



UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

PROJECT REPORT

Big Data Intelligence Using Distributed Deep Neural Networks

Felix Ongati

P52/6291/2017

MSc Computational Intelligence

Supervisor: Dr. Eng. Lawrence Muchemi

Submitted in partial fulfilment of the requirements for the award of the Degree of Master of Science in

Computational Intelligence at the University of Nairobi, Nairobi, Kenya.

Submitted on 23rd April 2019



DECLARATION

I hereby declare that this research project is my own work, and has to the best of my knowledge, not been submitted to any other institution of higher learning.

Student: Felix Ongati P52/6291/2017

Signature: Date:

This research project has been submitted as partial fulfilment of the requirements for the award of the Degree of Master of Science in Computational Intelligence at the University of Nairobi with my approval as the faculty supervisor.

Supervisor: Dr. Eng. Lawrence Muchemi

Signature: Date:



DEDICATION

To all you out there who equate data to solutions and to all who think literacy should be measured on the ability to unlearn and learn, this is to you.



ACKNOWLEDGEMENT

I am grateful to God for the good health and strength he grants me to learn and for providing priceless people who without their effort, collaboration and dedication, this project wouldn't have been a success.

I wish to personally thank the following people for their contributions to my inspiration and knowledge and other help in working through this project; my supervisor Dr. Eng. Lawrence Muchemi for his unwavering support and guidance even where I thought it was beyond my capacity, Prof. Peter Waiganjo Wagacha for his effort in continuously reviewing this work and aligning it to best health informatics practices, and all my lecturers at the School of Computing and Informatics who helped me in many ways and made my education journey at the University of Nairobi pleasant and unforgettable.

Lastly sincere gratitude to caffeine and sugar, my companions through many long nights of writing, coding and research.

Abstract

Large amount of data is often required to train and deploy useful machine learning models in industry. Smaller enterprises do not have the luxury of accessing enough data for machine learning, For privacy sensitive fields such as banking, insurance and healthcare, aggregating data to a data warehouse poses a challenge of data security and limited computational resources. These challenges are critical when developing machine learning algorithms in industry. Several attempts have been made to address the above challenges by using distributed learning techniques such as federated learning over disparate data stores in order to circumvent the need for centralised data aggregation.

This paper proposes an improved algorithm to securely train deep neural networks over several data sources in a distributed way, in order to eliminate the need to centrally aggregate the data and the need to share the data thus preserving privacy. The proposed method allows training of deep neural networks using data from multiple de-linked nodes in a distributed environment and to secure the representation shared during training. Only a representation of the trained models (network architecture and weights) are shared.

The algorithm was evaluated on existing healthcare patients data and the performance of this implementation was compared to that of a regular deep neural network trained on a single centralised architecture. This algorithm will pave a way for distributed training of neural networks on privacy sensitive applications where raw data may not be shared directly or centrally aggregating this data in a data warehouse is not feasible.


***Index Terms:* Big Data, Distributed Computing, Deep Learning**



Table of Contents

DECLARATION	1
DEDICATION	2
ACKNOWLEDGEMENT	3
Abstract	4
Introduction	8
1.1 Background	8
Sharing Representation Instead of Sharing Data	9
Decentralized Machine Learning	10
1.2 Problem Statement	10
1.3 Research Questions	10
1.4 Project Objectives	11
1.3.1 General Objective	11
1.3.2 Specific Objectives	12
1.5 Project Scope	12
1.6 Project Justification	12
2. Literature Review	13
2.1 Big Data Intelligence	13
2.2 Distributed Computing Systems	14
2.3 Deep Neural Networks	16
2.4 Predictive Analysis of Clinical Data	17
2.5 Blockchain Technology	18
2.6 Related Projects	20
2.6.1 SNIPS - Decentralized Machine Learning using MPC and Secret Sharing	20
2.6.2 Federated Learning at Google AI Lab (Google AI Blog, 2018)	22
2.6.3 OpenMined	22
2.6.3.1 OpenMined Platform Architecture	23
2.6.3.2 How it Works	23
2.6.5 Distributed Learning of Deep Neural Network Over Multiple Agents (Gupta and Raskar, 2018)	24
2.7 Related Research Work	24
2.7.1 The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets	24
2.7.2 Practical Secure Aggregation for Privacy Preserving Machine Learning	25
2.7.3 Communication-Efficient Learning of Deep Networks from Decentralized Data (McMahan et al., 2017)	25
2.8 Literature Review Summary	26

3. Research Methodology	27
3.1 Theoretical Studies	28
3.2 Simulation and Prototyping	28
3.3 Empirical Evaluation	28
3.4 Dataset	30
3.4.1 Data Preprocessing	32
4.0 Analysis and Design	33
4.1 Technical Requirements Analysis	33
4.1.1 Environment Software Setup	33
Numpy	33
Tensorflow & Tensorboard	33
Keras	33
CUDA & cuDNN	34
Docker for Distributed Computing Environment	34
4.2 Economic and Technical Feasibility Analysis	34
4.3 Prototype Design	34
Algorithm Design	34
4.4. Model Architecture	37
4.5 Initial Model Inputs	39
4.6. Implementation	39
4.7. Implementation Comparison with Blockchain	41
5.0 Results and Analysis	42
5.1 Proof of Correctness	42
5.2 Accuracy	42
5.2.1 Accuracy before smoothing	42
5.2.2 Accuracy after smoothing	43
5.3 Loss	43
5.3.1 Loss before smoothing	43
5.3.2 Loss after smoothing	44
5.4 Comparison with centralised models	44
5.5 Effect of increasing the number of layers on accuracy and loss	44
Accuracy	45
Loss	45
6.0 Conclusion and Future Work	46
References	47
Appendix 1: Project Plan and Management	49



1.2 Required Resources	49
Appendix 2: Sample Scripts	50
2.1 CNN Initialization	50
2.2 Model Instantiation, training and saving	51
2.3 Communication daemon script	52

1. Introduction

1.1 Background

Adoption of artificial intelligence techniques in business to enhance the quality of products, optimize planning, maximize output and improve customer satisfaction is at an all time high. Modern companies are now moving from mere predictive analytics to prescriptive analytics and possibilities beyond. We are now talking of artificial super intelligence (ASI), as path towards achieving technological singularity. Only time will settle the debate on the achievability of technological singularity as to surpass human intelligence, but right now we cannot overlook the possibilities that have been brought about by artificial intelligence, especially in enterprise operations. Data sensitive operations such as targeted marketing, that were traditionally based on purely human efforts have experienced great positive results attributed to adoption of AI techniques.

Large amount of data is often required to train and deploy useful machine learning models in industry. Labelled customer data cannot be shared for this purpose due to privacy issues and sensitivity of customer information that can land in unauthorised hands.

Successfully deployment of artificial intelligence solutions requires development of a very deliberate internal data sharing culture. Siloed data is one of the greatest killers of AI application in industry (Medium, 2018). Machine learning problems are known to be very data intensive, thus corporations wishing to explore machine learning have to first breakdown data silos, not only to increase the amount of data on which an AI model can be trained, but also to increase the diversity of the data. Though breakdown the data silos sounds like a brilliant idea, it becomes challenging in two ways;

1. When this data is individual customers' or patients' health data and breaking down the silos to share it will infringe on their privacy.
2. When resources to gather centrally and store data from these diverse sources are limited.

From challenge (2) above, the following pertinent question arises; how can health facilities, insurance companies and other SMEs with smaller or more specialized datasets benefit given that huge and diverse datasets are required to train and deploy usable modern machine learning systems? Thanks to recent progress in deep learning research, it is now possible for all corporations to benefit through sharing learnt representations other than sharing data.

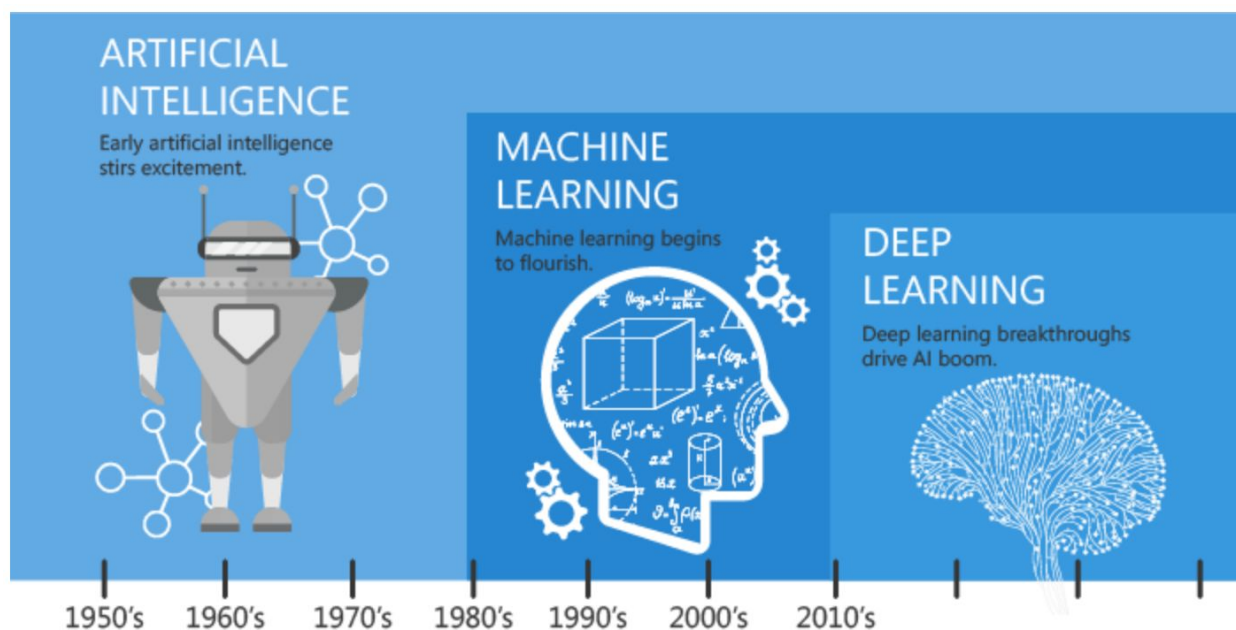


Figure 1.1: From general AI to Deep Learning (Hackernoon.com, 2019)

It is now possible to work with unlabelled and unstructured big data using deep neural networks and autoencoders. The success of machine learning methods in extracting patterns from structured data has led to its adoption in processing unstructured data such as images. In deep learning, the focus is on learning representation apart from learning patterns. Advancement in representational learning makes it possible for a trained model to share its knowledge with other deep learning systems.

Though there has been a surge in application of artificial intelligence in industry, its adoption in electronic medical records has been slow due to a myriad of problems, mainly privacy issues, the unstructured nature of most of the data and complicated regulatory requirements. According to AI Med (AI Med, 2019), about 80% of data in EHRs/EMRs is unstructured and housed in various silos across disparate systems. Sharing this data or aggregating to a data warehouse for centralised application of machine learning does not address any of the above problem. Unless a solution is found, the medical field will lag behind in this AI boom.

Sharing Representation Instead of Sharing Data

Since it is possible to share learned representation between different deep learning systems instead of sharing raw data, data security and privacy will no longer be a major issue in AI adoption. A description of the data, description of training model and the learned weights is what will need to be shared with other learning systems that would need to run the same on their data. Deep learning provides opportunities for continuous learning and improvement for AI systems as they get exposed to different data sets. As Philippe Beaudoin (Medium, 2018) correctly points

out; advances in representation learning will to a great extent enable sharing of expertise and in the future, the greatest killer of AI was siloed expertise.

Decentralized Machine Learning

There are already solutions built around decentralized machine learning, also known as federated learning, such as OpenMined (Openmined.org, 2018), DML (Decentralizedml.com, 2018) and BigAI (BigAI, 2018) that can be deployed on consumer applications on mobile devices for tasks such as learning the voice of the device owner so as to work with voice commands. Apart from being tailored to work with a single data source for both training and testing, the major challenge with these solutions is that they always assume that users have pre-labeled data on their devices. Despite this challenge, the techniques learned here can be adopted and curated to be applied in a wide range of fields including electronic health records. Instead of sharing data for machine learning, what if we share knowledge instead?


1.2 Problem Statement

For corporations to reap the full potential benefits of using artificial intelligence, enough and diverse data must be available to build models and extract knowledge. As mentioned earlier, one of the greatest killers of AI in industry is siloed data. Breaking the data silos by sharing data and storing it centrally face storage space, computational resources limitation and privacy concerns. It is also difficult for corporations with smaller or more specialized or sensitive datasets that cannot be shared to benefit from utilizing machine learning in their operations given that huge and diverse datasets are required to train modern machine learning systems. Thus there is a need to have a way in which smaller companies/entities in the same domain can apply machine learning techniques on their data collaboratively without having to share data while preserving data security and integrity. In summary, lack of a secure model for training deep neural networks on distributed/desperate and unstructured data environments poses a challenge to full scale application of artificial intelligence techniques in industry.

