Pneumonia Detection and Comparison Using Deep Learning and Machine Learning Techniques

Ananya Sairaj
ananya.sairaj@colorado.edu

Eric Dahl
eric.dahl@colorado.edu

Mohammed Adib Oumer
mohammed.oumer@colorado.edu

Yeshu Sharma
yeshu.sharma@colorado.edu

Abstract

Artificial intelligence has the prospective to transform disease diagnosis and management by performing classification difficult for specialists and has the potential of reviewing immense amounts of images in a very rapid manner. Despite its potential, clinical decipherability and practical implementation of AI is still challenging in the real world. In these challenging times where COVID-19 has severely impacted the economic, physical and mental well-being. This publication provides various techniques of differentiating between normal and pneumonic lungs using various deep learning and machine learning methods. Since Pneumonia is one of major factor that makes COVID-19 critical. It also provides a comparison between the models and concludes which model could provide the best accuracy. This publication also throws light on how the existing system can be further improved and used in diagnosis of other diseases.

1 Introduction

Artificial intelligence helps specialists to drastically increase the scale of medical analysis by the usage of techniques in AI such as machine learning (ML) and deep learning (DL) and be able to switch their attention from examining just individual patients to observing communities and being able to predict disease outbursts. With COVID-19 taking over the world in the last few years, AI can prove to be a very efficient tool in detecting lung infections in early stages of the disease. In most of COVID-19 patients, it is severe pneumonia that makes this disease extremely dangerous and can potentially turn malignant. Therefore, diagnosing pneumonia in COVID-19 patients at an early stage is critical. The use of deep learning techniques can aid us in achieving this goal. Deep learning algorithms can be used to detect and diagnose pneumonia using only chest X-ray images, in the process saving both money, time and lives.

Literature review

Researchers have been working on techniques and ways to help doctors and specialists in fighting diseases like skin cancer, liver cancer, heart disease, alzhemier, etc. that need to be diagnosed early using tools such as ML and DL.

Hence, the researchers have previously used various techniques like Boltzmann machine, K-nearest neighbour, support vector machine (SVM), decision tree and artificial neural network for disease diagnosis and. Some great examples are, a study in which a backpropagating neural network was used in diagnosis of skin disease with data collected from a real-world data (from a dermatological department) [4]. Other researchers used techniques such as recurrent neural network (RNN) and feed-forward neural network to diagnose liver disease hepatitis virus and were able to get accuracy of about 97.59% and 100% accuracy respectively [1]. Another group got 97.06% area under the curve by using residual neural network and long short-term memory (LSTM) to diagnose gastrointestinal diseases [8]. Others have introduced a computerized arrangement framework to recover the data designs. They proposed a five-phase machine learning pipeline that further arranged each stage in various sub levels [5].

These studies prove that the use of ML and DL is important in the medical field and can result in very reliable results that could facilitate most of the tasks involved in the field.

Objective

According to various surveys, pneumonia causes up to a million hospitalizations and has proven to be very fatal by causing approximately 50,000 deaths every year. If pneumonia is not treated during the early stages of its development, it can cause infection in one or both lungs, causing inflammation and fluid build-up [11]. This project's aim is to develop three models that will be able to distinguish between normal and pneumonic lungs, and to compare the performance across the different frameworks. The following three approaches were selected:

1. Boosting Algorithm:- This project throws light on how a machine learning algorithm would perform on the given dataset. The aim is to be able to draw a comparison between a ML approach and a DL approach.
2. Fully Connected Layer Algorithm:- The fully connected layer consists of neurons that are connected to each neuron from the previous stage where each connection has its own weight. The fully connected layer is considered more generic because it makes no presumption about the features in the dataset. Whereas in a convolutional layer each neuron is connected to some neurons in the previous stage.
3. State-Of-The-Art EfficientNetV2:- The last model implemented was EfficientNetV2, which is a type convolutional neural network (CNN) that has faster training speed superior parameter efficiency than other existing models.

1.1 Data Set and Preprocessing

This study has used a dataset provided by Kaggle, the dataset includes a collection of images which are labelled as pneumonic and normal. The total number of images are 5,232 Images. There are 3,883 images that are classified as pneumonic (2,538 bacterial and 1,345 viral), and 1,349 are classified as normal [6]. A sample of the images is shown in Figure 1 The images in the dataset were of different shapes and sizes and they needed to be preprocessed first by resizing all to a uniform shape and rotating some.

