DM996 - Intelligent Sensing, Reasoning And Deep Learning

Autonomous Diagnosis System for Parkinson's Disease

Group C:

Matthew Anderson, Ionela-Ancuța Cîrjilă,

Richard Gilchrist, Chavinpat Naimee

Radi Radev, Mohammed Salih

University of Strathclyde

Department of Design, Manufacturing & Engineering Management

Abstract

This report discusses the methodology and results obtained from the development of an autonomous system used to diagnose Parkinson's disease using speech data obtained from the University of Strathclyde's Department of Speech Therapy. Machine learning and deep learning methods investigated included support vector machines (SVM) using linear and radial basis functions, as well as a convolutional neural network (CNN) using a transfer learning approach. The data for Parkinson's patients and healthy control subjects were analysed and pre-processed. The SVM method using the linear and radial basis functions achieved test set accuracy of 78.84% and 82.75%, respectively, while the transfer learning CNN method achieved an accuracy of 83.34%.

Contents

List of Figures

List of Tables

1 Introduction

Artificial intelligence has seen increasing influence within healthcare in recent years, due to the increasing availability of big data, computational power and advancements in analysis methods. It has the potential to ease the strain on resources, free up time for doctor-patient interactions and aid the development of tailored treatment. A key application for machine learning and deep learning methods within healthcare is disease/ailment diagnosis, where recent advancements have provided diagnosis accuracy on par with qualified clinicians.

To gain an understanding of how AI can be used within this context, this project investigated machine learning and deep learning methods for the diagnosis of Parkinson's disease using recorded speech. Support vector machines were investigated for the machine learning approach, due to the algorithm's automatic control generalisation and parameterisation as part of the overall optimisation process. The deep learning method involved implementing a convolutional neural network, with this method being selected due to the algorithm's capability to handle high-dimensional data and perform automatic feature extraction. A transfer learning approach was adopted, making use of the pre-trained CNN model, 'ResNet-18', in order to take advantage of the model's pre-trained filter weights for feature extraction and to permit faster training times.

An autonomous system providing decision support has the potential to improve the accuracy of diagnoses by aiding clinicians in their assessment. Early and accurate diagnosis of Parkinson's disease is crucial to improve patient outcome and current validity of Parkinson's clinical diagnosis has been determined unsatisfactory and further improvement is required (Jellinger, 2019). This paper proposes a model which provides good classification performance for the Parkinson's disease speech data used, and has similar performance to values observed within clinical environments.

2 Background

2.1 Parkinson's Disease

Parkinson's disease is a neurodegenerative disease which effects approximately 145,000 people in the UK (Parkinson's UK, 2017) and can cause serious health effects such as balance and memory problems, insomnia and depression. Parkinson's is caused by a loss of nerve cells in an area of the brain called the substantia nigra, which leads to a reduction in the level of dopamine; a chemical which has a crucial role in regulating the movement of the body.

An accurate diagnosis of Parkinson's is important for both prognostic and therapeutic purposes, and can be beneficial for research of the disease. Clinicopathological studies based on brain bank

material from the UK and Canada have discovered that clinicians diagnose the disease incorrectly in approximately 25% of patients (Tolosa et al., 2006). A key reason for diagnosis errors are due to the similarity of Parkinson's with other neurodegenerative diseases including Alzheimer's disease and dementia with Lewy bodies. There is currently no cure for Parkinson's, however treatments currently being studied involve fetal cell transplantation, the use of stem cells, and gene therapy. The non-motor symptoms of Parkinson's disease - such as pain, fatigue and low blood pressure - may be treated with dopaminergic drugs (Chaudhuri and Schapira, 2009).

2.2 Machine Learning in Healthcare

A huge amount of complex data should be processed and interpreted by healthcare epidemics specialists (Kaye et al., 2015). E-health data has grown due the expansion of the role of these specialists in healthcare has expanded, e-health data has also expanded (Bates et al., 2014). Many new opportunities have been created due to the availability of huge quantities of high-quality data. Effectively exploiting this data may lead to better understand of the risk factors for the development of diseases and improve patient outcomes by implementing targeted approaches for prevention.

Previously, much of the available clinical data was neglected, or not even collected at all. This was happening because of the size and complexity of the data, as well as a lack of available techniques for the collection and storage of such data. However, more modern and improved techniques such as E-health records help to overcome this problem. Particularly, machine learning (ML) has begun to largely scale in entering the clinical literature. The appropriate application of ML in healthcare epidemiology has promising returns on investing in data collection.

2.3 Deep Learning in Healthcare

Healthcare is a key industry which is predicted to benefit from advancements in deep learning. Through an increased availability of healthcare data and development of new deep learning methods, it is poised to bring about a paradigm shift in the industry. The technology can be used for early detection and diagnosis, as well as outcome prediction and prognosis evaluation - all of which can be used to improve patient outcome and quality of life. The influence of AI within healthcare within the healthcare market was valued at approximately $2.1B in 2018 and is projected to be worth $36.1B in 2025 (*Artificial Intelligence in Healthcare: Industry Report*, 2019).

Deep learning algorithms are extensions of neural networks which have many layers, allowing the network to handle extremely large data sets and non-linear data. As can be seen in figure 1, the most common deep learning algorithms referenced with the healthcare and disease category on PubMed are convolutional neural networks (CNN), recurrent neural networks (RNN), deep belief

networks and deep neural networks (Long et al., 2017).

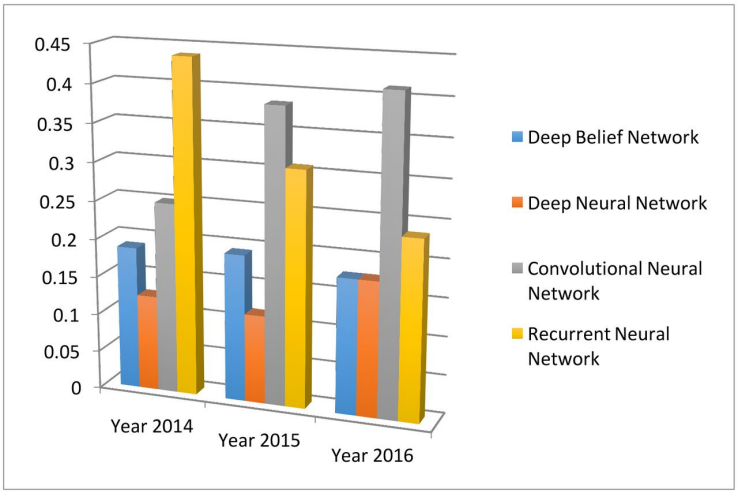


Figure 1: Four main deep learning algorithms and their popularity

CNNs have seen recent popularity due to their effectiveness in handling high-dimensional data. Classical machine learning approaches are typically better suited to data which has fewer dimensions and subsequent features, and therefore are not well-suited to tasks involving image data which are high-dimensional due to each image containing a large number of pixels. Traditionally, feature reduction could be used to help solve this issue when using ML methods, however this can result in the loss of information. CNNs solve this issue through automatic feature extraction from the images. Therefore, CNNs have been used frequently in the context of diagnosis from clinical images. A CNN was used within a hospital to diagnose congenital cataract disease from ocular images, and achieved a 90% accuracy on both diagnosis and treatment suggestion - which was determined to be similar in accuracy to that of a qualified physician (Long et al., 2017).

