Classification of X-Ray Chest Images to Detect Pneumonia

Setareh Roshan
Computer Department
Shahid Rajaee Teacher Training University
Tehran, Iran
setare.roshan1996@gmail.com

***Abstract*— Recent deep learning methods provide an effective way to construct an end-to-end model that can compute final classification labels with the raw pixels of medical images. Convolutional neural network (CNN) is a class of deep learning neural networks. CNNs represent a huge breakthrough in image recognition. In this article, we used CNN in order to recognizing Pneumonia infection from Xray images, also we extracted features from CNN and tried to increase accuracy by pattern recognition methods. We reached to 80% accuracy on test data.**

***Keywords*—cnn, pneumonia, classification infection recognition, Naïve Bayes, radial basis function, pattern recognition, decision tree**

I. Introduction

In the past few decades, Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks.

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

CNN application has an extensive use in medical image classifying. In this article we used [1] dataset which consist of normal and pneumonia Xray chest images. Pneumonia is an infection that inflames your lungs' air sacs (alveoli). The air sacs may fill up with fluid or pus, causing symptoms such as a cough, fever, chills and trouble breathing.

II. Methods

For classifying this issue, I used CNNs, RBF neural network, bayes classifier, and decision tree. Each is briefly explained in bellow.

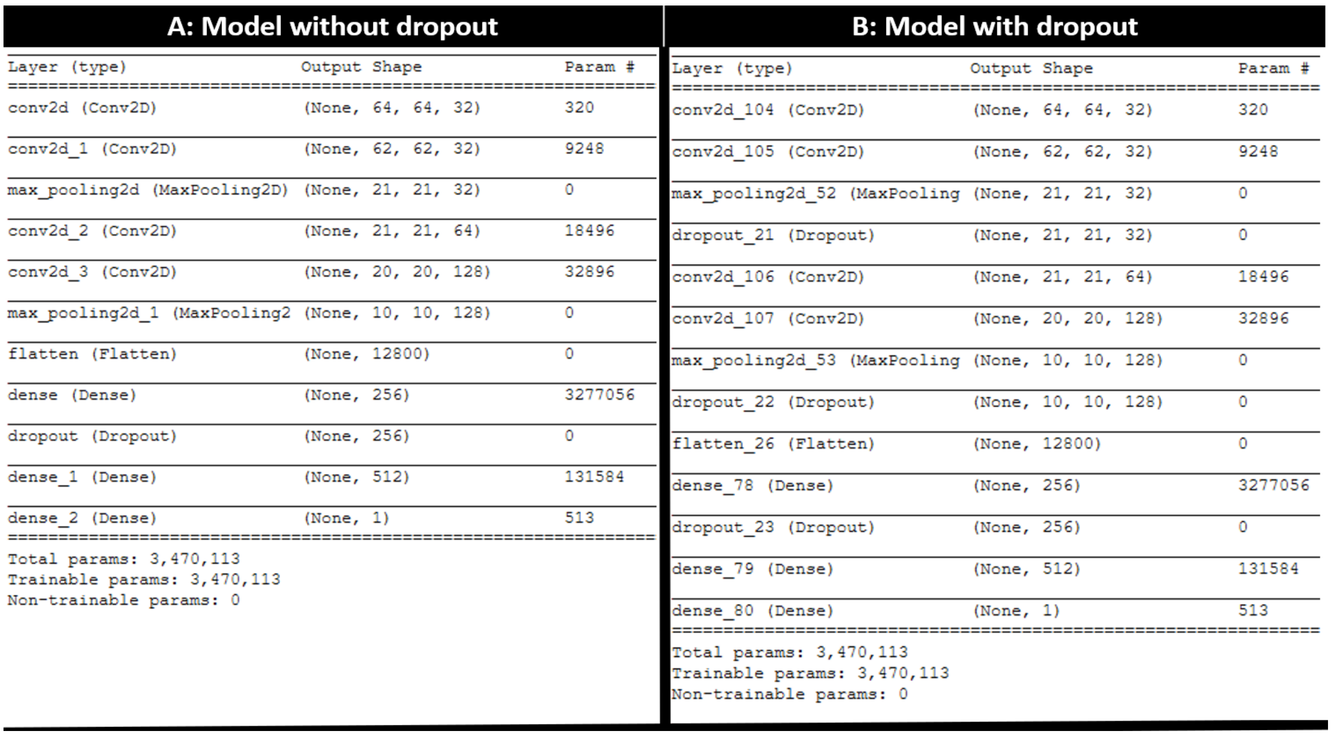
A. CNN ARCHITECTURE

The first layer in a CNN network is the convolution layer, which is the core building block and does most of the computational heavy lifting. Data or imaged is convolved using filters or kernels (or sometimes referred to as a neuron or a kernel). Filters are small units that we apply across the data through a sliding window. The depth of the image is the same as the input. Second is the activation layer which applies the Relu (Rectified Linear Unit), in this step we apply the rectifier function to increase non-linearity in the CNN. Images are made of different objects that are not linear to each other. Third, is the pooling layer, which involves down sampling of features. This layer uses a sliding window and pool (average pool or max pool) values in that window. At last, it uses dense layers which are the same as MLP. CNN uses backpropagation for learning. In implemented model Adam optimizer and for loss function binary cross entropy were used [2].

Here, we present two CNN architectures- one with a dropout layer and another without a dropout layer. Both CNN consist of convolution layer, max pooling and a classification layer. A series of convolution and max-pooling layers act as a feature extractor that is divided into two parts.

