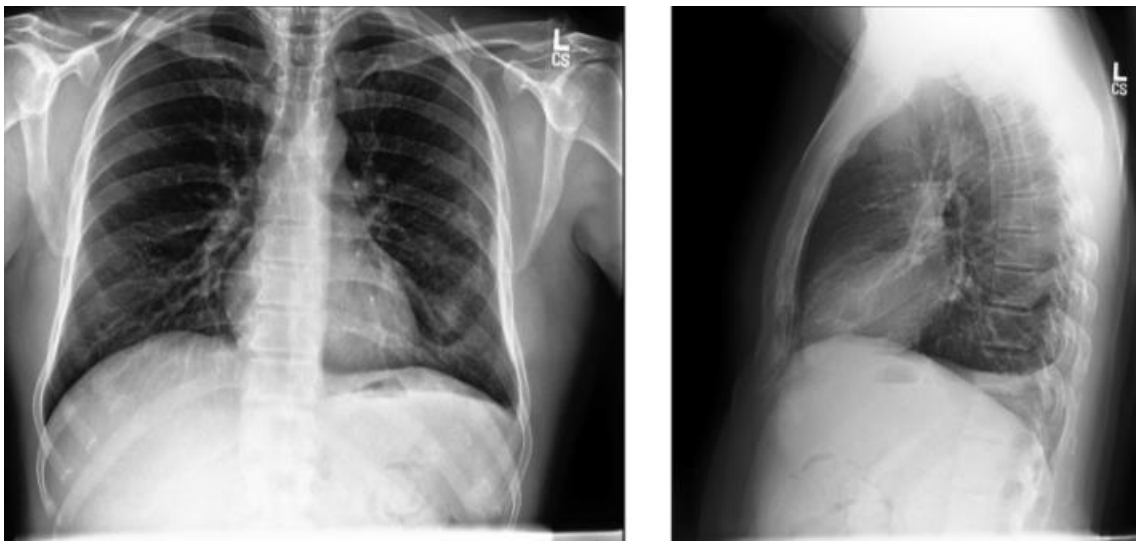
INTERNATIONAL HELLENIC UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF INFORMATION AND ELECTRONICS ENGINEERING



Detecting diseases from medical images using Deep Learning Techniques

BACHELOR'S THESIS

Student Name
Dimitrios Chasanidis
Reg. Number: 154564

Supervisor Professor:
Konstantinos Diamantaras

Ανίχνευση ασθενειών από ιατρικές εικόνες με μεθόδους βαθιάς μάθησης
19059
Χασανίδης Δημήτριος
Ονοματεπώνυμο εισηγητή Διαμαντάρας Κωνσταντίνος
Ημερομηνία ανάληψης Δ.Ε. 26-11-2019
Ημερομηνία περάτωσης Δ.Ε. 17-09-2020

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Abstract

There has been an extremely significant progress in the field of machine learning in the last few years. Deep learning and parallel computation using high end GPUs have proved to be invaluable tools that seem to lead the machine learning research. Research in deep neural networks has yielded amazing results in various tasks such as image classification, natural language processing, fraud detection, recommendation systems, generating artificial points, healthcare and others. This thesis explores two datasets and the potential for automatic detection of problems in chest X-rays. The NIH (ChestX-ray8) dataset which was the first big public dataset containing thoracic X-rays with its corresponding annotations and the CheXpert dataset. Apart from the classic way of attempting to create an image classifier, I attempted to conduct an experiment with the images which appeared to not be completely fruitful, but it was still a valuable experience and made me all the more interested in the field.

Acknowledgements

I would like to thank my supervisor Konstantinos Diamantaras for the valuable guidance, patience and of course for giving me access to his private server to conduct my experiments.

Contents

List of Figures

Machine Learning Introduction

Machine Learning vs Traditional Programming paradigm

Computer programming has been around for more than a century. The idea is that given a problem, we can write down a set of instructions that a computer can read and furthermore execute, that will ultimately solve this problem. A typical problem takes some input, executes all the given instructions and produces a result. This is called the **traditional programming paradigm.** A simple example of such problems is calculating the surface area of a circle. The surface area of a circle is calculated using the following formula

$$A = \pi r^2$$

In order to use this formula, we need to know the value of the radius (r) of the circle. In the traditional programming paradigm, r is the input of the program, A (area) is the output of the program and the formula is the set of instructions.

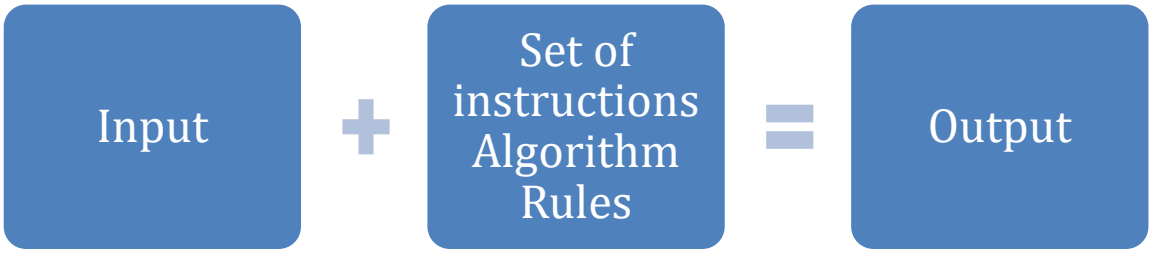


Figure 1 The traditional programming paradigm

This way of solving problems was and still is widely popular as it is a very natural way of thinking. In most situations in our lives we think algorithmically even if we are not aware of it. For example, the routine before going to work would be:

1. Wake up
2. Warm up water for coffee
3. Take a shower
4. Make coffee
5. Go to work

This way of thinking is efficient for problems that are simple to solve and we are able to find the set of instructions needed. But this is not always the case. More complicated problems cannot be solved using a strict instruction set. For example, spam detection/filtering. Spam email detectors are extremely useful but not so easy to build. Thinking a set of instructions that could solve this problem is way harder than one would expect. We could for example think of key-phrases used in this kind of

emails that are most likely used in most (if not all) of them. But even then, the rules would be arbitrary and based mostly on the experience. Problems like these are solved by experience. The problem is, how do we transfer experience as humans perceive it to a computer? Well the premise stays the same. We feed the computer with a big sum of data (just as we would with a human), give the result to the computer and expect it to learn by trial and error. This is called learning.
Learning is the process of improving a system (either human or computer) upon a certain task after the observation of several examples. Furthermore, the way we write programs for computers changes. Instead of feeding inputs and rules to a computer for it to produce a result, we now feed it inputs and outputs for it to produce rules. The paradigm changes.

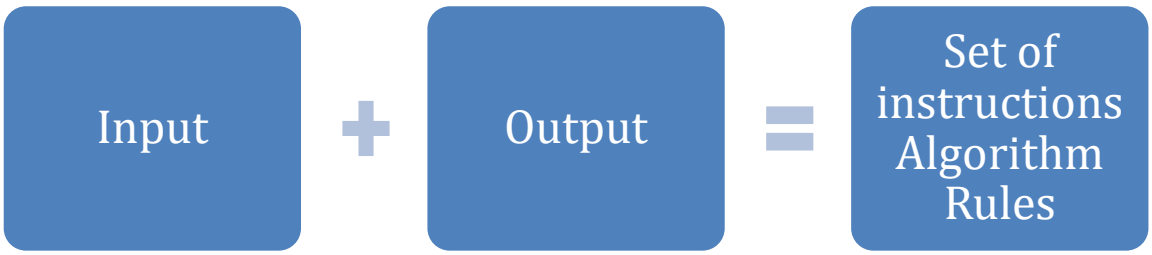


Figure 2 The Machine Learning programming paradigm

Applying the same example with the calculation of the surface of a circle with radius r, we would feed a system with multiple pairs of radius-area and expect the system to learn how to calculate the surface area. Given that the system would learn through experience, we expect it to learn an approximation of the function A.

Types of Machine Learning

Machine learning is consisted of 3 main categories:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement learning

Supervised Learning

In this category of learning, a model is fed with a series of inputs and outputs and is expected to learn how to connect these 2. In most cases, inputs and outputs are fed into these models in the form of tensors.

Unsupervised Learning

In this category of learning, a model is fed only a series of inputs and is expected to find some sort of pattern between them. The goal of using unsupervised learning is generally to model the underlying structure of the data and extract information from it. A typical example of such an algorithm is the k-means clustering algorithm.

Reinforcement Learning

This type of learning is more focused on decision making problems. The idea is that there is a model (decision maker) and an environment. The model is fed with inputs and a punishment or a reward depending on the input. The model is learning how to avoid punishment and get rewards. A great example of such a problem is playing chess. A model is given thousands of already played games with their corresponding result and is learning how to make decisions that will lead the model to win more games and get more reward.

Problem types and Metrics

Problem types

Machine learning aims to solve problems that are too difficult to tackle using the traditional programming paradigm. There are several types of these problems.

- Classification
- Regression
- Anomaly Detection
- Sample Generation
- Time series forecasting

Classification

This is one of the most common type of problems that employs machine learning algorithms to solve. In this type of problems, the program/algorithm/model is required to specify which of k categories the input belongs to. Assuming we call the produced model f and x the input:

$$f : \mathbb{R}^n \rightarrow \{1, \dots k\}$$

There are 2 subcategories in this kind of task. Multiclass classification and multilabel classification. Multiclass classification is when the model must choose exactly 1 of the k available classes to classify the input whereas in multilabel classification the model can choose any number $n \leq k$ of the k classes to classify the input. An intuitive way to better understand the difference between the 2 is the following

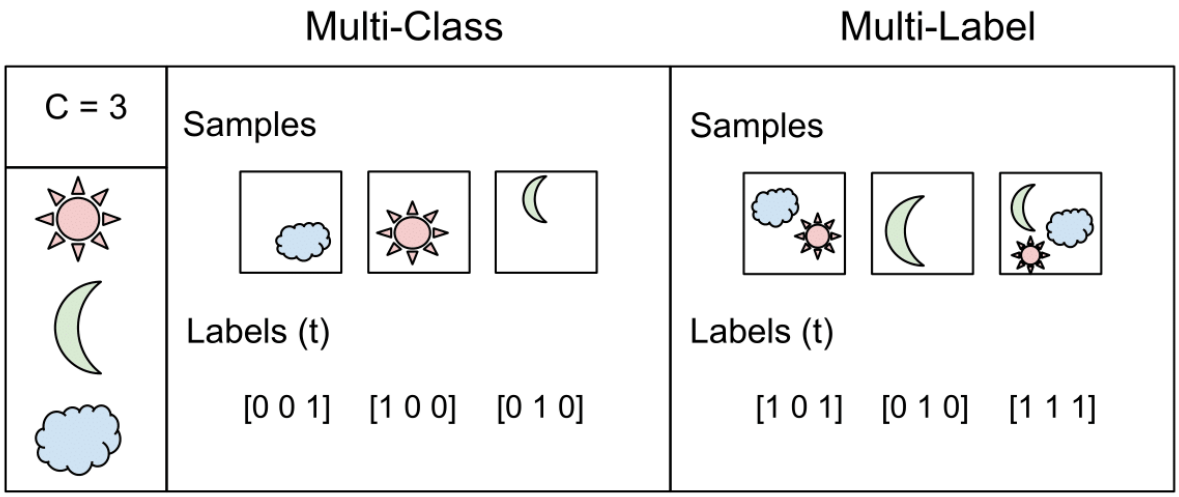


Figure 3 Multi-Class vs Mutli-Label classification intuition

In this example we have 3 labels. $Sun, Moon, Cloud$. The idea is that in multiclass classification the images are given in such a way that **at most 1 of the 3** classes appear (i.e. in multi-class problems the classes are mutually exclusive). In multilabel classification an image could contain **all 3 classes or none of them.**
It is worth noting that multilabel classification problems, depending on their complexity, are often treated as k multiclass classification tasks. The idea is that one could create k different models that each one is responsible for one single class, improving the results but also increasing the computational cost by k times. In tasks that big and complex neural network architectures are employed, this can be prohibitively costly in both time and computational cost (hardware).

Regression

Regression is much like the classification task. It is a supervised learning problem, but the targets of each input are real numbers instead of discrete ones. The idea

Metrics

In order to assess the model's performance, we need measurements that can quantify how well it does. In order to do this, we have created various metrics where each of them focuses on different aspects of the model's performance.

Classification Metrics

The basis of all classification metrics is the confusion matrix. The idea is that in a classification task the model's prediction can be one of the following:

1. True positive: The model's output is positive and so is the target
2. True negative: The model's output is negative and so is the target
3. False positive: The model's output is positive, but the target is negative
4. False negative: The model's output is negative, but the target is positive

The confusion matrix is also usually depicted as a table

Predictions	Actual Values		
		Actual positive	**Actual Negative**
	Predicted Positive	True Positive	False Positive
	Predicted Negative	False Negative	True Negative

When we are in possession of the confusion matrix, we can calculate almost all metrics available for a classification task.

