Using Machine Learning to Diagnose Pneumonia from Covid-19 in Patients from Chest X-Rays

James Lamb

Student Ref:10558719

BSc Mathematics with Theoretical Physics

School of Engineering, Computing and Mathematics

University of Plymouth

May 4, 2021

Contents

1 Fully Connected Neural Networks

1.1 Neurons and Networks

Historically, the nodes of a neural network were modelled after the neurons in the brain, hence the nomenclature. The fundamental concept being that each neuron is essentially a continuous function that maps its output $y \in [0,1]$. We consider some activation function Ω which defines whether or not this neuron "fires", and some weight function, w, to model the synaptic plasticity of biologically connected neurons. This neuron can then be described,

$$y = \Omega(\sum_{j=0}^{n} w_j x_j) \tag{1.1}$$

Note that the j subscript denotes each individual input value or "signal" connected to the neuron. So, extending this to a layer of multiple neurons; intuitively we have,

$$y_i = \Omega(\sum_{j=0}^{n} w_{j,i} x_j)$$

This describes a fully connected layer of neurons if $\{x_j\}$ contains every input value. Further, if we consider the set of outputs of this layer as the set of inputs for another subsequent layer of neurons, prescribed by its own individual weights and possibly its own activation functions[1] then we have a fully connected neural network, sometimes referred to as a dense neural network (in reference to the large number of connections between each neuron). It is important that we include a bias term within these functions also. Without it our model is restricted to learning a function that specifically passes through the origin which limits its capability to model even simple functions (Gebel, 2020). Typically, this is introduced as a "bias neuron", an extra neuron in each layer that does not take any input value but stores some constant multiplied by a weight as an input for further layers. This weight is learned just as every other weight value is learned by the model. The effect of this allows the estimated function learned by the model to shift appropriately, this is shown graphically in Figure 1.1.

[1] In the case of the utilisation of a parameterised channel-wise activation function within the model.

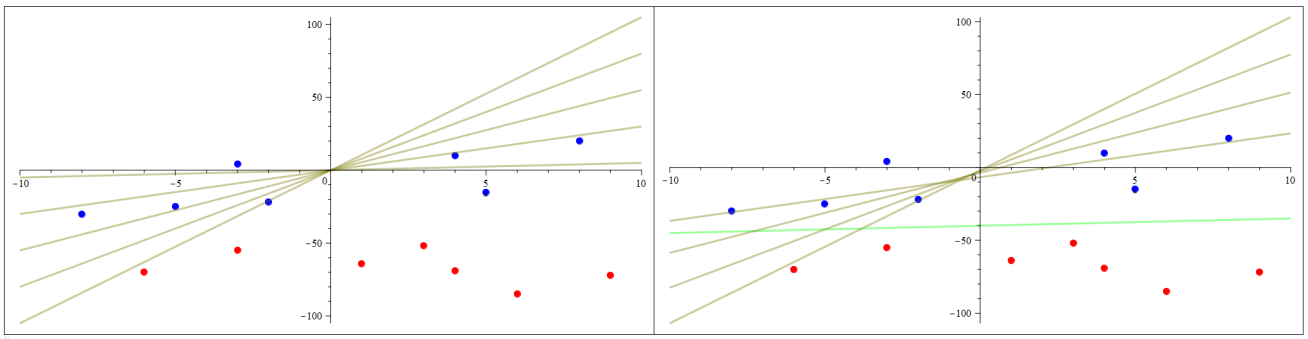


Figure 1.1: Consider an arbitrary binary classification problem, red or blue. The graph on the left shows linear representation without a bias term, the model can not find the linear function between the blue and red data points as it is restricted to passing through the origin. On the right graph, a bias term is included that allows the linear function to "shift" between the data points.

1.2 Non-Linearity

First let us consider a simple example of a neural network foregoing the use of an activation function, shown in Figure 1.2.

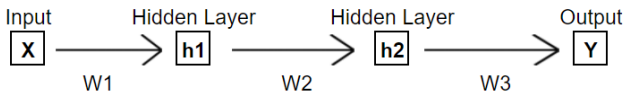


Figure 1.2: A simple neural network example.

Using the definition given for a neuron (1.1), and define the output of a hidden layer i to be h_i. We add bias terms for layer i, b_i, and omit the activation function, then we can describe this network as,

$$\begin{aligned} y &= h_2 \times w_3 + b_3 \\ &= (h_1 \times w_2 + b_2) \times w_3 + b_3 \\ &= h_1 \times w_2 \times w_3 + b_2 \times w_3 + b_3 \\ &= (x \times w_1 + b_1) \times w_2 \times w_3 + b_2 \times w_3 + b_3 \\ &= x \times w_1 \times w_2 \times w_3 + b_1 \times w_2 \times w_3 + b_2 \times w_3 + b_3 \\ &= x \times w_1 \times w_2 \times w_3 + b_1 \times w_2 \times w_3 + b_2 \times w_3 + b_3 \end{aligned} \tag{1.2}$$

Now we can combine each of the bias terms into a matrix, B and considering w_i to be some linear transformation and knowing that a combination of linear transformations is itself a linear transformation we can write them in a matrix, W, and rewrite equation 1.2 as,

$$y = x \times W + B. \tag{1.3}$$

Hence, any number of hidden layers with linear parameters will still result in a linear regression problem. This alone prevents the model from learning more complex functions. Further, a multi-neuron model is essentially no different to a single neuron model, hence adding more layers will not increase the efficacy.

To combat this, an activation function is used to adjust or limit the output and in doing so achieves non-linearity in the model parameters. There are a number of commonly used activation functions, we shall discuss a few to highlight the properties that are important when choosing a function.

1.2.1 Types of Activation Function

Heaviside Step Function

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}, \qquad \frac{d}{dx} f(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ undefined & \text{if } x = 0 \end{cases}$$

The simplest of activation functions; the binary step function activates a neuron in the model if positive and suppresses the neuron otherwise. This function is clearly suited to a simple binary classification model but is unsuitable for a multi-value classification system.

Tanh Function

$$f(x) = \frac{e^x - e^x}{e^x + e^x} \qquad \frac{d}{dx} f(x) = 1 - f(x)^2$$

The hyperbolic tangent function is essentially a scaled sigmoid function; rather than returning an output value $0 \leq y \leq 1$ it returns a value $-1 \leq y \leq 1$, meaning it has a stronger gradient and thus will train more aggressively. Furthermore, being a zero-centered function makes it a highly suitable choice for model inputs with neutral, strongly positive and negative values.

This has been a popular choice for non-linearity in machine learning due to the fact that it is non-linear in nature and has a smooth gradient. Also, because this gradient is steep, small changes in the input will result in a significant change in the output and hence brings the activation value to either side of the curve, making distinct predictions for a classification model.

It is also monotonic, so it will give better performance during the back propagation step of the model training.

There is a significant drawback to the Tanh activation function, however. On the extreme ends of the curve, the gradient becomes very small[2] which means as values tend further from 0, the change during back propagation (the gradient) tends toward 0. This can result in the network refusing to learn or becoming extremely slow in doing so.

[2] Also referred to as a vanishing gradient.

ReLu

$$f(x) = \max(0,x) \qquad \frac{d}{dx}f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ undefined & \text{if } x = 0 \end{cases}$$

The ReLu (Rectified Linear Unit) function has gained popularity in recent years due to its computational advantage over other activation functions. This is due to the fact that it does not activate every neuron simultaneously. Namely, if the linear transformation of the output is less than 0 then it is not activated (Pedamonti, 2018). Further, those that are activated do not then involve more complex computational methods such as division.

Another advantage to the function is that it is easy to optimise using gradient-descent due to the fact that it is otherwise linear.

It is important to note, however, the drawbacks of this function. With all values for $x < 0$ mapping to 0, the negative portion of the function then has a derivative of 0. As mentioned previously in discussing Tanh, this can easily result in the vanishing gradient problem; dead neurons that refuse to learn further. Although the effect is somewhat diminished in regards to positive values, it is certainly more pronounced for the negative portion.

Modified versions of the Relu function have been defined to address this issue:

- PReLu (Parameterised Relu) -

$$f(x) = \begin{cases} \alpha_i x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \qquad \frac{d}{dx}f(x) = \begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

  - Formally, if $\alpha = 0.01$ then this function is the Leaky Relu function. The aim of this function is to eliminate the vanishing gradient for $x < 0$. The range, then, of the function is no longer bounded to a minimum of 0, but retains the majority of its computational power. This may also introduce a new parameter in the model for training, α. Typically this function is used when the Relu function fails to train a model effectively.

    In the case that the parameter a is defined as a learnable parameter, the function is formally a PReLu function. There are two case distinctions to make when this is the case; Whether the parameter is distinct for each channel or whether the parameter is common to all channels. This is referred to as channel-wise or channel-shared PReLu, respectively. It is important to note that the increase in the number of parameters in regards to the channel-wise case also increases the computational cost of the model, but as the number of parameters introduced (equal to the number of channels) is vastly less than that of the remainder of the model this cost can be disregarded as negligible.

