Udacity Machine Learning Advanced Nanodegree

Capstone Project Proposal

Arrhythmia Detection

Introduction

A pipeline to detect beat annotations corresponding to the heartbeat irregularity in the MIT - BIH Arrhythmia database and the UCI Arrhythmia dataset.

Overview

The goal was to develop a model pipeline which can diagnose irregular heart rhythms, also known as arrhythmias, from ECG signals. The approach taken was deep convolutional network and ensemble machine learning algorithm which can map a

sequence of ECG samples to a sequence of arrhythmia annotations. Few literature referenced for the deep learning approach are the following:

1. https://www.sciencedirect.com/science/article/pii/S0169260715003314
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4897569/
3. https://www.ijimai.org/journal/sites/default/files/IJIMAI20111_4_11.pdf

All of these papers use the same dataset (MIT-BIH Arrhythmia dataset) and the same approach (Neural Networks),

Few literature for the machine learning approach is:

http://cs229.stanford.edu/proj2014/Manas%20Karandikar,%20Giulia%20Guidi,%20Classification%20Of%20Arrhythmia%20Using%20ECG%20Data.pdf

This uses the same dataset (UCI one) and one of the same algorithm used in Ensembler (Support Vector Machines, Decision Trees and Random Forest).

My personal motivation behind arrhythmia classification is the prevailing epidemic of cardiac deaths in my home country, India. Most of deaths in India are due to CVD (Cardiovascular Diseases) and this CVD also has a high prevalence among the youths.

The data consists of 12 unique cardiac types that needs to be classified. Though for some labels, instances are too few. Owning to sparse data, I may or may not try to classify all of them. That is, I may try to only classify, say top 5 most frequent arrhythmias so my network trains well.

Goals

1. To find a Database which contains substantial number of instances so that a model pipeline can be built.
2. After this formulate a way of working with the data. This involves converting the dataset for being worked upon by Deep Learning and Machine Learning approaches.
3. To build a overall pipeline which could be easily changed and tweaked towards working on any specific set of beat annotation found in the data.

Dependencies and softwares used

1. **Python 3.6**
2. **Numpy**
3. **Matplotlib**
4. **Wfdb**
5. **Tensorflow**
6. **Keras**
7. **Scikit**
8. **xgboost**
9. **Pandas**
10. **Scipy**

Datasets used

MIT-BIH Arrhythmia dataset retrieved from physionet.org

https://physionet.org/physiobank/database/mitdb/

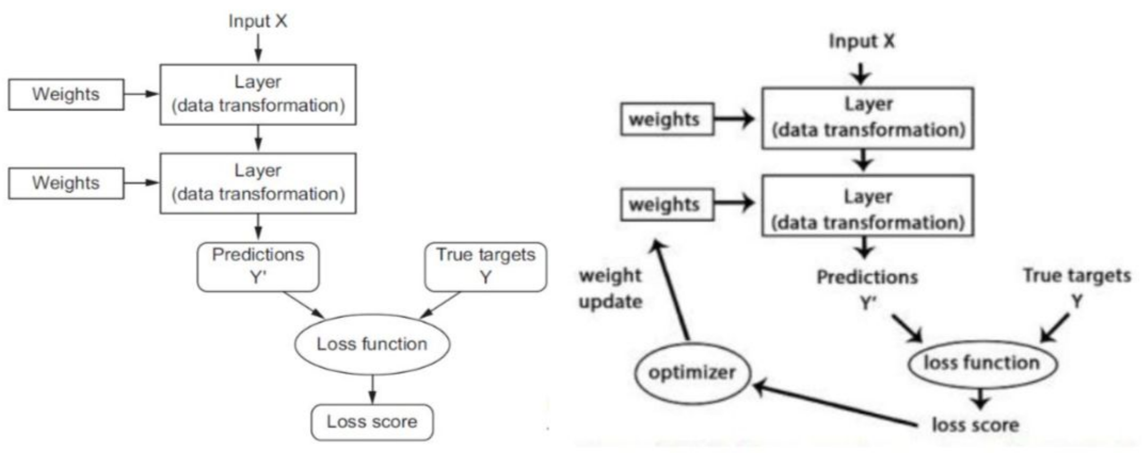
This dataset has 135 instances of ECG recordings from 47 subjects. The ID of patient has been retracted for privacy reasons. Each instance is ~48 hour long. There are about 65000 annotations in each signal. Frequency is 360 and the recorded voltage range is 10mV. The total arrhythmia classes here are 50.