1.3 Research Questions

Based on the problem statement as outlined above, the following questions need to be addressed towards solving the problem;

1. What is required for securely training a distributed deep learning model in a distributed unstructured data environment?
2. How can we design a distributed deep learning neural network model over distributed or disconnected enterprise data stores?
3. How can we prototype the above model in an environment with de-linked data stores?

- 
4. How can we evaluate the performance of a distributed deep learning model?

1.4 Project Objectives

1.3.1 General Objective

To develop and test a model for training distributed deep neural networks on a distributed big data environment. The model was trained on a radiology database consisting of chest x-ray images.

1.3.2 Specific Objectives

- a. To develop a deep learning algorithm for securely training a distributed deep neural network over distributed unstructured data sources.
- b. To design a distributed deep neural network model over distributed medical imagery databases?
- c. To prototype the distributed deep neural network model in a distributed unstructured data environment
- d. To evaluate the performance of the distributed deep learning model.

1.5 Project Scope

This research was limited to developing a distributed deep neural network model on distributed big data environment using chest x-ray images datasets. Security implementation was based on RSA encryption algorithm, evaluation of different security implementation algorithms was not within the scope of this research project.

1.6 Project Justification

This research project provides a solution to the problem of training deep learning models in a distributed environment without having to centrally aggregate the data in a data warehouse or having to share data in a privacy sensitive environments. This paves the way for collaborative learning in industry by small enterprises and extending the application of machine learning techniques to electronic health records, thus leveling the competitive field for every player who wishes to apply artificial intelligence techniques in industry.

This research project is also complex enough to warrant a computer science research approach to solution since it touches on important fields in computing, including Big Data, Distributed Computing, Synchronisation, Optimisation and Security.

2. Literature Review

2.1 Big Data Intelligence

Big data is a term that references sets of data whose size, type or structure is beyond the ability of traditional relational databases to capture, manage, and process with low-latency (Ibm.com, 2018). Big data does not necessarily mean a lot of data, though size is a big indicator in determining whether a set of data qualifies to be called Big Data, its nature of being highly unstructured carries more weight in the definition than size. The most common characteristics of big data are;

- I. Volume - big data is characterized by enormous volume of data being generated by machines, IoT devices or human interactions with systems such as social media.
- II. Variety - data is from multiple sources, which makes the data be in several types, either structured or unstructured.
- III. Velocity - refers to the speed at which data accumulates from different sources. The flow of data is normally massive and continuous.
- IV. Veracity - refers to the biases, noise and abnormality in the data, though not obvious, depending on the source of that particular data

Data intelligence is the analysis of various forms of data in such a way as to inform action and optimize decision making. Thus big data intelligence refers to application of data analytics methods and tools to analyze big data in order to enable companies to make better decisions and to inform action.

In the recent past, most big companies have been aggregating data from multiple sources and joining it together to exploit network effects to become a one stop shop for high quality data. In this model, if you're the aggregator you win, taking typically 60–80% of the revenue and the vast majority of the profits. If your data is aggregated you lose, but not as much as if you keep your data to yourself. Pretty much every industry now has their set of dominant aggregators, and so it's exceedingly hard to get in here unless you happen to be an industry-leading aggregator already and just woke up this morning with that realization (Bernes, 2017).



Figure 2.1 Big data business model

Data contained in EHRs and EMRs has all traits of big data. Doctors and medical imagery makes this data largely unstructured and disintegrated. When attempting to get useful insights from this data, a lot of effort would go towards cleaning the data and pre-processing.

2.2 Distributed Computing Systems

A distributed computing is the field in computer science that studies the design and behavior of systems that involve many loosely-coupled components (Cpsc.yale.edu, 2018). They consist of multiple software components that are on multiple computers, but run as a single system. The computers that are in a distributed system can be physically close together and connected by a local network, or they can be geographically distant and connected by a wide area network. A distributed system can consist of any number of possible configurations, such as mainframes, personal computers, workstations, minicomputers, and so on. The goal of distributed computing is to make such a network work as a single computer.

Several factors motivate enterprises to implement distributed systems as opposed to centralised ones. The two main factors are for scalability and redundancy. Scalability ensures that the systems can be expanded easily by adding more nodes as needed. Redundancy improves service availability and ensures security.

The figure below shows an example of a distributed computing systems;

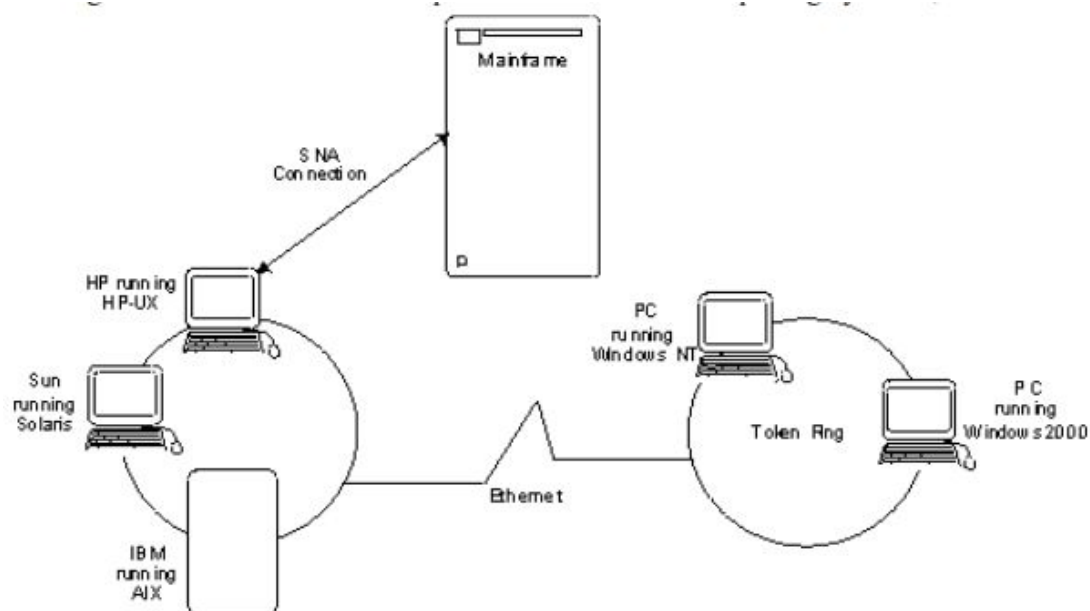


Figure 2.2 Distributed computing system

This research project implemented the deep learning model in a both decentralised and distributed way. The multiple data stores were decentralized and delinked whereas the learning process was distributed across different training nodes in the network. Each node has its own copy of the trained model depending on local stopping parameters. Thus a study of inner workings of distributed computing systems was necessary prior to the start of the project.

2.3 Deep Neural Networks

Deep learning is a machine learning method based on learning data representations and structure as opposed to learning task specific algorithms. Deep learning is also referred to as deep structured learning or hierarchical learning. Deep neural network is a deep learning architecture that uses artificial neural networks to learn representations. The diagram below a conceptual model of a neural network;

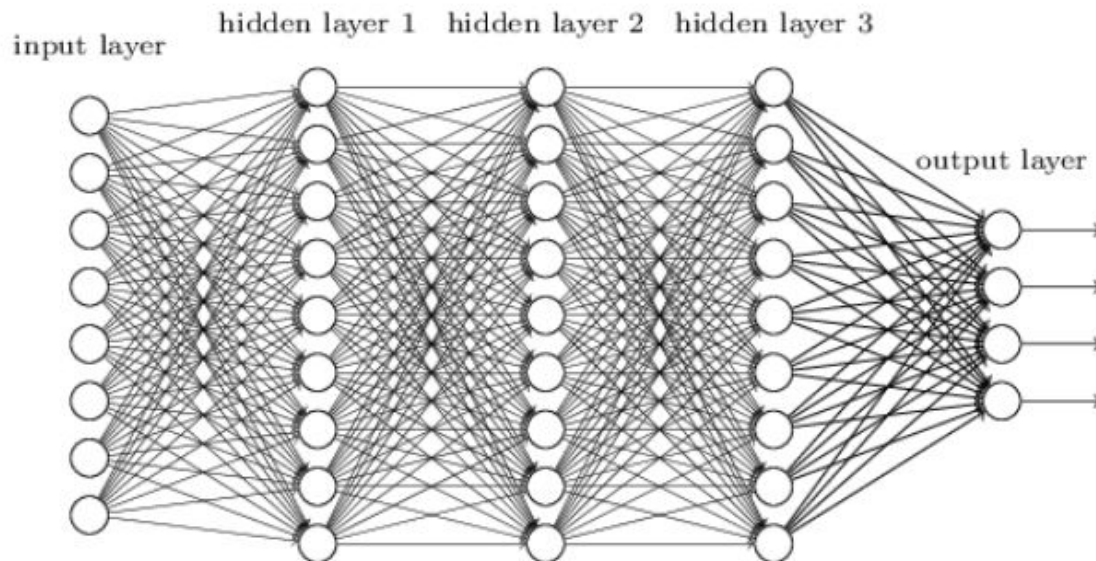


Figure 2.3 DNN model


A method known as gradient descent is used to train the network. Gradient descent requires a large amount of data in order to determine the gradient of a predefined cost function and to converge to a global minimum. Once the minimum is found, the network is considered trained and can be used for inference. Usually, no further training occurs at this stage. Multiple versions of gradient descent have been invented over time, with some of the more prominent being the Stochastic Gradient Descent (SGD) and Adadelta. The advantages of gradient descent include fast and simple implementation and relatively fast convergence as compared to algorithms such as Genetic Algorithms (GA). Genetic algorithm is more susceptible to premature convergence. The choice of deep learning as the machine learning technique to be used in this project was largely informed by the nature of data, which consists of labelled x-ray images from an EMR system.

2.4 Predictive Analysis of Clinical Data

Predictive data mining is a field of data mining that automatically creates a classification model from a given set of examples. Once the model is built, it can be used to predict the classes of other examples automatically. The outcome is generally because of certain probability or on the basis of detection theory. Many models and classifiers exist for predictive data mining namely k-Nearest Neighbours (k-NN), Naïve Bayes, Logistic Regression, and Majority Classifiers. The predictive data mining techniques has varied applications ranging from archaeology, medical sciences and bioinformatics. The predictive modelling is mainly preferred for biomedical research to target best diagnosis and appropriate treatment, developed detailed patients' and disease profile and also to diagnose and prevent diseases appropriately. The predictive data mining technique allows the physician/surgeon to predict the disease through symptoms quickly and accurately, predict the survivability rate of the patients and find the intensity of the diseases more precisely. Thus predictive data mining helps the medical practitioners to decide about the type of treatment based on the predictions (Elsevier Connect, 2016).

For these reasons, clinicians complement guidelines and protocols with their knowledge base and anecdotal experience. However, there is considerable variation in clinicians' prior experience, and in addition, clinician recall is often biased, with recent patients and patients with adverse outcomes being recalled most readily. Here are some benefits that can be attributed to clinical predictive analysis;

- Predictive analytics increase the accuracy of diagnoses.
- Predictive analytics will help preventive medicine and public health.
- Predictive analytics provides physicians with answers they are seeking for individual patients.
- Predictive analytics can provide employers and hospitals with predictions concerning insurance product costs.
- Predictive analytics allow researchers to develop prediction models that do not require thousands of cases and that can become more accurate over time.
- Pharmaceutical companies can use predictive analytics to best meet the needs of the public for medications.
- Patients have the potential benefit of better outcomes due to predictive analytics.



This approach represents a promising avenue toward reducing the current gap between research and practice across healthcare, developing data-driven clinical decision support based on real-world populations, and serving as a component of embedded clinical artificial intelligences that “learn” over time.

The use of predictive models for informing healthcare treatment algorithms accentuates the tension that exists between the art and science of treating common health disorders, between the knowledge of experienced clinicians and predictive recommendations derived from data. This old controversy is best characterized by Paul Meehl, who noted nearly half a century ago “When you are pushing [scores of] investigations [140 in 1991], predicting everything from the outcomes of football games to the diagnosis of liver disease and when you can hardly come up with a half dozen studies showing even a weak tendency in favor of the clinician, it is time to draw a practical conclusion” (p.372-373).

However, many current decision support systems in healthcare rely on experts’ or standards-based models, rather than models that adapt population-based guidelines to individual patient characteristics by utilizing existing EHR patient data.

Applying advanced learning techniques such as deep learning to clinical predictive analytics will improve the predictive models by extending them to cover unstructured data such as radiographic images. This research used chest x-ray images to build a model that would predict the probability of a patient having pneumonia based on their chest x-ray images.

2.5 Blockchain Technology

A blockchain is an open distributed, decentralized, public ledger that securely stores transaction records between parties by using public key cryptography on a peer-to-peer network (Harvard Business Review, 2019).

Transactions are recorded as a block of data which is hashed and distributed across multiple nodes in the network. Each block is linked to a preceding block in their order of generation,

forming a logical chain.