- SReLu (S-Shaped ReLu) -

$$f(x) = \begin{cases} t_i^r + a_i^r(x_i - t_i^r) & \text{if } x_i \geq t_i^r \\ x_i & \text{if } t_i^r > x_i > t_i^l \\ t_i^l + a_i^l(x_i - t_i^l) & \text{if } x_i \leq t_i^l \end{cases} \qquad \frac{d}{dx}f(x) = \begin{cases} a_i^r & \text{if } x_i \geq t_i^r \\ 1 & \text{if } t_i^r > x_i > t_i^l \\ a_i^l & \text{if } x_i \leq t_i^l \end{cases}$$

  - Consisting of three piecewise linear functions, the SReLu function introduces four trainable parameters in the model (Jin et al., 2015). t_i^l, t_i^r represent the threshold between the central and left or right functions and a_i^l, a_i^r the slope of the left and right function lines respectively. Similar to the PReLu function, these extra parameters ($4N$, where N is the number of channels, if channel-wise variant is used) increase the computational cost but as previously argued, the number remains negligible in comparison to the model as a whole.

The biggest advantage of using a modified ReLu function is in avoiding the dying neuron problem, further, parameterisation of the activation function allows a model to train more efficiently and reach greater accuracy at a minimal cost of computation time. It is worth noting though, in models that do not suffer from neuron necrosis, use of a modified Relu function is inefficient in comparison.

1.3 Learning in Dense Networks

Dense networks use the method of backpropagation in order to minimise error in its prediction. The network as described in the previous section is what is known as a feed forward network, information is passed and processed forward through the layers of the network to give the predicted output value, y. Backpropagation is an algorithm that essentially compares this given output to the expected value, $\hat{y}$, and adjusts the weights of the neurons proportionally to the respective derivative of the error. This is to minimise the overall cost function and hence produce more accurate results. It can be described in four steps (Cilimkovic, 2015):

Step 1: Feed-forward computation:

This is as described previously, only we now formally refer to layers with weight and activation functions as hidden layers. The original input is passed to the hidden layers whose function is y_i, the resultant values are fed forward to any further hidden layers and eventually to the output layer neuron, to give the prediction value.

Step 2: Back propagation to the output layer:

Firstly the error, the result of the model's cost function, is calculated, typically given as the mean squared error,

$$E = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1.4}$$

Intuitively it is clear that in minimising the cost function we inherently increase the accuracy of the model. In order to do this we need to understand the measure of change in the cost function in relation to a specific weight function, bias or activation (Hansen, 2020). First, for simplification, we incorporate the bias neurons into the weights, such that,

$$\begin{gathered} w_{0,i}^k = b_i^k \\ \longrightarrow a_i^k = b_i^k + \sum_{j=1}^{n_k-1} w_{j,i}^k x_j^{k-1} = \sum_{j=0}^{n_k-1} w_{j,i}^k x_j^{k-1} \end{gathered} \tag{1.5}$$

Where, n_k denotes the number of neurons in layer k. Then, considering that the derivative of a sum of functions is equal to that of the sum of the derivatives of each function, we can consider the derivation for a single input-output pair and derive a general form for all pairs. So,

$$C = \frac{1}{2}(y_i - \hat{y}_i)^2 \tag{1.6}$$

and by the chain rule,

$$\frac{\partial C}{\partial w_{j,i}^k} = \frac{\partial C}{\partial a_i^k}\frac{\partial a_i^k}{\partial w_{j,i}^k} \tag{1.7}$$

Here, a simply represents the sum of the products plus bias before applying the activation function. Now, as,

$$\begin{gathered} y = \Omega(a_i^k) \\ \Rightarrow \frac{\partial C}{\partial a_i^k} = (y - \hat{y})\Omega'(a_i^k) \end{gathered} \tag{1.8}$$

and,

$$\frac{\partial a_i^k}{\partial w_{j,i}^k} = x_j^{k-1} \tag{1.9}$$

So we have the derivative with respect to the weights given for the final layer,

$$\frac{\partial C}{\partial w_{j,i}^k} = (y - \hat{y})\Omega'(a_1^k)x_j^{k-1} \tag{1.10}$$

Step 3: Back propagation to the hidden layer/s:

Extending this to further layers, by the chain rule, we have,

$$\frac{\partial C}{\partial a_i^k} = \sum_{n=1}^{n_k+1} \frac{\partial C}{\partial a_n^{k+1}}\frac{\partial a_n^{k+1}}{\partial a_i^k} \tag{1.11}$$

The bias term is not dependant on previous layers so it is not included in this summation. It follows that,

$$\frac{\partial a_n^{k+1}}{\partial a_i^k} = w_{j,i}^{k+1}\Omega'(a_j^k)$$

$$\Rightarrow \frac{\partial C}{\partial a_i^k} = \Omega'(a_j^k)\sum_{n=1}^{n_k+1} w_{j,n}^{k+1}(y-\hat{y})\Omega'(a_j^{k+1}) \tag{1.12}$$

and

$$\frac{\partial C}{\partial w_{j,i}^k} = \frac{\partial C}{\partial a_i^k}\frac{\partial a_i^k}{\partial w_{j,i}^k} = \Omega'(a_j^k)x_j^{k-1}\sum_{n=1}^{n_k+1} w_{j,n}^{k+1}(y-\hat{y})\Omega'(a_n^{k+1}) \tag{1.13}$$

It is from here that the term backpropagation earns its nomenclature, the error calculated at a layer, k, depends on the error calculated at the next layer, $k+1$. Whereas the output layer's error is calculated only from the predicted output and the expected output, all previous layer's are essentially then a weighted product sum of this error scaled by the derivative of the activation function iterated back through the layer's to the input layer.

Step 4: Weight updates:

Combining these results and applying a scaling factor equivalent to the learning rate of the model,

$$\Delta w_{i,j}^k = -\alpha[(y-\hat{y})\Omega'(a_1^k)x_j^{k-1} + \Omega'(a_j^k)x_j^{k-1}\sum_{n=1}^{n_k+1} w_{j,n}^{k+1}(y-\hat{y})\Omega'(a_n^{k+1})]. \tag{1.14}$$

Here, α denotes the learning rate of the model. This is a hyper-parameter that restricts the magnitude of the change applied to every weight. This is generally a very small number but it's optimum value depends on the data and the model (This is discussed further in section 3.2).

It is conventional that the individual summation terms and the final layer's gradient are collected as a vector which is then applied to each weight (including bias) of the model. It is also conventional that the weights of the model therefore also be stored as a matrix (See section 2.1.1), this not only simplifies the application of the updates but specifically optimises the update for GPU computation.

This algorithm continues until the error function becomes sufficiently small. It is also a general convention that the algorithm is performed on small batches of the data at a time in order to improve the performance and computational time of the model (Paeedeh and Ghiasi-Shirazi, 2020).

2 Overview of Convolutional Neural Network

2.1 Elements of a CNN

2.1.1 Convolutions

The main difference of a Convolutional Neural Network (CNN) over other neural networks is the use of convolution, a linear operation used for feature extraction. CNNs have been found to be very useful in image classification problems. Formally, the convolutional formula (Albawi, Mohammed, and Al-Zawi, 2017) can be given as,

$$G[i,j] = f * k[i,j] = \sum_m \sum_n k[m,n] \times f[i-m, j-n] \ .$$

$$f = Image$$

$$k = Kernal$$

Consider an input image of 16×16 pixels with RGB channels. In a typical neural network, to connect this input to a single neuron would require $16 \times 16 \times 3 = 768$ weighted connections. To reduce this number, we define a "kernel" (perhaps several) as an $n \times m$ matrix [1] to isolate local regions of the input image.

The kernal can be visualised as a kind of view-port sliding over the image. This means an assumption can be made in that for each local region we use a duplicate neuron, the weights applied are remain fixed and identical across each neighbouring neuron, thus reducing further the number of parameters. Further, by considering that the filters which are intended to be applied to the input image can be represented in matrix form then the kernal itself can be used as a filter. To highlight this, consider a 1-dimensional convolutional layer wherein every neuron can be described in the function,

$$\Omega(\sum_{i=0}^{n}(w_i x_i) + b) \ . \tag{2.1}$$

Now consider the weight matrix, W, which consists of each weight element, w_i. In the case of a regular ANN, shown in (2.2), every input pixel is connected to each neuron with a different weight. However within a convolution layer (2.3), many of the neurons are disconnected and therefore have zero value,

[1] typically $n = m$, unless specific features of the input image are known prior to processing for which $n \neq m$ would be computationally optimal.

and due to the multiple copies of neurons we have the same weight functions applied in differing positions.

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & & \\ \vdots & & \ddots & \\ w_{n,0} & & & w_{n,n} \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} w_0 & w_1 & 0 & 0 & \dots \\ 0 & w_0 & w_1 & 0 & \dots \\ 0 & 0 & w_0 & w_1 & \dots \\ 0 & 0 & 0 & w_0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.3}$$

Hence each output neuron only has an interaction with the input pixel within the convolutional layer's kernal, and those weight parameters are shared across each neighbouring layer, vastly reducing the number of parameters requiring optimisation.

The fact that the weights are shared means that the features of an image are then considered to be spatially invariant. This implies that convolution may not be appropriate for applications where feature positions in the input image are important.

2.1.2 Stride

Simply put, the stride defines the pixel distance the kernal must translate to process each local region of the input image. By setting this value, and considering the chosen kernal size, the overlap of each region is also defined along with the size of the output (Krizhevsky, Sutskever, and Hinton, 2017).

