Disease Detection in NIH Chest X-Ray Images

Arsh Modak,[1] Omkar Waghmare,[2] Adithya Chenthilkannan[3]
Northeastern University[1,2,3]
modak.a@northeastern.edu,[1] waghmare.o@northeastern.edu,[2] chenthilkannan.a@northeastern.edu[3]
GitHub Link: https://github.com/arshmodak/NIH_Chest_XRays_Disease_Detection

Abstract

Chest X-ray exams are one of the most frequent and cost-effective medical imaging examinations available. However, clinical diagnosis of a chest X-ray can be challenging and sometimes more difficult than diagnosis via chest CT imaging. By using machine learning and deep learning techniques, we aim to ease the detection and classification of various, specifically 14, lung diseases and help radiologists speed up their diagnosis. This paper proposes and compares the performances between three distinct approaches, one using state-f-the-art (SOTA) CNN architectures and the other two using traditional machine learning models in order to compare performance of SOTA CNN architectures and traditional ML models. Out of these three approaches, one clearly outperforms the others while the other two perform similarly. In conclusion, we were able to successfully classify 14 diseases in the images with an average AUC of 0.7693 using ResNet-152. Furthermore, we were also able to achieve comparable results using CNNs as feature extractors and traditional ML models as our classifiers – achieving an average AUC of 0.75.

Introduction

Approximately 3 million people die of lung diseases each year, making them the third leading causes of deaths worldwide. To detect these diseases in patients, first you need an X-ray and then, to scan and analyze the X-ray, you need an expert radiologist.

Thousands of CXR images are captured in hospitals, making the computer-aided diagnosis (CAD) a very important but challenging task. Since most of these diseases are fatal and require immediate diagnosis, it is crucial for hospitals to have a readily available expert radiologist in house. However, in some cases, this is not possible, which may cause a delayed diagnosis, leading to the death of a patient.

Therefore, automatic analysis of CXR images would effectively assist clinical diagnosis and pathology finding. However, chest X-ray image analysis is a challenging task which suffers from the intrinsically complex relations of different pathologies. In this paper, we conduct experiments with deep learning and machine learning techniques to aid and assist radiologists in-order to accurately diagnose these diseases and speed up their workflow. These experiments include using pre-trained SOTA CNN architectures as classifiers, using SOTA CNN architectures as feature extractors and using these features to train traditional ML models and manually extracting important features from the images to train traditional ML models to compare performance of SOTA CNN architectures and traditional ML models.

Since each image can have multiple labels/diseases, this makes it a multi-label, multi-class classification problem.

Background

1. Manual Feature Extraction

1.1. Local Binary Pattern

LBP is a very effective and popular image processing method used to extract texture features from an image and is a visual descriptor. In its simplest form it works in the following manner: first, we divide the image into cells, then for each pixel in the cell, compare the center pixel with its neighboring pixels (8 pixels), assign 0 to those pixels for which the center pixel value is greater than the pixel itself, and 1 otherwise. Compute histogram for each cell, normalize the histogram, concatenate histograms of all cells to get a feature representation of the image. This histogram can then be flattened out to generate features for image classification tasks.

1.2. Grey Level Co-Occurrence Matrix

GLCM extracts statistical features from pixels by generating a co-occurrence matrix that signifies how frequently pair of pixels that have a specific value and a spatial relationship occur in an image and then extract measures such as homogeneity, contrast, correlation, dissimilarity, energy, etc. from this co-occurrence matrix. It is essentially a way of extracting second order statistical features from an input image. In terms of using GLCM as features for image classification, one of two approached can be used, generating LBP of an image and then extracting statistical measures from it or concatenating LBP features and GLCM measures to obtain a feature vector.

1.3. Discrete Wavelet Transforms

Discrete Wavelet transform is widely used in image classification as an image processing technique. The basic idea of Discrete Wavelet Transform is to provide the time-frequency representation. The 2D-DWT represents an image in terms of a set of shifted and dilated wavelet functions. In simple words, DWT for image processing transforms the input image into Wavelet coefficients that match spatial frequencies of an image. Low/medium frequencies are used to represent image content, high frequencies represent texture features or noise. Using a combination of these wavelets a compact approximation of the input image can be formed. This approximation has low dimensions (compressed form) as compared to the input image which in-turn can be used as a feature vector for image classification tasks.

1.4. Histogram of Gradients

Histogram of Gradient descriptor describes the object appearance and shapes within an image as a distribution of gradients. Essentially an image is divided into cells, for each pixel within these cells, a histogram of gradients descriptor is calculated which encodes the gradient direction and grayscale intensities within that cell, these descriptors are then concatenated and normalized to obtain a feature representation of the image which is invariant to illumination and geometric transformations. This feature representation can then be flattened out and used as a feature vector for image classification tasks. Histogram of Gradients and Local Binary Patterns are complimentary to each other which makes a compilation of these two a great feature for image classification/detection tasks.

2. Machine Learning Models

2.1. Multinomial Logistic Regression

Logistic Regression is a supervised ML algorithm used for classification. It returns the probability of a certain class existing – usually this is binary in nature, i.e., win/loss, pass/fail, 1/0 etc. Logistic Regression uses a sigmoid function which returns the probability. A threshold can be set to classify these logits into classes.

$$Logit(P(x)) = w_1x_1 + w_2x_2 + w_3x_3$$

$$Logit(P(y=1|x)) = \frac{1}{1+e^{-(w^tx)}}$$

Multinomial Logistic Regression is simply an extension of simple logistic regression with a major change where it allows for more than one class. This permits the model to have more than a binary outcome, which is necessary if you want to predict the probabilities in a multi class, multi label problem such as ours which requires us to predict the probability of 14 distinct lung disease labels. This algorithm is widely used in medical fields due to its linearity, simplicity and low computational requirements.

2.2. Naïve Bayes

Naïve Bayes is a supervised ML algorithm which is probabilistic in nature and based on Bayes' theorem. It is called "Naïve" Bayes since it naively assumes that there is strong independence between the features that are used as predictor variables.

In this project, we implement two Naïve Bayes algorithms, namely, Bernoulli Naïve Bayes and Gaussian Naïve Bayes.

Gaussian Naïve Bayes typically assumes that the continuous features follow a normal distribution and Bernoulli Naïve Bayes assumes the feature vectors are binary in nature. Additionally, Bernoulli Naïve Bayes penalizes those features that are not important for predicting the target variable.

2.3. Decision Trees Classifier

Decision Tree is a supervised ML algorithm used for both Classification and Regression problems. It uses a set of rules to make decisions by splitting and branching out until its leaves are completely "pure" unless specified. It uses entropy or Gini index and information gain to decide the splits and eventually classify a datapoint. Tree-based algorithms are highly interpretable and easy to understand once visualized. However, they tend to overfit if not implemented correctly.

2.4. Extra Tree Classifier

Extra Trees Classifier (Extremely Randomized Trees Classifier) is an ensemble learning techniques that utilizes Decision Trees as its base estimator. It creates a "forest" of Decision Trees and outputs a result based on the average of these trees. It is similar to Random Forest with one major change – it does not utilize bootstrapping, instead the entire dataset is passed to each underlying tree.