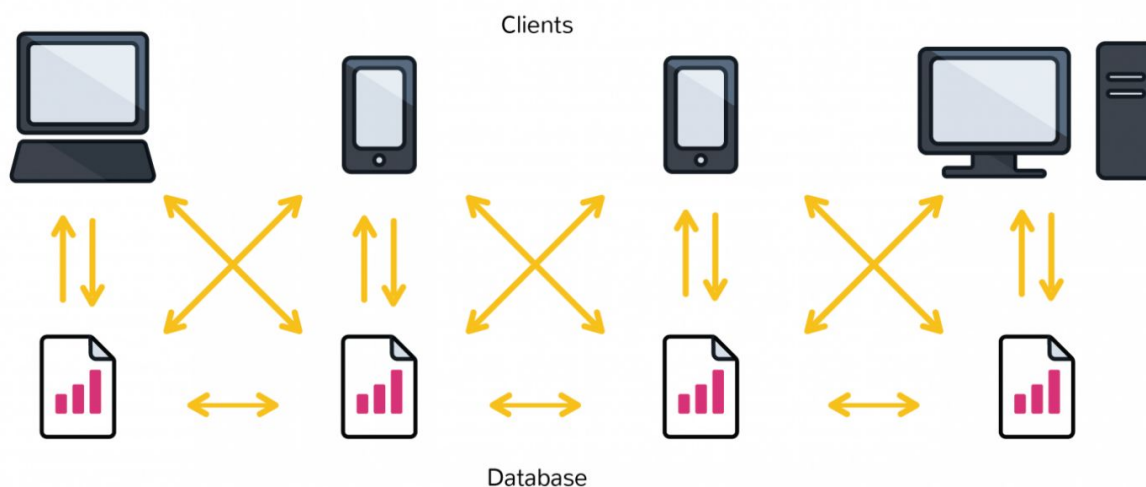


Figure 2.4: High level blockchain decentralization concept (CoinDesk, 2019)

Each block is generated by application of a complex mathematical computation that is resource intensive. Coupling this generation process with public key cryptography and the fact that the same block is replicated to many nodes in the network makes blockchain technology inherently secure.

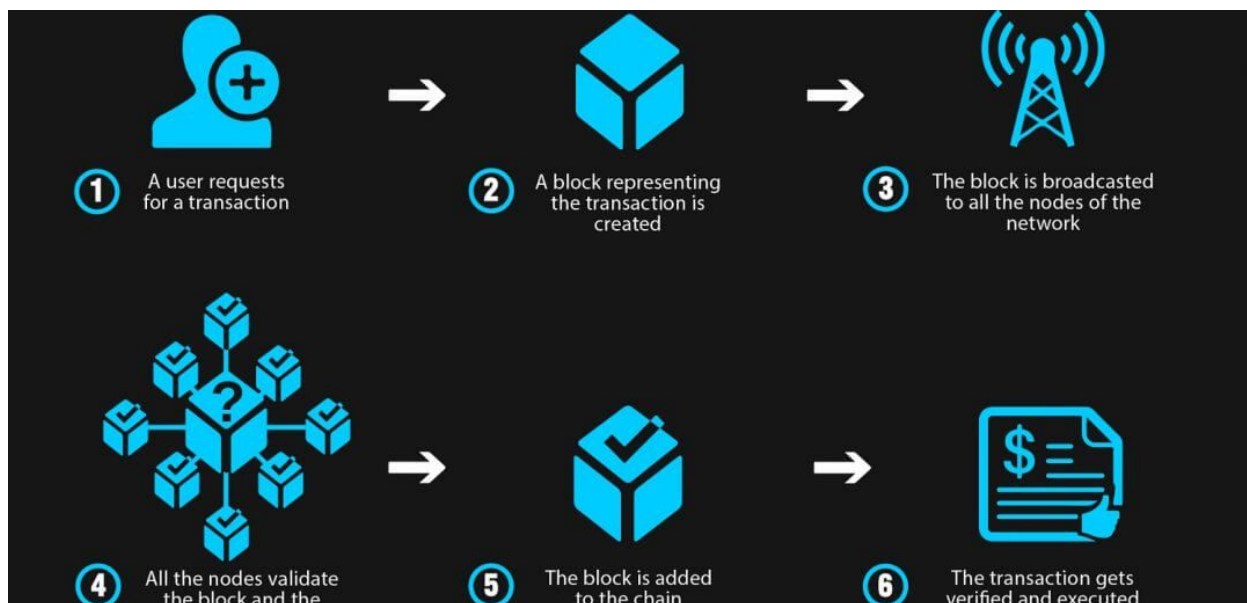


Figure 2.5: Simplified chain generation process (Anwar, 2019)

Blockchain is not completely anonymous but confidential, transactions are publicly recorded on the blockchain without complete user data. A user's public key is a shortened version of the private key that is created through a complicated mathematical algorithm. Due to the complexity of generating the keys, it is practically impossible to reverse the process and generate a private key from a public key. This is how blockchain technology achieves its confidentiality.

Implementation of the algorithm used in this research project borrows a lot from blockchain technology with few modifications but totally different objectives. They both run on distributed network, use private key cryptography and every node on the network comes to the same conclusion, each updating their records independently. Blockchain technology requires authorization and authentication to establish trust whereas in our case, the keys are used for encryption only to conceal weight transfer between nodes and not for authentication/authorization.

2.6 Related Projects


2.6.1 SNIPS - Decentralized Machine Learning using MPC and Secret Sharing

Snips (Snips, 2018) is a platform that helps people build private voice assistant applications for IoT. Snips provides NLP as a service without having to share user data. Privacy problem is addressed by a combination of 3 factors;

- I. Embedded processing of the voice query - voice processing is done directly inside the device the user is talking to. No data is sent outside the user's device to the cloud unlike traditional voice recognition devices. This 100% on-device approach has a number of advantages, from enabling offline use cases, to having no network latency and being resilient to cloud outages, mass surveillance and data breaches.
- II. Decentralized data generation - data for training examples is automatically generated from a handful given initially
- III. Decentralized machine learning - The voice app models are trained on the end user data without forcing the data to leave the devices they are sitting on. The goal of Decentralized Machine Learning (also called Federated Learning) is to train a neural network by updating the gradient locally on the device of the user, before aggregating it securely via a network of nodes.

Challenges;

- Automatic generation of data may not be an exact representation of data from the user given that different users may have different ways of saying the same thing.
- Since these models are built for a specific task of voice commands interpretation on a user's device, it will pose a challenge scaling them to be used in big data contexts with high demand for space resources.

- 
- At any given instance, a model on a device has a snapshot of its local environment, and blind to the global environment. Though this is not a requirement for a voice app, it is important for models in a distributed big data environment.

There is a key challenge that is often overlooked: most decentralised machine learning solutions, like OpenMined, DML or BigAI, assume that users have pre-labelled data on their devices.

2.6.2 Federated Learning at Google AI Lab (Google AI Blog, 2018)

Federated learning is an approach to training models from user interaction with mobile devices; models are trained on the devices. Federating learning decouples machine learning from data storage by enabling learning a shared prediction model with the training data on the device without sharing the data (McMahan and Ramage, 2017). In this approach, there is a shared model that resides in the cloud.

A mobile device gets the current up-to-date shared model from the cloud, runs it on the local data on the phone and stores the improvements as a small focused update. These changes to the model are then sent to the cloud, averaged with updates from other devices and applied to the central main model in the cloud. No training data leaves the users' devices.

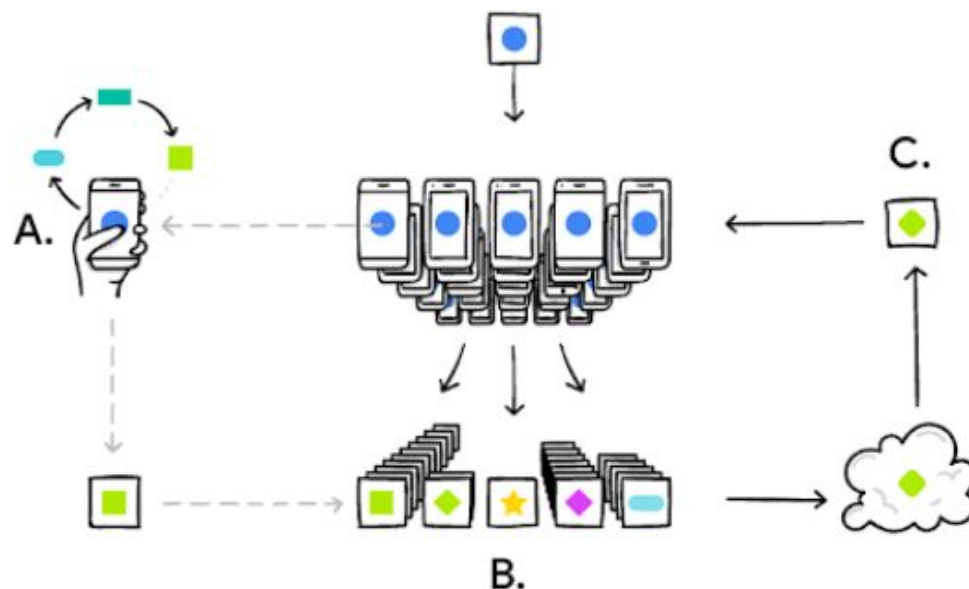


Figure 2.6 Federated learning architecture

A mobile device (A) localizes the model in the context of a user's interaction with the device. Users' changes are summed up (B) to form an aggregate change (C) which is then applied to the shared model and procedure repeated whenever new data is available. (Google AI Blog, 2018).

Federated learning proposes a mechanism suitable for training centralized models in an unreliable network connection environment where sharing data would be expensive in addition to privacy concerns. This work borrows a lot from federated learning techniques, albeit on training a decentralised model.

2.6.3 OpenMined

OpenMined is an open-source community project focused on researching, developing, and promoting tools for secure, privacy-preserving, value-aligned artificial intelligence (Openmined.org, 2018). The project has built a platform that combines the principles of federated learning with cutting edge techniques such as homomorphic encryption and blockchain

smart contracts to enable a collaborative model to implement deep learning application in a completely decentralized way (Rodriguez, 2018). The OpenMined platform allows data scientists to upload pre-defined models and private data and have the model trained by anonymous decentralized nodes (miners).

2.6.3.1 OpenMined Platform Architecture

The figure diagrammatically represents OpenMined approach towards decentralized collaborative learning;

Open Mined Architecture

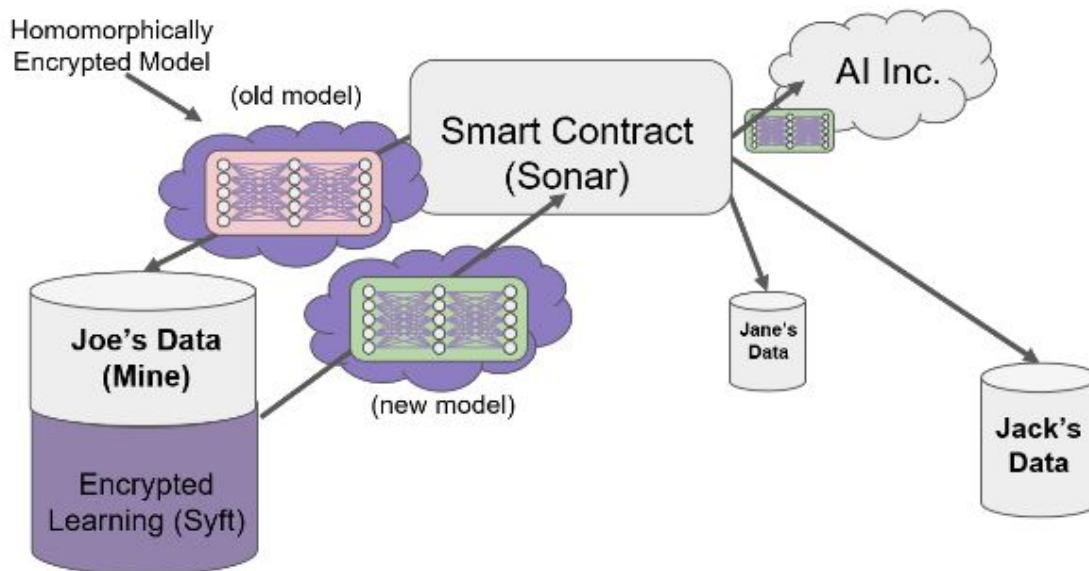


Figure 2.7 OpenMined architecture (Towards Data Science, 2018)

2.6.3.2 How it Works

1. A data scientist creates a desired model using an external framework such as Keras or Tensorflow and specifies the type of data required and the amount they are willing to pay for the model to be trained.
2. Upon submission the model is encrypted and uploaded to OpenGrid - a peer-to-peer network of data owners and data scientists who can collectively train AI models using Syft.
3. Members of the OpenGrid network anonymously download the encrypted model should they have the correct data required by the model. They train the model locally on their devices.
4. With each party unknown to the other, the miner uploads a new version of the trained model based on their local training
5. The model that meets a success criteria is the one that is picked by the data scientist.

