Abstract

This project applied convolutional neural network (CNN) for classification of electrocardiogram (ECG) signals labeled as SR (Sinus Rhythms), VT (Ventricular Tachycardia) and VF (Ventricular Fibrillation). Because CNN is well known for classifying 2-dimensional images, Short Time Fourier Transform has been used to transform the ECG signals to spectrogram images as input for CNN we created by ourselves. We have created images from 2 second to 10 second (or s in this report) with window size of 1s, 2s and 3s and overlap of 0, 25% and 50%. With 2s segment size, 2s window size and 25% overlap, we obtained the best result of sensitivities on SR, VT and VF, which are 97.6%, 76.9% and 78.1% respectively. Comparing the results of different window sizes and input image sizes, we draw the conclusion that the classifiers of selected window sizes (1s, 2s and 3s) perform similarly and images with smaller size (256×256 pixels compare with 192×192 pixels) achieve better results.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Zoran Cvetkovic for giving me the chance to work on such an interesting project and without his expert guidance, insight and patience, completion of this project would not have been possible.

Furthermore, I wish to thank my friends and families for supporting and encouraging me during this period of life.

Contents

List of Figures

List of Tables

1 Introduction and State of Art

1.1 Problem Statement

As the World Health Organization stated, cardiovascular disease has been one of the leading causes of human death for both developed and developing countries. Therefore, many researchers are trying to carry up appropriate and effective medical treatment for cardiac arrhythmia. One of the key factors of their success is to distinguish different forms of arrhythmias and give the definitions. Until recently, precise diagnostic criteria have been established in terms of language, which is a problem for computer-aid diagnosis because automatic discrimination between those diseases cannot be programmed without a numeric form. Among the diseases, Ventricular Fibrillation (VF) and Ventricular Tachycardia (VT) can be problematic for computer-aid diagnosis since VF and VT are not always unequivocal. This is because VT is not always monomorphic, and a range of transient and persistent forms of polymorphic VT can happen. Discrimination between VF and polymorphic ventricular tachycardia (VT) or Torsade De Pointes (TDP) is very challenging, but it is deemed critical since different treatment is needed for VT and VF as well as the false classfication may cause the damage to patient's health or even cause death.[1][2]

As the computer-aid diagnosis is one trend of machining learning application, the issue aforementioned has encouraged plenty of studies to find the reliable approach to classify VT and VF by utilizing fragments of electrocardiogram (ECG) signal. It is interesting to note that even many of these studies claimed that they have obtained very high accuracy in such case, but the death rate of the diseases is still extremely high and this could result from few causes, for example, the lack of ability of re-conducting the research because of the data authentication. [1][2]

1.2 Introduction

This project aims to classify normal Sinus Rhythms, Ventricular Fibrillation and Ventricular Tachycardia by using convolutions neural network as the classifier. The original

data is 1-dimensiona signal dataset which has the frequency of 100Hz and it contributes to one of the major difficulties of developing this project. Because CNN has good performance in image classification with 2-dimensional data, the visualization of the data by spectrogram is applied. The reason for choosing this method and how it can be developed will be explained in later sections. In addition, the essential properties such as segment size (how many samples in one spectrogram image), window size (size of window function) and overlap size (how many percents each window overlaps with adjacent windows) are used to achieve better sensitivity for each class (SR, VT, VF) and overall accuracy will be introduced in later parts as well.

1.3 Project Objective

The objective of this project is to evaluate the CNN classification performance of ECG signals labeled as SR, VF, and VT in terms of sensibility. The expected sensitivities of SR, VF, and VT are over 90%, 50%, and 50% respectively.

1.4 Literature Survey

To understand, design and develop this project, we have researched and applied the concepts or methodologies from several studies. This section lists all the important studies which give us the guide for designing or constructing this project.

Alwan et al. applied Support Vector Machine to classify ECG signals labelled as Sinus Rhythms, Ventricular Tachycardia and Ventricular Fibrillation. The study picks only 6000 seconds of each class from the original signal to deal with the unbalanced data and they obtained the result of 91% overall accuracy and 86% sensitives for each class with 2 second observation length.[1]

Research by Yuan and Cao used the convolutional neural network to classify the electroencephalogram (EEG) signals and evaluated the condition of patients suffering from brain damage. A novel method using Short-time Fourier Transform (STFT) to generate spectrogram from the EEG signals, and using the spectrograma images as the input dataset of CNN, is proposed in this paper. Achieving the highest accuracy of 96% and

94% for classification on common and brain death EEG signals, the approach is considered as a remarkable success.[3] Similar approaches have been also applied to other datasets such as audio signals.[4]

Research by Kostiantyn Pylypenko suggests that the preprocessing is necessary when the dimension of input dataset is huge. The reason is stated as this preprocessing has the computational benefit and this also helps deal with overfitting problem. [5]

According to Sinalis et al., the input of CNN can be raw 1-dimensional signals other than only 2-dimensional data. They have successfully applied convolutional neural network to EEG signals as input and compared the results with the outcomes from using spectrogram images as input images. Although the raw signal does not outperform spectrogram in this case, it still is able to learn usable features. As there is a growing interest in applying CNN with 1-dimensional data, this study appears to be very appealing. [6]

2 Technical Background

This section lists and describes all the technical knowledge the reader has to acquire to understand the report clearly.

2.1 Arrhythmia

This section explains the definition Sinus Rhythem, Ventricular Tachycardia and Ventricular Fibrillation as well as their characteristics and symptoms.

2.1.1 Sinus Rhythms

Sinus Rhythm or SR, sometimes refers as normal sinus rhythm, is considered as a normal and regular heart rate and rhythm. The shape of ECG tracing in Figure 1 shows the main traits of SR. As the figure illustrates, the first peak noted by P is the atria contraction, also called depolarization, which is caused by the atria squeezed. After that, there is an intensive wave noted as QRS wave causing by ventricle depolarize, which is much stronger because the ventricular beats stronger than atria. Then, a T wave appears because of the repolarization of ventricular, the atria also have this activity but are overwhelmed by ventricle depolarization. If the patient's ECG shows the characteristics of SR, there is no specific treatment required urgently.

2.1.2 Ventricular Tachycardia

Ventricular Tachycardia, also know as V-Tach or VT, is a fast yet regular heart rate raise from the lower chambers of the hear, namely ventricles, which is the main pumpers of the heart that pump blood to the rest of the body. In normal condition, the heart typically beats from 60-100 beats per minute (bpm), and between the heart beats the ventricular walls relax to fulfill the heart with blood then when heart beats the walls contract the blood has been pushed to the rest of the body. For VT, however, the heart is beating very fast, sometimes up to 250 bpm, and this results in the heart cannot circulate enough blood to the body. Although this might seem constitutive, as in normal

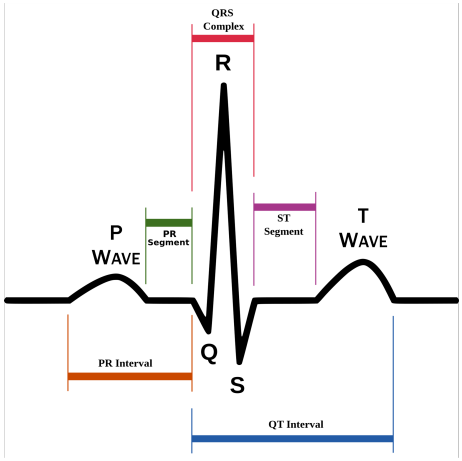


Figure 1: Example of Sinus Rhythm ECG tracing

situation we would assume that the faster the heart beats the more blood will be send to the body, the truth is that during the heart beats the blood will be fulfilled into the heart and if heart beats too fast it will not be enough time for heart to fill enough blood and as a result the blood circulate to the rest of the body will be insufficient. When one suffer from VT they might fell the shortness of breathless, chest pain, light headed or dizzy. In extreme cases, the oxygen to the brain is too slow so that the person might even faint or pass out. Therefore, VT is a very serious condition and requires immediate medical attention. [7]

2.1.3 Ventricular Fibrillation

Ventricular Fibrillation, also known as V-Fib or VF, is abnormal, deadly heart rhythm causing by the ventricles lose the ability to contract and circulate blood to the rest of the body. It is caused by the heart lose conducted the signal in ventricles, and instead, it will have the rapid, random and chaotic signals, which leads to the ventricular walls spasming. And when this happens, blood cannot be circulated to the body and brain, instead, they will stay in the heart and deprive all other organs of oxygen. If the VF is not reversed by electronic shock immediately, the patient is going to have permanent brain damage or even death, because the brain and the body are not getting enough

oxygen. [8]

2.2 Short Time Fourier Transformation

To comprehend better of Short Time Fourier Transform or STFT, a clear understanding of Discrete Fourier Transformation (DFT) is necessary. DFT is regarded as the Continuous Fourier Transformation but DFT is applied for signals which known only at N samples separated by sampling time T. The original Fourier transformation will be:

$$F(j,w) = \int_{-\infty}^{\infty} f(t)e^{-jwt}dt \tag{2.1}$$

where the $f(t)$ is the continuous signal, in DFT the N sample aforementioned can be denoted as as $f(0), f(1), ..., f(N-1)$, which results in:

$$f(0)e^{-j0} + f()e^{-jwT} + ... + f(N-1)e^{-jw(N-1)T} \tag{2.2}$$

or,

$$F(j,w) = \sum_{K=0}^{N-1} f([k])e^{-jwkT} \tag{2.3}$$

In principle, we could apply 2.3 for every w,which is the radial frequency equals to $\frac{2\Pi}{T}$, but since only the N samples significance in DFT, we will consider only the correlated outputs. As the DFT assumes the data are periodic, the w can be set as fundamental frequency and it's harmonics as

$$w = 0, \frac{2\Pi}{NT}, ..., \frac{2\Pi}{NT} \times (N-1) \tag{2.4}$$

Therefore, the equation 2.3 can be transformed as

$$F[n] = \sum_{k=0}^{N-1} f[k]e^{-j\frac{2\Pi}{N}nk} \tag{2.5}$$

$F[n]$ is the DFT coefficient of of the sequence $f[k]$.
Short Time Fourier Transformation is one type of Fourier Transformation which separates the data into few equal length segments and apply DFT to each of these segments to generate few results. Additionally, a window function has been multiplied to DFT function to each segment and these windows can usually overlap, typically from 25% to 50% with each to reduce the effect of spectral leakage. [9]

2.3 Resolution Issue

One thing to keep in mind when using STFT is the resolution issue according to the Uncertainty principle, which means the different window size will affect the way how the information will be presented. Trade off between good frequency resolution and time resolution have to be made. This principle suggests that the better frequency resolution but poor time resolution will be obtained from the longer window and vice versa. The example of this is shown as Figure 2. It is shown that image with 3s window size (the right part) has more DFT coefficients and displays more frequencies than 1s window size, which is located in the left part. This image presents the location of changes in a more precise way while the frequency resolution is reduced. This particular principle is one of the primary reasons why we have created spectrogram images with different window sizes.

2.4 Spectrogram

Spectrogram can be described as the visual presentation of the intensity of the frequencies of the signal which has the trait that the frequency changes over time. As Figure 3 illustrates, the X-axis represents frequency which in this case is the normalized frequency. Normalized frequency ranges from 0 to 1 and can be calculated as cycles/sample. For example of 800 seconds, if the STFT result is 500Hz, the exact number in this scale is 500/800 which is 0.62. Y-axis, on the other hand, always changes because it represents the time, which in this case, is actually the sample numbers. The frequency of the ECG signal of this project is 100Hz which means 100 samples will be the equivalence of 1s.

