Object recognition with ultrasonic sensors using LSTM

Autonomous Intelligent Systems SL-2
M.Eng Information Technology
Guide: Prof Pech , Prof. Nauth,Mr. Michalik
Submitted By :Yash Vyas-1266490
vyas@stud.fra-uas.de

ACKNOWLEDGMENT

I would like to thank Prof. Pech, Prof Nauth and Mr Michalik ,for the continuous support and guidance provided for this project .Furthermore, I would also like to put my sincere gratitude for providing all the necessary information and useful links via moodle which proved to be very helpful for successful completion of this project

Contents

Figures

***Abstract*— The topic of following project is to develop automotive sensor object recognition system using Redpitaya and Raspberry Pi. Following project is basically using sonar echo for object detection and Machine Learning algorithm for object recognition. Sonar echo are generated by SRF02 Ultrasonic range finder embedded on redpitaya and using LSTM neural network which is programmed on Raspberry Pi to recognize echo reflected back from three objects: i.e. Wall, Human, Car**

Keywords—Sonar echo, Raspberry Pi, Redpitaya, LSTM , SRF02 sensor,

I. INTRODUCTION

In the present age of advance technology Sensors and Machine Learning combination can be used for solving many traditional problems which can provide better security and will decrease loss in terms of health, wealth and energy. Following project is implementation of autonomous intelligent system which can be used in automotive industry for object recognition which can be advantageous and lifesaving.

In this project we have used sonar echo's reflected back from object to recognize the object with the help of LSTM neural network. Following project's LSTM network are trained to recognise three objects i.e. Car, Wall, Human but the network can be further be trained to recognise many object and fully established model can be used in automotive industry for better safety

II. AUTONOMUS INTELLIGENT SYSTEM DETAILS

A. *Hardware Details:*

Hardware system consist of an inexpensive ultrasonic range finder [SRF02] in a small footprint PCB embedded with Redpitaya sytem communicating with I2C protocol. The SRF02 uses a single transducer for both transmission and reception,. The minimum measurement range varies from around 17-18cm (7 inches) on a warm day down to around 15-16cm (6 inches) on a cool day, SRF02 can measure distance in uS, cm or inches

Sr.No	SRF 02 Specification
1	Range 16 cm to 6m
2	Power : 5v, 4mA , Typ
3.	Frequency: 40KHz
4.	Size: 24mm x 20 mm x 17mm height
5.	Analogue Gain : Automatic 64 step gain control
6.	Connection Modes: 1 - Standard I2C Bus 2 - Serial Bus
7.	Full Automatic Tuning: No calibration
8.	Timing: Fully timed echo, freeing host controller of task.
9.	Units: Range reported in uS, mm or inches
10.	Light Weight: 4.6gm

SRF02 Sensor Circuit Details

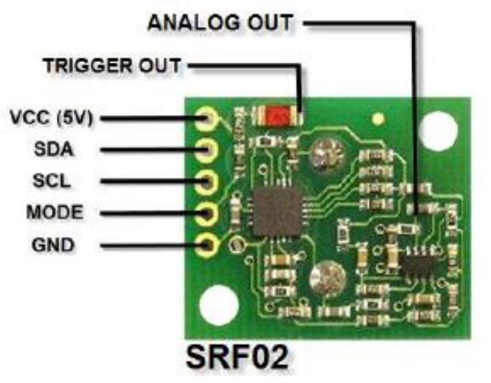


Figure 2 SRF02 Circuit Details

RedPitaya System in Detail:

Red Pitaya STEMlab is a flexible, open source, reliable,

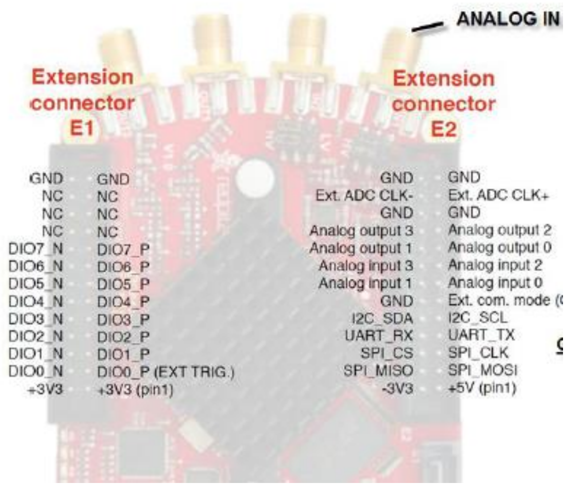
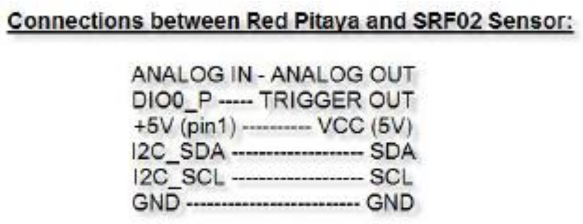


Figure 1 RedPitaya Details

fast data acquisition board.

