Machine Learning Engineer Nanodegree

Cardiac Arrhythmia ML

Capstone Project Report

Akshay Bhatia

June 2017

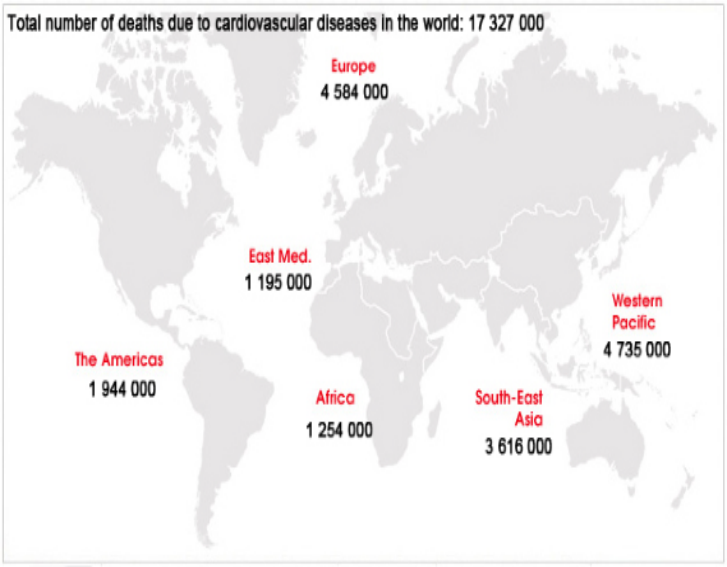
Definition

Project Overview

Cardiac Arrhythmia is a condition where a person suffers from an irregular or abnormal heart rhythm. It is due to the malfunction in the electrical impulses within the heart that coordinate how it beats. As a result, the heart beats too fast, too slowly, or irregularly. The rhythm of the heart is controlled by a node at the top of the heart, called the sinus node, which triggers an electrical signal that travels through the heart – causing the heart to beat, pumping blood around the body. Excess electrical activity in the top or bottom of the heart means that the heart doesn't pump efficiently. The most common symptoms of Arrhythmia include shortness of breath, fainting, an unexpected loss of heart function and unconsciousness that leads to death within minutes unless the person receives emergency medical treatment to restart the heart. So, it's vital to know about and understand the condition, what danger signs to look out for and how to diagnose it early.

To diagnose Cardiac Arrhythmia early, doctors need to carefully evaluate heartbeats from different locations of the body accurately. Reviewing these fundamental heart sounds (FHSs)

for every patient manually is very time consuming for medicians. A potential solution to this is to provide automated diagnosis using a mobile phone or through some other medium. Hence classification of heart sound recordings using Machine Learning techniques could help overcome this problem. Being a patient of Cardiac Arrhythmia, I was inspired to take up this challenge and apply Machine Learning techniques to this domain.



This report illustrates the Heart Sound classification process using machine learning techniques. It starts with a small introduction about the datasets, followed by an exploratory data analysis that shows some summary statistics about the data sets, and finally the best prediction algorithm.

Problem Statment

Automatic heart sound classification has promising potential to accurately detect heart diseases such as Cardiac. It could be used in non-clinical environments such as patient's residence by medical personnels as a quick heart pathology screening technique or at places with poor financial conditions to support healthcare.

The focus of this project is to classify whether the patient has "normal" or "abnormal" heart sound from the Phonocardiogram (PCG) or heartbeat recordings to quickly identify patients who would require further diagnosis. This is a supervised learning problem since we already know if the heart sound in training dataset is normal or abnormal. The basic idea is to convert each heart sound recording(wav file) to a spectrogram image and train a Convolutional Neural Network over those images. Then given a new PCG recording, we will be able to classify it as normal or abnormal.

Metrics

The metrics to determine how well the model performs on the entire dataset is logarithmic loss which is also named as ‘categorial cross entropy’.

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log\left(p_{ij}\right)$$

This can be described as negative the log likelihood of the model given each observation is chosen independently from a distribution that places the predicted probability mass on the corresponding class, for each observation. In our case, each recording is already labeled with one true class and for each one, a set of predicted probabilities is considered an outcome. Here *N* is the number of images in the test set, *M* is the number of image class labels i.e 2, *log* is the natural logarithm, *Yi,j* is 1 if observation belongs to class and 0 otherwise, and P*i,j* is the predicted probability that observation belongs to given class..

Since the dataset is imbalanced i.e number of ‘normal’ samples is greater than number of ‘abnormal’ samples in the training dataset, accuracy is not only the metric I considered. Hence the model was evaluated on ‘precision’ which can be described as the ratio of correct positive predictions made out of the total positive predictions made, ‘Recall’ which is the ratio of correct positive predictions made out of the actual total that were positive. I also calculated another metric known as ‘f-Beta score’ which is given by the weighted average of precision and recall.