The other dataset used is UCI Arrhythmia dataset. This has 452 instances that needs to be classified into 16 classes of arrhythmia. The names and id of patients has been retracted for privacy reasons. This data has recording of the various channels from ECG leads and the QRS complex wave which will be helpful in classifying it.

https://archive.ics.uci.edu/ml/datasets/arrhythmia (link to UCI dataset)

Problem Statement

My approach to the problem is to harness the capability of Deep Learning models and Machine Learning models to extract important features corresponding to the objective function. Let us talk about deep learning first.

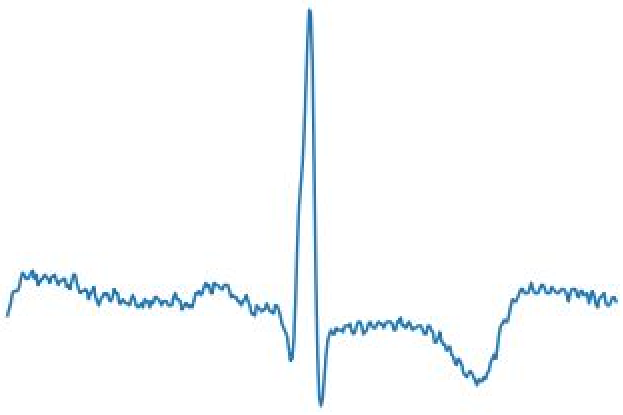


This flowchart explains the approach of deep learning. The input X (in our case it is the ECG) which is transformed by multiplication with weights(weights are matrix of certain dimensions). This is propagated forward which gives the prediction Y after being transformed by successive matrix multiplications. Now Y is compared with the original expected output and this gives a loss or a measure of the inaccuracy in the model. This loss is retracted back by back propagation algorithm to update the weights of the network. With increasing epochs the weights are updated and the network actually learns to make correct predictions. The optimiser updates the weights with gradient descent.

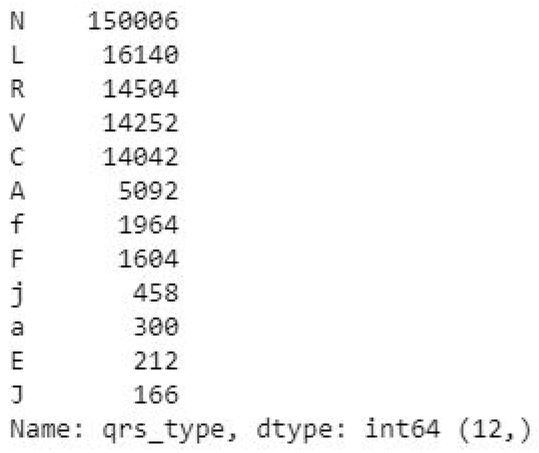
This is a classic CLASSIFICATION problem where the datasets has 50 unique arrhythmias that needs to be classified. I may or may not try to classify all of them, since few arrhythmias have less instances and network may not train well as such.

I will use keras to make 4-5 deep layer CNN network to try to classify it.

Inputs are the QRS complex of an ECG. This can be visualized a peaks in a signal. I have attached the photo below.



The output is the class label of predicted arrhythmia.



```
N    150006
L     16140
R     14504
V     14252
C     14042
A      5092
f      1964
F      1604
j       458
a       300
E       212
J       166
Name: qrs_type, dtype: int64 (12,)
```

Let us now visit the Machine Learning part on the UCI dataset.

