Detecting Pneumonia in Chest X-Rays with Supervised Learning

Benjamin Antin[1], Joshua Kravitz[2], and Emil Martayan[3]

[1]*bantin@stanford.edu*
[2]*kravitzj@stanford.edu*
[3]*emilmar@stanford.edu*

I. Introduction

Physicians often use chest X-rays to quickly and cheaply diagnose disease associated with the area. However, it is much more difficult to make clinical diagnoses with chest X-rays than with other imaging modalities such as CT or MRI. With computer-aided diagnosis, physicians can make chest X-ray diagnoses more quickly and accurately.

Pneumonia is often diagnosed with chest X-Rays and kills around 50,000 people each year [1]. With computer-aided diagnosis of pneumonia specifically, physicians can more accurately and efficiently diagnose the disease. In this project, we hope to train a model using the dataset described below to help physicians in making diagnoses of pneumonia in chest X-Rays.

Our problem is thus a binary classification where the inputs are chest X-ray images and the output is one of two classes: pneumonia or non-pneumonia.

II. Data

We use a dataset compiled by the NIH which contains 112,120 chest X-ray images from 30,805 unique patients [5]. The dataset is available from Kaggle [4]. Each image in the dataset is labeled with one or more diagnoses (“Pneumonia”, “Fibrosis”, “Mass”, etc), or “No finding” if the patient was healthy.

These labels were inferred through natural language processing by mining disease classification from the associated radiological reports, and are estimated to be at least 90% accurate. For the sake of this project, we follow past approaches [2] [5] and treat the labels as ground truth for the purpose of classification.

For this project, we focus on binary classification, attempting to classify a particular X-Ray as having pneumonia or not. There is a strong class imbalance in the dataset, with only about 1% of images labeled as having pneumonia. For this reason, we us AUC as our error metric, rather than accuracy.

Images from the NIH dataset are 1024x1024. To begin, we resized each image using an anti-aliasing filter. As explained below, our Logistic Regression baseline uses 32x32 resolution. Our Deep Learning model uses 224x224 resolution. We also standardize the data so that each feature (each pixel) has zero mean and approximately unit variance.

There are multiple images from many of the patients, and in order to ensure that our models do not see data from the same patient across training and test, we separate the data by patient before splitting into training, validation, and test sets.

We begin with a brief exploration of the data using unsupervised techniques. For the sake of data exploration, we explore 500 random samples due to resource constraints.

III. Data Exploration

We begin by plotting the first two principle components for our 500 random samples, and coloring them by class. As shown in Figure 1, the first principle components do not appear to be clustered based on class.

To better visualize the data, we run t-SNE (T-Distributed Stochastic Neighbor Embedding) for 2-dimensions, because it is often effective for visualizing extremely high dimensional data [7]. The two-dimensional visualization is show in Figure 2. Although there are two clusters present, they do not appear to be separable based on class.

Next, we run k-means on the 500 random samples (with $k = 2$) and label each point in the t-SNE visualization, based on its k-means cluster. We see in Figure 3 that the k-means clusters correspond well with the qualitative clusters we found with t-SNE. Looking at a few examples (see Figure 4 for two examples), we see that one cluster corresponds with darker images and the other with brighter images.

Although these results are interesting, we do not find them useful in developing the machine learning models described below. Additionally, we ultimately decide to use deep learning techniques, which can often infer complicated features through training. As such, we feel that any filters using unsupervised methods can be captured by the complexity of our neural network.

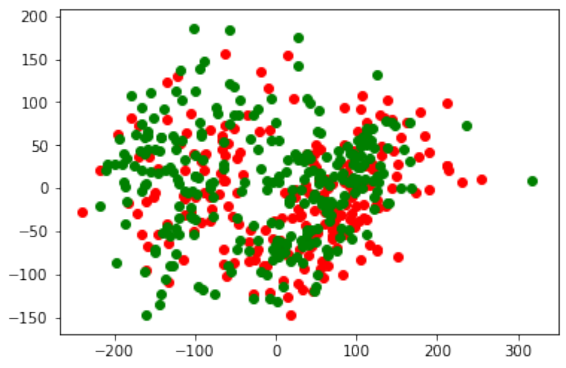
Fig. 1: First two components of PCA run on each sample, stratified by class.

