Improving X-Ray Disease Classification by combining Probabilistic Graphical Models and Variational Inference with Deep Neural Networks

Jagjit Singh | UNI: JS5958
BISTP 8124
Final Project
js5958@cumc.columbia.edu

Abstract

This project explores the use of probabilistic graphical models trained via variational inference for feature extraction from X-Ray scans, and using the extracted features as input to a Deep Neural Network for disease classification. The aim is to find methods to overcome the short-comings of Convolutional Neural Networks when it comes to chest X-Ray disease classification. Alternate methods for feature extraction are also demonstrated.

1 Introduction

X-ray exams are very common and are used for the diagnosis of several chest-related diseases, especially for Pneumonia. However, the resulting scans can be challenging to classify and are sometimes believed to be harder than diagnosis via chest CT imaging. Even for statistical algorithms such as a Convolutional Neural Network (CNN), there is still high classification error as the learned features may be superficial and may misrepresent the complexity of the anatomical structures. Different chest-related diseases such as Nodules or Edema result in different, white-colored processes which show up at different regions in the X-ray corresponding to the respective disease. However, the Convolutional Neural Network learns the entire image instead of looking for these specific features. And given the nature of supervised learning of a CNN, it will lead to over-fitting. Thus, I was interested in trying to use the course learnings in variational inference and probabilistic graphical models to better capture the disease-specific feature distributions, which could then be used as input data to a Deep Neural Network to increase predictive performance.

2 Dataset

The data used for the project is the X-Ray image dataset compiled by NIH. The dataset contains greater than 100,000 X-Ray scans. It also contains radiologist labels to indicate 14 common pathologies of the thorax depending on the X-Ray. The labels were mined through Natural Language Processing of the patient health records. In total there are 15 separate labels: Atelectasis, Cardiomegaly, Effusion, Infiltration, Mass, Nodule, Pneumonia, Pneumothorax, Consolidation, Edema, Emphysema, Fibrosis, Pleural Thickening, Hernia, and No Finding. The text mined disease labels are expected to have an accuracy of greater than 90 percent.

3 Data Preprocessing

The data was downloaded in separate batches and then unzipped into one directory thereby containing all the X-ray images. There were a total 112,120 frontal view chest X-ray scans. To match the images with the labels, NIH provides a CSV file that contains the image filenames and the corresponding labels. To get the data into a trainable form, I first had to create separate directories for each of the labels, to house all the X-Ray scans for those labels. This was done via a python code script. An important thing to note is that there were images classified with multiple labels for example: 'Atelectasis | Cardiomegaly'. Such multi-labeled images were not considered for the main analysis or inference as there were 808 multi-label categories with not many scans for each of these categories. Thus, our final dataset contained, 91324 scans belonging to 15 labels. This was then split into training and testing. Thus the final training dataset had a total of 73060 X-Ray scans/images and the testing data a total of 18264 scans. The testing dataset would be used to compare the performance of the Convolutional Neural Network versus the model that I develop. Lastly the image resolution was reduced to 128 by 128 pixels from 1024 by 1024 pixels. This was necessary as training on full resolution of 1024 by 1024 pixels is computationally intensive and requires a computer cluster. The scans were converted to RGB images, thus the final shape of an X Ray scan converted to a training scan/image (scan and image are the same) was (128,128,3) where 3 is the number of channels in the image. Also, the data was normalized during model training. Below is a sample of 3 scans from the training data with the labels:

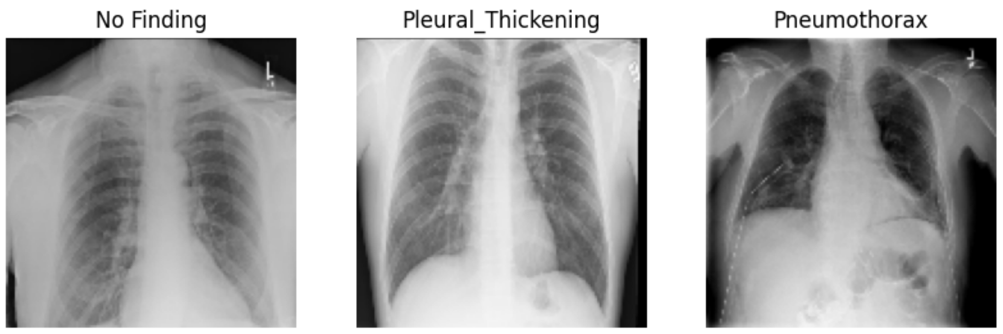


Figure 1: Sample X-Ray Scans from the Training Data

4 Methods and Model Training

X-Ray Disease Classification Via CNN: A Convolutional Neural Network is essentially a direct acyclic graph of a layers, where each layer comprises of multiple individual regressors. Given a sample image$\mathbf{X_{n_1 \times n_2 \times c_{in}}}$ a Kernel Matrix $\mathbf{W_{m_1 \times m_2 \times c_{in} \times c_{out}}}$ and bias value $\mathbf{b_{c_{out}}}$, the math behind a single convolution layer is:

$$\mathbf{Y^{Conv}}[:,:,\mathbf{k}] = \sum_{\mathbf{c=1}}^{\mathbf{c_{in}}} \mathbf{W}[:,:,\mathbf{c},\mathbf{k}] * \mathbf{X}[:,:,\mathbf{c}] \quad (1)$$

$$\mathbf{A}[:,:,\mathbf{k}] = \mathbf{Y^{Conv}}[:,:,\mathbf{k}] + \mathbf{b}[\mathbf{k}] \quad (2)$$

$$\mathbf{H}[:,:,\mathbf{k}] = \varnothing(\mathbf{A}[:,:,\mathbf{k}]) \quad (3)$$

The symbol $*$ represents a convolutional operation which gives the CNN its name. The final result H is called a feature map, which is essentially a representation of the input. The $\varnothing$ is an activation function. Since we converted the X-Ray images to RGB format in data preprocessing our $c_{in} = 3$, signifying 3 input channels. Figure 2 is a visual representation of the the CNN architecture:

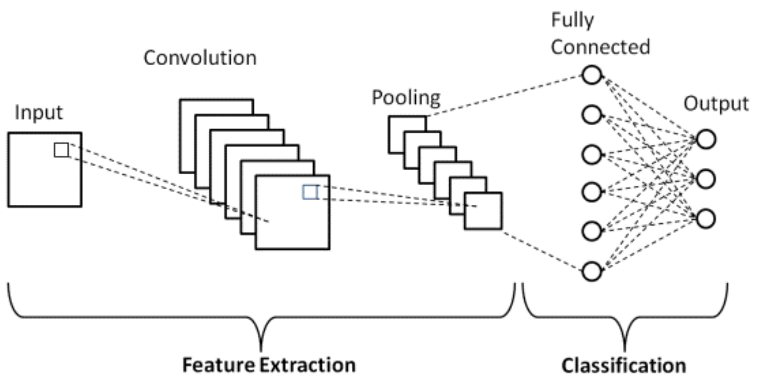
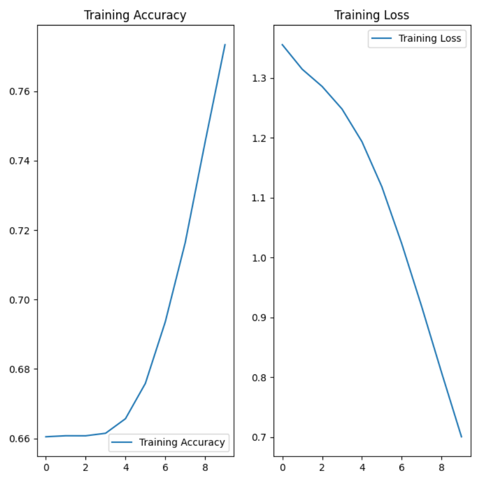
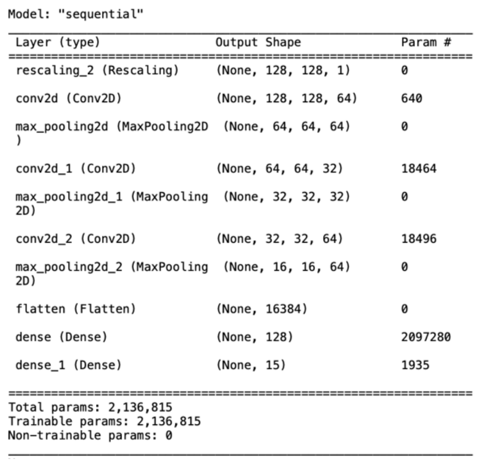


