Convolutional Neural Networks in Radiology

CS230-Fall 2018 Course Project

Max Norris | mpnorris@stanford.edu

Abstract

In this paper I describe how I have used Deep Learning to classify chest X-Ray images based on whether or not the X-Ray belongs to a patient with pneumonia. Pneumonia can cause coughs, fevers, shortness of breath, chills, shaking, fatigue, sweating, and muscle pain, so it is important to have a fast, accurate method for detecting such a disease before symptoms worsen. The goal of using CNNs in Radiology is to help Radiologists more accurately diagnose patients, to reduce fatigue-based errors, and to help patients who lack access to Radiologists diagnose themselves using their chest X-Ray images. I used partial transfer learning from VGG16 and a 24-layer CNN. I split my data (all of which were labeled) into a training set (N normal = 1341, N pneumonia = 3875), a cross validation set (N normal = 8, N pneumonia = 8), and a test set (N normal = 234, N pneumonia = 390). Pneumonia is not easy to see with the naked eye—when interpreting an X-Ray, a professional Radiologist will look for white spots in the lungs (called infiltrates) that identify an infection. Nevertheless my best model achieved 99% accuracy and an F1 Score of 0.99 on its training data and 84% accuracy and an F1 Score of 0.86 on test data.

1 Introduction

The past two summers I have worked in the healthcare industry, and I am hoping to work as a Data Scientist in the healthcare industry after earning my Master's here at Stanford, so for my CS 230 Deep Learning Final Project, I have chosen a project relating to healthcare. I am performing image classification on chest X-Rays to detect and diagnose pneumonia. It has been talked about in the news recently that Deep Learning and automatic image classification may one day be an aid for (or largely replace) Radiologists. From hearing about this in the news, I decided to do my course project on this subject coincidentally the week before Stanford PhD candidate, Pranav Rajpurkar, presented the research he's done with Professor Andrew Ng on this very subject to our class. The goal of using CNNs in Radiology is to help Radiologists more accurately diagnose patients, to reduce fatigue-based errors, and to help patients who lack access to Radiologists diagnose themselves using their chest X-Ray images. For my project I check specifically for pneumonia, which can cause coughs, fevers, shortness of breath, chills, shaking, fatigue, sweating, and muscle pain, so it is important to have a fast, accurate method for detecting such a disease before symptoms worsen. The input to my algorithm was a set of 5856 infant X-Ray images from Guangzhou Women and Children's Medical Center in China. I split my images (all of which were labeled with the ground truth—pneumonia vs. normal) into a training set (N normal = 1341, N pneumonia = 3875), a cross validation set (N normal = 8, N pneumonia = 8), and a test set (N normal = 234, N pneumonia = 390). The image data varied in dimension and some images were in color while others were in grayscale, so I converted all images to 244x244x3—the standard input for the VGG16 CNN. I use a 24-layer CNN, with the first four convolutional layers' weights set to the pre-trained weights of the VGG16 CNN, to predict whether or not each image belongs to a patient with pneumonia.

2 Related Work

The first paper I read relating to this subject came from a 2017 study published by the NIH, which analyzed 108,948 frontal-view X-Ray images from 32,717 patients. At my project proposal, my project mentor, Cristian Bartolomé Aramburu, told me that CS 230 Professor Andrew Ng and guest lecturer Pranav Rajpurkar coincidentally work on research on this very subject. After reading their 2018 paper, I found that they too use a large dataset with 112,120 frontal-view X-Ray images from 30,805 patients. Both research groups output an image that pinpoints the location on the X-Ray that causes the model to make its prediction. The 2017 study outputs bounding boxes around the corresponding health risk, and the 2018 outputs a heatmap, pinpointing the location(s) on the X-Ray that have a low, medium, or high probabilities of suggesting a health risk. Having a very large dataset and pinpointing these locations are great strengths of these two models: large datasets help the model learn how to detect many different diseases (8 in the case of the NIH paper and 14 in the case of the Stanford paper) and heatmaps offer insight into how the models are making their decisions and they can help Radiologists more quickly find the troublesome area(s) in X-Rays. The Stanford group's research is state of-the-art, as their model performs on part with top Radiologists. Neither research group published their code online however. Instead, I found and studied meticulously two open source code repositories, one on GitHub, published by Ayush Singh, and one on Kaggle, published by Aakash Nain. These projects are more simplistic, as neither outputs an image that pinpoints the location of the health risk in the X-Ray and both only perform binary classification. This was more manageable for me as a one-person team (and I'm a Statistics student, not a CS student, so setting up AWS and debugging their code was certainly difficult enough for me). Both projects used pre-trained VGG16 weights and batch normalization. The Kaggle model performed better, in part because it used a much deeper CNN, so I chose to base my code of the deeper, more successful CNN.