Consider a $N \times N$ image with a $K \times K$ kernal. Choosing the stride to be S, then the overlap is given by $K - S$ and the size of the output matrix is given by,

$$O = \frac{N-K}{S} + 1 \ .$$

Note that if stride and kernal are chosen such that $\frac{N-K}{S} \notin \mathbb{Z}$, then the remainder of the image pixels are not parsed by the kernal and do not factor into the output, hence the output size is more accurately given by,

$$O = \lfloor \frac{N-K}{S} \rfloor + 1 \ .$$

This can be further extended to three dimensions such that the process is repeated for each channel

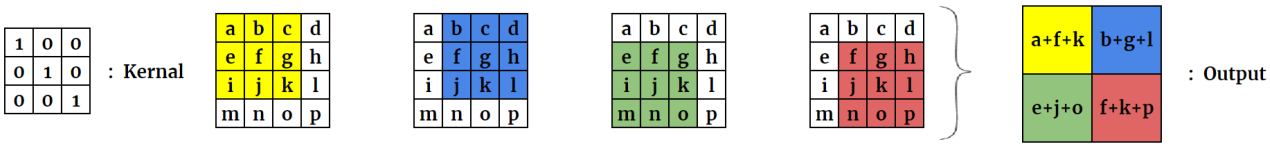


Figure 2.1: Kernal of order 3 acting on 4×4 image

separately and the resulting 2-dimensional outputs are collated into a single 3-dimensional tensor, ie.

$$O = [N,N,N_c] * [K,K,N_c] = [\lfloor \frac{N-K}{S} \rfloor + 1, \lfloor \frac{N-K}{S} \rfloor + 1, N_f] \ .$$

$$N_c = \text{Number of image channels}$$

$$N_f = \text{Number of filters}$$

A brief consideration of higher dimensions can be made. The fundamental amendment to the procedure here is in using a 3-dimensional tensor as a kernal. This acts on the input in a similar manner as before only it now slides in three dimensions, hence the output is also reduced in three dimensions.

2.1.3 Padding

When a convolution layer is formed as discussed, the kernal captures information and collates it into a tensor, as shown in figure 2.1. This typically results in a loss of information at the border of the local region.

In order to retain this information, a method called zero-padding[2] is utilised wherein an artificial border of zero value is added to the input matrix, as graphically depicted in Figure 2.2. Using this method prevents the output size reducing with the depth of the convolution layers and inherently then allows the network to have any number of convolutional layers.

It also permits further management of the size of the output, with the modification to the formula for output size given by,

$$O = \lfloor \frac{N-K+2P}{S} \rfloor + 1 \ .$$

Where P is the number of padding appended. Or the 3-D variant,

$$O = [N,N,N_c] * [K,K,N_c] = [\lfloor \frac{N-K+2P}{S} \rfloor + 1, \lfloor \frac{N-K+2P}{S} \rfloor + 1, N_f]$$

[2]Commonly referred to as "same" padding. Conversely, "valid" padding refers to convolution without any padding appended to the input image.

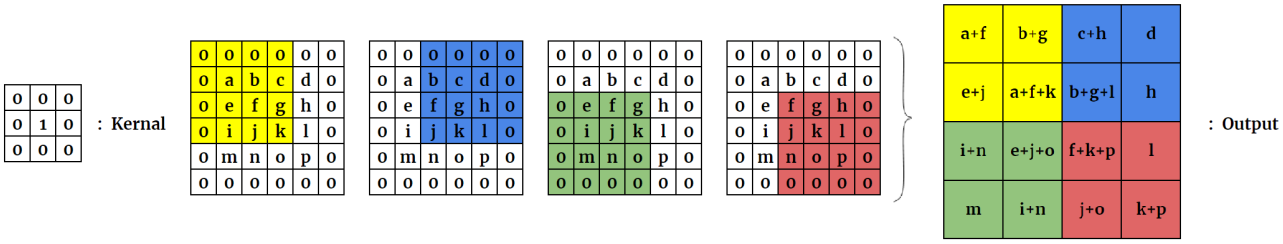


Figure 2.2: Kernal of order 4 acting on 4×4 image with zero-padding

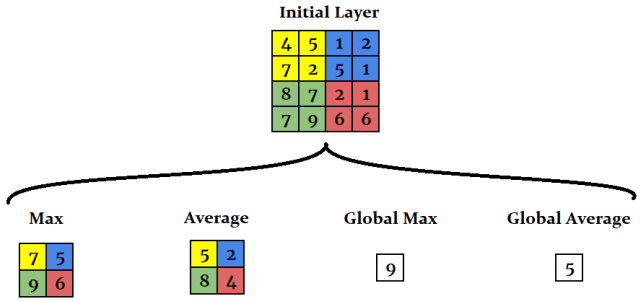


Figure 2.3: Depiction of different pooling outputs in regards to a sample initial layer

2.1.4 Pooling

Pooling, much like valid padded convolution, is a process utilised to reduce the sampling rate of its proceeding layers. Depending on whether max or average pooling is used this reduction is equivalent to either down-sampling or decimation, respectively. This acts to reduce noise in the input layers and secondly to speed up computation (Gholamalinezhad and Khosravi, 2020).

Pooling also has its own stride parameter which, much like the stride parameter for the kernal, defines how many pixels across the tensor moves each time an element of the output tensor is evaluated and again inherently defines the output size and overlap. In contrast however, it is not typical to have an overlap in the pooling layer, although it has been noted to provide more accurate results in the model. In either case, much like the kernal, a tensor evaluates a set of sub-regions of a layer and returns a new tensor consisting of values representing each of those sub-regions.

The most commonly used pooling technique is max pooling. The sub-region is evaluated such that the maximum value within the region is returned and the remaining values are discarded.

ie. Consider the set of sub-regions of the layer, $\mathfrak{s}$, then

$$p_{i,j} = \max x_{p,q} \; . \qquad\qquad x_{p,q} \in \mathfrak{s}_{\mathrm{i,j}}$$

The second pooling technique to consider is average pooling. In this case the average of all of the values within each sub-region becomes the relative element of the pooled output layer.

Both of these pooling techniques can also be extended to what is called global pooling. Instead of returning a tensor, a single number is returned, taken from either the maximum or average of all

the elements in the layer. This technique is more appropriate in applications such as natural language processing rather than purposes such as machine vision.

Each of these pooling techniques are shown graphically using an example layer in Figure 2.3.

3 Evaluation of the Model

Machine learning models generally do not work "out-of-the-box". There are a number of values to be modified in order to fine tune the performance of a model such that it is effective. Theses values are known as "hyperparameters" and refer to values which cannot be learned by the estimator of the model itself. Conversely, the term parameter refers to the values estimated within the model, although often "parameters" is used as an umbrella term for both hyperparameters and model parameters.

A large part of a model's effectiveness is then determined by the choice of these hyperparameters, it is then worth discussing the methods by which we come to these "best" values. Some are data and problem specific, much like the model architecture itself, and can be determined intuitively such as is the case of the choice of the activation function. E.g, a classification problem would imply the use of softmax/sigmoid function whereas a numerical problem would imply the use of a linear activation (Ronaghan, 2021).

Other times, however, the values may be less intuitive and there are a number of methods by which to determine them, these will be discussed further in the following sections. The important detail to note is that these methods generally rely on testing the model and evaluating the results, hence the model will typically evolve in accordance to its evaluation.

3.1 Data

The initial dataset, from which the model is trained, contained 5,856 grayscale chest x-ray images split into training, testing and evaluation sets (See Figure 3.1 for examples). The model does not partition the data[1]. The data was somewhat biased as it contained twice the number of pneumonia x-rays than that of the normal case. This bias inherently affects the learning of the model and so, to compensate for this, the model was designed to randomly filter pneumonia cases from each of the training, test and evaluation sets, such that the number of images for either case is equal and bias is eliminated.

This kind of filtering inherently reduces the number of data available for the model to train on however, which in turn reduces the amount of information available also. In some cases, this reduction in information is manageable, but as we effectively lose half of the

In diagnosis of pneumonia, the main consideration made of x-rays is the presence of opacity in the lungs. This opacity can be the result of a number of different causes from pneumonia to pleural

[1]This is for the sake of efficiency in testing the model. In further implementations, shuffling the data, (using the random forest algorithm, for example) would be a typical approach.

effusion, so we look to create a model that can not only detect whether opacity is present in the x-ray images, but also differentiate if that opacity is in fact pneumonia.

In any case, it is often prudent to visualise the architecture of a model, as in Figure 3.1.

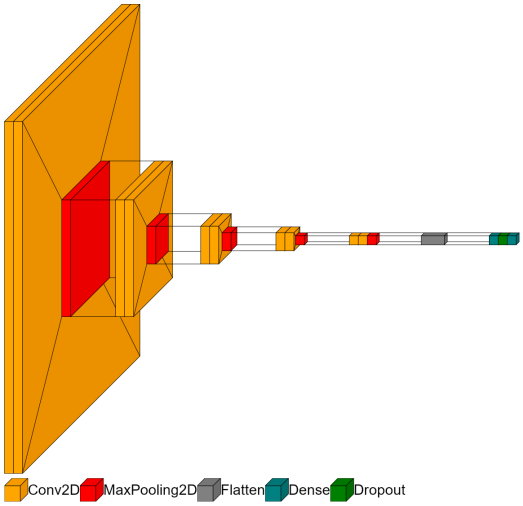
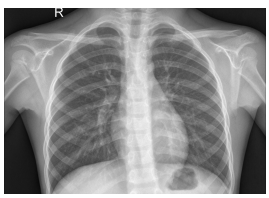


Figure 3.1: Our original normal vs pneumonia classification model architecture graphically represented.