Khojasteh et al. (2018) conducted a study using recordings of sustained vowel sounds from 81 subjects - 41 of which were Parkinson's patients and the remaining 40 were healthy control subjects. They constructed and trained a number of CNNs that had varying numbers of convolutional layers and feature maps, and different topologies in the fully-connected layer. The best accuracy achieved was 75.7%, with a sensitivity of 0.66. Vaiciukynas et al. (2018) used speech consisting of a four-word sentence recited by 268 subjects aged between 22 and 85 years old - 74 of which were Parkinson's patients and 194 were healthy control subjects. By considering combinations of words or syllables, nine unique segments were constructed for each subject. A number of variants of audio features were considered, giving nine input feature maps for each of the audio segments. The CNN used consisted of four convolutional-pooling layer pairs, one fully-connected dense layer and two softmax neurons in the output layer. A decision-level fusion approach led to an equal error rate (EER) of 14.1%.

2.4 Support Vector Machines

A support vector machine (SVM) is a supervised machine learning algorithm commonly used in classification problems. The principal idea is to find the best separation hyperplane between a set of points, keeping as far as possible from the boundary the nearest to the hyperplane ones. These latest points, that are the closest to the boundary separating the classes, are called 'support vectors'.

Sometimes, because the data set is not clearly segregated, further measures should be taken, e.g. mapping the data into a higher dimension by using kernels. The most common are the linear kernel applied for linearly separated data (for logistic regression) and the radial basis function or RBF, used for non-linear problems. The latter is far more computationally expensive than the linear kernel, but provides the best accuracy for its case of use. In general, the SVM algorithm tends to perform well on small and clean data sets, although the training time may be high.

The decision to use support vector machines over other methods such as k-nearest neighbour random forests, was based on positive results having been reported in the field (Sakar et al., 2013), (Mcsharry and Hunter, 2009), (Braga and Ajith, 2019). Although other methods or adjustments could have had a better impact on the overall classification performance, SVM proved to be a good approach. Further details are provided in the next sections.

2.5 Spectrograms

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. The spectrogram graphs the energy content of a signal expressed as a function of frequency and time. It is used as a basic tool in audio spectral analysis as well as other fields including sonar and radar. The short-time Fourier transform (STFT) is used to create a spectrogram, which involves a sequence of fast Fourier transforms of windowed data segments applied to a signal. The windows used within the STFT are usually allowed to overlap in time, typically by 25-50% (Allen and Rabiner, 1977).

There are two key methods of voice analysis performed by the spectrogram, which includes wideband (bandwidth of 300-500 Hz) and narrowband (bandwidth of 45-50 Hz). Wideband spectrograms can observe energy from several harmonics at once and adding them together. The wide bandwidth allows for good time resolution, where energy peaks from each individual vibration of the vocal folds can be determined. However, wide bandwidth also results in poor frequency resolution where individual harmonics cannot be accurately identified. The narrowband spectrogram can pick out each individual harmonic, resulting in strong frequency resolution. However, the narrowband has poor time resolution therefore is not capable of isolating each

individual cycle of vibration.

2.6 Convolutional Neural Networks

A convolutional neural network is a particular type of deep neural network often applied to problems involving image data. The key feature of a CNN is that the network learns the weights for its convolutional filters during training, rather than relying on specially hand-crafted filters that are designed to identify specific features within an image; such as straight lines or shapes. An example of the structure of a CNN is illustrated in figure 2, where it can be seen that when describing a CNN, we consider three-dimensional *volumes* rather than flat, two-dimensional structures as we see in regular deep neural networks.

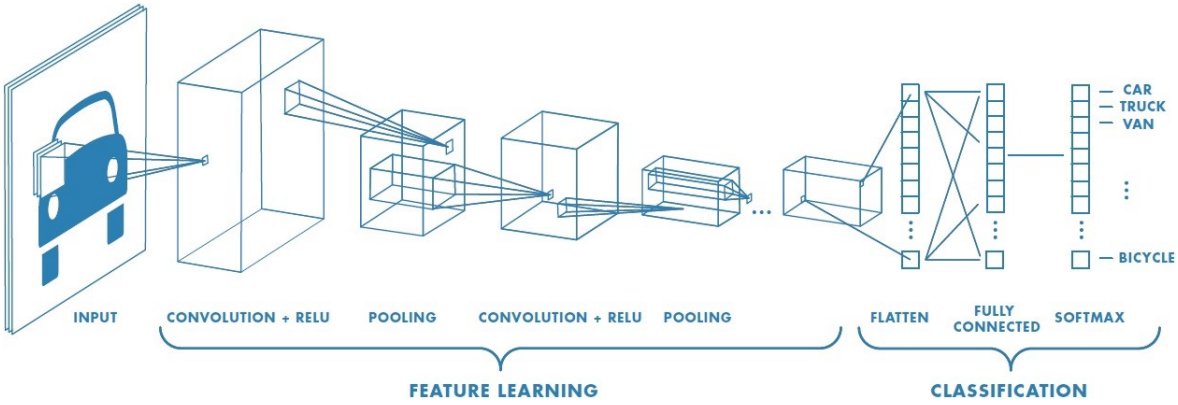


Figure 2: An example of the structure of a CNN (Mathworks, 2020)

A CNN has a number of different types of layers, including the input layer, a number of convolutional, pooling and rectified linear unit (ReLU) layers, as well as a fully-connected layer at the end to output the network's class prediction. The convolutional and fully-connected layers have trainable weights associated with them, while the ReLU and pooling layers do not.

A convolutional layer may be visualised as a three-dimensional volume, consisting of 'slices' of nodes, where each depth slice represents a single filter within that convolutional layer. Each node in a depth slice has a set of weight values and is connected to a small region of the input, referred to as the receptive field of that node. The filters in the first convolutional layer may be used to identify simple features such as straight lines, with subsequent convolutional layers being used to identify more complicated features composed of combinations of lower-level features. For the convolutional filters, the aim is to learn sets of weights that extract features from the input which prove useful for correctly classifying the input data, thus minimising the loss for the specific task under consideration.

CNN models which have achieved state-of-the-art performance on image classification tasks typically have a very large number of trainable parameters and are trained using very large data sets in a computationally expensive process. An example of such a model is a CNN model known as 'AlexNet'. The training data used by AlexNet, taken from the ImageNet data set, consisted of over one million images belonging to one thousand classes. The AlexNet network, consisting of 650,000 neurons and having 60 million parameters, surpassed the state-of-the-art performance on the test data (Krizhevsky et al., 2012). One approach to take advantage of these models without the data and computational requirements associated with training is known as *transfer learning*.

Transfer learning involves using a model that has been trained for a particular task, then modifying the trained model for application to another set of data. A pre-trained CNN model includes all the convolutional filter weights that were learned during the training of the original network. When applying the pre-trained model to a new data set, these filter weights are 'frozen', while fully-connected layers may be trained as the data are presented to the network, allowing the network to generate predictions on the new data set.