This technique overcomes the drawback of a single Decision Tree by reducing/controlling overfitting and improves the predictive accuracy and performance. This, however, comes with a cost, i.e., time and computational resources.

3. Deep Learning Models

3.1. Convolutional Neural Networks

A digital image consists of pixels in a grid fashion, a specific image is formed by a combination of RGB color and brightness specified in each pixel.

A Convolutional Neural Network, also known as CNN, is a type of Neural Network that is well suited for image classification tasks such as ours. A CNN is capable of learning abstract representations from the image by utilizing filters or kernels of different sizes. The deeper the network, the more intricate features it is able to learn, these include eyes, lips, designs etc. A CNN consists of convolutional layers which are the feature extractor followed by subsequent linear layers that make up the "classifier" or "regressor".

These CNNs are not limited to convolutional layers and linear layers, layers such as batch normalization and pooling can be used in combination to create a unique architecture fit for tasks at hand.

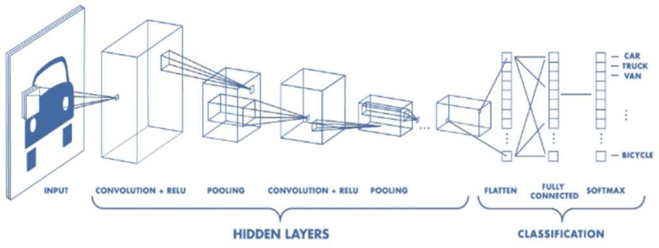


Figure 1: A simple CNN Architecture used for classification.

The convolutional layer is nothing but the dot product of the image pixel values and learnable weights of the layer (similar to weights in simple ML algorithms or ANNs).

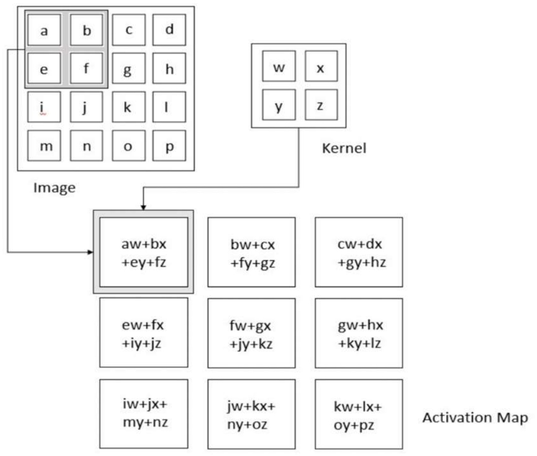


Figure 2: Working of a Kernel on an Image in a Convolutional Layer.

Linear layers are nothing but dense connections between the output of the preceding layer and input of the current layer. Here, each neuron is connected to each other between its immediate successor and predecessor layers.

An activation function such as Sigmoid or ReLU can be on any of the layers in the network. Additionally, Dropout, which switches off a neuron with some probability can be used in linear layers as a regularization technique which reduces overfitting.

Related Work

Advancements in Deep Learning has helped make significant improvements in medical fields. Rajpurkar et al. (2017) use deep learning to attain radiologist level results in detecting Pneumonia in chest X-ray images. They utilized an ImageNet pre-trained model, DenseNet-121, with a modified loss function that was trained on the NIH Chest X-ray dataset to accurately diagnose Pneumonia in chest X-Ray images. To interpret the networks prediction, they also produced heatmaps to visualize the areas of the image that the model predicted to have Pneumonia in, with a certain probability. They were able to beat the previous state of the are model proposed by Wang et al. (2017) who used ResNet-50 for weakly-supervised object localization, using various multi-labeled CNN losses and pooling strategies. Wang et al. (2017) also released "machine-human annotated" comprehensive NIH chest X-ray dataset, but was limited to classifying eight common thoracic pathologies as apposed to 14 in Rajpurkar et al. (2017). Very soon after Wang et al. (2017) was published, Yao et al. (2017) managed to leverage dependencies among target labels and achieve state of the art results in classifying all 14 pathologies available in the NIH chest X-ray dataset.

After extensive research, we noticed that none of these works tired and tested how traditional machine learning models would perform. We saw this as a potential opportunity. If we managed to achieve comparable results to these works, we would save a considerable amount of time and resources by using traditional machine learning models instead of large and computationally expensive ImageNet pretrained models to achieve similar results. Furthermore, incorporating traditional ML models into already existing workflows for radiologists is a much easier and less complex task.

Rajpurkar et al. (2017) and Wang et al. (2017) perform disease localization using attention layers and extracting feature/activation maps of the image with the disease. Another potentially useful method that could be implemented instead of using attention layers is "object-detection". CNN architectures such as various versions of YOLO (You Only Look Once) and U-Net can be used to detect exactly where a specific disease is on the image. However, this did not align with our use-case. Moreover, the bounding boxes provided to us were limited to just 30% of the images which would yield very poor and bias results. A major negative impact on this implementation with limited data is the large class imbalance in the dataset. Hence, we did not choose to adopt this path and stuck to training larger SOTA CNNs and traditional machine learning models.

Hardware and Software Requirements

1. Hardware Requirements

- 16 GB DDR4 RAM (32 GB recommended)
- NVIDIA Tesla P100 PCIE 16GB or higher
- Intel i7 8th Gen or higher

2. Software Requirements

- Python 3.7+
- PyTorch 1.10
- CUDA 10.2
- NumPy, Pandas
- Sci-kit Learn, Sci-kit Image
- Matplotlib, Seaborn
- OpenCV, Pillow

Project Description

1. Dataset

The dataset was provided by the NIH (National Institute of Health) with an aim to teach computers how to detect and diagnose diseases. It is the largest publicly available chest X-ray dataset. It comprises of more than 112k images of over 30k unique patients. NIH also provides a metadata file which consists the filename, disease labels, bounding boxes and demographics of the patients.

All images are single band grayscale images of the size 1024x1024. There are 14 distinct disease labels, namely, Atelectasis, Cardiomegaly, Consolidation, Edema, Effusion, Emphysema, Fibrosis, Hernia, Infiltration, Mass, Nodule, Pleural Thickening, Pneumonia and Pneumothorax. There is an additional label called "No Finding" which signifies images with no detected diseases. Each image can be associated with multiple labels.