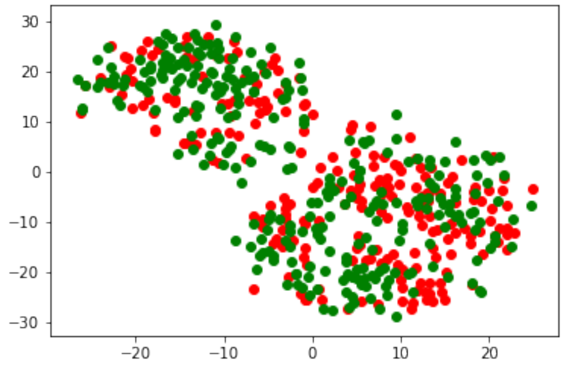
Fig. 2: First two components of running t-SNE on each sample, stratified by class.

IV. Infrastructure

Working with a dataset this large poses its own challenges. In order to satisfy memory and computation requirements, we use a Google Cloud instance with 8vCPUs and 52GB of memory, and store the data on a separate disk.

For the deep learning section of our project, we additionally use an NVIDIA P100 GPU. We implement the baseline using SciKit-Learn, and we use PyTorch for the Deep Learning section of our project. With the addition of the GPU, Logistic Regression actually becomes more memory-intensive than Deep Learning. As we explain below, SciKit-Learn requires that the entire design matrix be stored in memory, whereas our Deep Learning model only loads the pixel values for each mini-batch of images. In effect, this means that we were not able to run logistic regression on 224x224 images, and were limited by available memory.

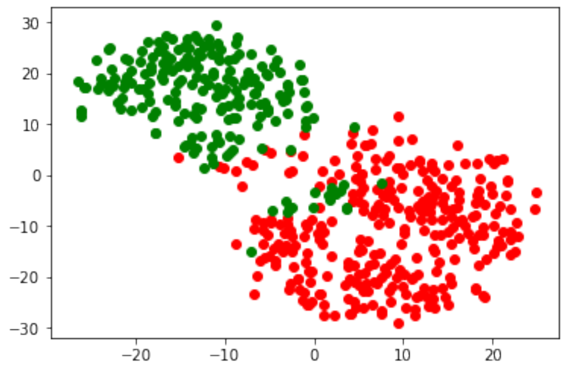
Fig. 3: t-SNE visualization of samples, colored by k-means cluster.

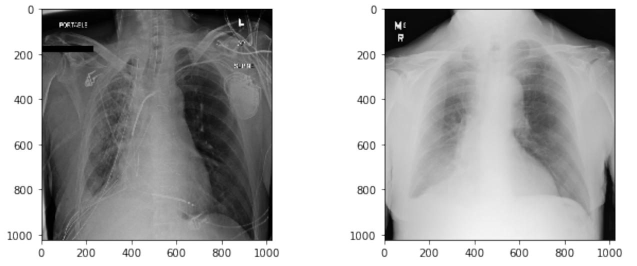
(a) Sample from cluster 1 (b) Sample from cluster 2

Fig. 4: Examples from both clusters in Figure 3

For the logistic regression baseline, we also utilize h5py in order to store the images in HD5 format. We do this in the hope that we could afford to run our baselines on larger images, since h5py doesn't load examples into memory until they are sliced in the python code. Ultimately, this doesn't alleviate memory pressure because SciKit-Learn loads every example into memory regardless.

V. Baseline Model: Logistic Regression

As a baseline, we use a logistic regression model to classify whether or not a given X-ray contains pneumonia. Logistic regression works well as a baseline because it is relatively easy to implement. We treat each pixel as a distinct feature, and add an $\mathcal{L}_2$ regularization term to our loss function. The loss function is given by

$$L = \sum_{i=1}^{m} \log(1 + exp(y^{(i)}\theta^T x^{(i)}) + \lambda||\theta||_2^2$$

To tune the regularization parameter λ, we use a fine range of values for λ, and perform 3-fold cross-validation to obtain average accuracy measures for each different value of λ. We then save the λ value that yields the best model and use it for the rest of our analysis. We find that $\lambda = 1500$ gives the best performance on the 32x32 images.

