Project 3 on Machine Learning - Pneumonia Detection on Chest X-rays with Support Vector Machines and Convolutional Neural Networks

Thomas Haaland
Jie Hou
Huiying Zhang
Martin Funderud Gimse
https://github.uio.no/thomhaa/Fys-Stk4155.git

Dec 2019

Contents

Chapter 1

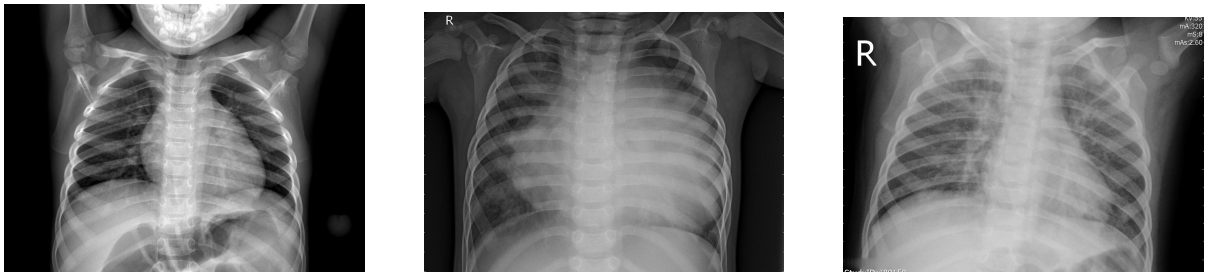
Abstract

In this project we will examine chest X-ray images of patients and attempt to classify whether the patient has pneumonia or not at a level exceeding practicing radiologists. We will use Support Vector Machines and Convolutional Neural Networks with various methods and compare their performance. Our dataset includes 5863 chest X-ray images with two diseases, bacterial and viral pneumonia.

With Support Vector Machines(SVM) we got an precision score 78% in detecting bacterial pneumonia and an precision of about 77% with Convolutional Neural Networks(CNN). However, we got much better results at detecting pneumonia in general, precision of 96% using CNN and an precision score of 89% using SVM. We also tried a Feed Forward Neural Network based on the features we got from PCA, KDE and Haralick feature extraction and got precision of 73% for detection of bacterial pneumonia. We will show that the CNN have perfmance comparable to that of physicians.

Chapter 2

Introduction



(a) No pneumonia (b) Bacterial pneumonia (c) Viral pneumonia

Figure 2.1: The three different classes in the dataset.

According to the World Health Organisation [9] pneumonia worldwide account for 15% of all deaths of children under the age of 5 years. In 2017 alone 808 694 died from pneumonia. While there can be many causes for pneumonia for example, bacterial, viral and fungal, only the bacterial variant can be treated with antibiotics. However, only one third of children with pneumonia receive the antibiotics they need. Chest X-rays are currently the best available method for diagnosing pneumonia[10], playing a crucial role in clinical care and epidemiological studies[2]. However, it is not easy to detect pneumonia in chest X-rays, it relies on the availability of expert radiologist. In addition, it's important to recognise that there is considerable inter-observer variation in the recording of symptoms and also a high degree of inter-observer error in the physical examination of the chest[18]. The motivation for using machine learning methods for this task is to quickly and accurately detect pneumonia and perhaps even differentiate between different kinds of pneumonia. Using machine learning methods for diagnosis can help by offering a diagnosis where health personnel are pressed for time or otherwise unavailable, and also assist in decision making at hospitals. In this work, we will use Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) to analyse the X-ray images and try to predict whether a patient has pneumonia or not and also what kind of pneumonia the patient has. We will first preprocess the images, which includes resizing of the images and make them greyscale. In addition, feature selection needs to be done to extract useful information from the images, this includes methods like Principal Component Analysis (PCA), Kernel Density Estimation(KDE), Recursive Feature Elimination(RFE), Haralick texture recognition as well as using only the images themselves. We then got our dataset ready to evaluate how different feature selection methods and scaling methods contributes to our classification problem. Moreover, we use gird search cross validation to find the best parameters for our model including leaning rate, epochs etc. Finally, we build our final SVM and CNN model respectively to detect whether a person has pneumonia

or not as well as what kind of pneumonia. For comparison, we also added Feed Forward Neural Network to compare with our Convolutional Neural Network model.

Chapter 3

Chest X-ray images

We will look at a dataset of X-ray images (link to images) and try to predict whether the patient in question has pneumonia or not. All chest X-ray images were selected from retrospective cohorts of paediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. [3] The dataset is organized into three folders, Train, Test and Validation. The dataset is split into two categories, normal and pneumonia. We have in total 5 863 chest X-ray images. Among the pneumonia data folder the set is further divided between bacterial and viral pneumonia. There are no examples of fungal pneumonia. For the analysis of chest x-ray images, all chest radio-graphs were initially screened for quality control by removing all low quality or unreadable scans. In order to ensure the quality of the training dataset, the diagnoses were graded by two expert physicians before being cleared for the training the neural network. In addition, concerning any grading errors, the evaluation set was also checked by a third expert[3].

If there are no pneumonia present, both lungs should be quite clear and the chest expands during inhalation. Bacterial pneumonia in particular is visible as a diffuse cloud in the affected area and for children under 5 years of age, bacterial pneumonia can often be diagnosed by the presence of fast breathing or low chest wall in-drawing where their chest retracts during inhalation instead of expanding movements. Viral pneumonia presents similar features, however it can be much more numerous, wheezing is common in viral infections.

3.1 Pre-processing and Feature Selection

In this section we will discuss how we prepare the images and chose features to work with. Before starting training on the dataset we need to learn a bit about it. The X-ray images are of different sizes with different dimensions, some are colour image format and some black and white. The first step is to convert all images to black and white since we need all images to have the same format, doing so means that we might loose some information on some images, but we will save space. We made an assumption that the colour feature space, probably is not important enough to keep. We also need the images to be in the same size. This introduces a number of problems, since we need to consider whether the images should be cropped or resized uniformly or stretched. Regardless how we do it we will end up loosing some information. We opted to make a simple resize to a format smaller than the smallest image.

3.1.1 Scaling

We scale the images initially by dividing by 255 and subtracting 0.5- This centers the data set and gives a maximum of 0.5 and minimum of -0.5. Additionally we employed the StandardScaler and QuantileTransformer algorithms. The StandardScaler is implemented by

$$\boldsymbol{z} = \frac{\boldsymbol{x} - \mu}{\sigma} \tag{3.1}$$

where μ is the mean and σ is the standard deviation. By normalizing the training data to have mean 0 and variance 1 along the features can often improve convergence during gradient descent, since it will

avoid many extra iterations that are required if one or more features have much larger values than the rest.

QuantileTransformer transforms the dataset to follow a uniform distribution. This scaler can deal well with outliers in the dataset and it uses quantile information to do so. That is along any axis it transforms from a gaussian distribution to uniform using the quantiles in the gaussian distribution. This method will shrink the distances between marginal outliers and inliers.

3.1.2 Recursive Feature Elimination(RFE)

Recursive Feature Elimination with Cross Validation ranks features in importance and recursively removes the least important features. This method tends to improve a models predictive properties since features with less predictive properties tend to confuse the model thus reducing model performance.

3.1.3 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is one of the most popular dimensionality reduction algorithms, it is a statistical procedure used to emphasize variation and bring out strong patterns in a dataset. It identifies the hyper-plane that lies closest to the data and then projects the data onto it [8]. When using this method it is important to first center the data set. Generally, PCA seeks a linear combination of variables such that the eigenvectors with the least variance are removed from the dataset. The eigenvector with the largest eigenvalue represents the direction in which the dataset has the largest variance.

Furthermore, to inspect where the entire data set varies the most we can look at the eigenvectors produced by the algorithm, when applied universally to the data set. In figure 3.1 we plot the first four eigenvectors and we can see where the set of images varies the most. It seems the PCA first picks up on variations in the position and size of each person's chest in the two first eigenvectors, then in the third and fourth eigenvectors variations in the lungs are registered. This seems promising, since we know that pneumonia can constrict inhalation, limiting chest expansion and we know that pneumonia should be visible as a shadow in the lung region on X-ray images.

A technique which worked surprisingly well was to use the principal components themselves as features. By applying PCA to each image separately we retrieved principal components which describes the variation of each image in relation to itself. The PCA procedure for a matrix X (in this case, one image) is (3.2)

$$X^T X = V \Sigma^2 V^T \tag{3.2}$$

where the matrix Σ^2 is a diagonal matrix containing the eigenvalues squared in a descending order and V is a matrix composed column wise from the eigenvectors given the respective eigenvalues. The matrix of eigenvectors V can be used as a feature set. These features provided decent predictive properties alone.

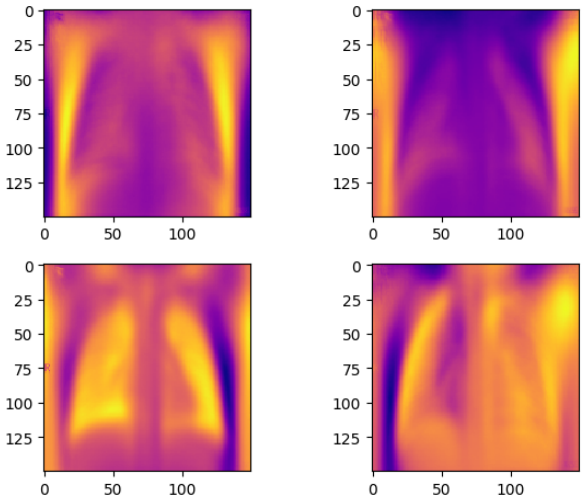


Figure 3.1: The first four eigenimages of the dataset. The PCA picks up on variations in the lungs especially in the image to the lower left.

PCA can be used for dimension reduction both on a per image basis and on the whole dataset.

3.1.4 Kernel Density Estimation(KDE)

Histogram analysis is a useful tool for feature extraction on images. The Kernel Density Estimation algorithm is related to histograms. The Kernel Density Estimation function is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_i K\left(\frac{x - x_i}{h}\right) \tag{3.3}$$

where h is the bandwidth, a free floating parameter which determines the smoothness of the estimate, K is the kernel in statistics, a Gaussian kernel was used in this project. For a sample image the Kernel Density estimation is shown in figure 3.2.

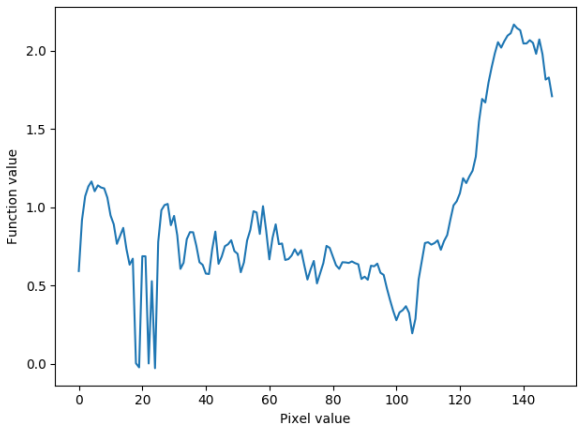


Figure 3.2: Kernel Density Estimation for a sample image.