Figure 2: CNN Architecture

Figure 3 contains the CNN model summary and a plot of the training process. It has three convolutional layers. The network was trained over 10 epochs. We can see the decrease in the training loss over the different epochs as training progresses.





```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 128, 128, 1)       0

 conv2d (Conv2D)             (None, 128, 128, 64)      640

 max_pooling2d (MaxPooling2D  (None, 64, 64, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 128)               2097280

 dense_1 (Dense)             (None, 15)                1935

=================================================================
Total params: 2,136,815
Trainable params: 2,136,815
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Training Summary (Left) and CNN Model Summary (Right)

Figure 4 is a feature visualization, in order to get a sense of the X-Ray patterns learnt by different CNN layers and provides a visual representation of the filter $(maxpooling2d)$. The trained CNN was fed an X-Ray Scan labeled as Pneumothorax. We can see that the lung cavity is an important feature.

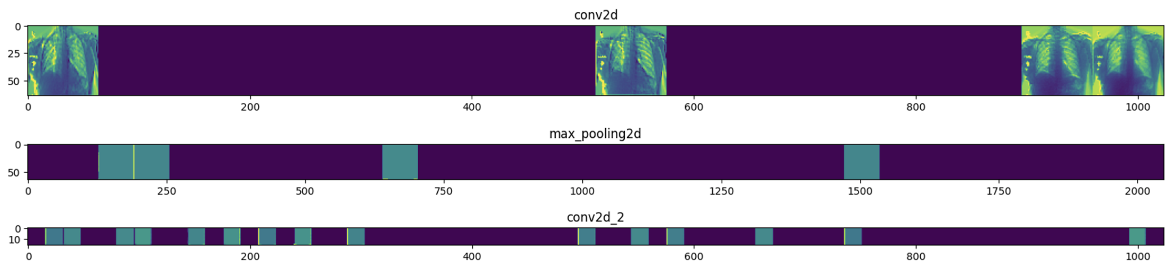


Figure 4: Filters and Feature Maps of CNN

Feature Encoding: From the feature map and filter representation we see how the CNN learns the feature space, to classify the image. However, we can we see that it learns the entire image space instead looking for disease specific distributions/patterns. Now we will try a different method for feature extraction. We will build latent representations of the feature distributions. This would be done via the use of a Gaussian Variational Auto-encoder while viewing it from a probabilistic graphical model perspective by specifying a joint probability model of the data x (X-ray Scans) and the latent variables z. Decomposing the joint distribution into the likelihood and prior we get: $p\,(x,z) = p\,(x|\,z)p(z)$. While training the autoencoder, we will apply variational inference to infer good values of the latent variables given the observed data. Thus, the latent variables would be the encoding of the different X-ray disease classes. A graphical presentation assuming M independently model samples would be:

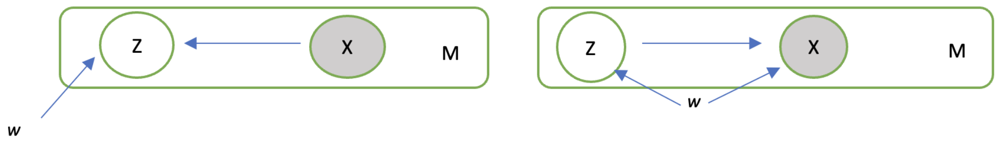


Figure 5: Graphical Representation of Inference (Left) and Generative Process (Right)

The left figure is the inference model $q(z|x)$. In this representation the latent variable Z is a standard normal and the data are drawn from $p\,(x|\,z)$. The shaded node X is the observed data. Thus, we can see that it enforces a prior $p(z)$. Both the encoder and decoder are governed by weights w which are neural network parameters.