2 Method I - Boosting

The boosting part of this project uses a custom built Naive Bayes classifier and scikit's Decision Tree classifier as a weak learner and applies the popular AdaBoost boosting algorithm.

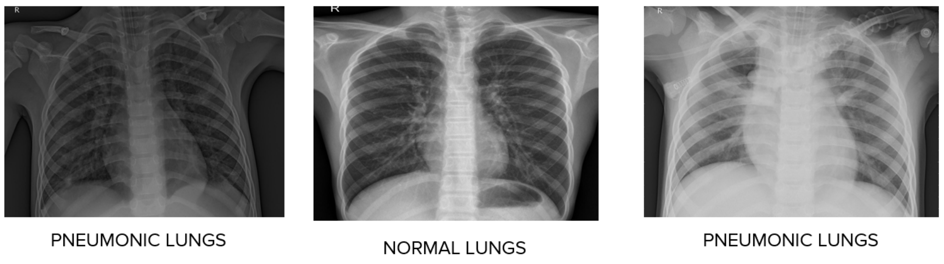

Figure 1: Example of Pneumonic and Normal Lungs

Naive Bayes classifier is a kind of Bayes classifier (classifier based on Bayes theorem). A Bayes classifier is given by [2]:

$$\hat{y}(\mathbf{x}) = arg\max_{y\in\mathcal{Y}} p(y|\mathbf{x}) \propto arg\max_{y\in\mathcal{Y}} p(y)p(\mathbf{x}|y) = arg\max_{y\in\mathcal{Y}} \mathbf{log}(p(y)) + \mathbf{log}(p(\mathbf{x}|y)) \tag{1}$$

where the proportionality applies through the Bayes theorem, y is our label and $\mathbf{x} \in \mathbb{R}^d$ is our feature vector test data. Naive Bayes assumes that the features in our feature vector have independent distributions, i.e.:

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y) \tag{2}$$

In our project, we took two approaches on estimating $p(\mathbf{x}|y)$. The first approach is to consider the grayscale nature of our image dataset to be a continuous data in the range between $[0, 1]$. This way, each of the probabilities for $p(\mathbf{x}|y)$ can be estimated using normal distributions with the empirical sample mean and sample variances of each class since they are joint maximum likelihood estimates. Then:

$$\begin{aligned} p(y) &= \frac{m_y}{m} \\ p(\mathbf{x}|y) &= \prod_{i=1}^{d} p(x_i|y) \sim \mathcal{N}(\hat{\mu}, \hat{\Sigma}^2) \end{aligned} \tag{3}$$

where m_y is number of class y in the dataset, m is total length of dataset, $\hat{\mu}$ is sample mean vector for each class y and $\hat{\Sigma}^2$ is the sample covariance matrix for each class y which is a diagonal matrix of the sample variances for each class y in this case (since we are assuming independence between features) [3].

The other approach is to take the feature vectors (pixel values) to be discrete integer values {0,1,...,255} and model them by multinomial distributions. We estimate $p(y)$ as given in Equation 3 but $p(x_i|y)$ is approximated by the smoothed version of the maximum likelihood estimate as given in Equation 4.

$$p(x_i|y) = \frac{N_{yi} + \alpha}{N_y + \alpha m} \tag{4}$$

where N_{yi} is the number of times feature x_i appears in a sample of class y in the training set, $\alpha = 1$ is the Laplace smoothing factor to prevent zero probability calculations, and N_y is the total count of all features for class y [10].

In practical implementation, probabilities can be very small and their products even smaller (potentially causing underflow). As a result, we take the log-likelihood equivalent to the optimization problem shown in Equation 1 using the expressions in Equation 3. We can easily calculate the posterior probability using its log scale version.

Now that we have our Naive Bayes classifier, we need to see how to build a stronger classifier using boosting. The idea behind boosting is that we start by sampling data points, do a classification on them, get the resulting accuracy, increase the sampling weight for misclassified points and iterate over a given number of iterations by sampling data points with their respective updated weights (misclassified points are more likely to be selected) and finally combine results from the weak learners to form a stronger learner. Thus, it is a sequential learning method. An illustration is shown in Figure 2 [15].

The steps in AdaBoost algorithm are given as follows [9]:

(a) Initialize the weights $\mathbf{p}_1$ (iteration 1) of all the data points $(\mathbf{x}_i, y_i)$ in our dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ using uniform distribution ($p_i = \frac{1}{m}$).

(b) In each iteration, get m samples from training data with replacement using the weights for each data point as the probability distribution, and find $p(\mathbf{x}|y)$ and $p(y)$.