The Kernel Density Estimation is related to histograms and displays a continuous histogram displaying something akin to pixel value frequency per row. This value and its distribution is, however, dependent on the free parameter bandwidth. So, finding the optimal bandwidth can be done with the grid search method.

3.1.5 Haralick texture recognition

The Haralick texture recognition algorithm are used to assign numerical values to different texture types. If the pneumonia is of a different texture than healthy tissue, we would expect this feature extraction method to give us more information. The Haralick texture recognition algorithm makes a classification between 14 different texture features [12].

3.1.6 The image itself

The image itself may provide valuable information. So we tried to include the image itself. In the case of Convolutional Neural Networks, feature extraction and selection are performed by the network and attempting to do feature extraction on the image before sending it into a CNN tends to make the network perform worse.

3.1.7 Considerations in feature extraction

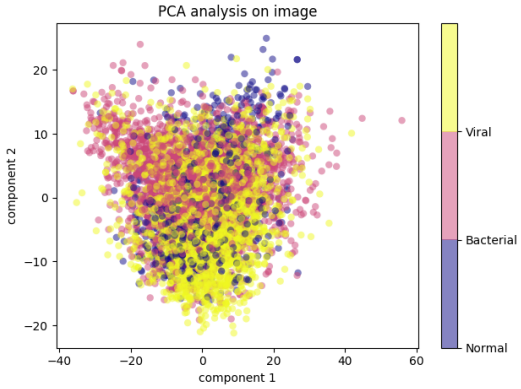
When extracting features it is easy to end up with more features than data samples, especially since most images initially has at least 400×350 pixels/features. When performing feature extraction, the goal is to get just enough features, but not more than what we need. Too many features presents problems for computer performance and cluttering the model. Additionally, introducing a large number of irrelevant features tends to reduce model performance.

PCA can be very handy in reducing the number of features since most features provide almost no variation and thus probably less predictive properties. A caveat is that PCA can only gain as many linearly independent features as there are sample points and thus for some data sets we can end up throwing away valuable information unless considered carefully.

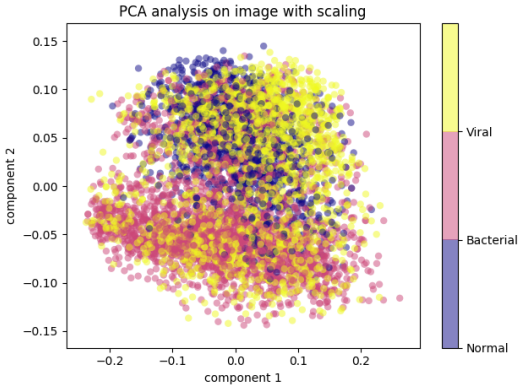
Additionally we can employ Recursive Feature Elimination with Cross Validation (RFECV), where we use some supervised learning method on the data set and progressively remove features which provide the least predictive properties. We then select the features which in the end results in the best fit and use these features to train the model properly. This works since irrelevant features tend to confuse the model. So, if we manage to filter out the least important features the overall performance of the model should improve.

3.2 Considering the dataset

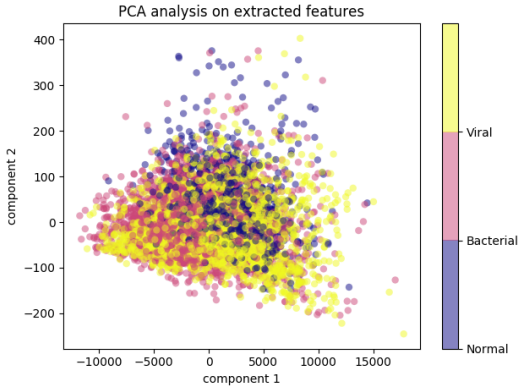
We need to look at the datasets and consider how separable the datasets are. Looking at the first two principal components in each image in figure 3.3 we can see how feature selection and scaling contribute to separate the three classes using passive learning methods. After feature extraction and scaling the two classes start to separate, which should simplify classification with supervised learning methods.



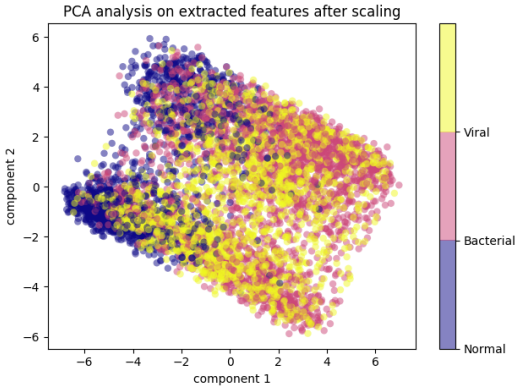
(a) Principal components of images before scaling



(b) Principal components of images after scaling.



(c) Principal components of features before scaling



(d) Principal components of features after scaling

Figure 3.3: The first two principal components on the images directly and on the selected features. We separated the dataset into three classes, which becomes increasingly apparent when using PCA on our feature selection.

Chapter 4

Theoretical Background

In this project we will classify images among three categories and need tools for this task. The two classification methods we chose to use are Support Vector Machines(SVM) and Convolutional Neural Network(CNN)s. While CNNs handle images well because of their structure, SVMs require careful image processing before they can be applied to image classification. The approach to these two methods is therefore a bit different.

4.1 SVM

The Support Vector Machine use a hyper plane to make a separation in a data set to make predictions. Generally, a p-dimensional plane is described by

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0. \tag{4.1}$$

which can be rewritten more succinctly as $\mathbf{x}^T\boldsymbol{\beta} + \beta_0 = 0$.

The distance $\mathbf{d}$ for some point $\mathbf{x}$ to the hyper plane is $\mathbf{d} = \mathbf{x} - \mathbf{x}_0$ where $\mathbf{x}_0$ is some point on the plane. Since $\boldsymbol{\beta}^T(x_0 - x_1) = 0$ for two points on a plane we get the signed distance δ perpendicular to the plane when

$$\boldsymbol{\beta}^T\boldsymbol{d} = \boldsymbol{\beta}^T(\boldsymbol{x} - \boldsymbol{x}_0) \tag{4.2}$$

$$\delta = \frac{1}{||\boldsymbol{\beta}||}(\boldsymbol{\beta}^T\boldsymbol{x} + \beta_0) \tag{4.3}$$

For any point not on the hyper plane δ is positive for one side of the plane and negative for the other. Thus, we can use this to make a classification given an appropriate hyper plane [6]. For multiple classes we would need multiple hyperplanes to divide the data sets further.

That the signed distance gives a negative number on one side and a positive number on the opposite side of the hyperplane, gives a natural cost function if we assign the two classes per hyperplane $y_i = 1 \wedge y_i = -1$

$$C(\boldsymbol{\beta}, \beta_0) = -\sum_i y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) \tag{4.4}$$

The product will always be positive for a correct classification and negative for a wrong classification.

However, when we try to minimize equation 4.4 using gradient descent it is not very likely to produce a unique solution. Since there are many planes which would fit well with the subscription above, we introduce the maximal margin classifier.

A margin is the distance between two parallel planes, and we want to maximize that distance. For a properly normalised dataset we have that the two planes at the margins are described by (4.5) for one

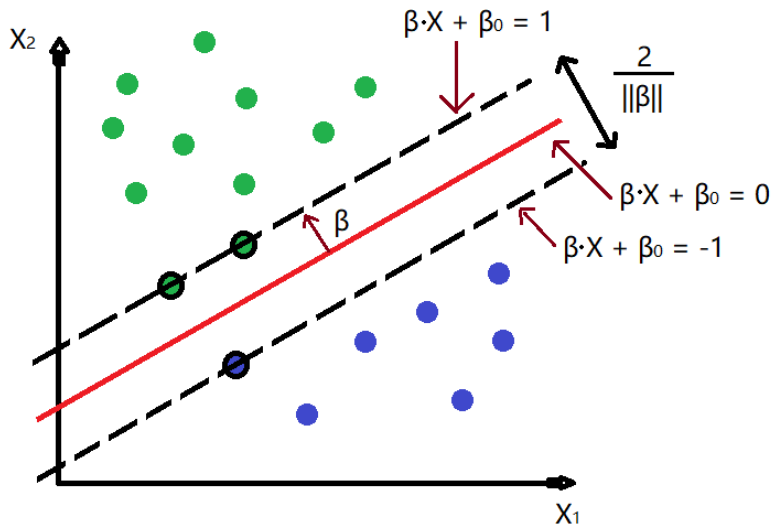


Figure 4.1: The maximal margin classifier tries to maximize the margin $M = \frac{1}{||\boldsymbol{\beta}||}$

side and (4.6) for the other side.

$$\boldsymbol{\beta}^T \boldsymbol{x} + \beta_0 = 1 \tag{4.5}$$
$$\boldsymbol{\beta}^T \boldsymbol{x} + \beta_0 = -1 \tag{4.6}$$

As described in equation 4.3, the distance between two planes is the signed distance δ, and we want to find the plane described in figure 4.1 where the mid plane require that the positive signed distance is equal to the negative signed distance $|\delta_+| = |\delta_-|$. Thus we want to maximize the distance

$$\text{maximize } |\delta_- - \delta_+| = 2|\delta| = \frac{2}{||\boldsymbol{\beta}||} = 2M \tag{4.7}$$

, where M is the margin, and we see that we need to minimize $||\boldsymbol{\beta}||$.

This classifier relies on the points closest to the margin and selects the points which gives the largest margin. These points are called support points and the vector pointing from the plane to the support point are called support vectors. This method makes the model more deterministic since we now should get just one hyperplane for any given training set. The maximal margin hyper plane only consider these support vectors, but not the remaining data set. The resulting classifier becomes

$$\text{maximize}_{\boldsymbol{\beta}, M} \quad M \tag{4.8}$$
$$\text{subject to } \boldsymbol{\beta}^2 = 1 \tag{4.9}$$
$$y_i(\beta_0 + \mathbf{x}_i^T \boldsymbol{\beta}) \geq 1 \ \ \forall i \tag{4.10}$$

where i is the different sample points, p is the number of features, β are the weights used to fit the model and M is the margin which we want to maximize.

