Bilkent University

CS464 - Introduction to Machine Learning

Project Final Report

Pneumonia Classifier

Section 1: Group 5

Contributors:

Ahmet Sayan - 21903426

Arda Erkan - 22103623

Bahar Özkırlı - 22102773

Damla Uçar - 22103276

Eren Keskin - 22002151

December 28, 2024

Contents

1 Introduction

In the medical field, crucial aspects of the treatments include accurate, early, and fast diagnosis. Advancements in imaging technology have transformed medical practice by providing detailed insights into a patient's condition through techniques such as X-rays, MRIs, and CT scans. These techniques are widely used to identify anomalies; however, interpreting these images may pose challenges regarding time constraints and human analysis. Integrating computational tools of machine learning into this analysis process offers significant potential to enhance the precision, speed, and consistency of diagnosis.

Our approach in this project includes various powerful tools offered by machine learning to address the challenges of such medical image analysis. K-Nearest Neighbor (KNN), one of the three basic methods used in this field, classifies a data point according to the class of its nearest neighbours. It is a simple and effective method, but its computational cost is high in large data sets. Support Vector Machines (SVM) perform linear and non-linear classification by creating hyperplanes that separate data with the largest boundary. It is powerful on small and high-dimensional datasets. Random Forests, on the other hand, create many decision trees and determine the outcome by majority vote; it is resistant to overfitting and does well on complex data sets. In addition to these traditional models, we implemented a deep learning model called Convolutional Neural Network (CNN). By the utilization of these tools, we examined which methods yield more accurate results in the context of image analysis and identified their strengths and limitations in this field.

1.1 Project Description

Pneumonia in children is an important cause of child deaths worldwide. About 2 million children under the age of five die from pneumonia each year, according to the World Health Organization (WHO), which consistently estimates it to be the leading cause of childhood mortality [1]. Pneumonia causes lung inflammation, usually due to bacterial, viral or fungal infections, and if not diagnosed early, it can cause serious health problems or even death. This project aims to use machine learning methods to detect pneumonia in children in the early stages. A classification model that can determine the presence of pneumonia will be developed with data obtained from patients' medical images (chest X-rays, etc.). Thus, the disease will be detected at an early stage, health professionals and families will be informed quickly, and the health of children will be protected by starting the treatment process on time. This solution aims to reduce pneumonia-related complications in children and promote early intervention.

1.2 Dataset Analysis

The dataset consists of chest X-ray images of pediatric patients aged one to five years from Guangzhou Women's and Children's Hospital. All chest X-ray images were taken as part of the patient's routine clinical care [2].

On an X-ray image, a healthy lung appears more black (air-filled), while infected or inflamed areas in the lung with pneumonia appear in shades of white or grey. This reflects the condensation and fluid accumulation that pneumonia creates in lung tissue.

As a result, to detect pneumonia on an X-ray image, attention should be paid to white areas (opacity), condensations, fluid accumulation and irregular densities in certain lung

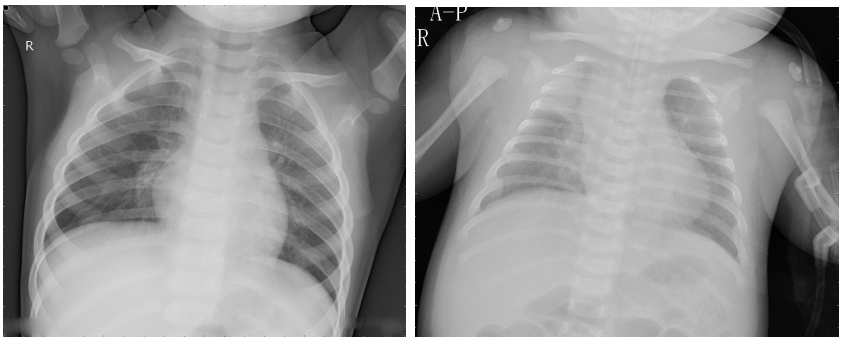

Figure 1: The examples of chest X-rays of a normal (left) and a sick (right) person.

areas. These symptoms indicate the presence and severity of pneumonia. These findings can indicate the presence and vulnerability of pneumonia. Figure 1 (left) shows the chest X-ray of a healthy child, and Figure 1 (right) shows the chest X-ray of a child with pneumonia. It can be clearly seen that in Figure 1, white or grey areas are more prominent and widespread, which is caused by pneumonia.

The dataset is divided into three main folders (train, test, val), with two subfolders in each folder: Pneumonia (pneumonia) and Normal (robust). There are 5,863 X-ray images in total, which are divided into categories to help diagnose pneumonia in children. These images are chest X-rays taken from an anterior-posterior angle and are divided into 2 categories: Pneumonia and Normal. In the training data set, there are 3,875 images characterized as depicting pneumonia and 1,341 normal. The test folder contains 390 pneumonia images and 234 normal images. For validation, there are 8 images for both categories. Since the number of validation images are too low, we transferred images from the test data to the validation folder to enhance the validation/hyperparameter tuning process. 100 images were reallocated from the test dataset for each category, normal and pneumonia. Therefore, there are 290 pneumonia and 134 normal images left in the test folder.

Before analysing the images, poor-quality or unreadable X-rays were removed for quality control. Then, all diagnoses were examined by two specialist physicians and made suitable for the training of the AI system. Additionally, the evaluation set was also checked by a third expert to avoid any evaluation errors.

1.3 Coding Environment

In our project, conducted primarily in Google Colab, we utilized a set of libraries for data manipulation, image processing, machine learning, deep learning, and visualization.

For data manipulation and analysis, we used NumPy and Pandas. NumPy was particularly useful for array operations and matrix computations, while Pandas excelled in data cleaning and manipulation tasks.

In image processing, the OpenCV (cv2) library played a key role in reading, displaying, and processing images. Additionally, the os module was used to efficiently manage files and directories during data handling.

For machine learning, we employed the sklearn library to implement algorithms such as RandomForestClassifier, SVC, and KNeighborsClassifier. Furthermore, sklearn's metrics modules were used to generate classification reports, calculate accuracy scores, recall scores, F1 scores, and produce confusion matrices for model evaluation.

In the deep learning domain, we leveraged the TensorFlow library and utilized several key modules:

- From tensorflow.keras.preprocessing.image, we used ImageDataGenerator for image augmentation.
- From tensorflow.keras.models, we implemented the Sequential model.
- From tensorflow.keras.layers, we incorporated layers such as Conv2D, MaxPool2D, Flatten, Dense, Dropout, BatchNormalization, and LeakyReLU.
- From tensorflow.keras.optimizers, we applied the Adam optimizer.
- From tensorflow.keras.callbacks, we used ModelCheckpoint and EarlyStopping to monitor and enhance model training.
- For learning rate adjustment, we employed ReduceLROnPlateau.
- For hyperparameter optimization, we used Keras Tuner, specifically employing kerastuner.tuners.Hyperband and kerastuner.tuners.RandomSearch to fine-tune our model parameters.

In terms of visualization, we relied on Matplotlib to generate clear and effective plots, while Seaborn provided more informative and statistically detailed visualizations. Additionally, the plot-model utility from keras.utils was used to visualize model architectures, allowing us to better understand the structure of our neural networks. This combination of libraries allowed us to effectively handle tasks from data preprocessing to model deployment, making it easier to develop and optimize our deep learning models.