3.2 Hyperparameters and Model Tuning

3.2.1 Learning Rate

Arguably the most important hyper-parameter to optimise in the model is the learning rate. As shown in section 1.2, the learning rate directly affects how large the error is when updating the weights. If the rate is too large, then the model may "overshoot" when correcting the weights and ultimately diverge away from optimal values; too small means the back propagation would slow down and could severely impact computational time. There is no sure-fire learning rate value for every model, it is dependent on both the model architecture and the function being learned. Instead, we start with a rough value and optimise the model through testing. A good typical starting value for the learning rate is 0.01 for



(a) A normal lung x-ray, no opacity.

(b) Opacity in the lungs indicating pneumonia.

Figure 3.2: Two example images from the testing subset of the dataset.

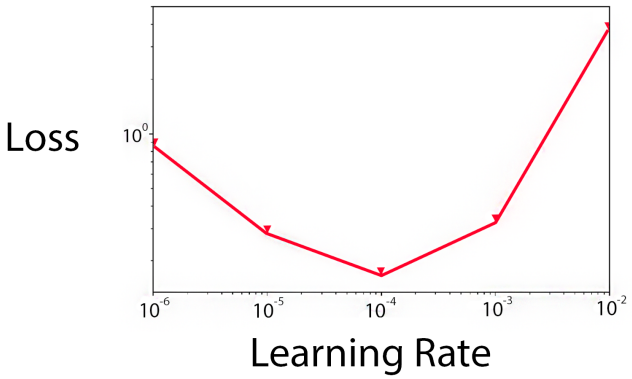


Figure 3.3: Cost vs Learning Rate of grid searches. The first graph suggest an optimal learning rate of 10^{-4} which is used as the centre point of the next set of experiments.

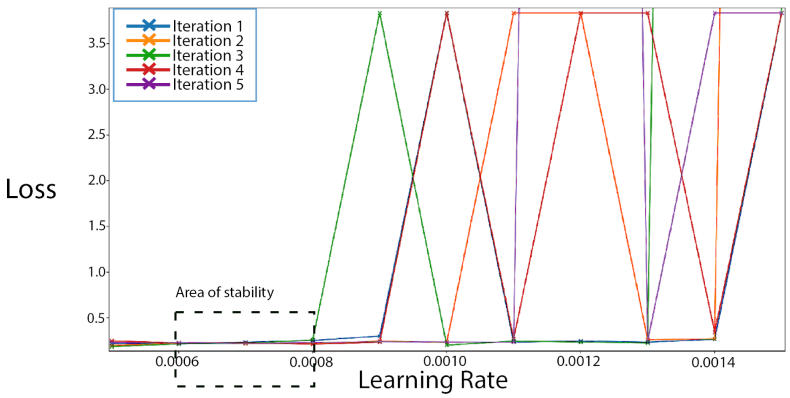


Figure 3.4: The graph suggests a maximum range for the next iteration of the gridsearch.

standard multi-layer neural networks (Bengio, 2012). For optimising the learning rate, the model was tested around the initial learning rate value. By graphing the learning rate against the loss of the model, an optimal point where the loss is least was identified and then the model was tested again around this new value. This method is known as a grid search (Goodfellow, Bengio, and Courville, 2016). From the graphs of the results, shown in Figure 3.2, it is clear that an optimal learning rate for the model is 0.00065.

3.2.2 Initial Class Weights

Although technically class weights are considered general parameters in the model, one can treat the initial values of class weights as a hyperparameter. The biggest reason for setting these initial values

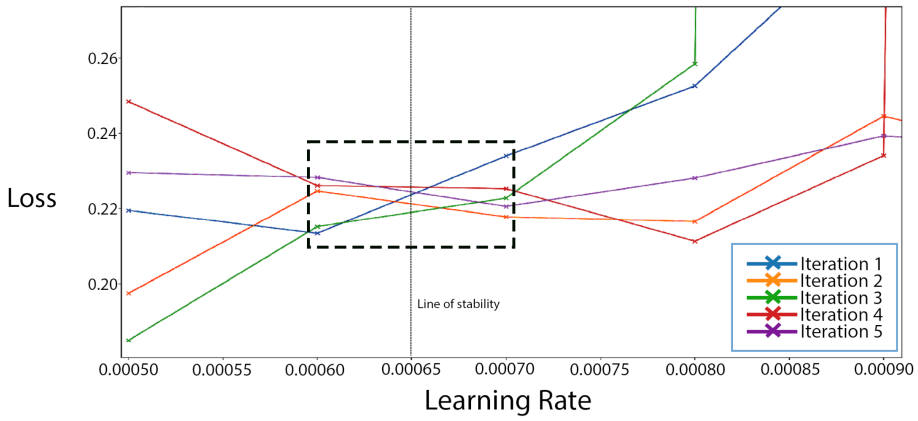


Figure 3.5: The grouping of points in third graph suggests a most stable value for the learning rate is 0.00065.

is to offset the impact of an imbalanced dataset. For example, in our original model, there was a clear imbalance that created a bias toward pneumonia. In this case we chose to address this issue by modifying the data directly such that we had "balanced data", ie. the number of data for each case fed into the model was equal. We could have instead found the ratio between the two classes and modified the initial class weights to represent this so that, during back propagation, the effects of learning on the under-represented class would be more impactful to the model. In this case, we would have "representative data".

It is also important to note that over consecutive epochs the effect of weighting tends to vanish and is only effective in the early stages of training (Byrd and Lipton, 2019). This is does not nullify the importance of weighting however. Even a balanced model may need an initial weighting in order to perform training more effectively. Consider our model, for example. In order to reduce the loss of information, we increased the number of data so that we had a balanced set without reducing information. Yet using a default non-weighted model we consistently received poor results in validation corresponding to little to no learning. So then we argue the case that what we are modelling is essentially an anomaly detection problem, it is more important that the model learns the normal case and classifies depending on it's detection of anomalies (opacity). So then we must modify the models weighting of the data in such a manner. By performing a grid search over an appropriate range of values we find an optimum weighting to be $\text{Class}_{\text{normal}} : 2.0$, $\text{Class}_{\text{opacity}} : 0.4$ and by initialising training in this manner, the model learns more effectively and can continue to train effectively throughout.

3.3 Evaluation

3.3.1 Accuracy, Precision and Recall: The F-Score.

Formally, when discussing accuracy of a model, we refer to the "correct" predictions of the models as being "relevant" data points. For the sake of simplification and ease of conceptualisation, we will focus on these metrics in terms of the model we have built, where positive predictions refer to relevant data points and analogously for negative predictions. Accuracy, as the name implies, measures how accurate the model is at predicting a true value/classification. It is given by it's expected mathematical expression,

$$\text{Accuracy} = \frac{\text{Accurate Predictions}}{\text{All Predictions}}$$

But this calculation only reveals so much. Consider the case of a model using inbalanced data, say 90% of the data is negative and 10% is positive. A model may learn to predict that all cases presented are negative, a poor assumption that would never predict a positive and yet the model would still report having 90% accuracy. In the case of healthcare, this is a potentially disastrous case.

So instead we delve further and determine more specific statistics. The precision and recall are two similar, but fundamentally different statistics of the model. Precision being a measure of the proportion of predicted positives that were in fact positive. Recall (Also known as sensitivity) being a measure of the proportion of accurate positive predictions over all positive data. They can be expressed,

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}, \quad \text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

These two metrics, precision and recall, each highlight a specific nuance of a model's accuracy. So in combination we can define a more precise estimation of accuracy. Specifically we define the F-Score of a model to be the harmonic mean of these metrics,

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

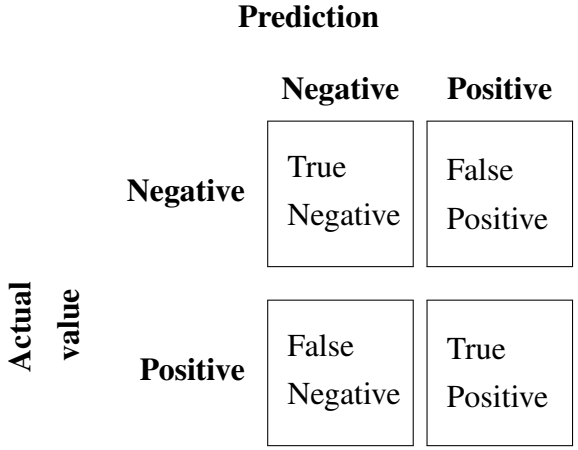
This is known as the F1-score, wherein a perfect model would give a result of 1. It is used to evaluate binary classification systems. (Wood, 2021) Often, however, the data which we work with will influence how much we depends on a particular metric to be more important. An in-balanced dataset biased towards negative results, as previously discussed, would give a high accuracy and a high recall intrinsically, but a true reflection of the model's effectiveness would be found in evaluating its precision. Further, in the field of healthcare, a false negative (say, in diagnosing Covid-19) is far more dangerous than a false positive, so it would be more important to obtain a higher recall in evaluation. Therefore it is preferable to use the generalised F-Score,

$$F_\beta = (1 + \beta^2) \times \frac{Precision \times Recall}{\beta^2 \times Precision + Recall}$$

Here we have a parameter β by which to weight the precision or recall more highly. $\beta > 1$ gives a score favouring recall; and $\beta < 1$, a score that favours precision.