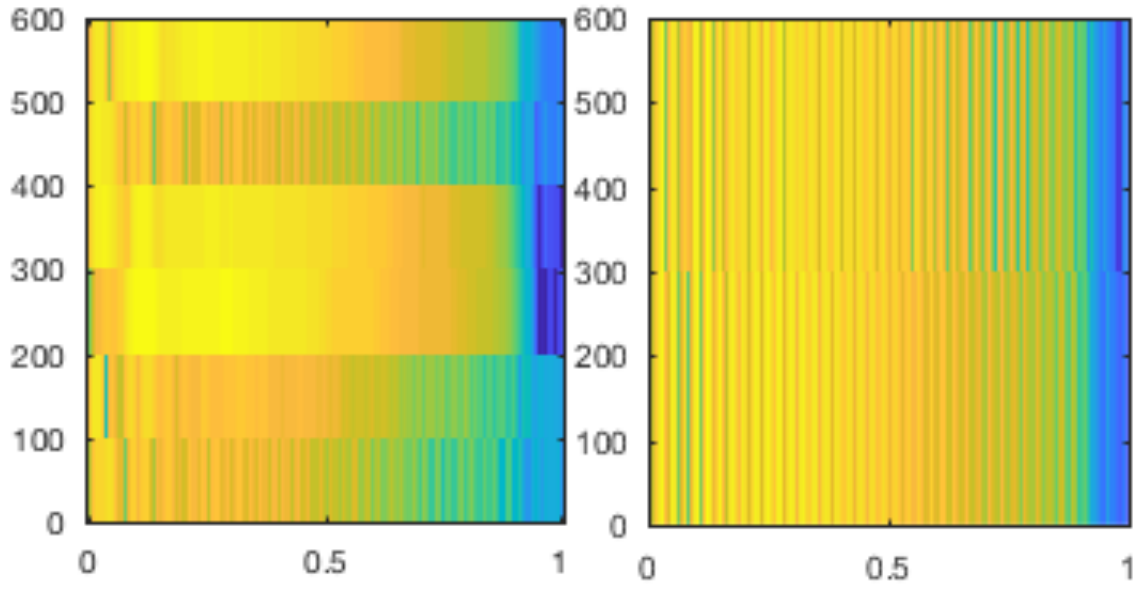


Figure 2: Comparison of 1s and 3s window size spectrogram images

Segment time is presented with the Y-axis. For instance, 3s figure has Y-axis equal to 300. Additionally, the color of the spectrogram stands for the intensity or magnitude. The brighter color in spectrogram means high intensity while dark color stands for low intensity. Furthermore, one can notice the horizontal lines when looking closely at the provided spectrogram images. Those lines are, in fact, the result of each window function and the size of each line are affected by the overlap of the windows. For instance, Figure 3 illustrates the spectrogram image result from 3s segment size (300 samples are used to generate one image) with 1s window function with 50% overlap on each window. From the example, it can be seen the length of first and last window is around 75 which is different from other windows with the length of 50. This is the result of the overlap operation. When 50% overlap window is specified, it has been split into two overlap part of 25% to each end of the window. And since the first and last window have only one side overlap with neighbour windows, these two windows have the size of 75 while others have 50 size due to the two overlaps at each end. [10]

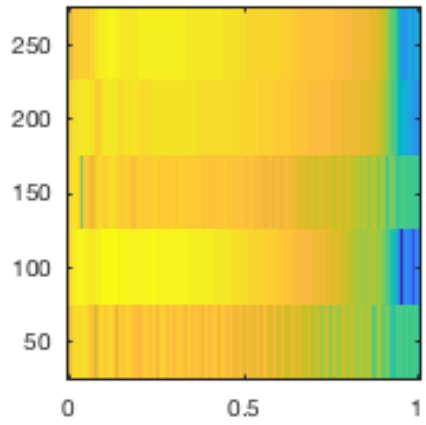


Figure 3: Example of Spectrogram with window size 3s

2.5 Convolutional Neural Network

Convolutional neural network, also known as CNN, has been proven very effective in some fields such as image recognition and classification. The basic idea of CNN is that giving the computer images, which will be dealt by the computer as an array of numbers, it will then calculate out the numbers that indicate the probability of the input being a certain class and claim the class input belong to, which is based on the highest probability. CNN is one type of feed-forward artificial neural network where the neurons inside each layer have learnable weights and biases. The connectivity pattern between the neurons in CNN layers is inspired by the animal or human visual cortex organization. Because when a person look at a picture of human, he is capable to classify what is the object that the picture presents if the image has identifiable features such as hands or fingers. Computers, on the other hand, is capable of performing tasks like image classification only by looking for features at low level such as edges and curves at first. Then, through a sequence of different layers, computers can then build abstract concepts such as fingers. CNN stacks multiple layers that are input layer, output layer and hidden layers which are convolution layer, pooling layer, Re-Lu layer and fully connected layer. In the following section, each layer will be described in detail.[11]

For the better understanding of convolutional neural network, there are two terminologies

have to comprehend ahead, which are Convolution Kernels and Receptive Fields.
A convolution kernel, also known as filter, is practically a small sized matrix of real valued entries and then convolve with the input volume to generate the Activation Maps. Those maps are created by summing the results of selected field using dot production. After record the dot production, the kernel then slides by a stride value until the entire input volume has been processed. This process happens in every color channel. Figure 4 illustrates how this operation works for an input matrix size of 7×7, which applies a 3×3 filter with stride equals to 1 and generates the activation map of 5×5. The green color highlighted result is the result of the red region from the dot production of the convolution kernel, the calculation is presented in Equation 2.6.[12] [13]

$$(1,1,0).(1,0,1) + (1,1,0).(0,1,0) + (1,1,1).(1,0,1) = 1 + 1 + 2 = 4 \tag{2.6}$$

Through few hidden layers, regular Neuron Network receives and transforms an input into output. Each of these hidden layer consists of a number of neurons and each of the neurons fully connected to all the neurons in the previous layer.
CNN defines each neuron connect to a relatively smaller 2-dimensional spatial region name Receptive Region ($Width \times Height$) and extends it to the full size of input volume depth by default. The way each neuron connects to paired Receptive Field is, however, fully connection and this means that the neuron connects to each pixel inside the filed. For example if we have an input size of $192 \times 192 \times 3$ and the receptive field is 3×3 then each neuron in this layer will have weights of 9 ($3 \times 3 \times 3$). These small regions enables the cross-section operation of CNN and produces activation maps. [12] [13]

2.5.1 Image Input Layer

Images are typical inputs of CNN and each one is treated as a matrix of pixel values by computer. Therefore, Image Input layer is the CNN layer for receiving images as the CNN input. Having different bit size, each pixel in a image encodes various values. Most common sizes of the pixel are 8 bit or 1 Byte, thus resulting the range from 0 to

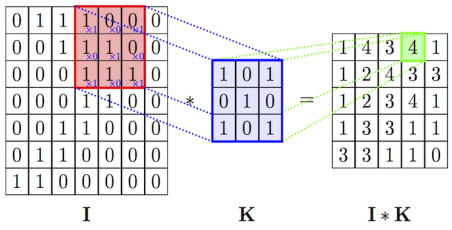


Figure 4: Convolutional layer example

255. Additionally, for colorful images, especially RGB (Red, Green, Blue) images, the separated color channels (3 if the input is RGB images) adds an additional depth field to the input data, making it 3-dimensional. As a result, for every given image if it is RGB and has 240 × 240 ($Width \times Height$) pixels, it is necessary to have 3 matrices for each image, besides the width and height we need another one for the color channel. Therefore, the images having a 3-dimensional structure are referred as the Input Volume, in this case, (240 × 240 × 3). [12] [13]

2.5.2 Convolutional Layer

Also known as Conv layer, convolutional layer is the crucial part of CNN and does the core operations such as training the network and refines the neurons. As the name suggests, it does convolution operations to the input as mentioned in convolutional kernel part, for each input it applies the filer named convolutional kernels, and move this filter around by a specifies number (stride) all over the input volume. Consisting of few neurons, Conv layer arranges them in a 3-dimensional manner with Width, Height and Depth Column which is same depth of the input volume and fully connects each neuron to a small region of input named Receptive Filed as stated before. Similar to all the feed-forward neural networks, CNN trains and upgrades the weights through back propagation algorithm on each learning iteration with a learning rate which defines how much does the parameters changes every iteration. [12] [13]

2.5.3 Pooling Layer

Also known as down-sampling layer, this layer is conventionally inserted between two Conv layers to reduce the spatial dimension, $Width \times Height$, of input volume of the Conv layer placed afterward, while the depth dimension will remain exactly same. As the name suggested, this layer does downsampling operation because of information loss due to size reduction, which is actually beneficial in Neuron Networks because not only it reduces the computation overhead for later layers, it helps with overfitting as well. The most common pooling layer applies 2×2 filters with a stride of 2 and picks out the max value in the selected region to form an output, discarding 75% of the activation map generated by the previous layer. Except for the max operation, pooling layer can perform other operations as well such as average pooling. In summary, pooling layer accepts an input volume of size $W \times H \times D$ and downsamples this input by two hyperparameters of spatial extent F and a stride of S then generates an output of $[(W-F)/S+1] \times [(H-F)/S+1] \times D$. It is worth to mention that the most common pooling layers are F=3, S=2 (which the filter overlap with each other) and F=2, S=2, larger receptive fields than this will be counterproductive. [12] [13]

2.5.4 ReLu Layer

ReLu here is the abbreviation of Rectifier Linear Unit, in this layer all the negative values in activation map change to be 0, which is for generalization. [11]

2.5.5 Fully Connected Layer

As its neurons is fully connected to all outputs of the previous layer like regular Neural Networks, Fully Connected layer takes the output matrix of the previous layer and generates an output vector. The vector generated by Fully Connected layer has the number of dimensions same as classes that the program has to predict the input to be. In this layer, all the values vote for the final result, and for each class the associated weight on each value will be different, thus, result in different outcomes. Regularly, the

outcome of highest values becomes the class from CNN predicted outcome and this is commonly the last stage of CNN. [12] [13]

2.5.6 Layer Pattern

The most common pattern of CNN is to stack up few Conv+ReLu layers follow by a Polling layer and repeat this pattern (normally, between 1 to 3 times) until the input image has been processed to a small spatial size. The last fully connected layer holds the class from CNN predicted outcome. [12] [13]

2.6 Data Balancing and Approaches

Typically, imbalanced data refers to the situation where the minority class is significantly smaller than the major class, which is very usual among medical data set, such as ECG or EEG, because there are always much more general behaviors than focused ones. This imbalanced data can make classification algorithms suffer due to the fact that typical classifier is more sensitive to major class while less sensitive to the minority class. The preprocessing of these imbalanced data before feeding them into classifiers is so called data balancing. One thing to note that the absolute equivalence of data is in fact very rare, so for the small difference we can just ignore while the significant difference is not tolerable.

According to the ECG signals we use for this project, we have much more SR data than VT and VF, therefore the preprocessing of the imbalance data is very necessary. There are more than one ways to deal with imbalanced data, in this report we will only emphasize on two methods, namely subsampling[1] and confusion matrix. As one of the most common and straightforward strategies to cope with imbalanced data, subsampling in this project means to pick the same amount of samples from each of 3 classes to achieve balanced data. As the accuracy is not the best measurement for imbalanced data, we choose another matrix named confusion matrix to see the details of accurate and inaccurate predictions, as well as other measurement terms such as sensitivities. The details of confusion matrix will be explained in the design chapter.

3 Project Design

3.1 Data Transformation

In this project, we process the original signal data with 2 steps. Firstly, the STFT has been used to transform the 1-dimensional signal data into 2-dimensional spectrogram images. Then, the unnecessary parts are discarded as well as the size of images are reduced for performance reasons.

3.1.1 Signal Processing

The original data of this project is 1-dimensional ECG signal and the classifier we choose to use is convolutional neural network which is typically used to classify 2-dimensional data such as images. As a consequence, it is not difficult to understand why it is necessary that we need to reprocess the original 1-dimensional data into 2-dimensional images. There is no research trying to apply CNN to ECG signals as of now, but luckily, we have found one research using the same classifier, CNN, on electroencephalogram (EEG) signal and choose this study to be the guideline of how to process the signal data. In the study, Yuan and Cao tried to use CNN to classify brain damage and normal EEG data by transforming every 20 seconds signals into spectrogram images using STFT method on MATLAB platform. Furthermore, to make the most of the finite size data, each window in STFT is 20% overlapped with the adjacent windows. Following this research, we decided to transform our ECG signal into spectrogram images to train CNN with parameters which are number of samples to generate one image (segment size), window size and overlap percentage remains to be exposed by experiment for achieving the optimal result. [3]

3.1.2 Image Re-processing

Due to the input layer format in CNN, the produced spectrograms have to be re-sized in order to be able to feed into the network.[5] Yuan and Cao suggest the image size of 256×256 [3] which is the max spatial limitation of the input layer of CNN while another

study conducted by Kostiantyn Pylypenko argued that re-sizing the image to be 192×192 actually leads to better performance in terms of running time and classification result.[5] To guarantee that we will achieve the best result as we can, in this project some images are re-sized both in 256×256 and 192×192 to compare the results and provide the guide for the future work.