3 Dataset and Features

The input to my algorithm was a set of 5856 infant X-Ray images from Guangzhou Women and Children's Medical Center in China. I split my images (all of which were labeled with the ground truth—pneumonia vs. normal) into a training set (N normal = 1341, N pneumonia = 3875), a cross validation set (N normal = 8, N pneumonia = 8), and a test set (N normal = 234, N pneumonia = 390). A difficulty faced was class imbalance—only 27% of the images in my dataset belonged to patients without pneumonia. In my python script (link in Contributions section) I download the library "imgaug" for image augmentation to artificially generated more non-pneumonia images. I took my original 1341 training images of normal patients and performed horizontal flips, rotations, and randomly changed the brightness of the X-Rays. The input image data I downloaded varied slightly in their dimensions, and some images were in color while others were in grayscale, so I converted all images to 244x244x3—the standard input for the VGG16 CNN.

4 Methods

I used a deep convolutional neural network to classify my X-Ray images. In my CNN I used convolutional layers, 2x2 max pooling layers, batch normalization, dense layers, dropout layers, and I end with a softmax layer for binary prediction. During forward propagation, each filter in a convolutional layer is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a two-dimensional activation map of that filter. The 2x2 max pooling layers partition the previous layer into a set of 2x2 squares, and for each square, they output the maximum value. Hence max pooling is a form of non-linear down-sampling. I perform batch normalization by subtracting the batch mean and dividing by the batch standard deviation and then, for each activation, multiplying back by the activation's optimal standard deviation parameter, gamma, and adding back the activation's optimal mean parameter, beta. Batch normalization ensures that the activations are all on the same scale, and this speeds up learning, increases the stability of the CNN, and has a slight regularization effect. After the fourth block in my CNN, all of which end with a max pooling layer, I flatten the three-dimensional matrix into a one-dimensional vector. With a one-dimensional vector I am able to add a "dense" layer, which is simply a fully connected layer in the typical neural network. I add two dropout layers to ensure that my fully connected layers learn robust, redundant representations, in hopes that this will prevent overfitting. Lastly, I use a softmax layer, which in my binary prediction case is the logistic function, to output the probability that the image comes from a patient with pneumonia. I used a 24-layer CNN instead of the 16-layer VGG16 CNN to find more subtle details in the image like the hard-to-spot infiltrates in the X-Rays, and I update my parameters according to Adaptive Moment Estimation (Adam). Adam is an optimization method that combines the advantages of Adaptive Gradient Algorithm (AdaGrad), which maintains a per-parameter learning rate that improves performance on problems with sparse gradients, and Root Mean Square Propagation (RMSProp), which also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight. Rather than using random initialization for my first four convolutional layers, which capture general details like blobs, patches, edges, etcetera, I found much more success through loading pre-trained weights from VGG16 (built by Visual Geometry Group) and fine tuning them. I use binary cross entropy as my loss function, and I experimented with different weight initializations, learning rates, and batch sizes.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Adam

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Note : default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ε

$$BCE = -\frac{1}{N} \sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1-y_i) \cdot log(1-\hat{y}_i)$$

5 Experiments/Results/Discussion

I trained each of my six models for six epochs. As one can see in my GitHub X-Ray repository where I include a photograph of my best model converging in Terminal on my laptop, my model's training accuracy had become 99% after six epochs. In hindsight, maybe I could have obtained better test results had I trained for more epochs but, as one can see in my table below, I definitely overfit my model to the training data, and training for more epochs would cause the model to overfit even more. (Another reason I only trained for six epochs was because although I was using AWS p2.xlarge and p3.2xlarge instances, I don't think I was connecting to their GPU, only their CPU, so my model took hours to train.) I also could have prevented overfitting by only fine tuning a subset of my hidden layers; for example, I could have kept the first four VGG16 pre-trained layers constant, while fine-tuning the other layers. Four of my six models worked horrendously. They got stuck in a local minimum, and simply predicted "pneumonia" for every patient. Since there were more pneumonia images than normal images in my training and test sets, these four models had train and test accuracies that looked decent (0.74 and 0.63, respectively), but these models were useless—effectively their error rates would have been 50% if there were as many patients that were healthy as patients with pneumonia. The model only became useful if the

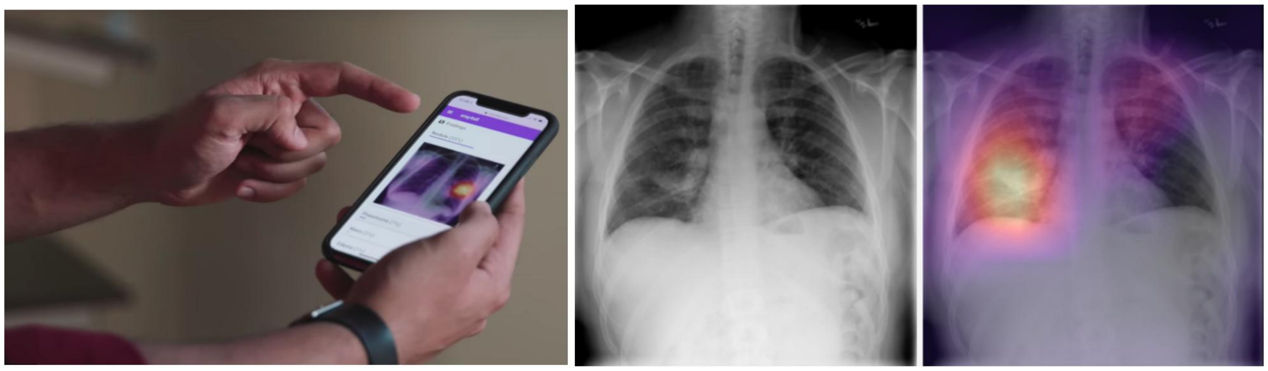
Weight Initializations	Batch Size	Learning Rate	Effective Error Train	F1 Score Train	Effective Error Test	F1 Score Test
Random	16	1x10^-3	0.50	0.85	0.50	0.77
Random	16	1x10^-4	0.50	0.85	0.50	0.77
VGG16	16	1x10^-3	0.50	0.85	0.50	0.77
VGG16	16	1x10^-4	0.01	0.99	0.16	0.86
VGG16	64	1x10^-3	0.50	0.85	0.50	0.77
VGG16	64	1x10^-4	0.01	0.99	0.16	0.86

