Evaluating the Implementation of Deep Learning in LibreHealth Radiology on Chest X-Rays

Saptarshi Purkayastha [1], Surendra Babu Buddi [1], Siddhartha Nuthakki [1], Bhawana Yadav [1] and Judy W. Gichoya [2]

[1] Indiana University Purdue University Indianapolis, Indianapolis IN 46202, USA
[2] Oregon Health & Science University, Portland, Oregon 97239, USA
saptpurk@iupui.edu, gichoya@ohsu.edu

Abstract. Respiratory diseases are the dominant cause of deaths worldwide. In the US, the number of deaths due to chronic lung infections (mostly pneumonia and tuberculosis), lung cancer and chronic obstructive pulmonary disease has increased. Timely and accurate diagnosis of the disease is highly imperative to diminish the deaths. Chest X-ray is a vital diagnostic tool used for diagnosing lung diseases. Delay in X-Ray diagnosis is run-of-the-mill milieu and the reasons for the impediment are mostly because the X-ray reports are arduous to interpret, due to the complex visual contents of radiographs containing superimposed anatomical structures. A shortage of trained radiologists is another cause of increased workload and thus delay. We integrated CheXNet, a neural network algorithm into the LibreHealth Radiology Information System, which allows physicians to upload Chest X-rays and identify diagnosis probabilities. The uploaded images are evaluated from labels for 14 thoracic diseases. The turnaround time for each evaluation is about 30 seconds, which does not affect clinical workflow. A Python Flask application hosted web service is used to upload radiographs into a GPU server containing the algorithm. Thus, the use of this system is not limited to clients having their GPU server, but instead, we provide a web service. To evaluate the model, we randomly split the dataset into training (70%), validation (10%) and test (20%) sets. With over 86% accuracy and turnaround time under 30 seconds, the application demonstrates the feasibility of a web service for machine learning based diagnosis of 14-lung pathologies from Chest X-rays.

Keywords: Deep Learning, Radiology, LibreHealth, Chest X-Ray, CheXNet.

1 Introduction

1.1 The challenge of chronic respiratory disease

Chronic respiratory diseases include Asthma, Chronic Obstructive pulmonary disease (COPD), Emphysema, cystic fibrosis, tuberculosis, lung cancer and pneumonia which are caused to the airways and other parts of the lungs. The deaths due to these diseases are increasing each year over the past 35 years in the United States of America. Many reports estimate that more than 4.6 million Americans have died due to chronic respiratory illness from 1980 to 2014 [1]. Late detection or misdiagnosis of the disease is the

main reason for increasing the mortality rates, length of stay and health care costs [2]. In the past, lung diseases were detected through a physical Chest checkup as they are routinely available and cost-effective [2]. In recent times, diagnosis is mainly done using laboratory reports, microbiological tests, and radiographs, mainly Chest X-ray images. However, the WHO mentions that Chest X-ray remains the most commonly used technique because of its wide availability, low radiation dose, relatively inexpensive and is the best available method in diagnosing pneumonia [8]. However, Chest X-ray images are very hard to interpret due to the complex anatomical structure of the lungs that are superimposed or due to the infiltrates formed in the blood vessels.

1.2 The promise of machine learning methods for reading Chest X-rays

In recent years, artificial intelligence has gained tremendous importance in assisting physicians to take better clinical decisions. The CheXNet algorithm is a state-of-the-art machine learning algorithm to detect pneumonia at a level of a human practicing radiologist. It is a 121 layer Convoluted Neural Network trained on the Chest X-ray 14 dataset, containing over 100,000 frontal view X-ray images, published by the National Institutes of Health. The CheXNet algorithm can identify 14 pathologies from a Chest X-ray. The performance of CheXNet reported an F1 score of 0.435 which exceeded the average F1 metric of 0.38 for four radiologists in previous studies [6]. Automated detection of diseases from Chest X-rays at this level of performance is invaluable in healthcare delivery in populations with limited access to diagnostic imaging specialties. However, there is no implementation of these algorithms in the real-world electronic health record (EHR) or radiology information system. We believe that such algorithms should be made available to the physicians and health care professionals through EHRs to evaluate the feasibility of such tools in clinical workflows, and the usability of such an integrated AI system.

