Chest X-ray Disease Diagnosis

Lin Shao, Tong Wu, Yahui Ke, Yinuo Liu

Abstract

We develop a web application that can diagnose diseases based on Chest X-ray. Users can upload Chest X-ray images, and our backend algorithms will provide the most possible diseases with probabilities. The backend model is a deep convolutional neural network trained based on ChestX-ray14 data set. The web application will use Django web framework which is hosted in AWS.

Introduction

Chest X-rays is the most popular method to diagnose related diseases, and early correct diagnosis and treatment is critical to save patients' lives and improve health care system efficiency [1]. However, accurate detections highly depend on the radiologist's expertise, which requires long time training and practicing. With the advancements in deep learning over the past decade, we can train computers to detect various objects or classify different groups with high accuracy. In this project, we implement four CNN architectures - CheXNet [2] and VGG16 to detect 14 diseases from the ChestX-ray14 dataset. We build a web application which applies the best model and hyper parameters at backend and provides potential user a user-friendly tool.

Literature Review

Recently, multiple deep neural networks [2-4, 7, 8] have been proposed to detect pathologic patterns from chest x-rays since a large-scale data set [3] from National Institutes of Health Clinical Center became available. Wang et al. [3] applied a unified Deep Convolutional Neural Network (DCNN) based on four pre-trained models, AlexNet, GoogleNet, VGGNet-16 and ResNet-50. This work is a performance benchmark and triggered several other groups to study the application of CNNS in disease detection. Yao et al. [4] utilized a combination of CNNs and Long Short-Term Memory Networks (LSTM) to predict the 14 thoracic diseases and explore the interdependencies among them. Rajpurkar et al. [2] presented a modified Densely Connected Convolutional Networks (DenseNet) to classify the 14 diseases. The combination of dense connections [5] and batch normalizations [6] raised the Area Under Curve (AUC) scores after fine-tuning. Li et al. [7] used a pre-trained residual neural network (ResNet) to extract features and deployed a CNN to produce a disease probability map. Guendel et al. [8] proposed a location aware Dense Networks (DNetLoc), which achieved better AUC scores on multilabel classification and pathology location.

Dataset

In this study, we use ChestX-ray dataset, which was extracted from the clinical PACS database at National Institutes of Health Clinical Center by Wang and et al [3]. It includes 112,120 frontal-view X-ray images of 30,805 patients with 14 disease image labels. The resolution is 1024×1024 for all images. This dataset is already split into training/validation and testing. We further split the training/validation by 95.5%/4.5% based on patient ID rather than image

indexes. The reason is we need make sure the X-Ray images of the same patient do not cross dataset. For typical CNN classification, image augmentation is necessary to cover the variations in actual prediction. However, Chest X-ray images are quite similar, we do not expect to see different rotation, lighting condition, or noises. Based on Rajpurkar et al. [2], we only do random horizontal flip for training. The input for our model is RGB image, and outputs are 14 class with probabilities.

CNN Training

We use Keras with TensorFlow as backend for our CNN model training. We choose AWS EC2 with NVIDIA Tesla V100 because of the big data size and network parameters. A demo test showed one epoch would cost 30 minutes on AWS. Due to the high price of AWS ($3.06/hour), we randomly selected 25% images from training/validation set, and keep the testing set the same for results comparison. In Keras CNN model development, we consider data generator, model architecture and callbacks as the critical components.

1. Data generator: In deep learning, it's usually difficult to load all data to the memory because of the data size. Besides, some computing intensive pre-processing is also not possible on big dataset. Therefore, we used a data generator to load and preprocess image data for each batch at runtime. In this project, the data generator loads a batch of 1024×1024 images in, then process with 3 steps. First, resize to 224×224; second, random horizontal flip; last, standardize based on ImageNet.
2. Model architecture: Model architecture is relatively simple in Keras as it already has all required models built-in. We added a fully connected layer with sigmoid activation at the end of the model.
3. Callbacks: In Keras, we utilize callbacks to calculate statistics and control hyperparameters during training. We calculated AUC, updated learning rate, saved logs and models and stopped training in callbacks,

Using 25% data, we evaluated DenseNet, VGG16, VGG19 and ResNet-50. For each model architecture, we selected the hyper parameters, including loss function, batch size and learning rate as follows.

1. Loss function: because this is a 14 classes classification problem, we chose 'categorical_crossentropy' as loss function.
2. Batch size: batch size means the number of images used in one forward/backward pass. Since the training dataset is huge for memory, we choose mini-batch gradient for the training. We tested batch size 16, 32 and 64. The results shows 64 is too big for GPU memory. There is no significant difference between 16 and 32. We selected 32 batch size for all models.
3. Learning rate: learning rate means how quickly the weights change. Big learning rate may not converge, while small learning rate can be slow. We tested various learning rates with Adam optimizer. Based on the results, we selected 0.001 for ResNet-50 and DenseNet, 0.0001 for VGG16 and VGG19. In callbacks, we reduced the learning rate by a factor 0.1 if no validation loss improvement for 1 epoch. These parameters will be further tuned in the final delivery.