However, since the hyperplane depends just on the closest points the model could dramatically change if any of the support vectors change, meaning it is sensitive to over-fitting. We thus introduce the slack variables $\boldsymbol{\epsilon}$. These allow room for misclassifications by reducing the margin on a point for point basis. This soft margin reduce the variance due to reducing the reliance on the closest support vectors. We now have the Support Vector Machine and the classifier is

$$\text{maximize}_{\boldsymbol{\beta},M} \ \ M \tag{4.11}$$

$$\text{subject to } \ \boldsymbol{\beta}^2 = 1 \tag{4.12}$$

$$y_i(\beta_0 + \mathbf{x}_i^T\boldsymbol{\beta}) \geq 1 - \epsilon_i \tag{4.13}$$

$$\epsilon_i \geq 0, \ \sum_i \epsilon_i \leq \tilde{C}. \tag{4.14}$$

$\tilde{C}$ is regularization strength we need to provide, and controls the amount of misclassification allowed. A larger $\tilde{C}$ value leads to a wider margin but more margin violations, using a smaller $\tilde{C}$ will give fewer margin violations but we end up with a smaller margin[5]. In most of the litterature we tune instead the regulariztion parameter C which is inversely proportional to the regalurization strength,

$$\tilde{C} \propto \frac{1}{C}. \tag{4.15}$$

When we later tune the model it is the parameter C we will tune.

For any given dataset, it is not always possible to make a meaningful boundary decision in the given dimension. By changing the dimension of the data, we can find new decision boundaries which divide the feature space in a better manner. Doing this is to change the kernel of the predictor, written as $K(x_i, x_j)$.

Popular Kernel functions are polynomial, linear, radial basis function and the sigmoid function, but depending on the dataset different Kernel functions can offer better performance. Different kernels introduce different tunable parameters.

4.1.1 Kernels

We used four kernels in an attempt to find a best fit. While it is simple enough to inspect data sets with few features and see which kernel might make a good fit, for more features it's not necessarily an easy spot.

- Linear Kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{y}$
- Polynomial Kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = (\gamma\boldsymbol{x}^T\boldsymbol{y} + c)^d$
- Radial Basis Function: $K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma||\boldsymbol{x}^T\boldsymbol{y}||^2\right)$
- Sigmoid Function: $K(\boldsymbol{x}, \boldsymbol{y}) = \tanh\left(\gamma\boldsymbol{x}^T\boldsymbol{y} + c\right)$

The Linear Kernel is the simplest kernel and models a simple flat hyper plane. The polynomial kernel lifts the data set to $d-1$ additional dimensions and has three additional parameters, γ, c and d. The radial basis function with one additional parameter γ. The sigmoid function has the additional parameters γ and c. The sigmoid function is not actually a kernel function since it asymmetrical. Still, it is popular since it seems to work well.

For all the kernels the different parameters have similar purpose. The parameter C has, as mentioned previously, the purpose of controlling misclassification. γ acts as a scale factor and scales the dataset. The parameter c shifts the kernel away from zero.

4.2 Feed Forward Neural Network(FFNN)

Neural networks are inspired by how biological neurons work. They work by feeding the input features corresponding to a datapoint to an initial layer of neurons. Summing the features multiplied by weights and adding a bias is the input for an activation function in the neuron. The resulting activations from all the neurons in the first layer are then fed forward to the next layer of neurons(called a hidden layer) and also multiplied by weights and added biases. This continues through all the hidden layers in the network until the output layer is reached. In the output layer we would have the same number of neurons as classes in the problem we are examining. The outputs of the neurons in the output layer are then read as probability that the initial datapoint belongs to a corresponding class.

We included this as a stepping stone to CNN's and we think it is a good idea to include it for comparison and to show the advantages of CNN.

4.3 Convolutional Neural Network (CNN)

Convolutional neural networks emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s.[5] A Convolutional Neural Network is a general Deep Learning algorithm, which is often used to analyse images due to their ability to catch main features from the image directly and recognising patterns allowing to accurately differentiate between images. CNN is chosen as a method here because the algorithm requires much less pre-processing compared to other classification algorithms[13]. CNNs are able to capture the spatial features in images with convolutional layers. Pooling layers reduces the size of the data output from a previous layer so that it is easier to process further. The dense layers performs the final part in the CNN, in our case the classification, based on the features captured by the previous layers. If the images are very simple and well formatted, we may get almost same results/accuracy with a FFNN. If the image however is complex and not that well formatted, CNNs will perform better because they are translationally invariant. FFNN not being translationally invariant means that if all the shapes that define the different classes are in the same location in every image a FFNN could capture them. However if they are in different locations, like the pneumonia in our images is, a FFNN can have a hard time making meaningful predictions. A CNN however is translationally invariant so that the location of the spatial features in the image does not matter.

Moreover, FFNN would break down for large images due to the huge number of neurons it requires to make good predictions [15]. Also in the X-ray images the chests are in slightly different positions and the pneumonia can also be concentrated in different locations in the lungs.

4.3.1 Convolutional layer

The most important building block of a CNN is the convolutional layer. The main task of the convolutional layer is to extract features from the image, by scanning the image with a filter. Which means that the convolutional layers computes the output by applying the kernel to an input array. Neurons in the first convolutional layer are not connected to every single pixel in the input image, but only to pixels in their receptive fields[5]. Conv2D layers were used in our model, it means that the input of the convolution operation is 3 dimensional. "2D" in "Conv2D" actually means that the filter will move through the image in two dimensions. For example, for each 5×5 pixel region of the image, the convolution operation computes the dot products between the values and the weights which is defined in the filter. As we can see in Figure 4.2, the original image is a black and white image, each position represents the pixel value. 0 represents white and 1 represents black. For this 4×4 image, we use two convolution kernels of 2×2. We start by taking the dot product of the filter with the 2×2 submatrix in the top left corner of the image. Here we set the step size to 1, which means that the next submatrix we make of the image to dot with the filter is shifted 1 position to the right. When the right side of the image is reached return to the left side and start again 1 row down. Taking the first convolution kernel filter1 as an example, the calculation of the feature map is as follows:

$$fm1(1,1) = 1 * 1 + 0 * (-1) + 1 * 1 + 1 * (-1) = 1 \tag{4.16}$$

$$fm1(1,2) = 0 * 1 + 1 * (-1) + 1 * 1 + 1 * (-1) = -1 \tag{4.17}$$

$$fm1(1,3) = 1 * 1 + 0 * (-1) + 1 * 1 + 0 * (-1) = 2 \tag{4.18}$$

The resulting feature map for the whole image for two filters can be seen in Figure 4.2.

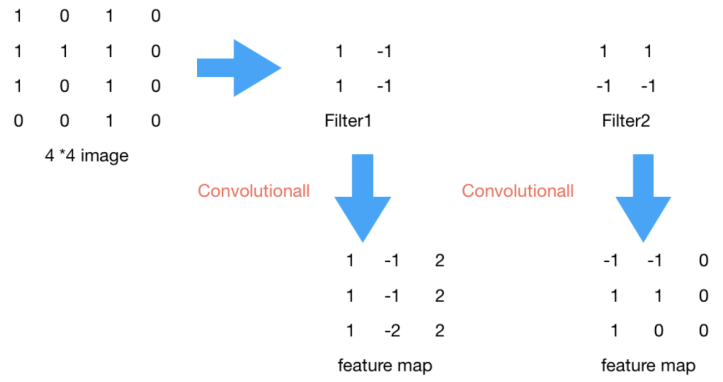


Figure 4.2: Convolutional Layer example [4]

The process of applying the convolutions can be seen as sliding a filter over the image. To better capture the features near the edges in the image padding can be added around the image in the form of rows and columns of zeros so that the filter can be applied further out 'over' the edges of the image. This is especially useful for larger filters. The process is exemplified in Figure 4.3

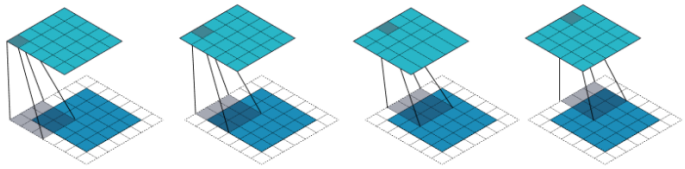


Figure 4.3: Convolution of a 5x5 input (blue) with 3x3 kernel (grey) with a stride of 2 and padding of 1. The 3x3 output is in green. [7]

Then we could write the input equation of every convolutional layer :

$$V = conv2(w, x, 'valid') + b \tag{4.19}$$

where w is the filter matrix (weights), x is the input matrix, 'valid' is the type of the convolutional computation which is described below and b is the bias. The output is as follows:

$$Y = \mathcal{C}(V) \tag{4.20}$$

where $\mathcal{C}$ is the activation function. ReLu was used in this project:

$$f(x) = max\{x, 0\} \tag{4.21}$$

It gives an output x if x is positive and gives 0 otherwise.

According to the convolutional layers' work process:

$$\delta_{11}^{L} = w_{11}^{L}\delta_{11}^{L-1} + w_{12}^{L}\delta_{12}^{L-1} + w_{21}^{L}\delta_{21}^{L-1} + w_{22}^{L}\delta_{22}^{L-1} \tag{4.22}$$

$$\delta_{12}^{L} = w_{11}^{L}\delta_{12}^{L-1} + w_{12}^{L}\delta_{13}^{L-1} + w_{21}^{L}\delta_{22}^{L-1} + w_{22}^{L}\delta_{23}^{L-1} \tag{4.23}$$

$$\delta_{21}^{L} = w_{11}^{L}\delta_{21}^{L-1} + w_{12}^{L}\delta_{22}^{L-1} + w_{21}^{L}\delta_{31}^{L-1} + w_{22}^{L}\delta_{32}^{L-1} \tag{4.24}$$

$$\delta_{22}^{L} = w_{11}^{L}\delta_{22}^{L-1} + w_{12}^{L}\delta_{23}^{L-1} + w_{21}^{L}\delta_{32}^{L-1} + w_{22}^{L}\delta_{33}^{L-1} \tag{4.25}$$

It is easy to find that they share the same weights which means the change of the weights w_{ij}^{L} will affect all δ_{ij}^{L}, and thus to get a certain weight, we need to use the components of total differential equation

$$\frac{\partial \mathcal{C}(\hat{W^L})}{\partial w_{11}^{L}} = \delta_{11}^{L}\delta_{11}^{L-1} + \delta_{12}^{L}\delta_{12}^{L-1} + \delta_{21}^{L}\delta_{21}^{L-1} + \delta_{22}^{L}\delta_{22}^{L-1} \tag{4.26}$$

$$\frac{\partial \mathcal{C}(\hat{W^L})}{\partial w_{12}^{L}} = \delta_{11}^{L}\delta_{12}^{L-1} + \delta_{12}^{L}\delta_{13}^{L-1} + \delta_{21}^{L}\delta_{22}^{L-1} + \delta_{22}^{L}\delta_{23}^{L-1} \tag{4.27}$$

$$\frac{\partial \mathcal{C}(\hat{W^L})}{\partial w_{21}^{L}} = \delta_{11}^{L}\delta_{21}^{L-1} + \delta_{12}^{L}\delta_{22}^{L-1} + \delta_{21}^{L}\delta_{31}^{L-1} + \delta_{22}^{L}\delta_{32}^{L-1} \tag{4.28}$$

$$\frac{\partial \mathcal{C}(\hat{W^L})}{\partial w_{22}^{L}} = \delta_{11}^{L}\delta_{22}^{L-1} + \delta_{12}^{L}\delta_{23}^{L-1} + \delta_{21}^{L}\delta_{32}^{L-1} + \delta_{22}^{L}\delta_{33}^{L-1} \tag{4.29}$$

which means we can get a general equation as following:

$$\frac{\partial \mathcal{C}(\hat{W^L})}{\partial w_{ij}^{L}} = conv2(\delta_{ij}^{L}, \delta_{ij}^{L-1}, ' \, valid') \tag{4.30}$$

Similarly, we can get:

$$\frac{\partial \mathcal{C}}{\partial b_{ij}^{L}} = \delta_{11}^{L} + \delta_{12}^{L} + \delta_{21}^{L} + \delta_{22}^{L} = \sum_{i}\sum_{j}\delta_{ij}^{L} \tag{4.31}$$

And the δ_{ij}^{L} is as follows:

$$\delta^{L-1} = conv2(rot180(w^2), \delta^{L}, ' \, valid')\mathcal{C}'(V^{L-1}) \tag{4.32}$$

This is a recursive formula. In this formula, the function rot180() means to rotate the matrix 180 degrees counterclockwise, and the parameter 'valid' represents the type of the convolutional computation. Here, the number of zeros layers is equal to the size of the convolution kernel matrix minus 1.