3.2 CNN Architecture and Parameter

The CNN architecture and parameters are defined with MATLAB provided functions, which will be explained in detail in Development section 4.3.

3.3 Result Measurement/Confusion Matrix

Because of the imbalanced original data and wishing to explore more aspects of experiment outcomes, we choose confusion matrix to be the measurement of classification result.

As Kevin Markham's descriptions, the confusion matrix is a table lists all the test data with their true values and predicted values. One thing to note is that the confusion matrix can be more than 2×2, in our case the matrix has size of 3×3. For easier understanding, Figure 5 is a given example of confusion matrix of a binary classifier. The further explanation will be based on this example. Firstly, there are four basic terms to understand which are:

True Positives (TP): This is the case when the classifier predicted yes and the actual value is yes as well, which is the cell of second row and second column.

True Negatives (TN): This is the case when the classifier predicted no and the actual value is also no , which is the cell of first row and first column.

False Positives (FP): This is the case when the classifier predicted yes and the actual value is no, which is the cell of second row and first column.

False negatives (FN): This is the case when the classifier predicted no and the actual value is yes, which is the cell of first row and second column.

Based on the four basic terms, we can continue to some rates used to describe the clas-

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Figure 5: Example of Confusion Matrix

sification performance of the classifier. [14]

3.3.1 Sensitivity

Sensitivity, also known as Recall, means when it is actual yes how often does the classifier predict yes or when it is actual no how often does the classifier predict no.

$$TP/ActualYes = 100/105 = 0.95 \tag{3.1}$$

$$TN/Actualno = 50/60 = 0.83 \tag{3.2}$$

3.3.2 Accuracy

Accuracy, which means in overall, how often does the classifier make correct predictions.

$$(TP + TN)/Total = (100 + 50)/165 = 0.91 \tag{3.3}$$

Error Rate, also known as misclassification rate, means how often does the classifier gives the wrong prediction. The rate can be calculated as one minus accuracy or as Equation 3.4.

$$(FP + FN)/Total = (10 + 5)/165 = 0.09 \tag{3.4}$$

3.3.3 Precision

This rate means when the classifier make a prediction, how often it is accurate.

$$TP/ActualYes = 100/110 = 0.91 \tag{3.5}$$

$$TN/ActualNo = 50/55 = 0.90 \tag{3.6}$$

[14]

4 Development

This section describes the development tools used for the project as well as the explanation of essential functions.
To develop this project, a Windows 10 computer with an Intel Core (TM) i7-5500 U (2.40 GHz), a 16.0 GB DDR3 RAM and a nVidia 940m with 4GB VRAM Graphics Processing Unit (GPU) was used.

4.1 Programming Language and Library

4.1.1 MATLAB

The programming language choice of this project is MATLAB (version R2017a-9.2.0.538062) which stands for MATrix LABoratory and was originally written for providing easy access to matrix software developed by linear system package and Eigen system package projects. Integrating visualization, programming environment, and computation, MATLAB is not only a high-performance programming language but also an language environment which contains sophisticated data structure and built-in debugging tools. Due to the mentioned traits, MATLAB has been the excellent tool for researching. Most importantly, MATLAB has powerful libraries for neural network, signal processing and image processing, which is extremely necessary and helpful for this project due to the problem domain. [15] [16]

4.1.2 Important Libraries

Specific applications which are collected in a package are referred as Toolbox in MATLAB. To complete this project, we choose a MATLAB toolbox called Neural Network Toolbox which provides apps, algorithms, and already trained models for creating and training the neural networks. In addition, the Parallel Computing and Signal Processing toolbox have been used for utilizing GPU in computation and processing signal data such as spectrogram image generation as well.

4.2 Spectrogram Generation

This section describes how spectrogram images are generated from the given ECG signals. The main steps are choosing the number of samples for generating one spectrogram image (segment size), image generation and re-process images for training performance.

4.2.1 Segment Size

The sample numbers inside one spectrogram image are defined by variable *segment_size*, and for each channel (SR, VT and VF) we define *start_point* which is initially 1 then change to the ending point after each iteration and end_point which equals to the size of *start_point* plus *segment_size*. After defining the variables, we use *while* loop to apply this interval iteratively in SR, VT and VF to pick around 6000s (600000 samples) from each of the three channels. One thing to notice is that the original data is stored in cells and as one cell contains different numbers of samples it will result in such case that the remaining samples in the cell is less than the window size we defined. Under such situation, we will discard the remaining data in this cell and jump to next cell with *start_point* equals to 1 again.

4.2.2 Spectrogram Images Generation

Using STFT to generating the spectrogram based on several given parameters, the *spectrogram(x, window, noverlap)* function is applied for crafting all the required spectrogram images. *x* is the input signal which in our case are defined by *start_point* and *end_point*, which picks out *segment_size* of samples from the cells of each class. Defined as variable *window_size*, parameter *window* in this function are used to split the signals into pieces and perform window function on each piece. The last parameter *overlap* means we will use this number of samples to overlap between adjacent pieces. This defined by the multiplying different percentages with *window_size* for the purpose of searching for optimal result. The parameters of this function change values according to the images we aim to produce, while hamming window function are chosen for all the experiments. Figure 6 is the snapshot of example spectrogram function.

```
spectrogram(VT{1,index}(start_pos:end_pos,1),hamming(window_size),ceil(window_size)/4)
```

Figure 6: The snapshot of spectrogram function.

```
xlabel('');
ylabel('');
set(findobj('type','axes'),'fontsize',5);
set(h, 'PaperUnits' , 'points');
set(h, 'PaperPosition' , [0 0 123 123]);
set(h, 'Position' , [0 0 123 123]);
```

Figure 7: The snapshot of code for reprocessing the image.

4.2.3 Re-process images

Typically, the images produced by *spectrogram(x, window, noverlap)* function will be larger than the defined spectrogram size of CNN image input layer. Because those images include other information such as annotations which does not contain any useful information for CNN. Consequentially, we remove the unnecessary part and re-size the images in order to use them as input images for CNN. Fortunately, this process can be simply completed by leaving *xlabel* and *ylabel* to be empty and set image position and paper position to be the aimed size. Figure 8 shows the comparison of processed and after-processed spectrogram images while Figure 7 is the snapshot of code for reprocessing the image.

4.3 Convolutional Neural Network

This sections focus on addressing how the CNN are created and how it is trained with the images generated in previous section.

4.3.1 Training Options

As it is providing a list of already written functions designed specifically for setting up and training a convolutional neural network, Neural Network Toolbox reduces our workload significantly. Before building up a network, we specified few training options such

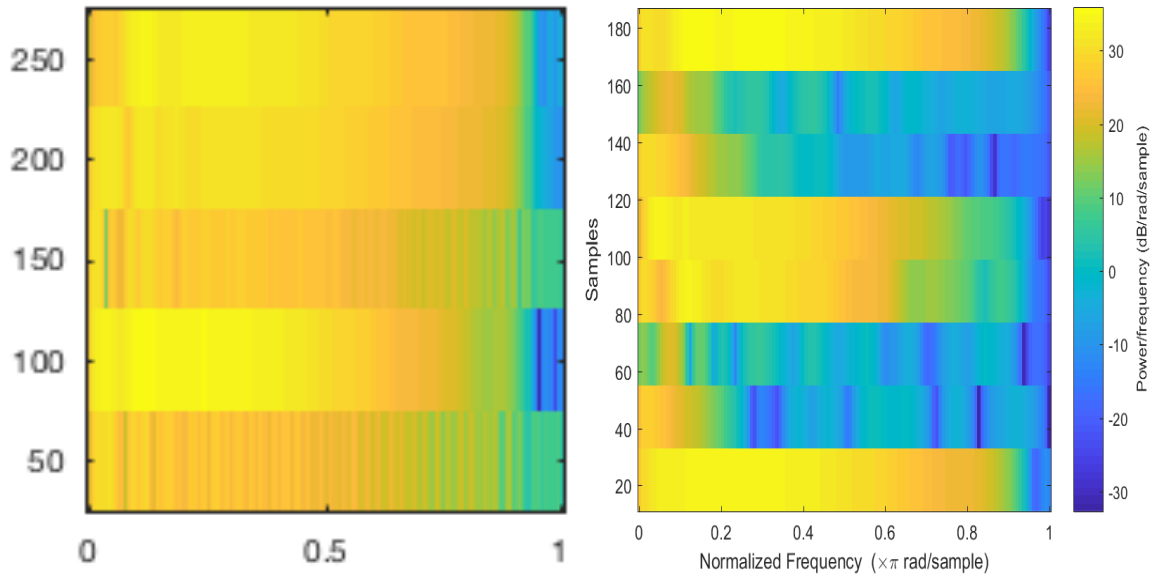


Figure 8: Comparison of prepossessed and after-processed spectrogram images

```
options = trainingOptions('sgdm','MaxEpochs',30, 'InitialLearnRate',0.0001);
convnet = trainNetwork(trainDigitData,layers,options);
YTest = classify(convnet,testDigitData);
```

Figure 9: Training Option in MATLAB

as learning rate by applying *trainingOptions(solverName,Name,Value)* function which returns a set of training options specified by name value argument pairs. Figure 9 is the screen shot of the training options we specified, where we set to train the network using stochastic gradient descent and maximum number of training epochs is 30 with initial learning rate is 0.0001 which remains same through the training process.

4.3.2 Network Architecture

Neural Network Toolbox provides functions for every CNN layer, here, we will only describe the one we need and used for this project. Figure 10 shows the CNN architecture and parameters used for this project.

Image input layer: This layer is defined by function *imageInputLayer(inputSize)* where

```
layers = [ imageInputLayer([256 256 3])
           convolution2dLayer(5,30)
           reluLayer
           maxPooling2dLayer(2,'Stride',2)
           convolution2dLayer(5,30)
           reluLayer
           maxPooling2dLayer(2,'Stride',2)
           fullyConnectedLayer(3)
           softmaxLayer
           classificationLayer()];
```

Figure 10: CNN Architecture and Parameters in MATLAB

inputSize specifies a vector of two integers standing for [Height,Width] or three numbers of [Height,Width,Channels]. To obtain the optimal result we use two size images as we mentioned before, the image input layers have two *inputSize* of $192 \times 192 \times 3$ and $256 \times 256 \times 3$.

Convolutional layer: This layer is defined by function *convolution2dLayer(filterSize,numFilters)*, in which the *filterSzie* defines the filters with the width and height and if only one number is given which means it is a scalar value, the width and height will have same value. Another parameter *numFilters*, number of filters, specifies an integer which determines the number of feature maps will be generated by the Conv layer. Having thirty filters with height and width of 5, our conv layer is defined as *convolution2dLayer(5,30)* which does not define stride and padding variables and let them to be default values of 1 and 0 respectively.

ReLu layer: This layer is defined by function *reluLayer()*, which performs the typical threshold operation, where all the negative values are set to be 0.

Pooling layer: This layer is defined by function *maxPooling2dLayer(poolSize)* which performs max pooling by dividing the input into regions with both Width and Height of *poolSzie* and returning the biggest value in each region. In this project, all the pooling layers are defined as *maxPooling2dLayer(2,'Stride',2)* which performs max pooling operation on 2×2 regions having no overlap with each other.

Fully connected layer: This layer is defined by function *fullyConnectedLayer(outputSize)*,

where *outputSize* should be equal to the number of classes to be classified if this is the last layer placed before the softmax layer. Thus, in our case, the *outputSize* is 3 because the signals data for our project are labeled as SR, VT and VF.
The last two layers of the CNN are softmax layer and classfication layer defined by *softmaxLayer()* and *classificationLayer()* respectively.
Instead of using pretrained CNN provide by Neural Network Toolbox, we created a new CNN from ground up. The architecture pattern of CNN of this project follows the suggestion mentioned in Section 2.5.6, and this means the CNN architecture we use is having 2 Conv-ReLu stacks, and is followed by ReLu layers.

After defining the parameters of each layer and the architecture, the network are trained with *trainNetwork(imds,layers,options)* provided by Nerual Network Toolbox, where *imds* parameter means labeled images for the classification, *layers* and *options* are CNN layers and the training option we defined earlier.