learning rate was low and the weights of the first four convolutional layers were initialized to the VGG16 pre-trained weights. After I obtained promising results using a learning rate of 1x10^-4, I thought that my model might converge faster with a higher learning rate like 1x10^-3, but instead it looks like this learning rate was too large, and the optimization method probably skipped right across the global minimum. I also tested out different batch sizes—from the Coursera videos I knew that batch sizes between 16 and 128 work well, so I tested out batch sizes of 16 and 64, and obtained the exact same results either way. Below, one can see the confusion matrix from my two top models, which achieved 84% accuracy, 0.90 precision, 0.83 recall, and an F1 Score of 0.86 on the test set. My model is sophisticated enough to aid Radiologists somewhat, for example a Radiologists could double-check their diagnosis and/or reduce fatigue-based errors with my model, but in practice I would still recommend Stanford's CheXNeXt team's more sophisticated algorithm over mine.

	Normal	Pneumonia	Total
Normal	197	37	234
Pneumonia	65	325	390
Total	262	362	624

6 Conclusion/Future Work

The CheXNeXt team at Stanford, which includes Professor Andrew Ng and CS 230 guest speaker Pranav Rajpurkar, has developed a CNN that classifies 14 different thoracic diseases roughly as well as the top Radiologists. Additionally, using class activation mappings (CAMs), their model outputs a heatmap pinpointing the location in the chest that caused the CNN to make its predictions, as seen in their colorized example below:



Now they are working to deploy a website where anyone in the world who lacks access to a Radiologist an upload their own X-Ray image file to get a fast, free diagnosis. If I had more time, more team members, and/or more computational resources, I would love to try to make my CNN perform as well or better than Stanford's CheXNeXt team's model. To do this I would have to a) have a more sophisticated model b) have labeled image data on 14 diseases instead of only one c) learn how to write the Python code to output the heatmap that pinpoints the location in the chest that caused the algorithm to make its prediction. I spent some time in October checking to see if the CheXNeXt team's code was open sourced / publically available so that my starting point for this project could be their ending point but I didn't find their code anywhere online. To perform better than their team I would have to gather more data, as the CheXNeXt website explains "the future of this research will depend on obtaining access to more sources of data for training and improving the model, as well as testing it on new populations and diagnoses." An idea I haven't seen published that I think would be interesting would be to have labeled data for training that explains the health outcome of the patient one or five or ten years down the road; currently the "gold standard" that these CNNs are trying to emulate is the top Radiologists and/or a committee of top Radiologists, but sometimes these Radiologists make a mistake. The true "gold standard" shouldn't be what a Radiologist predicts, but instead the future health of each patient.

7 Contributions

Since I am a one-person team, I performed all tasks outlined above. On my GitHub page (link below) I posted my project code, a README.md description of the code, my project poster, and a screenshot of my code running and my model converging in Terminal on my laptop.
https://github.com/maxnorris17/X-Ray/

References

[1] Professor Andrew Ng's research (2018) on X-Ray image classification
[Accessed: October 15, 2018]
https://arxiv.org/pdf/1711.05225.pdf

[2] Older research paper on this subject (2017) by NIH researchers
[Accessed: October 10, 2018]
https://nihcc.app.box.com/v/ChestXray-NIHCC/file/256057377774

[3] GitHub page that I referenced before the project Milestone
[Accessed: November 2, 2018]
https://github.com/ayush1997/Xvision

[4] VGG16 model information
[Accessed: December 2, 2018]
https://s3.amazonaws.com/cadl/models/vgg16.tfmodel

[5] More VGG16 model information
[Accessed: December 2, 2018]
https://www.quora.com/What-is-the-VGG-neural-network

[6] Open source project off which I based my model
[Accessed: December 2, 2018]
https://www.kaggle.com/aakashnain/beating-everything-with-depthwise-convolution

[7] Pneumonia information
[Accessed: December 10, 2018]
https://www.radiologyinfo.org/en/info.cfm?ph=pneumonia

[8] CNN information
[Accessed: December 14, 2018]
https://en.wikipedia.org/wiki/Convolutional_neural_network

[9] Using Keras (Class Coursera page)
[Accessed: November 3, 2018]
https://www.coursera.org/learn/convolutional-neural-networks/home/welcome