This paper describes the work of integrating the CheXNet deep learning algorithm into the LibreHealth Radiological Information System (RIS) which is an open source distribution of an EHR system.

2 Background

Wang et al. [5] proposed a 2D ConvNet for classifying abnormalities in Chest X-Ray images by using a simple binary approach and published the most extensive publicly available ChestX-Ray8 dataset. The dataset unified multi-label classification and disease localization techniques for identifying eight thoracic diseases. The eight diseases were used as keywords to pull information from radiology reports and the related images from a Picture archiving and communication system (PACS). This database consists of 108,948 frontal-view Chest X-ray images which were labeled according to their pathology keywords by using natural language processing (NLP). The images were labelled by searching for diseases in findings and impressions of radiology reports and was labelled as “Normal” if they don’t find any disease. These reports were further mined by using DNorm and MetaMap. The negative pathological statements were eliminated by using parsers (Bllip parser) in NLTK. The quality control of disease labelling

is done by considering the human annotators and a subset of reports were used as the gold standard. The images in the Chest X-ray 8 database consists of 1024 x 1024 pixels images along with detailed contents of the images. To evaluate the disease localization performance, some of the reports were hand labelled by adding bounding boxes (B-boxes). To detect the images with multiple labels, DCNN classification containing 4 pre-trained models (AlexNet, ResNet-50, VGGNet-16, GoogLeNet) was used to generate a heat map showing the likelihood of pathologies. This network takes the weights from these models, while the prediction and transition layers are trained from DCCN. For classifying and localizing pathologies, the data sets was divided into three subgroups - training (70%), testing (20%) and validation (10%). This DCNN model was trained using a Linux server containing 4 Titan GPU server. Due to the limited memory size of GPUs, the batch size had to be reduced to load the model.

Yao et al. [3] addressed the problems faced in conditional dependencies both in interface and training while predicting the multiple labels. They used a 2D ConvNet model to study the images and further used recurrent neural networks (RNN) to encode the information from the previous predictions done by the model. They used the sigmoid design to predict because that addressed the issue conveniently at each step in prediction. They tested the performance of the model by splitting the data sets in training (70%), testing (20%), and validation (10%), similar to Wang et al. [5].

Rajpurkar et al. [6] designed a 121-layer dense convolutional neural network (DenseNet) trained on the Chest X-ray 14 dataset containing 112,120 frontal Chest X-ray images labeled with 14 thoracic diseases, using the same process as Wang et al. [5]. They used batch normalization and dense connections techniques to improve the flow of gradients and information in the network. This network used the weights from a pre-trained model on ImageNet [4] and further trained end-to-end using standard parameters of Adam. As per the ImageNet training dataset, the images from the Chest X-ray 14 were downscaled from 1024x1024 to 224 x 224. For training the model, the dataset was split into training (98637 images), testing (420 images) and validation (6351 images). The test dataset was then labeled by four radiologists practicing at the Stanford University. This model was trained with the batch size of 16. The performance of the model is compared with that of radiologists. They found that the CheXNet model exceeded the average performance of radiology by F1 metrics.

Yadong Mu (2017) then improved this model by adopting a ten crop method both in validation and test sets. They have slightly improved the model to calculate the mean AUROC value and per-class AUROC value of the improved model is similar to the model CheXNet. We used this model, and its PyTorch implementation to integrate CheXNet into the LibreHealth RIS.