3 Methodology

3.1 Data Pre-Processing

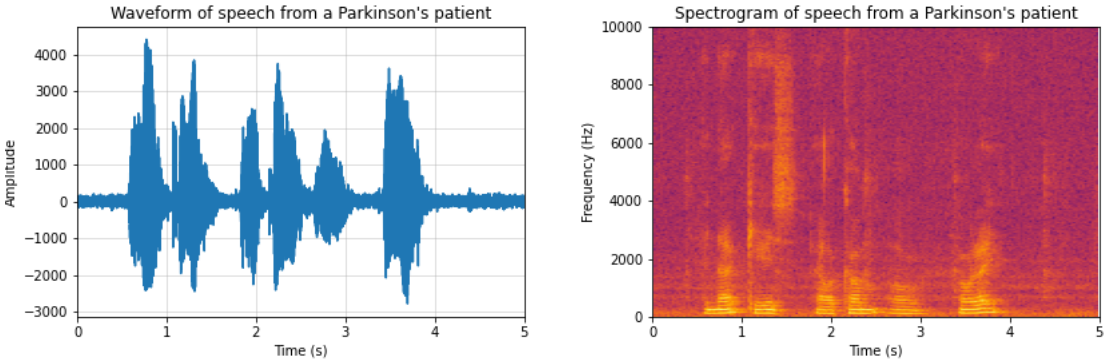
The data set used consisted of a collection of audio recordings featuring 30 patients with Parkinson's disease and 20 patients without Parkinson's, to be used as a control. The recordings featured patients reciting passages as well as vocalising vowel sounds with constant pitch, varying pitch or varying volume. Conversion to WAV format was automated using the scripting language in the open source software Praat.

The duration of the audio files varied widely, since some files contained only sustained vowels, while others contained recitations of a passage of text. Due to the fully-connected layers having a fixed number of nodes, the size of the input to the CNN must be consistent between instances. As a result, it was decided that the audio files should be cut into fixed-length segments such that the resulting spectrograms would be of equal size. Additionally, this ensured any distortions resulting from transformations applied to the spectrograms, such as re-sizing, were consistent between instances.

As an alternative method, a spectrogram of the full audio clip could have been generated before using a sliding window of fixed size in order to sample the data. Other approaches, such as those adopted by (Vaiciukynas et al., 2018), include using a fixed-size window to randomly sample from the spectrogram, or re-sizing the spectrogram image to a fixed width corresponding to the

minimum or mean width of all the images.

A Matlab script was written to split each audio file into five-second long clips. This resulted in the final clip generated from each audio file having a duration of less than five seconds. These final clips were discarded with the justification that the last few seconds of a recording were likely to contain silence rather than useful information. A spectrogram was generated for each of the five-second clips in the data set; an example of which is shown in figure 3.



(a) Waveform representation

(b) Spectrogram representation

Figure 3: Five seconds of speech from a Parkinson's patient

After the spectrograms were created they were resized to size 224, normalised with means (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225) for the 1st, 2nd and 3rd channels of the images respectively.This was done to create suitable inputs for the ResNet model.

3.2 Support Vector Machine

3.2.1 Feature Extraction

The vocal features considered in our first trial were extracted using Praat and are detailed in appendix A. These 26 vocal features were selected because they have been used in many published scientific papers which report quite high accuracy for using these features for diagnosing Parkinson's disease using SVM (**?**).

Many challenges were presented with the data we obtained from the University. Firstly, the recording was not standardised, with some of the recordings having been done in the homes of the patients. Also, many of the recordings for people with PD included the voices of people who are instructing them to say the sentences. Those additional voices are from people who are non-PD, so if we try to automate the sampling, we will include these voices with the people with PD, which could lead to confusion for our network. Secondly, due to a lack of familiarity with Praat's

scripting language, an attempt was made to extract the vocal features manually in Praat. The task was divided between three of us in order to cover a wide range of speakers and sentences and expressions like “da-da-da-da” for both PD and non-PD recordings. As a result, a matrix of size (231-by-28) was obtained.

As the results of the previously described approach were not satisfactory, alternative methods of feature extraction were investigated. This time a Matlab package was used, having been modified to fit our specific problem (Vasileiou, 2016). The process for extracting the features, which later will be passed to the SVM algorithm, consisted of the following stages.

The first stage consisted of pre-processing the audio samples, performing operations including re-sampling (changing the sample rate for space reduction), filtering (for noise reduction) and framing (division of speech). At the second stage, the Matlab package extracts 26 features of two types:

- Prosodic – that carry emotional content of the speech signal; they are derived from pitch, energy and formants frequency, and comprise: tempo, accentuation, volume, stress, duration.
- Spectral – that come from the vocal tract system: the Mel-frequency cepstral coefficients.

For the extraction of the pitch feature, which is generated by the vibration of the vocal cords, the autocorrelation method was used. For the four formants, which describe the vocal tract resonance and are responsible for the formation of the overall spectrum, the linear prediction method was used. The short-time energy is associated with the arousal level of emotions and equal to the sum of the squared absolute values of signal’s amplitudes. Regarding the extraction of the twelve Mel-frequency Cepstral coefficients, fast Fourier transform and discrete cosine transform were used.

Each sample from the data set had an associated set of frames, and for each frame a set of 26 features was computed. The third stage consisted of applying statistics on each of these 26-feature sets in order to obtain the mean, median, standard deviation, maximum, minimum and range. At the end, a single instance consisted of 6 x 26 = 156 features. The final matrix that represents the data set to be processed by SVM, contained 1,742 samples for speech affected by Parkinson’s and 1,863 samples for speech with no Parkinson’s characteristics.

3.2.2 Matlab Implementation for SVM

Many researchers use SVM to distinguish between people with Parkinson’s disease and people without - where SVM can provide good classification results (Sakar et al., 2013). In this study, SVM with linear and radial basis function kernels will be used to generate the model and then evaluate and compare the results of both kernels. Both kernels would be used because there are

many features and few samples in our dataset, and we still do not know that our dataset is linear or nonlinear. Consequently, the acceptable outcomes from SVM method can be compared to the CNN method by using the same data.

After getting the dataset from feature extraction, Matlab software would be used to train the SVM model. Firstly, the dataset would be normalised and split into 80% of the training set and 20% of the testing set. Then using SVM Matlab function "fitcsvm" to train the model and the linear kernel and RBF kernel would be used to learn data. Finally, the model performance could be measured by using the confusion matrix to compute accuracy, recall, specificity, and MCC. The Matlab code of our model is presented in appendix B.

3.3 Convolutional Neural Network

We also implemented an alternative approach of classification using a CNN. We decided to use a transfer learning approach, using a model pretrained on the ImageNet dataset. The ImageNet consists of 14 million images belonging to 1000 classes. Since the ImageNet dataset consists of images mostly of everyday objects, we rely on the assumption that the convolutional layers were able to extract features which could also be useful for classification of speech data in the form of spectrograms. This method has been successfully employed in literature for audio classification (Boddapati et al., 2017).

Another advantage of using the transfer learning approach is that it reduces computation times, which is especially important for for deep learning tasks. Our dataset of over 3000 images, while small in comparison to datasets such as ImageNet, was still large enough that taking the speed of training mattered. The transfer learning method was implemented in Python using the PyTorch library (Paszke et al., 2019). PyTorch is a deep learning high-level API which comes with various pre-trained models and allows for implementing them with ease.