The first part consists of two Convolution layers with 32 - 32 units each along with a max-pooling layer of size 3 × 3 and a Relu activator. While the other also has two Convolution layers but with 64 and 128 units respectively along with a max-pooling layer of size 2 × 2 and a Relu activator. Relu layer introduces nonlinearity to the model. Features extracted from the feature extractor part of the CNN are given as input to the dense layer which classifies the image. Before feeding the extracted features to the dense layer, a flatten layer a used. As the dense layer takes 1-Dimensional input, hence, flatten layer flattens the feature data and gives a 1-Dimensional output which is fed to the dense layer (Figure 2). While training a CNN it might be possible that output through a certain layer is more dependent on a few selected neural units. To reduce this dependency and prevent overfitting, the concept of dropout is introduced. During training, in each epoch, a neuron is momentarily dropped with a dropout probability p. Due to this, all the inputs and outputs to this neuron become disabled in the current epoch which results in some loss of data enhancing regularization in the model so that at times it would predict with much higher accuracy. The dropped-out neurons are resampled with probability p at every training step, so a dropped-out neuron at one step may become active in the next step. A dropout probability of 0.5, corresponds to 50% of the neurons being dropped out. In the proposed CNN architecture with dropout layer, we apply dropout at two places. First, it is applied at the feature extractor part i.e. after convolution and max-pool layers. As, the convolutional layers have not too many parameters, hence overfitting is not an issue in this case. Hence, here we take a low drop probability of 0.2. Second, it is used at dense layer. With dropout in this layer, overfitting is prevented (Figure 1).

This work is inspired by: https://ieeexplore.ieee.org/abstract/document/9057809



A: Model without dropout

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        320

conv2d_1 (Conv2D)            (None, 62, 62, 32)        9248

max_pooling2d (MaxPooling2D) (None, 21, 21, 32)        0

conv2d_2 (Conv2D)            (None, 21, 21, 64)        18496

conv2d_3 (Conv2D)            (None, 20, 20, 128)       32896

max_pooling2d_1 (MaxPooling2 (None, 10, 10, 128)       0

flatten (Flatten)            (None, 12800)             0

dense (Dense)                (None, 256)               3277056

dropout (Dropout)            (None, 256)               0

dense_1 (Dense)              (None, 512)               131584

dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,470,113
Trainable params: 3,470,113
Non-trainable params: 0
```

B: Model with dropout

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_104 (Conv2D)          (None, 64, 64, 32)        320

conv2d_105 (Conv2D)          (None, 62, 62, 32)        9248

max_pooling2d_52 (MaxPooling (None, 21, 21, 32)        0

dropout_21 (Dropout)         (None, 21, 21, 32)        0

conv2d_106 (Conv2D)          (None, 21, 21, 64)        18496

conv2d_107 (Conv2D)          (None, 20, 20, 128)       32896

max_pooling2d_53 (MaxPooling (None, 10, 10, 128)       0

dropout_22 (Dropout)         (None, 10, 10, 128)       0

flatten_26 (Flatten)         (None, 12800)             0

dense_78 (Dense)             (None, 256)               3277056

dropout_23 (Dropout)         (None, 256)               0

dense_79 (Dense)             (None, 512)               131584

dense_80 (Dense)             (None, 1)                 513
=================================================================
Total params: 3,470,113
Trainable params: 3,470,113
Non-trainable params: 0
```