Here, I am going to use ensemble learning for the classification. Ensemble as the name suggests, takes into account the various machine learning algorithms supplied to it and based on the weightage attached to each model, it tries to predict the correct class. Note that since it is an multi label classification problem, the algorithms used must output a class probabilities function. Hence, those algorithms like linear SVC which don't have class probabilities (no predict_proba method in scikit-learn. Predict_proba method, as of now, only supports hinge loss and modified huber loss) cannot be used in ensemble.

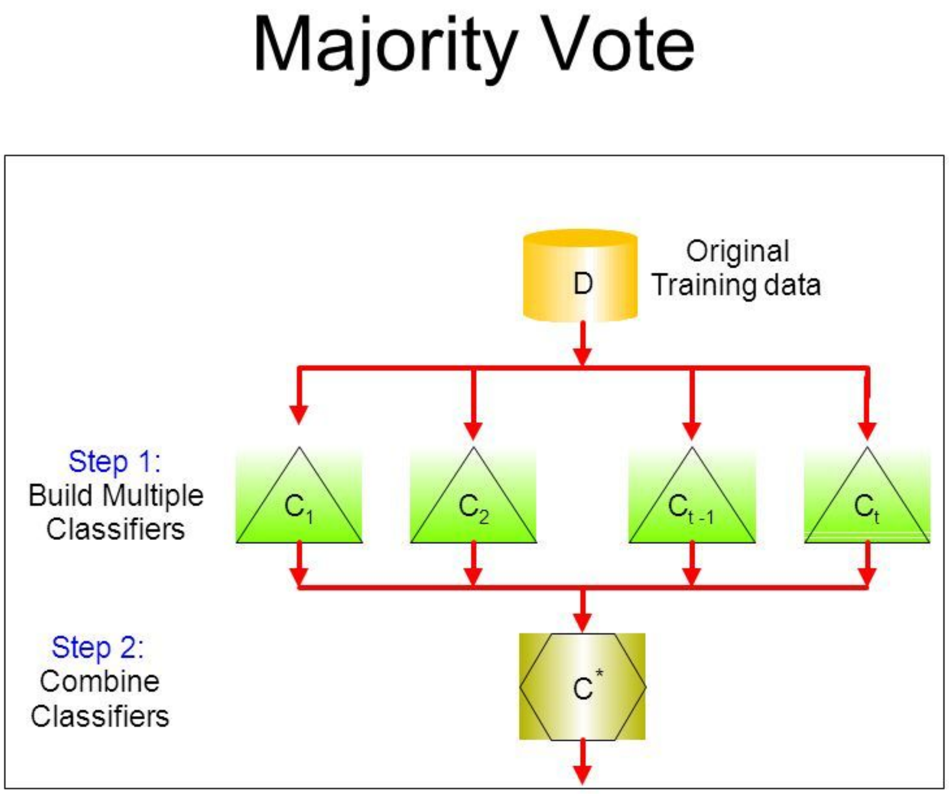


Image is of Ensemble learning using majority vote.

Benchmark Model-Deep Learning

Please refer to the following research paper whose implementation I am going to use as my benchmark
http://cs229.stanford.edu/proj2014/Samuel%20McCandlish,%20Taylor%20Barrella,%20Identifying%20Arrhythmia%20from%20Electrocardiogram%20Data.pdf

Identifying Arrhythmia from Electrocardiogram Data; By Taylor Barrella*, Samuel McCandlish; December 12, 2014

They have used 2 methods, namely, Support Vector Machines (SVM) and Multi Layer Perceptron (MLP).

Here I have attached a snapshot of the research paper detailing their MLP approach in the forthcoming pages.

- The model is evaluated four times: once using the raw data as features, once using discrete FFTs, once using discrete Haar wavelet transforms, and once using discrete db3 wavelet transforms.