Metric	Formula
True positive rate, recall	$\frac{TP}{TP+FN}$
False positive rate	$\frac{FP}{FP+TN}$
Precision	$\frac{TP}{TP+FP}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
F-measure	$\frac{2\cdot precision\cdot recall}{precision+recall}$

To summarise, recall addresses the question: "Given a positive example, will the classifier detect it ?” whereas precision addresses the question: "Given a positive prediction from the classifier, how likely is it to be correct ?”.

Analysis

Data Exploration and Visualisation

The dataset used for this capstone is available freely as part of the PhysioNet / Computing in Cardiology Challenge 2016 which focuses on automatic classification of normal / abnormal phonocardiogram (PCG) recording. The dataset has 4,430 recordings taken from 1,072 subjects, collected from both healthy subjects and patients with a variety of conditions such as heart valve disease. Along with clean heart sounds, the dataset also contains some noisy recordings. The samples have been obtained from both normal subjects and pathological patients, providing a variety of signal sources.

The training data consists of PCG signals of varying length, anywhere between 5s to just over 120s all sampled at 2000 Hz and are provided in .wav format. Each recording contains only one PCG lead. Each recording has been labelled as normal or abnormal in a separate file(.hea format). As expected, the distribution of data is highly imbalanced and therefore, various data augmentation techniques have been implemented. Since the dataset is very large in size, I decided to work with a smaller subset. Therefore I choose the training set to be about 1000 labeled recordings and the validation set to be 200 recordings.

The recordings in the dataset are based on time-frequency features of the raw signal. So to capture the intensity and the pitch of the recordings, I decided to use spectrograms. Due to its robustness and interpretability, Spectrograms have been applied to a wide range of areas since it a visual way of representing the signal strength, or “loudness”, of a signal over time at various frequencies present in a waveform. Therefore, raw wav files need parsed into spectrograms first and the class labels are extracted from header (.hea) files . To do this, I decided to write a python script ‘convert_to_spectrogram’ separately from the project notebook to convert every recording into its subsequent spectrogram. Splitting the dataset into training and testing sets is also automatically taken care by this script.

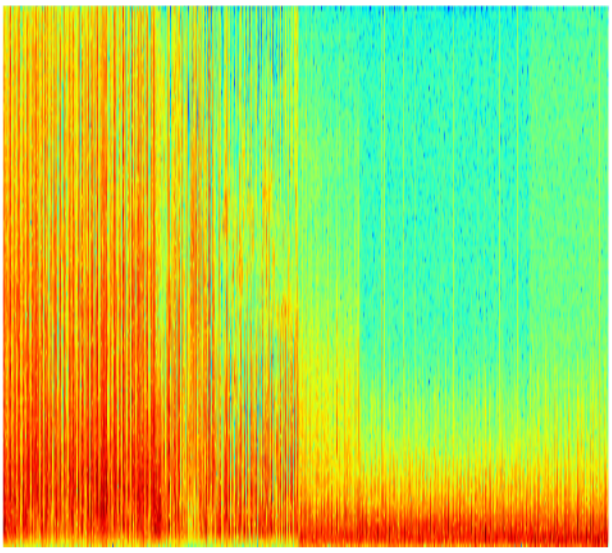
The script follows the following approach:

- Load the .wav files into memory.

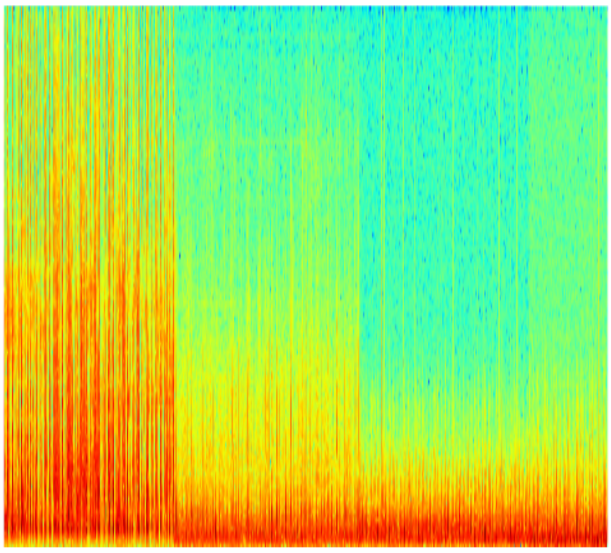
- Process all the recordings and header files and parse them for file names and class labels and store them in separate lists.
- Once we have all the class labels and file names, we use scipy's *wavfile* class to get the sample rate and data of the wav file.
- The length of the windowing segments and sampling frequency for the spectrograms will be 256. Matplotllib's *specgram*( ) function computes and plot a spectrogram of data. It takes minimum 3 parameters :
    - x. : Array or sequence containing the data or data of wav file
    - Fs : The sampling frequency or the samples per time unit which is used to calculate Fourier frequencies
    - NFFT: The number of data points used in each block for the FFT.