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

Accuracy is one of the most common (if not the most common) metric used in classification tasks. It basically calculates how correctly the model correctly predicts the classes. This metric although common, it is especially weak in cases where heavy imbalances occur in the dataset. Say for example we are in the possession of a dataset that has 900 samples that are positive and 100 samples that are negative.
If we correctly identify 90% of the positive samples (i.e. $90\% * 900 = 810$) and 0% of the negative classes (i.e. $0\% * 100 = 0$) we have the following confusion matrix:

True positive = 810 samples
True Negative = 0 samples
False Positive = 100
False Negative = 90

$$Accuracy = \frac{TN + TP}{TN + TP + FP + FN} = \frac{0 + 810}{0 + 810 + 100 + 90} = 0.81\ or\ 81\%$$

This is extremely high accuracy for a model that could not predict a single negative sample correctly.

In order to avoid having this problem the following metrics were introduced

- $Precision = \frac{TP}{TP+FP}$

Precision gives us the number of real positive cases in the dataset

- $Recall = \frac{TP}{TP+FN}$

Recall gives us the number of real negative cases in the dataset

A measure that combines both precision and recall

- $F - score = 2 * \frac{Precision*Recall}{Precision+Recall}$

F-score is considered a quite balanced metric and tries to alleviate the imbalance problems that accuracy suffers from.

A more focused approach would be to calculate the following:

- $Sensitivity\ (True\ Positive\ Rate, Recall) = \frac{TP}{TP+FN}$
- $Specificity (True\ Negative\ Rate) = \frac{TN}{TN+FP}$

By calculating these 2 we can calculate how well the system does in identifying positive samples exclusively or negative samples exclusively.
Lastly, a metric that more complicated is the AUC (Area Under the Curve). The idea is if we plot the True Positive Rate (y axis) and put False Positive Rate on the x axis we have the following:

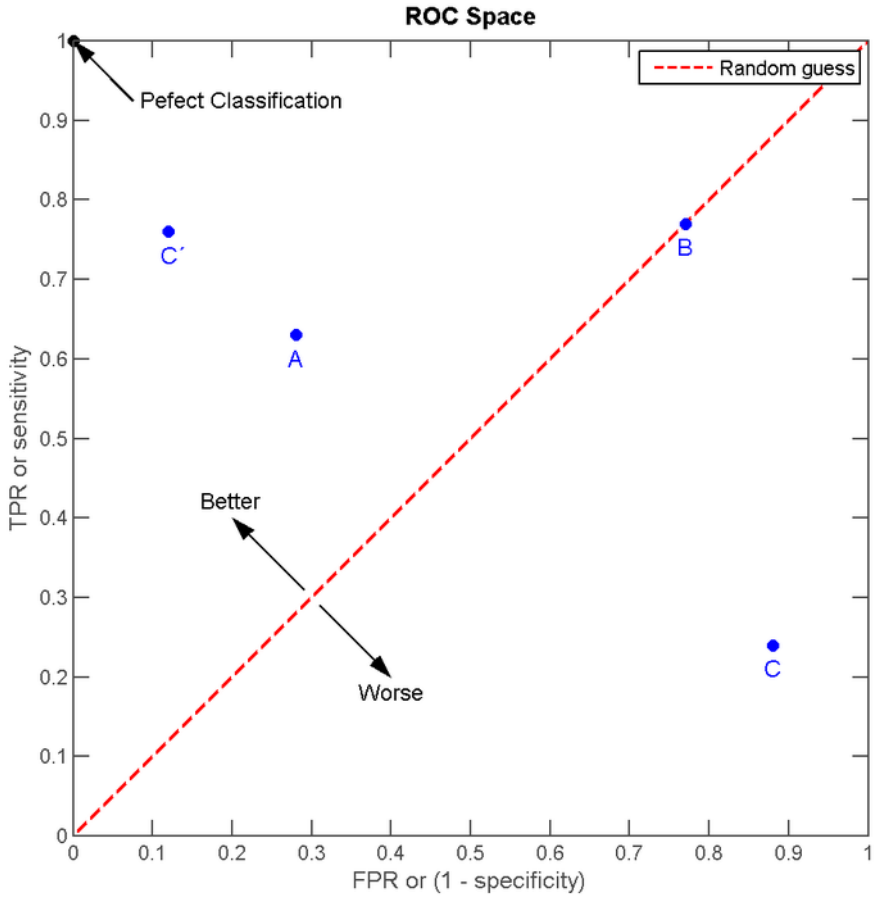


Figure 4 ROC Curve depiction

A completely random "classifier" would be in the middle. The idea is that as the classifier improves, the curve goes further up resulting in the following graph

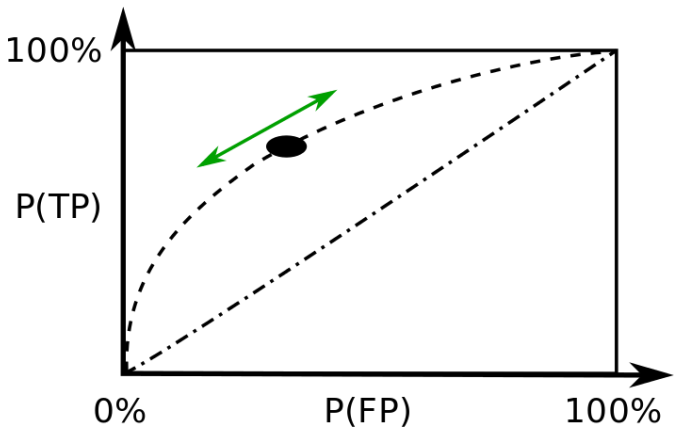


Figure 5 Area Under the Curve (AUC)

The AUC score is the surface area under the curve.

Regression Metrics

Regression metrics are fewer and do not require preprocessing of the model's outputs unlike in a classification task. Two representative examples are the Mean Absolute Error and the Mean Squared Error.

- $MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - Y_i^*|$
- $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i^*)^2$

Generalization – Underfitting – Overfitting

In machine learning, the whole premise is that we feed a model many inputs with their corresponding outputs and through the training process it learns how to better achieve its given task. So, a machine learning model's job is to approximate the underlying distribution of the dataset. The problem occurs when training goes "too far" or it does not go far enough.

Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize.

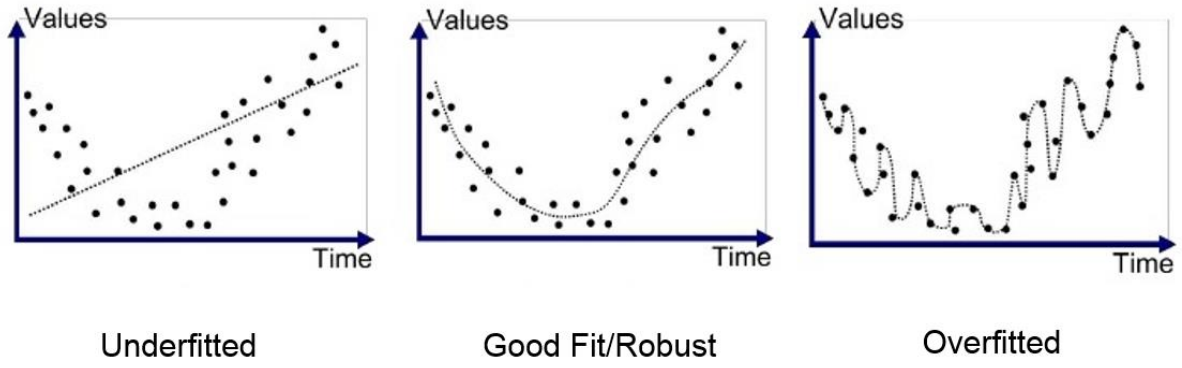


Figure 6 Depiction of Underfitted - Robust and Overfitted model

Artificial Neural Networks (ANNs)

Introduction

Artificial neural networks are loosely inspired by the brain. The way the brain works has been perplexing the scientific community for an awfully long time. Up until now, we still do not completely understand how it works. We know that the brain is an extremely complicated, non-linear, parallel computer. It consists of neurons - biological computing units- that form a huge dense network capable of computing speeds that far exceed the speed of a state-of-the-art digital computer. Another amazing ability of the brain is its adaptability. Neurons and the networks they form are not rigid. A great example of its adaptability is how a person recovers from a stroke. When a stroke occurs, a part of the brain stops getting the necessary oxygen which results in death of some brain cells which ultimately results in losing some of its abilities (e.g. speech, movement). This loss however is not permanent. Given some time and proper help a person can regain most (if not all) of the lost abilities. This property is called **neuron plasticity** and it is one of the basic properties of artificial neural networks.

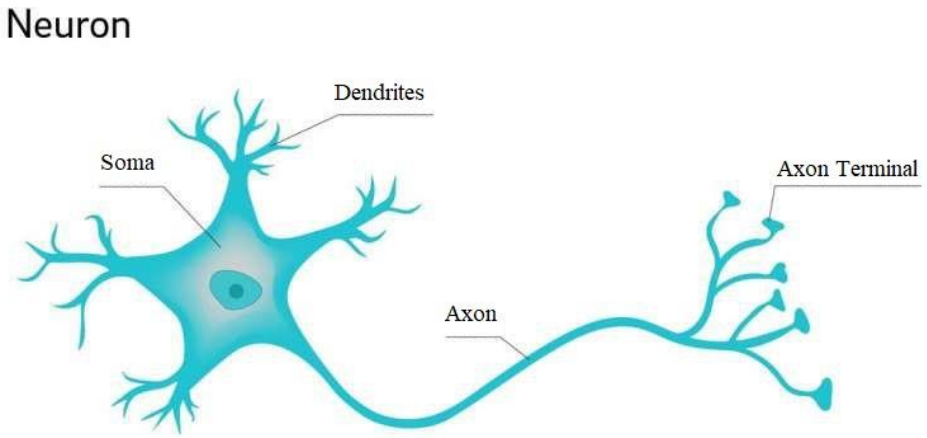


Figure 7 A biological neuron consists of (i) dendrites, (ii) cell body - soma, (iii) myelin sheath and the (iv) axon terminals - synapses.

The dendrites act as the input receivers of the electrical signals given by other neurons. The axon terminal acts as the output.

Linear Neural Networks

So, given the knowledge we have of how biological neurons work, computer scientists took inspiration from neurons, and tried to recreate these types of neural networks by mathematically modelling them. More specifically, in the 1940s Warren McCulloch and Walter Pitts introduced the first computational model of a simple neuron. This network is a function that given an input X it produces a value u that depending on the threshold value θ the neuron either fires or not. We assume that 0 means the neuron does not fire, and 1 means that the neuron fires. We can model this mathematically like this:

$$u = \sum_{i=1}^{n} w_i x_i + \theta, \qquad \theta = w_0 x_0$$

$$y = f(u) = \begin{cases} 0, & u \leq 0 \\ 1, & u > 0 \end{cases}$$

The value **u** is the neuron stimulation, and the function **f** is called activation function (in this particular case step function) and **θ** is the threshold.

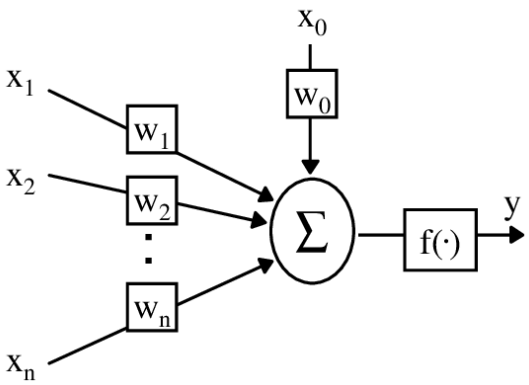


Figure 8 The McCulloch-Pitts neuron model

The x_i and w_i values are vectors containing the input values and their respective weights. We can also write this as:

$$\boldsymbol{x} = [x_0, x_1 \dots x_n]^T$$
$$\boldsymbol{w} = [w_0, w_1 \dots w_n]^T$$

These 2 vectors are of size n+1, where n is the number of inputs. The extra value is the threshold value, also called **bias.**
Linear networks are creating straight lines that aim to separate the classes.

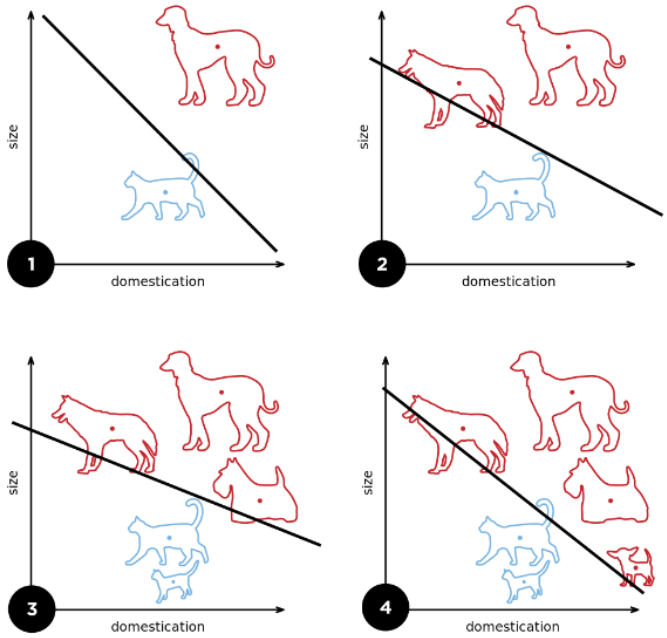


Figure 9 An intuitive example of how linear neural networks are producing

The obvious limitation of linear networks is that they can only represent linear functions and thus, every problem that cannot be linearly separated cannot be solved by this type of network. Thankfully, that is where multi-layer perceptrons come in.