2 Methods

2.1 Data Preprocessing and Augmentation

We started by uploading our images as 128x128 pixel arrays in grey scale meaning that we converted RGB standards to black and white colours that helped us to decrease our image dimensions [128,128,3] to [128,128]. We wrote all image pixel arrays to X data files and corresponding classes to Y files for training, validation, and testing. After uploading all images we got an array in the shape of [n, 128, 128] where n is the number of images in the array. However, our model needs to be a flat array meaning that it should be 2 dimensional. Therefore, we flattened our array as [n, 16384].

Since our training set is not sufficiently large enough, we generated images for the training part. This helped us to reduce overfitting in image classification and enhance our training dataset. ImageDataGenerator is used from TensorFlow library. This function created images with these features:

- **Rescaling:** All pixel values are rescaled to be in the range [0, 1] by dividing by 255.
- **Rotation:** Images are randomly rotated by up to 40 degrees.
- **Width Shift:** Images are randomly shifted horizontally by up to 20% of the width.

- **Height Shift:** Images are randomly shifted vertically by up to 20% of the height.
- **Shear Transformation:** Images are sheared by up to 20%.
- **Zoom:** Images are randomly zoomed in or out by up to 20
- **Horizontal Flip:** Images are randomly flipped horizontally.
- **Fill Mode:** When the transformations create empty areas in the image, these areas are filled using the nearest pixel values.

We created 10000 images for training by using this function and tested our model. Some examples of created images can be seen in Figure 2.

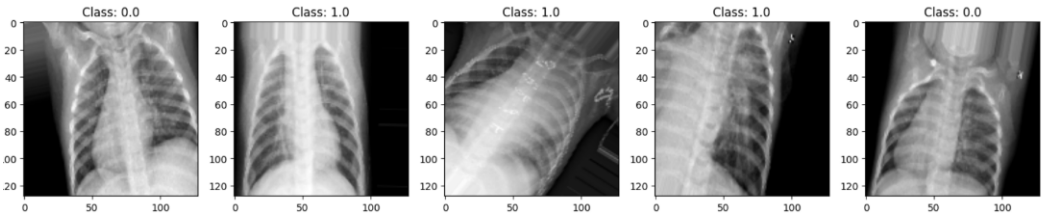

Figure 2: Sample examples generated after data augmentation.

2.2 K-Nearest Neighbour (KNN)

K-Nearest Neighbour (KNN) algorithm is a supervised machine learning algorithm that uses the proximity of data points to classify them under one of the given classes. It is one of the simplest classifiers that we use today since the test data points are classified only by looking at the classes of k-nearest data points (or as the name suggests, the neighbours) [3]. One can see the general KNN process scheme in Figure 3.

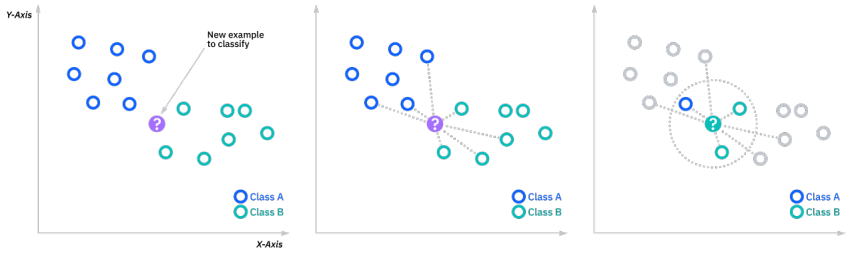

Figure 3: The general scheme of the KNN model process where k = 3 for the given model.

2.2.1 Distance Metric

As mentioned at the beginning of this subsection, the classification understanding of this model depends on the 'distance' of k-data points to the test sample we want to classify. The distance understanding of this algorithm is kind of different then we understand in our three-dimensional world since the algorithm looks at 'distances' in high-dimensional datasets. To use this algorithm, we first need to define a distance metric. This distance metric is mostly chosen as the Euclidean distance metric:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}, \tag{1}$$

Hoever, in the most general sense, the libraries of Python (especially the one we use, scikit-learn) use the Minkowski metric, which can be specified if required with the help of hyperparameter p:

$$d(x, y) = \left(\sum_{i=1}^{n} |y_i - x_i| \right)^{1/p}. \tag{2}$$

In these distance formulas, i denotes the number of features where x and y are the values of a single feature for two different data points, one being the test sample and one being one of the neighbours. Again, the usual choice for the distance metric is the Euclidean distance but some specific conditions might require different values of p.

2.2.2 Key Hyperparameters for the K-Nearest Neighbour Model

- **k-Number for k-Neighbours:** The choice of k value is quite important, and it highly depends on the dataset. The lower values of this parameter might cause overfitting, so one needs to be careful. In the cases where the dataset includes many outlier data or noise data, the higher cases would perform better, but these high values would cause a lower variance, which is something that we would want to protect in our classification. On the contrary, the lower values of k preserve the variance of the dataset but cause low-bias situations. Extreme cases of variance and bias are not good for the complexity of the model, so we are looking for an optimal point of variance and bias values [4].

- **Weights of the Data Points**: One can specify the weights of the data points in the model, and the model will be trained regarding these weights of points [4].

- **Algorithm Used to Compute the Nearest Neighbours:** One can specify which algorithm will be used to compute the nearest neighbours. BallTree, KDTree, or brute-force models might be used, or the model can automatically choose the most useful method by looking at the training dataset [4].

- **p-Number for the Distance Metric:** The choice of p changes how we define the distance metric of our model. For most cases, the Euclidean distance would do the job, but for lower accuracy cases, optimising the value of p for the model might be useful [4].

2.3 Support Vector Machines (SVM)

Support vector machine (SVM) is a supervised machine learning algorithm commonly used for classification tasks. It performs classification to find the optimal hyperplane (decision boundary) that maximizes the margin between the closest data points of opposite classes, which is basically the distance between classes [5]. The adjacent lines to each class in Fig 4 are known as support vectors, which determine those margins. The number of features determines the hyperplane. In a 2D space, it is a line; in an n-dimensional space, it is a plane. Since this project requires binary classification, 2D space will be used.

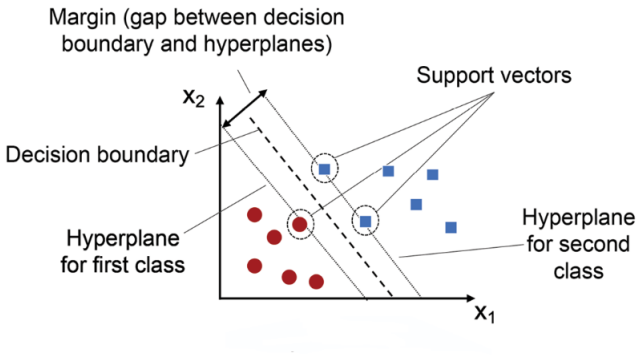

Figure 4: Linear SVM Model Representation [6].

Another property of the SVM algorithm is the kernel method. Kernel functions enable SVM to work with both linear and nonlinear classifications by altering the hyperplane[7]. These functions can be listed as linear kernels, polynomial kernels, radial basis function (RBF) kernels and sigmoid kernels. In this report, the RBF kernel is considered to be applied, which will be mentioned in the Results section.