Though OpenMined represent one of the best enterprise implementation of decentralized deep learning, this approach poses the following challenges;

1. The model is not trained on user's data, but on someone else's data located somewhere on the OpenGrid networks meaning the user has no control of the data.
2. This approach can only be used on pre-labelled data and can not be deployed in an environment with unstructured big data.
3. It requires the user/data scientist to have predefined models.
4. Scope of training of a model is limited to only a localised data store in the sense that the trained model does not have a view of the whole data on the network.

2.6.5 Distributed Learning of Deep Neural Network Over Multiple Agents (Gupta and Raskar, 2018)

This work by Gupta and Raskar explores an algorithm that allows for distributed deep learning over multiple agents. The algorithm was evaluated on existing mixed NIST (MNIST) datasets and it was shown that the obtained performance was similar to that of a regular neural network trained on a centralised data source. Gupta and Rasker raised security concerns on their algorithm evaluation which we seek to address here by running an improved algorithm on privacy sensitive electronic medical records patients' data.

2.7 Related Research Work

2.7.1 The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets

The communities of researchers that need to access and analyze this data (often using sophisticated and computationally expensive techniques) are often large and are almost always geographically distributed, as are the computing and storage resources that these communities rely upon to store and analyze their data [17]. This combination of large dataset size, geographic distribution of users and resources, and computationally intensive analysis results in complex and stringent performance demands that are not satisfied by any existing data management infrastructure. A large scientific collaboration may generate many queries, each involving access to or supercomputer-class computations on gigabytes or terabytes of data. Efficient and reliable execution of these queries may require careful management of terabyte caches, gigabit/s data transfer over wide area networks, coscheduling of data transfers and supercomputer computation, accurate performance estimations to guide the selection of dataset replicas, and other advanced techniques that collectively maximize use of scarce storage, networking, and computing resources. (Chervenak et al., 2000)

2.7.2 Practical Secure Aggregation for Privacy Preserving Machine Learning

This research paper by Bonawitz et al (Bonawitz et al., 2017) outlines an approach to advancing privacy preserving machine learning by leveraging secure multiparty computation (MPC) to compute sums of model parameter updates from individual users' devices in a secure manner. This approach attempts to address the issue of privacy/security in federated learning on mobile devices where communication is expensive and dropouts are common. This research developed a protocol for securely computing sums of vectors, which had a constant number of rounds, with low communication overhead, robustness to failures, and which required only one server with limited trust. The role of the server was to route messages between the other parties and compute the final result. The figure below depicts how secure aggregation is achieved;

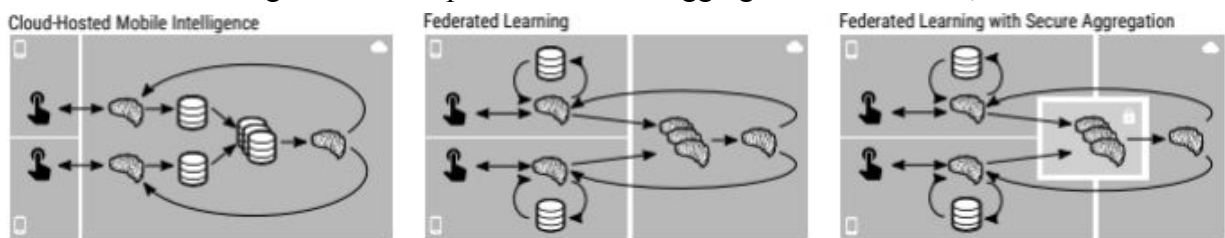


Figure 2.8 Secure aggregation

Though this research work focuses on securing and optimizing communication on federated learning on mobile devices, some aspects of this secure aggregation technique can be adopted and improved to secure weights sharing in a distributed big data environment.

2.7.3 Communication-Efficient Learning of Deep Networks from Decentralized Data (McMahan et al., 2017)

Modern mobile devices generate a lot of data suitable for learning models to improve user experience. However due to sensitivity and privacy issues, it is impossible to log this wealth of data centrally to a data center and train machine learning models on them using conventional approaches. In this research, McMahan et al advocate for an alternative approach that leaves the training data distributed on the mobile phone devices, and learns a shared model by aggregating locally computed updates, an approach known as Federated Learning.

A practical method for the federated learning of deep networks based on iterative model averaging is presented. An extensive empirical evaluation, considering five different model architectures and four datasets is also conducted.

Federated learning allows users to collectively reap the benefits of shared models trained from this rich data, without the need to centrally store it thus eliminating the need to share users' data. In their implementation of federated learning, this research uses a client-server approach. Each client has a local training dataset which is never uploaded to the server. Instead, each client computes an update to the current global model maintained by the server, and only this update is communicated, thus decoupling model training from the need for direct access to the raw

training data. An algorithm known as Federated Averaging, which combines local stochastic gradient descent (SGD) on each client with a server that performs model averaging was developed to implement federated learning.

2.8 Literature Review Summary

There has been a lot of effort both from academia and industry in an attempt to address the problem of implementing distributed machine learning on distributed systems especially on small pre-labelled user data sets. All methods that have been used presume pre labelled data. Thus there's a need to develop a generic model that can be used in big data settings where data is not necessarily labelled. Though federated learning on mobile devices has been proven to work well, the tasks that has been tested on such as image classification, for example predicting which photos are most likely to be viewed multiple times in the future, or shared; and language models, which can be used to improve voice recognition and text entry on touch-screen keyboards by improving decoding, next-word-prediction, and even predicting whole replies requires user data to be pre-labeled. While federated learning as implemented by McMahan et al offers many practical privacy benefits, there's a need to provide stronger privacy guarantees via application of differential privacy and/or secure multi-party computation.

Conceptual Framework

The diagram below depicts the current status of things against what this research project aims at achieving;

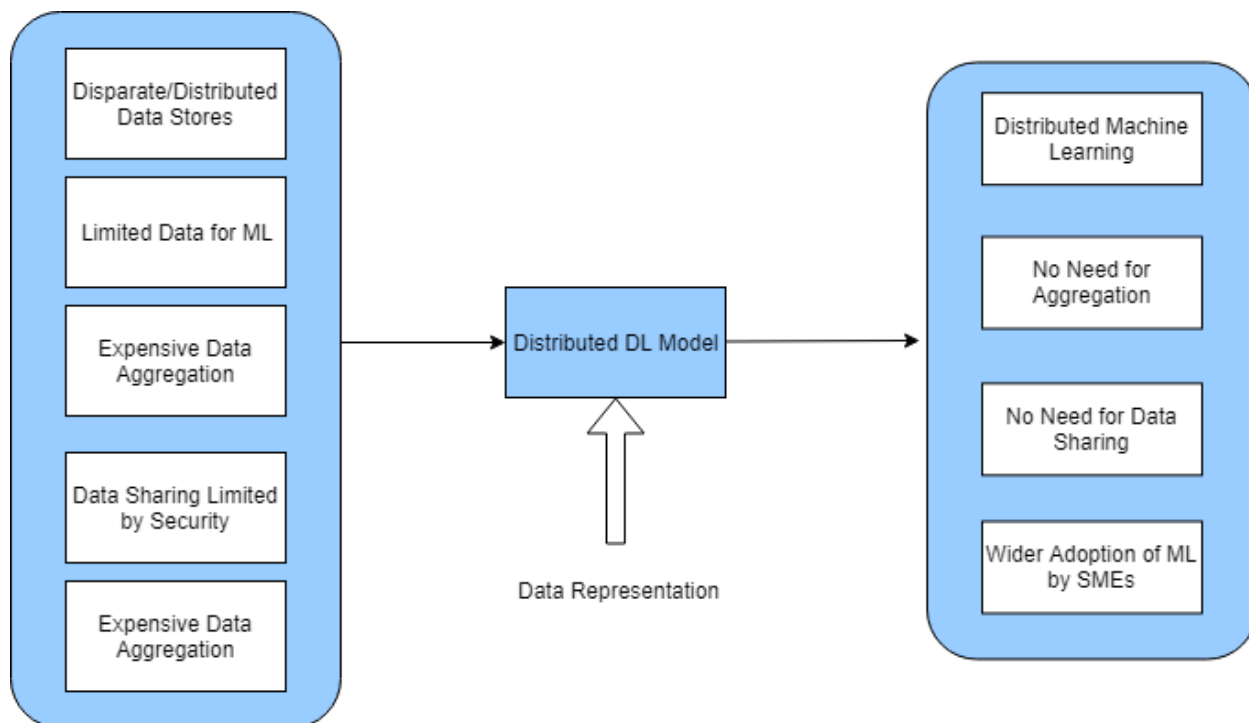


Figure 2.9: Conceptual Framework

3. Research Methodology

To answer each of the questions, this research involved theoretical studies of existing literature and implemented projects, experimental studies, simulation, prototyping and empirical evaluation. The diagram below gives a high level summary of the tasks that were accomplished in order to realize the study objectives;

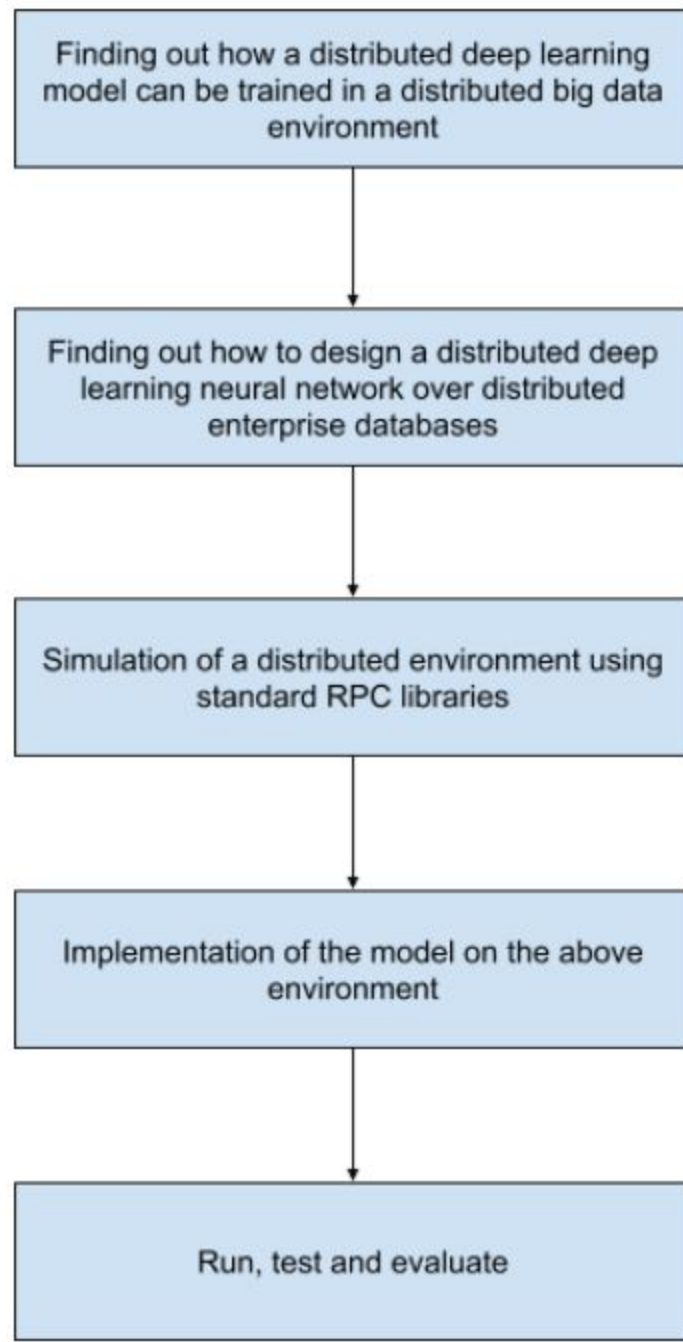


Figure 3.1 Project activity flow

To accomplish the above tasks, below is how each of the approaches was used;

3.1 Theoretical Studies

To find out an optimal algorithm for securely training a distributed deep learning model in a distributed big data environment, there was a review of existing literature. This involved reviewing of academic papers and white papers in the area of distributed machine learning on big data and centralised machine learning on big data environments. A study of other distributed machine learning open source and commercial projects was also conducted, including a study of federated learning approach widely used on training centralised models in mobile devices such as smartphones and tablets. The output of this study was an optimal algorithm that can be used to securely train a distributed deep learning neural network model in a distributed big data environment. Arriving at an optimal algorithm required coming up with a hybrid algorithm that borrows the best techniques from each of the identified approaches.

A study of the design of distributed machine learning open source projects was useful in designing a simulation of a distributed environment.

3.2 Simulation and Prototyping


This project prototypes a distributed deep learning environment using python parallel programming and secure shell protocol (SSH) for file transfer and communication; sending model architecture and weights. Convolution neural network was used for automatic feature extraction and representational learning on the data. The environment was setup on a personal computer with Nvidia GTX 1050 Graphics Processing Unit, running on Ubuntu 18.04 operating system with Python 3.6 with installed with Numpy, Tensorflow and Keras deep learning libraries.