3.3.2 Confusion Matrices

Confusion matrices show how a model evaluates data in comparison to how the data is actually categorised, tallying the comparison within a specific box denoting correct or incorrect assignment. (0,0) denotes a correctly predicted false; (0,1), a false positive; (1,0) a false negative and (1,1), a correctly predicted positive. This may be extended to a higher dimensional matrix, where positive and negative extends into classifications $1, 2, 3, ..., n$. For the sake of our discussion, however, we shall focus on the 2×2 case. By labelling the matrix,



We see that it is essentially a graphical representation of the metrics discussed in the previous section and is useful in representing a models performance. Figure 3.5 shows a sample of a set taken from a 25 epoch experiment using balanced data and a binary loss function.

Calculating the accuracy of the model, we find that it has a final accuracy score of 83.75%; a precision score of 100% and a recall score of 67.5%.

Thus we can calculate it's general f1 score as 80.6%. This is a good result, although as we are working with a model that will predict medical diagnosis we want to investigate it's predictive score prioritising a higher recall. We find the model has an f2-score of 72.2% which is clearly not as good. We will suggest methods to improve the model in the following section.

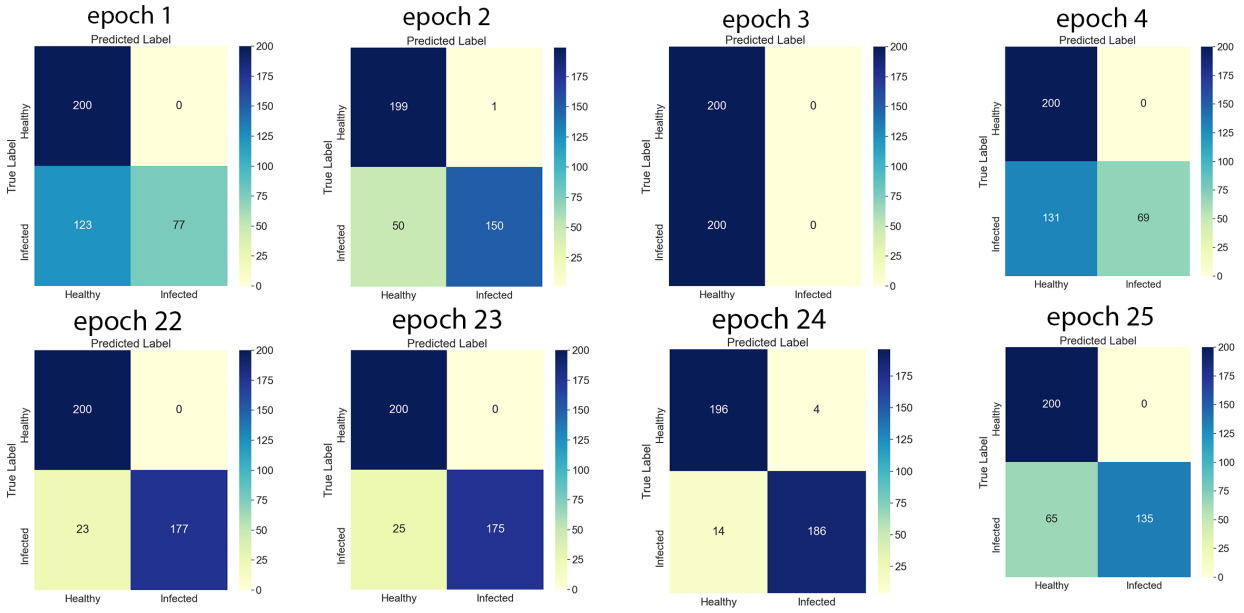


Figure 3.6: Confusion matrices of a 25 epoch run. Matrices taken show a gradual increase in the total number of correctly predicted values. With the majority of incorrect predictions being false negatives.

3.3.3 Explainability

A major challenge for machine-learning approaches to data analysis in general, and in particular to medical science is the inherent gap between human understanding and the predictions of the method of the model.

"The best performing methods are ”black boxes” and can not ”explain” why they came up with a certain decision." -Holzinger, 2018

We want to know *how* our model is determining it's results. For example consider our normal data set, many images show a similar feature in the x-ray, the letter R. If the model uses that letter to evaluate it's prediction then the presence of a letter R, or even annotations in general may influence the results. Further, if we look at the Covid-19 data set, we also see annotations, but further, a number of these images contain Holter monitor electrodes and boxes which are not present in the normal images.

Explainability in machine-learning is a vital aspect of bridging the gap between a predictive model and the understanding of how those predictions are being made. It is the visualisation of the model's reasoning in some aspect. For example, we can use a heatmap (Figure X.X) to show how the activation function is applied on an x-ray image, and in turn what is being observed by the model to make a class distinction.

These images are examples of a gradient-weighted class activation maps (Grad-CAMs). They are generated by processing an image through the usual convolutional layers of the model to generate convolutional feature maps. The average of the weights of the gradients is applied to each respective

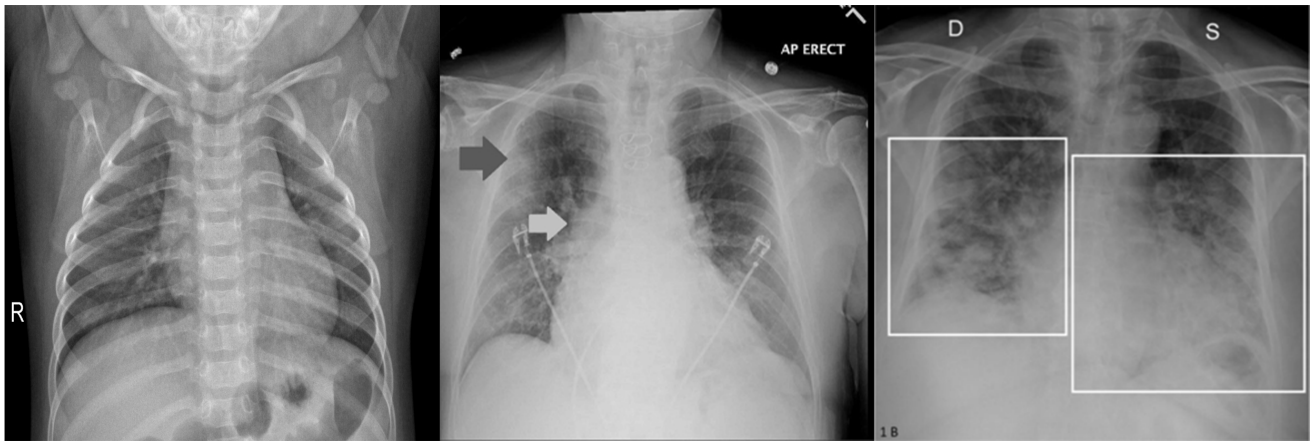


Figure 3.7: From left to right: A normal x-ray annotated with "R"; A Covid-19 x-ray with electrodes and text annotation; A covid-19 x-ray with text and box annotation.

convolutional map[2] and the sum of those maps together forms the class activation map used by the model to produce a desired classification, rather than a predictive one. Where the gradients are large we can infer that that subsection of the image to have affected the model's decision.

Although this doesn't translate specifically what the model may be considering when making class distinctions, it does provide an insight into what is affecting those decisions, positively or negatively. A simple case for example, the classification of a doctor vs a nurse as presented in Figure 3.8 by a biased (78% of images for doctors were men, and 93% of images for nurses were women) and unbiased model. Not only does the utilisation of class activation mapping show what parts of the images contribute to classification of a doctor or nurse but it also highlights how bias in the data set has affected the classifier.

This feature of Grad-Cams is clearly a useful tool for discerning issues in the model and therein ways of improving it. It allows a human to perceive how the evaluation is made and bridges the gap between model prediction and human interpretation, allowing a model to not only be predictive but also informative. This is prevalent in many fields, but certainly within healthcare.

If we refer back to the Grad-CAM images in Figure 3.7 we can ascertain some assumptions. For a normal classification the model appears to be looking at clearly defined opaque shapes, mostly the heart and the liver but we also see an example of it looking at the scapula (Top-right image) and brighter portions of the rib cage (Lower-left image). Whereas in a pneumonia classification the model appears to look at larger, less defined portions of opacity. In the top-left image, for example, we see the pneumonia classification has highlighted the same structures and in the normal case, but also the less defined opacity in the upper region of the chest. Similarly, in the lower-right image, the left scapula is much less defined and has a "smooth" opacity gradient which the model has determined as evidence of pneumonia, we could conclude then that opacity within the x-ray is a defining feature of pneumonia in

[2]Essentially, global average pooling is performed on each of the convolutional layers.