2.3.1 Linear Kernel

The linear kernel is ideal for cases where the data is linearly separable. For the linear kernel, the decision function is:

$$f(x) = w \cdot x + b, \tag{3}$$

where w is the weight vector and b is the bias term. In a binary classification problem, the decision boundary is defined such that if $f(x) > 0$, the data point x is assigned to the positive class; if $f(x) < 0$, it is assigned to the negative class. The margin maximization can be expressed as an optimization problem:

$$\text{minimize } \frac{1}{2}\|w\|^2, \tag{4}$$

subject to the constraint:

$$y_i(w \cdot x_i + b) \geq 1, \quad \text{for } i = 1, \ldots, N \tag{5}$$

2.3.2 Radial Basis Function (RBF) Kernel

The RBF kernel (Gaussian kernel) is commonly used for non-linear, separable data. It maps the data into a higher-dimensional space and then performs linear separation [7]. Its function is defined as:

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right) \tag{6}$$

where γ is a parameter that controls the spread of the kernel.

2.3.3 Key Hyperparameters for the Support Vector Machines Model

- **Kernel Function:** As mentioned, there are four main kernel functions that can be used with SVM.
- **Regularization Parameter:** The regularization parameter is denoted with C, and it can take a constant float or integer value. Picking a lower value of C results in a wider margin and prevents overfitting. While choosing a smaller value than intended can cause underfitting.
- **Gamma Coefficient:** The gamma coefficient is a multiplier that can be used for polynomial, RBF and sigmoid kernels. It defines how far the influence of a single training example reaches. A higher gamma means that the decision boundary becomes tighter and more complex. In this report, scale and auto functions are prioritized for gamma. When gamma is chosen to be scaled:

$$\gamma = \frac{1}{\text{number of features} \times \text{variance of features}} \tag{7}$$

When gamma is chosen to be 'auto':

$$\gamma = \frac{1}{\text{number of features}} \tag{8}$$

2.4 Random Forest

Random Forest is a machine learning algorithm commonly used for both classification and regression tasks. During training, it creates multiple decision trees with the subset of the training data to predict class and the final output is determined by taking a majority vote across all the trees. This approach helps Random Forest to reduce the risk of over-fitting and improve prediction accuracy.

$$\hat{Y} = \arg\max_{c} \sum_{i=1}^{N} 1(h_i(X) = c). \tag{9}$$

For input X (in our case, it's 128×128 pixel arrays of the images), each decision tree T_i provides a prediction $h_i(x)$, which is a class label in the classification task. Y is the final prediction determined by the majority predicted class by decision trees. Random Forest is robust to noise and outliers, and it handles high-dimensional data well. However, it can be computationally intensive for large datasets with many trees.

2.4.1 Decision Trees

Decision trees are non-parametric supervised machine learning models used for both classification and regression. Decision trees can be seen as piecewise constant approximations using the if-then-else decision rules. Decision trees consist of leaves, nodes and branches.

Nodes represent the data that split according to the conditions. Branches connect nodes and represent possible outcomes of the decision. Leaves are the endpoints of a branch where no further splitting occurs. It contains the outputs of the model.

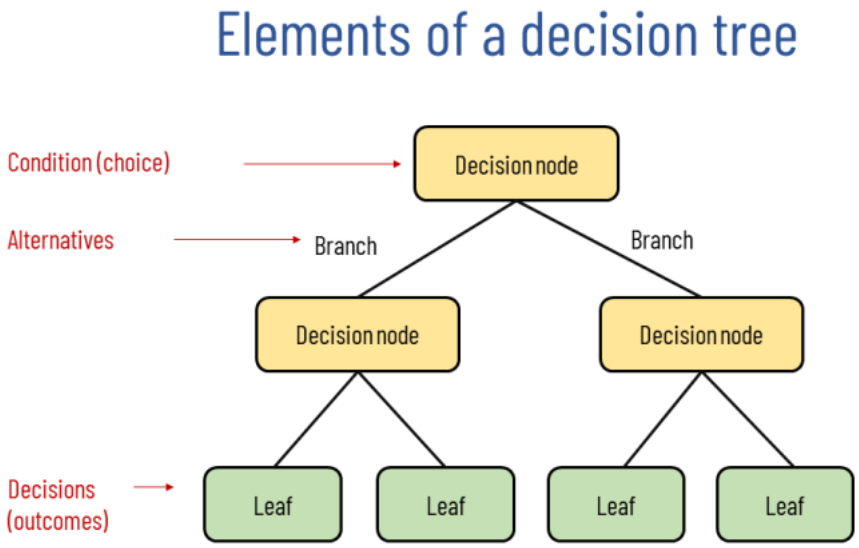

Figure 5: Decision Tree Diagram.

For example, in a classification task, a decision tree might split the data on the condition 'Age $>$ 30' at the root node, creating two branches: one for samples where the condition is true and another for samples where it is false

Training stops when all leaves are pure, when a node has fewer samples than a minimum threshold, or when the tree reaches a pre-defined maximum depth. These hyperparameters determine the complexity of the decision tree. Shallow trees might underfit the data, failing to capture important patterns, while very deep trees might overfit, capturing noise and reducing generalization.

Decision trees use Gini or Entropy functions to measure how good a split is to maximize homogeneity within each branch. The main goal is to maximize information gain and minimize the impurity [8].

$$D(\text{Entropy}) = -\sum_{k=1}^{K} \hat{p}_{mk}\, log\hat{p}_{mk}, \tag{10}$$

$$G(\text{Gini index}) = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}). \tag{11}$$

Either Gini index and entropy are generally used to evaluate the quality of the splits where $\hat{p}_{(mk)}$ represents the proportion of observations in the m-th region that are from k-th class.

2.4.2 Key Hyperparameters for the Random Forest Model

- **Max Depth:** Max depth defines the longest path between the root node and the leaf node [9].

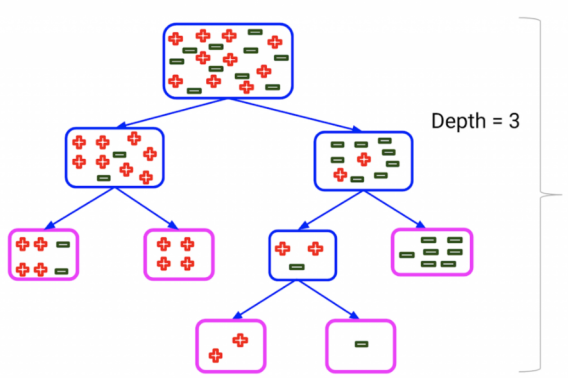

Figure 6: Depth of Decision Tree.

- **Min Sample Split:** Min sample split determines the threshold for the decision tree to a required number of observations in any given node to split it [9].

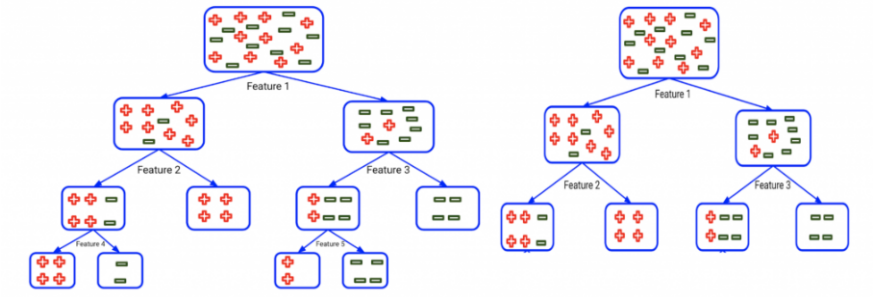

Figure 7: Effect of Min Split Value 2 and 6.