4.4 Confusion Matrix Generation

As it is sufficient for illustrating sensitivity, precision and overall accuracy, Confusion Matrix has been chosen to be the tool for showing the results. MATLAB provides specific function designed as *plotconfusion(targets,outputs)*. This function plots the confusion matrix based on targets matrix and outputs matrix. In this function, *targets* is a $N \times M$ matrix which represents the actual class labels, where N is class numbers and M is the number of all testing samples. Each column in *targets* has 1 one and N-1 zero, indicating the correct class and false class respectively. Similarly, *outputs* are specified by a $N \times M$ matrix as well which stands for the predictions made by CNN on same testing samples. The difference between *targets* and *outputs* is that, in *targets* each column contains values only 0 or 1 while for *outputs* each column can contains the possibilities between 0 to 1. And the summation those elements in one column will equal to 1. [17] The difficulty for applying this function directly to our project is that all the testing data we use are class labels not numeric matrix. Thus, we implement a function named *confusionmattransform()* to transform all the data to be same format as

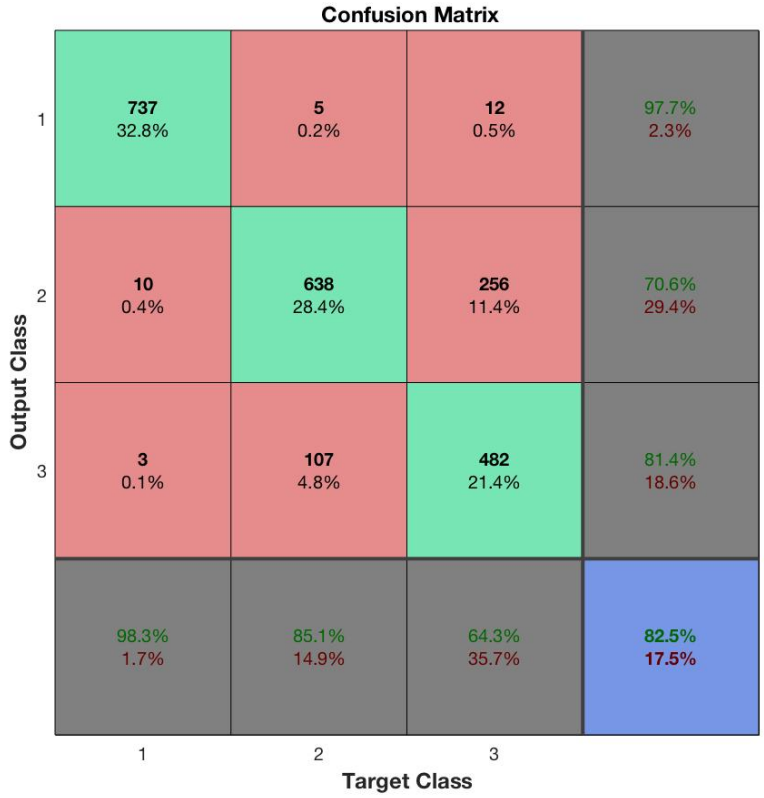


Figure 11: Example of confusion matrix

required by *plotconfusion(targets,outputs)*. Figure 11 shows one example of confusion matrix generated by MATLAB in which 1, 2, 3 stands for SR, VF and VT respectively.

5 Experiment Procedure and Result

This chapter reports the essential procedures during the experiments in sequence and reveals the important results as well as the analysis of the presented results.

5.1 Experiment Procedures

5.1.1 Data Preprocessing

As we mentioned before, to obtain the optimal result, the different lengths of segments for forming one spectrogram image are used, which the first step of our experiment. Until this point, we generate spectrograms from 2s (200 samples) until 10s (1000 samples). Furthermore, the images are generated by STFT, which means that the window overlap may effect the spectral leakage and consequentially effect the experiment results. Therefore, we also generate spectrogram with overlap of 0, 25% and 50% for every segment size.

5.1.2 Network Construction

The final CNN architecture of this project is two stacks of Conv-ReLu (the details are explained in Section 3.5.6), each one follow by a pooling layer and the last three layers are Fully connected layer, Softmax layer and classification layer. The filter sizes are set to be 5, while the numbers of filters are set to be 30 which is limited by the computation capability of the resources. The stride values for both Conv layer and pooling layer are set to be 1, and the regions in pooling layer where max pooling happens is always set to be 2×2. Moreover, the learning rate, epoch and optimization method stay as 0.01,30 and Stochastic gradient descent through all the experiments. Figure 7 and Figure 8 shows the training option and CNN, all the presented results in next section are obtained form this CNN.

5.2 Experiment Result and Analysis

This section lists the results obtained from all experiments in terms of sensitivities of each class and overall accuracy.

Table 1 shows the experiment outcomes using $192 \times 192 \times 3$ spectrogram images with 1s and 2s window size. According to the presented data, all the overall accuracy are considerably high with the least value at 76.90%. Also, all sensitivities of SR exceed 95% while the sensitivities for classifying VF varies significantly from 91.6% to 62.0% from with different segment size. Having even more gap between outcomes, sensitivities of VT cover from slightly below 50.0% to 88.0%. Also, when looking closely into Table 1, one may notice that for one training, usually when VF achieves very high sensitivity, the sensitivity of VT, however, is low. For example, the experiment with 10s segment size, 1s window size, 25% overlap gives highest sensitivity of VF of 92.0% while sensitivity of VT dropped dramatically to 48.7%. Thus, when we pick the best result to present, it is necessary to make the trade off between VF and VT sensitivities. And according to this, the classifier acting on 2s segment size, 2s window size and 25% percent overlap are considered to give best result, with sensitivities of SR , VF and VT achieving 97.6%, 78.1% and 76.9% respectively and overall accuracy of 84.20%.

As the Uncertainty principle suggests that the window size has the significant influence on how the information is presented in spectrogram, we create another 12 experiments of 3s window size on selected samples, the outcomes are shown in Table 2.

To compare the outcomes of 1s, 2s and 3s window size of selected samples, Table 3 has been created. As the outcomes indicate classifies on window size 3s perform similarly as 2s and 1s window size on same segment size and overlap.

To check the theory raised by Kostiantyn Pylypenko that smaller sized inputs lead to better classification performance,[5] we compared 5 sets of image with different sizes (256×256 pixels and 192×192 pixels) and present the comparison in Table 4. In general, the classifiers perform on smaller sized images give better outcomes.

Segment Size	Window Size	Overlap	SR Sens. (%)	VF Sens. (%)	VT Sens. (%)	Accuracy
2 seconds	1 second	0.00	96.10	83.50	66.30	82.00
		0.25	98.30	85.10	64.30	82.50
		0.50	98.50	78.10	72.50	83.10
	2 seconds	0.00	98.30	74.80	78.80	84.00
		0.25	97.60	78.10	76.90	84.20
		0.50	98.00	76.50	76.80	83.80
3 seconds	1 second	0.00	98.60	70.60	70.20	79.80
		0.25	98.60	68.80	71.80	79.70
		0.50	99.20	69.60	73.20	80.70
	2 seconds	0.00	98.40	73.80	66.80	79.70
		0.25	98.20	72.40	70.80	80.50
		0.50	98.60	78.60	67.40	81.50
4 seconds	1 second	0.00	95.50	56.30	82.40	78.10
		0.25	99.20	72.50	70.10	80.60
		0.50	98.90	63.20	80.00	80.70
	2 seconds	0.00	97.10	83.70	66.40	82.40
		0.25	97.90	77.30	69.30	81.50
		0.50	97.90	78.70	72.50	83.00
5 seconds	1 second	0.00	97.30	72.30	71.00	80.20
		0.25	98.00	72.70	68.70	79.80
		0.50	95.40	67.30	71.70	78.20
	2 seconds	0.00	99.00	79.30	63.00	80.40
		0.25	97.00	66.00	76.70	79.90
		0.50	97.70	74.00	66.70	79.40
6 seconds	1 second	0.00	99.60	74.40	56.80	76.90
		0.25	95.20	86.80	53.20	78.40
		0.50	99.20	70.40	70.00	79.90
	2 seconds	0.00	99.60	78.40	66.00	81.30
		0.25	99.60	72.00	78.40	82.90
		0.50	100.00	84.00	67.20	83.70
7 seconds	1 second	0.00	96.30	72.00	65.40	77.90
		0.25	98.10	80.80	59.30	79.40
		0.50	99.10	72.00	65.40	78.80
	2 seconds	0.00	99.10	80.80	62.10	80.70
		0.25	98.60	77.10	66.40	80.70
		0.50	99.10	86.40	59.30	81.60
8 seconds	1 second	0.00	98.90	73.30	62.60	78.30
		0.25	97.90	59.90	78.60	78.80
		0.50	99.50	66.30	71.10	79.00
	2 seconds	0.00	99.10	80.80	62.10	80.70
		0.25	99.50	66.80	73.30	79.90
		0.50	98.40	68.40	72.70	79.90
9 seconds	1 second	0.00	96.80	73.10	65.00	79.00
		0.25	98.80	91.60	49.40	79.90
		0.50	95.20	66.90	83.10	81.70
	2 seconds	0.00	96.80	75.10	65.00	79.00
		0.25	97.60	75.90	74.10	82.50
		0.50	98.20	68.70	77.10	81.30
10 seconds	1 second	0.00	98.00	84.70	61.30	81.30
		0.25	98.00	92.00	48.70	77.60
		0.50	97.30	70.00	73.30	80.20
	2 seconds	0.00	97.30	62.00	88.00	82.40
		0.25	97.30	82.70	62.70	80.90
		0.50	97.30	73.30	74.70	81.80

Table 1: Experiment Result of 192×192 size image

Segment Size	Window Size	Overlap (%)	SR Sens. (%)	VF Sens. (%)	VT Sens. (%)	Accuracy
3 seconds	3 seconds	0.00	100.00	73.40	75.40	82.90
		0.25	100.00	70.40	75.80	82.10
		0.50	99.80	75.20	72.00	82.30
4 seconds	3 seconds	0.00	98.90	76.50	74.40	83.30
		0.25	98.70	73.30	76.80	82.90
		0.50	98.70	70.70	77.90	82.40
5 seconds	3 seconds	0.00	97.00	68.70	73.00	79.60
		0.25	98.00	68.00	73.70	79.90
		0.50	98.30	78.70	66.70	81.20
6 seconds	3 seconds	0.00	99.20	78.40	66.40	81.30
		0.25	99.60	79.60	67.60	82.30
		0.50	99.60	73.60	78.00	83.70

Table 2: Experiment Result of the selected samples with 3 seconds window size

Segment Size	Overlap (%)	Window Size	SR Sens. (%)	VF Sens. (%)	VT Sens. (%)	Accuracy
3 seconds	0.00	1 second	98.60	70.60	70.20	79.80
		2 seconds	98.40	73.80	66.80	79.70
		3 seconds	100.00	73.40	75.40	82.90
	0.25	1 second	98.60	68.80	71.80	79.70
		2 seconds	98.20	72.40	70.80	80.50
		3 seconds	100.00	70.40	75.80	82.10
	0.50	1 second	99.20	69.60	73.20	80.70
		2 seconds	98.60	78.60	67.40	81.50
		3 seconds	99.80	75.20	72.00	82.30
4 seconds	0.00	1 second	95.50	56.30	82.40	78.10
		2 seconds	97.10	83.70	66.40	82.40
		3 seconds	98.90	76.50	74.40	83.30
	0.25	1 second	99.20	72.50	70.10	80.60
		2 seconds	97.90	77.30	69.30	81.50
		3 seconds	98.70	73.30	76.80	82.90
	0.50	1 second	98.90	63.20	80.00	80.70
		2 seconds	97.90	78.70	72.50	83.00
		3 seconds	98.70	70.70	77.90	82.40
5 seconds	0.00	1 second	97.30	72.30	71.00	80.20
		2 seconds	99.00	79.30	63.00	80.40
		3 seconds	97.00	68.70	73.00	79.60
	0.25	1 second	98.00	72.70	68.70	79.80
		2 seconds	97.00	66.00	76.70	79.90
		3 seconds	98.00	68.00	73.70	79.90
	0.50	1 second	95.40	67.30	71.70	78.20
		2 seconds	97.70	74.00	66.70	79.40
		3 seconds	98.30	78.70	66.70	81.20
6 seconds	0.00	1 second	99.60	74.40	56.80	76.90
		2 seconds	99.60	78.40	66.00	81.30
		3 seconds	99.20	78.40	66.40	81.30
	0.25	1 second	97.60	64.80	73.60	78.70
		2 seconds	99.60	72.00	78.40	82.90
		3 seconds	99.60	79.60	67.60	82.30
	0.50	1 second	99.20	70.40	70.00	79.90
		2 seconds	98.40	74.00	71.20	81.60
		3 seconds	99.60	73.60	78.00	83.70

Table 3: Comparison between different window sizes of 2-6 seconds samples

Segment Size	Window Size	Figure Dimension	Overlap (%)	SR Sens. (%)	VF Sens. (%)	VT Sens. (%)	Accuracy
2 seconds	2 seconds	192 x 192	0.00	98.30	74.80	78.80	84.00
		256 x 256	0.00	99.20	73.70	76.90	83.30
3 seconds	2 seconds	192 x 192	0.25	98.60	68.80	71.80	79.70
		256 x 256	0.25	98.60	70.40	68.60	79.20
4 seconds	2 seconds	192 x 192	0.50	97.90	78.70	72.50	83.00
		256 x 256	0.50	96.40	73.30	77.30	83.00
5 seconds	1 seconds	192 x 192	0.00	97.30	72.30	71.00	80.20
		256 x 256	0.00	97.30	45.30	83.70	75.40
6 seconds	2 seconds	192 x 192	0.25	99.60	72.00	78.40	82.90
		256 x 256	0.25	99.20	66.80	82.40	82.80

Table 4: Comparison between different dimension spectrograms from selected 2-6 seconds samples

6 Conclusion

6.1 Project Summary

This project proposes to solve the classification problem of Sinus Rhythms, Ventricular Tachycardia and Ventricular Fibrillation by Convolutional Neural Network. To use the original ECG signal data with CNN, which requires input data to be 2-dimensional as images, the Short Time Fourier Transform is applied to transform signals into spectrograms. Another important trait of the original data is that the data is imbalanced which means SR has much more samples than VT and VF. To address this problem, another signal processing method named subsampling has been applied, and this means that we picked 6000s samples from each class to manually make the data balance. The processed images are then fed into CNN we created to acquire confusion matrix which shows the sensitivity of each class and overall accuracy. And we have successfully achieve the objective of obtaining over 90% sensitivity on SR and 50% on VT and VF.