3 Methodology

3.1 Project Objective

The work reported in this paper brings an important addition to the abovementioned long-term progress of CheXNet. The following were our main objectives:

1. We wanted to implement the work done on CheXNet, which had proved to be more efficient diagnostic technique when compared to other algorithms, into a real-world EHR system and validate if the performance worked on other Chest X-ray images.
2. We wanted to get to the fastest time in diagnosing from a Chest X-ray, such that we could implement it locations where there is a lack of radiologists. LibreHealth RIS is commonly used in low- and middle-income countries, and thus, we decided to integrate our work with this EHR system.
3. We wanted to build an architectural innovation, such that the CheXNet algorithm could be deployed over a web service, such that physicians and other users of the LibreHealth EHR/RIS did not have to own a powerful graphics processing unit (GPU) or server, and could merely upload images and get back the diagnosis results from the web service. This would enable simplified distribution of our innovation, and would allow more substantial impact of our work

3.2 Software development

While initiating the work we reviewed all the existing models, we wanted to work on a model with a related dataset and strong, proven, and easily distributable algorithm. We selected the CheXNet model by Yadong Mu (2017), which is a PyTorch reimplementation of CheXNet. CheXNet currently works on the largest publicly available Chest X-ray dataset. It is a 121-layer convolutional neural network trained on Chest X-ray 14, containing over 100,000 frontal view X-ray images with 14 diseases.

For this project, we used the rapid prototyping methodology [7]. In the prototype that we built, the LibreHealth radiology module which consisted of all the capabilities of the radiology information system was used as the base module. The radiology module required the installation of a series of prerequisites which includes Java JDK 8, Maven Build tool, Node JS, Docker. We decided to use Java maven for this project, as it helped us in getting all the required libraries and plugins to handle the module's routine tasks. To use maven, we had to ensure its compatibility with the JDK version for which we had to make some minor changes to LibreHealth core codebase. In the development process, we followed an iterative process of design, build, test, feedback and re-design based on the feedback from the previous cycle. In the initial phase of the module design, we emphasized designing and evaluating the requirements of the project demonstrating the functionality and performance of the system. Use of Docker helped in streamlining the development lifecycle by allowing us to work in a standardized development environment using local containers with all parts of the LibreHealth core platform and its MySQL database. We used containers for continuous integration and continuous delivery workflows, such that any development that we did could be automatically tested for performance metrics which we planned to improve after every development cycle. The radiology module and its docker image are also built by using Maven, which made the build process easy and helped to provide a uniform build system.

We used the LibreHealth RIS codebase: https://gitlab.com/librehealth/radiology/lh-radiology) to fork and create a development branch. This proved to be hard to manage as there was parallel work that was done by other contributors from the open-source

community, and thus we had to switch to a modular approach, such that our web application does not get impacted by any code changes to the other parts of the LibreHealth RIS. LibreHealth RIS code repository contains useful tools for radiology and imaging, which proved very helpful in this project. Post repository cloning we used docker to build and run the radiology module, on top of which we started to develop our web application. This assembled docker image was later published to docker hub so that all developers could share the starting codebase to begin working on the app. The new Docker image was created based on the open source lh-radiology-docker, along with integrating required modules, which are described below. To retrieve all the radiology terminology services required by the radiology module, the core concept dictionary was installed. Radiology concepts were also imported from CIEL dictionary.

We also integrated four other publicly available open-source modules as shown in Figure 1. below. The radiology module v1.4.0 package file (called OMOD) of the radiology module along with LibreHealth core called lh-toolkit was integrated into this docker image. The radiology module is the core RIS features such as procedures for ordering or placing the radiology orders. The functionality to view the DICOM images and creating the reports after completing the radiology orders are also part of the radiology module. Secondly, the OMOD of the Legacy UI modules was downloaded from the add-ons repository of the OpenMRS EHR, another open-source EHR platform on which LibreHealth has been developed. This module consists of all the administrative functions of the OpenMRS along with the patient dashboards. This module includes features such as finding/creating patients, concept dictionary management and all the essential functions of an EHR system. Then the OMOD of REST web services module was downloaded from OpenMRS and integrated. It is used to run RESTful web services of the OpenMRS EHR, which can be used by client-side HTML applications for data exchange with the EHR system. Lastly, the open web apps (OWA) v1.4 module of OpenMRS was integrated into the docker image. This module allows users to deploy open web apps which consists of HTML, JavaScript, CSS, and a manifest file as a zip package, for it to be launched as client-side applications. The module was useful in altering the User Interface on top of Rest Webservices for our project.