3.3 Empirical Evaluation

The number of nodes in the distributed environment was increased periodically as performance was being monitored in terms of time taken to converge and accuracy so as to generate concrete recommendations for future work.

The performance of this model of training a distributed deep neural network on big data was compared against implementing the same model on a centralised deep learning environment where all the data is centrally aggregated. The comparison was in terms of accuracy of the clustering and time taken for the models to converge.

To ascertain performance against other existing implementations, this method was compared against the modern state-of-the-art methods including large-batch global SGD and federated averaging approaches from theoretical studies above.



K-fold validation was used to evaluate the accuracy of the model by itself. This was done by systematically splitting up the available data into k-folds, fitting the model on k-1 folds, evaluating it on the held out fold, and repeating this process for each fold.

3.4 Dataset

The data that was used to train the model consisted of x-ray images obtained from 28780 patients. There were two sets of data; xray images from normal patients and those from patients with pneumonia.

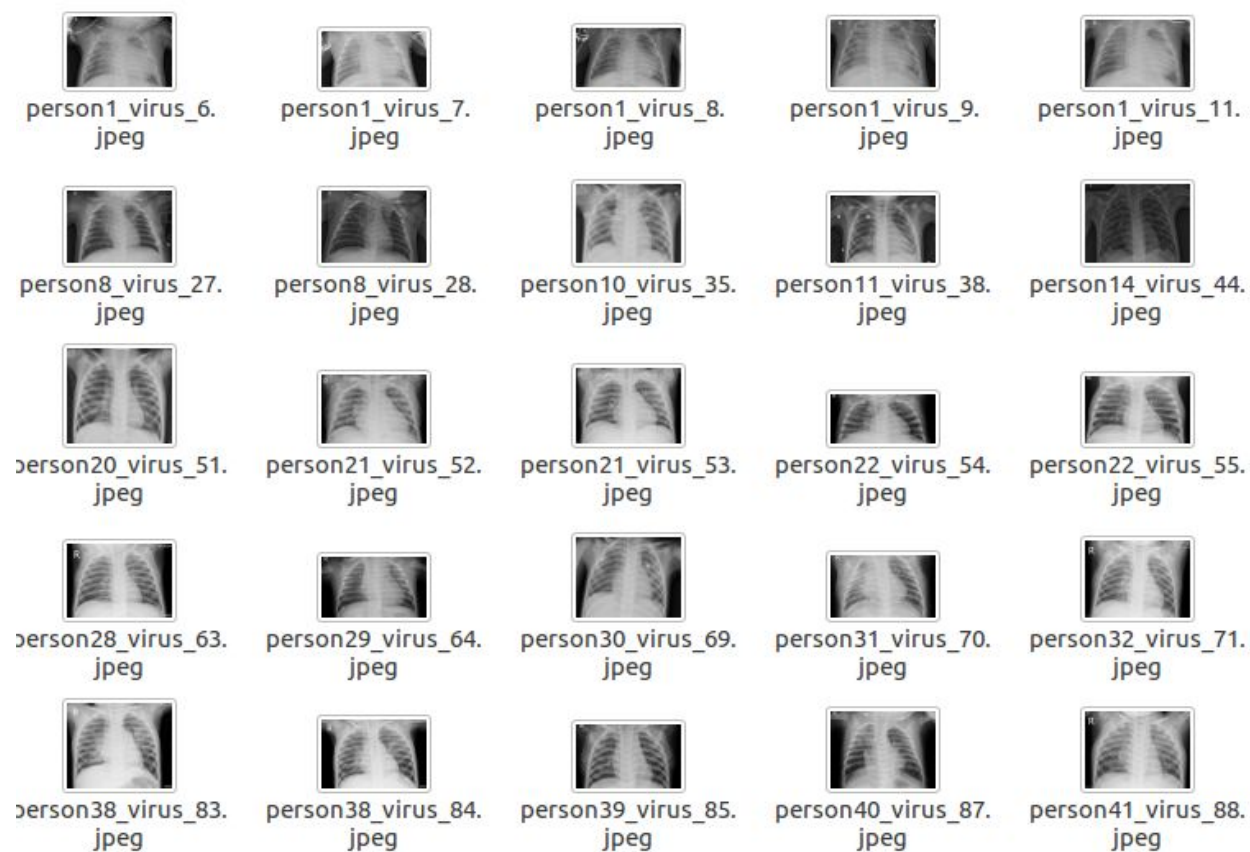


Figure 3.2: Dataset

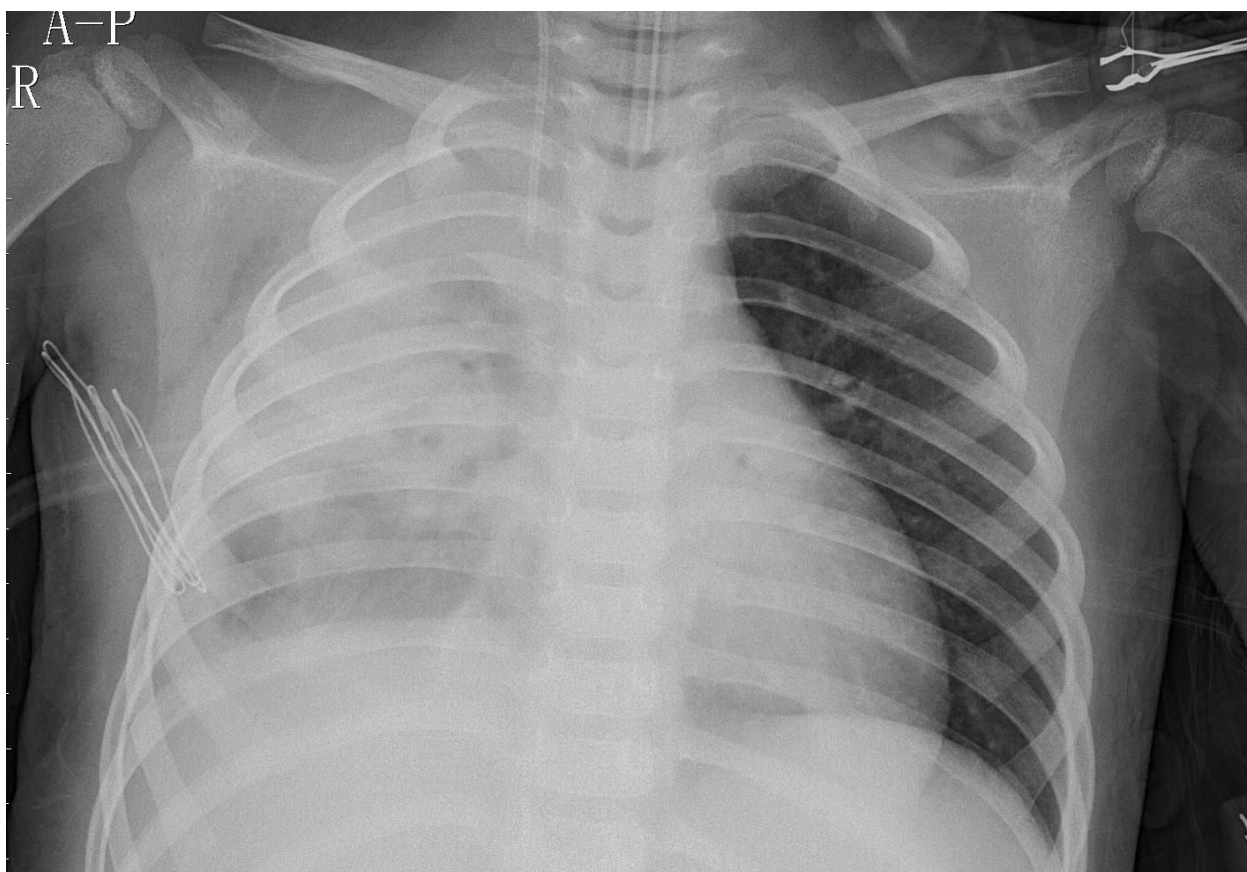


Figure 3.3: Xray image from patient with pneumonia

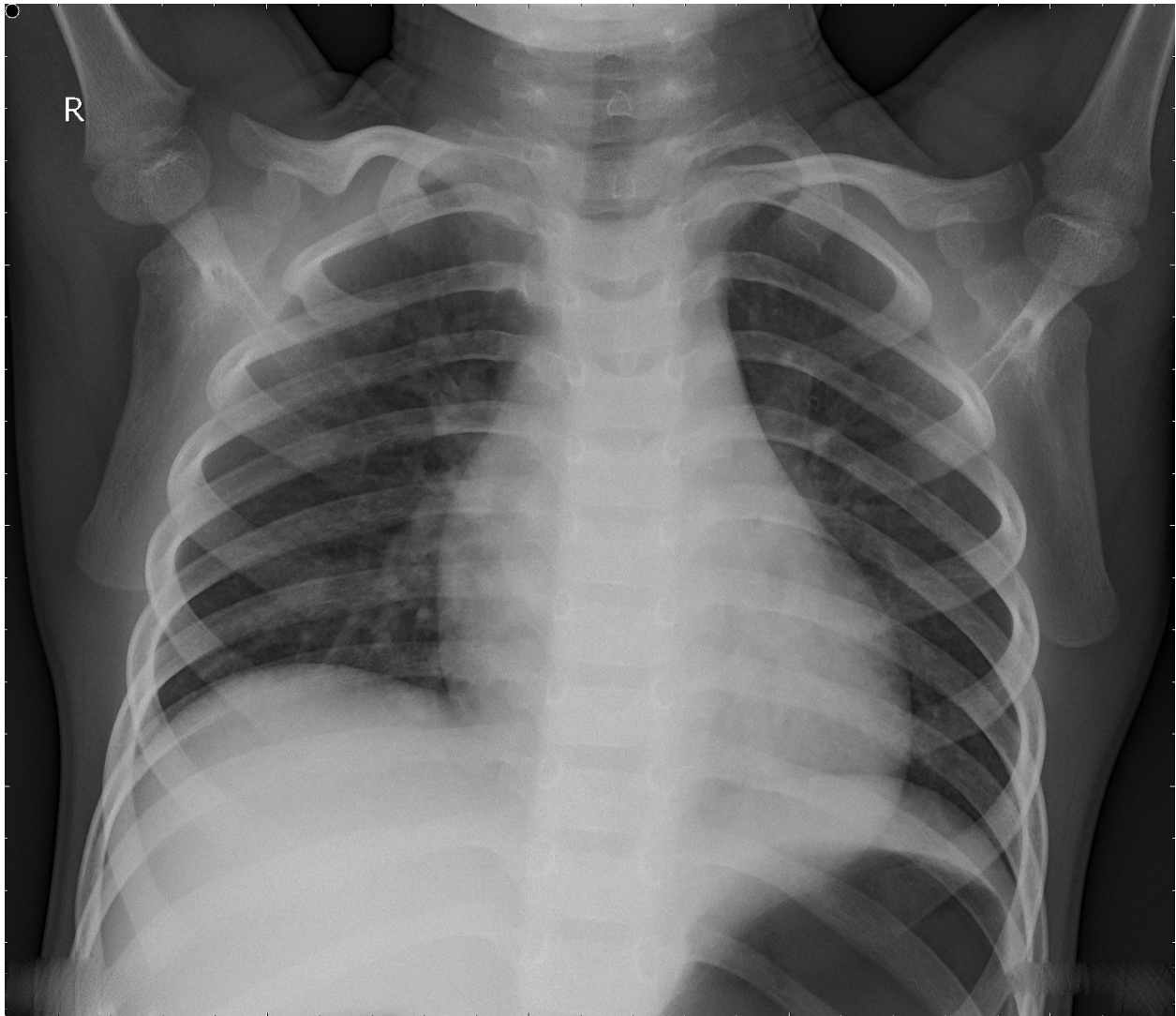


Figure 3.4 Chest Xray image from a normal person.

3.4.1 Data Preprocessing

The binary images was reformatted and spatially normalized to fit in a 20×20 bounding box. Anti-aliasing techniques was used to convert black and white (bilevel) images to grayscale images. Finally the images were placed in a 28×28 grid, by computing the center of mass of the pixels and shifting and superimposing images in the center of a 28×28 image.

4.0 Analysis and Design

4.1 Technical Requirements Analysis

4.1.1 Environment Software Setup

The environment was setup on a personal computer with Nvidia GTX 1050 Graphics Processing Unit, running on Ubuntu 16.04 operating system and Python 2.7 installed with the following packages;

Numpy

NumPy is a library for Python programming language that adds support for large multi-dimensional arrays and matrices, with a large collection of mathematical functions to operate on these arrays. Since the representation of the images in the data needed a matrix of size 256 x 256 with a depth of 3 for each image, NumPy came in handy in doing operations on array data structures of such size.