6.2 Chapters Overview

Chapter 1 presents the introduction of cardiovascular disease as well as the state-of-art which tries to classify the SR,VT and VF. It also gives the introduction of this project based on the approach used to solve same problem and the goal.
Chapter 2 defines a number of technical backgrounds which could help the reader to understand better of this project in terms of necessity and methodology.
Chapter 3 describes the important decisions about how the encountered problems are solved and the important decisions are made.
Chapter 4 provides the detailed explanation of how each design are actually implemented based on the chosen programming language and toolkit. In this chapter, the critical functions, either provided by toolkit or written by us, used to solve a major problem has been listed and explained as well as the parameters.
Chapter 5 gives the information about the procedures of how the experiments are conducted. More importantly, all the results,including sensitivity of each class and overall

accuracy, obtained in each experiment are listed in tables as well as the analysis from the obtained results.
Chapter 6 summarizes the work had been done and provides the directions of how this project could be improved and the new approach worth trying.

6.3 Achievement

Obtaining sensitivities of detection for SR with 97.6% and VF and VT with 78.1% and 76.9% respectively, this project is deemed to successfully achieve the objective which is to obtain over 90% sensitivity of SR and 50% for both VF and VT. Also, this project compares different window size on same samples, and give the conclusion that the classifiers applied with selected window sizes (1s,2s and 3s) do not behave drastically different. Last but not least, comparison between different input image sizes are made on this project, from the results we can draw the conclusion that the small-sized images bring good effect the training performance.

6.4 Future Works and Extension

This section lists and explains the potential future works which can be undertaken to improve the performance or to establishing a new approach that can be used to deal with this partuclar classification problem.

6.4.1 Fine-tuning Spectrogram Parameters

Due to the time and resource limitations, the segment sizes we used for experimenting are limited from 2s to 10s, and the window sizes are set to be only 1s, 2s and 3s on some selected samples. As the experiment result section indicates, the performance varies on different segment size. Also, according to the uncertainty principle, the window size affects both frequency resolution and time resolution. Hence, further experiments can be conducted with more segment sizes and window sizes to try to obtain the better result.

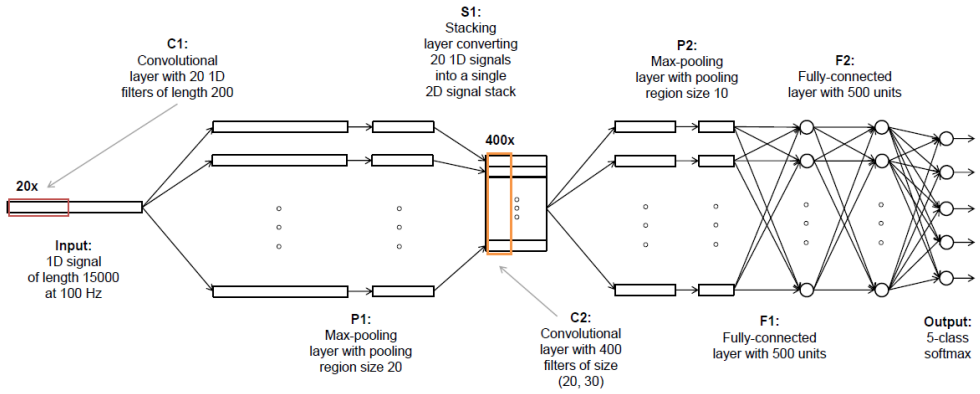


Figure 12: CNN architecture for 1D data

6.4.2 Different Window Function

As stated in development craft image section, the hamming window function is what we applied for generating all the images, the future experiment can try to use different window function to generate the spectrograms in order to pursue the better result.

6.4.3 The Application of 1-dimensional Data for CNN

Gaining remarkable results in image classification, CNN are normally used to deal with 2D data such as images and this is the reason why many researches using CNN with signals convert the 1d data to be 2d image such as spectrogram. However, there is a small but interesting trend which is using CNN directly on raw 1D data. This potentially could be the inspiration of the new ways to conduct the future work on this subject. There is one existed study raised by Orestis Tsinalis et. al. The study proposes the application of the raw EEG signal as input for CNN as well as shows that the overall the accuracy is more than 70%. Figure 12 shows the architecture of the CNN used in this study which can be the guideline of future work of this project due to the similarity of data set and classifier.[6]

References

[1] Y. Alwan, Z. Cvetkovic, and M. J. Curtis, "Classification of human ventricular arrhythmia in high dimensional representation spaces," *arXiv preprint arXiv:1312.5354*, 2014.

[2] Y. Alwan, Z. Cvetkovic, and M. J. Curtis, "Methods for improved discrimination between ventricular fibrillation and tachycardia," 2011.

[3] L. Yuan and J. Cao, "Patients eeg data analysis via spectrogram image with a convolution neural network," *Intelligent Decision Technologies 2017 Smart Innovation, Systems and Technologies*, p. 1321, 2017.

[4] S. Dieleman and B. Schrauwen, "End-to-end learning for music audio," 2014.

[5] K. Pylypenko, "Right whale detection using artificial neural network and principal component analysis," 2015.

[6] O. Tsinalis, P. M. Matthews, Y. Guo, and S. Zafeiriou, "Automatic sleep stage scoring with single-channel eeg using convolutional neural networks," 2015.

[7] "Ventricular tachycardia (vtach) (video) — khan academy." https://www.khanacademy.org/test-prep/nclex-rn/rn-cardiovascular-diseases/rn-dysrhythmia-and-tachycardia/v/ventricular-tachycardias. (Accessed on 08/24/2017).

[8] "What is ventricle fibrillation (vfib)? — khan academy." https://www.khanacademy.org/test-prep/nclex-rn/rn-cardiovascular-diseases/rn-dysrhythmia-and-tachycardia/v/ventricular-fibrillation. (Accessed on 08/24/2017).

[9] J. O. S. III, *Mathematics of the Discrete Fourier Transform (DFT)*. 8 2002.

[10] J. O. S. III, *Spectral Audio Signal Processing*.

[11] "A beginner's guide to understanding convolutional neural networks adit deshpande cs undergrad at ucla ('19)." https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/. (Accessed on 08/23/2017).

[12] "Convolutional neural networks (cnns): An illustrated explanation - xrdsxrds." http://xrds.acm.org/blog/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/. (Accessed on 08/23/2017).

[13] "Cs231n convolutional neural networks for visual recognition." http://cs231n.github.io/convolutional-networks/. (Accessed on 08/23/2017).

[14] "Simple guide to confusion matrix terminology." http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/. (Accessed on 08/23/2017).

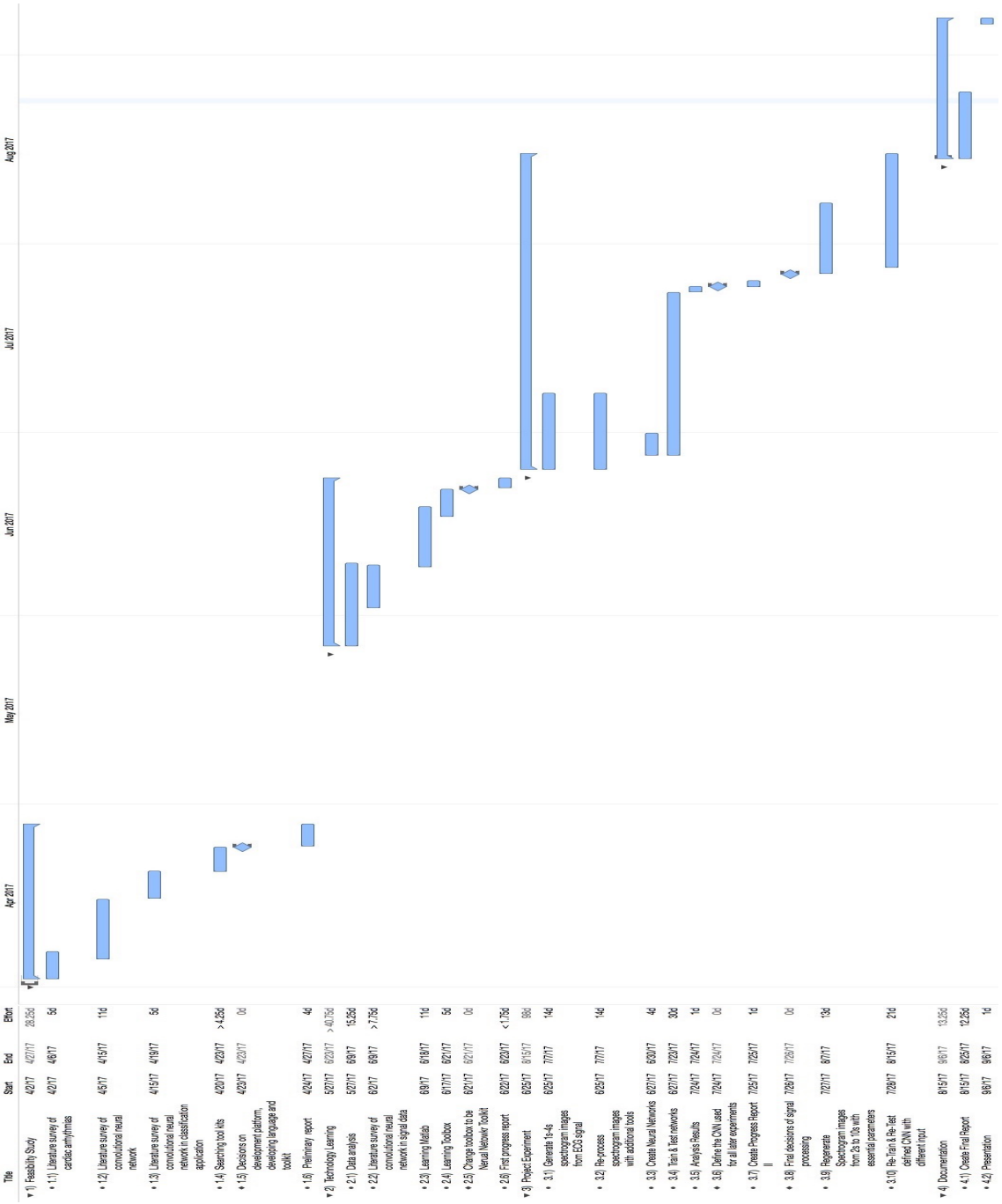
[15] D. Houcque, "Introduction to matlab for engineering students," 8 2005.

[16] "Introduction to matlab." https://www.math.utah.edu/~wright/misc/matlab/matlabintro.html. (Accessed on 08/23/2017).