The encoder network maps the observed data to a standard normal distribution and outputs two latent vectors Z_μ and Z_σ of shape depending on the dimensionality of the latent space. Thus

$$q(z|x) = \mathcal{N}(z; \mu(x;w), diag(\sigma^2(x;w))) \tag{4}$$
$$p(z) = \mathcal{N}(z; 0, \mathcal{I}_q) \tag{5}$$

with w being the dependence on the neural network weights. Following this $z \sim q(z|x)$ is passed to the decoder to get the reconstructed image $\hat{X}$. Below is the latent encoding in a 2-dimensional space of a random sample of 800 X-Ray scans from our training dataset:

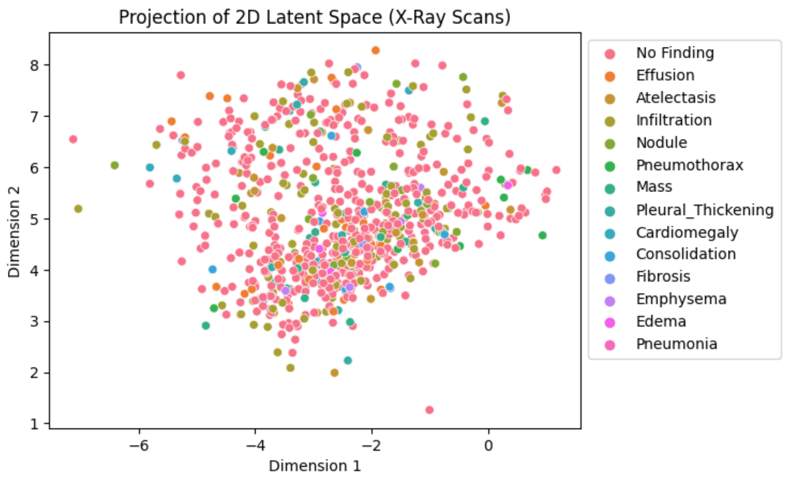


Figure 6: Latent Projection of X-Ray Scans

Thus, we started with an array of shape (800,128,128,3) which comprises of 800 images, each size of 128 by 128 pixels across 3 channels, and encoded the images to a latent representation of shape

(800,2). Variational Inference plays a big part in the training of a Variational Autoencoder as we are approximating the posterior with a family of distributions $q_\lambda(z|x)$. λ indexes the family of distributions. As stated previously I am working with a Gaussian VAE, thuss $\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})$. The most optimal posterior distribution $q(z|x)$ which best approximates the posterior $p(z|x)$ is

$$q^*_\lambda(z) = argmin_\lambda KL(q_\lambda(z|x) \parallel p(z|x)) \tag{6}$$

where KL is the Kullback Leibler divergence. Expanding it into its components we get:

$$KL[q_\lambda(z|x) \parallel p(z|x)] = \mathbf{E}_q[log(q_\lambda(z|x))] - \mathbf{E}_q[log(p(x,z))] + log(p(x)) \tag{7}$$

However, in the above equation p(x) is impossible to compute directly. This is where the Evidence Lowed Bound comes into play as it is a useful lower bound on log likelihood of the data. Thus,

$$ELBO\left(\lambda\right) = E_q\left[log(p\left(x,z\right))\right] - E_q\left[\log(q_\lambda\left(z|x\right))\right] \tag{8}$$

Combining this with the above equation of KL divergence, we get

$$log(p(x)) = ELBO(\lambda) + KL[q_\lambda(z|x) \parallel p(z|x)] \tag{9}$$