- **Min Sample Leaf:** This parameter determines the minimum number of samples that should exist in the leaf node after splitting it [9].
- **N Estimators:** N estimator adjusts how many trees random forests will train for classification [9].
- **Max Features:** Random Forest chooses random samples from the feature set to find the best split, and this parameter resembles the number of maximum features provided to each tree [9].

2.5 Convolutional Neural Network (CNN)

Neural networks are algorithms that take place at the heart of deep learning methods. In general, they are made by an input layer, a number of hidden layers and an output layer, where each layer consists of a previously determined number of nodes and each node is connected to some others with weights and a threshold function [10].

While neural networks help us with deep learning, convolutional neural networks are used especially for classification tasks because these neural networks leverage the principles of linear algebra by using different matrix multiplication methods and identifying patterns in classified objects. On the other hand, the methods of linear algebra also make the classification task more computationally demanding, which can be solved by using the power of graphical processing units (GPUs). We will now look more deeply into the details of convolutional neural networks [10].

2.5.1 Layers of Convolutional Neural Networks

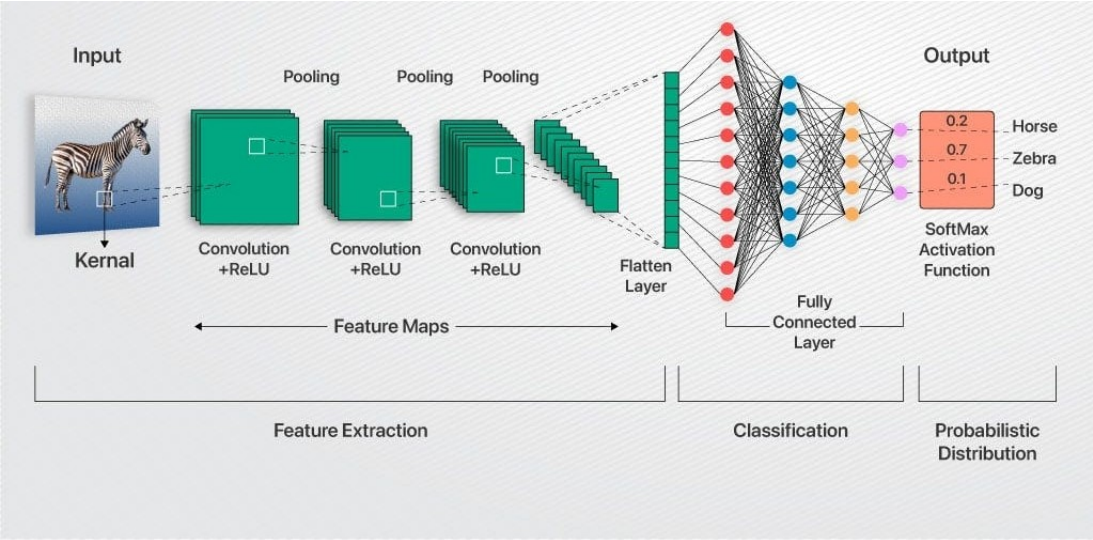

Figure 8: The scheme of a convolutional neural network [11]

The convolutional neural networks, which are mainly used to classify different types of inputs, consist of 2 main parts: the feature extraction part and the classification part.

The feature extraction part consists of two different types of layers: the convolutional layers and the pooling layers.

- **Convolutional Layers:** The convolutional layers are the first layers of our neural network, coming right after the initial input layer. They create the majority of computational load and complexity by using different-sized kernels, or filters, which get convolved with input data and extract features.

  A kernel is generally a 2D and usually 3x3 array of weights, which moves to different parts of the input data; its dot product with the overlapping part of the data is calculated and written as a single pixel to the output of that layer. Then, the filter is shifted by the stride value, and the same calculation is repeated until the whole image is covered by the filter. In the end, a nonlinear activation is used to make a nonlinear layer out of the linear calculation, and the final output, which is named a feature map, is obtained.

- **Pooling Layers:** After the convolutional layer, there comes a pooling layer, which downsamples the output of the convolutional layer, extracts further information from it and reduces the number of input parameters of other layers. These layers are still in the feature extraction part, but they work with a different principle.

Again, a filter is used for that purpose with a predetermined size, usually 2x2. However, this time, the filter can be one of the following two things: it can either use a max-pooling method, which looks at all values inside the filter and chooses the maximum value out of them, or an average-pooling method, which takes the average of all values inside our filter. These two functions are called aggregation functions. This process limits the possibility of overfitting and increases efficiency but causes a lot of information loss, so it should be used carefully.

- **Fully Connected Layers:** After the feature extraction part, we first flatten the matrix we have as the last output, take each pixel value as an input and give them to the fully connected layer. In this layer, each input node (pixel) is multiplied by a different weight for each node in the next layer, summed with a bias (if it exists) and put into a nonlinear activation function to introduce nonlinearity to the model. Then, the output of the first layer is taken as the input of the second layer, and this calculation is repeated for a determined number of times. At the end of this calculation, the output of the last layer is put into a sigmoid function if the classification is binary or a softmax function if the classification is multinomial. This is why this part of the network is called the classification layer.

2.5.2 Key Hyperparameters for Convolutional Neural Network Model

In the convolutional layers, there are 7 different key parameters one can set:

- **Filter Size:** One can adjust the size of the filter and the number of weights by doing so, which increases the computational cost.
- **Number of Filters:** The number of filters with different weights can be adjusted inside a convolutional layer, so from an N x N x 1 input with M filters of size 3 x 3, one can obtain (N-2) x (N-2) x M output.
- **Stride:** Stride is the amount that the filter moves in the input data array to calculate the value of each pixel.
- **Padding:** Padding determines the value added to all four sides of the 2D input array, and this changes the values of the output pixels, which are at the edges of the input array. It can be valid padding, which does not add any value to the sides; same padding, which keeps the size of the input and the output of the convolutional layer the same; and full padding, which increases the size of the output by adding zeros to the borders of the input and the convolution calculates the related pixels using the zeros at the borders.
- **Type of the Activation Function:** At the end of each convolutional layer, in order to introduce nonlinearity to the system, activation functions are used. There are many different activation functions, like ReLU, Leaky ReLU, tanh, Maxout, etc.
- **Additional Layers:** One can add batch normalization layers, which normalize the values inside a given input; dropout layers, which randomly make some elements of the input tensor zero with a given probability; or other types of layers between the convolutional layer and pooling layer.

- **Number of Convolutional Layers:** One can use several consecutive convolutional layers in one block, which ends with a pooling layer, and extract information further; or one can use several consecutive convolutional blocks, which can consist of several consecutive convolutional layers, additional layers mentioned above and a pooling layer at the end.

In the pooling layers, there are 2 key parameters we can set:

- **Filter Size:** One can adjust the size of the filter and the number of weights by doing so, which increases the computational cost.
- **Type of Pooling:** As mentioned in the description of the pooling layer, one can choose either the max-pooling or the average-pooling method.

In the fully connected layers, there are ... different key parameters we can set:

- **Number of Nodes:** Each of M input nodes is connected to N nodes, which creates N x M learnable parameters. So, by changing the number of nodes at each layer, one can set the number of learnable parameters, which may or may not increase the accuracy and affect the computational cost.
- **Type of the Activation Function:** Again, we can set the activation function at the end of each layer as ReLU, Leaky ReLU, tanh, Maxout, etc.
- **Additional Layers:** Again, one can add dropout layers or normalization layers as a part of fully connected layers.
- **Number of Fully Connected Layers:** One can add a different number of node layers and, by doing so, make a better classification if all the parameters are set correctly.