(c) Get the predictions using these probabilities using Equation 1. Calculate the total error e_j for iteration j given as sum of weights of misclassified points:

$$e_j = \sum_{i=1}^{m} \mathbb{1}(\hat{y}_i \neq y_i) \times p_i \tag{5}$$

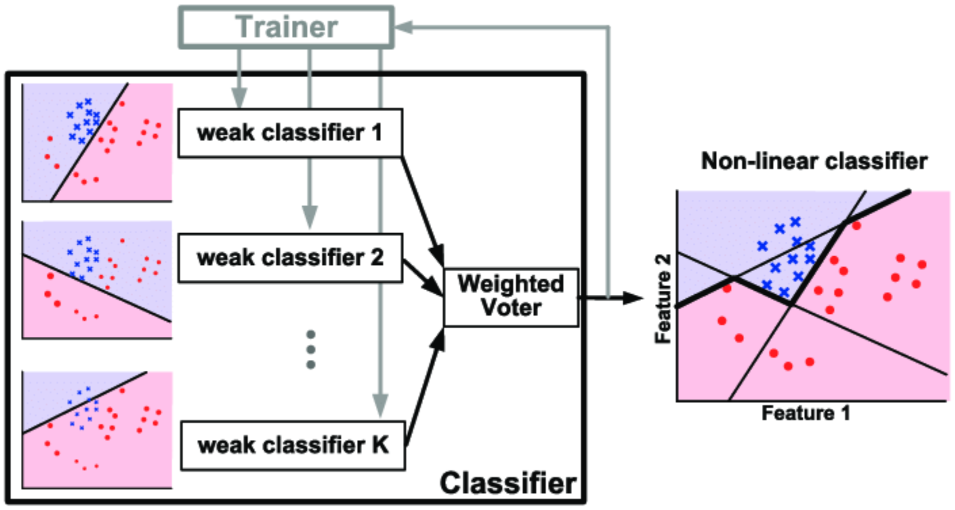

Figure 2: Boosting as combination of weak learners

(d) Calculate what is called the performance of the classifier given by α_j:

$$\alpha_j = \frac{1}{2}\mathbf{ln}\left(\frac{1-e_j}{e_j}\right) \tag{6}$$

Large α_j means good classification and small α_j means poor classification. Thus, using the performance parameter, we can update the weights.

(e) Misclassified points:

$$\mathbf{w}_{j+1} = \mathbf{p}_j \cdot e^{\alpha_j} \tag{7}$$

Properly classified points:

$$\mathbf{w}_{j+1} = \mathbf{p}_j \cdot e^{-\alpha_j} \tag{8}$$

We increase the weights of the wrongly classified records and decrease the weights of the correctly classified records. That is e^{α} is large when the performance α is relatively small (poor classification) and the new weight will be much larger than the old one. And $e^{-\alpha}$ is small when the performance α is relatively large (good classification) and the new weight will be much smaller than the old one. This way, when we choose samples from our training set on the next iteration, we are more likely to choose the misclassified points.

(f) A minor implementation detail here is that these updated weights do not necessarily form a probability mass function (pmf) so we normalize each weight by the sum of all the weights at that iteration. That is:

$$\mathbf{p}_{j+1} = \frac{1}{\mathbb{1}^T\mathbf{w}_{j+1}}\mathbf{w}_{j+1} \tag{9}$$

(g) Keep doing this until you reach the maximum iteration M (i.e. $j+1 = M$). Then finally, we combine the results from all iterations into one final prediction given as:

$$\mathbf{prediction}_{final} = \arg\max_k \sum_{j=1}^{M} \alpha_j \mathbb{1}(\mathbf{\hat{y}_j} = k) \tag{10}$$

where $\hat{y}_j$ is the prediction we got at iteration j.

We can then calculate accuracy using our true label and final prediction (for classification: $accuracy = 1 - misclassification\ rate$)

In Equation 10, the exact formulation can change depending on the type of classification. For our binary classification, we will use the sign of the above expression ($\mathcal{Y} = \{-1, 1\}$) instead of arg max. Also predictions on the validation/test dataset are carried out using the stored performances α, the fitted weak learners of each iteration and applying Equation 10.