  The spectrogram is then plotted as a colormap.

- Finally, using the list of class labels we separate the spectrograms into individual classes. This way we can easily feed the images directly into our model using Keras' inbuilt *ImageDataGenrator* class as shown in the notebook.



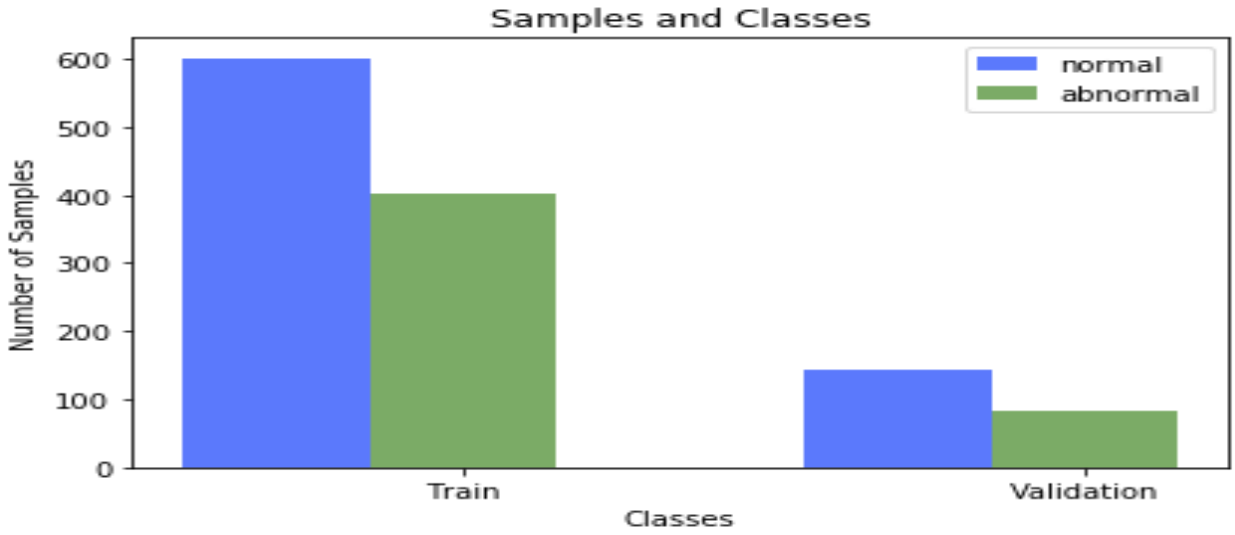
Example of an abnormal heartbeat
from the training set



Example of a normal heartbeat
from the training set

Let's take a look at what the spectrogram does. At the most basic level, the spectrogram is dividing the sound sample into multiple "blocks" (in time domain) and plotting the Fast Fourier Transform (FFT) of each block and displaying them in the same graph. The x-axis is time and the y axis is frequency and the analysis is done only on one of the "blocks" of the time domain. The amplitude of FFT for each time domain block is represented by colour and intensity. This is used to show how energy levels vary over time.

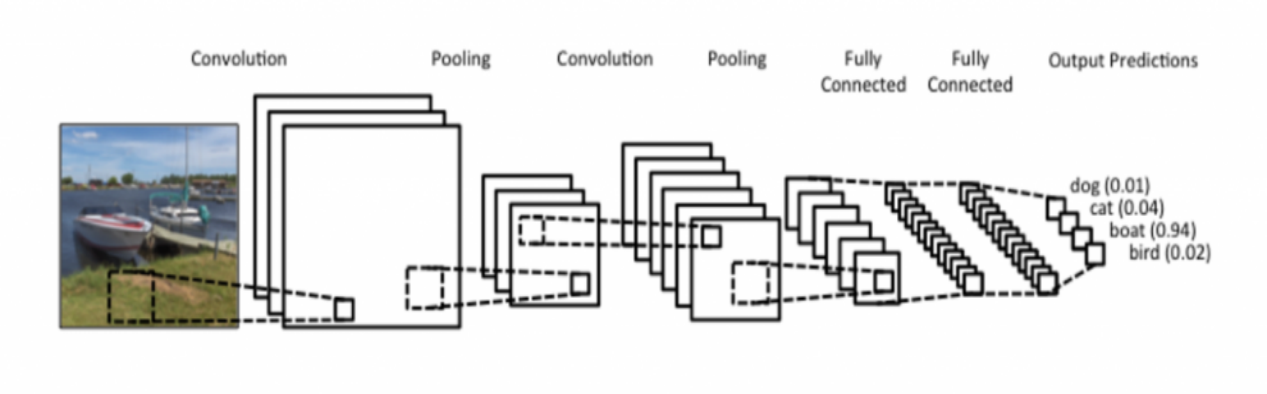
The image below shows the number of samples for each class in training and validation set.