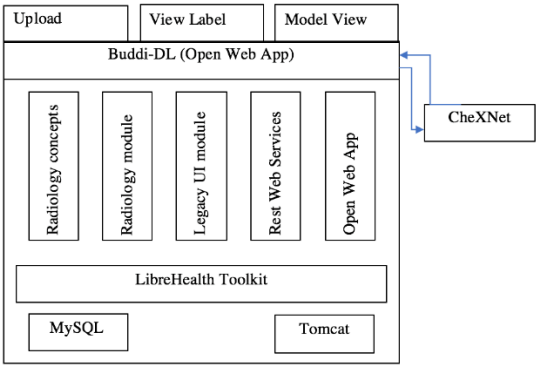

Fig. 1. The architecture of the project using LibreHealth Toolkit as the base and OpenMRS modules to deploy an open web app, which communicates with the CheXNet web service.

The best performing CheXNet was cloned from GitHub - https://github.com/arnoweng/CheXNet). Labeled ChestX-ray14 dataset images were downloaded from the National Institutes of Health - https://nihcc.app.box.com/v/ChestXray-NIHCC into a server which had four Nvidia 1080Ti graphics processing units (GPU). We uploaded the model.py, model.pth.tar (the trained model) to the GPU server and updated the model to use PyTorch 0.4.0. All the medical terms and concepts of the 14 thoracic diseases were created in LibreHealth RIS. These concepts were also clubbed into ConvSet (Convention Set) of clinical terms as a list so the CheXNet can report back accurately under the appropriate diagnosis concept.

3.3 Development of Buddi-DL, the open web app for LibreHealth RIS

An Open web app's rough skeleton was created and named Buddi-DL (Deep Learn) after the primary developer and one author of this paper. We scaffold the open web app by installing Node JS to generate the boilerplate Open Web App (OWA) by using a NodeJS tool called Yeoman, by following the development workflow defined here - https://wiki.openmrs.org/display/docs/Open+Web+App+Development+Workflow.
The HTML, CSS, and JavaScript for the OWA could then be uploaded into the OpenMRS server using the Open Web Apps module, as shown in Fig. 2. The code for the buddi-dl OWA development is here: https://gitlab.com/Surendra04buddi/buddi-dl