3 Method II - Fully Connected Layers

The first model that was implemented was a Adaboost model, which is primarily a machine learning model that does not make use of any convolutional layers or neural networks. The second model we are attempting to build is a basic single layer neural network with fully connected layers. It consists of a set of linear layers, with non-linear activation functions that inter-weave the linear layers. Here each function is a neuron or a more commonly referred to in deep learning methods, a perceptron. The model is called a fully connected model because it performs a dot product operation between a weight vector (w) and the output of the preceding layer which is non-linear due to the activation function. This can be formulated as in Equation 11.

$$\mathbf{y}_{jk(x)} = f(\sum_{i=1}^{n_H} w_{jk}x_i + w_{jo}) \tag{11}$$

where the w's are the various weights and the x is the input and f is the activation function. The model we have used is a basic model with only 2 linear layers. Increasing the depth of the model proved to be futile beyond 2 layers as the maximum accuracy was achieved at two and fluctuated after adding more layers without a clear increase, which suggested saturation in the scaling.

We attempted to use both Stochastic Gradient Descent (SGD) as well Adam as our optimizer function and received better results for SGD. The loss function chosen was the Cross-Entropy Loss. We also did feature engineering of the image to fit the model optimally. The image size that gave us the most optimal result was 32x32. Our model consists of three primary layers:

(a) A linear layer with 1024 features as input and 256 features as output. The bias was set to true.

(b) The activation function, we used the ReLu function as it is the default go to and state of the art option.

(c) Another linear function which takes the first 256 features now passed through the activation function as the input and finally gives the two binary classes- pneumonic or non-pneumonic as the output.

While most image classification models use convolutional layers, we found that our model worked better with fully connected layers. With the usage of convolutional layers, our model would saturate at around 62.5% accuracy after which it seemed to hit a trough that it was unable to get out of even for 40 epochs. Switching to a fully connected model enabled us to observe much better accuracy, but still only in par with the machine learning model. We deduced that the problem with the convolutional model is that it requires a lot more feature engineering in terms of scaling of the model across the depth and resolution to fully achieve the model capacity. Hence we also decided to study a state-of-the-art model that makes use of convolutional layers but also devices a systemic scaling method to achieve the best accuracy from the model.

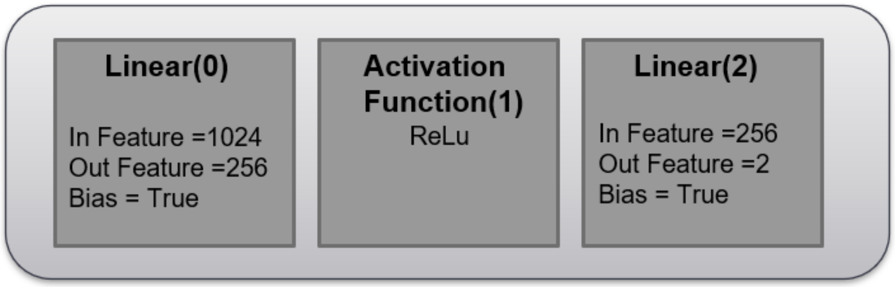

Figure 3: Block Diagram of Fully Connected Layer

4 Method III - State of The Art -EfficientNetV2

Convolutional neural networks are the current novel way for image recognition. But the scaling of convolutional neural networks with respect to network depth, resolution and width directly effects the performance. Hence it is important to device a way to efficiently and uniformly balance the scaling of the model for the most efficient performance. A baseline network is defined by using Neural Architecture Search to define it and is scaled up to obtain a family of models called EfficientNet's which enhance the accuracy and efficiency of convolutional networks. Scaling up the neural network in a principled and accurate manner along length, width and depth to achieve more efficiency is the backbone of EfficientNet. Traditional convolutional networks approach the issue of scaling with arbitrary values. However, EfficientNet implements a profound concept of scaling called compound scaling which uniformly scales the parameters of depth, width and resolution with a set of fixed scaling coefficients. An illustration is shown in Figure 4 [7].

Compound scaling takes a very intuitive approach to scaling of factors. This can be discerned from a basic example. Suppose we increase the resolution of an image, then it needs more layers to be resolved and more of a feature space to be more sensitive and receptive to the changes in the image. If we can afford to use more computational resources then scaling of the network can be done by increasing the depth by α^n, width by β^n and image size by γ^n ,where α, β and γ are the constant coefficients determined through a grid search. The EfficientNet model bases its effectiveness by building on the MobileNet model. While surveying the scaling of current models, we compare them using different resource constraints. ResNet is scaled up or down by adjusting the network depth while WideResNet and MobileNet are scaled usually by network width or the number of channels. Deep ConvNets use model compression as means to reduce model size by trading off between accuracy and efficiency. The challenge to apply Deep ConvNets to larger models that have a much larger design space and more tuning cost remains, as it requires a lot of feature engineering of the depth, kernel type and size to achieve desired efficiency results [12].