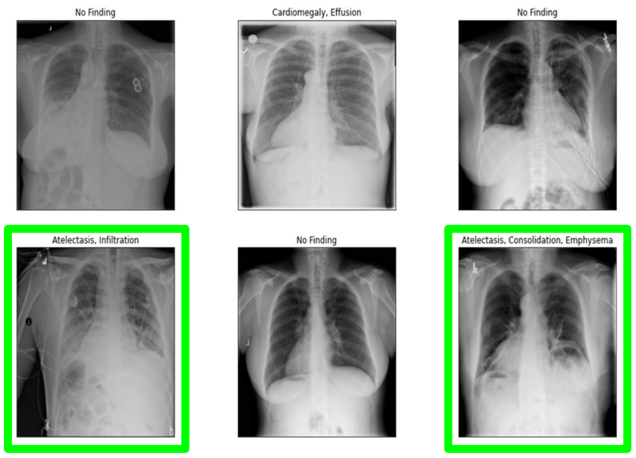
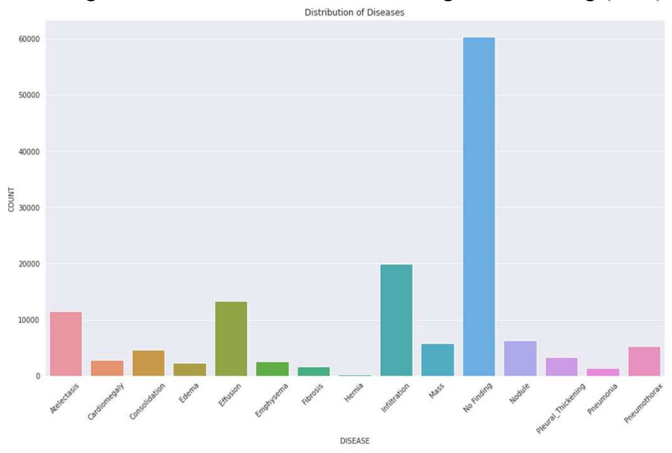

No Finding
Cardiomegaly, Effusion
No Finding
Atelectasis, Infiltration
No Finding
Atelectasis, Consolidation, Emphysema

Figure 3: Sample of raw images from the NIH Dataset. The highlighted images represent the multi-label classes, i.e., a single image having multiple diseases.

The disease labels are quite skewed due to the large imbalance in the classes. Figure 3 shows a distribution of diseases in the dataset. As we can see from Figure 3, the number of images with "No Finding" as the label dominates, consisting of over 50% or 60k+ images within the dataset. We also notice Hernia has the least number of samples in the data. This makes our multi-label, multi-class classification task more difficult.

Figure 4: Shows the distribution of images. No Finding (Blue)



2. Methods

We divided our projects in four phases. The first phase was completely dedicated to research and pre-processing of the images and metadata provided by NIH (National Institute of Health). The other three phases included designing and constructing three distinct pipelines for our disease detection/classification task.

The first pipeline we created, as depicted in Figure 5 below, was to train SOTA CNN architectures such as DenseNet-121, DenseNet-161, ResNet-152 and VGG-16.

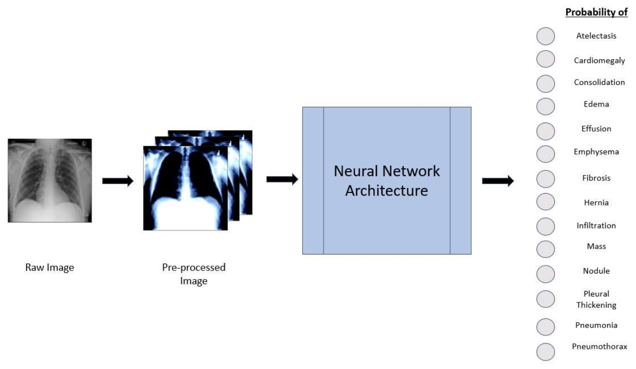


Figure 5: Pipeline to Train SOTA CNN Architectures.

The second pipeline we created, as depicted in Figure 6, was to use the best performing SOTA CNN architecture and use it as a feature extractor for our images. These extracted features were then used as inputs for our traditional machine learning algorithms such as Logistic Regression, Bernoulli Naïve Bayes, Gaussian Naïve Bayes, Decision Trees and Extra Trees Classifier. This was done to see how well traditional ML models perform in comparison to SOTA CNN architectures when fed with the same features.

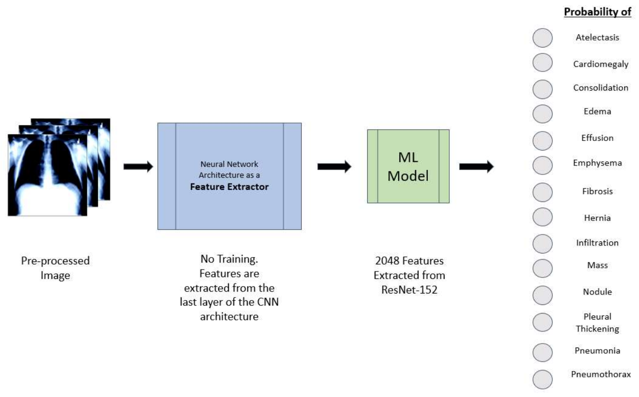


Figure 6: Pipeline to Extract Features using SOTA CNNs and Train Traditional ML models.

For the first two pipelines, we created a custom dataset class and data-loader which would read the metadata, open the image and convert it from a single-band image to a three-band image, apply all transformations and return the image and its corresponding disease labels.

Finally, we created a third pipeline, as depicted in Figure 7, which did not use CNNs at all, instead, we used manual feature extraction algorithms/techniques such as Local Binary Pattern, Gray Level Co-occurrence Matrix, Discrete Wavelet Transform and Histogram of Gradients to extract features from all the images in the dataset. These features were then fed to traditional ML models. The goal of creating this pipeline was to try and implement a feature extractor that did not require the use and knowledge of CNNs and to see how traditional Machine Learning models perform compared to State of The Art CNN architectures.

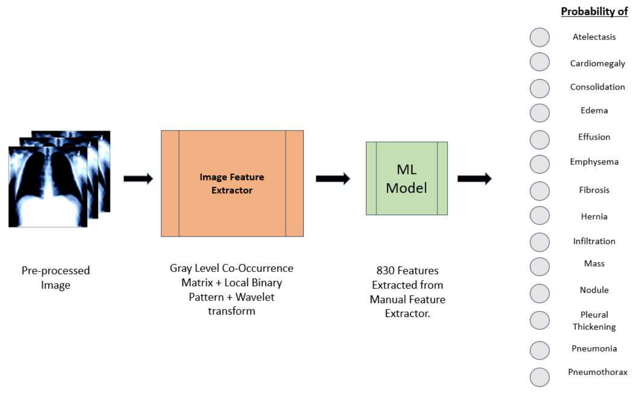


Figure 7: Pipeline to Extract Features Manually and Train Traditional ML models.

Experiments

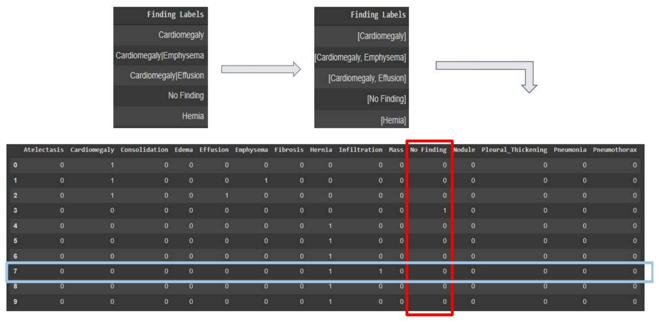
1. Metadata

Due to high class imbalance where more than 50% of the labels belonged to class No Finding, we chose to address this problem first. We started off by normalizing the class distributions, i.e., reducing No Finding images and increasing Hernia and other classes with low number of images. This was ideal as number of images considerably reduced and took less computational resources. We then one-hot encoded all labels, including No Finding, with 1 signifying presence of the disease and 0 signifying absence of the disease. This resulted in performance of SOTA CNN architectures. One obvious reasoning for this was, although the data is uniformly sampled, it was getting confused by No Findings.