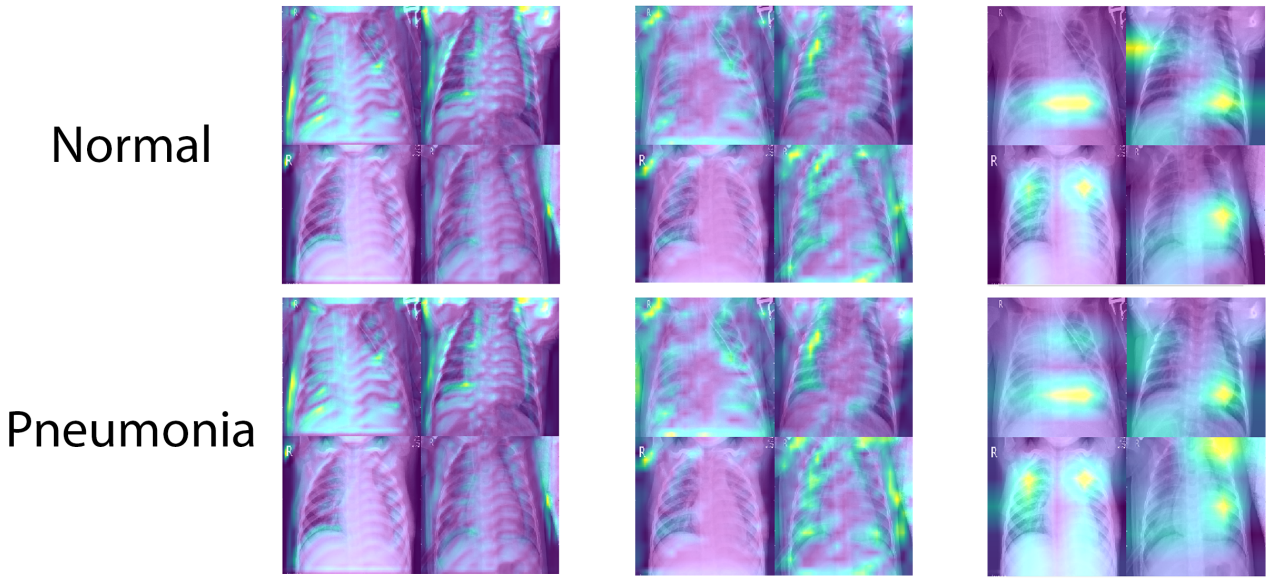


Figure 3.8: From left to right shows CAM heatmaps of the second, fourth and final activation layers of the model when classifying normal or pneumonia from 4 images from the testing set.

patients. This is already known, of course, but it does highlight how a model can be used to augment human understanding of a subject rather than just predict an answer.

Our example also highlights some issues with the model. We do indeed want it to consider levels of opacity in the images, but we certainly do not want it to consider opacity outside of the areas relevant to respiratory illness. That the bones of a patients shoulder is affecting the decision is worrisome to say the least. Fortunately it seems that the annotations within the model do not impact the classification in a major way as the final Grad-CAMs images in all four cases show no activation around the text.

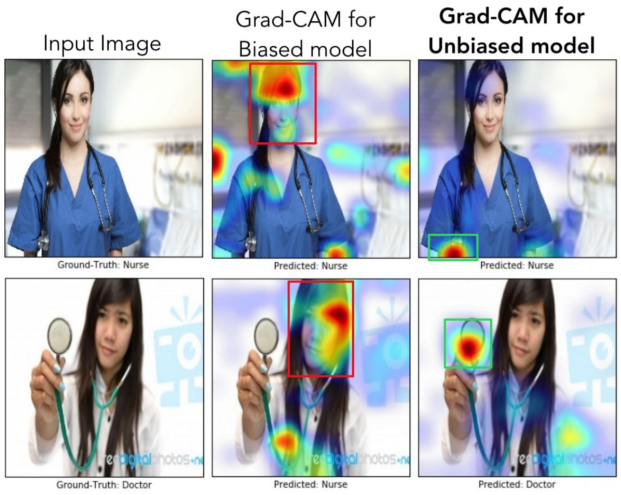


Figure 3.9: The first row of images show correct predictions made by both an biased and unbiased model. The biased model however shows that the face in the image largely contributed to the decision which is a correlation in the data rather than a desired classifying metric. The second row shows the biased model making the correct prediction using the same erroneous portion of the image whereas the unbiased model makes an incorrect prediction but by looking at relevant portions of the images. (Selvaraju et al., 2019)

4 Evolution of the Model

4.1 Data Augmentation

Firstly we want to address the issues highlighted by the Grad-CAM evaluation. We cannot tell the model not to look at areas outside of the rib cage explicitly, but we can influence the behaviour of the model indirectly through modifying the dataset. A method commonly used in machine-learning is to create artificial data from the existing data by applying rotations, translations, scaling, shears, reflections and adjusting pixel brightness. This serves to increase the number of data available to train the model, hence increasing it's effectiveness and also helps to train the model to handle non standard data, for example, a skewed x-ray, or an x-ray from a machine that produces dimmer images.

The model already utilises this technique, but very subtly in most regards. But in order to train the model to rely less on shoulder bones we increase the zoom applied to images when generating artificial data such that the shoulders and head are excluded. We must be careful in doing so though as any reduction in the data could be more harmful than helpful. For this model specifically, we find the optimum zoom to be 1.25x larger. This increased the accuracy and recall of the model, but did slightly reduce precision albeit within an acceptable margin.

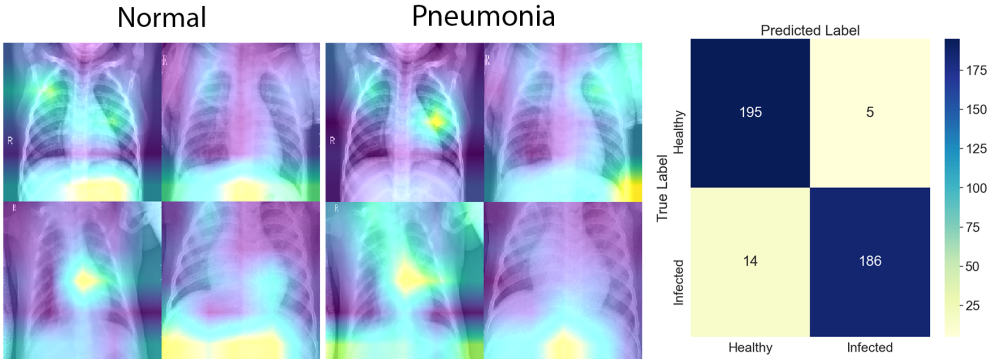


Figure 4.1: The Grad-CAM shows a slightly reduced dependency on parts of the image exterior to the chest by the model. The confusion matrix now shows an accuracy of $\approx 95.25\%$, a recall of 93% and a precision of $\approx 97.38\%$, giving an improved f2 score of $\approx 93.84\%$.

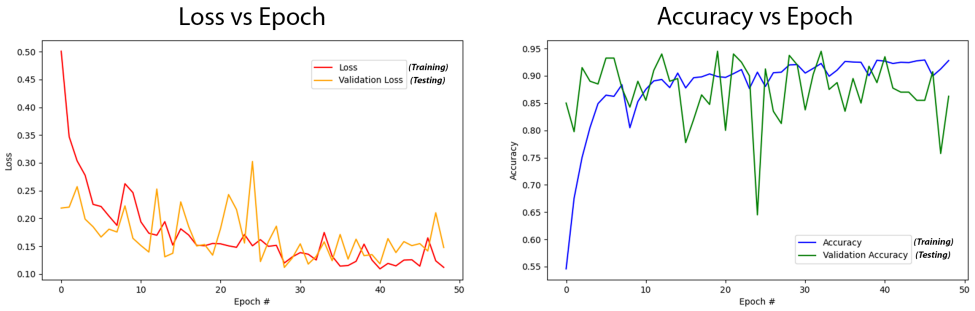


Figure 4.2: The loss vs epoch and accuracy vs epoch for normal vs pneumonia classification.

4.2 Extending the Model

With the model optimised for detecting pneumonia we can look into extending it's functionality to Covid-19 data. We first made a simple substitution of data with which to train the model and found the model to be just as effective if not more so. By inspecting the Grad-Cams of this new model we find that the improved efficacy may be a result of an increased presence of artifacts in the x-ray images, an unfortunate by-product of limited availability of public Covid-19 x-ray data.

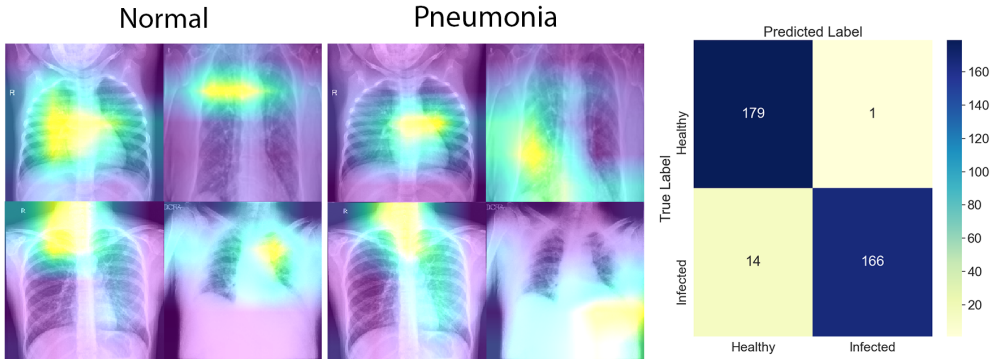


Figure 4.3: The Grad-CAM shows a similar result to pneumonia cases, with the model looking at smaller, less defined sections of opacity in the x-ray images. The confusion matrix reports an accuracy of $\approx 95.83\%$, a recall of $\approx 92.22\%$ and a precision of $\approx 99.4\%$, giving an f2 score of $\approx 93.57\%$.