Figure 1. Model Parameters. A. shows model without dropout. B. shows model with dropout. Except the dropout term there is absolutely no difference in two models.

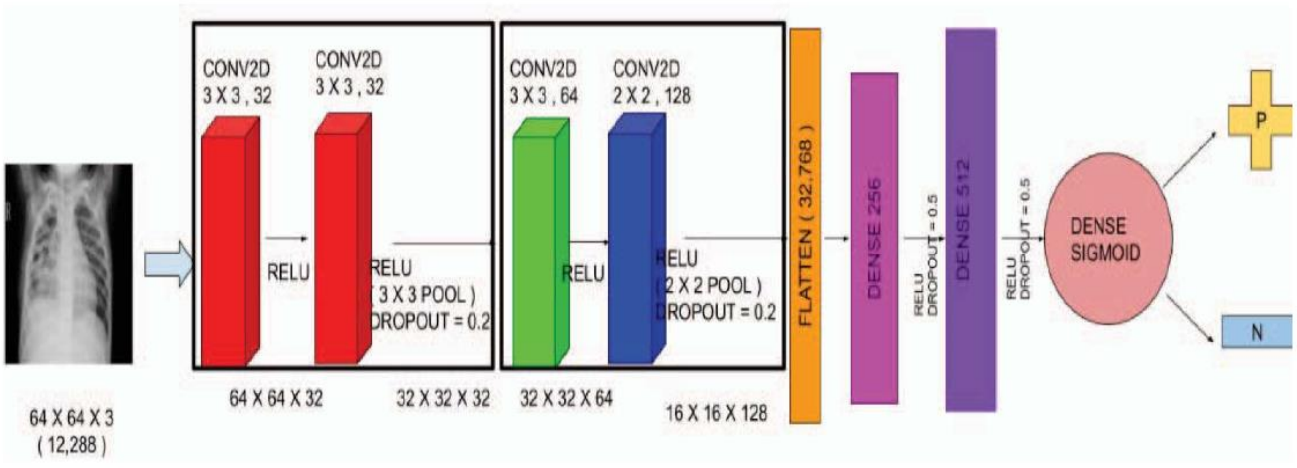


Figure 2 Architecture of CNN. Model consists of 2 convolution layers then a pool layer both conv layers have Relu activation function. Second box has the same architecte. Then 3 dense layers which two first layers have Relu activation functions. For a single neuron in last layer we use sigmoid activation function in order to predict 0 or 1 labels.

B. *Decision Tree*

Decision tree is learning method. The goal is to create a model that predicts the value of a target variable based on several input variables. The goal is to create a model that predicts the value of a target variable based on several input variables. A tree is built by splitting the source set, constituting the root node of the tree, into subsets—which constitute the successor children. The splitting is based on a set of splitting rules based on classification features.

It split the dataset based on purity and impurity. Gini index calculate impurity of the separation:

$$Gini(t) = 1 - \sum_j (p(c_j|t))^2$$

Equation 1

where, $p(c_j|t)$ is the probability of class c_j given node t. When a node p is split into k partitions, the quality of split is computed as follows:

$$Gini_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

Equation 2

where, n_i = number of records at t partition i, n = number of records at node p.

Another parameter that can measure the information is Entropy:

$$Entropy(t) = -\sum_j p(j|t)\log(j|t)$$

Equation 3

where, $p(j|t)$ is the relative frequency of class j at node t. Entropy is used in Information Gain (IG) to determine the best feature:

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{n} \frac{n_i}{n} Entropy(i)\right)$$

Equation 4

parent node, p is split into k partition; n_i number of records in partition i.