Because the dataset is so large, we are not able to perform hyperparameter sweeps on the full dataset. Instead, we perform our sweep on a random sample of 5606 images.

Using the value of $\lambda = 1500$ that we obtain from our initial experiments, we then run logistic regression on the entire dataset, splitting into train, val, and test sets.

After training, logistic regression achieves an AUC score of 0.60 on the test set, using 32x32 images (the corresponding ROC plot is shown in Figure 8). As mentioned above, our dataset contains highly skewed classes. Therefore, we use AUC as our primary error metric, rather than accuracy.

We also run logistic regression on larger 128x128 images. Interestingly, AUC evaluated on the test set for larger images is slightly lower, at 0.58. We hypothesize that the lower performance is because our regularization parameter was tuned on smaller images, and larger images would require more regularization because there are many more features. Even with our Google Cloud infrastructure, we are not able to run logistic regression on larger resolutions, because of the larger design matrix and SciKit-Learn's high memory requirements for logistic regression (all of which did not fit into main memory).

From the AUC of 0.60, we conclude that logistic regression does not adequately capture the complexity of our dataset. In particular, it is unlikely that individual pixel intensities are good features for this binary classification problem. Correlations between pixels, as well as higher level features like edges, are likely needed to perform well on this classification problem. For this reason, we next used a Convolutional Neural Network.

VI. Deep Learning

Following the approach of *CheXNet*[2] we use a 121-layer dense Convolutional Neural Network (DenseNet).

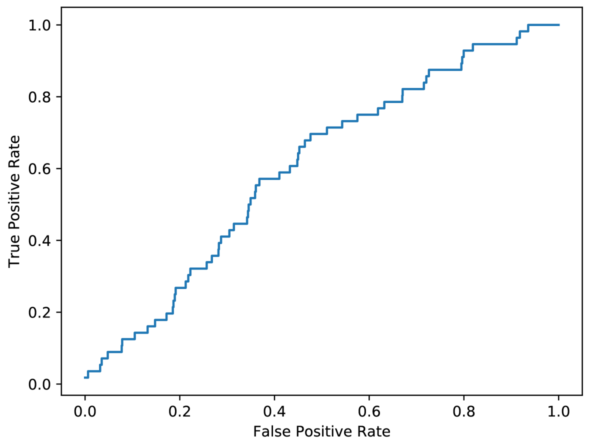

Fig. 5: ROC Curve for Logistic Regression on 32x32 Images. AUC = 0.6037

First proposed by *Huang, Liu, et al*, DenseNets make training very deep networks easier by connecting each layer to every prior layer [3]. To implement our network, we use transfer learning via PyTorch, a deep learning framework for building neural networks in Python with GPU acceleration.

The weights of the network are initialized using the weights from a model trained on the ImageNet dataset. We then replace the last layer with a 2-output Softmax layer. To help the model converge faster, we down-sample the 1028x1028 to size 224x224, which still preserves most of the fine details in the images.

Additionally, we augment our data by randomly flipping the images in the horizontal direction. In other image classification tasks, it is common to augment data with random cropping and rotation. However, because our dataset consists exclusively of X-Ray images which are centered in the field of view, we limited our data augmentation to random horizontal flips.

We use the *Adam*[6] algorithm to train our network, as it is well suited for problems that are large in terms of data and/or parameters. We use standard parameters $(\beta_1 = 0.9, \beta_2 = 0.999)$ and a learning rate of $\alpha = 0.001$, which we anneal by a factor of 10 every time the validation loss plateaus for one epoch. Additionally, we use a batch-size of $B = 16$. To account for the class imbalance we use the following weighted binary cross-entropy loss as our loss function:

$$L = -\frac{1}{B}\sum_{i=1}^{B} w^{(i)} \left(y^{(i)}log(\hat{y}^{(i)}) + (1 - y^{(i)})log(1 - \hat{y}^{(i)}) \right),$$

where

$$w^{(i)} = \begin{cases} \frac{|N|+|P|}{|P|} & if\ y^{(i)} = 1 \\ \frac{|N|+|P|}{|N|} & otherwise, \end{cases}$$

where $|N|$ and $|P|$ refer to the number of negative and positive examples, respectively.