Multilayer Perceptron – Deep Feedforward Networks

Simple, one layered neural networks (perceptrons) are the simplest networks we can study but are limited in their classification capabilities. In order to fix this problem, multilayer perceptrons (or deep feedforward neural networks or MLPs) came to be. Deep feedforward neural networks form the basis of convolutional neural networks that led the deep learning revolution that started in 2012. These networks are consisting of

- an input layer
- n-hidden layers
- output layer (or activation layer)

These models are called feedforward because information flows through the function being evaluated from ***x*** through intermediate calculations (the hidden layers) and finally output ***y***. They are called networks because they are usually represented by composing many different functions together.

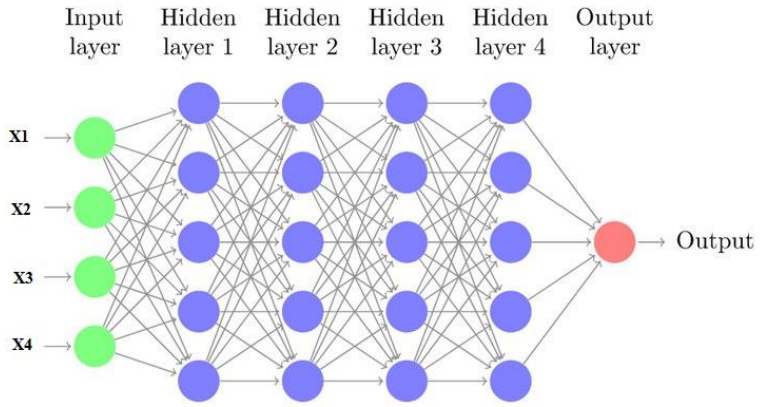


Figure 10 A typical Multilayer Network

Taking the above diagram as an example, we can represent the network as

$$y = OL\left(HL_4\left(HL_3\left(HL_2\left(HL_1(IL(x_1, x_2, x_3, x_4))\right)\right)\right)\right)$$

- IL → Input Layer
- HL_n → Hidden Layer
- OL → Output Layer (activation function)

This network is a deep feedforward fully connected (or dense) network, meaning that each output of every hidden layer is used as input to each node of the next layer. So, for example if IL is the input layer and its output is a vector $v = [x_1, x_2, x_3, x_4]$ $then$ HL_1 is a function that takes as input this vector v and outputs a transformed vector that is of shape k $[x_1, x_2, x_3, x_4, \ldots, x_k]$
Each hidden layer represents a function that takes as input the output of its previous layer.
The idea of stacking multiple layers was loosely inspired by neuroscience. Stacking layers has proven to be a highly effective idea to tackle complicated problems. A good way to think of these networks is as function approximating machines that achieve statistical generalization. The idea is, that if we can model a problem – task well enough and find the optimal constants-weights we can predict its state given an input. It is worth noticing that the "feedforward" part of the name is derived from the fact that the information flows along a single direction and thus the network does not contain any self-connections or recursive connections.
An interesting and useful theorem of MLPs is the universal approximation theorem. This theorem establishes that an MLP can approximate any function given enough layers and nodes. Of course, this also means that for complex data distribution the size of the neural network could be quite large.

Activation Functions

Apart from the step function that was mentioned in the previous chapter, there are more activation functions.

- Step function (0, 1)

$$f(u) = \begin{cases} 0, u \leq 0 \\ 1, u > 0 \end{cases}$$

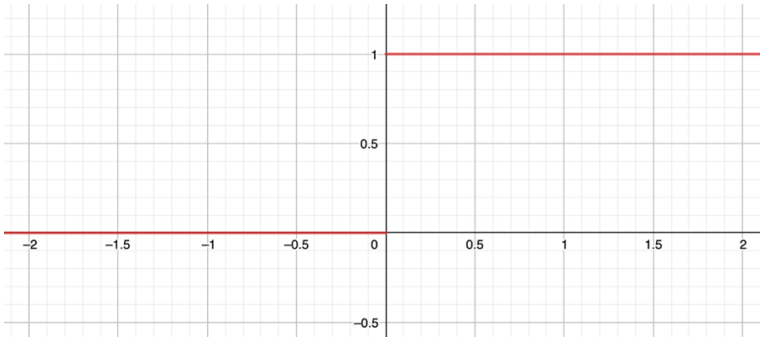


Figure 11 Step Function (0-1)

- Step function (-1, 1)

$$f(u) = \begin{cases} -1, u \leq 0 \\ 1, u > 0 \end{cases}$$

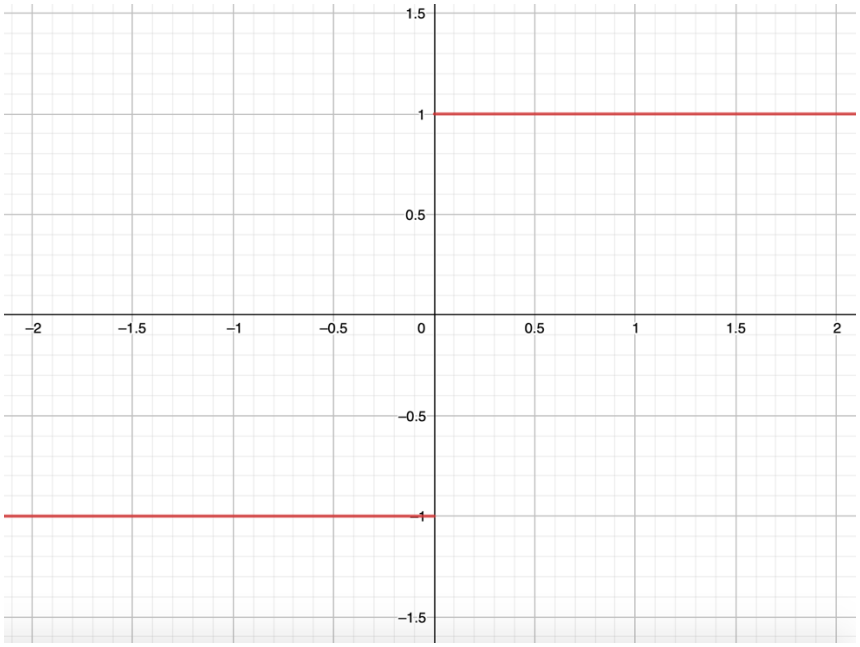


Figure 12 Step Function (-1,1)

- Hyperbolic Tangent

$$f(u) = \mathbf{tanh}(u) = \frac{1 - e^{-u}}{1 + e^{-u}}$$

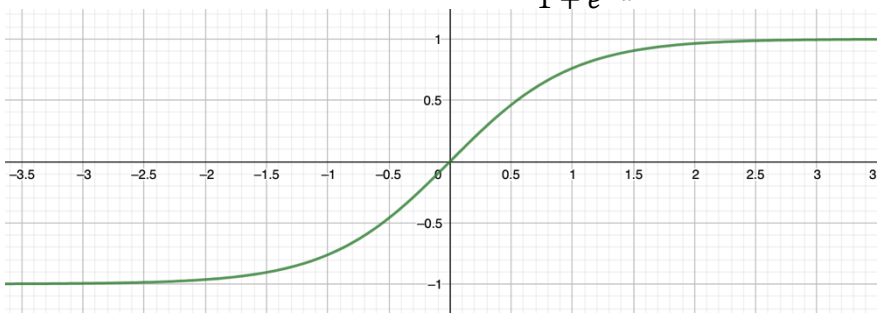


Figure 13 Hyperbolic Tangent

- Threshold function

$$f(u) = \begin{cases} 0, & u \leq 0 \\ u, & u < 1 \\ 1, & u \geq 1 \end{cases}$$

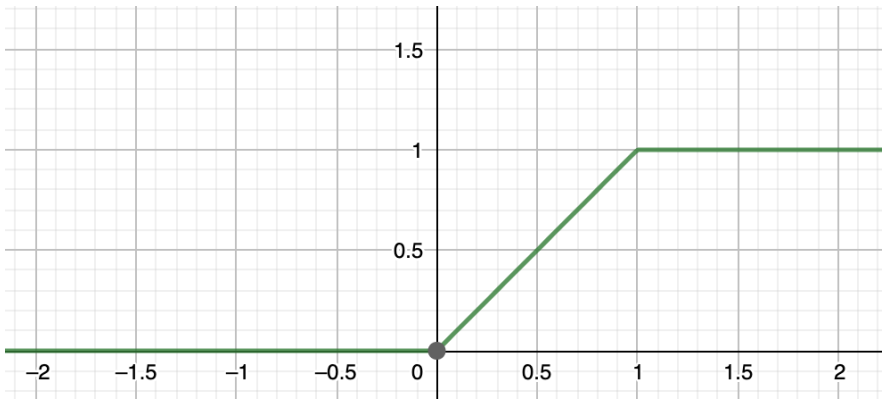


Figure 14 Threshold function

- **ReLU (Rectified Linear Unit)**

$$f(u) = \max(0, u)$$

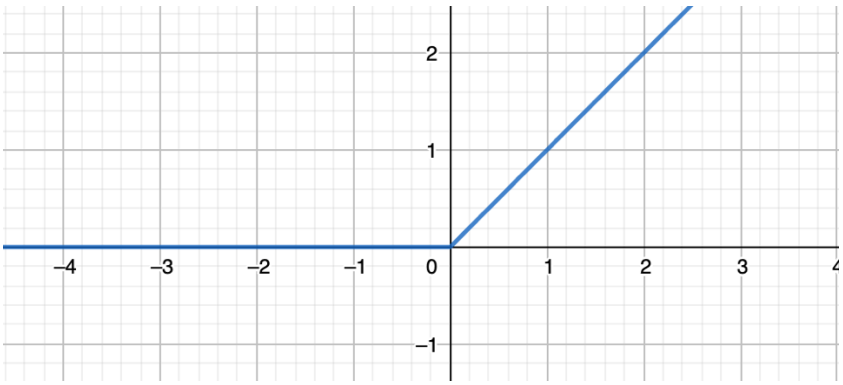


Figure 15 ReLU

- Sigmoid function
  This activation function is extremely useful in multi-label classification problems since it takes all the logits from the last layer and "squashes" them between 0 and 1. In the context of neural networks, the sigmoid function is applied to each output independently.

$$f(u) = \frac{1}{1 + e^{-u}}$$

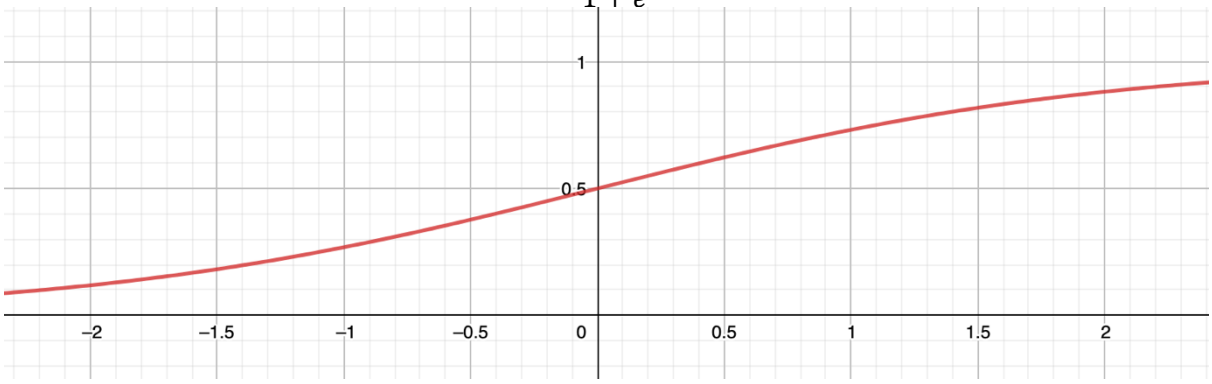


Figure 16 Sigmoid Function

- **Softmax**

$$f(u_i) = \frac{e^{u_i}}{\sum_{j=0}^{k} e^{u_j}}$$

Softmax function is an extremely useful tool – activation function in multi-class classification problems. It turns the **k** logits from the last fully connected layer of the network into **k** probabilities. The range of the probabilities will be from 0 to 1, and the sum of all the probabilities will be equal to one. It is especially useful in multi-class classification problems since it "forces" all the logits to compete with each other, ultimately increasing the probability of the logit with the highest value and decreasing the probability of all other logits.

Training – Optimization for Neural Networks

Cost Functions

Cost (or loss) functions are how the networks evaluate how well the network does during training. Depending on the task the cost function is different.

A typical loss function for binary classification tasks is the binary cross entropy loss function.

$$Binary\ Cross\ Entropy(y_{true}, p) = -\sum_{c=1}^{M} y_{true} * \log_e p$$

In multiclass classification problems where each class is mutually exclusive with the other the cross-entropy function is modified into the following:

$$Categorical\ Cross\ Entropy(y_{true}, p) = -\sum_{i}^{C} t_i * \log(f(s)_i),$$

where f(s) is the softmax function.