C. Bayes Classifier

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification. Gaussian Naïve Bayes can be calculated by Equation 5:

$$g_i = P(w_i|x)$$

Equation 5

In which g_i is the probability of belonging to a class, and we will calculate it as follows:

$$g_i = \frac{-d}{2}\log(2\pi) - \frac{1}{2}\log\left|\sum i\right| - \frac{1}{2}(x-\mu_i)^T \sum_i^{-1}(x-\mu_i) + \log P(w_i)$$

Equation 6

D. K-Means

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. k-means clustering minimizes within-cluster variances.

First, we started with k random values, that their dimensions are as many as input sample's dimensions. This k values can be selected from the training set. In each epoch, we've updated the k values, which they were our cluster's centers. We defined a new center for each cluster by averaging the amounts of its members. Elbow method were used to find the for optimum k. Elbow method uses sum of square of samples in an cluster to their center and find optimal when the SEE remain unchanged.

E. Radial Basis Function

The training of the RBF model is terminated once the calculated error reached the desired values or number of training iterations already was completed. An RBF network with a specific number of nodes in its hidden layer is chosen. A Gaussian function Equation 7 is used as the transfer function in computational units [3].

$$hidden\ layer\ output_j = \exp\left(-\frac{||sample - mean_{cluster_i}||}{2\sigma^2_{cluster_i}}\right)$$

Equation 7

For each sample in training set we compute its hidden layer output then we will compute output like we did in MLP. In computing hidden layer output, we don't have weights. In contrast in output layer, we have weights (random initial weights). For computing output, we use Equation 6:

$$output_z = 1/(1 + \exp(-hidden\ layer\ output_j * weight_{z,j}))$$

Equation 8

For each $output_z$ we should multiple all hidden layer output that it's connected to $output_z$ with their weights and perform a sigmoid function on them. At last, we will compute an error (for each iteration) then with this error we will be able to update the weights (Equation 9).

$$new\ weight_{z,k} = old\ weight_{z,j} + \eta * error_z * (hidden\ layer\ output_j - output_z)$$

Equation 9

Which η is learning rate.

F. Dataset

The dataset employed in this work is picked from Kaggle [1]. It consists of 5863 images of chest X-ray. The dataset is split into three portions namely training dataset, validation dataset and testing dataset. The details of number of images under each part is shown in Table I, obviously data is unbalanced so 1341 samples from 3875 samples and I've done this for validation dataset but not tests.

Initially, the images in the dataset are of varying sizes. However, all the input images given to CNN should be of same size. To solve this problem, all the images in the dataset are resized to 64 × 64. To avoid overfitting and to enhance the generalization capabilities of the proposed CNN architectures, various in-place data augmentation techniques such as rescaling, flipping etc. are used in this paper. Use of this type of data augmentation ensures that our CNN see new variations of data at each and every epoch during training. The various data augmentation techniques used in this work are listed in Table II, also some samples and variated ones are shown in Figure 3.

Table I Dataset Information

Dataset	Number of Normal Images	Number of Pneumonia Images
Training set	1341	1341/3875
Validation set	72	72/119

Testing set	170	279

Table II. Data Augmentation Techniques

Data Augmentation Techniques	Values
Rescale	1.0/255
Rotation Range	20
Zoom Range	0.2
Width Shift Range	0.2
Height Shift Range	0.2
Shear Range	0.2
Horizontal Flip	True
Fill Mode	Nearest

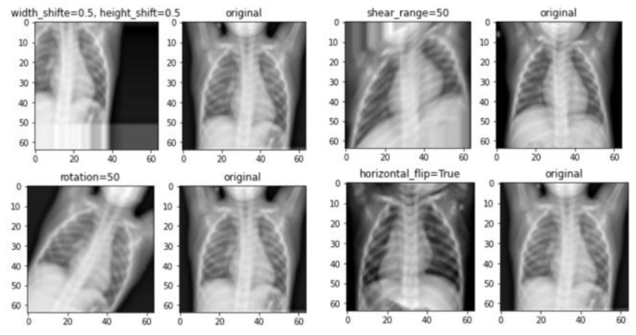


Figure 3. Variated Images. Four original and variated images are shown. Each image contains its information above itself.