Next, we removed all No Finding images and its corresponding metadata, uniformly sampled the distribution, one hot encoded the remaining labels in the manner specified above and ran our pretrained CNN architectures on it. This too resulted in poor model performance. We reasoned that the data that we provide was very less for these models to learn. We then decided to not sample our data and one hot encode all labels including No Finding and train our CNN architectures. This also resulted in poor model performance.

Finally, after closely analyzing the model results, we found out that passing No Finding in the metadata with label

1 was resulting our architectures to overfit on it and classify all other diseases as No Finding. Hence, we chose to dummy encode our data, with a vector of 14 dimensions, where all 0's indicated that the image belonged to the No Finding Class. This configuration yielded us the best results so far.



Finding Labels
Cardiomegaly
Cardiomegaly|Emphysema
Cardiomegaly|Effusion
No Finding
Hernia

Finding Labels
[Cardiomegaly]
[Cardiomegaly, Emphysema]
[Cardiomegaly, Effusion]
[No Finding]
[Hernia]

	Atelectasis	Cardiomegaly	Consolidation	Edema	Effusion	Emphysema	Fibrosis	Hernia	Infiltration	Mass	No Finding	Nodule	Pleural_Thickening	Pneumonia	Pneumothorax
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 8: This table shows the preprocessing of the labels. Raw data (top left), Intermediately processed data (top center) and One-Hot encoded labels (bottom). The blue box depicts an image having more than one disease (Hernia and Infiltration). The red box depicts the "No Findings" column, which we decide to discard as discussed above. Hence leading to the "dummy" encoding of the labels.

2. Images

For images, we tried various sizes and data augmentation/transformation techniques. The very first models we trained with our best metadata configuration explained above was with images of size 512 x 512 x 3 and no transformations. This was done, as we thought that providing images of bigger sizes would help our models learn better. But, on the contrary, we found out that these pre-trained models did not work very well. This could be due to two main reasons; one the models are designed to perform efficiently on 224 x 224 x 3 image sizes and hence aren't working well on larger images, two the models are working well on larger images, but are overfitting. Hence to test our assumptions we tried running our models on 224 x 224 x 3 with no data transformations/augmentations and found that the models were performing better than the previous configuration. However, they weren't any closer to the performances mentioned in Rajpurkar et al. (2017) or Yao et al. (2017).

One of the reasons for this could be, that these works implemented a custom loss function and hence resulted in better model performance, the other was we did not apply image transformations and augmentation techniques.

We then tried running our models on 224 x 224 x 3 images with normalization and random horizontal flips and found that this configuration (Figure 9) gave us the best results. We finalized this configuration and moved on to experimenting with various architectures and fine-tuning them.

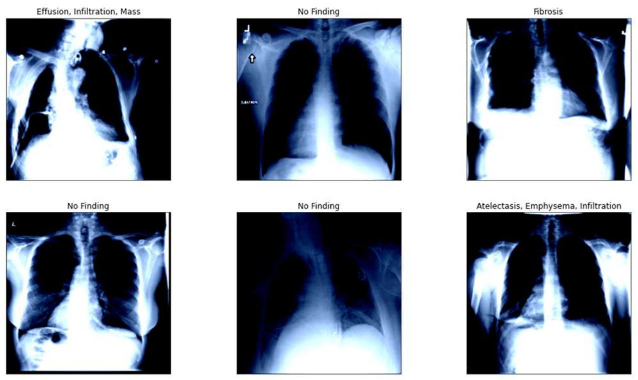

Effusion, Infiltration, Mass
No Finding
Fibrosis
No Finding
No Finding
Atelectasis, Emphysema, Infiltration

Figure 9: Sample of the preprocessed images. Resized to 224x224 and normalized.

3. CNN Architectures

Since Rajpurkar et al. (2017) and Yao et al. (2017) were able to achieve success in their work by using DenseNet-121 and ResNet-50 respectively, we chose similar but bigger architectures to work with. Essentially, we chose three unique SOTA CNN architectures.

First, VGG-16 (Figure 10), which is a general CNN architecture utilizing multiple convolutional layers followed by max-pooling layers.

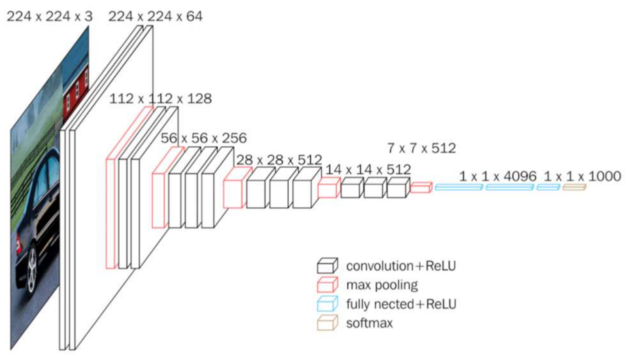


Figure 10: VGG-16 Architecture. Simonyan et al. (2014).

Second, ResNet-152 (Figure 12), which is three times deeper than ResNet-50. What's unique about this architecture is that it utilizes "skip-connections" or "identity-mapping" (Figure 11). This allows the network to learn richer features than the VGG-16 since it not only learns from features extracted in the current layer, but also a combination of the raw input from the previous layer.

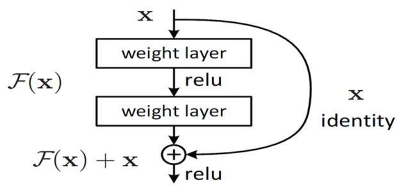

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 11: Skip connections. Here, "x" is the raw input and F(x) is the features extracted from "x"

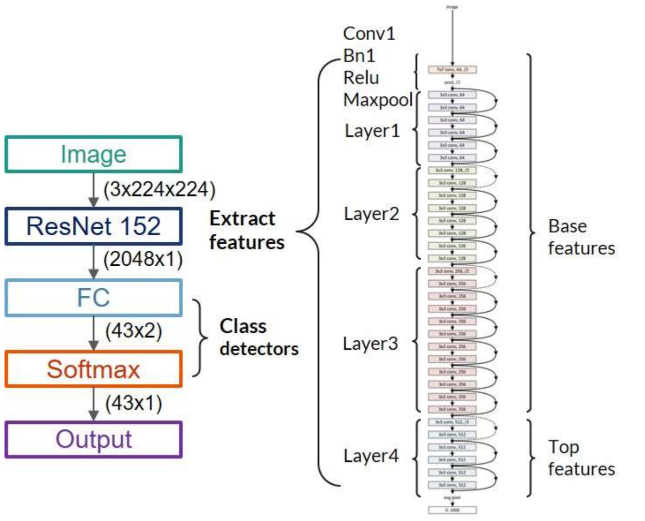


Figure 12: ResNet-152 Architecture. He et al. (2015).

The third and final architecture is a DenseNet-121 and DenseNet-161 (Figure 13). These architectures utilize a similar technique as ResNets, i.e., utilizing skip connections. However, the skip connections in this architecture are connected to each layer in the “Dense-Blocks” and each dense-block is connected to each other. This architecture allows the network to learn even more richer features and is a great option for fine-grained classification task.