Tensorflow & Tensorboard

Tensorflow is an open source library for developing machine learning algorithms. It has a comprehensive, flexible collection of tools, libraries and community resources that lets researchers develop state-of-the-art machine learning models with ease and for developers to easily build and deploy machine learning powered applications. Tensorflow was used to abstract low level implementation of the convolutional neural network since this research project focuses on building a model for distributed machine learning and not the nitty gritty of low level implementation. Tensorboard is a visualization tool that runs on tensorflow and helps in visualization of the results during training.

Keras

Keras is a high-level neural networks API, written in Python and runs on top of Tensorflow. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier.

CUDA & cuDNN

CUDA is a parallel computing platform and programming model developed by NVIDIA (NVIDIA Developer, 2019) for general computing on its own GPUs. CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation. NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. CUDA and cuDNN were used to implement parallel computing so as to speedup training.

Docker for Distributed Computing Environment

Implementation of the distributed computing environment requires a docker image running multiple containers as semi-autonomous computing agents. Docker is a container technology for Linux that performs operating-system-level virtualization.

4.2 Economic and Technical Feasibility Analysis

All the technical software requirements listed above are free and open source thus there is minimal cost implication on the prototype development. Being open source, the technologies have a wider community for technical support

4.3 Prototype Design


Algorithm Design

This algorithm implements a technique that was used to train deep neural networks over multiple data sources in a secure electronic medical records environment while mitigating the need to share raw data directly. The algorithm implementation used three categories of agents as below;

- I. Alice → deep neural network that runs at the data source nodes.
- II. Bob → acts as a watchdog, monitoring file changes (model improvements) which are encrypted and send to Charity. Bob also gets initial models from Charity when Alice wants to train on new data.
- III. Charity → remote deep neural network that coordinates sharing of weights among the training nodes and stores up-to-date model architecture.

START

- I. Charity initializes a network architecture with random weights and other parameters.
- II. Charity compiles the model and trains with its local data for 1 epoch, writing the best output model to a file.
- III. Charity encrypts the model file using her own private key.
- IV. When Alice_k wants to train;

- 
- a. Bob_k fetches the current model from Charity via secure shell (SSH) protocol
 - b. Bob_k decrypt the model file using Charity's public key and makes it available for Alice_k
 - V. Alice_k reads the model, compiles it trains (a series of forward and back propagation) the model using its local data.
 - VI. While training, Alice_k monitors model improvements in terms of accuracy and loss at the end of every epoch, with an early stopping patience of 10.
 - VII. If there is an improvement, Alice_k saves the improved model to a file, this alerts Bob_k.
 - VIII. Bob_k encrypts the new improved model file using Charity's public key and send the file to Charity
 - IX. Upon receipt, Charity runs the model from Bob_k on local data while comparing with her current model
 - X. If the new model from Bob_k is better than the current model, Charity saves the new model as the currently available model, else the model from Bob_k is discarded.
 - XI. When Alice_{k+1} wants to train, it will use the currently available model from Charity as the starting point.
 - XII. End

Pictorial Representation of the Algorithm

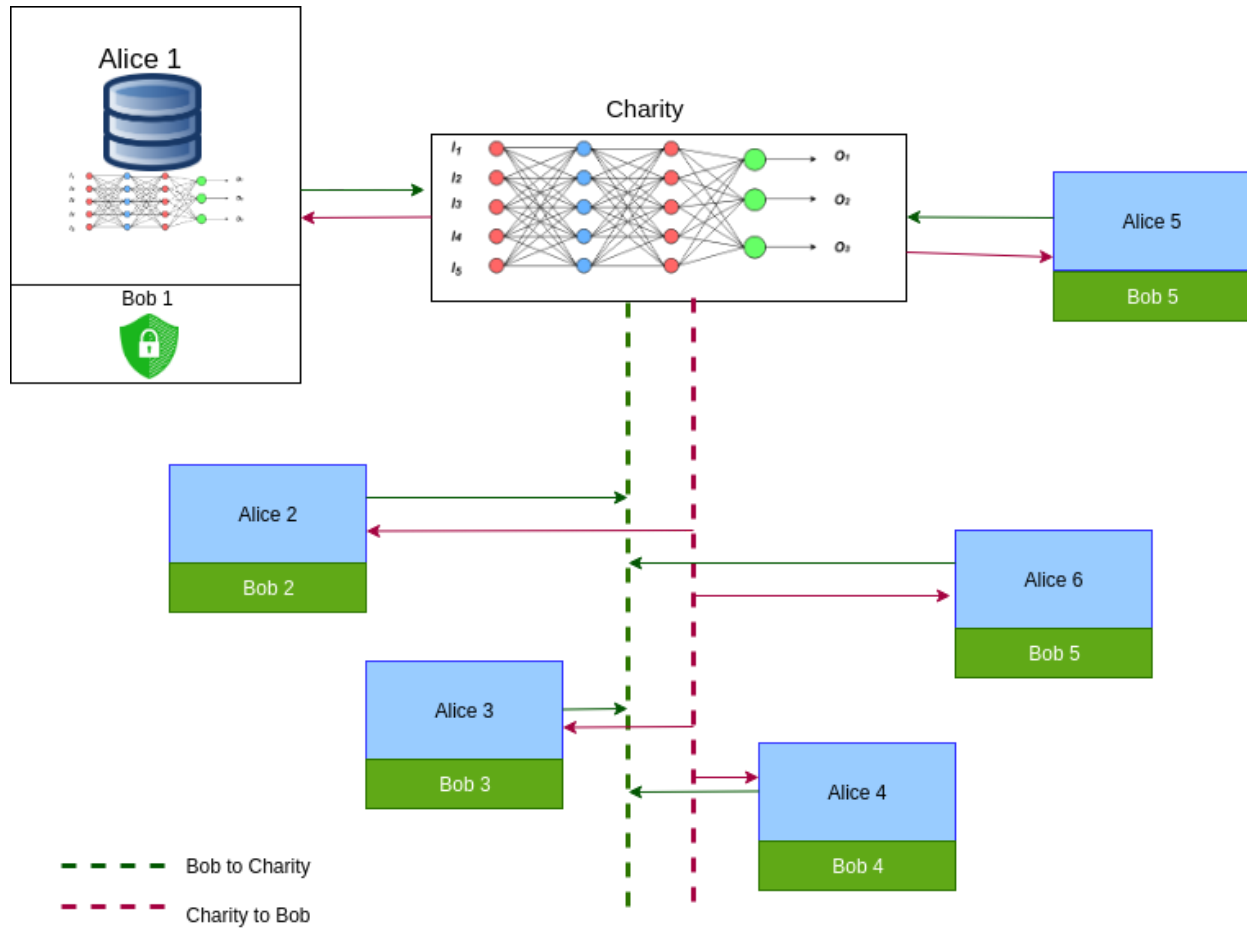


Figure 4.1: Pictorial representation of the algorithm.

4.4. Model Architecture

The figure below shows a summary of the convoluted neural network architecture that was used;

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 18, 18, 20)	560
activation_1 (Activation)	(None, 18, 18, 20)	0
conv2d_2 (Conv2D)	(None, 16, 16, 44)	7964
activation_2 (Activation)	(None, 16, 16, 44)	0
conv2d_3 (Conv2D)	(None, 14, 14, 68)	26996
activation_3 (Activation)	(None, 14, 14, 68)	0
conv2d_4 (Conv2D)	(None, 12, 12, 92)	56396
activation_4 (Activation)	(None, 12, 12, 92)	0
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 92)	0
flatten_1 (Flatten)	(None, 3312)	0
dense_1 (Dense)	(None, 120)	397560
activation_5 (Activation)	(None, 120)	0
dense_2 (Dense)	(None, 1)	121
activation_6 (Activation)	(None, 1)	0
Total params: 489,597		
Trainable params: 489,597		
Non-trainable params: 0		

Figure 4.2: Summary of the model architecture and parameters

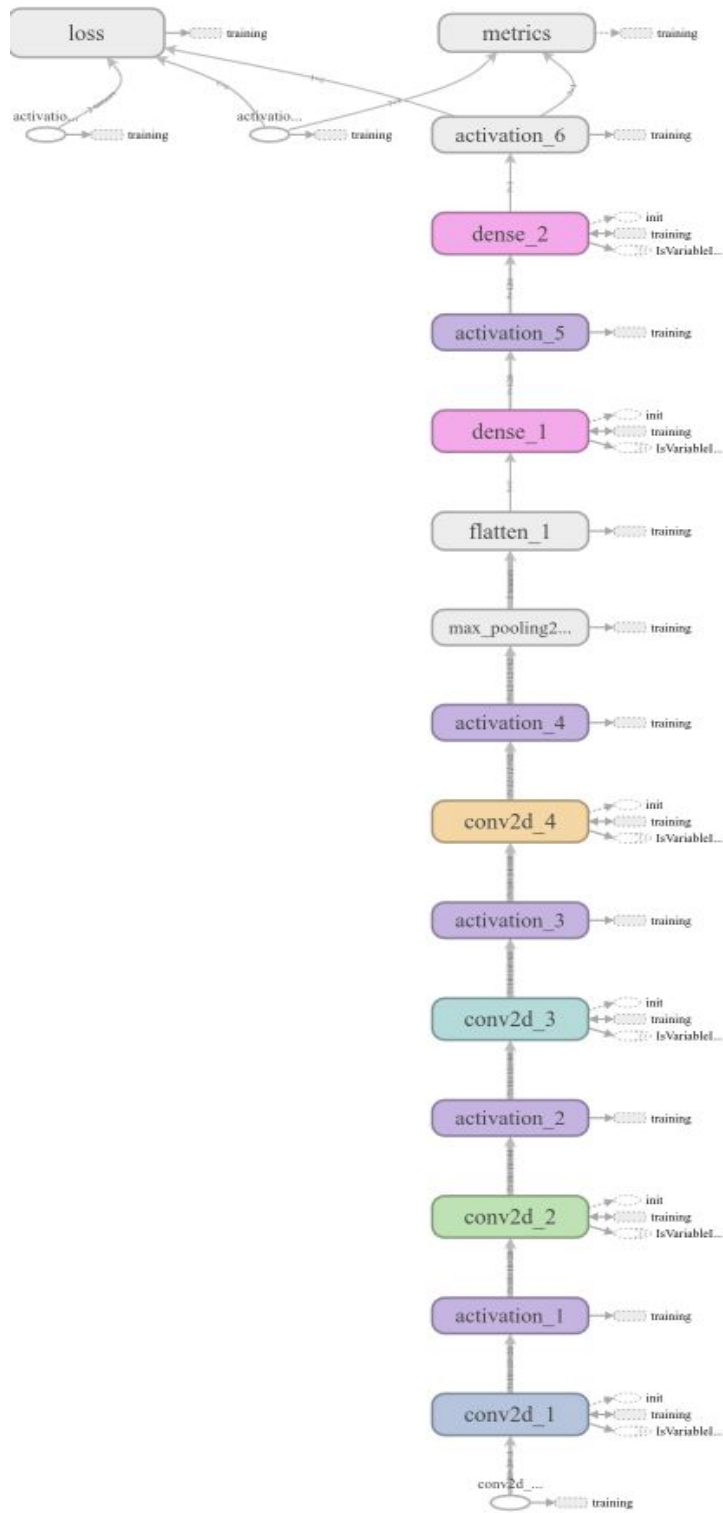


Figure 4.3: Model architecture

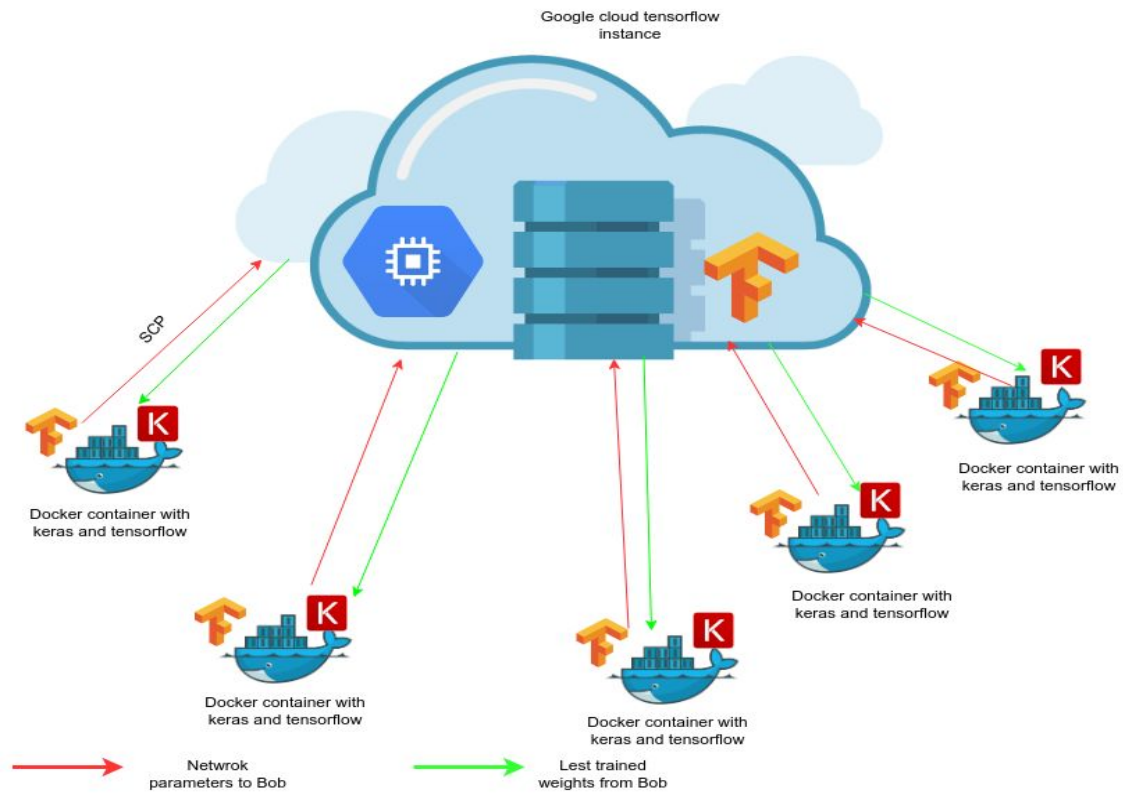
4.5 Initial Model Inputs

- Loss - The loss function was used to determine the model's success or failure. Each iteration of training, it calculated its loss using this function to evaluate its current performance, and tweak the model parameters based on this feedback. Binary cross entropy was used as the loss function..
- Optimizer - Defined how the parameters will be tweaked (i.e., should the parameters be modified by a large or small amount?). Adam optimizer was used in this model.
- Metrics - Accuracy was used as the main metric to judge performance of the model.