During the I2C communication between Redpitya system and sensor, sensor starts ranging and Redpitaya is acquiring signal coming from the sensor. In parallel the internal SCPI server started automatically and data is acquired from the system using SCPI commands



<u>Connections between Red Pitaya and SRF02 Sensor:</u>

ANALOG IN - ANALOG OUT
DIO0_P ----- TRIGGER OUT
+5V (pin1) ---------- VCC (5V)
I2C_SDA ------------------- SDA
I2C_SCL ------------------- SCL
GND ------------------------- GND

B. *System Architecture*

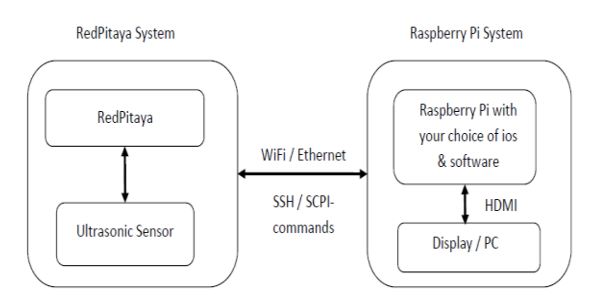


Figure 5 Basic architecture of System

C. *Project flow:*

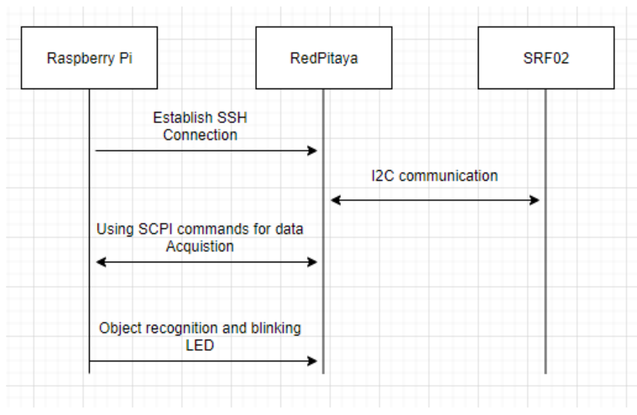

Raspberry Pi
RedPitaya
SRF02
Establish SSH Connection
I2C communication
Using SCPI commands for data Acquistion
Object recognition and blinking LED

Figure 4 Basic Project Flow

Data acquired from Redpitaya is used as time series input for the trained LSTM neural network model and after the classification of the signal data SCPI commands are used for the LED blinking and showing output on Redpitaya system.

III. Data Acquisition and Feature Extraction

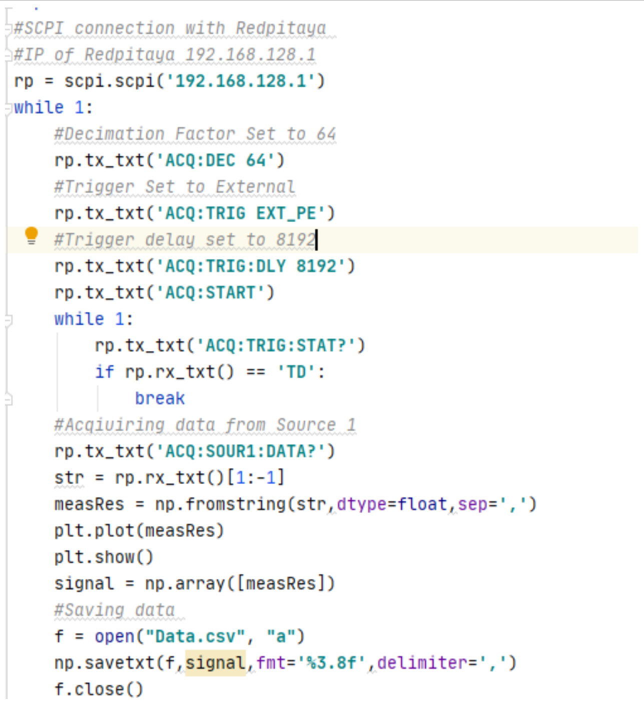
A. *Data Acquisition*

SRF02 sensor generates Ultrasonic signal which is reflected back by the object in front of sensor and received by the sensor .For the data acquisition from the Redpitaya system python script is prepared which acquires the signal data using SCPI command which are used to access redpitaya system in remote controlled mode after establishing SSH connection between RaspberryPi and Redpitaya. Redpitaya system's Signal acquisition on external trigger mode is used which provides the signal acquired by the sensor which was received by the sensor after reflection.

