

[29] Ramprasaath R. Selvaraju et al. "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". In: *CoRR* abs/1610.02391 (2016). arXiv: 1610.02391. URL: http://arxiv.org/abs/1610.02391.

[30] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: http://arxiv.org/abs/1409.1556.

[31] Daniel Smilkov et al. "TensorFlow.js: Machine Learning for the Web and Beyond". In: *CoRR* abs/1901.05350 (2019). arXiv: 1901.05350. URL: http://arxiv.org/abs/1901.05350.

[32] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[33] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[34] Xiaosong Wang et al. "ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases". In: *CoRR* abs/1705.02315 (2017). arXiv: 1705.02315. URL: http://arxiv.org/abs/1705.02315.

[35] Ashia C Wilson et al. "The Marginal Value of Adaptive Gradient Methods in Machine Learning". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4148–4158. URL: http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.pdf.

[36] D. H. Wolpert and W. G. Macready. "No Free Lunch Theorems for Optimization". In: *Trans. Evol. Comp* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1089-778X. DOI: 10.1109/4235.585893. URL: https://doi.org/10.1109/4235.585893.

[37] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3320–3328. URL: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

[38] Bolei Zhou et al. "Learning Deep Features for Discriminative Localization". In: *CoRR* abs/1512.04150 (2015). arXiv: 1512.04150. URL: http://arxiv.org/abs/1512.04150.