After training, the network achieved an AUC of 0.609, only slightly better than logistic regression. ROC Curves for train and test are shown in Figures 6 and 7. As we discuss in the next section, we hypothesize that the network failed to generalize well to unseen data.

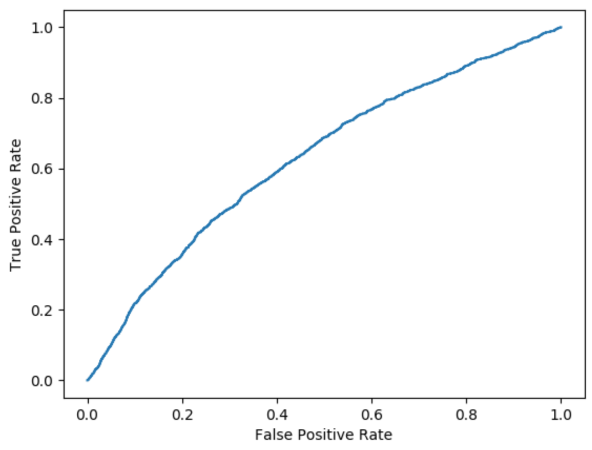

Fig. 6: Train ROC for DenseNet-121. AUC = 0.684.

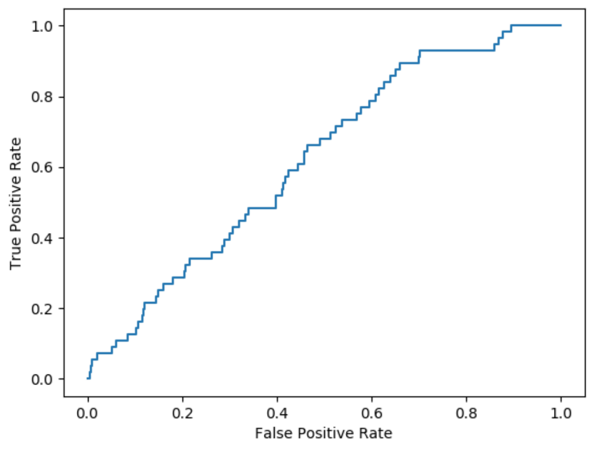

Fig. 7: Test ROC for DenseNet-121. AUC = 0.609.

VII. Discussion and Error Analysis

As expected, Logistic Regression does not adequately capture the complexities of this dataset. Diagnosis of pneumonia from chest X-rays alone is a difficult task that requires knowledge of disease pathology as well as human anatomy [2]. From inspecting the dataset, it is clear that it presents a challenging problem: areas of interest are often obscured by the ribs, and other diseases in the dataset look visually similar to pneumonia.

Using a DenseNet, *CheXNet* documents an AUC of 0.828 on the pneumonia classification task, which is significantly better than our network [2]. We are unable to replicate the results described in the *CheXNet* paper. As shown in Figure 8, the training loss decays smoothly with every training epoch, but validation loss fluctuates seemingly at random. This suggests that the loss function is indeed being optimized, but that the features learned on the training set are not generalizing to the validation set.

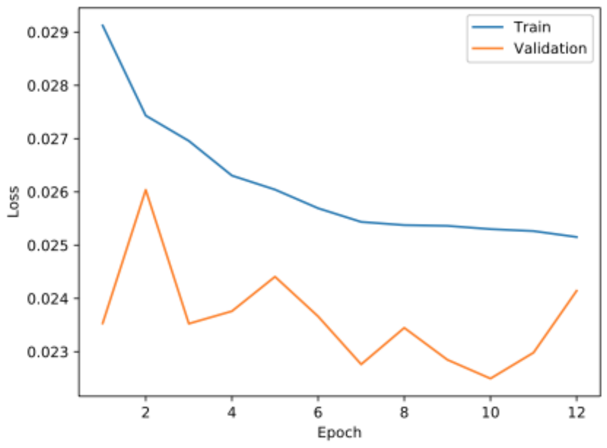

Fig. 8: Training and Validation Loss vs. Epoch for DenseNet-121.

We suspect that this is the case because the number of pneumonia cases in the training set is so much smaller than the number of non-pneumonia cases.