3 Results

3.1 K-Nearest Neighbour (KNN)

After the image data is loaded and flattened, the hyperparameter tuning is made with the non-augmented data. During the hyperparameter tuning, we only looked at the k-parameter since the default "algorithm" parameter of the model finds the most optimal nearest neighbour search algorithm automatically, and the change of p-value is not necessary for our model. Using the Euclidean distance metric, we can find the most optimal k-value, which will work for us.

In order to find the optimal value of our k-parameter, we used the GridSearchCV function of scikit-learn. This method takes the different values of parameters and uses n-fold cross-validation to find which of these parameters is the optimal one. Normally, this process takes a long time because of the cross-validation, but since we only have one parameter for the KNN model, we chose to use this method.

The n-fold cross-validation model works by separating the given dataset into n pieces. From these n pieces, 1 piece is left out, and the other n-1 pieces are used to train the model. The training process with n-1 pieces is repeated for each of the given parameters, and each trained set is tested with the piece the model left out at the beginning. This

process is repeated n times with different pieces of the dataset being left out, and different n-1 pieces are used to train the model. At the end of the process, the model chooses the best parameter (or parameter combination) for our model by looking at the validation parameters.

For this project, we wanted the model to test all values of k from 1 to 53, which is one per cent of our training dataset size, and chose n (the cross-validation size) as 5, which is the default size for cross-validation of the GridSearchCV function. At the end of the calculation, the model told us that the best parameter is k=6, with cross-validation accuracy being 0.93873; so we chose k=6 for the rest of our calculations.

After selecting the best hyperparameter, confusion matrices are plotted for non-augmented and augmented data, which can be seen in Figure 9.

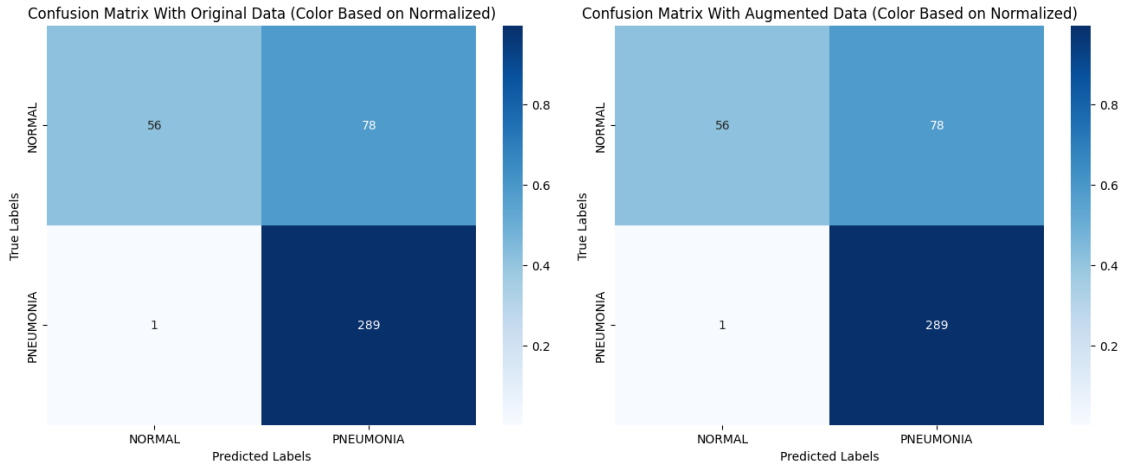

Figure 9: Confusion matrices of the original data (left) and augmented data (right) for KNN model.

With respect to confusion matrices, precision and recalls, F1 scores and accuracies are calculated in Table 1.

Performance Metric	Original Data	Augmented Data
Precision	0.785	0.787
Recall	0.997	0.997
Accuracy	0.811	0.814
F1 Score	0.878	0.879

Table 1: Performance metrics of original and augmented data of KNN model.

3.2 Support Vector Machines (SVM)

After the image data is loaded and flattened, the hyperparameter tuning is made with the non-augmented data. Some of the tuning parameters and F1 scores from 144 unique combinations are represented in Table 2.

It is observed that the highest F1 score is achieved with RBF kernel when $C = 0.1$ and $\gamma =$ Scale with 0.738. After selecting the best hyperparameters, confusion matrices are plotted for non-augmented and augmented data which can be seen in Figure 17 .

SVM Hyperparameters				
Kernel Function	**Reg. Param. (C)**	**Gamma**	**Polynomial Degree**	**F1 Score**
Linear	10	—	—	0.737
RBF	0.1	Auto	—	0.667
RBF	0.1	Scale	—	0.738
RBF	100	Scale	—	0.732
Poly	0.1	Scale	2	0.725
Poly	0.1	Scale	3	0.737
Sigmoid	0.1	Scale	—	0.650

Table 2: Hyperparameter tuning for SVM model.

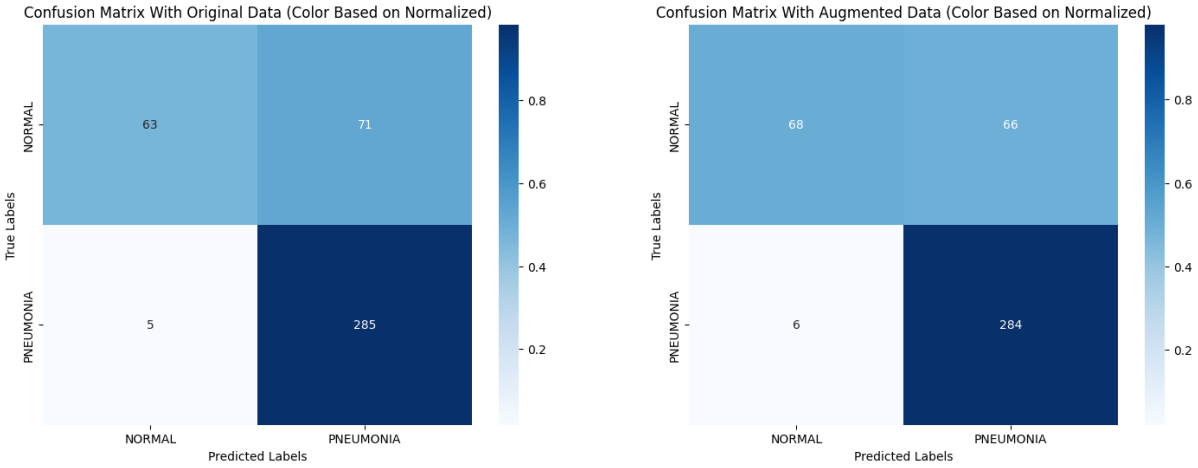

Figure 10: Confusion matrices of the original data (left) and augmented data (right) for SVM model.

With respect to confusion matrices, precision and recalls, F1 scores and accuracies are calculated in Table 3.

Performance Metric	Original Data	Augmented Data
Precision	0.801	0.800
Recall	0.983	0.979
Accuracy	0.821	0.818
F1 Score	0.882	0.881

Table 3: Performance metrics of original and augmented data of SVM model.

3.3 Random Forest

After the image data is loaded and flattened, the hyperparameter tuning is made with the non-augmented data. One can see the results of the hyperparameter tuning process in Table 4.