Backpropagation – Stochastic Gradient Descent – Vanishing Gradient Problem

Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptrons to multilayer feedforward neural networks. The "backwards" part of the name comes from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately.

Stochastic Gradient Descent is an optimization method used for training of neural networks. The concept is that in order to train a neural network, we need to have the following:

1. A differentiable loss function
2. Target values for the network to try to predict

We could imagine that the loss function is space where there are “hills” and “valleys”. Since we have an optimization problem in our hands, we are trying to find a way to minimize the loss function. If we make of the loss function with respect to the weights, we have the following:

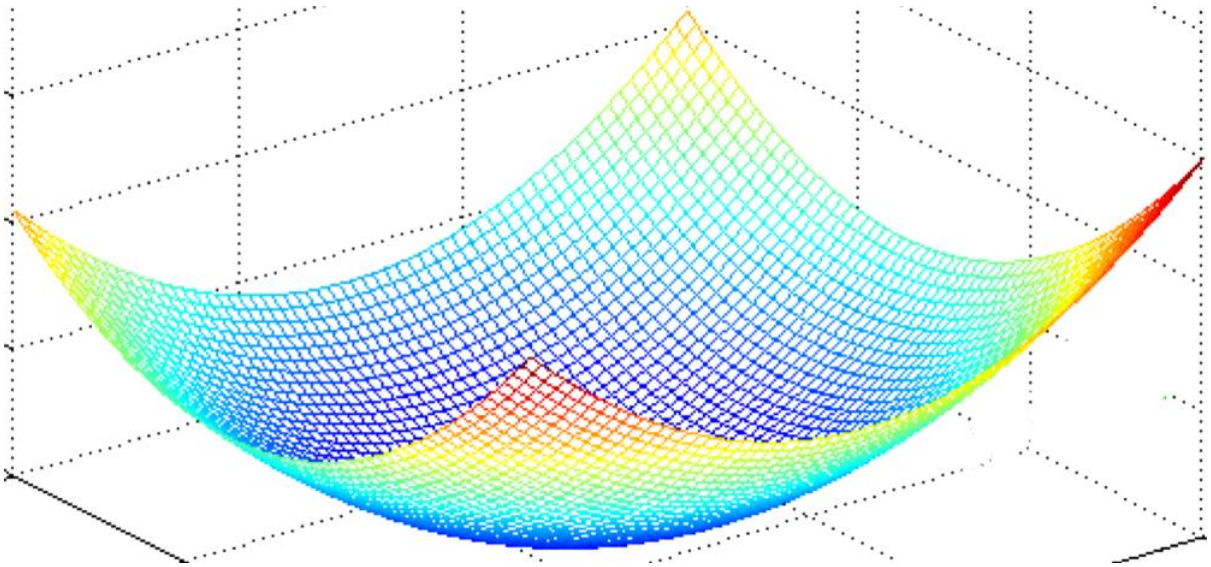


Figure 17 A 3-dimensional depiction of a function

The idea is that we need to find a way to reach the lowest possible point of this function in order to have the lowest possible loss value and by extension the best model performance.
In order to do that we use the concept of the partial derivative. In calculus when we want to minimize a function, we find its derivative and depending on its value on a the current point we either move “up” or down. In neural networks finding the partial derivative for each single weight is tremendously costly, and that is why the backpropagation algorithm is used. The weights (w) are updated using the following formula

$$w \leftarrow w - b \sum_{n=(i-1)B+1}^{iB} \nabla_w E_n$$

The problem that arises from this

Learning Rate Schedulers

Learning rate is one of the most important hyper parameter during the training procure of a neural network. Learning is so important because it is the one determining how fast (or how slow) a model will converge to its minima. Too high a learning rate could result in the neural network to never actually converge to the minima or even worse to completely go off the rails and diverge. On the other hand, too small of a learning rate would make the training procedure go awfully slow, wasting precious resources such as time and computation.

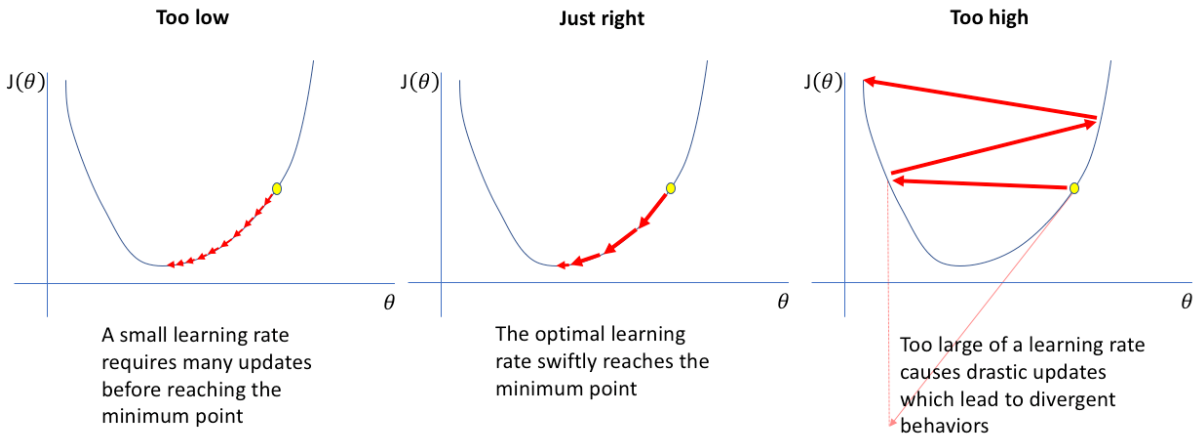


Figure 18 Learning rate and its influence in the optimization of the neural network

Finding the optimal learning rate for a task is a task of its own. In a perfect situation one could try infinite experiments using infinite different learning rates and choose the one that helps the neural network converge the fastest. Since this is obviously impossible one would have to experiment with a few learning rates and choose the best among them. Another way that was recently introduced is the Cyclical Learning Rate Scheduling. The idea is that changing the learning rate value from epoch to epoch seems to have a positive effect during training, accelerating the neural network's convergence to its minima. The problem is that we can not know from the start when we should increase and when to decrease the learning rate. So, the idea of cyclical learning rate was introduced. With this method, the learning changes each epoch. It has a minimum and a maximum value (predetermined from the user). There are several ways the learning rate can go from minimum to maximum but in this work, I used the triangular one. With this method, the learning rate difference is made half at the end of each cycle. This means the learning rate difference drops after each cycle.
Cycle length is the number of iterations until the learning rate returns to the initial value and is defined as:

$$cycle = FLOOR(1 + epochCounter / (2 * stepsize)),$$

$$Stepsize = \frac{cycle}{2}$$

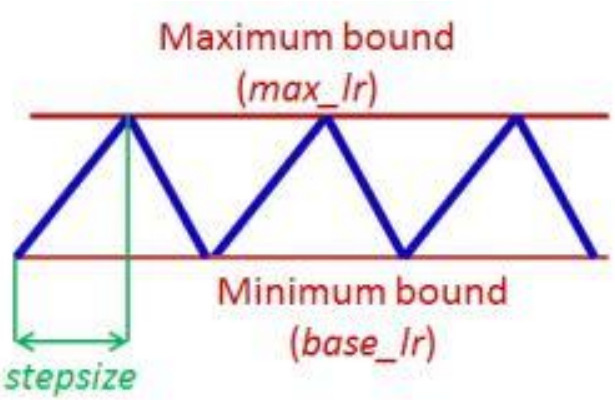


Figure 19 Triangular Cyclic Learning Rate

Convolutional Neural Networks

Convolution operation

Convolutional neural networks (CNNs) were introduced by Yann LeCun in the 1980s. Convolutional neural networks is a way scientists have found to attempt to give vision to computers. The idea is that a computer does not have vision, so it can only perceive an image as a 2-dimensional array of pixel values. In order to make sense of this huge array CNNs were introduced. The idea is that given a 2-dimensional map if we create a 2-dimensional filter (also called kernel) that would slide all over the image, it could in theory learn spatial patterns. So, convolution is a mathematical operation between 2 2-dimensional arrays (the dimensions can change, but for the purpose of explaining the convolution for image recognition I am focusing on the 2-dimensional part). It is denoted usually denoted with an asterisk (*). Assuming I is the image (or the input) and K is the kernel we have the following formula:

$$Convolution(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

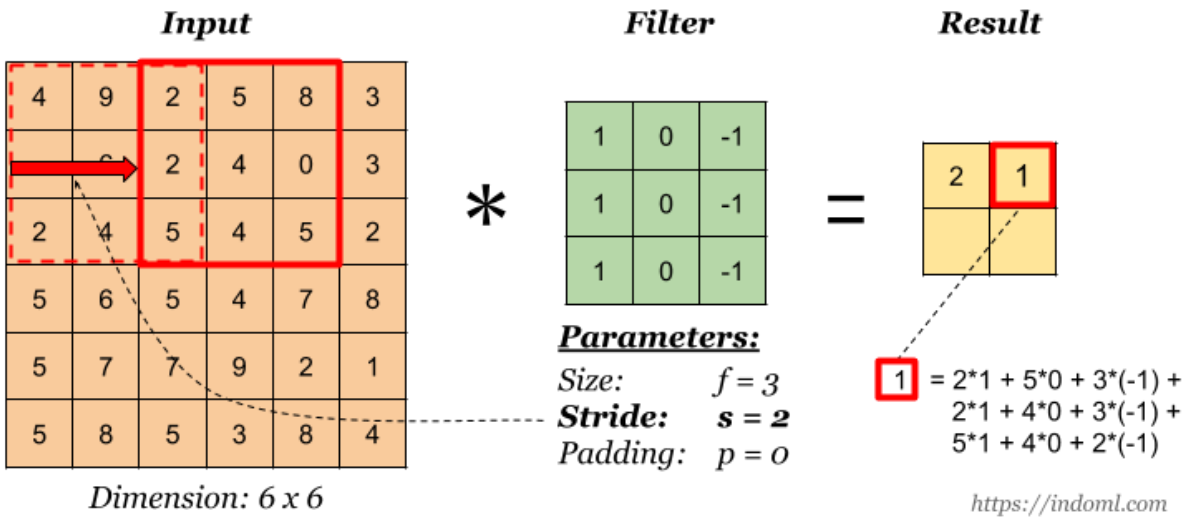


Figure 20 Convolution Operation

A convolutional neural network is an artificial multilayer network that has multiple of these convolutional layers stacked on after another (usually with other layers between them). Each layer could have multiple convolutional filters that feed the next layer which in turn could have multiple convolutional filters and so on.
In CNNs there are a few terms that were introduced to help with problems that might occur.
For starters we have to define how "far" we slide the kernel over the input. The amount by which the filter shifts is the **stride**. In the above image, we are sliding filter by a factor of 2. Stride is not a constant and is different from kernel to kernel.
Another thing that we must take under consideration is the case where the kernel goes over the image and has no values to apply the convolution operation on. In order to avoid having this problem, **padding** was introduced. Padding refers to the number of pixels added to an image when it is being processed by the kernel of a CNN and exceeds the number of pixels that the image has.

Pooling operation

Apart from the convolution operation, pooling operations were also introduced. Pooling works similarly with the convolution operation but instead of applying a kernel function upon a 2-dimensional array, a max or average function is applied. Pooling is used to better help the convolutional layers to "focus" on the image by giving concentrated information of a certain space on the image. There are 2 pooling operations. Max and average pooling. As their names suggest, max pooling operation takes the largest number of the input it has been applied on, and average pooling takes all values and returns their average value. Below there is an image of how the pooling operations work.

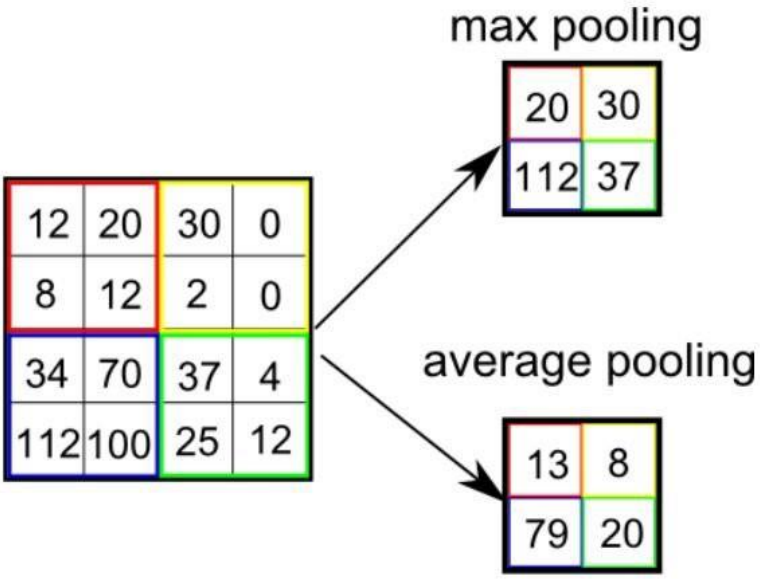


Figure 21 Pooling Operation

Training CNNs works the same way with a typical MLP network but instead of synaptic weights, the kernels are the ones that change value and are optimized through the network.
A typical CNN consists of multiple convolution, pooling and activation layers followed by a fully connected (or dense) layer that is responsible for the classification task. An intuitive figure is shown below.