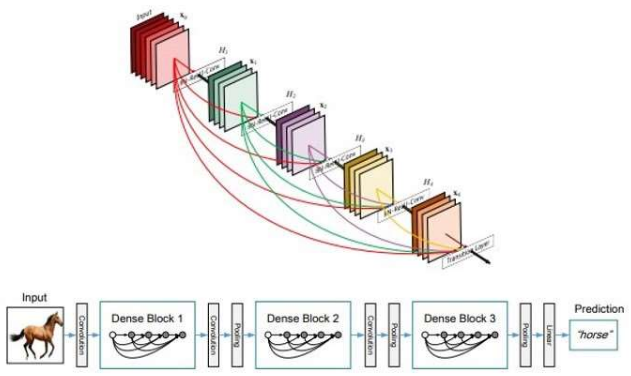


Figure 13: A General DenseNet Architecture. Huang et al. (2016)

For each architecture, we modified its classifier by simply changing the number of outputs in the last layer from 1000 to 14 and added dropout with a probability of 0.2. Initially, we also added an additional linear layer to the classifier, however, we did not see any significant improvement and thus discarded it to conserve some training time.

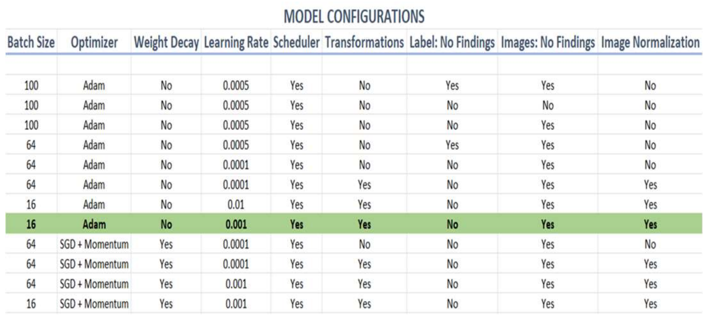
We experimented with a plethora of hyperparameters for each of these architectures. First, lets discuss what we kept common. We trained each model for 35 epochs, which took approximately 10 hours per model. We used the Binary Cross Entropy Loss (Figure 14) as our “criterion” or loss function since each disease label was binary in nature. The BCE Loss or Log Loss compares each predicted probability to the actual class (binary in this case), then penalizes the probabilities based on the distance from the expected value. Essentially, it is the negative average of the log of corrected predicted probabilities.

$$\text{Log loss} = \frac{1}{N}\sum_{i=1}^{N} -(y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

Figure 14: Formula of Binary Cross Entropy/Log Loss. *N* is the number of observations, *y* is the actual class and *p* is the predicted probability of the positive class, i.e., class 1.

We also used a Scheduler which decreased the learning rate with a factor of 30 if the validation loss of the model did not decrease for more than 3 epochs. Additionally, we only saved the model with the lowest validation loss so that we would retain the best model after the training was completed.

The hyperparameters we experimented with include the Optimizer, Batch Size, Weight Decay and Learning Rate. We experimented with two Optimizers, namely Adam and SGD with Momentum with the momentum of 0.9. In the case of SGD, we always used a weight decay of 0.1. Adam always performed better for our case. We varied the learning rate from 0.0001 to 0.01 and found that 0.001 worked the best for us. In the case of batch size, we varied it from 16 to 100 and found that although training took a little longer, a batch size of 16 outperformed higher batch sizes.


MODEL CONFIGURATIONS

Batch Size	Optimizer	Weight Decay	Learning Rate	Scheduler	Transformations	Label: No Findings	Images: No Findings	Image Normalization
100	Adam	No	0.0005	Yes	No	Yes	Yes	No
100	Adam	No	0.0005	Yes	No	No	No	No
100	Adam	No	0.0005	Yes	No	No	Yes	No
64	Adam	No	0.0005	Yes	No	Yes	Yes	No
64	Adam	No	0.0001	Yes	No	No	Yes	No
64	Adam	No	0.0001	Yes	Yes	No	Yes	Yes
16	Adam	No	0.01	Yes	Yes	No	Yes	Yes
16	**Adam**	**No**	**0.001**	**Yes**	**Yes**	**No**	**Yes**	**Yes**
64	SGD + Momentum	Yes	0.0001	Yes	No	No	Yes	No
64	SGD + Momentum	Yes	0.0001	Yes	Yes	No	Yes	Yes
64	SGD + Momentum	Yes	0.001	Yes	Yes	No	Yes	Yes
16	SGD + Momentum	Yes	0.001	Yes	Yes	No	Yes	Yes

Figure 15: A comparison of various hyperparameter configurations for each model. The green row highlights the best combination of hyperparameters. We noticed that this specific configuration works well for all three SOTA CNN architectures we have implemented.

4. Feature Extraction

4.1. Manual Feature Extraction

For the first iteration of manual feature extraction, we implemented all four methods described in the background section for Manual Features, but for some reason 16GB of ram was not enough. Hence, we played around with few parameters of Histogram of Gradients, as this was the feature extraction method that returned the largest features. Figure 16 explains the configurations we tried for HOG.

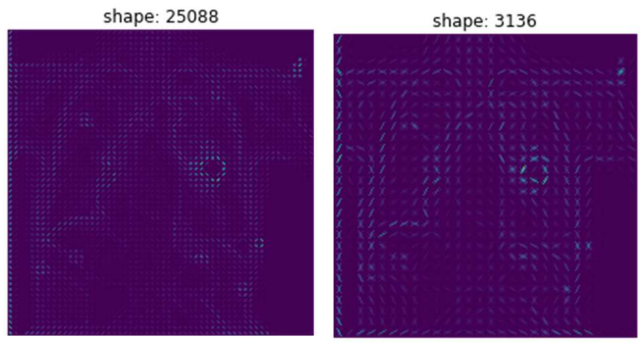


Figure 16: histogram of gradients with 8 orientations, (2,2) pixels per cell and (1,1) cells per block (Left); histogram of gradients with 8 orientations, (4,4) pixels per cell and (1,1) cells per block (Right)

Reducing the number of orientations and pixels per cells drastically decreased the feature vector size. Although it did not capture more details, the figure on the right was providing us enough information for our use case.

GLCM returned a feature vector of dimension 20, 4 angles (0, 45,90,180) and 5 second order statistical measures (contrast, dissimilarity, homogeneity, energy, and correlation). Figure 17 shows two measures calculated for 8 points on a sample chest X-ray.

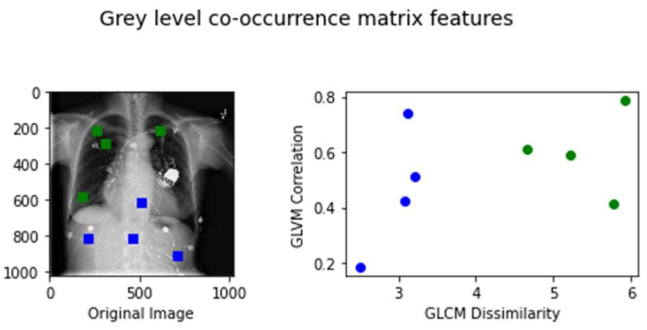


Figure 17: the green dots indicate pixels for lungs and the blue dots indicated pixels for non-lungs (Left). As we can see pixels with similar textures are grouped together (Right)

Even after reducing the HOG feature variable and converting the data type to float16 from float64 we were running into memory issues. Hence, we decided to perform chunking and remove HOG features completely. We were able to successfully extract features from all images, but the models did not perform that well.