The deep learning model implemented was ResNet-18 (He et al., 2015*a*). The ResNet model is a deep residual neural network. The model makes use of residual blocks, through the implementation of shortcut connections, where layers in the network are skipped and the output from one layer is fed directly to the input of consecutive layer. The residual blocks were introduced to help address the problem of vanishing/exploding gradients.

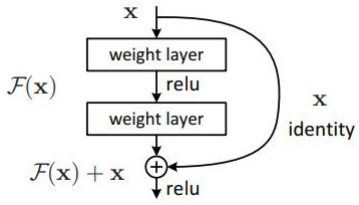


Figure 4: Residual learning: a building block

(He et al., 2015*a*)

3.4 Evaluation Metrics

A confusion matrix or error matrix is the method to measure the performance of classification model on a testing data that the actual values are known. Accuracy, recall (sensitivity), specificity, and Matthews correlation coefficient (MCC), can be computed from the confusion matrix.
The elements of the confusion matrix are summarised in table 1.

	Predict: Non-Parkinson's	Predict: Parkinson's
Actual: Non-Parkinson's	TN	FP
Actual: Parkinson's	FN	TP

Table 1: The confusion matrix

Where TN = True negative, FP = False positive, FN = False negative and TP = True positive.

Accuracy is the ratio of the number of correct predictions to the total number of predictions made, as per equation 1.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

Recall or sensitivity is the ratio of correctly classified positive instances to the total number of positive instances, calculated as per equation 2.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

On the other hand, specificity is the statistical measure of correctly classified negative instances, computed as per equation 3.

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{3}$$

MCC is a measure of the performance of binary classification in machine learning. Even if the class densities are noticeably distinct, MCC is still stable. It is a correlation coefficient between the

prediction and observation of binary classifications that may have a value between -1 and 1. A value of 1 means the model makes perfect predictions, while a value of -1 means the prediction and actual values are exactly opposite. The MCC metric is computed as per equation 4.

$$\text{MCC} = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4}$$

4 Results

4.1 Support Vector Machine

After the data extraction step and the normalisation method as a pre-processing step, columns with a zero in every row are set as a zero value - such that every feature has a zero standard deviation and zero means. The dataset is fed into the SVM technique using a linear and radial basis function (RBF) kernels to diagnose the Parkinson's disease.

The sample of the confusion matrix of the highest accuracy by RBF kernel demonstrates in table 2. It shows that the classifier made a total of 690 predictions from the test data, which it predicted Parkinson's 389 times, and non-Parkinson 301 times. However in reality, 352 subjects had the disease, and 338 subjects did not have the disease. In a clinical setting it is clearly desirable to minimise the number of false negative classifications made by the model. That is, the sensitivity should be maximised (Lalkhen and McCluskey, 2008).

	Predicted: Non-Parkinson's	Predicted: Parkinson's
Actual: Non-Parkinson's	260	78
Actual: Parkinson's	41	311

Table 2: The confusion matrix of the best performance of the SVM model

Table 3 illustrates the performance of SVM models in both linear kernel and RBF kernel among the five runs of creating random training set and test set. From the results, it can be seen that the RBF kernel performed better than linear kernel for this dataset. The SVM classifier using RBF kernel produced the highest accuracy as 82.75%, while the highest accuracy of the linear kernel is 78.84%. This SVM-RBF model also provided the highest MCC and recall, which are 0.658 and 88.35%, respectively. As a result of the SVM analysis, it seems that this algorithm performed an acceptable performance for our dataset.

Number	Kernel Type	Accuracy (%)	MCC	Recall (%)	Specificity (%)
1	Linear Kernel	78.84	0.578	82.95	74.56
	RBF Kernel	82.75	0.658	88.35	76.92
2	Linear Kernel	74.64	0.499	69.42	80.43
	RBF Kernel	78.70	0.575	76.86	80.73
3	Linear Kernel	76.38	0.540	69.29	84.47
	RBF Kernel	80.29	0.611	76.36	84.78
4	Linear Kernel	74.35	0.487	73.45	75.30
	RBF Kernel	77.68	0.553	79.10	76.19
5	Linear Kernel	78.55	0.571	80.40	76.68
	RBF Kernel	80.72	0.616	83.86	77.55

Table 3: The results of the SVM models with linear kernel and RBF kernel

4.2 Convolutional Neural Network

It is noted that preliminary results were presented of the model having an accuracy of over 90%. However, after completing some sanity checks, we realised that the data set was incorrectly split. Namely, we had the data from the same patients in the testing and training sets (albeit it being from different parts of the speech files). It is possible that as such the model was being able to identify the individual patients, rather than detecting the disease. To account for this, it was made sure that none of the patients were present in both sets, which brought down the accuracy of the model.

Nevertheless, the model outperforms the SMV model. In Figure 5, the loss and accuracy of the model during the training process are presented. The model was trained for 20 epochs and it had the best convergence in epoch 17. The results using the various evaluation metrics discussed earlier are presented in table 5. The high recall is important for such an application since patients that have the disease are a lot less likely to not be detected.

The code used to implement and train the ResNet model is presented in appendix B. The model used stochastic gradient descent as the optimisation algorithm and cross-entropy loss (which is particularly suitable for classification algorithms). The learning rate was 0.001 with a momentum parameter of 0.5. There was also a learning decay factor of 0.1 every 7 epochs.

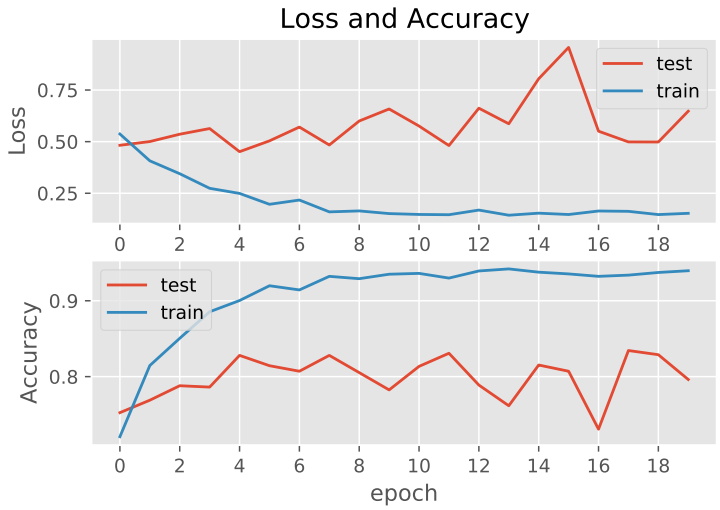


Figure 5: Loss (top) and accuracy (bottom) of the network on the testing and training sets

	Predicted: Non-Parkinson's	Predicted: Parkinson's
Actual: Non-Parkinson's	410	148
Actual: Parkinson's	34	507

Table 4: Confusion matrix of the CNN model

Model	Accuracy (%)	MCC	Recall (%)	Specificity (%)
ResNet-18	83.34	0.68	93.72	73.48

Table 5: Result of the CNN Model

5 Discussion

Future work which could be done to improve the methods discussed could include the following:

- Implement hypothesis generation which can help build better features later on, which are not biased by the data available in the data-set. This is a critical step which usually improves a model's accuracy.
- Undertake feature engineering which is highly influenced by hypothesis generation, as good hypothesis results in good features.