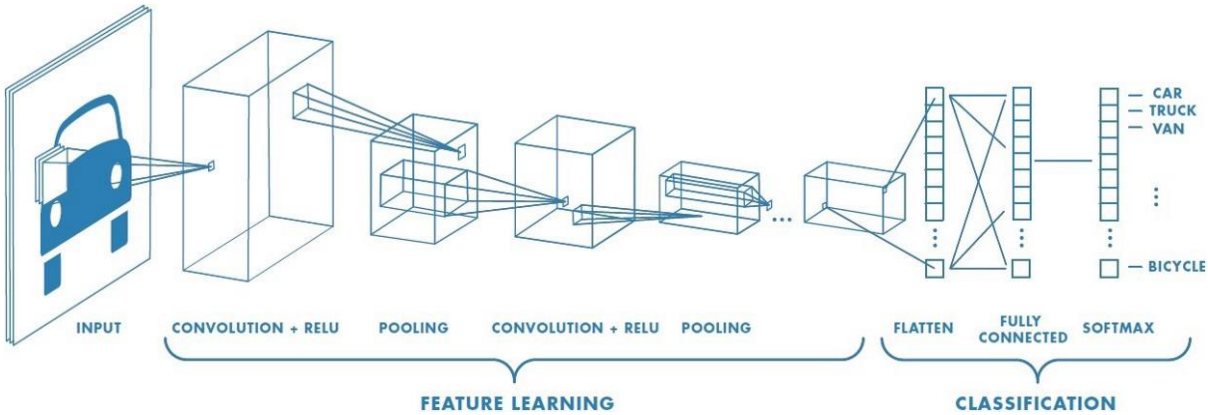


Figure 22 A convolutional Neural Network (CNN)

Batch normalization

Another interesting concept that should be paid attention is **batch normalization**. Training such big neural networks proves to be a very tricky job in practice. The tricky part stems from the fact that each layer's weights change one-by-one backwards from the output to the input using an estimate of error that assumes the weights in the layers prior to the current layer are fixed. Because all layers are changed during an update, the update procedure is forever chasing a moving target. This problem in the bibliography is referred as "internal covariate shift". In order to fix this problem, the concept of batch normalization (also referred as bn in bibliography) was introduced. The idea is that

in order to somehow stabilize the network during training and coordinate the changes of weights of each layer. In order to do that we standardize each input variable per mini-batch, such as the activations of a node from the previous layer. Standardization refers to rescaling the data of each mini batch to have a mean value of zero and standard deviation of one. In image processing/computer vision, the standardization process is also referred to as whitening. Ideally, the normalization would be conducted over the entire dataset, but since the datasets that are used to train deep networks are quite large it is impractical to do this, and so we use mini-batches and conduct bn on each mini-batch respectively. Assuming μ is the mean, σ is the standard deviation and $\hat{x}_i$ each sample we have the following:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2}}}$$

Convolutional Neural Network Architectures

Given the tremendous recognition and research that there is in the field of convolutional neural networks there are many architectures. In this work I use 2 of them that have been benchmarked in various image classification tasks and seem to do exceedingly well.

DenseNet

The DenseNet architecture was introduced in 2018. After the success of the Residual Networks, Dense networks came into the picture. The idea is that in a convolutional neural network, each layer feeds its output to the next layer as input and so forth until the last (classification layer) is reached. DenseNet introduced the "dense blocks". Here, instead of only passing the outputs as inputs to the next layer, they pass the inputs themselves as outputs. The following graph will help visualize.

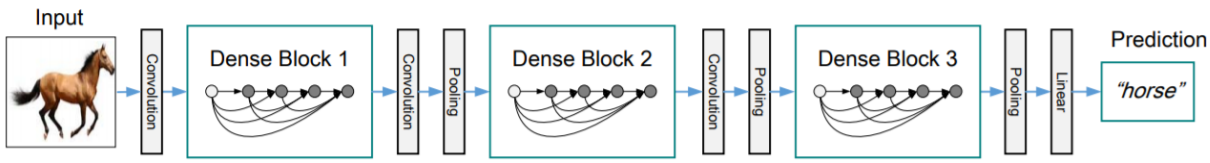


Figure 23 DenseNet

By giving the inputs as outputs, DenseNets manage to achieve a couple of things. For starters it helps alleviate the vanishing gradient problem we saw in the previous chapter, and by extension it makes the training procedure easier without having to worry as much. Secondly, it helps the network "remember" what the previous layers "saw" and even combine knowledge from the inputs of previous blocks. The n-th layer receives the feature-maps of all preceding layers x_0, …, x_{n-1}, as input:

$$x_n = \mathrm{H}([x_0, x_1 \ldots x_{n-1}])$$

In order to better organize and furthermore to put the architecture in code more easily, they introduced **dense blocks.** A dense block is a composite function of three consecutive operations: (i) batch normalization, followed by a rectified linear unit (ReLU) and a 3x3 convolution. A DenseNet is multiple dense blocks stacked. As most modern convolutional neural networks, DenseNet architecture comes in different depths, depending on each use case's needs.

EfficientNet

EfficientNets were introduced in 2020. Deep convolutional networks are often overparameterized. The idea is to find a way to balance performance and efficiency. To do that they introduce compound scaling. Neural network scaling in general can be done with the following ways:

- Width scaling. Scaling network width is commonly used for small size models
- Depth scaling. The intuition is that a deeper network can capture richer and more complex features and generalize well on new tasks. However, deeper networks are also more difficult to train due to the vanishing gradient problem
- Resolution scaling. With higher resolution input images, ConvNets can potentially capture more fine-grained patterns. Starting from 224x224 in early ConvNets, modern ConvNets tend to use 299x299 for better accuracy.
- Compound scaling (new). Using this method, scaling balances out, and the result seems to outperform previous models with more traditional scaling models, while also not increasing complexity too much

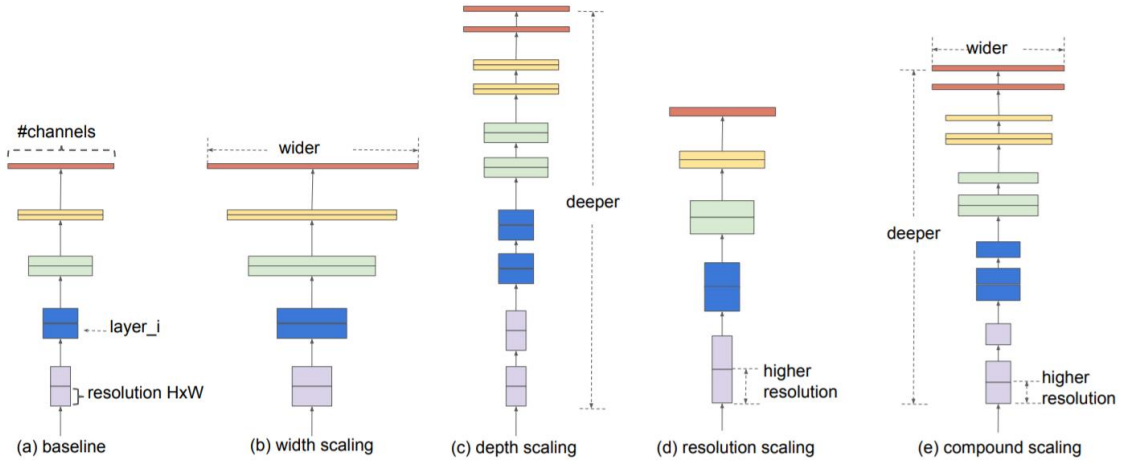


Figure 24 EfficientNet Scaling

EfficientNets are compound scaling convolutional neural networks, that like any other modern convolutional neural network comes into different depth architectures.

Datasets – Data Exploration

ChestX-ray8 – NIH – National Institute of Health

In 2017, the American National Institute of Health published a paper called "ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases" and along with it, they made public a huge (for that time) dataset of frontal thoracic x-rays. It is comprised of 108,948 frontal-view x-ray images from 32.717 unique patients along with the text-mined disease labels from the associated radiological reports. There were previous attempts for a big dataset of thoracic x-rays, but up until then, none even came close to its size. Apart from the size that this dataset offered, it offered a great diversity in disease-labels. The ChestX-ray8 dataset has the following classes: **atelectasis, cardiomegaly, effusion, infiltrate, mass, nodule, pneumonia, pneumothorax, normal.** To an untrained eye, the only class which could be predicted is cardiomegaly. The "normal" label indicates that the current sample is totally healthy. All the previous datasets, offered at most one class (e.g. pneumonia, one of the most common labels in medical datasets).
Given the size of the dataset, manually extracting the text labels would be extremely inefficient, so automated processes were used. The labels were text-mined from the radiological reports using two tools, DNorm and MetaMap. Both tools are machine learning based, and they are both focused on medical/clinical terminology. DNorm maps every keyword in the radiological report to a unique concept in a standardized dictionary which contains clinical health information. MetaMap is a similar tool. It is an ontology-based approach for the detection of Unified Medical Language System. In order to maximize accuracy, they merge the results of the two models.
One of the challenges those tools must face is the uncertainty and negation in some reports. Phrases like "suggesting X disease" can throw off the models. Negation also poses a significantly difficult problem since the model cannot be completely sure as to when the negation ends. Phrases like "clear of disease A and disease B" the models could easily be thrown off and assign a negative sample for disease A, but completely omit the negation on disease B, and assign a positive sample. In order to circumvent this problem, they wrote a few hand-written rules in order to help the system better process the radiological reports. But even with the extra rules the label extraction is still an approximation and could contain mistakes.
The samples from this dataset were imbalanced. Every sample could have one or more of the 8 diseases. So, each sample is mapped to an 8-dimensional vector

$$\boldsymbol{y} = [y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8], y_c \in 0,1$$

A healthy sample would simply be mapped to an all-zero vector $\boldsymbol{y} = [0,0,0,0,0,0,0,0]$. The frequency of each disease makes the classification even more challenging. More than half the samples in this dataset (53%), are classified as "healthy". As for the non-healthy samples, we see that in general there are very few samples for each class.

Class/Disease	Positive Samples	Negative Samples
Atelectasis	11559	100561
Cardiomegaly	2776	109344
Consolidation	4667	107453
Edema	2303	109817
Effusion	13317	98803
Emphysema	2516	109604
Fibrosis	1686	110434

Hernia	227	111893
Infiltration	2303	109817
Mass	1431	110689
Nodule	1431	110689
Pleural Thickening	3385	108735
Pneumonia	1431	110689
Pneumothorax	5302	106818

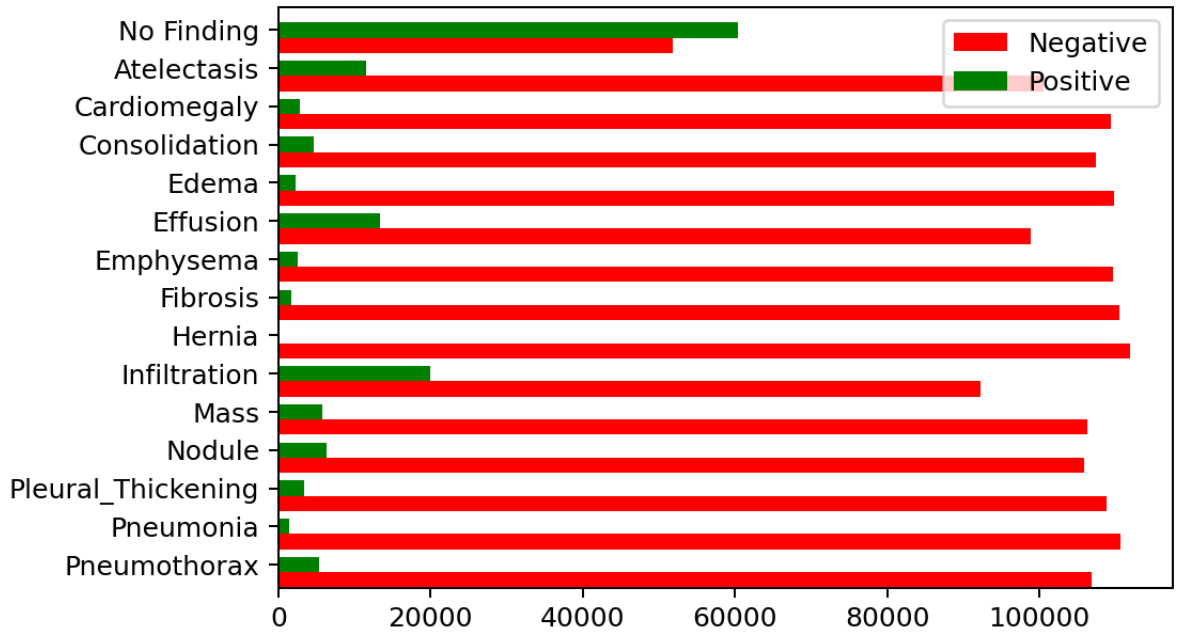


Figure 25 ChestX-ray8 dataset, class frequencies

It is clear that the healthy samples are overwhelmingly more and makes the classification task even more challenging and basically forces us to explore ways to balance the problem. This part is extremely important, since in literature, we often see authors proposing models and new methodologies that "achieve 90%" in accuracy and other evaluation metrics while at the same time, completely omitting the number of samples of X class in the testing dataset and thus publishing biased results that don't necessarily translate to a good model or methodology.
Along with the labels, this dataset provides us with 984 bounding boxes for the corresponding disease occurrences. Those bounding boxes were provided by doctors manually annotating the x-rays and if they were more they could be used to train an object detection algorithm to not only find the disease, but actually indicate the radiologist to where the problem lies. The images are 1024x1024 pixel grayscale PNG images. The vast majority of the images were crystal-clear, and the few that were blurry were left there since they were so few, it would hardly affect the model's performance. All together the dataset's size is 42.1GB. In deep learning tasks, it is a common practice, to pre-load the dataset in the computer's memory in order to access it faster while training. Given its staggering size, this was unfeasible, and thus the dataset was "chopped" in batches. The number of samples in each batch varied depending on the model's architecture (and the memory it needed to fit in the GPU). This paper shows just how difficult it is to create a dataset that is both good in image quality and balanced in terms of class occurrence frequency.