- The model is a multilayer perceptron (MLP) with a single hidden layer. More details follow.

- The model is trained and then evaluated by using 12-fold cross validation.

A tutorial was used to implement the MLP [4]. This uses the Theano package for Python [5].

With a single hidden layer, the MLP works by making a prediction based off of an output vector

$$f(x) = G\left(b^{(2)} + W^{(2)}s(b^{(1)} + W^{(1)}x)\right). \tag{2}$$

The function G is for logistic regression. To accommodate multiclass classification, G is the softmax function

$$G(x; W, b)_i = \frac{e^{(Wx+b)_i}}{\sum_j e^{(Wx+b)_j}}. \tag{3}$$

The function s is a nonlinear activation function for the hidden layer. Here, we choose

$$s(x) = \tanh x. \tag{4}$$

Finally, a prediction is made by

$$h(x) = \arg\max_i f(x)_i. \tag{5}$$

The parameters $W^{(2)}$ (a matrix) and $b^{(2)}$ (a vector) are weights. The additional weights for the hidden layer, $W^{(1)}$ and $b^{(1)}$, are called hyperparameters. These four sets of parameters are learned by training and using backpropagation to calculate the error (cost function). Theano is able to calculate gradients so that backpropagation doesn't have to be implemented.

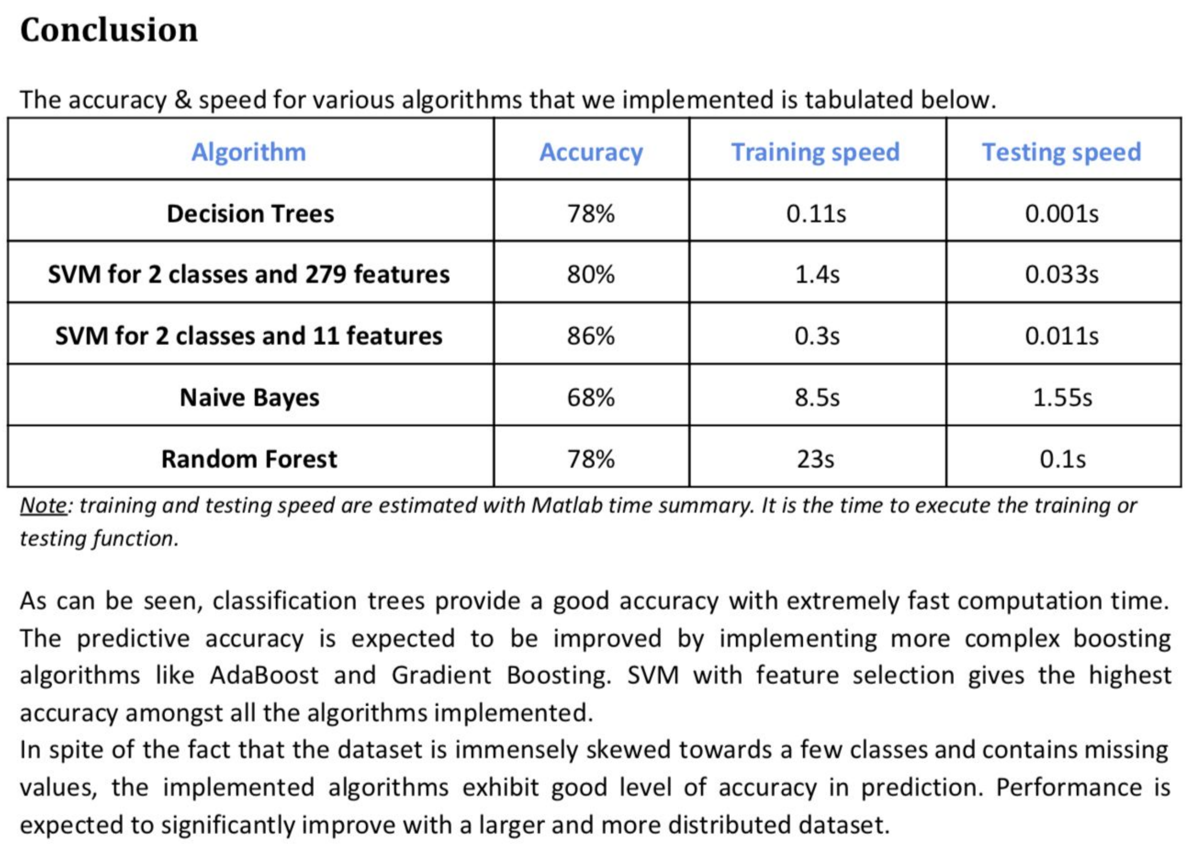
Next we comment on the dimensions of these parameters. Two of the dimensions are given by the dimension of the input vector ($D = 216000$) and the number of possible classifications for the output ($L = 2$). The other dimension, D_h, is the dimension associated with the hidden layer. i.e., $b^{(1)}$ is a vector of dimension D_h, $W^{(1)}$ is a $D_h \times D$ matrix, $b^{(2)}$ is a vector of dimension L, and $W^{(2)}$ is an $L \times D_h$ matrix. We chose $D_h = 500$.