- Crop out the voices of healthy people from the People with PD before we automate the sampling to avoid network confusion.
- An investigation into the p-values, information values and other statistical metrics could be undertaken to select the features which provides the highest accuracy. Similarly to PCA used within the SVM approach where 156 features were obtained which helped to represent training data into a lower dimension, but still characterise the inherent relationships in the data.

6 Conclusion

Recent advancements within AI have allowed for vast new applications within the domain of healthcare. ML/DL algorithms have the potential to emulate human cognition for the analysis of complex clinical data and make justified decisions. As the world population continues to grow such processes will see stronger relevance to relieve the strain on resources within clinical environments.

This paper has presented both machine learning and deep learning approaches for the diagnosis of Parkinson's disease from speech data. Using support vector machines with a radial basis function, an accuracy of 82.75% was observed, whereas using a pre-trained CNN ResNet-18 model, an accuracy of 83.34% was observed. Both accuracies were higher than the pooled diagnostic accuracy from the UK Parkinson's Disease Society Brain Bank Research Centre criteria, with an accuracy of 82.7% (Jellinger, 2019).

To validate the accuracies observed, future work would involve testing on a different data set to determine that the high accuracy was not due to potential irregularities within the data set used, as well as to evaluate the generalisation of the models used. In order to further develop this work, a number of additional approaches may be taken. Further pre-processing could be done in order to remove background noise, normalise the levels of the recordings and ensure that only the patient's voice was present in the speech segments used. Different methods of sampling the data from the spectrograms could be investigated, as discussed in section 3.1. Additionally, methods such as spatial pyramid pooling (He et al., 2015*b*) could be investigated in order to remove the requirement for a fixed-size input to the network.

A Vocal Features

Pitch:

- Median pitch
- Mean pitch
- Standard deviation
- Minimum pitch
- Maximum pitch

Pulses:

- Number of pulses
- Number of periods
- Mean period
- Standard deviation of period

Voicing:

- Fraction of locally unvoiced frames
- Number of voice breaks
- Degree of voice breaks

Jitter: which is the irregularities in the duration

- Jitter (local)
- Jitter (local, absolute)
- Jitter (rap)
- Jitter (ppq5)
- Jitter (ddp)

Shimmer: which is the amplitude of individual cycles

- Shimmer (local)
- Shimmer (local, dB)
- Shimmer (apq3)
- Shimmer (apq5)
- Shimmer (apq11)
- Shimmer (dda)
- Harmonicity of the voiced parts only
- Mean autocorrelation
- Mean noise-to-harmonics ratio
- Mean harmonics-to-noise ratio

B Code

Praat script to convert all files from NSP format to wav format (Yoon, 2008).

```
# Praat script
# Converts NSP file format to wav file format
# Usage:
#       Provide a directory in the form of:
#       Directory D:\tests\nsp\
# Output:
#       NSP file will be saved as wav file
#
# Script written by:
# Tae-Jin Yoon
# tyoon@uvic.ca
# November 24, 2008
# Modified by Richard Gilchrist 16/02/2020

form Configuration for NSP2WAV
    comment NSP files are in the following directory
    sentence Dir C:\Users\Dick Gilchrist\Dropbox\MSc MLDL\DM996\DM996 Group ...
        Project\PD_Master_test\MASTER\
endform

Create Strings as file list... nsplist 'dir$'*.NSP
num_files = Get number of strings

for i from 1 to num_files
      select Strings nsplist
      fname$ = Get string... 'i'
      Read from file... 'dir$''fname$'
      basename$ = fname$-".NSP"
      wavname$ = basename$+".wav"

      Write to WAV file... 'dir$''wavname$'
      select Sound 'basename$'
      Remove

printline No. 'i': 'fname$' is saved as 'wavname$'

endfor
printline 'num_files' wave files are saved.
select Strings nsplist
Remove
```

Python code to extract all audio files from sub-folders and copy them all into one master folder.

```
# A script to extract files from subfolders and
# place them all in a 'MASTER' folder.

import os
import shutil

# Specify the parent folder
src = r'C:\Users\Dick Gilchrist\Dropbox\MSc MLDL\DM996\DM996 Group Project\...
                                            PD_Master_test'

# Specify the destination folder (MASTER folder)
dest = r'C:\Users\Dick Gilchrist\Dropbox\MSc MLDL\DM996\DM996 Group Project\...
                                            PD_Master_test\MASTER'

# For each file, create the full filepath for the filename
# Copy the filename to the destination folder
for path, subdirs, files in os.walk(src):
    for name in files:
        filename = os.path.join(path, name)
        shutil.copy2(filename, dest)
```

Matlab function to split an audio file into x-second long chunks.

```
function clip_array = split_audio(filename, x)
    % Function to chop an audio file into x-second long chunks
    % Inputs:
    % filename - The filename of the audio file to be split
    % x - Number of seconds each chunk of audio should be
    % Requires the output path be defined (below)

    out_path = "C:\Users\Dick Gilchrist\Dropbox\MSc MLDL\DM996\DM996 Group ...
        Project\Audio_clip_test\five_seconds\";

    % Load the samples and sample rate from the file
    [sig, fs] = audioread(filename);

    % Store the number of samples in a variable
    n = length(sig);

    % Make a cell array to store the clips.
    % Initialised to some size dependent on n and fs.
    clip_array = cell(floor(n/(x*fs)) +1, 1);

    % Initialise index for populating cell array
    i = 1;

    % Loop through the samples in intervals of x*fs
    for k = 1:x*fs:n
        % Populate the clip array with a chunk of fs samples until end
        clip_array{i} = sig(k:min(k + x*fs - 1, n), :);
        % Increment cell array index
        i = i + 1;
    end

    % Remove the last entry in the clip_array if it has fewer samples.
    % We need all chunks of audio to be the same size and the end will likely
    % just contain silence, anyway.
    if length(clip_array{end,1}) ~= x*fs
        clip_array(end) = [];
    end

    % Erase the '.wav' part of the filename
    filename = erase(filename, '.wav');

    % Loop through the clip_array and write the wav files
    for k = 1:length(clip_array)
        % Create the new file name
```

```
44          outname = sprintf("%s%s_%d.wav", out_path, filename, k);
45          audiowrite(outname, clip_array{k, 1}, fs);
46      end
47  end
```

Matlab script to call the `split_audio` Matlab function on each wav file in a specified folder.

```
1  % Script to take each wav file in a directory and pass it to the
2  % split_audio function to generate the shorter clips.
3
4  % Define the location of the wav files
5  in_path = "D:\Dropbox\MSc MLDL\DM996\DM996 Group Project\PD_Master_test\MASTER\";
6  % Create Matlab struct containing wav file information
7  files = dir(strcat(in_path,'*.wav'));
8
9  % Specify the number of seconds you want each file to be
10 x = 5;
11
12 % For each file in the 'in_path' directory, call the split_audio() func
13 for i = 1:length(files)
14     split_audio(files(i).name, x);
15 end
```