[17] "Plot classification confusion matrix - matlab plotconfusion - mathworks united kingdom." https://uk.mathworks.com/help/nnet/ref/plotconfusion.html. (Accessed on 08/23/2017).

A Appendix

A.1 Project Gantt Chart



A.2 Preprocessing Example Code

A.2.1 Preprocessing Example Code for 192 x 192 image data sets

Note: XXX is replaced with the segment size during the implementation.

On 1 second window with no overlapping on window

```
load('RHYTHMS.mat');
segment_size = XXX;
window_size = 100;
limit = 600000;
expected_capped_input_no = limit/segment_size;
sr_size = size(SR);
sr_size = sr_size(2);
sr_cell_size = zeros(sr_size,1);
for i=1:sr_size
sr_cell_size(i) = size(SR{i},1);
end
sr_sum = sum(sr_cell_size);
expected_sr_input_no = int16(sr_sum/segment_size);
sr_imported_count = 0;
sr_imported_segment = 0;
index = 1;
name = 1;
start_pos = 1;
end_pos = segment_size;
while sr_imported_count < expected_capped_input_no
    if end_pos < size(SR{1,index},1)
            h = figure('Visible','off')
            spectrogram(SR{1,index}(start_pos:end_pos,1),
                hamming(window_size),0)
```

```
24              xlabel('');
25              ylabel('');
26              set(findobj('type','axes'),'fontsize',5);
27              set(h, 'PaperUnits' , 'points');
28              set(h, 'PaperPosition' , [0 0 92 92]);
29              set(h, 'Position' , [0 0 92 92]);
30              tmpstr = sprintf('Output/SR/%d.png',name);
31              ch=findall(h,'tag','Colorbar');
32              delete(ch);
33              saveas(h,tmpstr);
34          sr_imported_segment = sr_imported_segment+segment_size;
35          sr_imported_count = sr_imported_count + 1;
36          name = name + 1;
37          start_pos = end_pos;
38          end_pos = end_pos + segment_size;
39      else
40          index = index + 1;
41          sr_imported_segment = 0;
42          start_pos = 1;
43          end_pos = segment_size;
44      end
45      close all
46 end
47 vf_size = size(VF);
48 vf_size = vf_size(2);
49 vf_cell_size = zeros(vf_size,1);
50 for i=1:vf_size
51 vf_cell_size(i) = size(VF{i},1);
52 end
```

```
53 vf_sum = sum(vf_cell_size);
54 expected_vf_input_no = int16(vf_sum/segment_size);
55 vf_imported_count = 0;
56 vf_imported_segment = 0;
57 index = 1;
58 name = 1;
59 start_pos = 1;
60 end_pos = segment_size;
61 while vf_imported_count < expected_capped_input_no
62     if end_pos < size(VF{1,index},1)
63             h = figure('Visible','off')
64             spectrogram(VF{1,index}(start_pos:end_pos,1),
                   hamming(window_size),0)
65             xlabel('');
66             ylabel('');
67             set(findobj('type','axes'),'fontsize',5);
68             set(h, 'PaperUnits' , 'points');
69             set(h, 'PaperPosition' , [0 0 92 92]);
70             set(h, 'Position' , [0 0 92 92]);
71             tmpstr = sprintf('Output/VF/%d.png',name);
72             ch=findall(h,'tag','Colorbar');
73             delete(ch);
74             saveas(h,tmpstr);
75         vf_imported_segment = vf_imported_segment+segment_size;
76         vf_imported_count = vf_imported_count + 1;
77         name = name + 1;
78         start_pos = end_pos;
79         end_pos = end_pos + segment_size;
80     else
```

```
81          index = index + 1;
82          vf_imported_segment = 0;
83          start_pos = 1;
84          end_pos = segment_size;
85      end
86      close all
87  end
88  vt_size = size(VT);
89  vt_size = vt_size(2);
90  vt_cell_size = zeros(vt_size,1);
91  for i=1:vt_size
92  vt_cell_size(i) = size(VT{i},1);
93  end
94  vt_sum = sum(vt_cell_size);
95  expected_vt_input_no = int16(vt_sum/segment_size);
96  vt_imported_count = 0;
97  vt_imported_segment = 0;
98  index = 1;
99  name = 1;
100 start_pos = 1;
101 end_pos = segment_size;
102 while vt_imported_count < expected_capped_input_no
103     if end_pos < size(VT{1,index},1)
104         h = figure('Visible','off')
105             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),0)
106             xlabel('');
107             ylabel('');
108             set(findobj('type','axes'),'fontsize',5);
```

```
109             set(h, 'PaperUnits' , 'points');
110             set(h, 'PaperPosition' , [0 0 92 92]);
111             set(h, 'Position' , [0 0 92 92]);
112             tmpstr = sprintf('Output/VT/%d.png',name);
113             ch=findall(h,'tag','Colorbar');
114             delete(ch);
115             saveas(h,tmpstr);
116         vt_imported_segment = vt_imported_segment+segment_size;
117         vt_imported_count = vt_imported_count + 1;
118         name = name + 1;
119         start_pos = end_pos;
120         end_pos = end_pos + segment_size;
121     else
122         index = index + 1;
123         vt_imported_segment = 0;
124         start_pos = 1;
125         end_pos = segment_size;
126     end
127     close all
128 end
```

On 1 second window with 25% overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 100;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
```

```
    train_ratio/100));
sr_size = size(SR);
sr_size = sr_size(2);
sr_cell_size = zeros(sr_size,1);
for i=1:sr_size
sr_cell_size(i) = size(SR{i},1);
end
sr_sum = sum(sr_cell_size);
expected_sr_input_no = int16(sr_sum/segment_size);
expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
    /100));
sr_imported_count = 0;
sr_imported_segment = 0;
index = 1;
name = 1;
start_pos = 1;
end_pos = segment_size;
while sr_imported_count < expected_capped_input_no
    if end_pos < size(SR{1,index},1)
            h = figure('Visible','off')
            spectrogram(SR{1,index}(start_pos:end_pos,1),
               hamming(window_size),ceil(window_size)/4)
            xlabel('');
            ylabel('');
            set(findobj('type','axes'),'fontsize',5);
            set(h, 'PaperUnits', 'points');
            set(h, 'PaperPosition', [0 0 92 92]);
            set(h, 'Position', [0 0 92 92]);
            tmpstr = sprintf('Output/SR/%d.png',name);
```

```
34              ch=findall(h,'tag','Colorbar');
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
       /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
```

```
name = 1;
start_pos = 1;
end_pos = segment_size;
while vf_imported_count < expected_capped_input_no
    if end_pos < size(VF{1,index},1)
            h = figure('Visible','off')
            spectrogram(VF{1,index}(start_pos:end_pos,1),
               hamming(window_size),ceil(window_size)/4)
            xlabel('');
            ylabel('');
            set(findobj('type','axes'),'fontsize',5);
            set(h, 'PaperUnits' , 'points');
            set(h, 'PaperPosition' , [0 0 92 92]);
            set(h, 'Position' , [0 0 92 92]);
            tmpstr = sprintf('Output/VF/%d.png',name);
            ch=findall(h,'tag','Colorbar');
            delete(ch);
            saveas(h,tmpstr);
        vf_imported_segment = vf_imported_segment+segment_size;
        vf_imported_count = vf_imported_count + 1;
        name = name + 1;
        start_pos = end_pos;
        end_pos = end_pos + segment_size;
    else
        index = index + 1;
        vf_imported_segment = 0;
        start_pos = 1;
        end_pos = segment_size;
    end
```

```
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                  hamming(window_size),ceil(window_size)/4)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits', 'points');
115             set(h, 'PaperPosition', [0 0 92 92]);
116             set(h, 'Position', [0 0 92 92]);
```

```
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 1 second window with 50% overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 100;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
    train_ratio/100));
8 sr_size = size(SR);
9 sr_size = sr_size(2);
```

```
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
     /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                 hamming(window_size),ceil(window_size/2))
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits' , 'points');
31             set(h, 'PaperPosition' , [0 0 92 92]);
32             set(h, 'Position' , [0 0 92 92]);
33             tmpstr = sprintf('22-08-17/6-2-0.25x256/SR/%d.png',
                 name);
34             ch=findall(h,'tag','Colorbar');
35             delete(ch);
```

```
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
       /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
```

```
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                   hamming(window_size),ceil(window_size/2))
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 92 92]);
74             set(h, 'Position' , [0 0 92 92]);
75             tmpstr = sprintf('22-08-17/6-2-0.25x256/VF/%d.png',
                   name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
79         vf_imported_segment = vf_imported_segment+segment_size;
80         vf_imported_count = vf_imported_count + 1;
81         name = name + 1;
82         start_pos = end_pos;
83         end_pos = end_pos + segment_size;
84     else
85         index = index + 1;
86         vf_imported_segment = 0;
87         start_pos = 1;
88         end_pos = segment_size;
89     end
90     close all
```

```
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                  hamming(window_size),ceil(window_size/2))
111             xabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits', 'points');
115             set(h, 'PaperPosition', [0 0 92 92]);
116             set(h, 'Position', [0 0 92 92]);
117             tmpstr = sprintf('22-08-17/6-2-0.25x256/VT/%d.png',
```