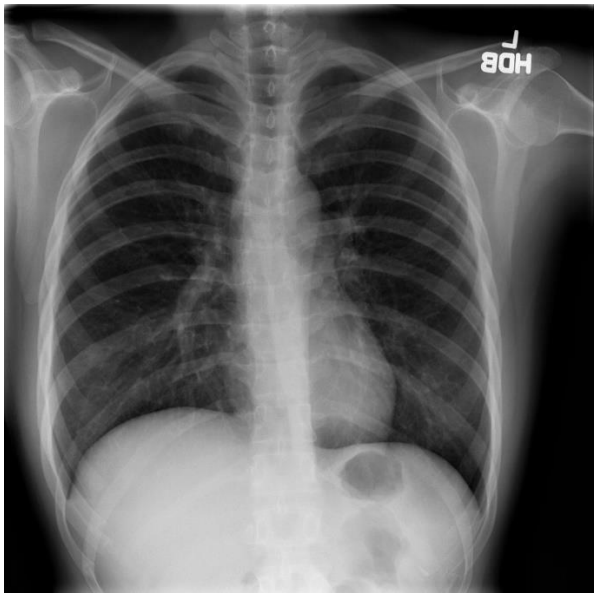


Figure 26 Healthy sample

Figure 27 Cardiomegaly - Effusion

CheXpert Dataset

In 2019, Stanford University published a paper called “CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison” and along with it, made public another x-ray dataset containing 224,316 chest x-rays, from 65.240 unique patients. The images themselves were Like in the ChestX-ray8 dataset, the radiological reports were text mined using automated tools for the same reasons (manually annotating images using each radiological report would be tremendously costly in both time and human resources). One of the key differences between the 2 datasets is that the ChestX-ray8 dataset contains only frontal radiographs, whereas CheXpert contains both frontal and lateral ones. Apart from the great number of samples, this dataset also introduces two new concepts. First is their different approach in how the labels affect one another. In the ChestX-ray8 dataset, every class occurrence was completely independent and had no effect on other classes. In the CheXpert dataset, we see a different approach. Most of the classes/diseases we are trying to identify, also belong to a larger set of diseases. They used the following structure.

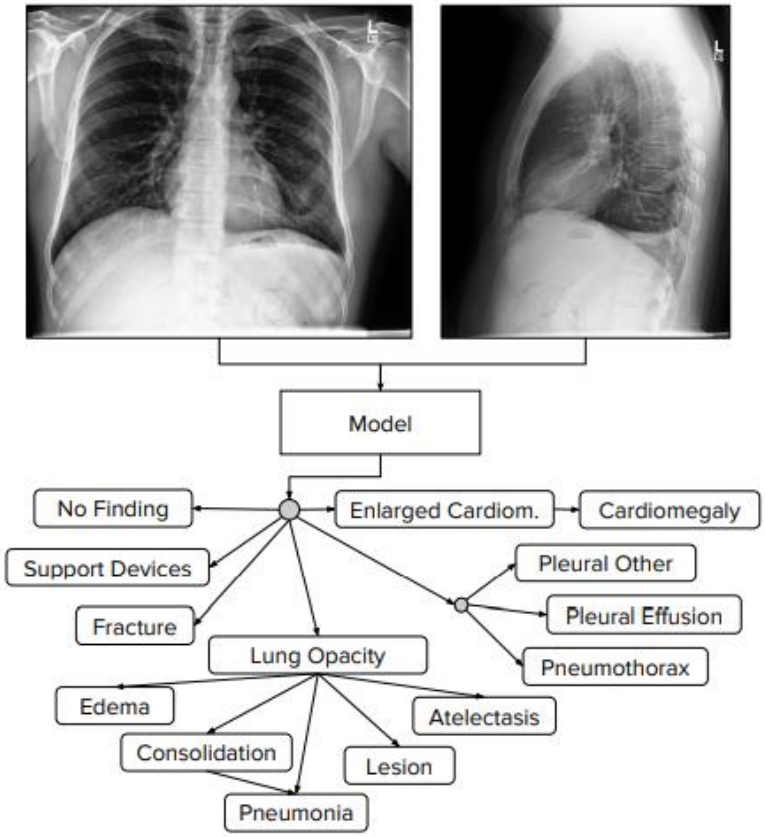


Figure 28 CheXpert Dataset Tree diagram

- Cardiomegaly is a subset of enlarged cardiomediastinum
- Edema is a subset of lung opacity
- Consolidation is a subset of lung opacity
- Lesion is a subset of lung opacity
- Atelectasis is a subset of lung opacity
- Pneumonia is a subset of consolidation

In this dataset, no finding means that there **no occurrence** of diseases that the dataset focuses on could be found, unlike the "No finding" label in the NIH dataset, where it meant that the x-ray belonged to a **healthy** person.
The tree structure of the dataset aims to help scientists train their models in stages. First train on the parent classes and then fine-tune the model to the child classes.
Next, they introduce the concept of class **uncertainty**. In a real-world situation, a radiograph could be indicating a disease, but the doctor cannot be completely sure without further examining the patient or ordering more tests. This uncertain state could be helpful in training a better and more accurate model.
The class frequency of each disease is much more balanced than the frequency in the NIH dataset. Here, almost 90% of the samples are unhealthy. Like in the NIH dataset, the frequency of each class is not balanced, but it is significantly better, providing us with at least 3500 samples per class.

Class	Positive Samples	Negative Samples	Uncertain Samples
No Finding	22419	201229	0
Atelectasis	33456	156453	33739
Cardiomegaly	27068	188493	8087
Consolidation	19841	176610	27197
Edema	52291	158373	12984

Enlarged Cardiomediastinum	36006	175550	12092
Fracture	9040	213966	642
Lung Lesion	9187	212973	1488
Lung Opacity	151641	68524	3483
Pleural Effusion	86254	125766	11628
Pleural Other	3524	217471	2653
Pneumonia	6047	198831	18770
Pneumothorax	19456	201047	3145

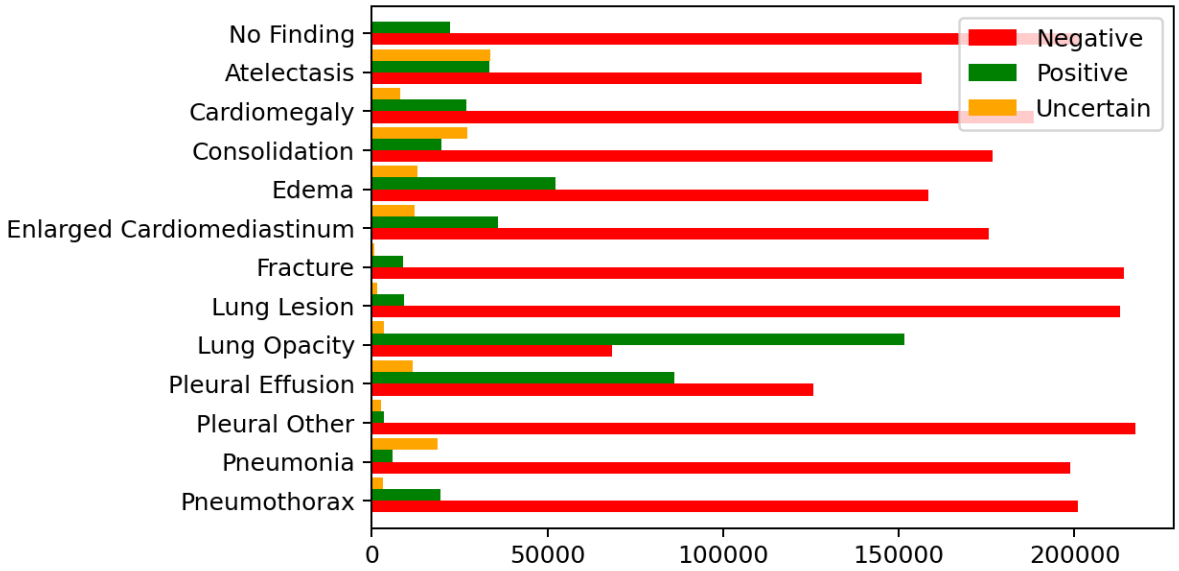


Figure 29 CheXpert dataset, class frequencies

Problem formulation and Methodology

Problem – Task

Thoracic x-ray is the most commonly asked and used type of radiograph. Automating the process of interpretation of said radiographs, could significantly help the medical field in various situations. It could pre-screen patients in order to find healthy ones, and exclude them from various and unnecessary tests, thus saving the already limited resources, or it could be a "guiding tool" for a radiologist, helping him better interpret the radiograph. In this work, I used the 2 datasets to train various deep learning models to classify the radiographs. Stanford university also hosted a classification competition, focusing on the following classes: **Atelectasis, Cardiomegaly,**

Consolidation, Edema, Pleural Effusion. The reason behind this choice, is probably because these are the most common problems in a chest radiograph. In this work, I too focus on those classes.

Methodology

Data preparation

In this work, I decided it would benefit me, to use both datasets. Using both was tricky, since not every column/class matched. Specifically, atelectasis, cardiomegaly, consolidation and edema exist in both datasets, but pleural effusion does not exist in the NIH dataset. So, the experiments I would run had to take this absence of the class into consideration. Later I explain how every experiment took place. The CheXpert dataset, as mentioned, contained "uncertain" samples. In the paper, Stanford scientists took 3 approaches and tested them.

1. Consider every uncertain sample as a positive one
2. Consider every uncertain sample as a negative one
3. Use the uncertain label as an extra class. So, each sample could be classified as positive, negative or uncertain.

In their paper, every approach yielded different results, and none of them outperformed the other in all classes. In this work I use the first approach and consider every uncertain sample a positive one. The thinking behind this decision was that the problem at hand, **has far more serious consequences when classifying a sample as false negative than if it would classify it as false positive.**
Next step was to fix inconsistences in the CheXpert dataset. The first major one, were the missing values. Most of the samples did not have values at all at certain classes. This was because of the NLP technique they used to extract the class labels. When a disease was not mentioned, it did not automatically assign a negative value. So, all I did was replace empty values with 0s. The second and more concerning problem, not all samples followed the tree structure. It goes without saying, that when a sample was positive for a disease that belonged to a child class, it automatically means that it also belongs to its parent class. This was not the case. Many samples did not follow this structure, and most of the samples even though were positive or uncertain for a child class, had an empty value for its parent classes. This was most likely for the same reason as the previous problem. Diseases that were not mentioned, were not given a value. It was fixed in the same way, every sample containing a positive or an uncertain sample of child class was assigned a positive value to the parent classes of said class.
For example: Sample A is positive for cardiomegaly but enlarged cardiomediastinum is empty. This changes to a positive.
Same thing would apply even if A had uncertain label for cardiomegaly.
Next, the images themselves had to be reduced in size. Originally, they were 320x320 jpg images, but given that the convolutional neural networks I used took as input images of size 224x224, they were all resized.

The multilabel stratification problem

The task at hand is a multi-label classification problem. Given the tremendous imbalance of each class, splitting the dataset into train-validation-testing proved to be a problem, since when I used sklearn's function, it would randomly assign samples to train-validation-split and the datasets would end up with extreme imbalances in class frequencies. For example, edema could end up having 0 positive samples in the train set and all of them to be put in validation-testing. This would, of course, result in the network not learning at all how to classify edema cases. One thing that had to be taken into consideration was co-occurrences of classes. For example, a sample could contain all 5 focused

classes or a subset of them. In order to bypass this problem, I used the following algorithm to better balance the 3 splits.