Python code to create the spectrograms and split the data into training and testing sets.

```
1
2 #By Richard Gilchrist and Radi Radev
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 from google.colab import drive
8 drive.mount('/content/gdrive/', force_remount=True)
9
10 !fusermount -u gdrive
11 !google-drive-ocamlfuse gdrive
12
13 #location to 5sec wav files
14
15 #Spectrogram
16
17
18 import numpy as np
19 from matplotlib import pyplot as plt
20 import scipy.io.wavfile as wav
21 from numpy.lib import stride_tricks
22
23 """ short time fourier transform of audio signal """
24 def stft(sig, frameSize, overlapFac=0.5, window=np.hanning):
25     win = window(frameSize)
26     hopSize = int(frameSize - np.floor(overlapFac * frameSize))
27
28     # zeros at beginning (thus center of 1st window should be for sample nr. 0)
29     samples = np.append(np.zeros(int(np.floor(frameSize/2.0))), sig)
30     # cols for windowing
31     cols = np.ceil( (len(samples) - frameSize) / float(hopSize)) + 1
32     # zeros at end (thus samples can be fully covered by frames)
33     samples = np.append(samples, np.zeros(frameSize))
34
35     frames = stride_tricks.as_strided(samples, shape=(int(cols), frameSize), ...
           strides=(samples.strides[0]*hopSize, samples.strides[0])).copy()
36     frames *= win
37
38     return np.fft.rfft(frames)
39
40 """ scale frequency axis logarithmically """
41 def logscale_spec(spec, sr=44100, factor=20.):
42     timebins, freqbins = np.shape(spec)
43
```

```
44      scale = np.linspace(0, 1, freqbins) ** factor
45      scale *= (freqbins-1)/max(scale)
46      scale = np.unique(np.round(scale))
47
48      # create spectrogram with new freq bins
49      newspec = np.complex128(np.zeros([timebins, len(scale)]))
50      for i in range(0, len(scale)):
51          if i == len(scale)-1:
52              newspec[:,i] = np.sum(spec[:,int(scale[i]):], axis=1)
53          else:
54              newspec[:,i] = np.sum(spec[:,int(scale[i]):int(scale[i+1])], axis=1)
55
56      # list center freq of bins
57      allfreqs = np.abs(np.fft.fftfreq(freqbins*2, 1./sr)[:freqbins+1])
58      freqs = []
59      for i in range(0, len(scale)):
60          if i == len(scale)-1:
61              freqs += [np.mean(allfreqs[int(scale[i]):])]
62          else:
63              freqs += [np.mean(allfreqs[int(scale[i]):int(scale[i+1])])]
64
65      return newspec, freqs
66
67  """ plot spectrogram"""
68  def plotstft(audiopath, binsize=2**10, plotpath=None, colormap="jet", to_plot=...
        False):
69      samplerate, samples = wav.read(audiopath)
70
71      s = stft(samples, binsize)
72
73      sshow, freq = logscale_spec(s, factor=1.0, sr=samplerate)
74
75      ims = 20.*np.log10(np.abs(sshow)/10e-6) # amplitude to decibel
76
77      if to_plot == True:
78
79        timebins, freqbins = np.shape(ims)
80
81        print("timebins: ", timebins)
82        print("freqbins: ", freqbins)
83
84        plt.figure(figsize=(15, 7.5))
85        plt.imshow(np.transpose(ims), origin="lower", aspect="auto", cmap=colormap,...
              interpolation="none")
86        #plt.colorbar()
87
88        plt.xlabel("time (s)")
```

```
89        plt.ylabel("frequency (hz)")
90        plt.xlim([0, timebins-1])
91        plt.ylim([0, freqbins])
92
93        xlocs = np.float32(np.linspace(0, timebins-1, 5))
94        plt.xticks(xlocs, ["%.02f" % l for l in ((xlocs*len(samples)/timebins)...
              +(0.5*binsize))/samplerate])
95        ylocs = np.int16(np.round(np.linspace(0, freqbins-1, 10)))
96        plt.yticks(ylocs, ["%.02f" % freq[i] for i in ylocs])
97        plt.axis("off")
98        if plotpath:
99            plt.savefig(plotpath, bbox_inches="tight")
100       else:
101           plt.show()
102
103       plt.clf()
104
105     return ims
106
107 # Generate a spectrogram
108 ims = plotstft('/content/gdrive/My Drive/DM996/CO_5_SEC_WAV/NO17_sen_14.wav', ...
        to_plot=True, plotpath="/content/gdrive/My Drive/DM996/Spectrogram Data/train...
        /Healthy/test.png")
109
110
111 import glob
112 count=0
113 list=glob.glob('/content/gdrive/My Drive/DM996/PD_5_SEC_WAV/*.wav')
114
115 print (len(list))
116
117
118 for filepath in list:
119   if count/len(list)<=0.7:
120     ims = plotstft(filepath,
121                    to_plot=True, plotpath="/content/gdrive/My Drive/DM996/NewSPC/...
                           train/Parkinsons/train%s.png" %count)
122   else:
123     ims = plotstft(filepath,
124                    to_plot=True, plotpath="/content/gdrive/My Drive/DM996/NewSPC/...
                           val/Parkinsons/val%s.png" %count)
125   count+=1
```

ResNet-18 Implementation

```
1
2 #ResNet with SpectrogramData
3
4 #Automatically generated by Colaboratory.
5
6 # License: BSD
7 # Author: Sasank Chilamkurthy
8 # Adopted by: Radi Radev
9
10 import torch
11 import torch.nn as nn
12 import torch.optim as optim
13 from torch.optim import lr_scheduler
14 import numpy as np
15 import torchvision
16 from torchvision import datasets, models, transforms
17 import matplotlib.pyplot as plt
18 import time
19 import os
20 import copy
21
22 plt.ion()   # interactive mode
23
24 from google.colab import drive
25 drive.mount('/content/gdrive/')
26
27 import glob
28 len(glob.glob("/content/gdrive/My Drive/DM996/NewSPC/val/Parkinsons/*"))
29
30 !fusermount -u drive
31 !google-drive-ocamlfuse drive
32
33
34
35 # Data augmentation and normalization for training
36 # Just normalization for validation
37 data_transforms = {
38     'train': transforms.Compose([
39         transforms.Resize(224),
40         transforms.RandomHorizontalFlip(),
41         transforms.ToTensor(),
42         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
43     ]),
44     'val': transforms.Compose([
```

```
        transforms.Resize(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = '/content/gdrive/My Drive/DM996/NewSPC'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                              shuffle=True, num_workers=4)
               for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)


def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
```

```
92              for inputs, labels in dataloaders[phase]:
93                  inputs = inputs.to(device)
94                  labels = labels.to(device)
95
96                  # zero the parameter gradients
97                  optimizer.zero_grad()
98
99                  # forward
100                 # track history if only in train
101                 with torch.set_grad_enabled(phase == 'train'):
102                     outputs = model(inputs)
103                     _, preds = torch.max(outputs, 1)
104                     loss = criterion(outputs, labels)
105
106                     # backward + optimize only if in training phase
107                     if phase == 'train':
108                         loss.backward()
109                         optimizer.step()
110
111                 # statistics
112                 running_loss += loss.item() * inputs.size(0)
113                 running_corrects += torch.sum(preds == labels.data)
114             if phase == 'train':
115                 scheduler.step()
116
117             epoch_loss = running_loss / dataset_sizes[phase]
118             epoch_acc = running_corrects.double() / dataset_sizes[phase]
119
120             print('{} Loss: {:.4f} Acc: {:.4f}'.format(
121                 phase, epoch_loss, epoch_acc))
122
123             # deep copy the model
124             if phase == 'val' and epoch_acc > best_acc:
125                 best_acc = epoch_acc
126                 best_model_wts = copy.deepcopy(model.state_dict())
127
128         print()
129
130     time_elapsed = time.time() - since
131     print('Training complete in {:.0f}m {:.0f}s'.format(
132         time_elapsed // 60, time_elapsed % 60))
133     print('Best val Acc: {:4f}'.format(best_acc))
134
135     # load best model weights
136     model.load_state_dict(best_model_wts)
137     return model
138
```