III. Experimental Results

First CNN is trained with and without dropouts contrary to my expectation with dropout CNN performance wasn't as well as without it (Figure 4). After training, I performed a global pooling before flatten layer and extracted 128 features. I used these features for training other classifiers. For testing classifiers, I performed this method on test data and 128 features were extracted.

According to Figure 4 without data generator and without dropout performance was better, but I used features from data generator trained network but without dropout.

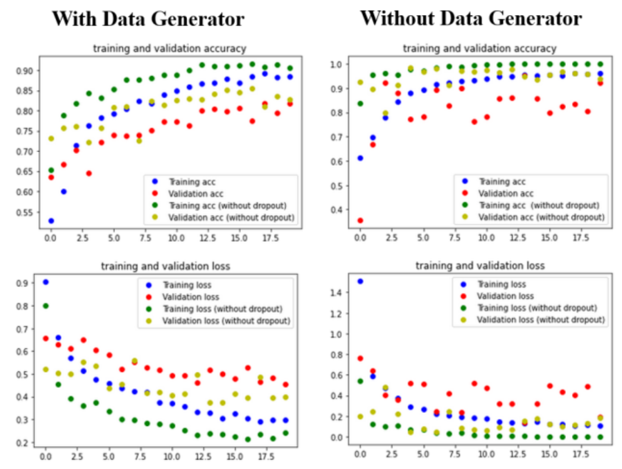


Figure 4. train performance diagrams. All y axes are performance and all of x axes are number of epochs.

On test data accuracy and loss in Figure 5 are reported. I used network which trained by variated data with and without dropout. As we expected (like training phase) test performance for network without dropout is better.

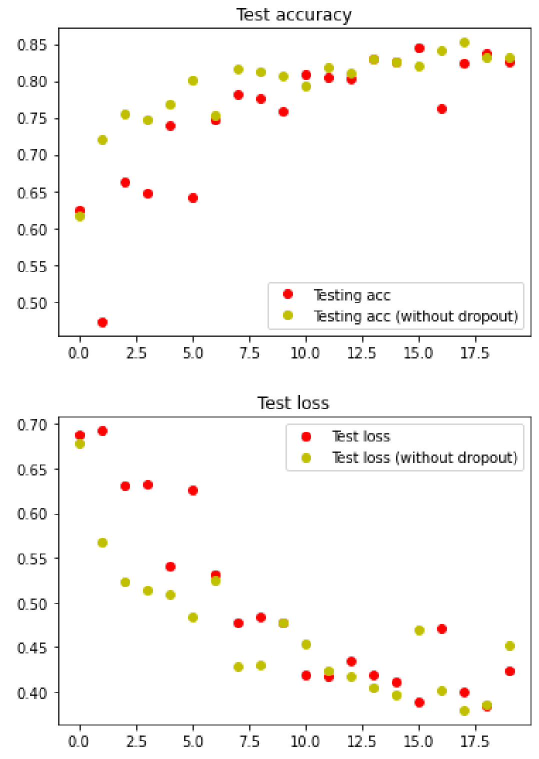


Figure 5. test performance diagrams. All y axes are performance and all of x axes are number of epochs.

In k-mean clustering we should find an optimal k (number of clusters) using elbow method (Figure 6). Other values after k=10 haven't changed much, so optimal k is 10. After optimum k was found we considered centers of RBF kernel, same centers in k-means and variances of each cluster considered as variance in RBF multiplied by 5. Because of small variance in RBF in first try it didn't converge well (original variances). I considered RBF centers in between largest and smallest samples but distance in RBF was so large and it didn't converge either

Table III RBF Parameters

parameters	values
centers	10 centers from k-means
Variance	Variance in k-means * 5
Initial weights	Random from uniform distribution
Learning rate	0.001
Activation function	sigmoid
Updating weights	Delta rule
epochs	350

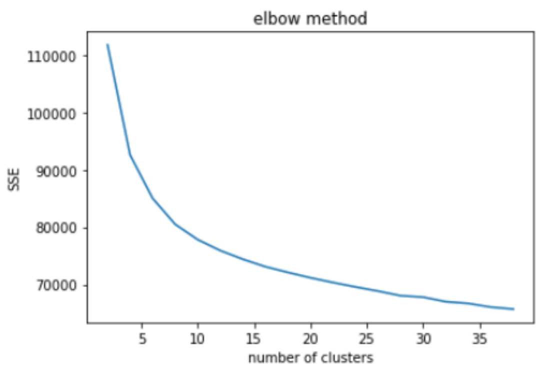


Figure 6 represents elbow method on K-Means clustering. Horizontal line shows number of clusters (k) and vertical line shows the maximum distance in each k.