Thus, we are maximizing the ELBO by minimizing the KL divergence, and this is essentially the variational inference part where we are maximizing the ELBO with respect to variational parameters. Thus we defined a probability model p of the X-Ray Scans and the latent variables, and then a latent family q for latent variables to approximate p(z|x). On the next page are the reconstructed X-ray scans for a random samples from our training dataset. The number of latent dimensions were increased to 6272. This is because a latent dimensionality of 2 was too less for any significant feature extraction and reconstruction. 6272 dimensions is still a significant dimensionalty reduction as the original image had total 128*128 = 16384 features.

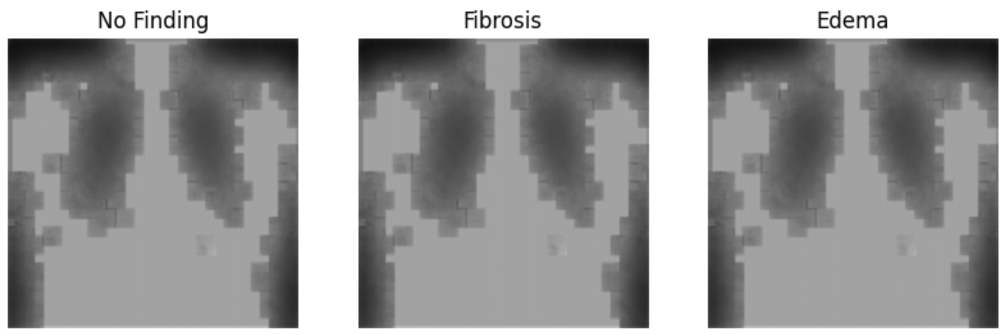
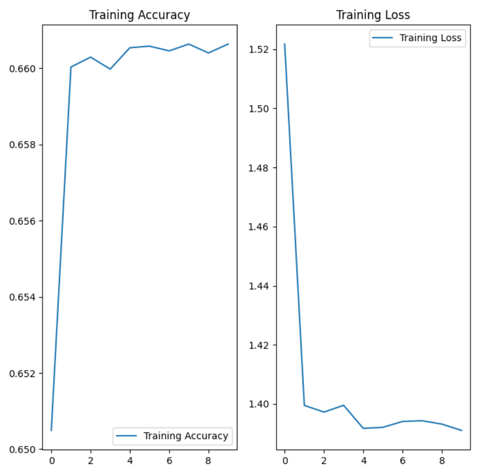
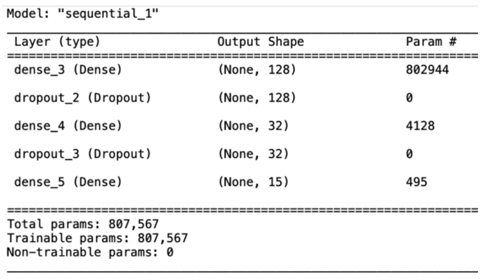


Figure 7: Reconstructed X-Ray Scans from the Decoder

Using Latent Representations for Classification: Now we will make use of the computed latent encoding for classification of X-Ray Scans. This would be done via using the encoding as an input to a Dense Neural Network and training the network via back-propagation. Thus our input dataset would be a matrix of shape (73060, 6276), where 73060 is the number of X-Ray scans and 6272 is the number of dimensions of the latent encoding per scan. Figure 8 is the model summary and a graph of the training process. I have also added 2 dropout layers in between the dense layers to avoid over-fitting.





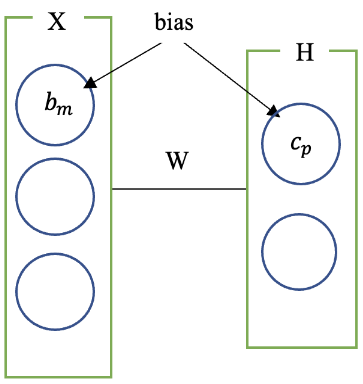
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               802944

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 32)                4128

 dropout_3 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 15)                495

=================================================================
Total params: 807,567
Trainable params: 807,567
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Training Summary (Left) and Model Summary (Right)

5 Test Results:

The performance of the basic CNN model and the model that I defined which is using the encoder first for feature extraction and then applying a Dense Neural Network for classification, were compared on the same test data. The CNN model had a test accuracy of 0.58 and my model had a test accuracy of 0.67. This an increase in accuracy by approximately 15 percent. Thus, I was able to address the problem in a useful way, or at-least do a proof of concept that this methodology can be implemented. My initial assumption was that the CNN would over-fit the training dataset which has turned out to be correct. The training accuracy of the CNN was around 0.78, which is much higher than its test accuracy. The model that I defined performed consistently when transitioning to the test dataset.