4.3.2 Pooling layer

The pooling layer is mainly for reducing the spatial size of the convoluted feature, in order to reduce the computational power through dimensionality reduction of the output from the convolutional layers. Each neuron in a pooling layer is connected to the outputs of a limited number of neurons in the previous layer. Note that a pooling neuron has no weights, it just aggregates the inputs using max or average. Since the optimization complexity grows exponentially with the growth of dimension. It extracts the dominant features in an area of the output from the convolutional layers. [5] Max pooling is a form of down-sampling and also noise reduction process. Max pooling works similarly to convolutional layers. It uses a kernel instead of a filter with a step size larger than 1. Max pooling simply looks at all the values in a given submatrix and selects the largest value as the output, only the max input value in each kernel makes it to the next layer, other inputs will be dropped. Average pooling is another method of pooling that instead returns the average of all the values in the kernel as the output. Average pooling can be seen as simply noise reduction while max pooling also extracts dominant features [5]. Based on these, Max pooling can keep the position and rotation of the feature constant, which is great for image processing. Also, it can reduce the number of model parameters and reduce problems with over-fitting. When 2D or 1D arrays are converted to a single value, it reduces the number of single filter parameter or hidden layer neurons for the subsequent convolution layer or full connection hidden layer. In this project we use Max pooling.

4.3.3 Fully connected dense layer

We use fully connected dense layers as the last layers in the neural network. This is done by flattening the output from the last pooling and convolutional output, and use this as the input to the dense layer. The purpose of the fully connected dense layers is to perform the classification based on the features extracted by the convolutional layers. In this part, we use the Softmax function as our activation function to do the classification, which is similar to what we did in the project2. For a given feature vector x, if we want to figure out the probability for each of these categories i, $P(y = i|x; \theta)$, then the results of our hypothesis function would be a C dimensional vector whose sum of the vector elements is 1, represents estimated probability values of these C types. So for each sample, the probability that it belongs to category k is:

$$a_k^L = \frac{e^{z_k^L}}{\sum_{n=1}^{C} e^{z_n^L}} \tag{4.33}$$

where $\forall i \in 1...C$ Take the derivative of above equation:

$$\frac{\partial a_k}{\partial z_j} = a_j(\delta_{kj} - a_k) \tag{4.34}$$

We used cross-entropy as loss function. The cross-entropy function between two probability distributions, p and q, can be written as $H(p, q)$.

$$H(p, q) = \sum_{x \in \mathcal{X}} p(x) log(q(x)) \tag{4.35}$$

Where p is the target distribution and q is the approximation of the target distribution, both follow a distribution with parameter p from 0 to 1, $X \sim B(1, p)$.

4.3.4 Dropout

Dropout refers to ignoring units(or neurons) during the training stage, a certain set of neurons is chosen randomly, these neurons are then not considered during forward and backward pass.

At each training stage, some neurons are dropped out of the network, with the probability $1 - p$, then we will have a smaller network, thus the training time required for each epoch will be less. A fully connected dense layer will occupy most of the parameters, therefore, there will be a co-dependency among each neuron during training process. This will lead to inhibition of individual power of each neuron which may cause over-fitting.

One drawback could be that dropout will almost double the number of iterations required to converge. With dropout, the training accuracy may suffer, but the test accuracy will increase, since during the test phase all neurons will be used to make predictions.

4.4 Receiver operating characteristic(ROC)

There are some metrics that we can use to determine the fitness of the model, for example precision, recall and f1-score, described in the table 4.1. Precision is the classifiers ability to avoid labelling a negative to a positive. Recall is a classifiers ability to find all positive instances. Lastly, the f1-score is a weighted mean between recall and precision. This value is meaningful when considering other classifiers on the same dataset and not to be considered as a global average.

Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
f1-score	$2 \times \frac{Recall \times Precision}{Recall+Precision}$

Table 4.1: A list of useful definitions of metrics used to gauge the fitness of a model. TP is True Positives, FP is False Positive, TN is True Negative and FN is False Negative.

ROC graphs are useful for visualizing the performance of classifiers. It is created by plotting the true positive rate against the false positive rate at different threshold settings. ROC analysis helps us to select possibly optimal models and remove the suboptimal ones independently from the class distribution. It is tight related to cost/benefit analysis of diagnostic decision making.

$$Accuracy = \frac{\text{True positives+True negatives}}{\text{Positives+Negatives}}$$

$$\text{Sensitivity} = \frac{\text{True positives}}{\text{False negatives+True positives}}$$

= probability of a positive test given that the patient has the disease

$$\text{Specificity} = \frac{\text{True negatives}}{\text{False positives+True negatives}}$$

= probability of a negative test given that the patient is healthy

In this project we plotted one ROC curve per class, Normal, Bacterial and Viral in order to visualize the performance of classifiers.

Chapter 5

Algorithms and implementation

Here we will discuss the different methods and algorithms we have used. We first discuss the different methods we used to prepare the data set, both to understand it and our efforts to separate the different classes from each other to make learning easier and more efficient.

5.1 Feature extraction

To extract valuable features from the image we started out with standardising the images. All the images need to be the same dimension and format. The feature values can also be re-scaled and possibly centred. Once that is achieved we need to extract features.

5.1.1 Flatten images and features

The format acceptable to SVM's are of matrix form, with dimension (samples, features). To achieve this format, we flatten the image or feature from matrix to array shape and append all the images into a single list, and we end up with a matrix. While it is necessary to flatten the image when using a SVM or a dense neural network, the CNN requires that the images are unflattened for the image processing layers to work.

5.1.2 Convert to grey scale

Since most images are grey scale, but some are coloured, we ensure that all images are grey scale. We feel this is a good approach since the X-ray images are in fact grey scale and no information is stored in colours, which is apparent upon inspection.

5.1.3 Resize image

The size of the images needed some consideration. Since we risk loosing information when downsizing the images. However, the database is large, containing 5855 images, so we need a manageable image size. All the images are larger than 400×300, so the images should not be larger than the smallest image. Additionally, some of the images are square while some are rectangular. We chose the image size of 150×150 for the reason that such a size is manageable memory wise, while still large enough to retain enough information. Larger images would probably yield better results, but their size would be prohibitive at the moment.

5.1.4 PCA components

PCA is useful at extracting features with maximum variance. It is possible to extract where and how an image varies the most, and use this as features in image recognition.

```
for image in images:
    pca = PCA(n_components)
    pca.fit(image/255-0.5)
```

```
    pcaComponents = pca.components_
    features.append(pcaComponents)
```

Listing 5.1: Python code for extracting PCA features

The image is normalized by dividing by the max possible value for a pixel, 255 and subtract 0.5 to center it. The variable n_components is found using a simple grid search.

5.1.5 Kernel Density Estimation

Kernel Density Estimation is a powerful tool to capture unique features in an image by capturing pixel frequencies in the image.

```
for image in images:
    kde = KernelDensity(bandwidth)
    kde.fit(image/255)
    kdeComponents = kde.score_samples(img/255)
    features.append(kdeComponents)
```

Listing 5.2: Python code for extracting KDE features

Bandwidth is a variable we need to find using a simple grid search.

5.1.6 Haralick texture recognition

The Haralick texture recognition is a potentially useful tool for categorising textures in an image. Given an image, it gives a numerical value to the prevalence of certain features, meant to capture the different textures present.

```
for image in images:
    haralickComponents = np.mean(mt.features.haralick(img), axis=0)
    features.append(haralickComponents)
```

Listing 5.3: Python code for Haralick

It appears it is not possible to normalize the image before applying Haralick, and there is no tuning necessary.

5.1.7 Recursive Feature Elimination(RFE)

Recursive Feature Elimination is a method to remove least important features. It does this by fitting the model to n features and rank these features after importance. Then the least important feature is removed and the model is fitted again with $n-1$ features and the process is repeated until there are no more features to remove. Since there is no way to know which features are the least important the model need to run the fit n times for each cross validation iteration to know in which order to remove the least important feature, since the features work together and it is not possible to know which features are least important considered in total [14].

5.2 SVM model

The Support Vector Machine is implemented from SK-Learn using the lines

```
svc = SVC(C, kernel, parameters, gamma='scale')
svc.fit(X, y)
```

Listing 5.4: Python code for implementation of SVM

The model is now fitted to the data X using y to measure how well the model is doing. The parameter C is always included and controls the SVM's soft/hard border for tolerating misclassifications. The parameter kernel controls how the data is lifted into higher dimensions in order to make an easier fit. Depending on the kernel chose, we need different parameters.

5.3 CNN model layers

Inspired by the model from [16] and VGG16 Convolutional Network for Classification[17] which also performs learning on x-ray images. After trying different number of nodes at each layer, different filter sizes as well as different number of layers, we ended up with the model below which gave us the best results.

5.3.1 Conv2D layer

Ten Conv2D layers was added to the model, with a MaxPooling2D layer after every other conv2D layer. The first conv2D layer is used to identify features in the original image, then to identify sub-features within smaller parts of the image. The output of Down-sampling or pooling was preformed to the results from the previous convolutional layer. After going though all the layers, the results should be the essential features that can help the model to classify the image.

```
model.add(Conv2D(16, (3, 3), activation='relu', padding="same", input_shape= (150,150,1)
    ))
model.add(Conv2D(16, (3, 3), padding="same", activation='relu'))
```

Listing 5.5: Python code for first two convolutional layers in the model

```
model.add(Conv2D(128, (3, 3), dilation_rate=(2, 2), activation='relu', padding="same"))
model.add(Conv2D(128, (3, 3), padding="valid", activation='relu'))
```

Listing 5.6: Python code for last two convolutional layers in the model

16 is the number of different filters in the convolutional layer and (3,3) is the size of the filters. We double the number of filters every second layer, the higher the number of filters, the higher the number of abstractions our model will be able to extract. Since the first layer will receive raw pixel data, and they are usually noisy, so we let the model extract some relevant information from the "noisy" data first. While we go deeper in layers and after some useful information have been extracted, the feature map size decreases, then we make the model elaborate more complex abstractions by increasing the number of filters.