Algorithms and Techniques

Recent algorithms applied to Cardiology challenges include Heart sound segmentation, transformation of one-dimensional waveforms into two-dimensional time frequency heat map representations using Mel-frequency coefficients and Classification of MFCC heat maps. But given the recent success of Deep Neural Networks in computer vision, I decided use Convolutional Neural Networks for this problem. CNNs are the modern day state-of-the-art algorithm for image classification and visual recognition tasks and have now improved to the point where they now outperform humans on computer vision challenges such as Imagenet and COCO.

CNN are like normal neural networks. Every neuron in each layer receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network expresses a single differentiable score function. But instead they take images as input which allows us to encode certain properties into the architecture and outputs classes (dog, cat, tree, etc.) according to our dataset. A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume. They are composed of many layers like convolution, pooling, fully connected as shown below.



Implementing a CNN from scratch:

Consider our input image to be 32x32x3. On each 2D array of data (image), we train a whole bunch of N x N kernels. These kernels are nothing but filters that we run across the 2D array. For each position (x,y), we compute the dot product summation between this kernel and the image values around that point. This process is called convolution, hence the name Convolutional Neural Networks. In this way, we are able to detect edges, lines, blobs of colours and other visual elements. Then we apply a nonlinear layer (or activation layer) immediately afterward to introduce nonlinearity to a system that basically has just been computing linear operations during the convolution layers. We will then apply a pooling layer which outputs the maximum number in every subregion that the filter convolves around. This reduces the spatial dimension (the length and the width but not the depth) of the input volume (from 32x32 to 16x16) and the amount of parameters or weights drastically. In the end, we take all these features and apply what is called a 'fully connected' layer to it. In this layer, each neuron in this layer will be connected to all the neutrons in the previous one just like in a normal neural network. It looks at the high level features which correlate to a certain class so that when we compute the products between the weights and the previous layer, we get the correct probabilities for the different classes. In a nutshell I decided to use CNN for this problem because contrary to normal neural networks where all the neurones in one

layer are connected to all the neurones in the previous layer and the operation becomes computationally very intensive, Convolutional layers are technically locally connected layers and all the pixel positions share the same filter weights. This significantly reduced number of parameters and gives better results.

Finally, I would also be implementing Transfer learning. This is the technique of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset) and then fine-tuning the model with my own dataset and classifier. The idea is that the pre-trained model will act as a feature extractor. What this does is that rather than training the whole network through a random initialisation of weights, we can use the weights of the pre-trained model (and freeze them) and focus on the more important layers for training. For this purpose, I decided to use VGG16 model since it was the runner-up in ILSVRC 2014 and has been used in many production level products.

Benchmark Model

Baseline model for this project would be an accuracy of about 60%. This is because randomly selecting a sample from the training dataset as normal would result in accuracy of 60% since the training set has about 600 samples as normal and 400 as abnormal. Also the Physionet Cardiology Challenge 2016 has provided description of the baseline classification method with a score of 0.71 or 71%.

Methodology

Data Preprocessing

Data Augmentation will be important for our model so that it never sees the exact same picture twice since different spectrograms look similar. This helps prevent overfitting and the

model generalises better. For images, this could be done by rotating the original image, changing lighting conditions, cropping it differently, so for one image we can generate different sub-samples.

For this capstone, the following data augmentation techniques were applied :

- *Resizing* : All the images in the dataset(training and testing) were resized to 150x150x3 (width = height = 150 ; colour channel = 3) before feeding into the CNN.
- *Normalise* : Pixel values for all images were normalised between 0 and 1. This was done by subtracting the minimum pixel (i.e 0) and dividing by maximum pixel value(i.e 255). In *Keras*, this was done by setting the 'rescale' attribute to 1/.255.
- *Shear Transformation* : Shear Transformation were applied to control the shear intensity of the input images. It was set to 0.2 using 'shear_range' attribute.
- *Zooming* : Randomly zooming inside images by setting 'zoom_range' attribute to 0.2
- *Flipping* : Half of the training inputs were randomly flipped horizontally using the 'horizontal_flip' attribute.