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.5)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=20)

from sklearn.metrics import confusion_matrix

nb_classes = 2

# Initialize the prediction and label lists(tensors)
predlist=torch.zeros(0,dtype=torch.long, device='cpu')
lbllist=torch.zeros(0,dtype=torch.long, device='cpu')

with torch.no_grad():
    for i, (inputs, classes) in enumerate(dataloaders['val']):
        inputs = inputs.to(device)
        classes = classes.to(device)
        outputs = model_ft(inputs)
        _, preds = torch.max(outputs, 1)

        # Append batch prediction results
        predlist=torch.cat([predlist,preds.view(-1).cpu()])
        lbllist=torch.cat([lbllist,classes.view(-1).cpu()])

# Confusion matrix
conf_mat=confusion_matrix(lbllist.numpy(), predlist.numpy())
print(conf_mat)


# Per-class accuracy
class_accuracy=100*conf_mat.diagonal()/conf_mat.sum(1)
print(class_accuracy)
```

Feature extraction for SVM algorithm

```
%-------------------------------------------------------------------------
%   Name:     Extractfeatures(xnew,fsnew),EnergyoofFrame(xframe),
%             PitchAutoCorre(xnew1, fsnew,wh),FormantsLpc(xnew1,fsnew),
%             MFCC(xnew1,fsnew);
%   Author:   Kleitia Vasileiou
%   Adjusted for this project: the team / DM996's Group C of 2019-2020
%
%   Description: Extracting the speech features (pitch, formants,
%                energy, MFCC).
%-------------------------------------------------------------------------

function [FundamentalFreq,Formants,Energy,MelFreq,index] = Extractfeatures( xnew,...
    fsnew )

%--------------------Framing wideband for pitch detection-----------------

wlen=256;   % with short-time spectrum calculations
            % (about 30ms) and a power of 2 samples needed.
wover=226;  % 226 samples overlap.
shift=wlen-wover;

N=floor(((length(xnew)-wlen)/shift)+1); % Number of frames.
xframe=zeros(N,1);

for i=1:1:N  % frame by frame.

xframe=xnew(1+shift*(i-1):wlen+shift*(i-1)); % samples of the corresponding frame...
    .

    [r]=EnergyofFrame(xframe); % EnergyoofFrame fuction.
    Energy(i)=r; % Extract the energy

    %---------------------------------Windowing-------------------------------
    wlen=length(xframe);
    wh=hamming(wlen);    % 256x1
    xframe1=xframe.*wh; % windowed signal xw

    %-----------------------------Pre-emphasis filter-------------------------
    a=0.97; % filter coefficients
    xnew1=filter([1 -a],1,xframe1); % filter applied

    %-------------------------Features extraction-----------------------------

    [rp,f0] = PitchAutoCorr(xnew1, fsnew,wh); % PitchAutoCorre function.
```

```
43      FundamentalFreq(i)=f0; % Extract the pitch frequency.
44
45      [rf,formants] = FormantsLpc(xnew1, fsnew); % FormantsLpc function.
46      Formants{i}=formants; % Extract the formants frequency.
47
48      [MelCoef]=MFCC(xnew1,fsnew); % MFCC function.
49      MelFreq{i}=MelCoef; % Extract the MFCC.
50
51  end
52
53  %------------------------------ Thresholding ---------------------------------
54  thredb=-20;         % set the threshold to -20dB.
55  thre=10^(thredb/20); % convert -20dB into units.
56  index=zeros(1,N);
57
58  for i=1:N
59  if (Energy(i)>=thre) % Extracting the samples in Energy which are below
60                   % the threshold value.
61  index(i)=1;
62  end
63  end
64
65  end
```

SVM Matlab Code

```
clear all
clear

%function parkinson_svm(dir_name)

dir_name = "C:\Users\Angom\Documents\Strathclyde\Autonomous Deep Learning — DM996...
    \sem2\project\Matlab model\";

data = readmatrix(dir_name+"dataset_156features.xlsx");
%data = table2array(datatable);

data = data(:,2:size(data,2));

%% random order of the dataset
dataset = data(randperm(size(data, 1)), :);

inputColumns = 1:size(dataset,2)-1;
outputColumn = size(dataset,2);
numberOfSamples = size(dataset,1);

%Prepare training set and test set
selection = randperm( numberOfSamples, floor(numberOfSamples*80/100));
total = 1:numberOfSamples;
complement = total(:, ~ismember(total,selection));

trainingSet = dataset(selection, :);
testSet = dataset(complement, :);

%% Data Preprocessing

% Normalization using mean method
referenceNorm = zeros(size(inputColumns,2), 1);
for i=inputColumns
    referenceNorm(i,1) = norm(trainingSet(:,i));
    trainingSet(:, i) = trainingSet(:,i) / referenceNorm(i,:);
    testSet(:, i) = testSet(:,i) / referenceNorm(i,:);
end

% some feature values are 0 in every samples will be set as 0
trainingSet(isnan(trainingSet))=0;
testSet(isnan(testSet))=0;

% Principal component analysis
%numberOfEigenvectors = 11;
```

```
%[u,d,v] = svd( trainingSet(:, inputColumns)' * trainingSet(:, inputColumns));
%trainingSet = [trainingSet(:, inputColumns) * u(:, 1:numberOfEigenvectors), ...
    trainingSet(:, outputColumn1), trainingSet(:, outputColumn2)];
%testSet = [testSet(:, inputColumns) * u(:, 1:numberOfEigenvectors), testSet(:, ...
    outputColumn1), testSet(:, outputColumn2)];
%inputColumns = 1:numberOfEigenvectors;
%outputColumn1 = numberOfEigenvectors+1;
%outputColumn2 = numberOfEigenvectors + 2;

X_train = trainingSet(:,inputColumns);
Y_train = trainingSet(:,outputColumn);

X_test = testSet(:,inputColumns);
Y_test = testSet(:,outputColumn);

%% Train the SVM Classifier

SVMModel_rbf = fitcsvm(X_train,Y_train,'KernelFunction','RBF');
SVMModel_linear = fitcsvm(X_train,Y_train,'KernelFunction','linear');
%SVMModel_linear = fitcsvm(X_train,Y_train);
%SVMModel_linear = fitclinear(X_train,Y_train);
%% Perform cross-validation of rbf svm method
partitionedModel1 = crossval(SVMModel_rbf, 'KFold', 10);
% Compute validation predictions
[validationPredictions1, validationScores1] = kfoldPredict(partitionedModel1);
% Compute validation accuracy
validation_error1 = kfoldLoss(partitionedModel1, 'LossFun', 'ClassifError'); % ...
    validation error
validationAccuracy_rbf = 1 - validation_error1