This model is, however, not an entirely viable model for use in healthcare, it may predict correctly in the scenario wherein the patient either does or does not have Covid-19, but suppose a more likely scenario where the patient may have Covid-19, pneumonia or neither, does the model still maintain it's efficacy? In order to address this we further extend the model from binary classification to a 3

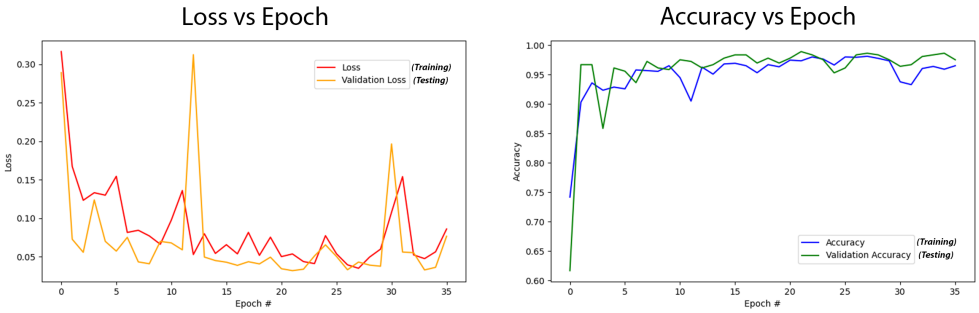


Figure 4.4: The loss vs epoch and accuracy vs epoch for normal vs Covid classification.

classification model. Here, rather than using the sigmoid activation function, we opt for the more appropriate softmax function and utilise one-hot encoding for our classification labels.

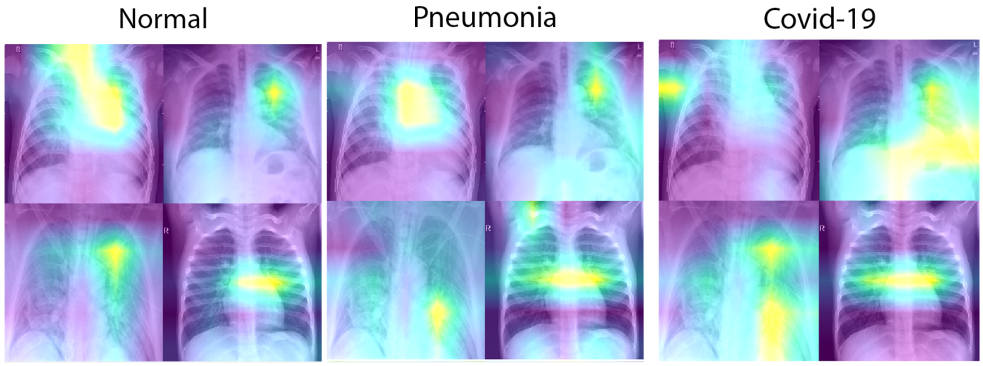


Figure 4.5: The Grad-CAM shows that the model is looking at areas of opacity with blurry edges in the x-ray images when classifying Covid-19 as present, and looks for smaller opacity with defined edges when classifying normal.

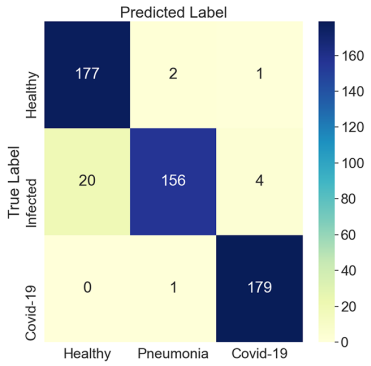


Figure 4.6: The confusion matrix reports an accuracy of $\approx 94.8\%$, a recall of $\approx 97.95\%$ and a precision of $\approx 94.1\%$, giving an f2 score of $\approx 97.16\%$.

A very brief note needs to be made regarding the loss vs epoch and accuracy vs epoch graphs displayed in Figures 4.2 and 4.4. The graphs depict the typical curve we would expect of a reasonable model. However, towards the end of each training we see some instability, this suggests that we are training the model for too long, implying the possibility that the model is over-fitting to the data. Our options to combat this is either to reduce the patience of the model[1] or to explicitly reduce the number of epochs on which the model trains.

Finally, to remark on the effect of larger datasets on the training of the model. Demonstrably there is a clear relationship between increasing the size of the dataset and the accuracy of the model, to an extent. In fact, this relationship has been found to follow a "power-law" learning curve. Wherein small datasets typically provide very large generalisation losses, comparable to a "best guess", and there is an upper limit to the number of data before this loss becomes "irreducible". Between these points there is an otherwise linear relationship that implies the larger the dataset becomes, the more accurate the model will be (Hestness et al., 2017). Looking at our model, and training it using varying amounts of data we find this is the case, see Figure 4.7. We also amended the model to run on 50 epochs explicitly over varying dataset sizes in order to determine how the time the model takes to train would scale. Again we see in Figure 4.7 a clear linear relationship, as was to be expected.

The latter experiment was important to demonstrate considering our model is designed for use in a data-generating time-critical environment. Although the model itself does not need to be trained each time it is used, it makes sense that as further images are acquired, the model be trained further using the new data. Further, if we consider that Covid and it's x-ray patterns evolve in any significant manner, it would be unwise to continue training using the new dissimilar data. Instead it is far more advisable to retrain the model from scratch to prevent the model persisting a suddenly outdated bias. Therefore it is important that whomever adopts use of the model also has the computational capability of retraining it.

It is worth noting that the computation time shown in Figure 4.7 is a result of training the model

[1]Reducing the number of epochs that return similar loss within a defined margin required to stop the training early.

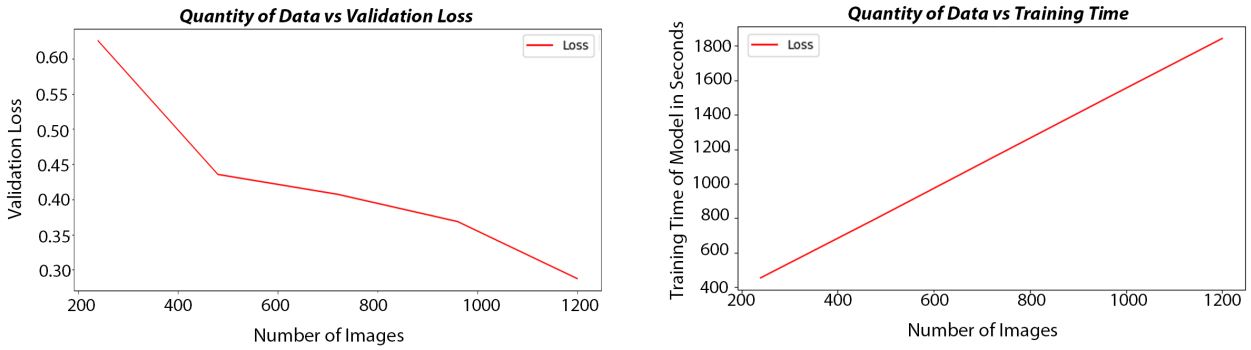


Figure 4.7: On the left we see that the validation loss is reduced as we introduce more data to the model. On the right we see as more data is added to the model, the time taken to train the model increases, there is an evident linear relationship.

using a general non-optimised Intel CPU variant of the python package Tensorflow on an i7-9700k Intel Core (8 CPUs running at 3.6GHz), supported by a GeForce RTX 2060 GPU with 6GB of dedicated memory. The training time using an optimised CPU installation of Tensorflow has been shown to speed up code computation time by up to 3 times (Singh, 2020). Further it has been shown that a GPU specific installation of Tensorflow can result in a code computational speed up of up to 6 times in comparison to CPU only installations.

5 Machine-Learning in Healthcare

Public spending on modern healthcare is seeing an exponential increase in it's rate of growth over time in the UK. It is the largest item of government expenditure wherein spending in the sector has only met demographic pressures from the previous decade. (Stoye and Zaranko, n.d.). Because of this there is a great incentive to optimise the cost-effectiveness of the sector. Further, the healthcare industry has transitioned from a reliance on hospital care to preventative, out-patient care. As a result doctor's are urged to see more patients in a reduced amount of time (McDonald et al., 2015) and a greater focus is placed on establishing relationships or rapport with the patients as the way that a physician interacts with their patient typically determines the patient's satisfaction, response to medication, and their tendency to schedule follow up appointments (Bhardwaj, Nambiar, and Dutta, 2017).

A similar growth in seen in that of the collection of data, in the US alone, the volume of medical data was reported to be around 150 exabytes, with a California-based health network, believed to have contributed between 26.5 and 44 petabytes of potentially rich data from EHRs, including images and annotations (Raghupathi and Raghupathi, 2014). Beyond even this, the expanding integration of technology and lifestyle presents a clear opportunity for the collection of biometric data. Undoubtedly we have the data foundation for effective implementation of machine-learning within healthcare.

The challenges to this though are fairly obvious and apparent; Privacy arguably being the most patent. Recent regulatory requirements imposed by the EU General Data Privacy Regulation (GDPR) or similarly by the US Health Insurance Portability and Accountability Act (HIPAA) aims to protect sensitive personal data. In terms of medical images, this means a restricted resource for developing effective models, a roadblock that has thus far been circumvented through specific institutional research and anonymisation of data. These not only reduce the amount of data available for machine-learning models but may actually lead to the loss of desired utility of the data (Iyengar, Kundu, and Pallis, 2018).

A 2018 computing conference proposed a potential solution to these privacy concerns in the use of what they refer to as "LearningChain", a method based off of blockchain technology. The network is initialised by and consists of several nodes, then each node computes it's respective local gradients and broadcasts these gradients to the network. According to the network consensus of which node holds the authority of the subsequent block in the chain, that node then aggregates the global gradient. Through repetition, a model can be trained without any of the nodes revealing any of their respective data to the network (Chen et al., 2018).