Alternate Method: Another method to do feature extraction for X-Ray disease classification would be via a Restricted Boltzmann Machine (RBM). An RBM is a latent variable model inspired by the "layered" structure of the Neural Network. Like an encoder, it learns a probability of distribution over its set of inputs. It comprises of a visible layer and a hidden layer. The nodes of the same layer are not connected with each other. It is restricted to form an undirected bipartite graph. Following is a visual representation of a Bernoulli RBM with one hidden layer:



Bernoulli RBM:

$$p(x,h;\theta) = \frac{1}{Z(\theta)} e^{(-E(x,h;\theta))}$$

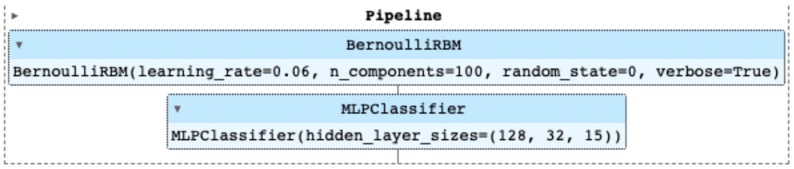
$$-E(x,h;\theta) = -\sum_{i=1}^{p}\sum_{m=1}^{M} x_i h_m W_{im} - \sum_{i}^{M} x_i b_i - \sum_{m=1}^{M} h_m c_m$$

$$Z(\theta) = \sum_x \sum_h e^{-E(x,h;\theta)}$$

I converted the normalized grayscale X-Ray images to binary images by applying a threshold function. This was necessary for the use of a Bernoulli RBM. It is fine to use a Bernoulli RBM on X-Ray images as pixel values can be interpreted as degrees of whiteness on a black background, thus the Bernoulli RBM can perform effective non-liner feature extraction.

Figure 9: Binary/Bernoulli Restricted Boltzmann Machine

The training was done via persistent contractive divergence. Persistent contractive divergence algorithm uses the final state of the Gibbs sampler as the start for the next iteration. Gibbs sampling is a Markov Chain Monte Carlo algorithm used to obtain an approximate sequence of observations from a multivariate probability distribution by making use of the conditional distributions. Thus given the X-Ray scan vector x we use $p(h|x)$ for prediction of hidden values h. Once that is done we use the hidden values to get $p(x|h)$ which is the reconstruction of the input vector. For training the RBM I reloaded the training data without converting it in RGB format. Thus our input shape per image was now (128,128,1). For demonstration, I have used 3200 observations for training, as training the RBM on the entire training dataset was taking too long and became difficult due to time constraints. The train dataset was then reshaped from (3200, 128, 128, 1) to (3200, 128$*$128$*$1). The images were essentially flattened and a binary threshold function was applied. Thus our final shape of training dataset was (3200, 16384). This data was then fed into a pipepline built via the Scikit learn package in python. Following is a visual representation of the pipeline. A Dense Neural Network was stacked below the Bernoulli RBM for classification. The Bernoulli RBM had 1 hidden layer of 256 binary units.