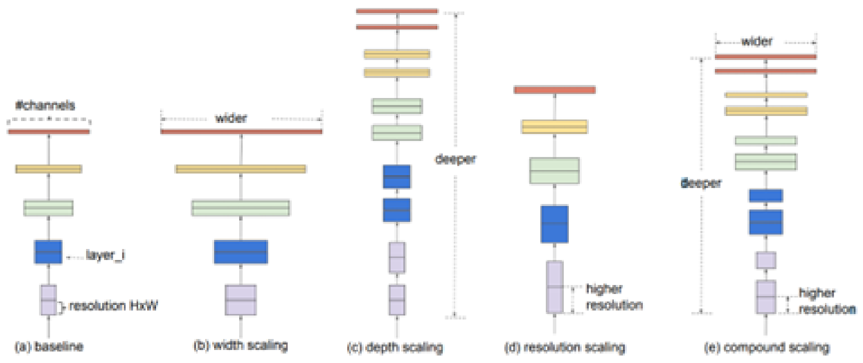

Figure 4: EfficientNet methodologies

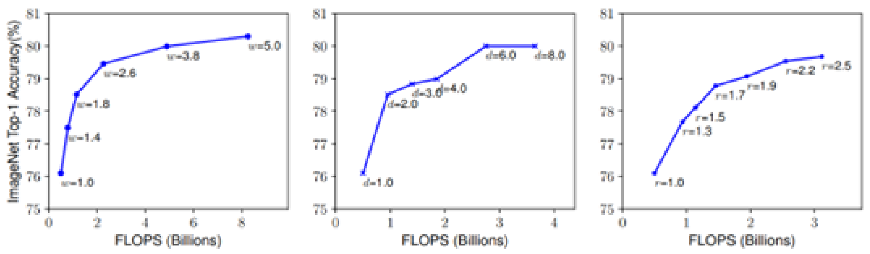

Figure 5: Trends for one dimensional feature scaling

Model scaling needs to be done across different constraints in relation with each other as all the constraints affect the expressivity of the model and the overhead caused by the FLOPS (Floating Point Operations Per Second). Compound model scaling is a primary approach of EfficientNet for empirically studying the scaling of the convolutional networks through the three parameters. Since it is tricky to accommodate the optimal depth, width, and resolution which are all dependent on each other under various constraints, they are usually scaled as one constraint at a time. Depth is scaled by increasing the layers and is most often used in ConvNets. The more the layers added, the more the sensitivity of the model is the general perception. But there exists a trade off of deeper models with the overhead of the complexity, as one would expect. Another common issue of the deeper networks is the training difficulty caused by the vanishing gradient problem. Though there are various ways to combat this, batch normalization being the most commonly used one, there is still a significant decrease in the accuracy of deep neural network and we find that after a certain point adding more layers becomes redundant. Figure 5 shows the visualization of scaling a model with respect to the depth coefficient d which shows a clear saturation at higher depth values.

Let us validate the intuition with a graphical representation. If we scale the network width w without changing the depth and the resolution by keeping both their parameters as 1, we observe that the saturation of the accuracy parameter happens quickly. A network with a deeper and higher resolution width achieves a significantly higher accuracy for the same FLOPS cost. Hence we can provide a backing to our intuition that the scaling parameter needs to be balanced for efficient results. In Figure 6, the d, r, w values determine the scaling of the depth, resolution and width respectively.

Similarly, network width is scaled for smaller size models since they require an extension of the feature space to capture the fine-grained features to easily train the model. But this widening of the network also has a saturation which we see as in Figure 5 for higher w values. The same pattern can be found for image resolution when the image resolutions are scaled up. Networks with objective of higher intelligence like object detection or segmentation have resolutions up to 600x600. However as shown in Figure 6, there is a saturation in this as well. Though the increase in the resolution increases the accuracy after a certain point, the accuracy gain flattens as well. Hence, we can make a primary and important observation: the gain for scaling up the individual dimensions for bigger models diminishes. Therefore, there arises a need for compound scaling. A clear observation can be drawn that the different scaling dimensions cannot be treated as independent quantities. By pure intuition we can see that for images of higher resolution we should increase the network depth so that the larger receptive fields can capture the similar features according to the pixel resolution. Similarly, we should also increase the network width in accordance to the resolution. Hence there needs to be a balance in the scaling dimensions rather than the one-dimensional scaling of the features.