Benchmark Model-Machine Learning

For the machine learning approach: Please refer to the following:

http://cs229.stanford.edu/proj2014/Manas%20Karandikar,%20Giulia%20Guidi,%20Classification%20Of%20Arrhythmia%20Using%20ECG%20Data.pdf

They have used some of the algorithms that I used (namely, Decision Trees and Naive Bayes along with SVM). though they are only predicting for few classes which are very well represented. I will try to represent it for every class first and see how the accuracy turns out. If required, classes with few or no instances will be discarded. Here, I have few screenshots from their paper detailing their results.



Conclusion

The accuracy & speed for various algorithms that we implemented is tabulated below.

Algorithm	Accuracy	Training speed	Testing speed
Decision Trees	78%	0.11s	0.001s
SVM for 2 classes and 279 features	80%	1.4s	0.033s
SVM for 2 classes and 11 features	86%	0.3s	0.011s
Naive Bayes	68%	8.5s	1.55s
Random Forest	78%	23s	0.1s

Note: training and testing speed are estimated with Matlab time summary. It is the time to execute the training or testing function.

As can be seen, classification trees provide a good accuracy with extremely fast computation time. The predictive accuracy is expected to be improved by implementing more complex boosting algorithms like AdaBoost and Gradient Boosting. SVM with feature selection gives the highest accuracy amongst all the algorithms implemented.
In spite of the fact that the dataset is immensely skewed towards a few classes and contains missing values, the implemented algorithms exhibit good level of accuracy in prediction. Performance is expected to significantly improve with a larger and more distributed dataset.

Evaluation Metric

The conventional approach of validation, i.e., dividing the data into train set and test set. I will use scikit-learn test-train-split to divide the dataset into train set (80%) and test set (20%). I will divide both the datasets in the same way.

For the DL approach, the trained model will be tested on the test case. I will plot the validation loss and accuracy. This is easier to do since the output keras model object has the hist property which easily plots it using the intrinsic matplotlib object of the IPython notebook.

For the ML approach, I will be using the prediction from test and train score. That is, I will fit the train data to get the trained model and will use it later to make prediction on the test set. The test set will output a score based on the percentage of right over all the test set instances (accuracy). This accuracy is the evaluating metric.

Project Design-DL

I will use a Convolution Neural Network based approach to solve the given problem.

Convolution neural networks are used owing to their speciality of extracting patterns in the data with the help of kernels which actually convolve to produce the output. These kernels when trained can detect important features in the data from the perspective of the problem at hand.