All the augmentation techniques were implemented using Keras' '*ImageDataGenerator*' class which takes in the above mentioned attributes and provides us with real-time data augmentation. Apart from these, the '*ImageDataGenerator*' class has the method '*flow_from_directory*' which it takes the path to a directory, and generates batches of augmented data. The data was looped over (in batches) indefinitely. It further provides us with the number of images in each set. Splitting the dataset into a training and testing set was done manually before applying the data preprocessing steps.

Implementation

The implantation of the model is done and described briefly in *CardiacML Final.ipynb.* It follows the following structure:

- Installing the libraries and downloading and importing the datasets into memory. Converting the raw recordings (.wav) files into spectrograms and extracting the class labels from header (.hea) files.

- The spectrograms(images) were augmented so that the model generalises better on test data. This included normalising pixel values of images, zooming, flipping on training set.
- Defining a helper function for evaluating metrics like precision, recall and f-Beta score since Keras only had ‘accuracy’ as an evaluator. Apart from helping functions for metrics, simple visualisation methods to visualise the metrics were also written.
- Defining the model architecture. This includes the types and numbers of layers such as convolution, pooling, fully connected layers.
- Deciding the hyper parameters for the network such as dropout to reduce overfitting, filter size, stride size and type of padding, activation functions, number of iterations and learning rate.
- Finally choosing the loss function, optimiser and metrics and training and evaluating the model.

Since the first two steps and basics of CNN were already discussed in detail earlier, we take a look at rest of them. (Note: the number of epochs, learning rate, optimiser, loss function are same for the following models)

Model Architecture for initial network

I first implemented a small covnet with a simple stack of 3 convolution layers with a ReLU activation and followed by max-pooling layer. This is then followed by two fully-connected layers. In the end we are left with a single unit and hence use sigmoid activation since this is a binary classification. Each convolution block creates convolution kernel which is convolved with the input layer over a single spatial followed by a non linear activation function. The pooling layer downscales the input in both spatial dimension. The [number of filters, filter size and downscale(pooling) size] for 1st, 2nd and 3rd layers are [32, (3,3), (2,2)], [32, (3,3), (2,2)] and [64, (3,3), (2,2)] respectively. After flattening the image i.e converting our 3D feature maps to 1D feature vectors we apply the first fully connected layer which has 64 units and ReLU activation. Finally we apply sigmoid activation to the last unit to classify into ‘normal’ or ‘abnormal’.

Model Architecture for updated (deeper) network

After the initial model, I decided to implement a deeper covnet. The stack for each block is as follows - I first applied zero padding i.e. adding rows and columns of zeros at the top, bottom, left and right side of an image. Then convolutions and activation function is followed by Batch Normalisation layer. This way the activations of the previous layer at each batch are normalised i.e mean activation is almost 0 and the activation standard deviation close to 1. After this I applied a max-pooling layer. This architecture has 5 such blocks. This is then followed by two fully-connected layers. I also used dropout to avoid overfitting. In the end we are left with a single unit and hence use sigmoid activation since this is a binary classification. Kernel size for zero padding, filter size, downscale size in each block is same and is (1,1), (3,3) and (2,2) respectively. The number of filters for 1st, 2nd, 3rd, 4th and 5th layers are 64, 128, 256, 512, 512 respectively. After flattening the image i.e converting our 3D feature maps to 1D feature vectors we apply the first fully connected layer which has 64 units and ReLU activation. Finally we apply sigmoid activation to the last unit to classify into 'normal' or 'abnormal'.

Refinement

Now to improve upon the original model, I decided to implement a pre trained model VGG16. This process is known as Transfer Learning and allows us to use the pre-trained models trained on datasets with millions of images such as COCO, Imagenet etc. This means instead of building a model from scratch to solve a similar dataset, we can use the model trained on some other dataset as a starting point.

Implementation

The last layer of the network was removed and replaced with custom classifier i.e classifying between normal and abnormal. This means I used the network only upto the fully connected layers and removed all subsequent layers. Since the structure of my model was not exactly the same as the one used when training weights, I needed to load the weights (vgg16_weights.h5) file. At last, the bottleneck features VGG16 model were saved.

The above figure shows the architecture for VGG16 model. It is important to note that input dimensions for this model were 150x150 instead of 224x224 as shown in the figure. The architecture for this model is as follows: Each block is stacked with convolution layers and at the end of every block a pooling layer to reduce spatial dimension is applied. The output dimensions(channel) is increased by a factor of 2 every block. It should also be noted that zero padding is applied before every convolution layer in each block since it is not shown in the figure. The last three fully connection layers were removed as discussed above.