The compound scaling method uses a compound coefficient (denoted by the variable ϕ). The variables used in it are then:

α - Depth parameter
β - Width parameter
γ - Resolution parameter
ϕ - Scaling parameter

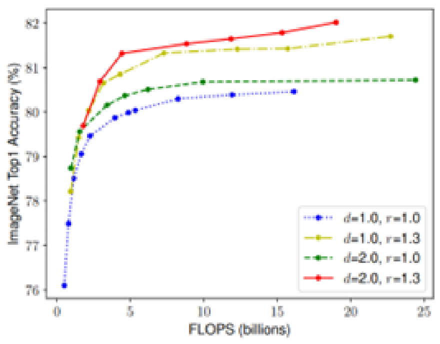

Figure 6: Trends for compound scaling

The training of the model is done in two stages.

Stage one:

1. Fix ϕ =1
2. Set $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$
3. Set α>=1, β>=1, γ>=1 as the constraint, obtain α, β and γ values for the simple model with grid search
4. Take $\alpha = 1.2$, $\beta = 1.1$, $\gamma = 1.15$ for the V2 model. This is B0 under the first constraint.

Stage two:

1. Change ϕ form 2-7. This gives B1-B7 values.

For every increase in the scaling parameter there is an increase in accuracy.

There is a possibility of achieving better performance by searching for α, β and γ directly around a large model but the cost trade off proves to be unworthy. Therefore, doing a search once on the small baseline network and then scaling to higher coefficients is the most efficient approach.

EfficientNetV2 is the latest in the EfficientNet family and solves the problems of training speed and parameter efficiency of the previous models. In this model, we optimize both training speed and parameter efficiency jointly by using training aware Neural Network Architecture and scaling. Hence the training speed of this is much more efficient compared to other state-of-the-art models. Since the training speed can be sped up by increasing the size of the image progressively during training but also trades off the speed with a steep decrease in accuracy, an improved method of progressive learning is suggested that adaptively adjusts the regularization with proportion to the image size [13].

EfficientNet's method of making use of large image size results in significant memory usage. This leads to training these models to train with smaller batch size to accommodate the fixed total memory of the GPU/TPU and this inversely affects the training time. A seemingly simple fix could be to use a smaller image size for training and a larger one for inference. This enables the usage of larger batch size and lesser computational overheads and hence decreasing the training speeds. This can also lead to slight increase in the accuracy. Another common issue in the EfficientNet is due to the extensive usage of depth wise convolution. They have fewer parameters but are unable to make use of the modern accelerator. A fused MBConv is hence proposed to better utilize mobile or server accelerator's and replace higher order depth wise convolution to that of a lower order.

5 Results

5.1 Boosting

The data was organized in folders with respective labels. The feature engineering aspect here happens in loading the images, resizing each image to custom dimension, saving the resized images, and organizing the training, validation and test data in the form of $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ where $\mathbf{x}_i \in \{0, 1, ...255\}^d$ is a 1-D reshaped (flattened) version of the image (later rescaled to $[0, 1]$, original dimension $\sqrt{d} \times \sqrt{d}$) and y_i are the labels for each image.

The algorithm was applied for 100 iterations (we found this to be reasonable amount of time to run and debug the code). We tried image dimensions of 40x40, 100x100 and 128x128. The results are each shown in Tables 1, 2 and 3 respectively (accuracy to the nearest integer).

5.2 Fully Connected Layer

One of the deep learning algorithms we tested on our dataset was a simple neural network consisting of two linear layers, with a Rectified Linear Unit function between the two dense layers. We chose to use fully connected layers instead of convolutional layers because each pixel in an image from our dataset is related to all of the other pixels in that image. To make the training feasible for a fully connected neural network, we chose to resize our images to a relatively small 32x32 matrix.

Table 1: Results for 40x40 images

Iterations = 100	Accuracy (%)		
	Training	Validation	Testing
Single Gaussian Naïve Bayes	85	56	71
AdaBoost Gaussian Naïve Bayes	68	56	72
Single Multinomial Naïve Bayes	86	75	75
AdaBoost Multinomial Naïve Bayes	91	81	79
AdaBoost decision tree (scikit)	99	75	75

Table 2: Results for 100x100 images

Iterations = 100	Accuracy (%)		
	Training	Validation	Testing
Single Gaussian Naïve Bayes	85	69	72
AdaBoost Gaussian Naïve Bayes	67	69	71
Single Multinomial Naïve Bayes	85	75	74
AdaBoost Multinomial Naïve Bayes	90	81	76
AdaBoost decision tree (scikit)	99	69	73