Padding = "same" specifies that the output size should be the same size as the input size, there is a one pixel width padding around the image, the filter slides outside the image into the padding area, generating output the same size as the input. An example of padding is shown in figure 4.3. Padding = "valid" means that the convolutional layer does not use zero padding, and may ignore some rows and columns at the bottom and right of the input image. [5]

dilationrate defines a spacing between the values in a kernel, for example, a 3×3 kernel with a dilation rate of 2 will have the same field of view as a 5×5 kernel, but it will only use 9 parameters by deleting every second column and row. This will give a wider field of view at the same computational cost.

We use rectified linear unit 'relu' as the activation in the convolutional layers, it is a bit faster to compute than other activation function, and the gradient descent does not get stuck as much, in addition, it does not saturate for large input values[5].

5.3.2 MaxPooling2D layer

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Listing 5.7: Python code for convolutional layers in the model

With no given step size the step size will default to the size of the pool size. With pool size (2,2) this will lead to the output being a fourth of the size of the input.

5.3.3 Fully connected dense layer

```
    model.add(Flatten())
    model.add(Dense(64, activation='sigmoid'))
    model.add(Dropout(0.35))
    model.add(Dense(n_categories , activation='softmax'))
```

Listing 5.8: Python code for dense and dropout layers in the model

Flatten reduces the dimension of the input data into a column vector so that it suits the dense layer, the flattened output is then fed through a dropout layer that is used during training. Then finally a dense layer with output nodes equal to the number of categories. Softmax was used as the activation function for the output layer, as it is generally good for classification tasks.

Dropout was added in order to prevent over-fitting and forces the model to learn from more robust features.

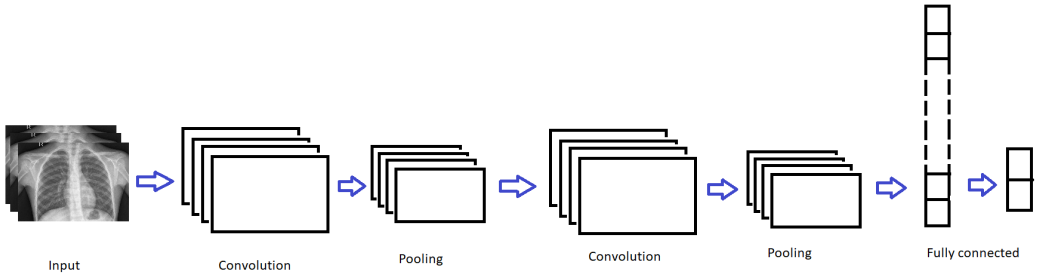


Figure 5.1: Convolutional neural network architecture

Chapter 6

Results and findings

6.1 Optimizing number of PCA components

Since uninformative features tend to reduce the model performance it can be helpful to try and determine the optimal number of features returned by the PCA component-wise features extraction method. To determine this we used a simple grid search method with a simple SVM classifier and found the optimal number of components to be about 4. The grid search can be seen in figure 6.1. For finding the optimal number of components we used a SVM with a linear kernel and $C = 1$. What we found was that approximately 3 or 4 features were optimal. We chose 4 features since we later planned on removing some features with other methods.

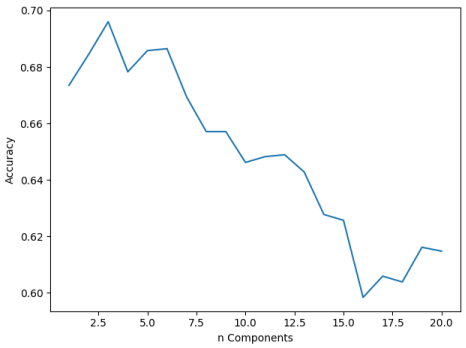


Figure 6.1: The optimal number of principal components to be included.

6.2 Optimizing Kernel Density Estimation

The Kernel Density Estimation algorithm is dependent on the variable Bandwidth. How informative the features are to the model seems highly dependent this variable. From figure 6.2 we see that the KDE algorithm work well for a region of bandwidth, but very quickly loose information outside this region. For our case we got a maximum at around bandwidth = 0.5. For values higher than about 800 the model did worse than bandwidths lower than 0.1.

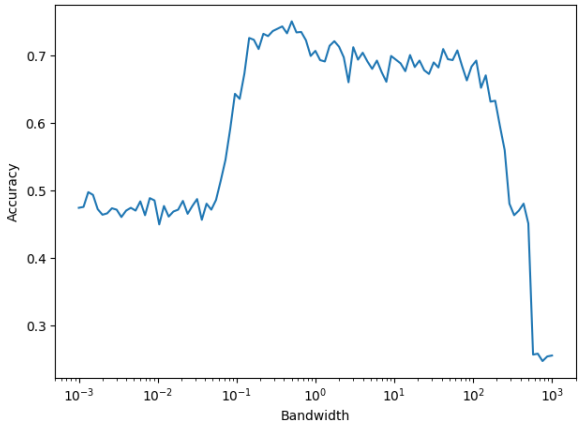


Figure 6.2: Using grid search for the optimal bandwidth yields a value around 0.5

6.3 Effect of combining features and scaling

When combining features there is a risk that the features replicate or otherwise confuse the model. In an effort to discover how the different features interact and look at the effects of scaling we can look at figure 6.3. From figure 6.3 we can see how concatenating the different features affected model accuracy.

We tested the image and that got good results. The image was scaled used a variant of the StandardScaler (We divided by 255 and subtracted 0.5 manually) before any other features were extracted, except for Haralick which require the image to be non-scaled. However, due to time constraints we ended up with looking at the other features for the final model, since the image itself contributed 22500 features, which is considerably larger than the number of features provided by the other methods.

Without scaling Kernel Density Estimation worked best for the support vector machine (for the linear kernel at $C = 1$) with PCA components worked almost as well as KDE. Haralick, benefited the most from scaling. However, when combining the features all combinations did poorly unless scaled. We chose to use all the feature extraction methods since we later would remove superfluous features.

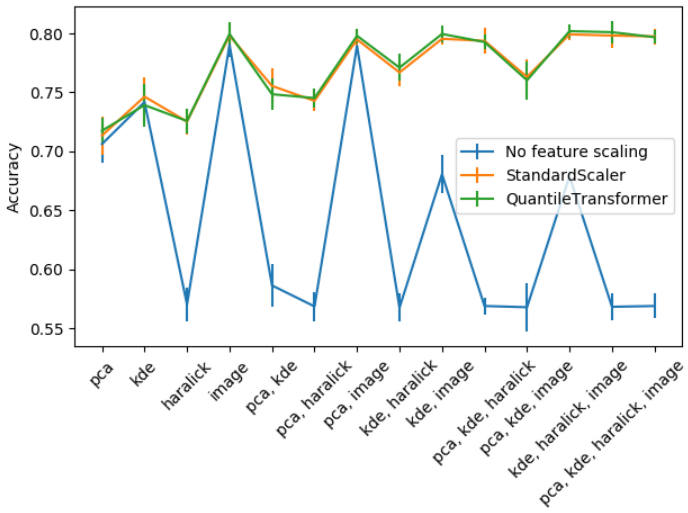


Figure 6.3: Without scaling, the features doesn't interact well. However, after scaling they instead reinforce each other.

We used two different types of scaling, StandardScaler and QuantileTransformer. For this dataset it is not clear if one is better, but the StandardScaler is somewhat faster.

6.4 Recursive Feature Elimination(RFE)

Since unimportant features can lead to poorer model performance we used Recursive Feature Elimination to filter out features which made the model performance worse. The elimination of features lead to an increase in model accuracy at around 75 features. For this data set the accuracy improved steadily until the peak was reached, after that, the model accuracy quickly dropped. The evolution of the model performance can be seen in figure 6.4. The process of Recursive Feature Elimination has the benefits of both improving accuracy and making later models easier to train since the number of features is reduced. This will in turn lead to a more comprehensive grid search.

6.4.1 Correlation Matrix for the final Features

Finally, we can look at the correlation matrix for the final features we are going to train on. Looking at the correlation matrix there are four distinct regions. There is visible difference between PCA, KDE and Haralick features. Additionally, the very last row and column are the target column. From the correlation matrix it can be seen that the targets are correlated with the KDE features, slightly negatively correlated with PCA features and the dependency on Haralick seem to vary quite strongly and negatively.

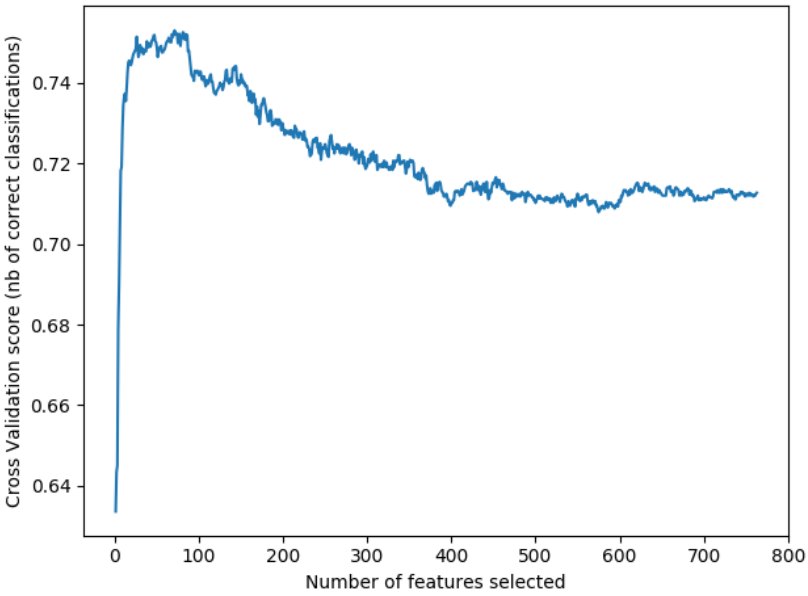


Figure 6.4: Recursive Feature Elimination. The presence of uninformative features confuses the model. Removing them gradually improves the model performance until we are only left with informative features.