In order to better understand the performance of network, we use t-SNE to visualize the features output from the last convolutional layer. We run t-SNE on the features output from 154 images from each class. As described in [7], t-SNE attempts to maximize the Kull-back Leibler divergence between the reduced dimension components. Thus, if the features output by the last convolutional layer were meaningful for distinguishing between classes, we would expect distinct clustering. However, as shown in Figure 9, we see no distinct clusters when we use t-SNE on the activations of the final convolutional layer. We conclude that the issue is with the features learned by the network, rather than an issue with the final linear layer. We hypothesize that more examples with Pneumonia would help the network learn more salient features.

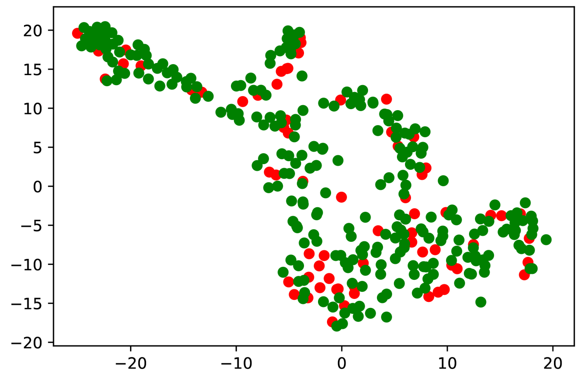

Fig. 9: t-SNE Clustering of Final Layer Activations, colored by class. No distinct clustering is present.

VIII. Future Work

Below are a variety of techniques and ideas one might pursue to expand on the work of this project.

A. *Different Features*

One could try running PCA on the images to decorrelate the data and use these results on a logistic regression model. (In other words, use a subset of principal components as features.)

It may also be worth exploring different image feature extraction techniques to get a better baseline.

B. *Improved Logistic Regression Baseline*

Although we were able to run logistic regression on 32x32 and 128x128 down-sampled images, we were unable to both do a hyperparameter sweep on images larger than 32x32 and use the same image size as our deep learning model, 224x224. With more time, we would implement a Spark cluster on Google Cloud and attempt to find the best regularization value for 224x224 images. We predict that our baseline would improve fairly significantly as a result.

C. *More Error Analysis*

One might explore the DenseNet's activations using Class Activation Maps in order to better understand its behavior on this dataset. This might inform why it achieves such low AUC as compared to similar work.

D. *Bounding Boxes*

A small subset of the dataset has bounding boxes around diseased areas. One could use these bounding boxes to train a CNN to not only classify images with pneumonia, but also identify where in the image the pneumonia is located.

IX. Acknowledgements

We would like to thank Dr. Andrew Ng and Dr. Dan Boneh for exposing us to a wide breadth of machine learning concepts. Furthermore, we would like to thank Mr. Jeremy Irvin for assisting us in understanding relevant topics outside of class time and for helping us implement and debug our neural network.

X. Contributions

Ben worked on preprocessing the images, configuring Google Cloud, and implementing transfer learning in PyTorch.

Joshua worked on data exploration and the logistic regression baseline on the full 112,120 images. He helped to set up the GPU on Google Cloud and assisted with debugging the CNN.

Emil worked on the cross-validated logistic regression model, helped set up the GPU on Google Cloud, and helped to write the code for the DenseNet CNN.

References

[1] https://www.cdc.gov/nchs/fastats/pneumonia.htm

[2] Pranav Rajpurkar, Jeremy Irvin, et al. *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*, https://arxiv.org/pdf/1711.05225.pdf 2017.

[3] Gao Huang, Zhuang Liu, Laurens van der Maaten. *Densely Connected Convolutional Neural Networks* https://arxiv.org/abs/1608.06993

[4] https://www.kaggle.com/nih-chest-xrays/datasets

[5] Wang, et al. *ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases* http://openaccess.thecvf.com/content_cvpr_2017/papers/Wang_ChestX-ray8_Hospital-Scale_Chest_CVPR_2017_paper.pdf

[6] Diederik P. Kingma, Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization* https://arxiv.org/pdf/1412.6980.pdf

[7] Laurens van der Maaten, Geoffrey Hinton *Visualizing Data using t-SNE* 2008 http://www.cs.toronto.edu/~hinton/absps/tsne.pdf