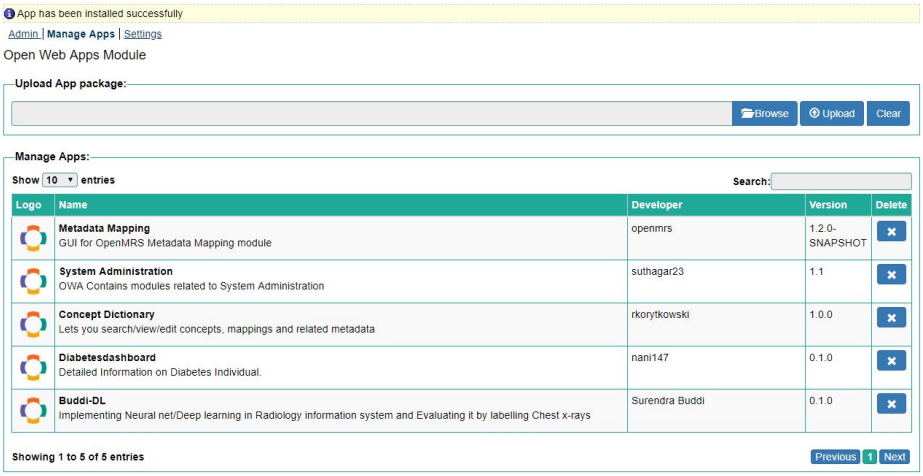

Fig. 2. The LibreHealth user-interface to upload the Buddi-DL OWA

The main innovation of the project was creating a Flask application with Python-based Flask framework. The Flask application hosts a web service which receives an image and then executes the code in the backend on the GPU server for CheXNet evaluation on that image, and then returns the list of lung diagnosis and their probabilities. This innovative web service allows the client-side application not to own a GPU server to run the CheXNet algorithm. The Flask application serves as a medium between the OWA and the CheXNet model on the GPU server, so that the user can use the OWA to

upload the images. Using RESTful API that was deployed using the Flask framework, the OWA calls the URL endpoint to post the image, and this triggers the execution of the CheXNet model. All computation intensive tasks are completed on the server, and a response is sent back once the processing is completed. The OWA receives a push callback from the Flask REST service, once the processing is completed and the output diagnosis is shown on the OWA.

Fig. 3. The form on the LibreHealth RIS using the OWA through which the Chest X-Ray image is uploaded, and the Impression is returned into the form

4 Results

The radiologist or clinical user can log in to the EHR system and open the patient record and then upload the images into the OWA using a mobile phone or low-end laptop, which has any web browser. We tested the OWA to work with all major web browsers such as Internet Explorer, Chrome, Firefox, and Edge. All browsers were able to upload the images and receive the results back from the Flask CheXNet web service. A total of 180 new images were uploaded using the OWA for the test process. These images were different from the Chest X-rays on which the CheXNet algorithm had been trained. These were labeled images with about 70% (n=126) were positive for pneumonia. This made the test dataset to be somewhat biased towards a positive detection of pneumonia, but we deliberately selected those images, because we wanted to test if out of the 14 diagnoses, we would get the accurate probabilities back from the CheXNet algorithm. Our system performed with 86% accuracy such that 108 images were returned with the highest probability of the diagnosis of pneumonia. In the remaining 18 positive images, pneumonia was among the top-3 diagnosis, but not the one with the highest probability. Thus, the integrated system worked reasonably accurately and showed similar performance to the model performance from previous studies on CheXNet.

All the calls made by OWA correctly triggered by Flask App, and further to the algorithm. After the images are successfully posted, the OWA makes a remote call to execute the model.py using a subprocess. After completing execution of the model, the Flask app was able to push a callback and send the data for the probability of the 14 thoracic diseases. The diseases with the highest probability should be considered by the clients. The model is altered such that after successfully executing the model the images posted is moved into other directories so that we were able to upload new images. All

of the functionality worked correctly and even with a poor and intermittent internet connection, we were able to receive data back whenever the client and server had re-connected. Thus, the flask application continued to push, until a successful acknowledgment from the client came back, and the similar case was from the client-side when the images were being sent by the client application to the Flask web service.

The primary metric for the evaluation of our project was the turnaround time for the AI system to work, since the accuracy of the model has been established by other researchers, as well as the usability of such an integrated system. All the 180 images returned their diagnosis within 30-seconds timeframe, with the minimum time being 22 seconds and the maximum time being 29 seconds, when the internet connections were kept stable. We deliberately disconnected and reconnected the internet to verify that the push callback was working correctly, and that was verified to work correctly. Formal usability tests were performed in each of the iterative cycles during the development phase with one informatician and one clinical user. That feedback was used to improve the usability of the system. Finally, as can be seen in Fig. 3., the form is very straightforward and does not feel any different from a regular LibreHealth or OpenMRS EHR system. The attempt here is to black box the backend of the AI system, such that the user only sees a simple front-end and all the heavy-lifting is done on the GPU server, without the user knowing the internal complexities. After the image is uploaded, we show a progress bar for the upload, and a waiting icon till the diagnosis impression is shown. Due to the complexity of implementing an accurate progress bar of the detection model from the CheXNet web service, we do not display it for the user.