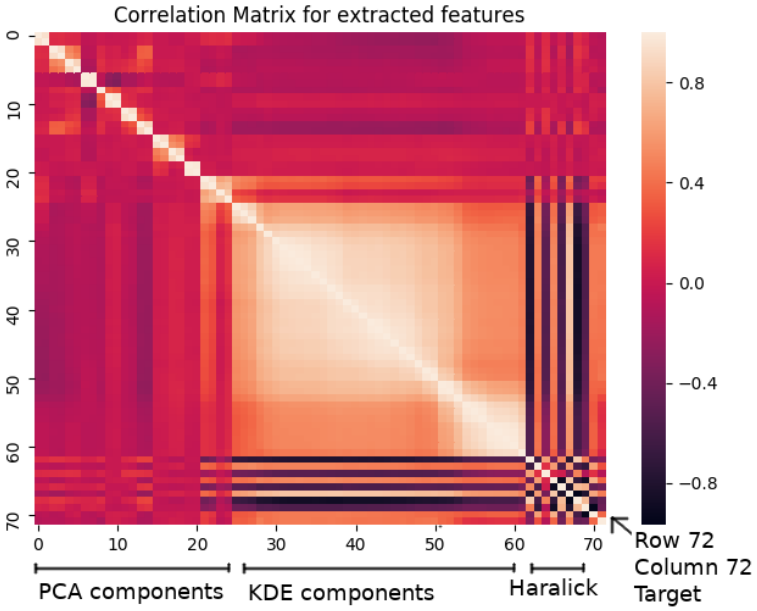
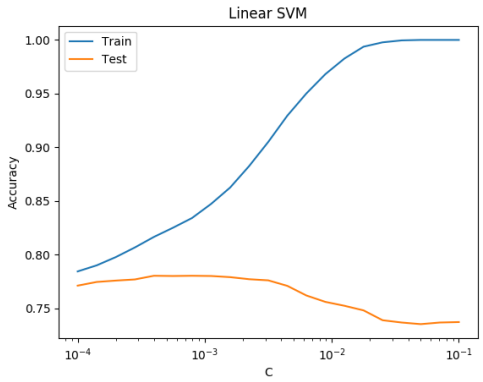
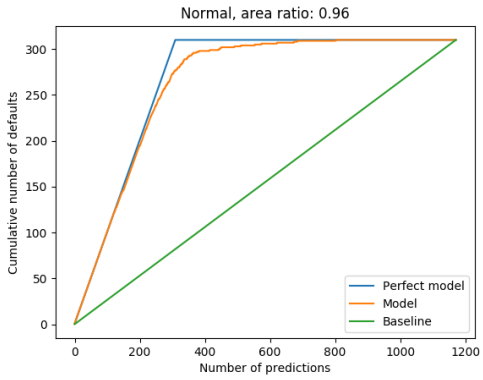


Figure 6.5: Correlation matrix for the final feature matrix, divided into four regions. The last row and column are the target. Darker colors indicates negative correlation and lighter colors indicates positive correlation

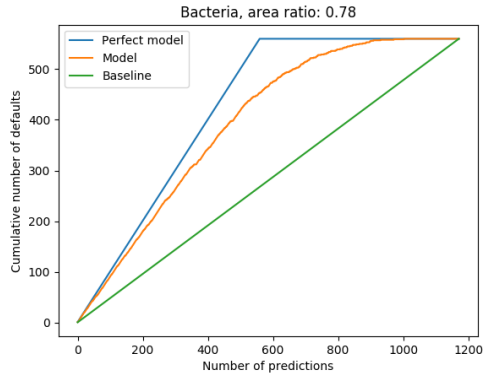
6.5 SVM cross validation



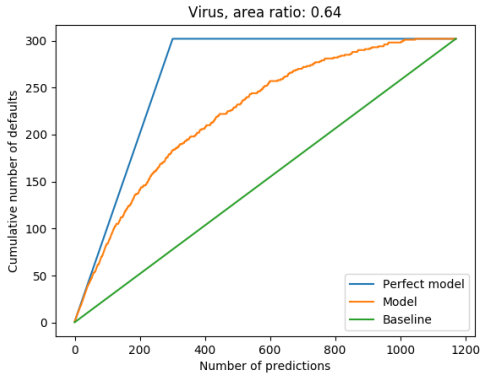
(a) Parameter C against accuracy.



(b) ROC curve for healthy people.



(c) ROC curve for bacterial pneumonia.



(d) ROC curve for viral pneumonia.

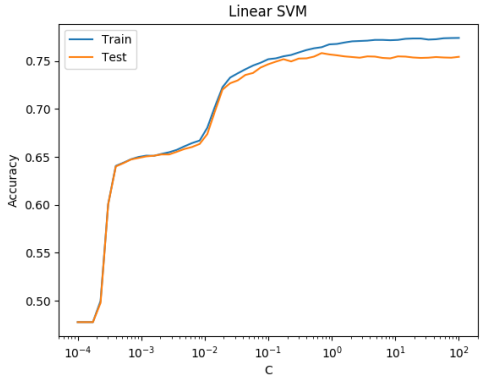
Figure 6.6: SVM for all PCA, KDE, Haralick and Image features.

While preparing the dataset we had to make some choices. If we use the best features from figure 6.3 we have the problem of feature explosion. The amount of features renders PCA to reduce number of features problematic and Recursive Feature Elimination took too long. We therefore tried to fit a linear model using cross validation. The results from this method is in figure 6.6 and $C = 0.0004$ turned out to make the best model.

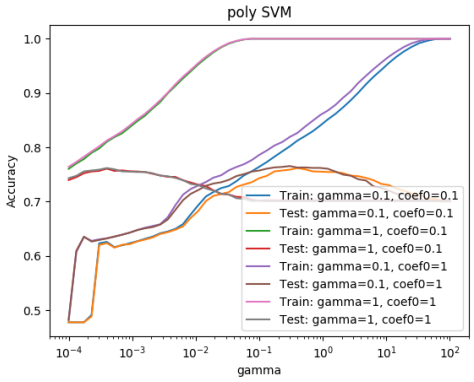
The other approach was to avoid using the image as features, which allowed for the possibility of using more methods further reducing the number of features and at the same time increasing the quality of the model. We prepared this data set using RFE as discussed in section 6.4.

With the dataset prepared we had to scan the parameter space. For the SVMs we tried the linear, polynomial, radial basis function and sigmoid function kernels for a total of 10 tunable parameters. We used grid search and cross validation with K-Fold with 3 folds. From figure 6.8 we see that the area both varies and there can be narrow peaks where the model makes a good prediction for both the train and validation sets.

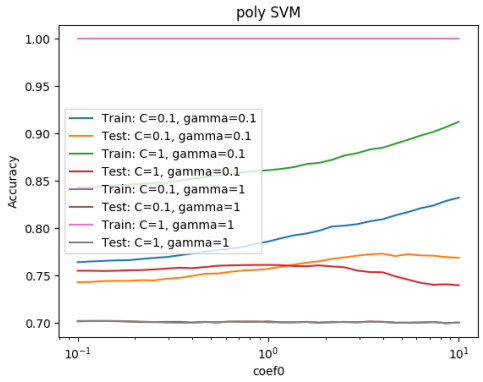
Looking at figure 6.7a it seems that the linear model is resistant to over fitting at least on this dataset, but still has poor performance for a too small value of C. Comparing to the polynomial kernel from figures 6.7b, 6.7c and 6.7d, we can see that the model requires that all four parameters (C, coef0,



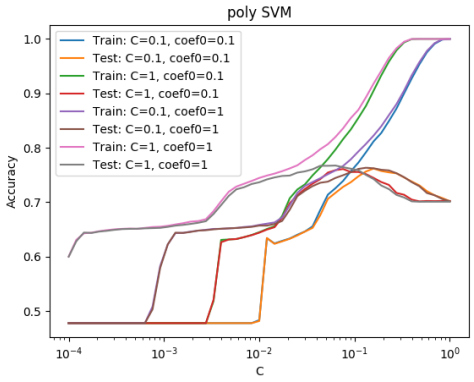
(a) Search area for linear SVM considering the C variable.



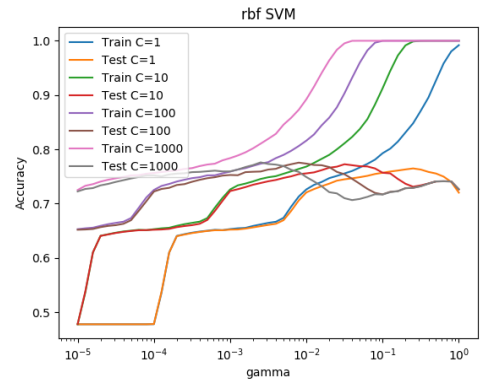
(b) Polynomial kernel considering variable C



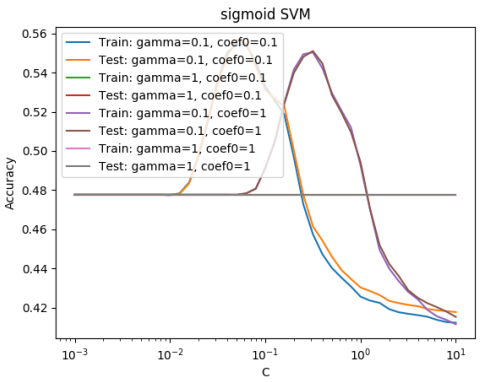
(c) Polynomial kernel considering the coef0 variable.



(d) Polynomial kernel considering the gamma variable.



(e) RBF kernel considering the C variable.



(f) Sigmoid kernel considering the C variable.

Figure 6.7: Some examples of how the accuracy of a model depends on the different variables.

gamma and degree) are sensitive.

From image 6.7e we see similar behaviour from the RBF kernel as we do from the polynomial kernel. The model is prone to over-fitting and both variables (C and gamma) contribute to a high accuracy. Finally, looking at figure 6.7f we see that this model has a very narrow area of maximum, while tapering off to a low accuracy.

From these preliminary looks at the model behaviour on different parameters, we made a grid search using K-Fold cross validation. From this we found a best model which we used to make some further considerations. The best model we found using this method were a model with the radial basis function kernel with $C = 31.62$ and $\gamma = 0.0178$.

It is also worth pointing out that the grid could have been expanded a bit, since it is possible that maxima is outside the search area, looking at figure 6.7f, for instance it looks like there should be an maximum at higher values for C, but since the grid search are time consuming we were forced to limit the search.

6.6 SVM using best model

To study the best models we found using cross validation we plotted the ROC curves for predicting the three conditions normal (fig. 6.8a), bacterial pneumonia (fig. 6.8b) and viral pneumonia (fig. 6.8c) when using the features from PCA, KDE and Haralick. From these curves it appears the model struggles to separate the two types of pneumonia, but is a lot better at predicting if someone has pneumonia or not. This is evident from the confusion matrix 6.1 where we see that the model wrongly predicts 16.0% of the normal X-ray images to be bacterial or viral pneumonia, while 3.3% of the bacterial pneumonia is predicted to be normal and 21.6% of the bacterial pneumonia was predicted to be viral. Likewise 7% of the viral pneumonia was predicted to be normal and 23.7% is predicted to be bacterial.