4.6. Implementation

The algorithm and protocol was implemented using python Keras library running on docker containers with Tensorflow backend. Each of the docker containers serve as independent agents/nodes, called Alice in this case, on the distributed environment with their own data of about 5000 images and running through a copy of the CNN model. Though the nodes are distributed, they communicate through a centralised node called Bob in this case.

At the beginning of a training cycle, Bob initializes random weights and sets them as weights of last trained node. Before any Alice starts training, they request for the weights of the last trained node from Bob and use the data to initialize their network. Communication between Alices and Bob is by file transfer using the SCP protocol. The diagram below depicts the algorithm implementation. For efficiency, Bob runs on a cloud instance with better infrastructure.



Text

Figure 4.4: Distributed architecture

4.7. Implementation Comparison with Blockchain

A blockchain is a linked list of digital records in packages (blocks) which are distributed and secured using cryptography. Each block contains data that is cryptographically hashed and timestamped. Subsequent blocks in the chain depend on previous blocks thus ensuring that data is tamper proof. From the recent past, blockchain technology is being adopted in implementing distributed machine learning where computing resources are offered as a service with notable projects such as OpenMined (Openmined.org, 2018) and DML (Decentralizedml.com, 2018). Though there are several similarities between this implementation and blockchain-based implementation of distributed machine learning, below are the key differences between this approach and implementing machine learning on blockchain;

1. The main focus of blockchain machine learning is to offer computing resources as a service whereas the focus of this implementation is to share data representation as a service.
2. In blockchain implementation, the model is not trained on user's data, but on someone else's data located somewhere on the blockchain network meaning the user has no control of the data. In this project's implementation, though there is no data sharing, all the nodes on the distributed network are well known because the network coordination is still centralised.
3. In blockchain implementation, the scope of training of a model is limited to only a localised data store in the sense that the resultant model is not trained on all data on the network. Each node runs an independent training and one with the best model is picked. In this implementation, the resultant model is trained jointly by sharing weights from each node, thus at the end of the training, all nodes will have a copy of the same trained model.

5.0 Results and Analysis

5.1 Proof of Correctness

The algorithm correctness stems from the fact that Bob and at least one of $Alice_k$ have identical neural network parameters to regular training at iteration k . The neural network being trained at iteration k is identical to the neural network if it was trained by just one entity. Alice 1 randomly initialized weights and Bob used these weights during first iteration. We assume that this initialization is consistent when training with single entity. In case another $Alice_j$ attempts to train, it will refresh the weights to correct value. $Alice_j$ performs backpropagation as the final step in iteration i . Since this backpropagation is functionally equivalent to backpropagation applied over the entire neural network at once, $Alice_j$ continues to have correct parameters at the end of one training iteration. ($F^T(\text{gradient})$ is functionally identical to sequential application of $F_{a,j}(F_b^T(\text{data}))$)

5.2 Accuracy

Accuracy was used to evaluate the performance of this model. The plots below indicate that the model generally improved every epoch until it hit a maximum accuracy of 93.63% before smoothing and a final accuracy of 93.55% after smoothing. Accuracy was calculated by finding the ratio between the correctly predicted classes and the total number of predictions. The slope of this curve is relatively similar to one achieve by training a centralised model on a centralised database.

5.2.1 Accuracy before smoothing

Maximum Value: 99.81%

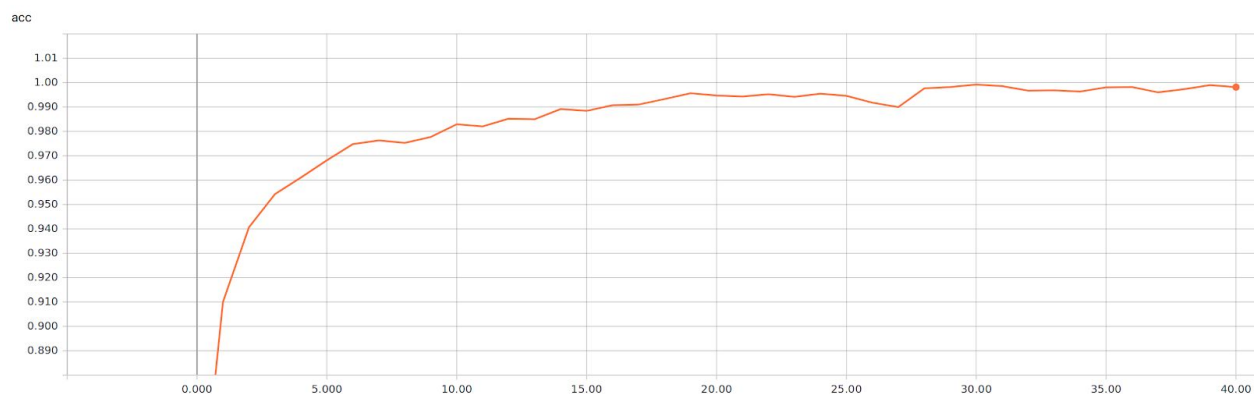


Figure 5.1: Accuracy before smoothing; y-axis = accuracy, x-axis = number of epochs

5.2.2 Accuracy after smoothing

Initial Value: 0

Final Value: 99.81%

Smoothing Factor: 0.6

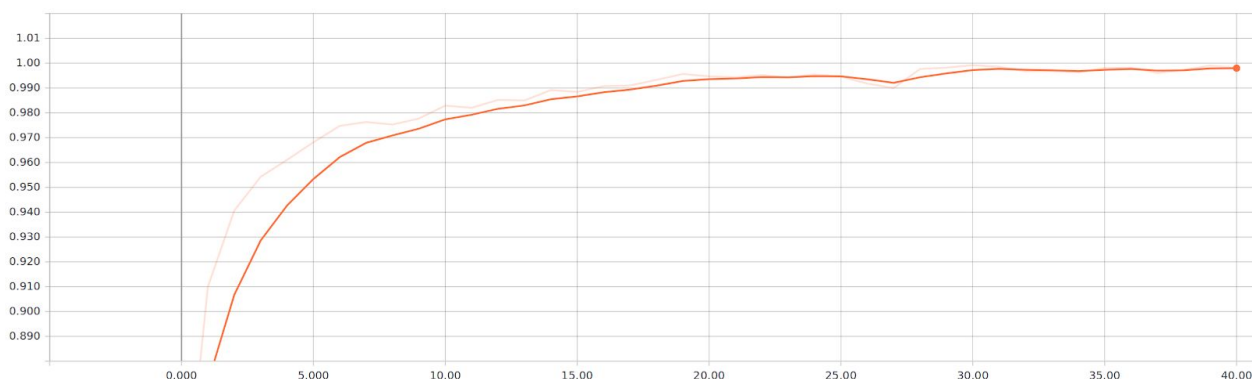


Figure 5.2: Accuracy after smoothing; y-axis = accuracy, x-axis = number of epochs

5.3 Loss

The objective of the model was to minimise loss while increasing accuracy. During each iteration of training, the loss was calculated and used to evaluate current performance, and tweak the model parameters based on this feedback. Binary cross entropy was used as the loss function. From the graphs below, the loss decreases steadily in every epoch.

5.3.1 Loss before smoothing

Minimum Value: 0.1936

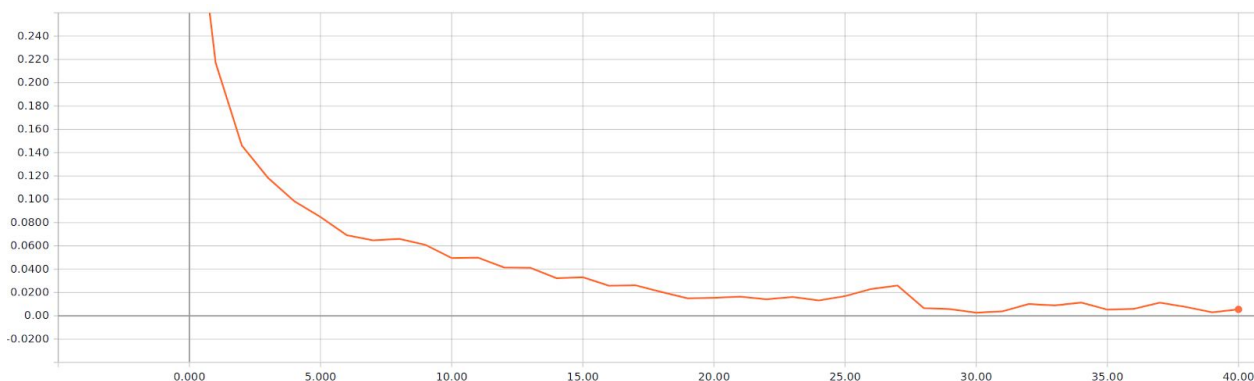


Figure 5.3: Loss before smoothing; y-axis = loss, x-axis = number of epochs

5.3.2 Loss after smoothing

Final Value: 5.4970e-3

Smoothing Factor: 0.6

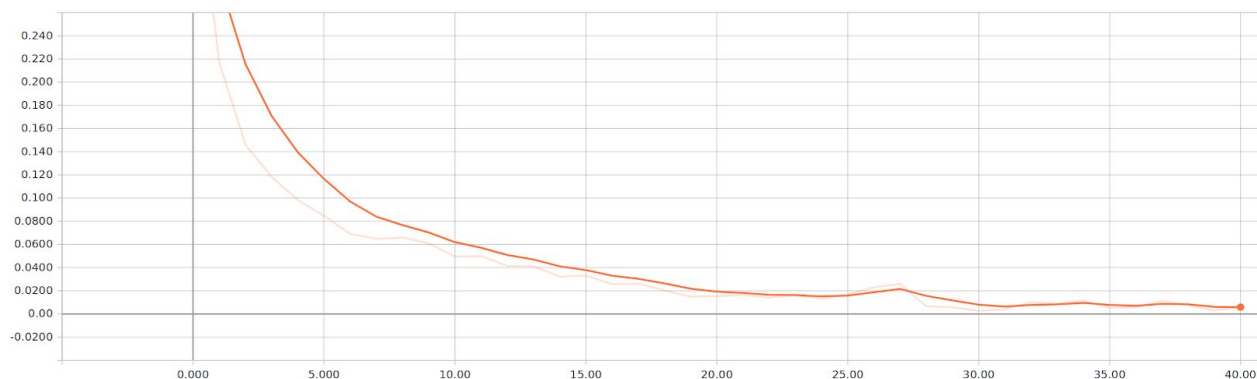


Figure 5.4: Loss after smoothing; y-axis = loss, x-axis = number of epochs

5.4 Comparison with centralised models

The same model was also trained on centralised architecture. There was no difference between the distributed architecture and the centralised architecture in terms of final loss and accuracy achieved, but there was a significant difference in terms of time taken to converge. Each node on the distributed network training on about 37000 images, doing 100 epochs and a batch size of 200 images took an average of 96 minutes to converge. For a centralised architecture with the same number of images, batch size, epoch and same infrastructure resource configuration as one node on the distributed architecture, took close to 3 times average time taken on the distributed learning architecture.