The best 5 and worst 5 F1 metrics, which were found with different hyperparameters, can be seen in Table 4. The F1 metric is used instead of accuracy since the data

n_estimators	max_depth	min_samples_split	min_samples_leaf	max_features	F1 Score	Category
300	20	2	4	log2	0.7714	Best
300	20	2	1	log2	0.7687	Best
200	20	10	4	log2	0.7687	Best
100	20	2	4	log2	0.7660	Best
300	30	2	4	log2	0.7714	Best
300	10	10	1	sqrt	0.7397	Worst
300	20	10	1	sqrt	0.7397	Worst
200	20	10	4	sqrt	0.7423	Worst
100	20	10	4	sqrt	0.7397	Worst
200	10	10	1	sqrt	0.7423	Worst

Table 4: Hyperparameter tuning for random forest model.

does not have the same number of observations from each class. Models with different hyperparameters are trained with train data to test which models give the best F1 score on validation data. The model with n-estimators = 300, max depth 20/30, min samples split 2, min samples leaf 4, max features log 2 gave the best F1-Score out of 240 different models. We chose the maximum depth of 20 to avoid overfitting and joint training and validation to train our model.

Performance Metric	Original Data	Augmented Data
Precision	0.819	0.830
Recall	0.986	0.983
Accuracy	0.841	0.851
F1 Score	0.895	0.900

Table 5: Performance metrics of original and augmented data for random forest model.

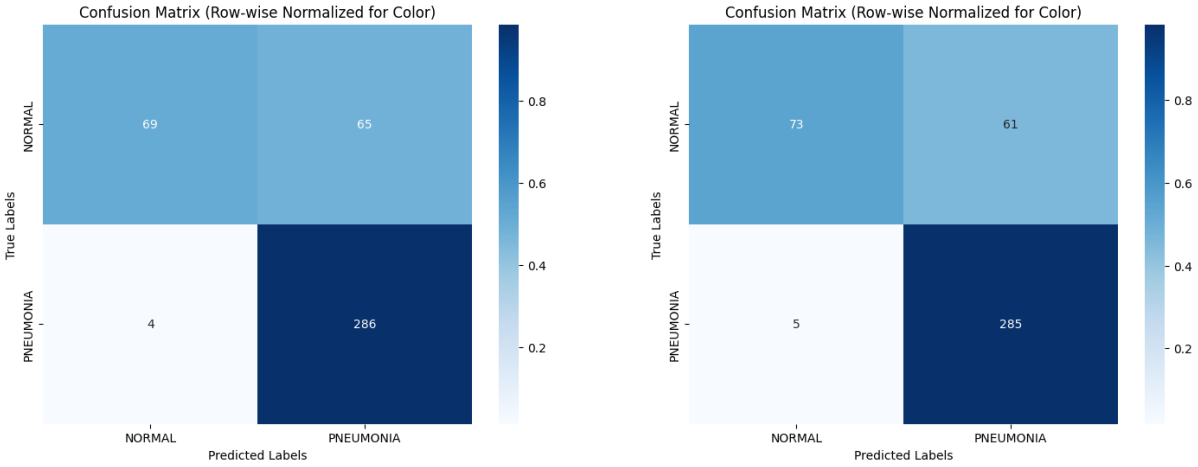

Figure 11: Confusion matrices of the original data (left) and augmented data (right) for random forest model.

With respect to confusion matrices, precisions and recalls, F1 scores, and accuracies are calculated.

3.4 Convolutional Neural Network (CNN)

After the image data is loaded and flattened, the hyperparameter tuning is made for both the original and augmented data. Since the model is more complex with respect to the ML models it has been decided that a hyperparameter tuning for augmented data is required. For the tuning, RandomSearch package from kerastuner is used. During the tuning, the complexity of the model is increased with each tuning trial, the result of some of the trials can be seen in Figure 12.

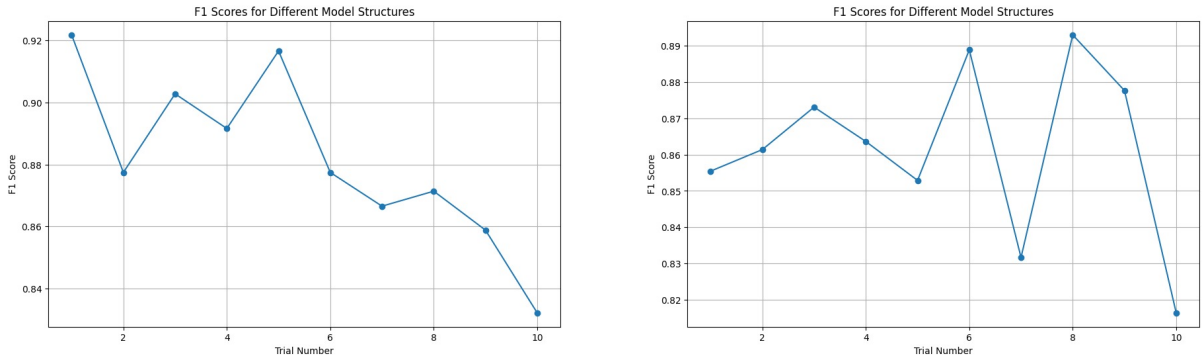

Figure 12: Hyperparameter tuning trials for original data (left) and augmented data (right) for CNN model.

It can clearly be observed that, as the model gets more complex, the F1-score starts to decrease and it is more prominent in the original data. After the tuning is completed for the model structures, the following structures are determined to be the best resulting ones which can be seen in Figure 13 and Figure 14.

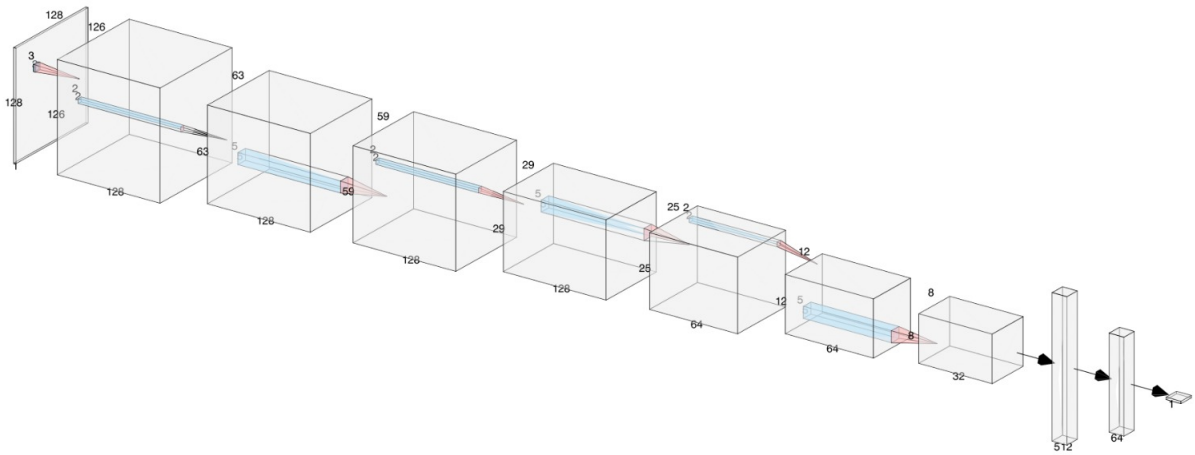

Figure 13: Best CNN model structure for the original data.