Just as a mere experiment, I also decided implemented the VGG16 model architecture from scratch including the fully connected layer to see how it performed.

Results

Model Evaluation, Validation and Justification

The loss function and metrics for each model was ‘binary cross_entropy’ and ‘accuracy’, ‘recall’, ‘precision’, ‘F-beta score’ respectively. The data for all metrics is displayed in the notebook. The results and justification for each model are as follows:

Model 1: The training accuracy increases every epoch and the loss also goes down. But validation accuracy first goes down and then goes up while validation loss first increases and then decreases. This shows that the model is not very efficient and could be improved using a complex model.

Model 2: The training accuracy increases first but goes down during final epoch. But the improvement could been seen as validation accuracy increases and loss decreases. This confirms the hypothesis established from previous model.

Model 3 (Transfer Learning using VGG16): This model was trained for 50 epoch to capture to capture all features the VGG16 model has learned. The training accuracy increases all the way up to 88% which is very interesting. But the major drawback was the drastic decrease in validation accuracy than previous two models. This large difference in accuracies is harder to explain but may be due to a smaller dataset which results in overfitting. One interesting reason could be that the Imagenet dataset(on which VGG16 and all other pre-trained model are trained) does not have any spectrogram images and this means that the model was not able to use any information it learned previously to our set of data.

Model 4 (Final): An average result using a the VGG16 pre-trained model inspired me to use its architecture and implement it from scratch. I thus implemented the VGG16 architecture with two fully connected layers at the end consisting of 4096 units each. The validation accuracy is higher than training accuracy which is a good sign. But the overall training accuracy is much lower than previous model.

Based on the above evaluations , it is obvious that every model has its merits and flaws. Even though no model could capture the complex features to detect the heartbeats as ‘normal’ or ‘abnormal’ and the accuracy results of the capstone did not match the benchmark confidently, which would be due to many factors such as a small dataset, the validation accuracy of 65% and training accuracy of 86% could be considered reasonable. It came close enough to validate the approach that I took while building the model. It’s likely that additional experimentation fine tuning the model and adjusting and tweaking the hyperparameters could lead to even better results.

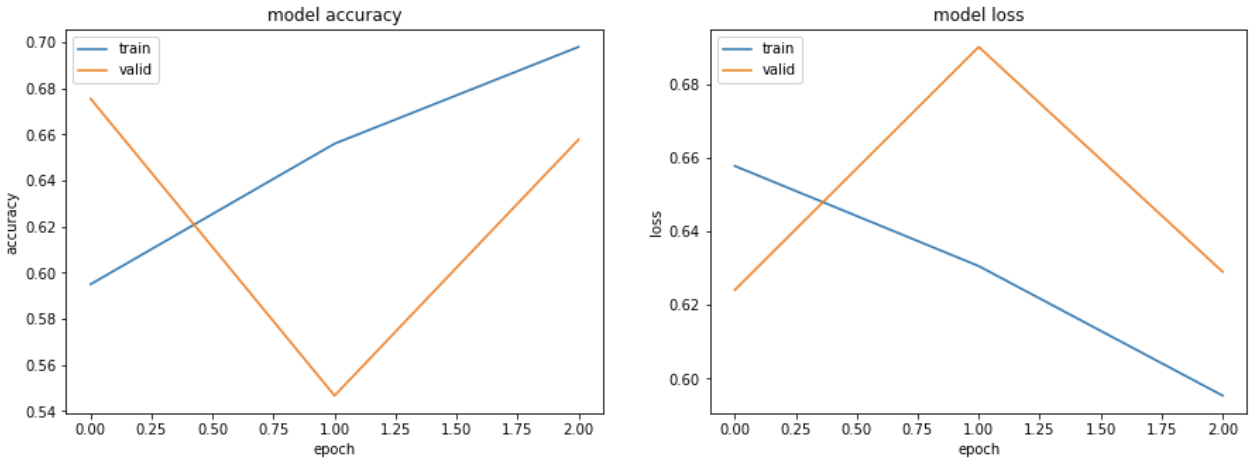
A suitable solution would be a hybrid combination of the models with more data and compute power. Given the nature of data along with the noise and complexity of the problem, I am satisfied with how the results turned out. Although the final model is not a robust enough to be applied as a real world application, it is a good introduction to the problem and could be build upon with more experimentation.