```
                    name);
118                 ch=findall(h,'tag','Colorbar');
119                 delete(ch);
120                 saveas(h,tmpstr);
121             vt_imported_segment = vt_imported_segment+segment_size;
122             vt_imported_count = vt_imported_count + 1;
123             name = name + 1;
124             start_pos = end_pos;
125             end_pos = end_pos + segment_size;
126         else
127             index = index + 1;
128             vt_imported_segment = 0;
129             start_pos = 1;
130             end_pos = segment_size;
131         end
132         close all
133 end
```

On 2 seconds window with no overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 200;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
      train_ratio/100));
8 sr_size = size(SR);
9 sr_size = sr_size(2);
```

```
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
     /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                  hamming(window_size),0)
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits' , 'points');
31             set(h, 'PaperPosition' , [0 0 92 92]);
32             set(h, 'Position' , [0 0 92 92]);
33             tmpstr = sprintf('Output/SR/%d.png',name);
34             ch=findall(h,'tag','Colorbar');
35             delete(ch);
36             saveas(h,tmpstr);
```

```
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
       /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
```

```
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                   hamming(window_size),0)
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 92 92]);
74             set(h, 'Position' , [0 0 92 92]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
79         vf_imported_segment = vf_imported_segment+segment_size;
80         vf_imported_count = vf_imported_count + 1;
81         name = name + 1;
82         start_pos = end_pos;
83         end_pos = end_pos + segment_size;
84     else
85         index = index + 1;
86         vf_imported_segment = 0;
87         start_pos = 1;
88         end_pos = segment_size;
89     end
90     close all
91 end
92 vt_size = size(VT);
```

```
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                  hamming(window_size),0)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits', 'points');
115             set(h, 'PaperPosition', [0 0 92 92]);
116             set(h, 'Position', [0 0 92 92]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
```

```
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 2 seconds window with 25% overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 200;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
      train_ratio/100));
8 sr_size = size(SR);
9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
```

```
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
       /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                   hamming(window_size),ceil(window_size)/4)
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits', 'points');
31             set(h, 'PaperPosition', [0 0 92 92]);
32             set(h, 'Position', [0 0 92 92]);
33             tmpstr = sprintf('Output/SR/%d.png',name);
34             ch=findall(h,'tag','Colorbar');
35             delete(ch);
36             saveas(h,tmpstr);
37         sr_imported_segment = sr_imported_segment+segment_size;
38         sr_imported_count = sr_imported_count + 1;
39         name = name + 1;
```

```
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
       /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66      if end_pos < size(VF{1,index},1)
67              h = figure('Visible','off')
```

```
68              spectrogram(VF{1,index}(start_pos:end_pos,1),
                    hamming(window_size),ceil(window_size)/4)
69              xlabel('');
70              ylabel('');
71              set(findobj('type','axes'),'fontsize',5);
72              set(h, 'PaperUnits' , 'points');
73              set(h, 'PaperPosition' , [0 0 92 92]);
74              set(h, 'Position' , [0 0 92 92]);
75              tmpstr = sprintf('Output/VF/%d.png',name);
76              ch=findall(h,'tag','Colorbar');
77              delete(ch);
78              saveas(h,tmpstr);
79          vf_imported_segment = vf_imported_segment+segment_size;
80          vf_imported_count = vf_imported_count + 1;
81          name = name + 1;
82          start_pos = end_pos;
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
```

```
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                  hamming(window_size),ceil(window_size)/4)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 92 92]);
116             set(h, 'Position' , [0 0 92 92]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
```

```
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 2 seconds window with 50% overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 200;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
    train_ratio/100));
8 sr_size = size(SR);
9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
```

```
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
    /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                   hamming(window_size),ceil(window_size/2))
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits' , 'points');
31             set(h, 'PaperPosition' , [0 0 92 92]);
32             set(h, 'Position' , [0 0 92 92]);
33             tmpstr = sprintf('Output/SR/%d.png',name);
34             ch=findall(h,'tag','Colorbar');
35             delete(ch);
36             saveas(h,tmpstr);
37         sr_imported_segment = sr_imported_segment+segment_size;
38         sr_imported_count = sr_imported_count + 1;
39         name = name + 1;
40         start_pos = end_pos;
41         end_pos = end_pos + segment_size;
42     else
```

```
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
       /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66      if end_pos < size(VF{1,index},1)
67              h = figure('Visible','off')
68              spectrogram(VF{1,index}(start_pos:end_pos,1),
                    hamming(window_size),ceil(window_size/2))
69              xlabel('');
```

```
70              ylabel('');
71              set(findobj('type','axes'),'fontsize',5);
72              set(h, 'PaperUnits' , 'points');
73              set(h, 'PaperPosition' , [0 0 92 92]);
74              set(h, 'Position' , [0 0 92 92]);
75              tmpstr = sprintf('Output/VF/%d.png',name);
76              ch=findall(h,'tag','Colorbar');
77              delete(ch);
78              saveas(h,tmpstr);
79          vf_imported_segment = vf_imported_segment+segment_size;
80          vf_imported_count = vf_imported_count + 1;
81          name = name + 1;
82          start_pos = end_pos;
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
```

```
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
    /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                hamming(window_size),ceil(window_size/2))
111             xabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 92 92]);
116             set(h, 'Position' , [0 0 92 92]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
```

```
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 3 seconds window with no overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 300;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
      train_ratio/100));
8 sr_size = size(SR);
9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
      /100));
17 sr_imported_count = 0;
```

```
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                   hamming(window_size),0)
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits' , 'points');
31             set(h, 'PaperPosition' , [0 0 92 92]);
32             set(h, 'Position' , [0 0 92 92]);
33             tmpstr = sprintf('Output/SR/%d.png',name);
34             ch=findall(h,'tag','Colorbar');
35             delete(ch);
36             saveas(h,tmpstr);
37         sr_imported_segment = sr_imported_segment+segment_size;
38         sr_imported_count = sr_imported_count + 1;
39         name = name + 1;
40         start_pos = end_pos;
41         end_pos = end_pos + segment_size;
42     else
43         index = index + 1;
44         sr_imported_segment = 0;
45         start_pos = 1;
```

```
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
      /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66      if end_pos < size(VF{1,index},1)
67              h = figure('Visible','off')
68              spectrogram(VF{1,index}(start_pos:end_pos,1),
                    hamming(window_size),0)
69              xlabel('');
70              ylabel('');
71              set(findobj('type','axes'),'fontsize',5);
72              set(h, 'PaperUnits' , 'points');
```

```
73              set(h, 'PaperPosition' , [0 0 92 92]);
74              set(h, 'Position' , [0 0 92 92]);
75              tmpstr = sprintf('Output/VF/%d.png',name);
76              ch=findall(h,'tag','Colorbar');
77              delete(ch);
78              saveas(h,tmpstr);
79          vf_imported_segment = vf_imported_segment+segment_size;
80          vf_imported_count = vf_imported_count + 1;
81          name = name + 1;
82          start_pos = end_pos;
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
```

```
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),0)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 92 92]);
116             set(h, 'Position' , [0 0 92 92]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
```

```
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 3 seconds window with 25% overlapping on window

```
 1 load('RHYTHMS.mat');
 2 segment_size = XXX;
 3 window_size = 300;
 4 limit = 600000;
 5 train_ratio = 75;
 6 expected_capped_input_no = limit/segment_size;
 7 expected_capped_train_no = int16(expected_capped_input_no*(
     train_ratio/100));
 8 sr_size = size(SR);
 9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
     /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
```

```
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                   hamming(window_size),ceil(window_size)/4)
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits' , 'points');
31             set(h, 'PaperPosition' , [0 0 92 92]);
32             set(h, 'Position' , [0 0 92 92]);
33             tmpstr = sprintf('Output/SR/%d.png',name);
34             ch=findall(h,'tag','Colorbar');
35             delete(ch);
36             saveas(h,tmpstr);
37         sr_imported_segment = sr_imported_segment+segment_size;
38         sr_imported_count = sr_imported_count + 1;
39         name = name + 1;
40         start_pos = end_pos;
41         end_pos = end_pos + segment_size;
42     else
43         index = index + 1;
44         sr_imported_segment = 0;
45         start_pos = 1;
46         end_pos = segment_size;
47     end
48     close all
```

```
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
     /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                 hamming(window_size),ceil(window_size)/4)
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits', 'points');
73             set(h, 'PaperPosition', [0 0 92 92]);
74             set(h, 'Position', [0 0 92 92]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
```

```
76              ch=findall(h,'tag','Colorbar');
77              delete(ch);
78              saveas(h,tmpstr);
79          vf_imported_segment = vf_imported_segment+segment_size;
80          vf_imported_count = vf_imported_count + 1;
81          name = name + 1;
82          start_pos = end_pos;
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
```

```
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),ceil(window_size)/4)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 92 92]);
116             set(h, 'Position' , [0 0 92 92]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
```

```
132     close all
133 end
```

On 3 seconds window with 50% overlapping on window

```
 1 load('RHYTHMS.mat');
 2 segment_size = XXX;
 3 window_size = 300;
 4 limit = 600000;
 5 train_ratio = 75;
 6 expected_capped_input_no = limit/segment_size;
 7 expected_capped_train_no = int16(expected_capped_input_no*(
       train_ratio/100));
 8 sr_size = size(SR);
 9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
       /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
```

```
24      if end_pos < size(SR{1,index},1)
25              h = figure('Visible','off')
26              spectrogram(SR{1,index}(start_pos:end_pos,1),
                    hamming(window_size),ceil(window_size/2))
27              xlabel('');
28              ylabel('');
29              set(findobj('type','axes'),'fontsize',5);
30              set(h, 'PaperUnits' , 'points');
31              set(h, 'PaperPosition' , [0 0 92 92]);
32              set(h, 'Position' , [0 0 92 92]);
33              tmpstr = sprintf('Output/SR/%d.png',name);
34              ch=findall(h,'tag','Colorbar');
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49  end
50  vf_size = size(VF);
51  vf_size = vf_size(2);
```

```
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
     /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                 hamming(window_size),ceil(window_size/2))
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 92 92]);
74             set(h, 'Position' , [0 0 92 92]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
```

```
79          vf_imported_segment = vf_imported_segment+segment_size;
80          vf_imported_count = vf_imported_count + 1;
81          name = name + 1;
82          start_pos = end_pos;
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
```

```
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),ceil(window_size/2))
111             xabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 92 92]);
116             set(h, 'Position' , [0 0 92 92]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

A.2.2 Preprocessing Example Code for 256 x 256 image data sets

Note: XXX is replaced with the segment size during the implementation.

On 1 second window with no overlapping on window

```
load('RHYTHMS.mat');
segment_size = XXX;
window_size = 100;
limit = 600000;
train_ratio = 75;
expected_capped_input_no = limit/segment_size;
expected_capped_train_no = int16(expected_capped_input_no*(
    train_ratio/100));
sr_size = size(SR);
sr_size = sr_size(2);
sr_cell_size = zeros(sr_size,1);
for i=1:sr_size
sr_cell_size(i) = size(SR{i},1);
end
sr_sum = sum(sr_cell_size);
expected_sr_input_no = int16(sr_sum/segment_size);
expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
    /100));
sr_imported_count = 0;
sr_imported_segment = 0;
index = 1;
name = 1;
start_pos = 1;
end_pos = segment_size;
while sr_imported_count < expected_capped_input_no
```

```
24      if end_pos < size(SR{1,index},1)
25              h = figure('Visible','off')
26              spectrogram(SR{1,index}(start_pos:end_pos,1),
                    hamming(window_size),0)
27              xlabel('');
28              ylabel('');
29              set(findobj('type','axes'),'fontsize',5);
30              set(h, 'PaperUnits' , 'points');
31              set(h, 'PaperPosition' , [0 0 123 123]);
32              set(h, 'Position' , [0 0 123 123]);
33              tmpstr = sprintf('Output/SR/%d.png',name);
34              ch=findall(h,'tag','Colorbar');
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49  end
50  vf_size = size(VF);
51  vf_size = vf_size(2);
```

```
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
     /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                 hamming(window_size),0)
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 123 123]);
74             set(h, 'Position' , [0 0 123 123]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
```

```
79          vf_imported_segment = vf_imported_segment+segment_size;
80          vf_imported_count = vf_imported_count + 1;
81          name = name + 1;
82          start_pos = end_pos;
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
```

```
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),0)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 123 123]);
116             set(h, 'Position' , [0 0 123 123]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 1 second window with 25% overlapping on window

```
load('RHYTHMS.mat');
segment_size = XXX;
window_size = 100;
limit = 600000;
train_ratio = 75;
expected_capped_input_no = limit/segment_size;
expected_capped_train_no = int16(expected_capped_input_no*(
    train_ratio/100));
sr_size = size(SR);
sr_size = sr_size(2);
sr_cell_size = zeros(sr_size,1);
for i=1:sr_size
sr_cell_size(i) = size(SR{i},1);
end
sr_sum = sum(sr_cell_size);
expected_sr_input_no = int16(sr_sum/segment_size);
expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
    /100));
sr_imported_count = 0;
sr_imported_segment = 0;
index = 1;
name = 1;
start_pos = 1;
end_pos = segment_size;
while sr_imported_count < expected_capped_input_no
    if end_pos < size(SR{1,index},1)
            h = figure('Visible','off')
            spectrogram(SR{1,index}(start_pos:end_pos,1),
```

```
                hamming(window_size),ceil(window_size)/4)
27              xlabel('');
28              ylabel('');
29              set(findobj('type','axes'),'fontsize',5);
30              set(h, 'PaperUnits' , 'points');
31              set(h, 'PaperPosition' , [0 0 123 123]);
32              set(h, 'Position' , [0 0 123 123]);
33              tmpstr = sprintf('Output/SR/%d.png',name);
34              ch=findall(h,'tag','Colorbar');
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
```

```
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
     /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                  hamming(window_size),ceil(window_size)/4)
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 123 123]);
74             set(h, 'Position' , [0 0 123 123]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
79         vf_imported_segment = vf_imported_segment+segment_size;
80         vf_imported_count = vf_imported_count + 1;
81         name = name + 1;
```

```
82          start_pos = end_pos;
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
```

```
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),ceil(window_size)/4)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 123 123]);
116             set(h, 'Position' , [0 0 123 123]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 1 second window with 50% overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
```

```
window_size = 100;
limit = 600000;
train_ratio = 75;
expected_capped_input_no = limit/segment_size;
expected_capped_train_no = int16(expected_capped_input_no*(
    train_ratio/100));
sr_size = size(SR);
sr_size = sr_size(2);
sr_cell_size = zeros(sr_size,1);
for i=1:sr_size
sr_cell_size(i) = size(SR{i},1);
end
sr_sum = sum(sr_cell_size);
expected_sr_input_no = int16(sr_sum/segment_size);
expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
    /100));
sr_imported_count = 0;
sr_imported_segment = 0;
index = 1;
name = 1;
start_pos = 1;
end_pos = segment_size;
while sr_imported_count < expected_capped_input_no
    if end_pos < size(SR{1,index},1)
            h = figure('Visible','off')
            spectrogram(SR{1,index}(start_pos:end_pos,1),
                hamming(window_size),ceil(window_size/2))
            xlabel('');
            ylabel('');
```

```
29              set(findobj('type','axes'),'fontsize',5);
30              set(h, 'PaperUnits' , 'points');
31              set(h, 'PaperPosition' , [0 0 123 123]);
32              set(h, 'Position' , [0 0 123 123]);
33              tmpstr = sprintf('22-08-17/6-2-0.25x256/SR/%d.png',
                    name);
34              ch=findall(h,'tag','Colorbar');
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
```