After doing some more research we found out the LBP and HOG features combined are excellent for image classification tasks, hence we added the HOG features, performed chunking. This resulted in a considerable improvement in model performance.

4.2. CNN Extracted Feature

In this method, we utilized the SOTA CNN architectures as feature extractors. We extracted features from the last layer of the best performing SOTA CNN architecture.

To do this, we first chose the layer we wanted to extract the features from and registered a "hook" on it. Next, we passed the data to the network in batches using our custom data-loader and collected the outputs from the "hooked" layer and converted it into a data frame which was then used to train our traditional ML models as specified below.

5. Traditional ML Models

We used three distinct types of ML models, namely, linear (Logistic Regression), probabilistic (Bernoulli and Gaussian Naïve Bayes) and tree-based (Decision Trees and Extra Trees). A brief description of these models is given in the "Background" section of this paper.

We also performed hyperparameter tuning for these models so that we could achieve more accurate results and robust models.

For Logistic Regression, we experimented with the penalty, i.e., L1, L2 or a combination of both known as Elastic Net and the solver, i.e., "lbfgs" or "saga". Additionally, we increased the maximum iterations from 100 to 200. The model with the L2 penalty and "lbfgs" solver yielded the best results.

In the case of Naïve Bayes, we stuck to the default parameters, since we were comparing two types of Naïve Bayes. Also, we did not have precomputed priors to test.

Both Logistic Regression and Naïve Bayes were used as estimators for a One-vs-Rest (OVR) Classifier. An OVR classifier basically creates a separate classifier for each class and then compares the predictions over all classes and returns the probability of a particular observation belonging to the positive class.

For the tree-based classifiers, Decision Trees and Extra Trees, we experimented with the criterion entropy and Gini index, the maximum depth of the tree, the minimum samples required to split a branch, the minimum number of samples required to consider a node as a leaf node and for Extra Trees, we also experimented with the number of estimators to train. Approximately 3600 fits were being evaluated which had to be terminate after 4 days. We did not find the best parameters for decision trees using grid search, hence we decided to manually test out the best configuration. After careful consideration, we were able to come up with values for each of the discussed hyperparameters we chose were

Gini index as the criterion, 50, 150 and 75 as maximum depth, minimum samples to split and minimum samples for leaf node respectively. For Extra Trees, we chose the number of estimators to be 300, i.e., 300 decision trees were trained and used in an ensemble.

Other than the methods mentioned in the Background section for Machine Learning we tried implementing and running two more models, namely, SVR and Random Forest. Both these models were resource intensive and didn't converge even after running for few days, hence we decided not to use them for our use case.

To keep all our results reproducible, we set a random state (previously known as setting a seed).

6. Results

First, we will talk about the results of our CNN architectures. Then we will compare these results with the ML models that were trained on features extracting the best performing CNN architecture.

To evaluate all our models, we compute the false positive rate (FPR), true positive rate (TPR) and plot a ROC (Receiver Operating Characteristic) Curve. We also compute the AUC (Area Under Curve) for each label from this curve. This tells us how well the model is performing for each level at every threshold. It ranges from 0.0 being the lowest to 1.0 being the highest value.

The results on train set are shown in Figure 18a and the results on the test set are shown in Figure 18b.

Taking a first glance at the results, we can clearly see that VGG-16 performs the worst out of the four SOTA CNN architectures. It is not able to generalize well on the data and clearly underfits with an average AUC of 0.5506. This poor performance can be attributed to its relatively small and simple architecture. It performs similarly on the test set, achieving an average AUC of 0.5591. Next, we can easily identify that both Naïve Bayes algorithms have performed poorly on the train and test set achieving an average AUC of approximately 0.65 to 0.69. Here we can clearly see that a traditional ML model is performing much better than a SOTA CNN architecture. However, remember that the features fed to the traditional ML models have been extracted from the best performing SOTA CNN architecture and clearly VGG-16 is not the one. Since the traditional ML models are trained on better features than the one extracted by VGG-16, they are able to outperform it.

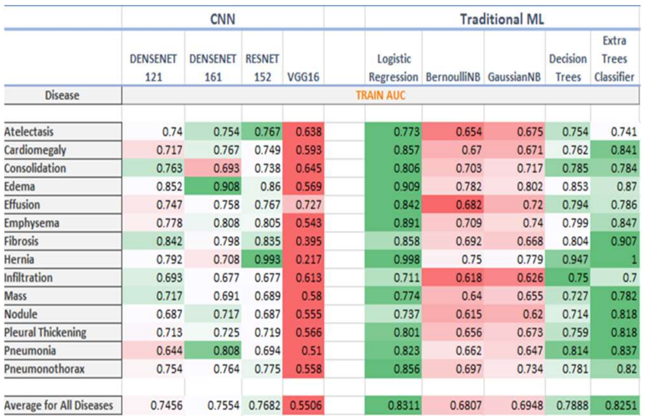
Although after some hyperparameter tuning, we were able to drastically reduce overfitting in the tree-based models, we were not successful in completely getting rid of it. As clearly shown in Figure 18a and 18b, both tree-based models overfit on the train data and perform poorly on the test data. However, we see a clear difference in results between Decision Trees and Extra Trees. The latter performs better and controls overfitting better than the former,

We trained two architectures of Dense Nets, namely DenseNet-121 and DenseNet-161. A brief description about this architecture is given in subsection 3 of "Experiments". As expected, the deeper architecture – DenseNet-161 performed slightly better, achieving an average AUC of 0.7554 on the train set and 0.7605 on the test set, whereas the DenseNet-121 achieved an average AUC of 0.7456 on the train set and 0.7458 on the test set. These results are promising, but are still of the best performing model.

Comparable to these SOTA CNN architectures was the Logistic Regression model. Although clearly overfitting on the train set and achieving the highest average AUC on the train set of 0.8311, it achieves an average AUC of 0.75 on the test set.

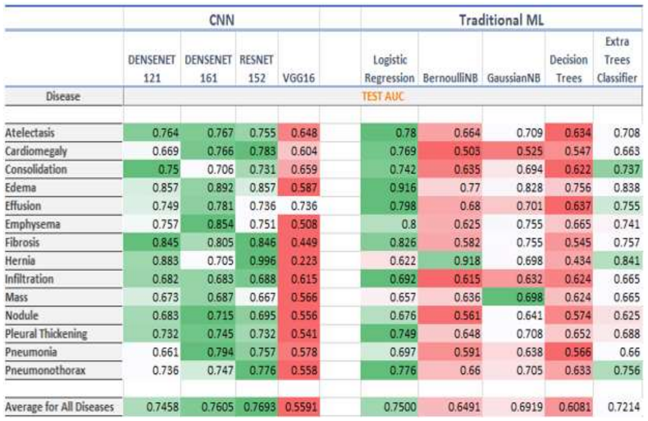
Finally, we train the largest pre-trained Residual Network provided by PyTorch – the ResNet-152. This network performed the best on both the train set and test set, achieving an average AUC of 0.7682 and 0.7693 respectively.