Table 3: Results for 128x128 images

Iterations = 100	Accuracy (%)		
	Training	Validation	Testing
Single Gaussian Naïve Bayes	85	69	72
AdaBoost Gaussian Naïve Bayes	67	69	72
Single Multinomial Naïve Bayes	85	75	75
AdaBoost Multinomial Naïve Bayes	87	75	77
AdaBoost decision tree (scikit)	99	81	74

Table 4: Results for 32x32 images (Iterations = 1000)

	Loss	Testing accuracy
Two Fully Connected Layers	0.46	76.2
Convolutional Layers	0.66	65.2

Table 5: Results for different Epochs for EfficientNetV2

Epochs	Accuracy (%)			
	Training loss	Validation loss	Training accuracy	Validation accuracy
4	19	82	92	56
5	18	55	92	68
6	16	52	93	67

The high level view of our simple neural network is two dense layers separated by a ReLU layer. After making these adjustments to our architecture and dataset, we can compare the performance of a network with convolutional layers with that of a dense, fully connected neural network. As can be seen in Table 4, the fully connected, linear model outperforms the convolutional network.

5.3 EfficientNetV2

The efficiency of the state-of-the-art efficient net model is much better than both our fully connected and AdaBoost model. This is because the manual feature engineering required for this model is the least and the number of layers as well as the width and resolution are all adjusted to best capture the difference in the image. Even though the performance of this is the best compared to the other models it also took us the most time to execute. Hence there is an obvious trade off with the accuracy and time/resource of training. We trained the model for 6 epochs as shown in Table 5 and then obtained the testing accuracy.

Table 6: Results for Testing Loss and Accuracy for EfficientNetV2

	Epoch	Loss	Accuracy
Training	10	0.16	0.93
Validation	10	0.50	0.75
Testing	1	0.28	0.88

As we can see in Table 6, there is a clear increase in the training accuracy and decrease in the training loss which is an expected trend. Training for a higher number of epochs will give us higher accuracy and lower loss but will also take more time to converge.

6 Conclusions

The multinomial modelling seems to be a reasonable fit in this case. In contrast, the boosting based on Gaussian model diminishes the accuracy on the training set and barely affects the accuracy on the validation and testing set. One explanation could be that the Gaussian model is not a good guess of the distribution of the dataset we have. Based on these results, we do not recommend using one type of boosting but rather compare multiple approaches and decide the better options. In general, it seems 75% is about as good as we can get with the testing set. We surmise this could be a result of the distribution of the dataset we used. In contrast, the scikit AdaBoost classifier based on decision trees seems to work well on training data but the validation and testing accuracies are comparable to the other weak learners. The poor performance on the validation and testing sets could be a further indication of the fact that the validation and testing sets are from a different distribution than the training set. Another theoretically expected and empirically proven explanation is that a decision tree is a better weak learner than the given models of Naive Bayes in boosting. Boosting works better when the base learner is weaker and the base learner is characterized by high variance and low bias. This means every iteration is more likely to lead to different classification results such that when combined, we get to address all the data points more accurately. Naive Bayes is a low variance, high bias stable classifier that changes in the data points barely affects the results [14]. Hence, the final prediction is more of an averaging of results over several iterations.

Further works on this project in terms of boosting could be implementing logistic regression (also low variance, high bias classifier) as the weak learner and compare it with the Naive Bayes models (as a comparision of discriminative vs generative learning) but we still do not expect a better result. Another classifier option that would likely give good results is K-means neighbor as a weak learner (another high variance, low bias classifier). Another direction could be trying multiclass classification on a different dataset. Finally, we can also try using a different boosting algorithm such as Gradient Boosting and compare it to AdaBoost.

Our first attempt in the DL approach was to make a convolutional neural network. The image sizes taken in the beginning were 264x264. The final loss function used is the cross entropy loss, which is considered as a standard for deep learning problems and the final optimizer used was SGD. After testing the different types of loss functions and optimizers and resizing the images from sizes 264x264 to 32x32 we came to the conclusion that our convolutional neural network was not able to perform well. We deduced from our experiment that the reason behind it might be because it is really difficult to tune a CNN, and that the level of difficulty to scale it is complex. To improve our level of accuracy we traced back and implemented a simple fully connected layer model. It was a simple model consisting two linear layer and an activation layer of ReLU activation function. The loss function used was cross entropy loss and the optimizer as SGD. We used a smaller size of images (32x32) and trained the model for 1000 iterations. We were able to see a significant improvement in the accuracy on the test data set. Our conclusion for this stage of experiment was that a simple fully connected layer model was able to perform better on our dataset rather than a complicated convolutional neural network.