Conclusion

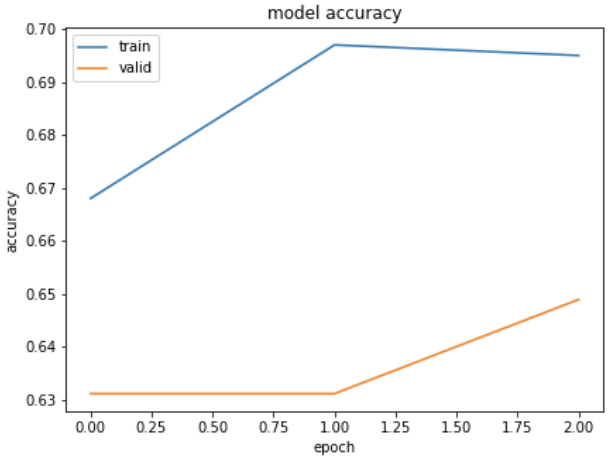
Free-Form Visualisation

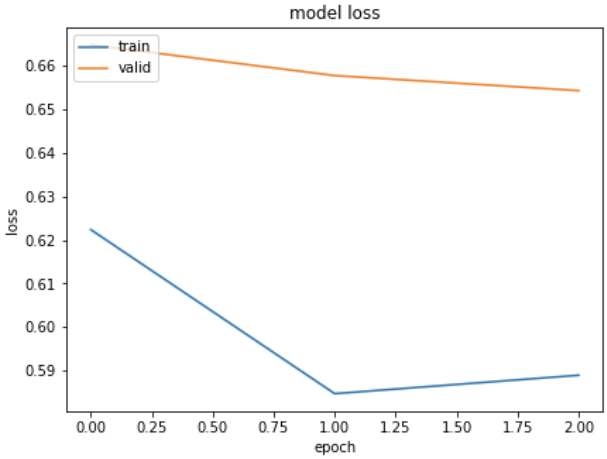
Since the spectrograms images are not similar to normal real world images (such as of a animal, car or fruit), it did not make sense to use them as visualisations for this project in contrast to other Image Classification problems. Hence I will use the accuracy and loss graphs for each model as visualisations.

The first model was a very small covnet consisting of a simple stack of 3 convolution layers with a ReLU activation and followed by max-pooling layer followed by two fully-connected layers. The accuracy and loss plots are as follows:

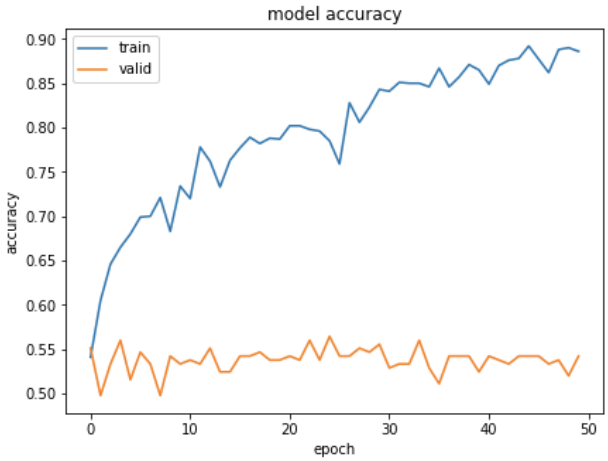


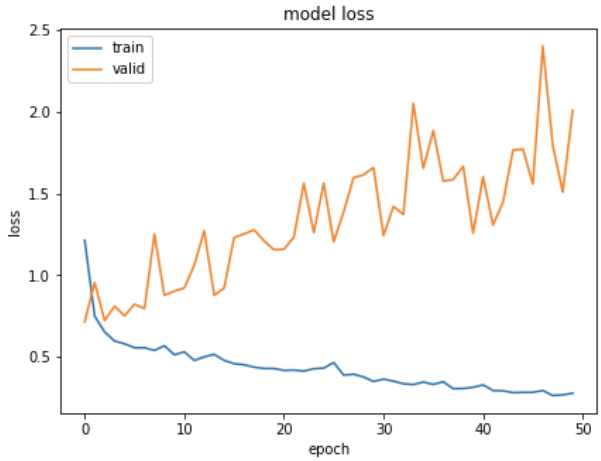
For the second model, the stack for each block consists of zero padding, convolutions and activation function followed by max pooling layer. It is a more deeper network as compared to the previous model and the plots are as follows:



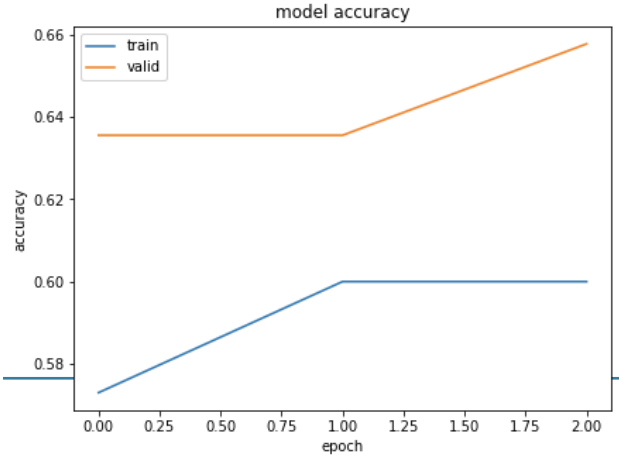


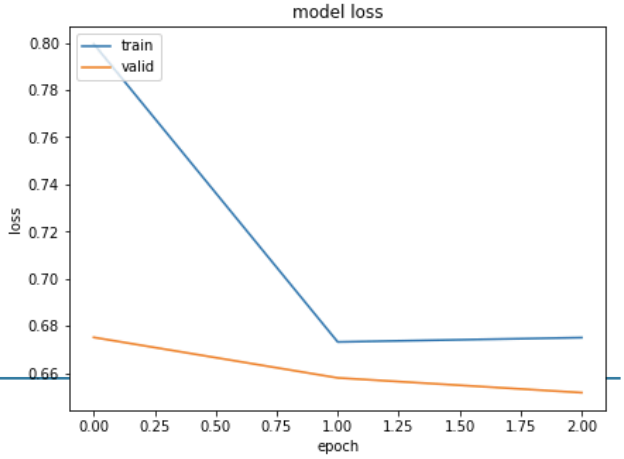
Model 3 which implies Transfer Learning gives the following results:





Implementing VGG16's architecture from scratch gave the following results:





Reflection

It is a very well known fact that Deep Convolutional Networks outperform every algorithm when is comes to Computer Vision and Image Classification tasks. However, as this was a non-trivial problem, I was not easily convinced that a single network would be able to classify the recordings with great accuracy. The results confirm that getting a better accuracy and more efficient solution is possible with few tweaks as mentioned in Improvements section.

The approach for a end to end solution was to covert the recordings to spectrogram images and then train a Deep Convolutional Networks. I progressively implemented different architectures to classify the images and obtain a better result. I also applied various data augmentation techniques. Initially, I started with a simple CNN and then shifted to one with more layers. To further refine and improve results, I implemented a technique known as 'Transfer Learning'. I used the pre trained VGG16 model and extracted features from images and then ran my own classifier over those features. I also trained the images on a custom model based on VGG16 model architecture from scratch to improve results.

The most difficult part was to extract the spectrogram images from the recordings and train the network over them on my local machine since the computational resources were limited and this made the project very daunting to perform. Another challenging aspect of the project was to get a high recall and accuracy with lack of training samples and recordings filled with noise. It was very hard achieve great results for this type of classification. The techniques discussed in improvements section will probably help to achieve a better overall result.

At the same time, it is extremely important to note that though the purpose of the capstone was to automate the process of the classifying the recordings, the model in this project is not intended for diagnostic purposes. It is only designed to provide an overview of how Machine Learning techniques can be used to solve real world problems.

Improvement

One of the most simple ways to improve upon the existing model is to use more data. Due to high computational cost and time and memory constraints, I was only able to use a small subset of the complete dataset(1000 samples for training and 200 for validation). This can be further overcome by training on a GPU or in the cloud. Another way that will be effective in improving the model would be to use a different model architecture such as InceptionV3 or RESNET.

The dataset also had a lot of noise since the heart recordings consisted of people talking and breathing heavily and due to high computational cost it was not possible to do so. But doing so would significantly improve our model. To accurately separate signal from noise, the original signal can be filtered with a high pass Butterworth to remove noise above 4Hz (or 240 beats per minute). The filtered signal then could be transformed to it's approximate frequency domain. Given the way that our model reached a relatively high accuracy rate and then plateaued on both the training and validation sets suggests that we may have hit a limit on the complexity of features our model was able to encode. We could attempt to address this by adding additional convolution layers, expanding the number of filters in existing layers, or even trying some novel new architectures. Microsoft's winning ImageNet model, for example, uses residual layers to make features detected at lower layers directly available to higher levels.

Another option that could be explored is the use of different colour scale. While the RGB colour scheme that we used to train this model is useful to represent colours in computers, it may not mean that it is the best colour scale to use for image recognition. Therefore, other colour scales such as grayscale or HSV may be useful in various computer vision techniques.

References

https://physionet.org/challenge/2016/

https://www.kaggle.com/kinguistics/heartbeat-sounds

http://www.heart.org/HEARTORG/Conditions/Arrhythmia/AboutArrhythmia/About-Arrhythmia_UCM_002010_Article.jsp#.WTeSZROGMUs

http://www.kdnuggets.com/2016/09/urban-sound-classification-neural-networks-tensorflow.html

https://www.physionet.org/challenge/2016/papers/challenge2016.pdf