Figure 19a and 19b displays the ROC curve and individual AUC for all diseases of the ResNet-152



	CNN				Traditional ML				
	DENSENET 121	DENSENET 161	RESNET 152	VGG16	Logistic Regression	BernoulliNB	GaussianNB	Decision Trees	Extra Trees Classifier
Disease	TRAIN AUC								
Atelectasis	0.74	0.754	0.767	0.638	0.773	0.654	0.675	0.754	0.741
Cardiomegaly	0.717	0.767	0.749	0.593	0.857	0.67	0.671	0.762	0.841
Consolidation	0.763	0.693	0.738	0.645	0.806	0.703	0.717	0.785	0.784
Edema	0.852	0.908	0.86	0.569	0.909	0.782	0.802	0.853	0.87
Effusion	0.747	0.758	0.767	0.727	0.842	0.682	0.72	0.794	0.786
Emphysema	0.778	0.808	0.805	0.543	0.891	0.709	0.74	0.799	0.847
Fibrosis	0.842	0.798	0.835	0.395	0.858	0.692	0.668	0.804	0.907
Hernia	0.792	0.708	0.993	0.217	0.998	0.75	0.779	0.947	1
Infiltration	0.693	0.677	0.677	0.613	0.711	0.618	0.626	0.75	0.7
Mass	0.717	0.691	0.689	0.58	0.774	0.64	0.655	0.727	0.782
Nodule	0.687	0.717	0.687	0.555	0.737	0.615	0.62	0.714	0.818
Pleural Thickening	0.713	0.725	0.719	0.566	0.801	0.656	0.673	0.759	0.818
Pneumonia	0.644	0.808	0.694	0.51	0.823	0.662	0.647	0.814	0.837
Pneumonothorax	0.754	0.764	0.775	0.558	0.856	0.697	0.734	0.781	0.82
Average for All Diseases	0.7456	0.7554	0.7682	0.5506	0.8311	0.6807	0.6948	0.7888	0.8251

Figure 18a: AUC of all diseases of all implemented models on the Train set.



	CNN				Traditional ML				
	DENSENET 121	DENSENET 161	RESNET 152	VGG16	Logistic Regression	BernoulliNB	GaussianNB	Decision Trees	Extra Trees Classifier
Disease	TEST AUC								
Atelectasis	0.764	0.767	0.755	0.648	0.78	0.664	0.709	0.634	0.708
Cardiomegaly	0.669	0.766	0.783	0.604	0.769	0.503	0.525	0.547	0.663
Consolidation	0.75	0.706	0.731	0.659	0.742	0.635	0.694	0.622	0.737
Edema	0.857	0.892	0.857	0.587	0.916	0.77	0.828	0.756	0.838
Effusion	0.749	0.781	0.736	0.736	0.798	0.68	0.701	0.637	0.755
Emphysema	0.757	0.854	0.751	0.508	0.8	0.625	0.755	0.665	0.741
Fibrosis	0.845	0.805	0.846	0.449	0.826	0.582	0.755	0.545	0.757
Hernia	0.883	0.705	0.996	0.223	0.622	0.918	0.698	0.434	0.841
Infiltration	0.682	0.683	0.688	0.615	0.692	0.615	0.632	0.624	0.665
Mass	0.673	0.687	0.667	0.566	0.657	0.636	0.698	0.624	0.665
Nodule	0.683	0.715	0.695	0.556	0.676	0.561	0.641	0.574	0.625
Pleural Thickening	0.732	0.745	0.732	0.541	0.749	0.648	0.708	0.652	0.688
Pneumonia	0.661	0.794	0.757	0.578	0.697	0.591	0.638	0.566	0.66
Pneumonothorax	0.736	0.747	0.776	0.558	0.776	0.66	0.705	0.633	0.756
Average for All Diseases	0.7458	0.7605	0.7693	0.5591	0.7500	0.6491	0.6919	0.6081	0.7214

Figure 18b: AUC of all diseases of all implemented models on the Test set.

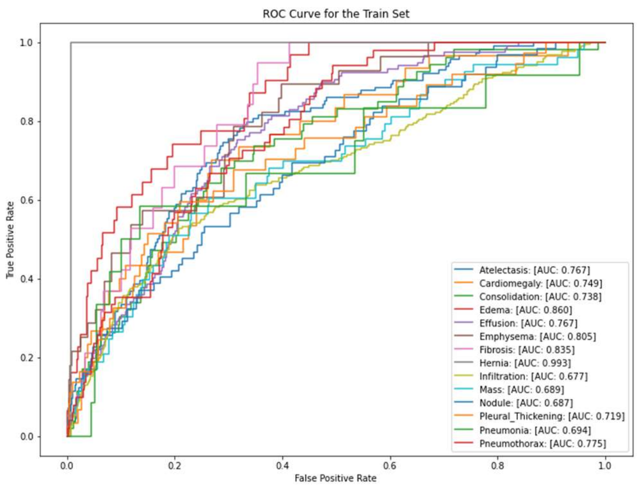


Figure 19a: ROC Curve for ResNet-152 on the Train set.

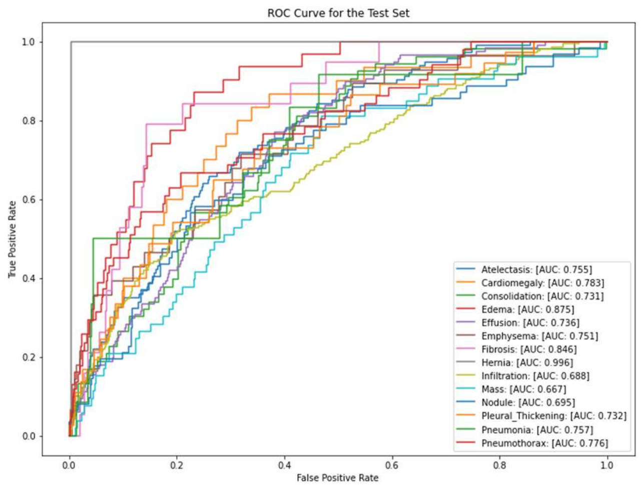
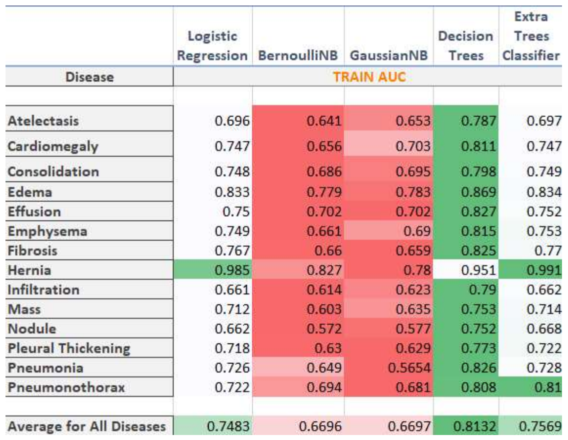


Figure 19a: ROC Curve for ResNet-152 on the Test set.