Machine-learning can be utilised for improving efficiency in the healthcare system also. Consider TzanckNet, a convolutional neural network designed to recognize six cell types related to diseases such as herpetic infections, pemphigus, and spongiotic dermatitis (Noyan, Durdu, and Eskiocak, 2020). The

model was pretrained on ImageNet and the classification layer was replaced and retrained with six output nodes to classify six individual cell types. The study defined a reference standard consisting of images labelled by two experienced dermatologists decided through adjudication between the pair and used these to validate the results from the developed convolutional network. The model's accuracy was reported to be ~95%.

The implementation of this system does not then *replace* the role of dermatologists, but instead aims to *augment* the dermatologist in their role. The network can process hundreds of images in a minute so works considerably faster than any human counterpart. Hence, a dermatologist may only need to evaluate the model's findings for any particular case hence reducing the time needed to process a patient. This kind of implementation is known as a clinical decision support (CDS) tool. Further, any image flagged as having a low confidence in the results can be evaluated and, if necessary, a new image can be taken. In this particular case, the images are quick and cheap to produce and thus doing so would not impact cost in any significant manner. Furthermore a report on studies investigating the implementation of clinical decision support (CDS) tools utilising electronic health records (EHR) show a positive economic impact of those systems, in the short-term at least (Lewkowicz, Wohlbrandt, and Boettinger, 2020).

These CDS tools are not limited to simply reducing the workloads of medical practitioners either. As noted, these models can evaluate thousands of pieces of data in a matter of minutes and are then almost *over*-productive in terms of traditional medical practice. With the increasing use of EHRs within the healthcare industry, these models also stand to provide service outside of explicitly case by case need. Screening models may evaluate EHRs to highlight risks or potential disease and illness, which can be pro-actively investigated by a medical practitioner during a regular patient interaction. This greatly improves efficiency of the system through enhancing pro-active preventative healthcare, especially in regards to "hidden" illnesses, such as mental health issues.

For example, one study, which aimed to build a model to predict risk of suicide attempts in patients, did in fact predict suicide attempts far more accurately (80%) than traditional methods (50%~60%). The model evaluated EHRs of the patient for flags indicating risk including diagnostic, demographic, medication, and socioeconomic factors (Walsh, Ribeiro, and Franklin, 2018).

Medical research too has benefited from the rise of machine-learning in healthcare. A deep generative tensorial reinforcement learning, "GENTRL", was developed to evaluate and generate possible DDR1-inhibitor drugs. Of the six compounds the model generated, two were validated in cell-based assays, one of which performed well in testing on mice. Needless to say, although the compounds may require further optimization in terms of selectivity, specificity, and other medicinal chemistry properties, the fact that a model generated them in less than two months (Compared to a traditional timelines of 10-20 years) and at a fraction of the cost displays an inherent advantage in utilising machine-learning in this field.

6 Conclusions

Our results suggest a feasible model for detecting Covid-19 in patients through x-ray imaging with some caveats. Firstly, it must be noted that the images used in the dataset were datasets released to the public for general investigation. Mostly all images of the healthy and pneumonia set were ideal, with little to no annotation. Whereas the images within the Covid-19 set contained annotations and artifacts, such as electrodes. This is a result of the urgency with which these images were processed, there was clearly a need to evaluate the images and provide data for public-led investigative work due to the state of a global pandemic.

A clear conclusion then, in terms of the model itself, is the need for more suitable data. Ideally, specialised data; plain unmarked images with non-visible meta-data generated by working directly with medical professionals in order to develop a robust model.

Secondly, considering the feasibility of the model in the healthcare system. A model designed specifically for analysing x-ray images has only so much mileage. As treatment and identification of Covid-19 in patients becomes more sophisticated, the use of x-rays on patients moves to redundancy for the sake of reducing risk of exposure to radiation. There is still however a potential for the in detecting long-term effects of coronavirus (long Covid). Therefore we would want to develop a similar model to factor more general data as would be found within an EHR, for example, as a model for use in screening patients for long Covid, the x-ray component of the model could then be extended for passive use to detect long-Covid in otherwise necessary x-rays, or actively in the more urgent of cases.

Outside of the model specifically, another clear conclusion that we must make is the need for a concerted effort in generating specialised data for machine-learning in healthcare. Block-chain technologies provide an opportunity for data sharing without impacting the privacy of patients, and as current and developing ML models are beginning to consistently outperform medical professionals, it makes sense to expand data collection and sharing of technological advances across healthcare system globally. A unified effort in this regard would improve existing models and technologies and in turn improve the global healthcare system.

References

Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. IEEE, pp. 1–6.

Bengio, Y. (2012). *Practical recommendations for gradient-based training of deep architectures*. arXiv: 1206.5533 [cs.LG].

Bhardwaj, R., Nambiar, A. R., and Dutta, D. (2017). "A Study of Machine Learning in Healthcare". In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 2, pp. 236–241. DOI: 10.1109/COMPSAC.2017.164.

Byrd, J. and Lipton, Z. C. (2019). *What is the Effect of Importance Weighting in Deep Learning?* arXiv: 1812.03372 [cs.LG].

Chen, X., Ji, J., Luo, C., Liao, W., and Li, P. (2018). "When Machine Learning Meets Blockchain: A Decentralized, Privacy-preserving and Secure Design". In: *2018 IEEE International Conference on Big Data (Big Data)*, pp. 1178–1187. DOI: 10.1109/BigData.2018.8622598.

Cilimkovic, M. (2015). Neural networks and back propagation algorithm. *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin*. 15.

Gebel, Ł. (Nov. 2020). *Why We Need Bias in Neural Networks*. Available from: https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98.

Gholamalinezhad, H. and Khosravi, H. (2020). *Pooling Methods in Deep Neural Networks, a Review*. arXiv: 2009.07485 [cs.CV].

Hansen, C. (Mar. 2020). *Neural Networks: Feedforward and Backpropagation Explained*. Available from: https://mlfromscratch.com/neural-networks-explained/#/.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. (2017). *Deep Learning Scaling is Predictable, Empirically*. arXiv: 1712.00409 [cs.LG].

Holzinger, A. (2018). *From Machine Learning to Explainable AI*. Available from: https://www.aholzinger.at/wordpress/wp-content/uploads/2020/07/For-Students-HOLZINGER-2018.pdf.

Iyengar, A., Kundu, A., and Pallis, G. (2018). Healthcare Informatics and Privacy. *IEEE Internet Computing* [online]. 22.2, pp. 29–31. DOI: 10.1109/MIC.2018.022021660.

Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., and Yan, S. (2015). *Deep Learning with S-shaped Rectified Linear Activation Units*. arXiv: 1512.07030 [cs.CV].

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*. 60.6, pp. 84–90.

Lewkowicz, D., Wohlbrandt, A., and Boettinger, E. (2020). Economic impact of clinical decision support interventions based on electronic health records. *BMC health services research*. 20.1, pp. 1–12.

McDonald, R. J., Schwartz, K. M., Eckel, L. J., Diehn, F. E., Hunt, C. H., Bartholmai, B. J., Erickson, B. J., and Kallmes, D. F. (2015). The Effects of Changes in Utilization and Technological Advancements of Cross-Sectional Imaging on Radiologist Workload. *Academic Radiology* [online]. 22.9, pp. 1191–1198. ISSN: 1076-6332. DOI: https://doi.org/10.1016/j.acra.2015.05.007. Available from: https://www.sciencedirect.com/science/article/pii/S1076633215002457.

Noyan, M. A., Durdu, M., and Eskiocak, A. H. (2020). TzanckNet: A convolutional neural network to identify cells in the cytology of erosive-vesiculobullous diseases. *Scientific Reports*. 10.1, pp. 1–7.

Paeedeh, N. and Ghiasi-Shirazi, K. (2020). *Improving the Backpropagation Algorithm with Consequentialism Weight Updates over Mini-Batches*. arXiv: 2003.05164 [cs.LG].

Pedamonti, D. (2018). *Comparison of non-linear activation functions for deep neural networks on MNIST classification task*. arXiv: 1804.02763 [cs.LG].

Raghupathi, W. and Raghupathi, V. (2014). Big data analytics in healthcare: promise and potential. *Health information science and systems*. 2.1, pp. 1–10.

Ronaghan, S. (2021). *Deep Learning: Which Loss and Activation Functions should I use?* Available from: https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (Oct. 2019). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision* [online]. 128.2, pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. Available from: http://dx.doi.org/10.1007/s11263-019-01228-7.

Singh, R. (2020). *Accelerate your training and inference running on Tensorflow*. Available from: https://towardsdatascience.com/accelerate-your-training-and-inference-running-on-tensorflow-896aa963aa70.

Stoye, G. and Zaranko, B. (n.d.). *UK health spending*. Available from: https://www.ifs.org.uk/uploads/R165-UK-health-spending2.pdf.

Walsh, C. G., Ribeiro, J. D., and Franklin, J. C. (2018). Predicting suicide attempts in adolescents with longitudinal clinical data and machine learning. *Journal of child psychology and psychiatry*. 59.12, pp. 1261–1270.

Wood, T. (2021). *F-Score*. Available from: https://deepai.org/machine-learning-glossary-and-terms/f-score.

Bibliography

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press. Chap. 11, pp. 416–422. ISBN: 9780262035613. Available from: https://books.google.co.uk/books?id=Np9SDQAAQBAJ.