Algorithm for multilabel stratification

```
train-validation = []
test = []
calculate all possible labelset combinations
for every combination c in labelset combinations (from largest in size to smallest):
       train-validation = train-validation + 80% random samples from c
       test = test + the rest of the samples left in c
```

The algorithm is quite simple but effective. It was only possible to apply this algorithm because the number of classes was so small. Multilabel classification problems are quite common in the world of machine learning today, but the number of classes is most usually much larger. Since the complexity of the algorithm is exponentially increasing with each class even one more class would rend the algorithm completely useless. A good example of such a problem is classifying videos by content. YouTube videos have an enormous number of categories.

- Unboxing videos
- Educational videos
- Favorites/Best of videos
- Tag or challenge videos
- Comedy/skit videos
- Gaming videos
- Vlogs
- How-To videos
- Product Review videos

The number of possible labelsets/combinations in this example is 512.
Thankfully, the number of classes of my task was 5 so the possible labelsets was 32. By applying this algorithm, I hoped to better split the dataset and eliminate imbalances in class frequencies between train-validation set and the test set.

Grayscale images and Information waste

An important property of radiographs is that they are represented as grayscale images. A digital image is composed of 3 channels, red, green and blue (RGB). So, when looking closer, an image is a 3-dimensional array, where each dimension represents each channel. So, an image contains N pixels

$$Image = N = width * height * channels, \;\; channels = 3$$

The difference in a grayscale image, is that every channel contains the exact same information as all the other channels. So, it is basically an array of 3 dimensions, where each dimension is equal to all the other ones.

$$Grayscale\ Image = N = width * height * channels, \;\; channels = 3$$
$$Red\ channel \; = \; Green\ channel \; = \; Blue\ channel$$

This is a big information waste. Considering that an artificial neural network is a function approximation algorithm, we can assume that it processes the image somehow and finds optimal ways to transform the original image and then find key features that will lead to the better classification. Given that we have 2 channels that are essentially useless, I had 2 choices. Either modify the network architectures to accept images of 2 dimensions (i.e. remove the 2 extra useless channels) or

experiment with the 2 extra channels and keep the 3-dimensional form of the images. I decided to do the latter.
I swapped those extra channels and replaced them with:

- One channel that is the original image after being passed from a histogram equalization filter
- One channel that is the original image after being passed from a contrast increasing filter

Histogram equalization on radiographs is a commonly used way that radiologists use to better assess the state of an x-ray. Increasing the contrast of the x-ray was a personal idea that just seemed like the logical next step, and I do not know if there is any validity in it since I have no medical expertise or any doctor to confirm my hypothesis.

Training

Training neural networks for this task proved to be even trickier than I initially thought. The dataset stratification solved the frequency imbalance between the train set and the test set, but the core problem remained. The negative samples to positive samples ratio was still exceedingly high, making the classification problem even more challenging.

Class	Negative to Positive ratio
No Finding	8.97582408
Atelectasis	**2.32834288**
Cardiomegaly	**5.36176931**
Consolidation	**3.75462392**
Edema	**2.42624282**
Enlarged Cardiomediastinum	**3.64983991**
Fracture	22.0993596
Lung Lesion	19.9506323
Lung Opacity	**0.441736933**
Pleural Effusion	**1.28487362**
Pleural Other	35.2065728
Pneumonia	8.01188701
Pneumothorax	8.89549135

Typically, the loss function which is used for multilabel classification problems is the binary cross entropy loss function. For 1 class binary classification tasks the loss can be expressed as:

$$Binary\ Cross\ Entropy(y_{true}, p) = -(y_{true} * \log_e p + (1 - y_{true}) * \log_e(1 - p))$$

Furthermore, when the task is multiple binary classification tasks (multilabel classification), the loss will be the sum of all the classification loss results

$$Binary\ Cross\ Entropy(y_{true}, p) = -\sum_{c=1}^{M} y_{true} * \log_e p$$

Since the datasets are much more “negative-oriented” the trained networks would most likely overfit to classify most samples as “negative”. To avoid this problem, I derived from the work of CheXNet and modified the cross-entropy loss. More specifically, the modified loss is

$$Weighted\ Binary\ Cross\ Entropy(y_{true}, p) = -\sum_{c=1}^{M} w * y_{true} * \log_e p\,, w = weight$$

$$w_+ = \frac{|P|}{|P| + |N|}$$

$$w_- = \frac{|N|}{|P| + |N|}$$

Since the dataset was extremely large to fit the 12GB of memory my GPU had available, I had to train in batches of 8 samples.

Experiments – Results

After careful consideration, I decided to not use the ChestX-ray8 dataset since the imbalance it would provide would only worsen the already high imbalance that existed in the CheXpert dataset. Two series of experiments were conducted. The structure of the experiments is the following:

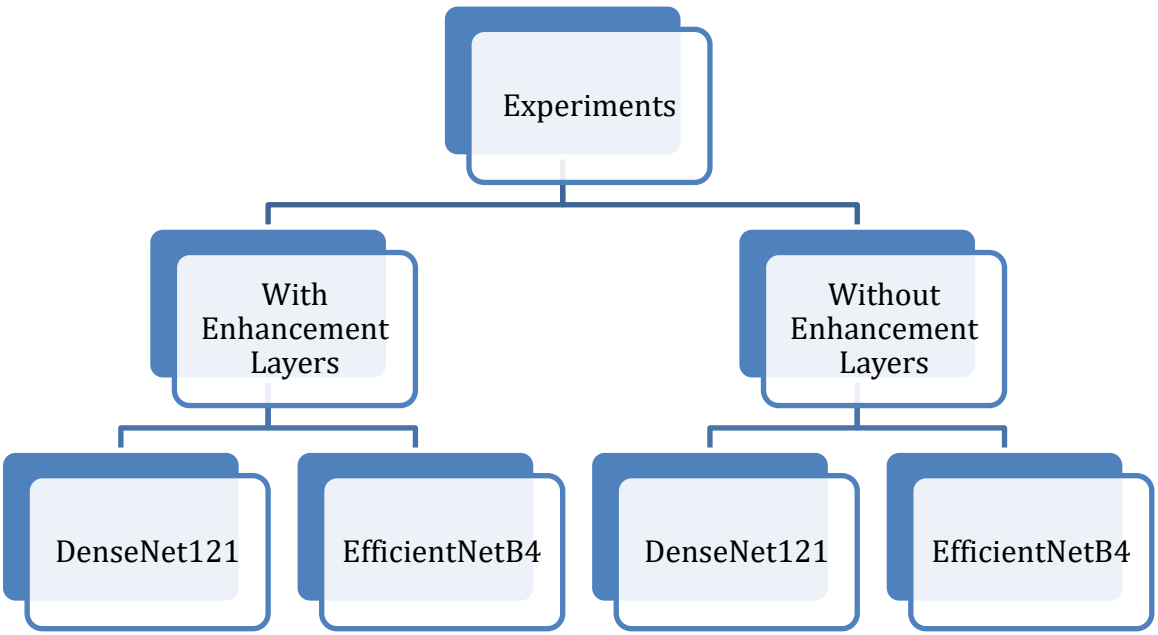


Figure 30 Experiments structure

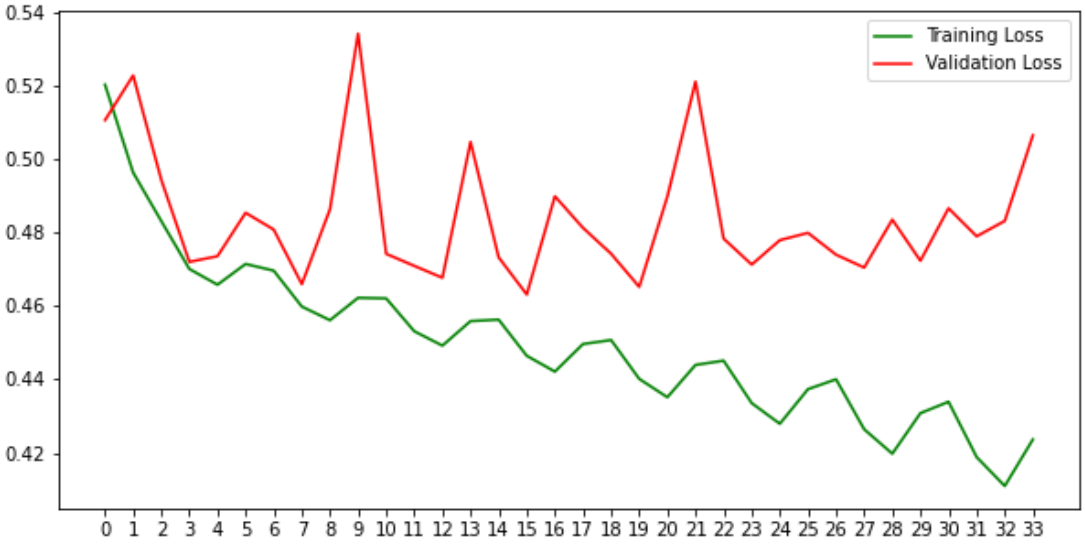


Figure 31 DenseNet121 – With Enhancement Layers

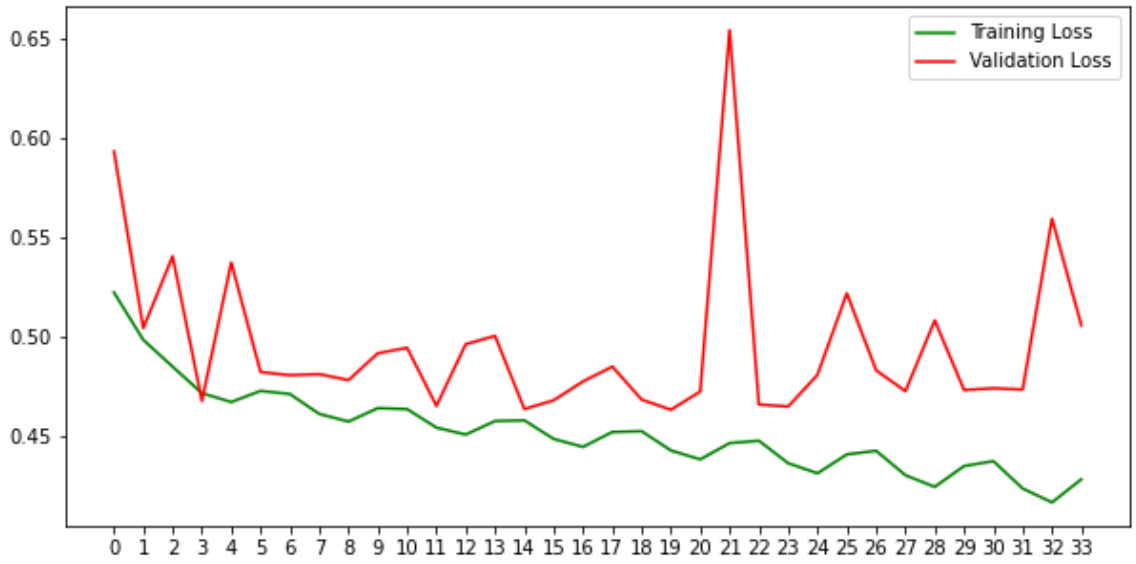


Figure 32 DenseNet121 - Without Enhancement Layers

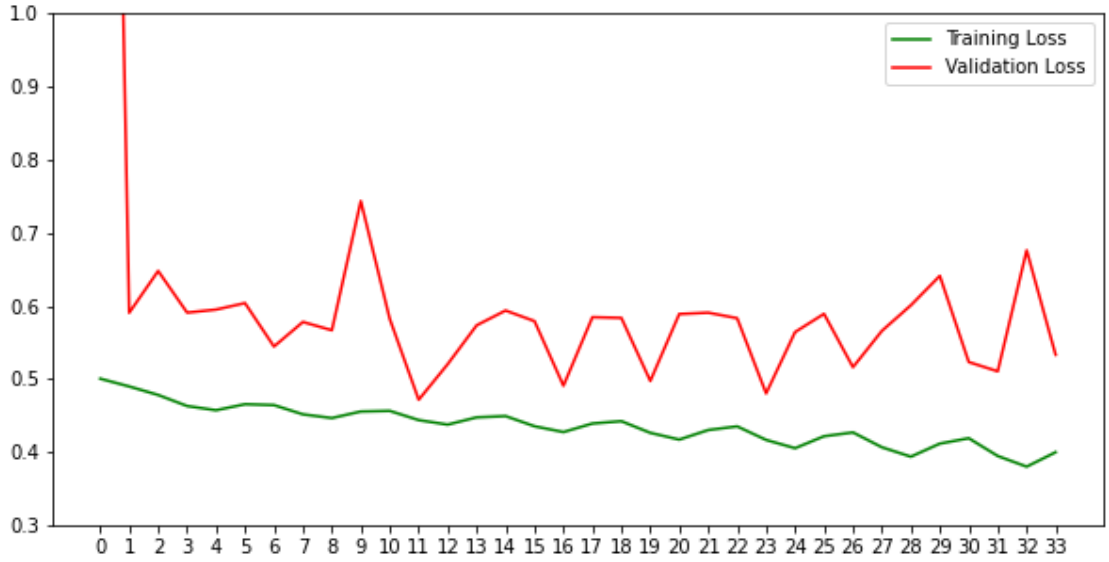


Figure 33 EfficientNetB4 – With Enhancement Layers

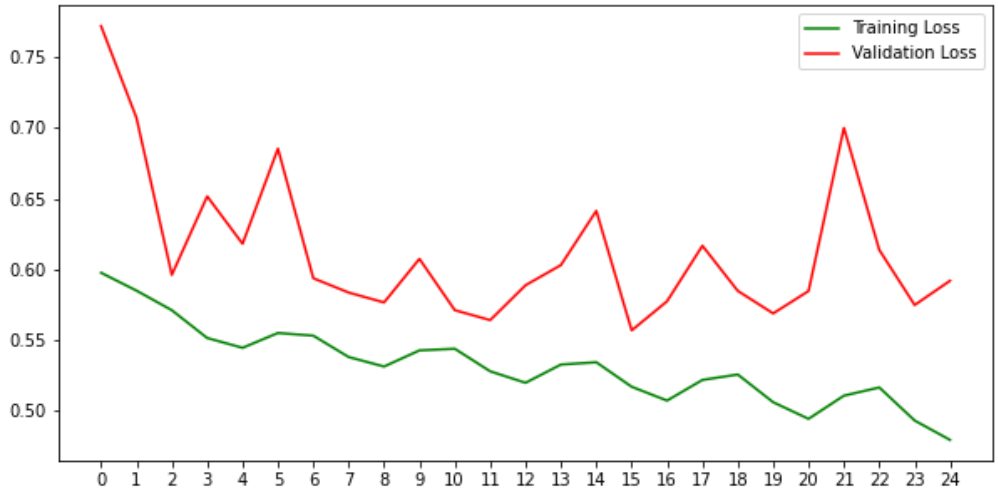


Figure 34 EfficientNetB4 – With Enhancement Layers

It is worth noting, that due to the cyclical learning rate scheduling, the spikes in validation loss are due to the resetting of the learning rate back to its maximum value.