The CNN model for the original data structure consists of 4 convolutional layers and 2 dense layers. The convolutional layers are configured as follows:

- Layer 0 has 128 filters with a kernel size of 3 and Leaky ReLU activation ($\alpha = 0.1$)
- Layer 1 has 128 filters with a kernel size of 5 and Leaky ReLU activation ($\alpha = 0.1$)
- Layer 2 has 64 filters with a kernel size of 3 and Leaky ReLU activation ($\alpha = 0.1$)
- Layer 3 has 32 filters with a kernel size of 5 and Leaky ReLU activation ($\alpha = 0.1$)

The dense layers are configured as follows:

- Dense Layer 0 has 64 units with a dropout rate of 0.4

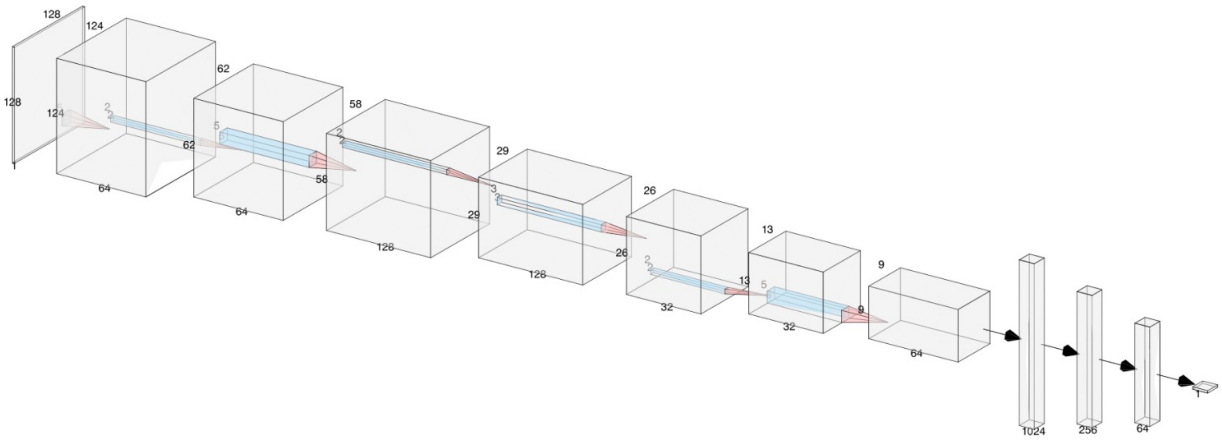

Figure 14: Best CNN model structure for the augmented data.

The CNN model for the augmented data structure consists of 4 convolutional layers and 2 dense layers. The convolutional layers are configured as follows:

- Layer 0 has 64 filters with a kernel size of 5 and Leaky ReLU activation ($\alpha = 0.01$)
- Layer 1 has 128 filters with a kernel size of 5 and Leaky ReLU activation ($\alpha = 0.01$)
- Layer 2 has 32 filters with a kernel size of 3 and Leaky ReLU activation ($\alpha = 0.01$)
- Layer 3 has 64 filters with a kernel size of 5 and Leaky ReLU activation ($\alpha = 0.01$)

The dense layers are configured as follows:

- Dense Layer 0 has 256 units with a dropout rate of 0.4
- Dense Layer 1 has 64 units with a dropout rate of 0.3

After the model structures are determined a hyperparameter tuning is done for batch size. The results can be seen in Figure 15. From the results, it is determined that the best performing batch size is 34 for original and 58 for augmented data. Moreover, The Adam optimizer with a learning rate of 0.001 is utilized for model optimization, and the

loss function is set to binary crossentropy. The output layer uses the Sigmoid activation function as the task involves binary classification. The training process incorporates callbacks such as 'ReduceLROnPlateau', which dynamically adjusts the learning rate by reducing it by a factor of 0.5 after 3 epochs of no improvement in validation loss, with a minimum learning rate of 1×10^{-6}, eliminating the need for manual tuning of the learning rate. Additionally, a checkpoint callback saves the best-performing model based on validation accuracy, and early stopping is implemented with a patience of 30 epochs to prevent overfitting. Each model is trained for up to 100 epochs.

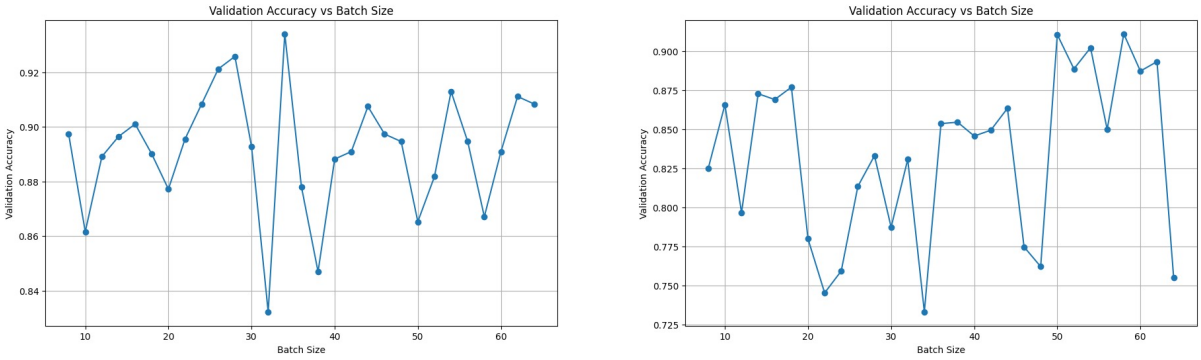

Figure 15: Hyperparameter tuning for batch sizes of the original data (left) and augmented data (right) for CNN model.

Lastly, after selecting the best hyperparameters, confusion matrices are plotted for original and augmented data which can be seen in Figure 16. With respect to confusion matrices, precision and recalls, F1 scores and accuracies are calculated in Table 6.

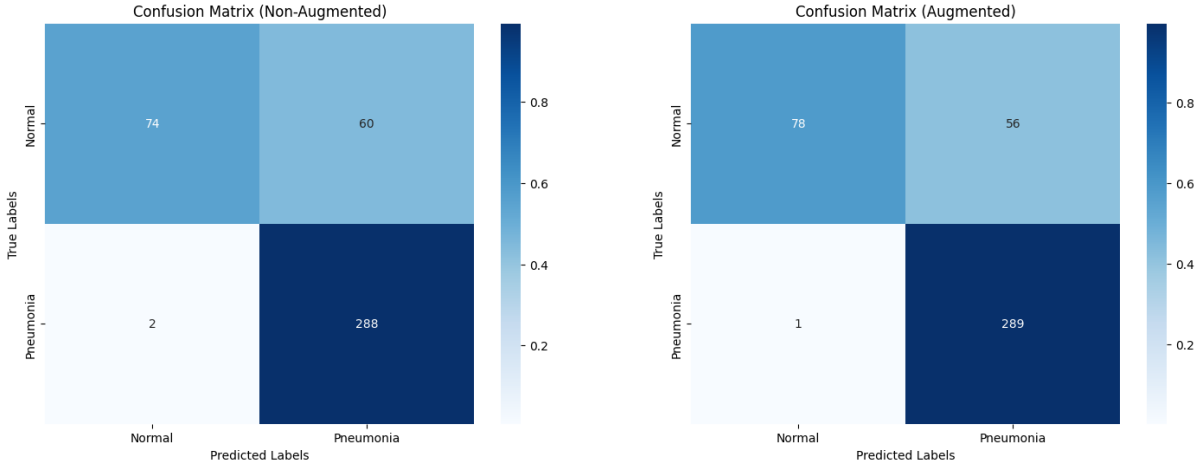

Figure 16: Confusion matrices of the original data (left) and augmented data (right) for CNN model.