5.5 Effect of increasing the number of layers on accuracy and loss

There was no significant difference on accuracy and loss when the number of hidden layers was doubled from 4 to 8 but there was a significant difference on the number of epochs required and time taken to converge as shown on the charts below;

Accuracy

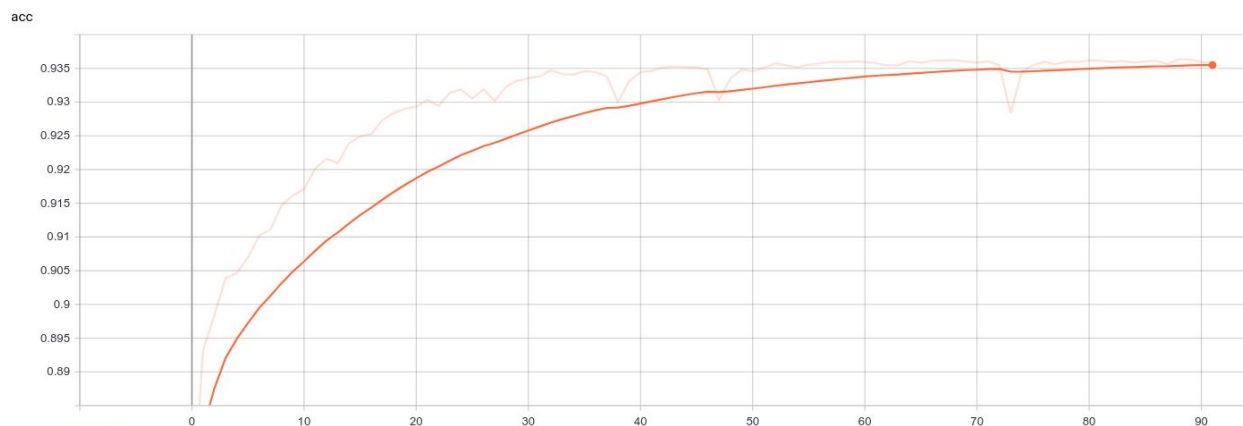


Figure 5.5: Accuracy after doubling number of hidden layers; y-axis = accuracy, x-axis = number of epochs

Loss

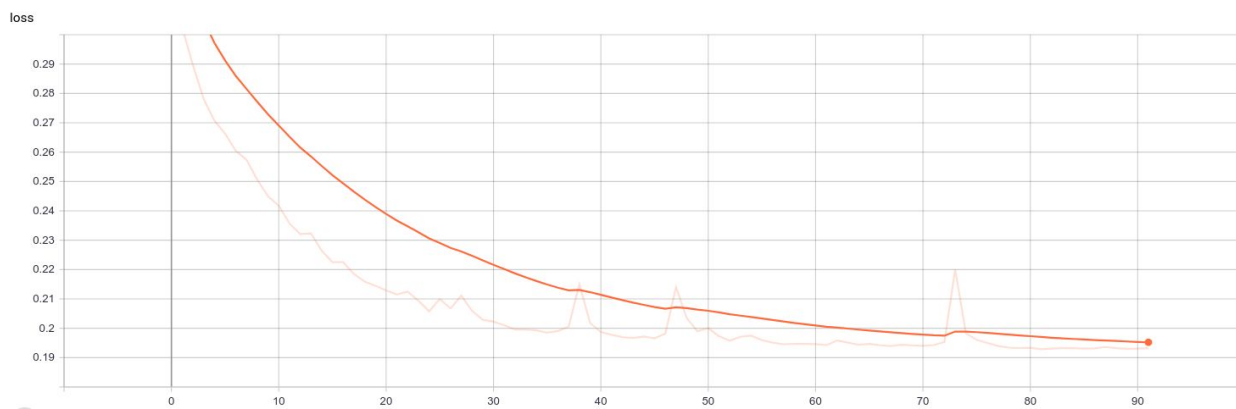


Figure 5.6: Accuracy after doubling number of hidden layers; y-axis = accuracy, x-axis = number of epochs

6.0 Conclusion and Future Work

This research work has proved beyond reasonable doubt the practicability of training a shared deep learning model in an unstructured data environments without having to share actual data. Eliminating the need to share data or the need to aggregate data in a data warehouse ensures security of the data in the nodes where it is generated. Communication security while exchanging weights and other network parameters can be ensured by leveraging private-public authentication on SCP,SFTP and SSH mechanisms. Thus databases that handle sensitive data such as patients' medical records in electronic medical records can benefit from the application of machine learning techniques on their data without compromising on data security and privacy of patients.

This approach can also be beneficial in low resource and low data scenarios by training a deep learning model on several smaller databases as if they were a single data repository. Thus big data, data security and limited computational resources should not hinder developing and adopting machine learning algorithms in industry.

Recommendations and Future work

Though this approach has been proven to work here, having a centralised coordinator (Bob) will be will break the whole network whenever Bob is down or his resources are overwhelmed by the number of nodes connecting. A reasonable extension to this research work can be to develop a technique to have several coordinators or who are synchronised and the nodes can connect to the nearest one. An algorithm can also be developed to have the nodes intelligently pick a new coordinator whenever the central coordinator breaks down. It would also be interesting to explore using blockchain technology to shared model parameters in situations where each of the nodes should remain completely hidden from the other nodes or where each node should be rewarded based on their contribution to the final model.

References

1. Blueshift.com. (2018). *The Leading Customer Data and Engagement Platform for Marketers*. [online] Available at: <https://blueshift.com> [Accessed 24 Oct. 2018].
2. Columbus, L. (2018). *77% Of Marketing Execs See AI Adoption Growing This Year*. [online] Forbes. Available at: <https://www.forbes.com/sites/louiscolombus/2018/05/14/77-of-marketing-execs-see-ai-adoption-growing-this-year/#45af2fd77ef8> [Accessed 24 Oct. 2018].
3. Medium. (2018). *The future of AI: from siloed data to shared expertise*. [online] Available at: <https://medium.com/element-ai/the-future-of-ai-from-siloed-data-to-shared-expertise-a5391ee18444> [Accessed 24 Oct. 2018].
4. Decentralizedml.com. (2018). *Decentralized Machine Learning - DML*. [online] Available at: <https://decentralizedml.com/> [Accessed 25 Oct. 2018].
5. BigAI. (2018). *Blockchain based Decentralized Artificial Intelligence & Machine Learning - BigAI*. [online] Available at: <https://bigai.io/en/> [Accessed 25 Oct. 2018].
6. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. (2000). The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23(3), pp.187-200.
7. Snips. (2018). *Snips — Using Voice to Make Technology Disappear*. [online] Available at: <https://snips.ai/> [Accessed 27 Nov. 2018].
8. McMahan, B. and Ramage, D. (2017). *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. [online] Google AI Blog. Available at: <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html> [Accessed 28 Nov. 2018].
9. McMahan, B., Moore, E., Ramage, D., Hampson, S. and Aguera, B. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data.
10. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, B. and Patel, S. (2017). Practical Secure Aggregation for Privacy-Preserving Machine Learning.
11. Openmined.org. (2018). *OpenMined*. [online] Available at: <https://openmined.org/> [Accessed 3 Dec. 2018].
12. Rodriguez, J. (2018). *Technology Fridays: OpenMined Powers Federated AI Using the Blockchain*. [online] Towards Data Science. Available at: <https://towardsdatascience.com/technology-fridays-openmined-powers-federated-ai-using-the-blockchain-d124e6560dd> [Accessed 3 Dec. 2018].
13. Cohen, G., Afshar, S., Tapson, J. and van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters.
14. IBM.com. (2018). *Big Data Analytics*. [online] Available at: <https://www.ibm.com/analytics/hadoop/big-data-analytics> [Accessed 6 Dec. 2018].

15. Bernes, J. (2017). *The AI-First Business Model – Element AI – Medium*. [online] Medium. Available at: <https://medium.com/element-ai/the-ai-first-business-model-fcc41c069440> [Accessed 6 Dec. 2018].
16. Cpsc.yale.edu. (2018). *Distributed Computing | Computer Science*. [online] Available at: <https://cpsc.yale.edu/research/distributed-computing> [Accessed 6 Dec. 2018].
17. Elsevier Connect. (2016). *Seven ways predictive analytics can improve healthcare*. [online] Available at: <https://www.elsevier.com/connect/seven-ways-predictive-analytics-can-improve-healthcare> [Accessed 17 May 2016].
18. Hackernoon.com. (2019). *3 Ways Blockchain Could Unleash the Full Potential of Machine Learning*. [online] Available at: <https://hackernoon.com/3-ways-blockchain-will-unleash-the-full-potential-of-machine-learning-3d3a4d350b1> [Accessed 8 Aug. 2019].
19. CoinDesk. (2019). *What is Blockchain Technology? - CoinDesk*. [online] Available at: <https://www.coindesk.com/information/what-is-blockchain-technology> [Accessed 8 Aug. 2019].
20. Harvard Business Review. (2019). *The Truth About Blockchain*. [online] Available at: <https://hbr.org/2017/01/the-truth-about-blockchain> [Accessed 9 Aug. 2019].
21. Anwar, H. (2019). *The Ultimate Blockchain Technology Guide: A Revolution to Change the World*. [online] 101 Blockchains. Available at: <https://101blockchains.com/ultimate-blockchain-technology-guide/> [Accessed 9 Aug. 2019].
22. Gupta, O. and Raskar, R. (2018). Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116, pp.1-8.

Appendix 1: Project Plan and Management

1.2 Required Resources

a. Software Resources

- i. Pycharm IDE (Ultimate Edition V14.1)
- ii. PostgreSQL 10.6 Server
- iii. Git and Github (Version Control)
- iv. GanttProject for Project Management
- v. Ubuntu 18.04 LTS Operating System for deployment
- vi. Java Enterprise Edition - for backend development
- vii. PostgreSQL JDBC Driver
- viii. Python Keras/Tensorflow ML Library

2. Hardware Resources

- a. PC (Intel(R) Core(TM) i7 Processor, 16GB RAM)
- b. 10 Linode cloud server; 2GB RAM each

Appendix 2: Sample Scripts

2.1 CNN Initialization

```
def cnn(size, n_layers):
    # INPUTS
    # size - size of the input images
    # n_layers - number of layers
    # OUTPUTS
    # model - compiled CNN

    # Define hyperparameters
    MIN_NEURONS = 20
    MAX_NEURONS = 120
    KERNEL = (3, 3)

    # Determine the # of neurons in each convolutional layer
    steps = np.floor(MAX_NEURONS / (n_layers + 1))
    neurons = np.arange(MIN_NEURONS, MAX_NEURONS, steps)
    neurons = neurons.astype(np.int32)

    # Define a model
    model = Sequential()

    # Add convolutional layers
    for i in range(0, n_layers):
        if i == 0:
            shape = (size[0], size[1], size[2])
            model.add(Conv2D(neurons[i], KERNEL, input_shape=shape))
        else:
            model.add(Conv2D(neurons[i], KERNEL))

        model.add(Activation('relu'))
    # Add max pooling layer
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(MAX_NEURONS))
    model.add(Activation('relu'))

    # Add output layer
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    # Compile the model
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    # Print a summary of the model
    model.summary()

    return model
```

2.2 Model Instantiation, training and saving

```
# Instantiate the model
model = cnn(size=image_size, n_layers=N_LAYERS)

# Training hyperparameters
EPOCHS = 150
BATCH_SIZE = 200

# Early stopping callback
PATIENCE = 10
early_stopping = EarlyStopping(monitor='loss', min_delta=0, patience=PATIENCE, verbose=0, mode='auto')

# Tensorboard callback
LOG_DIRECTORY_ROOT = '/tmp/tensorflow'
now = datetime.utcnow().strftime("%Y%m%d%H%M%S")
log_dir = "{}run-{}".format(LOG_DIRECTORY_ROOT, now)
tensorboard = TensorBoard(log_dir=log_dir, write_graph=True, write_images=True)

# callback to save model at the every epoch
model_save_file = 'epoch-model.hdf5'
save_model = ModelCheckpoint(model_save_file, monitor='loss', verbose=0, save_best_only=True,
                             save_weights_only=False, mode='auto', period=1)

# Place the callbacks in a list
callbacks = [early_stopping, tensorboard, save_model]

# Train the model
model.fit(x_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE, callbacks=callbacks, verbose=0)
# print(x_train)
```

2.3 Communication daemon script

```
def scptoserver(filename):
    ssh = SSHClient()
    ssh.load_system_host_keys()
    ssh.connect('orxbit@68.183.162.143', port=22)
    with SCPClient(ssh.get_transport()) as scp:
        scp.put(filename)

class Watcher:
    DIRECTORY_TO_WATCH = "models/in"

    def __init__(self):
        self.observer = Observer()

    def run(self):
        event_handler = Handler()
        self.observer.schedule(event_handler, self.DIRECTORY_TO_WATCH, recursive=False)
        self.observer.start()

        try:
            while True:
                time.sleep(5)

        except:
            self.observer.stop()

        self.observer.join()

class Handler(FileSystemEventHandler):
    @staticmethod
    def on_any_event(event):
        if event.is_directory:
            print "Received created event."

            elif event.event_type == 'created':
                scptoserver('models/epoch-model.hdf5')

            elif event.event_type == 'modified':
                scptoserver('models/epoch-model.hdf5')

if __name__ == '__main__':
    w = Watcher()
    w.run()
```