```
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
     /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                  hamming(window_size),ceil(window_size/2))
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 123 123]);
74             set(h, 'Position' , [0 0 123 123]);
75             tmpstr = sprintf('22-08-17/6-2-0.25x256/VF/%d.png',
                  name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
79         vf_imported_segment = vf_imported_segment+segment_size;
80         vf_imported_count = vf_imported_count + 1;
81         name = name + 1;
82         start_pos = end_pos;
```

```
83          end_pos = end_pos + segment_size;
84      else
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
```

```
                    hamming(window_size),ceil(window_size/2))
111             xabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 123 123]);
116             set(h, 'Position' , [0 0 123 123]);
117             tmpstr = sprintf('22-08-17/6-2-0.25x256/VT/%d.png',
                    name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 2 seconds window with no overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
```

```
3 window_size = 200;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
      train_ratio/100));
8 sr_size = size(SR);
9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
      /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                  hamming(window_size),0)
27             xlabel('');
28             ylabel('');
```

```
29              set(findobj('type','axes'),'fontsize',5);
30              set(h, 'PaperUnits' , 'points');
31              set(h, 'PaperPosition' , [0 0 123 123]);
32              set(h, 'Position' , [0 0 123 123]);
33              tmpstr = sprintf('Output/SR/%d.png',name);
34              ch=findall(h,'tag','Colorbar');
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
```

```
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
    /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                hamming(window_size),0)
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 123 123]);
74             set(h, 'Position' , [0 0 123 123]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
79         vf_imported_segment = vf_imported_segment+segment_size;
80         vf_imported_count = vf_imported_count + 1;
81         name = name + 1;
82         start_pos = end_pos;
83         end_pos = end_pos + segment_size;
84     else
```

```
85          index = index + 1;
86          vf_imported_segment = 0;
87          start_pos = 1;
88          end_pos = segment_size;
89      end
90      close all
91  end
92  vt_size = size(VT);
93  vt_size = vt_size(2);
94  vt_cell_size = zeros(vt_size,1);
95  for i=1:vt_size
96  vt_cell_size(i) = size(VT{i},1);
97  end
98  vt_sum = sum(vt_cell_size);
99  expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
        /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),0)
111             xlabel('');
```

```
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
115             set(h, 'PaperPosition' , [0 0 123 123]);
116             set(h, 'Position' , [0 0 123 123]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 2 seconds window with 25% overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 200;
4 limit = 600000;
5 train_ratio = 75;
```

```
 6 expected_capped_input_no = limit/segment_size;
 7 expected_capped_train_no = int16(expected_capped_input_no*(
     train_ratio/100));
 8 sr_size = size(SR);
 9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
     /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                 hamming(window_size),ceil(window_size)/4)
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits' , 'points');
31             set(h, 'PaperPosition' , [0 0 123 123]);
```

```
32              set(h, 'Position' , [0 0 123 123]);
33              tmpstr = sprintf('Output/SR/%d.png',name);
34              ch=findall(h,'tag','Colorbar');
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
       /100));
59 vf_imported_count = 0;
```

```
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                   hamming(window_size),ceil(window_size)/4)
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 123 123]);
74             set(h, 'Position' , [0 0 123 123]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
79         vf_imported_segment = vf_imported_segment+segment_size;
80         vf_imported_count = vf_imported_count + 1;
81         name = name + 1;
82         start_pos = end_pos;
83         end_pos = end_pos + segment_size;
84     else
85         index = index + 1;
86         vf_imported_segment = 0;
87         start_pos = 1;
```

```
88          end_pos = segment_size;
89      end
90      close all
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                    hamming(window_size),ceil(window_size)/4)
111             xlabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits' , 'points');
```

```
115             set(h, 'PaperPosition' , [0 0 123 123]);
116             set(h, 'Position' , [0 0 123 123]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

On 2 seconds window with 50% overlapping on window

```
1 load('RHYTHMS.mat');
2 segment_size = XXX;
3 window_size = 200;
4 limit = 600000;
5 train_ratio = 75;
6 expected_capped_input_no = limit/segment_size;
7 expected_capped_train_no = int16(expected_capped_input_no*(
    train_ratio/100));
```

```
 8 sr_size = size(SR);
 9 sr_size = sr_size(2);
10 sr_cell_size = zeros(sr_size,1);
11 for i=1:sr_size
12 sr_cell_size(i) = size(SR{i},1);
13 end
14 sr_sum = sum(sr_cell_size);
15 expected_sr_input_no = int16(sr_sum/segment_size);
16 expected_sr_train_no = int16(expected_sr_input_no*(train_ratio
     /100));
17 sr_imported_count = 0;
18 sr_imported_segment = 0;
19 index = 1;
20 name = 1;
21 start_pos = 1;
22 end_pos = segment_size;
23 while sr_imported_count < expected_capped_input_no
24     if end_pos < size(SR{1,index},1)
25             h = figure('Visible','off')
26             spectrogram(SR{1,index}(start_pos:end_pos,1),
                  hamming(window_size),ceil(window_size/2))
27             xlabel('');
28             ylabel('');
29             set(findobj('type','axes'),'fontsize',5);
30             set(h, 'PaperUnits' , 'points');
31             set(h, 'PaperPosition' , [0 0 123 123]);
32             set(h, 'Position' , [0 0 123 123]);
33             tmpstr = sprintf('Output/SR/%d.png',name);
34             ch=findall(h,'tag','Colorbar');
```

```
35              delete(ch);
36              saveas(h,tmpstr);
37          sr_imported_segment = sr_imported_segment+segment_size;
38          sr_imported_count = sr_imported_count + 1;
39          name = name + 1;
40          start_pos = end_pos;
41          end_pos = end_pos + segment_size;
42      else
43          index = index + 1;
44          sr_imported_segment = 0;
45          start_pos = 1;
46          end_pos = segment_size;
47      end
48      close all
49 end
50 vf_size = size(VF);
51 vf_size = vf_size(2);
52 vf_cell_size = zeros(vf_size,1);
53 for i=1:vf_size
54 vf_cell_size(i) = size(VF{i},1);
55 end
56 vf_sum = sum(vf_cell_size);
57 expected_vf_input_no = int16(vf_sum/segment_size);
58 expected_vf_train_no = int16(expected_vf_input_no*(train_ratio
       /100));
59 vf_imported_count = 0;
60 vf_imported_segment = 0;
61 index = 1;
62 name = 1;
```

```
63 start_pos = 1;
64 end_pos = segment_size;
65 while vf_imported_count < expected_capped_input_no
66     if end_pos < size(VF{1,index},1)
67             h = figure('Visible','off')
68             spectrogram(VF{1,index}(start_pos:end_pos,1),
                   hamming(window_size),ceil(window_size/2))
69             xlabel('');
70             ylabel('');
71             set(findobj('type','axes'),'fontsize',5);
72             set(h, 'PaperUnits' , 'points');
73             set(h, 'PaperPosition' , [0 0 123 123]);
74             set(h, 'Position' , [0 0 123 123]);
75             tmpstr = sprintf('Output/VF/%d.png',name);
76             ch=findall(h,'tag','Colorbar');
77             delete(ch);
78             saveas(h,tmpstr);
79         vf_imported_segment = vf_imported_segment+segment_size;
80         vf_imported_count = vf_imported_count + 1;
81         name = name + 1;
82         start_pos = end_pos;
83         end_pos = end_pos + segment_size;
84     else
85         index = index + 1;
86         vf_imported_segment = 0;
87         start_pos = 1;
88         end_pos = segment_size;
89     end
90     close all
```

```
91 end
92 vt_size = size(VT);
93 vt_size = vt_size(2);
94 vt_cell_size = zeros(vt_size,1);
95 for i=1:vt_size
96 vt_cell_size(i) = size(VT{i},1);
97 end
98 vt_sum = sum(vt_cell_size);
99 expected_vt_input_no = int16(vt_sum/segment_size);
100 expected_vt_train_no = int16(expected_vt_input_no*(train_ratio
      /100));
101 vt_imported_count = 0;
102 vt_imported_segment = 0;
103 index = 1;
104 name = 1;
105 start_pos = 1;
106 end_pos = segment_size;
107 while vt_imported_count < expected_capped_input_no
108     if end_pos < size(VT{1,index},1)
109         h = figure('Visible','off')
110             spectrogram(VT{1,index}(start_pos:end_pos,1),
                  hamming(window_size),ceil(window_size/2))
111             xabel('');
112             ylabel('');
113             set(findobj('type','axes'),'fontsize',5);
114             set(h, 'PaperUnits', 'points');
115             set(h, 'PaperPosition', [0 0 123 123]);
116             set(h, 'Position', [0 0 123 123]);
117             tmpstr = sprintf('Output/VT/%d.png',name);
```

```
118             ch=findall(h,'tag','Colorbar');
119             delete(ch);
120             saveas(h,tmpstr);
121         vt_imported_segment = vt_imported_segment+segment_size;
122         vt_imported_count = vt_imported_count + 1;
123         name = name + 1;
124         start_pos = end_pos;
125         end_pos = end_pos + segment_size;
126     else
127         index = index + 1;
128         vt_imported_segment = 0;
129         start_pos = 1;
130         end_pos = segment_size;
131     end
132     close all
133 end
```

A.3 Training Scripts

A.3.1 Training Scripts for 192 x 192 images data sets

Note: XXX is replaced with the data set name during the implementation and YYY is replaced with the desired confusion matrix output directory.

```
1 dataset_name = XXX;
2 figure_path = YYY;
3
4 digitDatasetPath = fullfile(dataset_name);
5 digitData = imageDatastore(digitDatasetPath,'IncludeSubfolders'
      ,true,'LabelSource','foldernames');
6
```

```
CountLabel = digitData.countEachLabel;
img = readimage(digitData,1);
size(img)
trainingNumFiles = ceil(CountLabel{1,2}*0.75) ;
rng(1) % For reproducibility
[trainDigitData,testDigitData] = splitEachLabel(digitData,
    trainingNumFiles,'randomize');
  layers = [ imageInputLayer([192 192 3])
             convolution2dLayer(5,30)
             reluLayer
             maxPooling2dLayer(2,'Stride',2)
             convolution2dLayer(5,30)
             reluLayer
             maxPooling2dLayer(2,'Stride',2)
             fullyConnectedLayer(3)
             softmaxLayer
             classificationLayer()];

options = trainingOptions('sgdm','MaxEpochs',30, '
    InitialLearnRate',0.0001);
convnet = trainNetwork(trainDigitData,layers,options);
YTest = classify(convnet,testDigitData);
TTest = testDigitData.Labels;

t = confusionmattransform(TTest);
y = confusionmattransform(YTest);

h = figure('Visible','off')
plotconfusion(t,y);
```

```
34 strname = strcat(dataset_name, '.fig');
35 output = strcat(figure_path,strname);
36 savefig(h,output);
37 close all
```

A.3.2 Training Scripts for 256 x 256 images data sets

Note: XXX is replaced with the data set name during the implementation and YYY is replaced with the desired confusion matrix output directory.

```
1 dataset_name = XXX;
2 figure_path = YYY;
3
4 digitDatasetPath = fullfile(dataset_name);
5 digitData = imageDatastore(digitDatasetPath,'IncludeSubfolders'
      ,true,'LabelSource','foldernames');
6
7 CountLabel = digitData.countEachLabel;
8 img = readimage(digitData,1);
9 size(img)
10 trainingNumFiles = ceil(CountLabel{1,2}*0.75) ;
11 rng(1) % For reproducibility
12 [trainDigitData,testDigitData] = splitEachLabel(digitData,
      trainingNumFiles,'randomize');
13   layers = [ imageInputLayer([256 256 3])
14              convolution2dLayer(5,30)
15              reluLayer
16              maxPooling2dLayer(2,'Stride',2)
17              convolution2dLayer(5,30)
18              reluLayer
```

```
19              maxPooling2dLayer(2,'Stride',2)
20              fullyConnectedLayer(3)
21              softmaxLayer
22              classificationLayer()];
23
24 options = trainingOptions('sgdm','MaxEpochs',30, '
     InitialLearnRate',0.0001);
25 convnet = trainNetwork(trainDigitData,layers,options);
26 YTest = classify(convnet,testDigitData);
27 TTest = testDigitData.Labels;
28
29 t = confusionmattransform(TTest);
30 y = confusionmattransform(YTest);
31
32 h = figure('Visible','off')
33 plotconfusion(t,y);
34 strname = strcat(dataset_name, '.fig');
35 output = strcat(figure_path,strname);
36 savefig(h,output);
37 close all
```