Performance Metric	Original Data	Augmented Data
Precision	0.8276	0.8377
Recall	0.9931	0.9966
Accuracy	0.8538	0.8656
F1 Score	0.9028	0.9102

Table 6: Performance metrics of original and augmented data for CNN model.

4 Discussion

In our exploration of various models to predict Pneumonia, we observed distinct outcomes with KNN, SVM, Random Forest and CNN. We used F1-Score's to compare model performances since our test data isn't equally distributed over classes. Most of the models gave slightly better results with augmented data. However, augmented data is not necessarily improve all of our models but more complex models, like CNN, needing more training benefit more from augmented data better.

The reason augmented data did not significantly improve our model's performance is that our test set consists solely of images from the original dataset. Meaning that all of our test data can already be explained with the original data. However, using augmented data enhances the model's ability to generalize. If the model encounters flipped or differently formatted images in real-world scenarios, it will be better equipped to predict them accurately.

Random Forest gave the best F1-Scores among other simpler machine learning models with 90% F1-Score. Then SVM gives 88.1% F1-Score which is slightly better than KNN which performs with 87.9% F-Score.

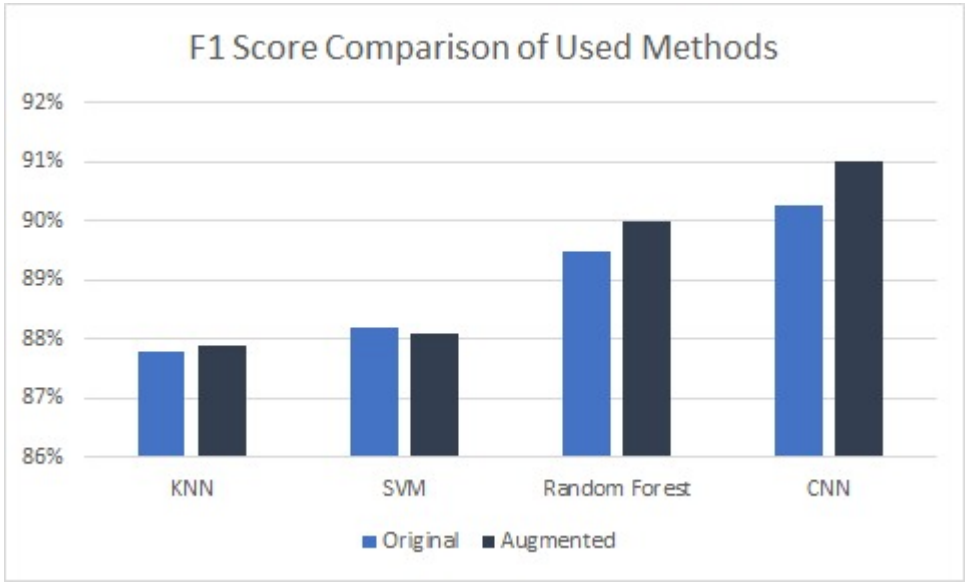

Figure 17: F1-Scores of the models with augmented and original data

When tuning our models, we discovered that our data doesn't give better results with complex models. In Random Forest, increasing the depth and complexity of the decision trees decreases accuracy. The same problem occurs with SVM too. Higher C

values easily cause overfitting. This problem was more severe for more complex models like CNN. We observed extreme overfitting results with CNN. Especially, increasing the complexity of the model architecture backfired. We achieved an optimal balance between model complexity and effectively learning from the data by tuning parameters. Therefore, Random Forest is complex enough to understand the patterns of the data without causing overfitting since it's an ensembling method which creates n number of different models to decrease bias and overfitting whereas CNN is harder to train and causes more overfitting with our dataset. Overall, Random Forest seems like better model choice for our dataset since CNN and Random Forest almost have the same F1-Scores.

5 Conclusion

In conclusion, we implemented and evaluated four models for the task of analyzing medical images to detect pneumonia. These models included a Convolutional Neural Network (CNN) as the deep learning model, as well as Random Forest, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) as traditional machine learning approaches. Each of the models were tested for different hyperparameters.

Since our test set is not evenly distributed, we prioritized F1-Score over accuracy in evaluating model performance. In general, CNN provided the best results with 91.2% F1-Score. Even though CNN yielded the best result, following CNN was Random Forest with a 90% F1-Score. The worst model obtained was KNN, which achieved a F1-Score of 87.9%. Furthermore, it was seen that although a higher score was obtained with CNN, because of the overfitting challenges and complexity of the model, Random Forest was seen to be a better solution for this specific image classification task.

Overall, this project demonstrated the diverse capabilities of machine learning models for image classification tasks, highlighting the importance of selecting appropriate models based on the nature of the data and practical constraints.

6 Workload Distribution

- **Ahmet Sayan:** Responsible for data preprocessing and implementation of Random Forest models.
- **Arda Erkan:** Responsible for the implementation of the KNN model and CNN. Worked together with Damla Uçar and Eren Keskin.
- **Bahar Özkırlı:** Responsible for data preprocessing and worked together with Ahmet Sayan.
- **Damla Uçar:** Responsible for the implementation of the KNN model. Worked together with Arda Erkan.
- **Eren Keskin:** Responsible for the implementation of the SVM model and CNN model. Worked together with Arda Erkan.

Additionally, each individual worked on the report and the presentation equally.

References

[1] I. Rudan, C. Boschi-Pinto, Z. Biloglav, K. Mulholland, and H. Campbell, "Epidemiology and etiology of childhood pneumonia," *Bulletin of the World Health Organization*, vol. 86, pp. 408–416, 2008.

[2] P. Mooney, "Chest x-ray images (pneumonia)," 2024, accessed: Oct. 7, 2024. [Online]. Available: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data

[3] IBM, "What is the k-nearest neighbors algorithm?" Oct 2024. [Online]. Available: https://www.ibm.com/topics/knn

[4] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[5] IBM, "What is a support vector machine?" 2024, accessed: 2024-11-17. [Online]. Available: https://www.ibm.com/topics/support-vector-machine#:~:text=A%20support%20vector%20machine%20(SVM,in%20an%20N%2Ddimensional%20space

[6] S. Saxena, "Classification model using svm classifier in python [example]," 2024, accessed: 2024-11-17. [Online]. Available: https://vitalflux.com/classification-model-svm-classifier-python-example/

[7] S. learn Developers, "Support vector machines," 2024, accessed: 2024-11-17. [Online]. Available: https://scikit-learn.org/1.5/modules/svm.html

[8] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*, 2nd ed. New York, NY: Springer, 2021, available for free online. [Online]. Available: https://www.statlearning.com/

[9] S. Saxena, "Beginner's guide to random forest hyperparameter tuning," 2020, accessed: 2024-11-14. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

[10] IBM, "What are convolutional neural networks?" Dec 2024. [Online]. Available: https://www.ibm.com/think/topics/convolutional-neural-networks

[11] AnalytixLabs, "Convolutional neural networks: definition, architecture, types, applications, and more," Sep 2024. [Online]. Available: https://medium.com/@byanalytixlabs/convolutional-neural-networks-definition-architecture-types-applications-and-more-106ca69d9ae6