. Therefore, best performance was with parameters in Table I. performance per epochs diagram is reported in Figure 7. After 350 epochs validation accuracy is decreased so best performance for this data is between 65 and 70 percent.

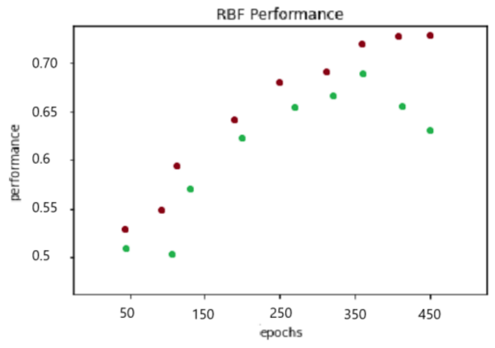


Figure 7. RBF convergence. Green dots are validation data accuracy and darkred dots are train data accuracy.

I also trained decision tree classifier with trained data (feature extracted ones). Decision tree has two parameters gini and entropy so I compered these two on data. as you can see in Figure 8 entropy (yellow dots) had better performance in 20 trials.

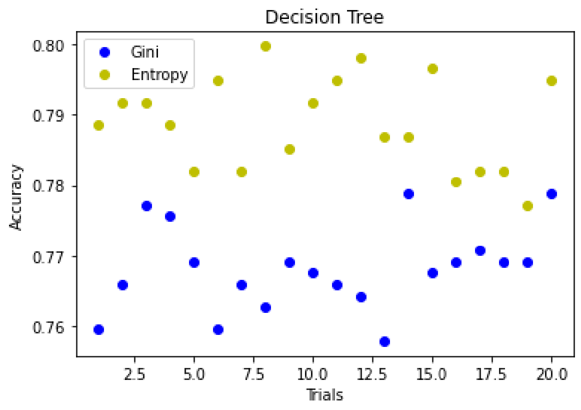


Figure 8. Decision Tree Performance. yellow dots show performance of decision tree which used entropy for separating features and blue dots shows this phenomenon for gini. In all 20 trials entropy had better performance on data.

Lastly, I compared DT with entropy performance with Naïve Bayes performance. Decision Tree had better performance according to Figure 9 yellow dots which are DT are better than blue dots (NB)

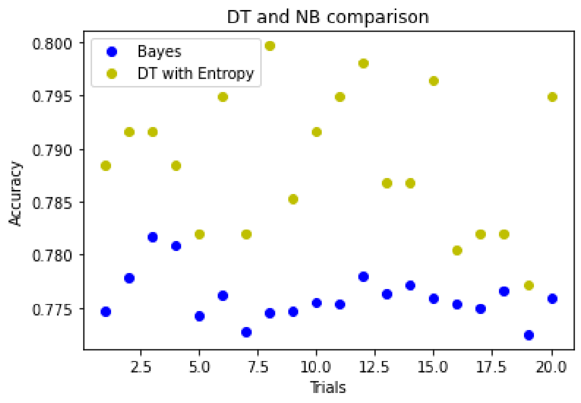


Figure 9. Decision Tree and Naïve Bayes Performance. yellow dots show performance of decision tree which used entropy for separating features and blue dots shows Naïve Bayes performance.

IV. Conclusion

We can conclude from these series of studies that Decision Tree has the best performance on this data. You can compare results from different methods that we used in this study (Table IV).

Table IV Performance Comparison

Algorithm	Performance
RBF	69%
Decision Tree	79%
Naïve Bayes	77%

V. References

[1]. https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

[2]. https://medium.datadriveninvestor.com/introduction-to-how-cnns-work-77e0e4cde99b

[3]. Buhmann, Martin Dietrich. (2003). Radial basis functions : theory and implementations. *Cambridge University Press.*