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. **During the forward pass of the training process, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and**

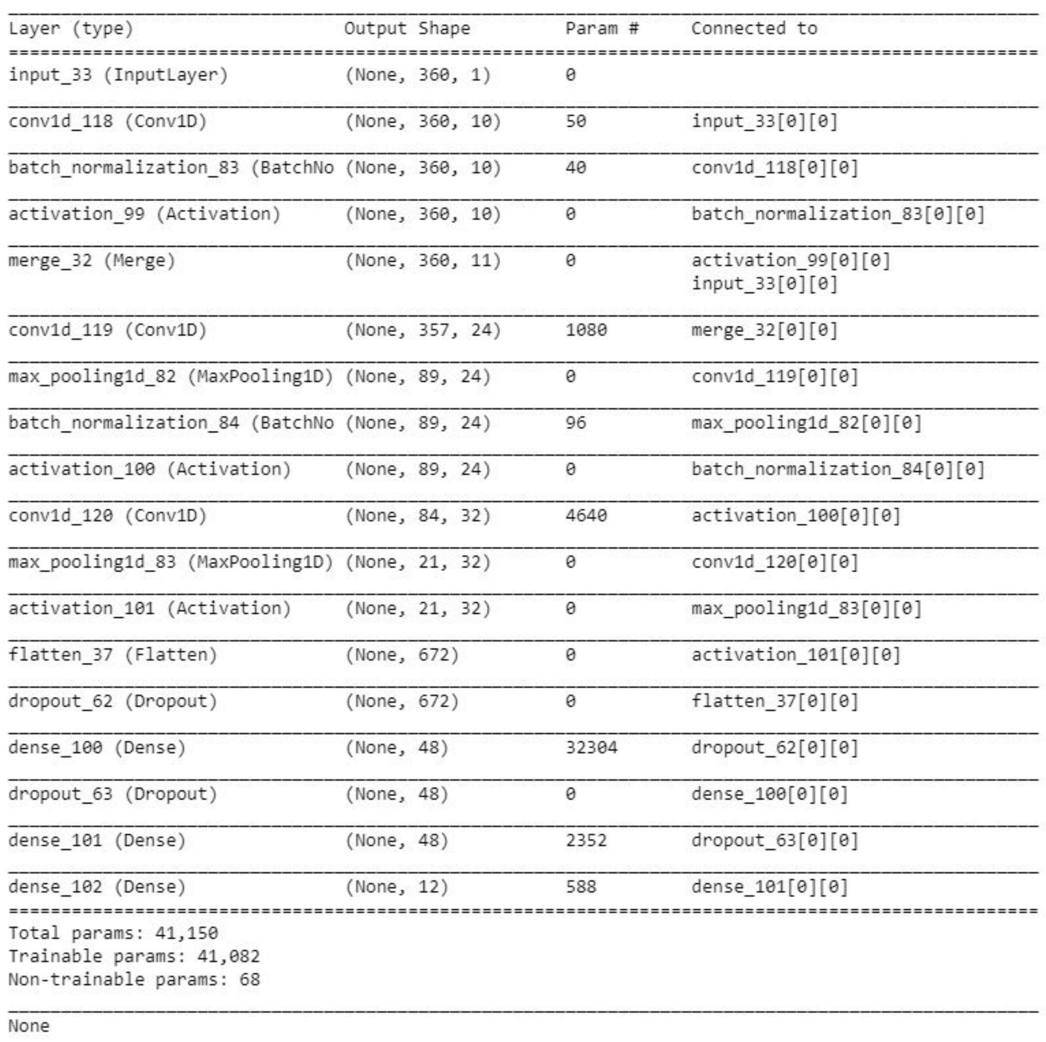
producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. **Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons** in the same activation map.

Max Pooling Layers, **Batch Normalisation Layers**,etc will be used. These layers have specific functionalities and can be found out from the web. They increase the robustness of the model by taking care of Spatial Invariance and also avoid overfitting of the model to the database. The goal is not to optimise to the dataset rather learn to find patterns which can actually help to detect problems in a new individual.