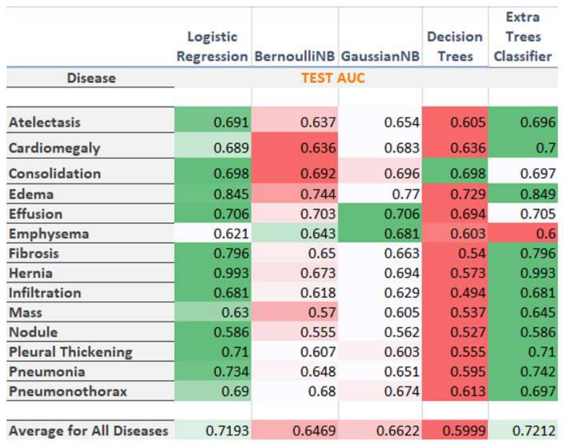
Initially, due to the enhanced architecture of the Dense Nets, we presumed them to work the best, even better than the Residual Nets. However, the results are contradicting to out initial assumption. Although similar in performance but shallower than DenseNet-161, ResNet-152 was our best performing model. After some research, we found that since DenseNet-161 has higher capacity with multi-layer feature concatenation, it requires more GPU resources and a greater number of time/epochs to train successfully. ResNets overcome this due to their ability of limit representation capacity using their identity mapping which stabilizes their training more than Dense Nets. Also, we believe, since ResNets have a greater number of parameters to train, they learn better relationships early in their training.

Figure 20a and 20b display the results for traditional machine learning models on Manually Extracted Features. As seen from the results, these models are performing very similar to the ML models on CNN Extracted Features. For both these cases, hyperparameters were not changed as we wanted to compare how well traditional ML models perform on manually extracted features. Bernoulli Naïve bayes and Gaussian Naïve Bayes are very similar in performance for both the cases, achieving an average AUC of approximately 0.64 to 0.67 on the train and test sets. In both the cases Decision Tress are slightly overfitting on the Training set, and hence have a poor performance on the test set, the slight increase in performance for CNN extracted Features can be attributed to more overfitting. Contrary to the previous case, where ML models were trained on CNN Extracted Features, Logistic Regression and Extra Trees Classifier are not overfitting to the training data, this can be because the number of features returned by the CNN extraction method is 2048 and the number of features returned by the Manual Feature Extraction methods is 3996. Hence, due to increase in feature size, these models are not overfitting in this case. The best performing model on Manually Extracted Features is Extra trees with an average train AUC of 0.75 and test AUC of 0.72. Logistic regression comes next, with a small difference of 0.01 for the performance on test set with an average AUC of 0.71.



	Logistic Regression	BernoulliNB	GaussianNB	Decision Trees	Extra Trees Classifier
Disease	TRAIN AUC				
Atelectasis	0.696	0.641	0.653	0.787	0.697
Cardiomegaly	0.747	0.656	0.703	0.811	0.747
Consolidation	0.748	0.686	0.695	0.798	0.749
Edema	0.833	0.779	0.783	0.869	0.834
Effusion	0.75	0.702	0.702	0.827	0.752
Emphysema	0.749	0.661	0.69	0.815	0.753
Fibrosis	0.767	0.66	0.659	0.825	0.77
Hernia	0.985	0.827	0.78	0.951	0.991
Infiltration	0.661	0.614	0.623	0.79	0.662
Mass	0.712	0.603	0.635	0.753	0.714
Nodule	0.662	0.572	0.577	0.752	0.668
Pleural Thickening	0.718	0.63	0.629	0.773	0.722
Pneumonia	0.726	0.649	0.5654	0.826	0.728
Pneumonothorax	0.722	0.694	0.681	0.808	0.81
Average for All Diseases	0.7483	0.6696	0.6697	0.8132	0.7569

Figure 20a: AUC for all classes for ML models on Manually Extracted Features on the Train Set



	Logistic Regression	BernoulliNB	GaussianNB	Decision Trees	Extra Trees Classifier
Disease	TEST AUC				
Atelectasis	0.691	0.637	0.654	0.605	0.696
Cardiomegaly	0.689	0.636	0.683	0.636	0.7
Consolidation	0.698	0.692	0.696	0.698	0.697
Edema	0.845	0.744	0.77	0.729	0.849
Effusion	0.706	0.703	0.706	0.694	0.705
Emphysema	0.621	0.643	0.681	0.603	0.6
Fibrosis	0.796	0.65	0.663	0.54	0.796
Hernia	0.993	0.673	0.694	0.573	0.993
Infiltration	0.681	0.618	0.629	0.494	0.681
Mass	0.63	0.57	0.605	0.537	0.645
Nodule	0.586	0.555	0.562	0.527	0.586
Pleural Thickening	0.71	0.607	0.603	0.555	0.71
Pneumonia	0.734	0.648	0.651	0.595	0.742
Pneumonothorax	0.69	0.68	0.674	0.613	0.697
Average for All Diseases	0.7193	0.6469	0.6622	0.5999	0.7212

Figure 20b: AUC for all classes for ML models on Manually Extracted Features on the Test Set

Conclusion

Although State of the Art CNNs are known for their excellent learning capabilities using their flexibility and non-linearity, traditional Machine Learning models are powerful too. Specifically, for our use case we saw that some traditional ML models, Logistic Regression and Extra Trees, with manually extracted features performed very similar to state-of-the-art CNN models. This observation can effectively help us replace computationally expensive and heavy CNNs with light weight, compact and easy to use traditional machine learning models.

Furthermore, we also concluded that CNNs are excellent at learning patterns and similarities between the input data. This can be observed from the fact that, even though number of features returned by CNNs were less in number as compared to manually extracted features, the traditional machine learning models performed quite similar in both the cases, sometimes better than manually extracted features models.

For future works, we would like to improve out traditional models on manually extracted features by focusing more on feature engineering and adding more meaningful features. An example of this could be using the larger Histogram of Gradients feature vector along with LBP. Along with that, we would also like to fine-tune their performance and reduce overfitting by cross-validating the data.

References

Data Source: https://www.nih.gov/news-events/news-releases/nih-clinical-center-provides-one-largest-publicly-available-chest-x-ray-datasets-scientific-community

Rajpurkar et. al, 2017, CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. https://arxiv.org/pdf/1711.05225.pdf

Wang et al. 2017, ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. https://openaccess.thecvf.com/content_cvpr_2017/papers/Wang_ChestX-ray8_Hospital-Scale_Chest_CVPR_2017_paper.pdf

Yao et al. 2017, Learning to diagnose from scratch by exploiting dependencies among labels. https://arxiv.org/abs/1710.10501

Zhang et al. (2020), ResNet or DenseNet? Introducing Dense Shortcuts to ResNet. https://arxiv.org/pdf/2010.12496.pdf

Manual Feature Extraction: https://www.researchgate.net/publication/342710840_Analysis_on_Various_Feature_Extraction_Methods_for_Medical_Image_Classification

He et al. (2015), Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385

Simonyan et al. (2014), Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556

Huang et al. (2016), Densely Connected Convolutional Networks. https://arxiv.org/abs/1409.1556