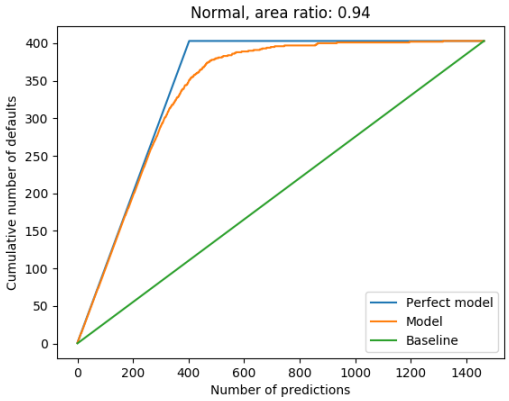
		Actual		
		normal	bacterial	viral
Predicted	normal	358	26	19
	bacterial	42	576	64
	viral	26	166	187

Table 6.1: Confusion matrix for Support Vector Machines on the test dataset.

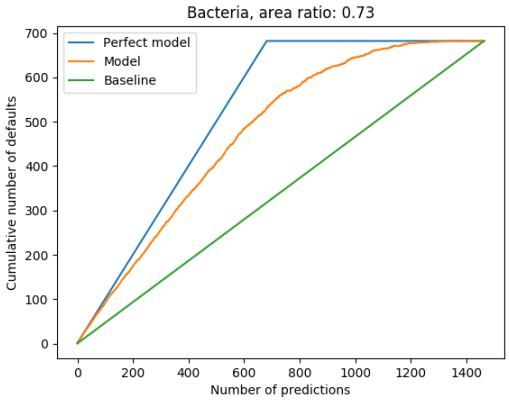
The results from using features from the image as well can be seen in figure 6.6b, 6.6c and 6.6d. Looking further at the precision and recall values from table 6.3 we see that the model is more accurate for the normal images than bacterial and the worst for viral. Also, the results are generally better when including the Image features. Looking at the weighted and macro averages in table 6.3 and table 6.4 we see that precision, recall and f1-score is a bit better. Still, trying to guess viral pneumonia is the hardest task for the models.

		Actual		
		normal	bacterial	viral
Predicted	normal	276	14	20
	bacterial	13	483	64
	viral	20	126	156

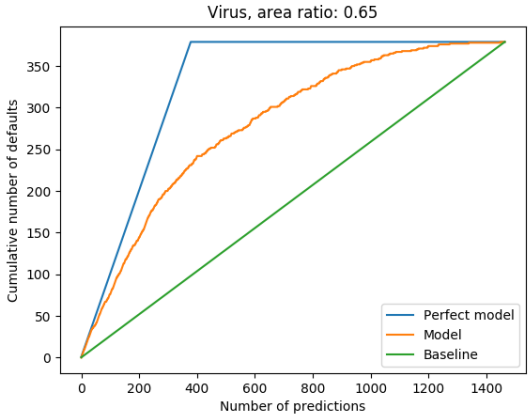
Table 6.2: Confusion matrix for Support Vector Machines on the test dataset while using the features PCA, KDE, Haralick and Image.



(a) ROC curve for normal using SVM.



(b) ROC curve for bacterial pneumonia.



(c) ROC curve for viral pneumonia.

Figure 6.8: ROC curve for normal, bacterial and viral pneumonia using SVM.

	precision	recall	f1-score
Normal	0.84	0.89	0.86
Bacterial	0.75	0.84	0.79
Viral	0.69	0.49	0.58
accuracy			0.77
macro avg	0.76	0.74	0.74
weighted avg	0.76	0.77	0.76

Table 6.3: Classification report for SVM using features from PCA, KDE and Haralick.

	precision	recall	f1-score
Normal	0.89	0.89	0.89
Bacterial	0.78	0.86	0.82
Viral	0.65	0.52	0.58
accuracy			0.78
macro avg	0.77	0.76	0.76
weighted avg	0.77	0.78	0.77

Table 6.4: Classification report for SVM using the features PCA, KDE, Haralick and Image.

Some considerations before moving on

While searching for the best SVM model, we found that there is a trade off between number of features to include and computation time. While more features with the right scaling were promising when considering model accuracy, certain methods becomes prohibitive in practice. We landed on two approaches, including the most important PCA components, KDE and Haralick vs. including the image as well. With the first approach Recursive Feature Elimination became viable as well as an extensive grid search with cross validation. This allows us to do a more thorough and exhaustive search in addition to the RFE bringing the up to almost the same level as using more features. On the other hand, with the image itself included, the number of features increased to a level where we were cut off from using Recursive Feature Elimination and limited the grid search severely. This means the first approach was a lot more exhaustive and we can be more confident with that the SVM is optimal on these features. Conversely, there is good reason to believe the performance on the feature space including the image can be improved.

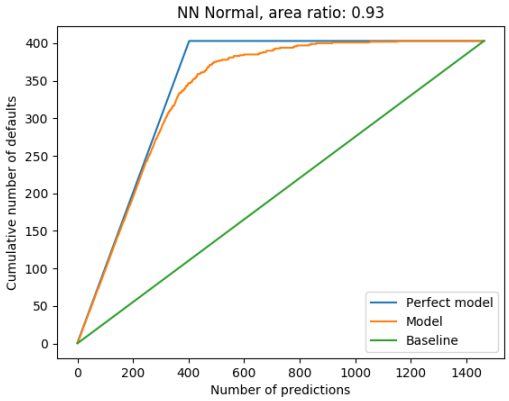
6.7 Simple Neural Network using features from SVM

Before implementing CNN we want to look at a simple feed-forward network using only Dense layers. Using a simple neural network consisting of only 3 Dense hidden layers, adam optimizer and the categorical cross entropy loss function we got the results from table 6.5, 6.6 and figure 6.9. With this network we got the worst results. However, the network could feasibly be better tuned.

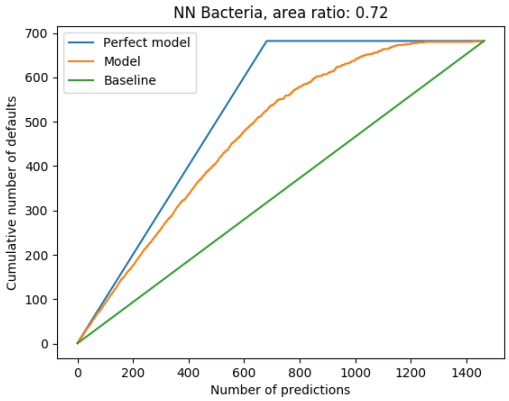
		Actual		
		normal	bacterial	viral
Predicted	normal	344	26	33
	bacterial	37	572	73
	viral	23	190	166

Table 6.5: Confusion matrix for NN on the test dataset.

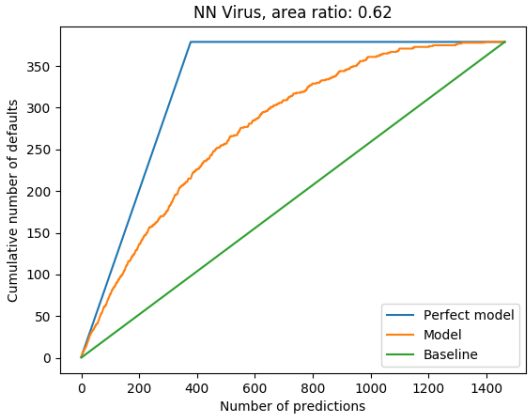
The cumulative plots are



(a) ROC for normal using NN.



(b) ROC for bacterial pneumonia using NN.



(c) ROC for viral pneumonia using NN.

Figure 6.9: Cumulative curves for normal, bacterial and viral pneumonia using NN.

	precision	recall	f1-score
Normal	0.85	0.85	0.85
Bacterial	0.73	0.84	0.78
Viral	0.61	0.44	0.51
accuracy			0.74
macro avg	0.73	0.71	0.71
weighted avg	0.73	0.74	0.73

Table 6.6: Classification report for plain NN.

6.8 Convolutional Neural Network (CNN)

For the CNN we worked with viral pneumonia and normal(healthy) in one class. Since only the bacterial pneumonia is treatable with antibiotics. With GridSearchCV we found that an initial learning rate of 0.000055 worked best. With a fixed CNN model structure the initial learning rate was the main

parameter to be tuned. In figure 6.10 we see the resulting ROC curve with the validation data. In Table 6.7 we have the resulting confusion matrix from the validation data.

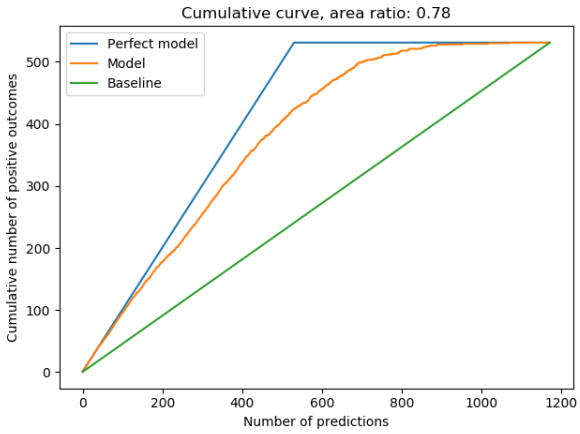


Figure 6.10: ROC for bacterial pneumonia using CNN

We also ran the CNN with bacterial and viral pneumonia in the same class to be able to compare our findings to :[11]

	precision	recall	f1-score
Normal and viral	0.86	0.77	0.81
Bacterial	0.77	0.85	0.81

Table 6.7: Classification report for CNN. With healthy patients and patients with viral pneumonia in one class, bacterial pneumonia in the other class.

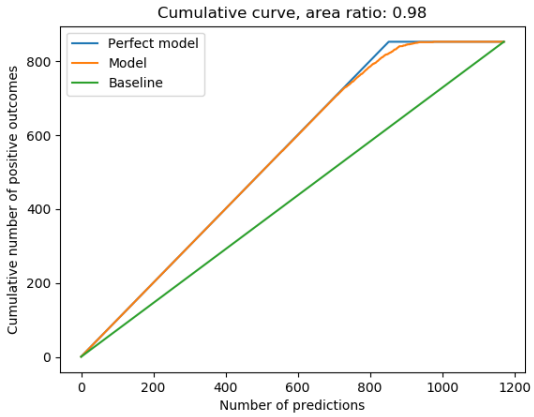
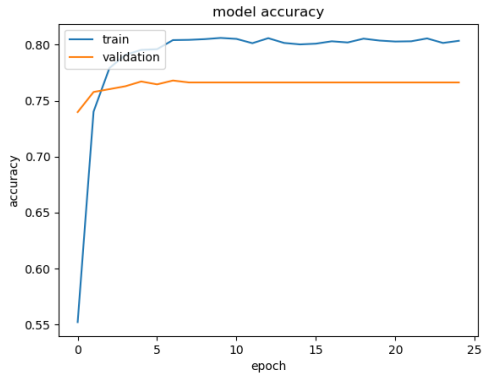


Figure 6.11: ROC for pneumonia(bacterial and viral) using CNN

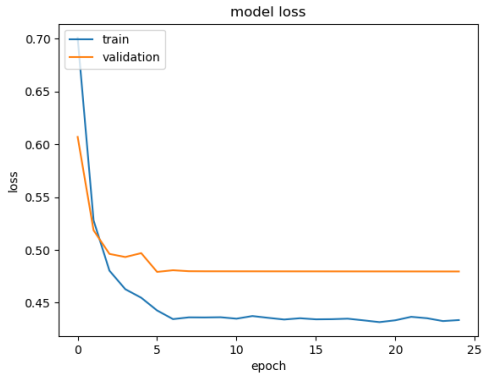
	precision	recall	f1-score
Normal	0.91	0.90	0.90
Pneumonia (Viral and Bacterial)	0.96	0.96	0.96

Table 6.8: Classification report for CNN, with bacterial and viral pneumonia in one class.