Table 1 Model architectures with their respective AUC values

Model Name	Enhancement layers	Multilabel AUC	Atelectasis AUC	Cardiomegaly AUC	Consolidation AUC	Edema AUC	Pneumothorax AUC	Accuracy
DenseNet121	True	0.79	0.697	0.696	0.697	0.698	0.696	0.782
DenseNet121	False	0.77	0.679	0.679	0.679	0.679	0.679	0.77
EfficientNetB4	True	0.79	0.70	0.70	0.70	0.70	0.70	0.778
EfficientNetB4	False	0.78	0.69	0.69	0.69	0.69	0.69	0.775

Table 2 Model architectures with their respective summed confusion matrices

Model Name	Enhancement layers	True Positives	False Negatives	True Negatives	False Positives
DenseNet121	True	13474	14551	61925	6450
DenseNet121	False	11760	16265	62580	5795
EfficientNetB4	True	11476	16549	63618	4757
EfficientNetB4	False	11671	16354	63098	5277

Table 3 DenseNet121 with enhancement Layers Separate confusion matrices

Atelectasis		Cardiomegaly		Consolidation		Edema		Pleural Effusion	
TP = 1510	FP = 1333	TP = 1318	FP = 879	TP = 147	FP = 150	TP = 3694	FP = 1868	TP = 6803	FP = 2201
FN = 4495	TN = 11942	FN = 1757	TN = 15326	FN = 3955	TN = 15028	FN = 2481	TN = 11237	FN = 1865	TN = 8411

Table 4 DenseNet121 with enhancement Layers Separate Metrics

	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Average
True Positive Rate	0.251	0.428	0.035	0.598	0.784	0.4192
True Negative Rate	0.899	0.945	0.990	0.857	0.792	0.8966
F-score	0.341	0.5	0.066	0.629	0.769	0.46

Table 5 DenseNet121 No enhancement Layers Separate confusion matrices

Atelectasis		Cardiomegaly		Consolidation		Edema		Pleural Effusion	
TP = 444	FP = 379	TP = 1669	FP = 1676	TP = 225	FP = 281	TP = 2964	FP = 1354	TP = 6435	FP = 2085
FN = 5561	TN = 12896	FN = 1406	TN = 14529	FN = 3877	TN = 14897	FN = 3211	TN = 11751	FN = 2233	TN = 8527

Table 6 DenseNet121 No enhancement Layers Separate Metrics

	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Average
True Positive Rate	0.073	0.542	0.054	0.48	0.742	0.37
True Negative Rate	0.971	0.896	0.981	0.896	0.803	0.909
F-score	0.130	0.519	0.097	0.564	0.748	0.411

Table 7 EfficientNetB4 with enhancement Layers Separate confusion matrices

Atelectasis		Cardiomegaly		Consolidation		Edema		Pleural Effusion	
TP = 1108	FP = 933	TP = 1214	FP = 753	TP = 72	FP = 77	TP = 2255	FP = 742	TP = 6834	FP = 2201
FN = 4897	TN = 12342	FN = 1861	TN = 15452	FN = 4030	TN = 15101	FN = 3920	TN = 12363	FN = 1834	TN = 8411

Table 8 EfficientNetB4 with enhancement Layers Separate Metrics

	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Average
True Positive Rate	0.184	0.394	0.017	0.365	0.788	0.34
True Negative Rate	0.929	0.953	0.994	0.943	0.792	0.922
F-score	0.275	0.481	0.033	0.491	0.77	0.41

Table 9 EfficientNetB4 No enhancement Layers Separate Confusion Matrices

Atelectasis		Cardiomegaly		Consolidation		Edema		Pleural Effusion	
TP = 4649	FP = 6914	TP = 2424	FP = 4454	TP = 3004	FP = 7408	TP = 2384	FP = 983	TP = 6752	FP = 2442
FN = 1356	TN = 6361	FN = 651	TN = 11751	FN = 1098	TN = 7770	FN = 3791	TN = 12122	FN = 1916	TN = 8170

Table 10 EfficientNetB4 No enhancement Layers Separate Metrics

	Atelectasis	Cardiomegaly	Consolidation	Edema	Pleural Effusion	Average
True Positive Rate	0.774	0.788	0.732	0.386	0.778	0.691
True Negative Rate	0.479	0.725	0.511	0.924	0.769	0.681
F-score	0.529	0.487	0.413	0.499	0.756	0.536

Out of the 2 networks I would choose the EfficientNetB4 model with no enhancement layers since it scores the most. Due to the nature of the problem, we must choose the one with the best true positive rate, since not identifying a potential problem, is a lot worse than falsely predicting there is one. It is worth noting that the enhancement layers slightly improved the results with the DenseNet121 architecture but seemed to have no, to almost negative effect with the EfficientNetb4 architecture. The results seem to still be inclined towards the negative class even though the loss function that was used tried to alleviate this problem.

Hardware Requirements

Hardware and computational times were and still are a major problem in tasks that employ deep learning solutionsThe number of operations that must be executed is enormously big. The one thing that we can take advantage of is the parallelization of the operations. CPUs execute operations in a serial fashion making them incredibly time consuming. GPUs on the other hand, are designed by default to execute operations in a parallel fashion and therefore are excellent for training deep learning models. All of the above experiments were conducted with an Nvidia Titan Xp GPU

Model Name	Time for 34 epochs
Densenet121 with Enhancement Layers	824.10 minutes or 13.7333 hours
Densenet121 without Enhancement Layers	738.37 or 12.3 hours
EfficientNetB4 with Enhancement Layers	1867.43 minutes or 31.1 hours
EfficientNetB4 without Enhancement Layers	1810.42 minutes or 30.1 hours

Densenet121 architecture was much faster during training, needing almost half the time of the EfficientNetB4. The difference in times between with and without enhancement layers in the training of the same architecture is because the addition of the enhancement layers was executed

on the fly since saving it would consume almost 14 extra gigabytes in storage which I did not want to waste on the server.

Software

Right now, there are 2 main open-source deep learning libraries that provide GPU support. Tensorflow as and PyTorch. **Tensorflow** is developed and maintained by Google whereas **PyTorch** is developed and maintained by Facebook. In the initial part of this work, I experimented with both frameworks with several trivial introductory tasks and in the end, I decided to work with Tensorflow. Tensorflow has a very convenient higher-level API called Keras. **Keras** contains many convenient tools that help speed up the process of writing code, such as a "fit" function. In PyTorch I would have to write by myself the whole training function from scratch which although interesting seemed like "reinventing the wheel" at this point. Another interesting feature that at the time I started writing code was that calling pre-processing functions was much simpler in Keras than in PyTorch.

Conclusions - Future Goals – Improvements

After the end of the experiments I had many thoughts about the future and how I would like to improve upon my work. The test results were not as good as I would hope them to be. Seeing the CheXpert ladder on the official website of the challenge my results seem to not be as good as the top results. Seeing closer though, we can notice that the top 10 models are ensemble models. As much as I would like to experiment with ensemble models, both the computational and the time cost would be just too much. The second place in the competition has also taken advantage of the tree structure of the classes. Their idea was that if they could pretrain a convolutional neural network in the "parent classes" and then freeze training on all layers except the last ones. This way the network would have a general knowledge of the underlying data distribution and could therefore finetune the last layers to classify the "child" classes. This idea combined with the ensemble models seemed to have given them the second place in the challenge ranking ladder. Furthermore, I believe that since the X-rays depict not only the chest, but often a part of the abdomen, the trained model would benefit from isolating the lungs themselves. Giving the model less "noise" would improve its results, but the cost of manually annotating bounding boxes for the lung part of each Xray would be high since we are talking about more. A good but most likely infeasible idea is to take the positive samples of the dataset and ask a doctor to localize the problems thus turning the classification task into a classification and localization task. If I had localization data, I could use more techniques. An extremely good localization network is the YOLO architecture. YOLO has been trained and used for many different tasks, and I see no reason why it would not work with this task.
The main problem that I faced in this work was the severe class imbalance. The negative samples for each class were overwhelmingly more making the training procedure challenging. This problem could be solved in 2 ways. We could always obtain more images with their corresponding annotations hoping that we get more positive samples. Another way would be to artificially create more positive samples by employing Generative Adversarial Networks
Lastly a big future step would be to make a comparative study using various convolutional neural network architectures, with different setups (learning rate schedulers, different optimizers, pretrained on parent classes). The size of this study and the time it would take for the experiments to finish would be too big and would not be done in 6 months.

In general, the task was remarkably interesting and challenging. My main problem, especially when compared to other machine learning tasks, is that I could in no way intervene in the datasets themselves. In Natural Language Processing tasks for example, I would have been able to make changes or simply understand how well the model does by evaluating it myself without relying completely on the metrics like in this task.

Bibliography

Bishop, C. M. (n.d.). *Pattern Recognition and Machine Learning.*

Bugnon, L. A., & C. Yones, D. H. (2018). Deep neural architectures for highly imbalanced.

Cohen, J. P., Bertin, P., & Frappier, V. (2019). Chester: A Web Delivered Locally Computed Chest X-Ray.

Gao, X. W., Hui, R., & Tian, Z. (2016). Classification of CT brain images based on deep. *Elsevier*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.

Guan, Q., Huang, Y., Zhong, Z., Zheng, Z., Zheng, L., & Yang, Y. (2018). Diagnose like a Radiologist: Attention Guided.

Guo, S., Wu, X., & Luo, Z. (2008). Automatic Extraction of Control Points for.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition.*

Huang, G., Liu, Z., & Maaten, L. v. (2018). Densely Connected Convolutional Networks.

Islam, M. T., Aowal, M. A., Minhaz, A. T., & Ashraf, K. (2017). Abnormality Detection and Localization in Chest X-Rays.

Jaynes, E. T. (n.d.). *probability theory the logic of science.*

Jing, B., Xie, P., & Xing, E. P. (2018). On the Automatic Generation of Medical Imaging Reports.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional.

Lakhani, P., & Sundaram, B. (n.d.). Deep Learning at Chest Radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks.

Li, Z., Wang, C., Han, M., Xue, Y., Wei, W., Li, L.-J., & Fei-Fei, L. (2018). Thoracic Disease Identification and Localization with Limited Supervision.

Murphy, K. P. (n.d.). *Machine Learning A Probabilistic Perspective.*

Pham, H. H., Le, T. T., Tran, D. Q., Ngo, D. T., & Nguyen, H. Q. (2019). Interpreting chest X-rays via CNNs that exploit.

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., . . . Ng, A. Y. (2017). CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical.

Smith, L. N. (2017). *Cyclical Learning Rates for Training Neural Networks.*

Szegedy, C., Ioffe, S., & Vanhoucke, V. (2016). Inception-v4, Inception-ResNet and.

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.

Tataru, C., Yi, D., Shenoyas, A., & Ma, A. (2017). Deep Learning for abnormality detection in Chest X-Ray images.

Udeshani, K., Meegama, R., & Fernando, T. (2011). Statistical Feature-based Neural Network Approach for the.

Wang, J., Yao, J., Zhang, Y., & Zhang, R. (2018). Collaborative Learning for Weakly Supervised Object Detection.

Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017). ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on.

Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2020). *Dive into Deep Learning.*

Διαμαντάρας, Κ., & Μπότσης, Δ. (2019). *Μηχανική Μάθηση.* Θεσσαλονίκη: Κλειδάριθμος.