The main drawback of the convolutional model of deep nueral networks is the difficulties incurred while trying to scale them up. Hence we wanted to explore a model that approaches the scaling up of the convolutional model in a systemic and efficient way. This led us to our last leg of implementation , the state-of-the-art EfficientNet model. It devices a way to compoundly scale up the convolutional layers.The version two model that we have implemented also improves the training speed and the

Table 7: Comparison Between The Three Models

	Speed	Accuracy	Complexity
Boosting	25 min (CPU)	0.76	Medium
Fully Connected	15 min (GPU)	0.7612	Low
EfficientNet	2.5 hrs (GPU)	0.88	High

parameter efficiency as well as systemically scale up the model in accordance with depth, width and resolution. This can be seen as the results obtained by this model is the best of the three. It gave us a very high training and test accuracy and minimal loss as well. Hence the performance of the state of the art model was in accordance with our intuition that the reason for the issues of the performance of our initial convolutional model is the scaling and tuning of the layers that is mitigated in the EfficientNet model. But there still exists issues with time of convergence of the algorithm and the memory overhead usage.

Overall, comparing the results of the three models as shown in Table 7, we see that while the complexity of the boosting algorithm and the fully connected layer are medium and low, the time taken to execute them are in a similar range. The state-of-the-art EfficientNet model takes the most time as it is the most complex but also gives the most accuracy.

(A quick remark here is that there is more explanation towards the performance of ML algorithms compared to DL because ML is more supported by theory while DL has more of a "black box" nature).

References

[1] S. Ansari, I. Shafi, A. Ansari, J. Ahmad, and S. I. Shah, “Diagnosis of liver disease induced by hepatitis virus using artificial neural networks,” in *2011 IEEE 14th international multitopic conference*. IEEE, 2011, pp. 8–12.

[2] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[3] C. Bustamante, L. Garrido, and R. Soto, “Comparing fuzzy naive bayes and gaussian naive bayes for decision making in robocup 3d,” in *Mexican International Conference on Artificial Intelligence*. Springer, 2006, pp. 237–247.

[4] N. I. A. Dabowsa, N. M. Amaitik, A. M. Maatuk, and S. A. Aljawarneh, “A hybrid intelligent system for skin disease diagnosis,” in *2017 International Conference on Engineering and Technology (ICET)*. IEEE, 2017, pp. 1–6.

[5] M. A. Khan, “An iot framework for heart disease prediction based on mdcnn classifier,” *IEEE Access*, vol. 8, pp. 34 717–34 727, 2020.

[6] P. Mooney, “Chest x-ray images (pneumonia) | kaggle,” https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia, (Accessed on 05/04/2022).

[7] A. Nain, “Efficientnet: Rethinking model scaling for convolutional neural networks | medium,” https://medium.com/@nainaakash012/efficientnet-rethinking-model-scaling-for-convolutional-neural-networks-92941c5bfb95, June 2019, (Accessed on 05/04/2022).

[8] M. Owais, M. Arsalan, J. Choi, T. Mahmood, and K. R. Park, “Artificial intelligence-based classification of multiple gastrointestinal diseases using endoscopy videos for clinical diagnosis,” *Journal of clinical medicine*, vol. 8, no. 7, p. 986, 2019.

[9] R. Rojas *et al.*, “Adaboost and the super bowl of classifiers a tutorial introduction to adaptive boosting,” *Freie University, Berlin, Tech. Rep*, 2009.

[10] scikit-learn Developers, “1.9. naive bayes — scikit-learn 1.0.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html

[11] A. T. Society, “top-pneumonia-facts.pdf,” https://www.thoracic.org/patients/patient-resources/resources/top-pneumonia-facts.pdf, (Accessed on 05/04/2022).

[12] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[13] M. Tan and Q. Le, “Efficientnetv2: Smaller models and faster training,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 096–10 106.

[14] K. M. Ting and Z. Zheng, “A study of adaboost with naive bayesian classifiers: Weakness and improvement,” *Computational Intelligence*, vol. 19, no. 2, pp. 186–200, 2003.

[15] Z. Wang, J. Zhang, and N. Verma, “Realizing low-energy classification systems by implementing matrix multiplication directly within an adc,” *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 6, pp. 825–837, 2015.