To match results in literatures, we used average AUC to select the best model, which is different from validation loss from the project draft. We stopped training when average AUC did not improve for two epochs.

Preliminary Training Results

We compared the test set AUC of each architecture with Rajpurkar et al. [2]. The results are shown in Table 1. As we can see, all four models perform significantly worse than the paper benchmark. This is within our expectation as we only used 25% training data. DenseNet and VGG16 performed better than the rest, we selected these two for next step with 100% data training.

Table 1. Performance Comparison (AUC)

AUC	Dense Net	VGG16	VGG19	ResNet50	Rajpurkar et al.
Atelectasis	0.7319	0.7478	0.7433	0.7269	0.8094
Cardiomegaly	0.8378	0.8776	0.8085	0.8400	0.9248
Effusion	0.8469	0.8574	0.8498	0.8526	0.8638
Infiltration	0.7269	0.7063	0.7043	0.7048	0.7345
Mass	0.8387	0.7915	0.7696	0.8041	0.8676
Nodule	0.7150	0.6782	0.6969	0.6872	0.7802
Pneumonia	0.7798	0.7825	0.7801	0.7129	0.7680
Pneumothorax	0.7822	0.7701	0.7694	0.7083	0.8887
Consolidation	0.7358	0.7539	0.7340	0.7342	0.7901
Edema	0.8395	0.8333	0.8332	0.8275	0.8878
Emphysema	0.8339	0.8201	0.7956	0.7683	0.9371
Fibrosis	0.6039	0.6333	0.6800	0.6413	0.8047
Pleural Thickening	0.7839	0.7699	0.7556	0.7352	0.8062
Hernia	0.9574	0.9281	0.9517	0.9334	0.9164
Mean AUC	0.7867	0.7821	0.7766	0.7626	0.8414

The training curves for DenseNet and VGG16 are shown in Figure 1 and Figure 2. These two models have similar converge speed and validation performance. VGG16 has smaller training loss. VGG16 may overfits the training data comparing with DenseNet. This is possible as VGG16 has almost 100% more trainable parameters than DenseNet. We will check this again in our full data set training later.

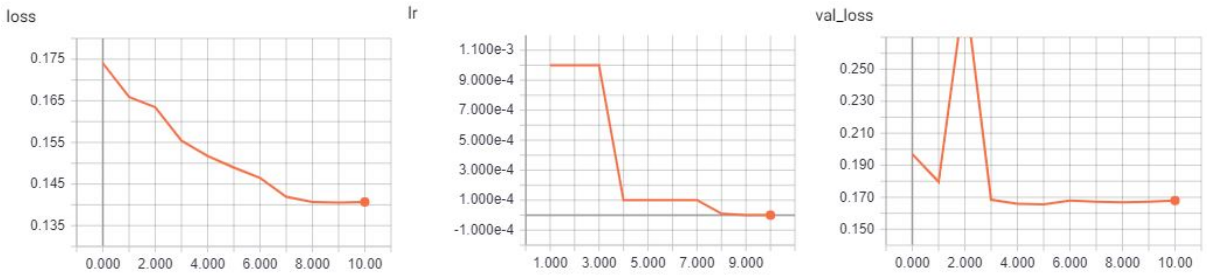

Figure 1. DenseNet 25% Data Training Curves

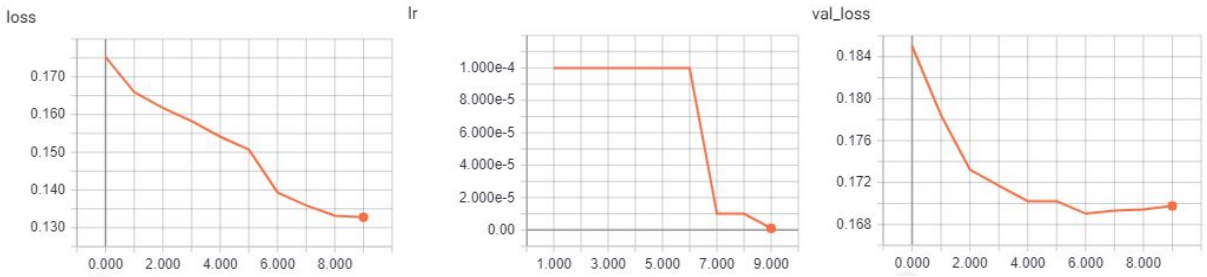

Figure 2. VGG16 25% Data Training Curves

Software Design

The final software product will include a web application, which has an image uploading page, prediction display page, and a prediction images list page.