Dropouts will also be used to avoid overfitting.

Here, is a sample model that I tried to make in these 2 weeks:



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_33 (InputLayer)            (None, 360, 1)        0
____________________________________________________________________________________________________
conv1d_118 (Conv1D)              (None, 360, 10)       50          input_33[0][0]
____________________________________________________________________________________________________
batch_normalization_83 (BatchNo  (None, 360, 10)       40          conv1d_118[0][0]
____________________________________________________________________________________________________
activation_99 (Activation)       (None, 360, 10)       0           batch_normalization_83[0][0]
____________________________________________________________________________________________________
merge_32 (Merge)                 (None, 360, 11)       0           activation_99[0][0]
                                                                   input_33[0][0]
____________________________________________________________________________________________________
conv1d_119 (Conv1D)              (None, 357, 24)       1080        merge_32[0][0]
____________________________________________________________________________________________________
max_pooling1d_82 (MaxPooling1D)  (None, 89, 24)        0           conv1d_119[0][0]
____________________________________________________________________________________________________
batch_normalization_84 (BatchNo  (None, 89, 24)        96          max_pooling1d_82[0][0]
____________________________________________________________________________________________________
activation_100 (Activation)      (None, 89, 24)        0           batch_normalization_84[0][0]
____________________________________________________________________________________________________
conv1d_120 (Conv1D)              (None, 84, 32)        4640        activation_100[0][0]
____________________________________________________________________________________________________
max_pooling1d_83 (MaxPooling1D)  (None, 21, 32)        0           conv1d_120[0][0]
____________________________________________________________________________________________________
activation_101 (Activation)      (None, 21, 32)        0           max_pooling1d_83[0][0]
____________________________________________________________________________________________________
flatten_37 (Flatten)             (None, 672)           0           activation_101[0][0]
____________________________________________________________________________________________________
dropout_62 (Dropout)             (None, 672)           0           flatten_37[0][0]
____________________________________________________________________________________________________
dense_100 (Dense)                (None, 48)            32304       dropout_62[0][0]
____________________________________________________________________________________________________
dropout_63 (Dropout)             (None, 48)            0           dense_100[0][0]
____________________________________________________________________________________________________
dense_101 (Dense)                (None, 48)            2352        dropout_63[0][0]
____________________________________________________________________________________________________
dense_102 (Dense)                (None, 12)            588         dense_101[0][0]
====================================================================================================
Total params: 41,150
Trainable params: 41,082
Non-trainable params: 68
____________________________________________________________________________________________________
None
```

Project Design-ML

The UCI dataset has missing values. These will be filled using the Imputer Class. Filling can be done using either the 'most_frequent' metric or the 'median' metric. First the data will be ran through looking for '?' which determined the missing values. It will be converted to '0' (empty string) so that when I use Panda's data frame structure to represent it, it is converted to int64 type (reason I cannot use an alphabet is that it will be converted to an object which will be difficult to deal with). This DF can be fed to two pipelines respectively: Median or Most Frequent Pipeline. The median pipeline replaces the missing data with the median of the remaining data whereas the Most Frequent Pipeline replaces it with the mode of the data. After replacing, the data is scaled using Standard Scaling. This scales the data to unit variance after centering it. It is extremely important as some algorithms like k-nearest-Neighbours require scaled data (Some algorithms require only scaled data with regularizations like Random Forest).

After scaling the data, boosting is applied. Boosting helps improve the prediction by making sure that model learns from the mistakes it did before in the prediction and tries to rectify it. I have used two boosting measures which are AdaBoost and Gradient Boosting algorithms. I will also use bagging here. Then I will train various ML algorithms like Stochastic Gradient Descent, Random Forest, Logistic Regression, Single Vector Machines, linear SVM, k Nearest Neighbors and Extra trees. These will be fed to Ensembler learning which will then predict on the test set.