%% Perform cross-validation of linear svm method
partitionedModel2 = crossval(SVMModel_linear, 'KFold', 10);
% Compute validation predictions
[validationPredictions2, validationScores2] = kfoldPredict(partitionedModel2);
% Compute validation accuracy
validation_error2 = kfoldLoss(partitionedModel2, 'LossFun', 'ClassifError'); % ...
    validation error
validationAccuracy_linear = 1 - validation_error2

%% Model perfomance
[prediction_linear,score] = predict(SVMModel_linear,X_test);

% Compute confusion matrix
C2 = confusionmat(Y_test, prediction_linear)
figure, confusionchart(C2,{'non','parkinson'});
title('SVM linear kernel')
TN2 = C2(1,1)
```

```
FN2 = C2(2,1)
FP2 = C2(1,2)
TP2 = C2(2,2)
accuracy_linear = (TP2+TN2)./(TP2+FP2+TN2+FN2)
Recall_linear = TP2./(TP2+FN2)
Specificity_linear = TN2./(TN2+FP2)
MCC_linear = ((TP2*TN2)-(FP2*FN2))./sqrt((TP2+FP2)*(TP2+FN2)*(TN2+FP2)*(TN2+FN2))
%% Model perfomance
[prediction_rbf,score_rbf] = predict(SVMModel_rbf,X_test);

% Compute confusion matrix
C1 = confusionmat(Y_test, prediction_rbf)
figure, confusionchart(C1,{'non','parkinson'});
title('SVM radial basis function kernel')

TN1 = C1(1,1)
FN1 = C1(2,1)
FP1 = C1(1,2)
TP1 = C1(2,2)
accuracy_rbf = (TP1+TN1)./(TP1+FP1+TN1+FN1)
Recall_rbf = TP1./(TP1+FN1)
Specificity_rbf = TN1./(TN1+FP1)
MCC_rbf = ((TP1*TN1)-(FP1*FN1))./sqrt((TP1+FP1)*(TP1+FN1)*(TN1+FP1)*(TN1+FN1))
```

References

Allen, J. B. and Rabiner, L. R. (1977), 'A unified approach to short-time fourier analysis and synthesis', *Proceedings of the IEEE* **65**(11), 1558–1564.

Artificial Intelligence in Healthcare: Industry Report (2019).
URL: *https://www.grandviewresearch.com/industry-analysis/artificial-intelligence-ai-healthcare-market*

Bates, D., Saria, S., Ohno-Machado, L. and Shah, A. (2014), 'Big data in health care: Using analytics to identify and manage high-risk and high-cost patients', *Health affairs (Project Hope)* **33**, 1123–31.

Boddapati, V., Petef, A., Rasmusson, J. and Lundberg, L. (2017), 'Classifying environmental sounds using image recognition networks', *Procedia Computer Science* **112**, 2048–2056.

Braga, Diogo, M. A. M. C. L. and Ajith, R. (2019), 'Automatic detection of parkinson's disease based on acoustic analysis of speech', *Engineering Application of Artificial Intelligence* .

Chaudhuri, K. R. and Schapira, A. H. (2009), 'Non-motor symptoms of parkinson's disease: dopaminergic pathophysiology and treatment', *The Lancet Neurology* **8**(5), 464 – 474.

He, K., Zhang, X., Ren, S. and Sun, J. (2015*a*), *Deep Residual Learning for Image Recognition*.
URL: *https://arxiv.org/pdf/1512.03385.pdf*

He, K., Zhang, X., Ren, S. and Sun, J. (2015*b*), 'Spatial pyramid pooling in deep convolutional networks for visual recognition', *IEEE transactions on pattern analysis and machine intelligence* **37**(9), 1904–1916.

Jellinger, K. A. (2019), 'Accuracy of clinical diagnosis of parkinson disease'.

Kaye, K. S., Anderson, D. J., Cook, E., Huang, S. S., Siegel, J. D., Zuckerman, J. M. and Talbot, T. R. (2015), 'Guidance for infection prevention and healthcare epidemiology programs: Healthcare epidemiologist skills and competencies', *Infection Control & Hospital Epidemiology* **36**(4), 369–380.

Khojasteh, P., Viswanathan, R., Aliahmad, B., Ragnav, S., Zham, P. and Kumar, D. K. (2018), Parkinson's disease diagnosis based on multivariate deep features of speech signal, *in* '2018 IEEE Life Sciences Conference (LSC)', pp. 187–190.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* 'Advances in neural information processing systems', pp. 1097–1105.

Lalkhen, A. G. and McCluskey, A. (2008), 'Clinical tests: sensitivity and specificity', *Continuing Education in Anaesthesia Critical Care & Pain* **8**(6), 221–223.

Long, E., Lin, H., Liu, Z., Wu, X., Wang, L., Jiang, J., An, Y., Lin, Z., Li, X., Chen, J. et al. (2017), 'An artificial intelligence platform for the multihospital collaborative management of congenital cataracts', *Nature biomedical engineering* **1**(2), 1–8.

Mathworks (2020), 'Convolutional neural network'. Accessed: 27.02.2020.
URL: *https://uk.mathworks.com/solutions/deep-learning/convolutional-neural-network.html*

Mcsharry, Patrick, S. J. and Hunter, E. J. (2009), 'Suitability of dysphonia measurements for telemonitoring of parkinson's disease', *Transactions on bio-medical engineering* .

Parkinson's UK (2017), 'The incidence and prevalence of parkinson's in the uk'. Accessed 2nd March 2020.

URL: *https://www.parkinsons.org.uk/professionals/resources/incidence-and-prevalence-parkinsons-uk-report*

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. (2019), Pytorch: An imperative style, high-performance deep learning library, *in* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds, 'Advances in Neural Information Processing Systems 32', Curran Associates, Inc., pp. 8024–8035.

Sakar, B., Isenkul, M., Sakar, C. O., Sertbaş, A., Gurgen, F., Delil, S., Apaydin, H. and Kursun, O. (2013), 'Collection and analysis of a parkinson speech dataset with multiple types of sound recordings', *Biomedical and Health Informatics, IEEE Journal of* **17**, 828–834.

Tolosa, E., Wenning, G. and Poewe, W. (2006), 'The diagnosis of parkinson's disease', *The Lancet Neurology* **5**(1), 75–86.

Vaiciukynas, E., Gelzinis, A., Verikas, A. and Bacauskiene, M. (2018), *Parkinson's Disease Detection from Speech Using Convolutional Neural Networks*, pp. 206–215.

Vasileiou, K. (2016), 'Msc thesis: Speech emotion recognition (ser) system', *Department of Electronic and Electrical Engineering, University of Strathclyde* .

Yoon, T.-J. (2008), 'Nsp2wav'. Accessed 20th March 2020.

URL: *https://www.uvic.ca/humanities/linguistics/assets/docs/praat/NSP2WAV.praat*