From 6.12 we can see how accuracy and loss for detection of bacterial pneumonia vary with number of epochs. Similarly, 6.13 shows the model performance on detection of pneumonia, including both bacterial and viral. We can clearly see from 6.12b and 6.13b that the model plateaus faster when training to detect bacterial pneumonia than when training to detect general pneumonia.

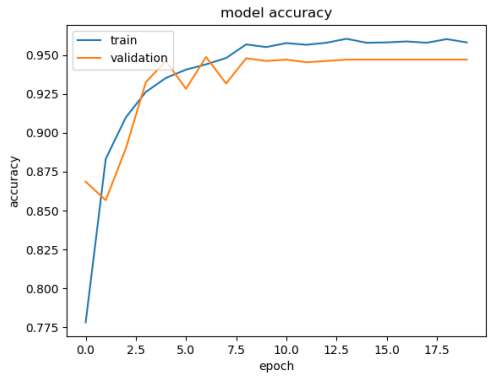


(a) CNN validation accuracy for bacterial after each epoch

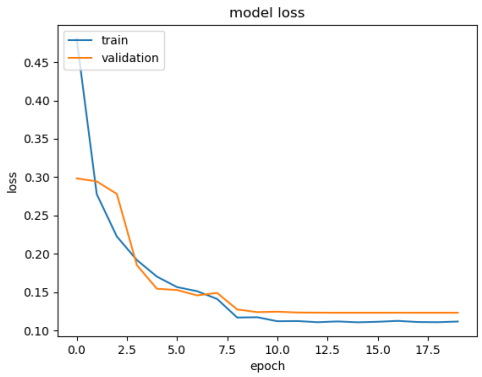


(b) CNN validation loss for bacterial after each epoch

Figure 6.12: CNN accuracy and loss for validation data. Viral pneumonia and normal in one class and bacterial pneumonia in the other class. Learning rate=0.000055



(a) CNN validation accuracy for pneumonia(bacterial and viral) after each epoch



(b) CNN validation loss for pneumonia(bacterial and viral) after each epoch

Figure 6.13: CNN accuracy and loss for validation data. Normal in one class and bacterial and viral pneumonia in the other class. learning rate=0.000055

6.8.1 Comparing our results with physicians using lung ultrasound

In table 6.9 we have a comparison of our results with those of physicians using lung ultrasound(LUS) which is seen to outperform physicians diagnosing from x-ray images [1] our specificity is lower so we do not have as good true negative rate but we have higher sensitivity, meaning that we have a higher true positive rate.

Method	Specificity	Sensitivity
SVM w/ PCA, KDE, Haralick	0.84	0.94
SVM w/ PCA, KDE, Haralick, Image	0.89	0.95
FFNN	0.85	0.95
CNN	0.91	0.96
Doctor w/ ultrasound	0.93	0.85

Table 6.9: Specificity and sensitivity comparing our models with the rate by physicians using lung ultrasound [1]

Chapter 7

Conclusions and Critical assessments

Using SVM we found that the best precision we had (precision 78%, recall 86%) for detecting bacterial pneumonia performed better than CNN (precision 77%, recall 85%). The SVM, however, has the disadvantage of being more reliant on data preprocessing. We speculate the performance of the SVM could be further improved with better feature extraction methods. Additionally, we could have studied the effects of scaling methods since from what we have seen in this project, the scaling affects the SVM performance noticeably.

The dense neural network, based on features from PCA, KDE and Haralick. Had a precision for finding bacterial pneumonia of 73% and recall of 84%. Which is close to SVM and CNN but slightly lower.

We see that our CNN does not perform as well as the baseline from [11]. Our precision for pneumonia vs normal 96% is close to the result obtained with convolutional neural network model VGG16 that they used in the paper, 97.7% in [11]. Our CNN precision for bacterial vs normal and viral of 77% is considerably worse than the baseline of VGG16, 91.7%, in [11].

In general we see that CNNs perform similarily to SVMs. When it comes to detection of bacterial pneumonia. Our models do perform well enough that they could be considered as supplementary tools to diagnose general pneumonia since they have a higher true positive rate than physicians using LUS 6.9. But do however not perform as well as the Neural networks in [11] at differentiating between viral and bacterial pneumonia which is crucial, since only cases of bacterial pneumonia should be treated with antibiotics. In conclusion our methods could possibly be helpful when diagnosing patients but do not work as well as other already developed machine learning methods like those of [11]. Our methods also don't take into account that X-ray images of patients suspected of pneumonia could have other diseases. To combat this we could possibly have used different datasets that also contain images of patients with other conditions. which in turn could lead to classification of other diseases from chest X-ray images and how the occurrence of other diseases impacts the precision of correctly detecting pneumonia.

One topic which we did not fully examine in this project was the effect of reducing the image size, it could be interesting to examine the effect on the accuracy of using higher resolution images. Further work could also be done in trying to determine the optimal structure of the CNN. Because of time constraints the structure we landed on was mostly developed through trial and error. Specifically how different structures affect accuracy, stability and training times and comparing to the performance of other machine learning algorithms.

Chapter 8

Appendix

8.1 List of codes on GitHub

`project3_header.py`

This file contains often used functions which most files share.

`featureExtraction_SVM.py`

Contains code for extracting features from the X-ray images. It also contains a set of flags in the beginning of the code to do tasks related to feature extraction, such as finding the best parameter for KDE and PCA or analysing the images and results.

`NN_X-ray.py`

Contains the setup for the FFNN.

`RFE.py`

For performing recursive feature elimination (RFE). This file require that featureExtraction.py has extracted and saved the features to hard drive.

`SVM_X-ray.py`

Here we perform the classification and analysis by reading in features created either using featureExtraction.py or RFE.py.

`SVM_X-ray_Image.py`

This is a variant of the SVM analysis code, but tooled for the features including the image.

`conv_xray.py`

Used to set up and train CNNs for classification of images.

Bibliography

[1] Saeed Ali Alzahrani et al. "Systematic review and meta-analysis for the use of ultrasound versus radiology in diagnosing of pneumonia". In: *Critical Ultrasound Journal* 9.1 (Feb. 27, 2017), p. 6. ISSN: 2036-7902. DOI: 10.1186/s13089-017-0059-y. URL: https://doi.org/10.1186/s13089-017-0059-y (visited on 12/18/2019).

[2] Thomas Cherian et al. "Standardized interpretation of paediatric chest radiographs for the diagnosis of pneumonia in epidemiological studies". en. In: *Bulletin of the World Health Organization* (2005), p. 11.

[3] *Chest X-Ray Images (Pneumonia) — Kaggle*. URL: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia (visited on 12/12/2019).

[4] *convolution network basics - Charlotte77*. zh-cn. URL: https://www.cnblogs.com/charlotte77/p/7759802.html (visited on 12/16/2019).

[5] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition [Book]*. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ (visited on 12/12/2019).

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. en. Google-Books-ID: tVIjmNS3Ob8C. Springer Science & Business Media, Aug. 2009. ISBN: 978-0-387-84858-7.

[7] Antonino Ingargiola. *Deep-dive into Convolutional Networks*. en. Apr. 2019. URL: https://towardsdatascience.com/deep-dive-into-convolutional-networks-48db75969fdf (visited on 12/11/2019).

[8] *MachineLearning/DimRed-minted.pdf at master · CompPhysics/MachineLearning · GitHub*. URL: https://github.com/CompPhysics/MachineLearning/blob/master/doc/pub/DimRed/pdf/DimRed-minted.pdf (visited on 12/12/2019).

[9] *Pneumonia*. en. URL: https://www.who.int/news-room/fact-sheets/detail/pneumonia (visited on 11/29/2019).

[10] *Pneumonia Can Be Prevented—Vaccines Can Help — CDC*. URL: https://www.cdc.gov/pneumonia/prevention.html?CDC_AA_refVal=https%3A%2F%2Fwww.cdc.gov%2Ffeatures%2Fpneumonia%2Findex.html (visited on 12/12/2019).

[11] Sivaramakrishnan Rajaraman et al. "Visualization and Interpretation of Convolutional Neural Network Predictions in Detecting Pneumonia in Pediatric Chest Radiographs". In: *Applied Sciences* 8.10 (2018), p. 1715. DOI: 10.3390/app8101715.

[12] Haralick Robert M. *Textural Features for Image Classification - IEEE Journals & Magazine*. URL: https://ieeexplore.ieee.org/abstract/document/4309314 (visited on 12/12/2019).

[13] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. en. Dec. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 12/11/2019).

[14] Hector Sanz et al. "SVM-RFE: selection and visualization of the most relevant features through non-linear kernels". In: *BMC Bioinformatics* 19.1 (Nov. 19, 2018), p. 432. ISSN: 1471-2105. DOI: 10.1186/s12859-018-2451-4. URL: https://doi.org/10.1186/s12859-018-2451-4 (visited on 12/18/2019).

[15] Hadaate Ullah and Mohammad Bhuiyan. "Performance Evaluation of Feed Forward Neural Network for Image Classification". In: *Journal of Science and Technology* 10 (May 2018). DOI: 10.30880/jst.2018.10.01.004.

[16] Rohit Verma. *deadskull7/Pneumonia-Diagnosis-using-XRays-96-percent-Recall*. original-date: 2018-05-29T11:51:48Z. Dec. 2019. URL: https://github.com/deadskull7/Pneumonia-Diagnosis-using-XRays-96-percent-Recall (visited on 12/16/2019).

[17] *VGG16 - Convolutional Network for Classification and Detection*. en-US. Nov. 2018. URL: https://neurohive.io/en/popular-networks/vgg16/ (visited on 12/16/2019).

[18] Dan Wootton and Charles Feldman. "The diagnosis of pneumonia requires a chest radiograph (x-ray)—yes, no or sometimes?" en. In: *Pneumonia* 5.1 (Dec. 2014), pp. 1–7. ISSN: 2200-6133. DOI: 10.15172/pneu.2014.5/464. URL: https://pneumonia.biomedcentral.com/articles/10.15172/pneu.2014.5/464 (visited on 11/30/2019).