5 Discussion

The integrated system met all the objectives that we had set when integrating the AI system with LibreHealth RIS. The performance of the models was as expected and did not require much effort to execute. The challenge for integration was mainly to do with creating the Flask web service which can deal with push callbacks and RESTful API which is generic for some different algorithms in the future. Another major challenge was to simplify the OWA user experience. With such a state-of-the-art backend, there was a lot of motivation from the developers to showcase a fancy internal view of whatever happened within the model and how the deep learning model worked in the hidden layers and made the classification. Yet, the simplification made it clear for clinical and informatics users, that this was better designed as a black box.

5.1 Limitations

The composition of Docker image which required the high runtime for the tomcat to launch on its own port was a major bottleneck at the start of the project. A persistent issue has to be able to deal with CAS authentication (Internal authentication system of Indiana University) which prevented posting of images into the GPU server. When we are trying to post images into the server, it was getting redirected to the CAS authentication page. The OWA failed to verify the authenticity of the login, particularly in

getting a ticket for the secure connection. Cross-Origin Resource Sharing (CORS) in most modern browsers also prevented the OWA from posting images and receiving the callback. Since the OWA is launched on a different domain from the Flask application, the CORS mechanism blocked the API requests from OWA. We dealt with this by implementing a JSONP format for data exchange. CUDA run-time errors for running PyTorch continue to be an issue, and the model had to be retrained due to backward compatibility issues with different versions of PyTorch. We have used the 128GB RAM, 4x Nvidia GTX 1080Ti GPU server, which is somewhat limited, if a large number of simultaneous models have to start evaluating images uploaded by many users. The server showed out of memory errors multiple times due to large batch size.

6 Conclusion

The detection of Chest X-ray images at the level of expert radiologists might be advantageous in assisting clinicians in healthcare settings, particularly in places with very limited radiologists. In other places, this might augment radiologists and help them evaluate Chest X-rays with a second perspective. The impressions obtained from this application can be overwritten by the physicians or radiologists and treatment can be given accordingly. This is amongst the first attempts at integration of Artificial Intelligence into clinical workflow in an open-source product. The primary goal of this project is to design a clinically meaningful automated system that can assist the physicians in analyzing the radiology images before they can get detailed reports from radiologists. We have developed an OWA which can be launched into any Docker image containing the Radiology module and Open Web App module for OpenMRS EHR or LibreHealth. This OWA may serve as a template of integration of Artificial Intelligence in Radiology.

References

1. World Health Organization. (2017). Noncommunicable diseases: progress monitor 2017.
2. M. Fazal, M. Patel, J. Tye, and Y. Gupta, "The past, present and future role of artificial intelligence in imaging," European Journal of Radiology, vol. 105, pp. 246-250, 2018.
3. Yao, Li, Eric Poblenz, Dmitry Dagunts, Ben Covington, Devon Bernard, and Kevin Lyman. "Learning to diagnose from scratch by exploiting dependencies among labels." arXiv preprint arXiv:1710.10501 (2017).
4. Fei-Fei, Li, Jia Deng, and Kai Li. "ImageNet: Constructing a large-scale image database." Journal of Vision 9, no. 8 (2009): 1037-1037.
5. Wang, Xiaosong, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M. Summers. "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases." In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pp. 3462-3471. IEEE, 2017.
6. Rajpurkar, Pranav, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding et al. "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning." arXiv preprint arXiv:1711.05225 (2017).

7. Jones, Toni Stokes, and Rita C. Richey. "Rapid prototyping methodology in action: A developmental study." Educational Technology Research and Development 48, no. 2 (2000): 63-80.
8. WHO. Standardization of interpretation of chest radiographs for the diagnosis of pneumonia in children. 2001.