```
Pipeline
BernoulliRBM
BernoulliRBM(learning_rate=0.06, n_components=100, random_state=0, verbose=True)
MLPClassifier
MLPClassifier(hidden_layer_sizes=(128, 32, 15))
```

Figure 10: Pipeline Conponents

The training accuracy of the RBM model was around 0.67. Following is a list of components extracted by the RBM:

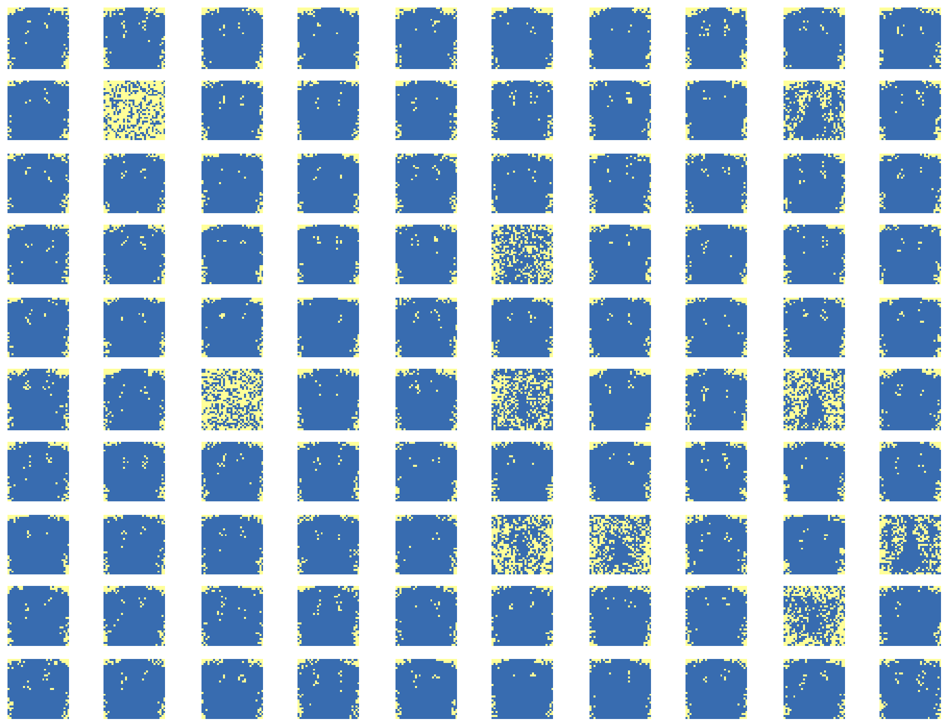


Figure 11: RBM Component Grid

We can see from the white dots in each component that it has learnt the lung cavities. However upon having a closer to the train and test predictions I saw it is classifying everything to ‘No Findings’ and since a good portion of the data is ‘No Findings’ it performs reasonably well in training. This might also be because only 3200 scans were used instead of the entire training dataset of 73060. More importantly, the currently specified architecture is also very simple with just one hidden layer of 256 binary units, as it is mainly for demonstration purposes. Comparing the results with the Convolutional Neural Network and my model, I believe that my model performs best in the testing scenario. However, further structural changes can be made to all three: the Convolutional Neural Network, My Model, and the RBM based Classification Model, which can significantly impact predictive performance.

6 Discussion & Conclusion:

In this project I started with the short-comings that Convolutional Neural Networks have with X-Ray disease classification. I tried to address this problem by using a probabilistic graphical model (Variational Autoencoder) to infer latent representations of the observed X-Ray scans and then used these representations as input to a Deep Neural Network for disease classification. I then demonstrated another method which is using a Restricted Boltzmann Machine for feature extraction, and using the extracted features as input into a Dense Neural Network for Classification. An important thing to note is that, in this project I reduced the resolution when loading the training data, which can have significant impact on the model performance. I hope the analysis that I have performed is good motivation to invest time and resources for training and development using the full resolution (1024,1024) of the X-Ray Scans, as that will lead to a significant increase in accuracy. I have made a github repository to house the code for the project. To view it, please click here.