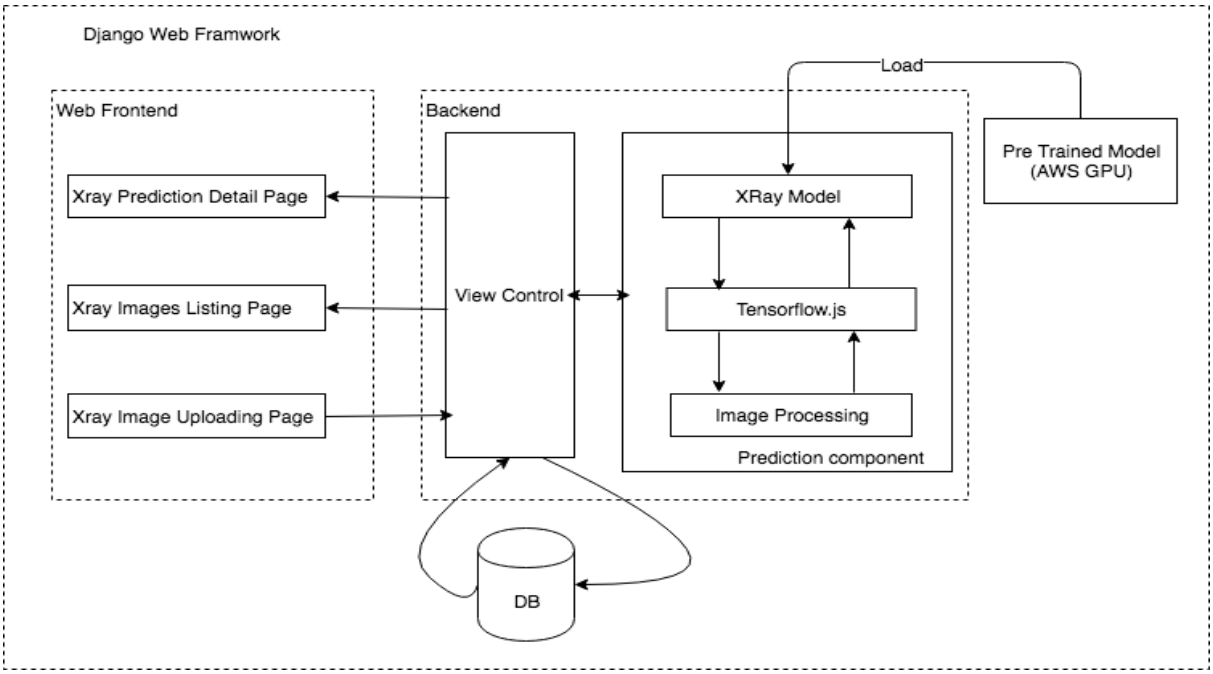

Figure 3. System Component Overview

The image uploading page allows the user to upload an image of X-ray and then the image will be processed in the backend using the pre-trained model. The prediction/classification result will be saved into database and also will be displayed in the prediction display page. We also provide a prediction listing page which will present the latest prediction X-Ray images together with the prediction result so the user can compare the different X-Ray images with the result. The overview of the system is shown in Figure 3.

In the web backend, we have the prediction component which will process the X-Ray images and feed the images into the pre-trained model. The latest Tensflow.js library will be used as a core part of prediction component. The library provides the necessary APIs to process the images into tensors and run pre-trained models in reference mode in JavaScript side.

The final web application will be host in AWS, we will provide an admin account to manage the websites. And if possible, the user uploading images can be also be downloaded and used as training data to improve the model performance.

Conclusion

In this project, we will build a Chest X-ray disease diagnosis web application based on ChestX-ray14 dataset. This application utilizes deep learning to give health care system users an easy to use tool in disease detection.

References

1. Mollura, Daniel J., Ezana M. Azene, Anna Starikovsky, Aduke Thelwell, Sarah Iosifescu, Cary Kimble, Ann Polin et al. "White paper report of the RAD-AID Conference on International Radiology for Developing Countries: identifying challenges, opportunities, and strategies for imaging services in the developing world." *Journal of the American College of Radiology* 7, no. 7 (2010): 495-500.
2. 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).
3. 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.
4. 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).
5. Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely connected convolutional networks." In *CVPR*, vol. 1, no. 2, p. 3. 2017.
6. Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).
7. Li, Zhe, Chong Wang, Mei Han, Yuan Xue, Wei Wei, Li-Jia Li, and F. Li. "Thoracic disease identification and localization with limited supervision." *arXiv preprint arXiv:1711.06373* (2017).
8. Guendel, Sebastian, Sasa Grbic, Bogdan Georgescu, Kevin Zhou, Ludwig Ritschl, Andreas Meier, and Dorin Comaniciu. "Learning to recognize abnormalities in chest x-rays with location-aware dense networks." *arXiv preprint arXiv:1803.04565* (2018).