Sr.No	RedPitaya Configuration
1	Signal Acquisition : External Trigger
2	Decimation Factor : 64
3	Sampling Rate : 1.9 MS/s
4	Samples : 16834
5.	Timescale: 8.389 ms

Used Redpitaya SCPI command details:

Sr. No	SCPI Command Description	
1	ACQ:DEC <decimation>	Set decimation factor.
2.	SOUR<n>:TRIG:SOUR <trigger>	Set trigger source for selected signal.
3.	ACQ:TRIG:DLY <time>	Set trigger delay in samples.
4.	ACQ:START	Starts acquisition.
5	ACQ:TRIG:STAT?	Get trigger status. If DISABLED -> TD else WAIT.
6.	ACQ:SOUR<n>:DATA?	Read full buffer



```
#SCPI connection with Redpitaya
#IP of Redpitaya 192.168.128.1
rp = scpi.scpi('192.168.128.1')
while 1:
    #Decimation Factor Set to 64
    rp.tx_txt('ACQ:DEC 64')
    #Trigger Set to External
    rp.tx_txt('ACQ:TRIG EXT_PE')
    #Trigger delay set to 8192
    rp.tx_txt('ACQ:TRIG:DLY 8192')
    rp.tx_txt('ACQ:START')
    while 1:
        rp.tx_txt('ACQ:TRIG:STAT?')
        if rp.rx_txt() == 'TD':
            break
    #Acqiuiring data from Source 1
    rp.tx_txt('ACQ:SOUR1:DATA?')
    str = rp.rx_txt()[1:-1]
    measRes = np.fromstring(str,dtype=float,sep=',')
    plt.plot(measRes)
    plt.show()
    signal = np.array([measRes])
    #Saving data
    f = open("Data.csv", "a")
    np.savetxt(f,signal,fmt='%3.8f',delimiter=',')
    f.close()
```

Figure 6 Python Script for data acquisiton

B. *Feature Extraction*

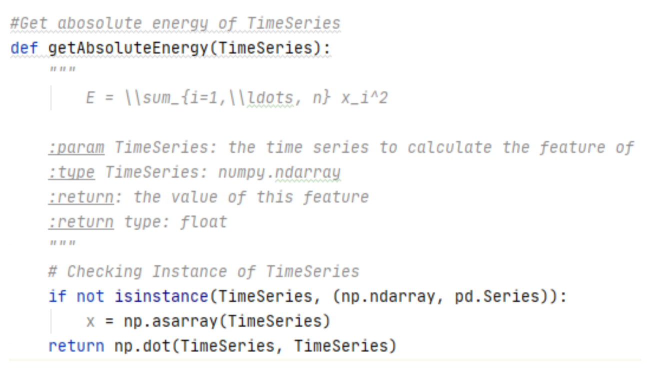
From the time series data different features are extracted like Absolute Energy, Skewness, Sum values over signal, FFT values. Below all the extraction processes are explained and data plot are show for the following.

1) *Absolute Energy:*

The power of a signal is the sum of the absolute squares of its time-domain samples divided by the signal length, or, equivalently, the square of its RMS level.

Mathematical Formulae:

$$E = \int_{-x}^{x} |y(t)|^2 \, dt$$ **Equation 1**



```
#Get abosolute energy of TimeSeries
def getAbsoluteEnergy(TimeSeries):
    """
        E = \\sum_{i=1,\\ldots, n} x_i^2

    :param TimeSeries: the time series to calculate the feature of
    :type TimeSeries: numpy.ndarray
    :return: the value of this feature
    :return type: float
    """
    # Checking Instance of TimeSeries
    if not isinstance(TimeSeries, (np.ndarray, pd.Series)):
        x = np.asarray(TimeSeries)
    return np.dot(TimeSeries, TimeSeries)
```

Figure 8 Absolute Energy Function

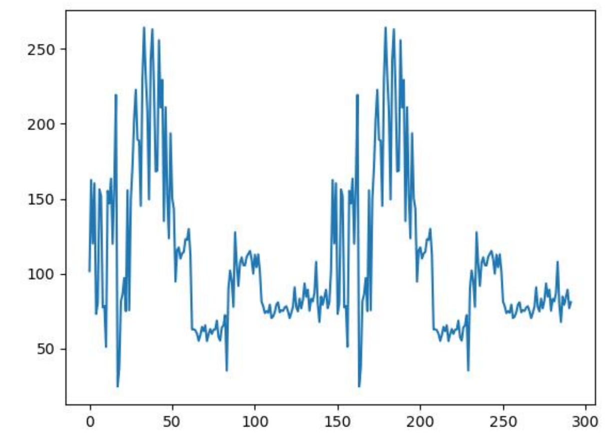


Figure 7 AbsEnergy of Signal reflected back from Wall

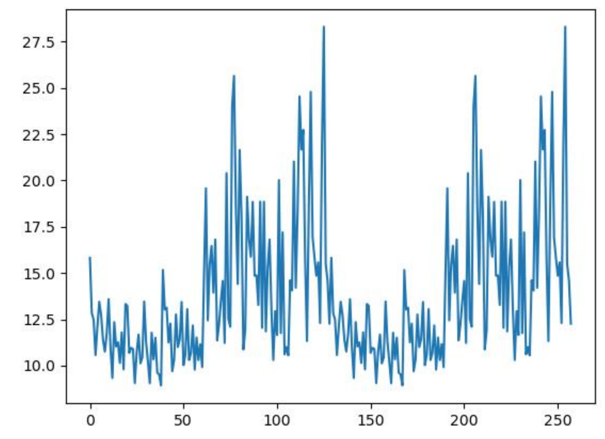


Figure 9 AbsEnergy of Signal reflected back from Human

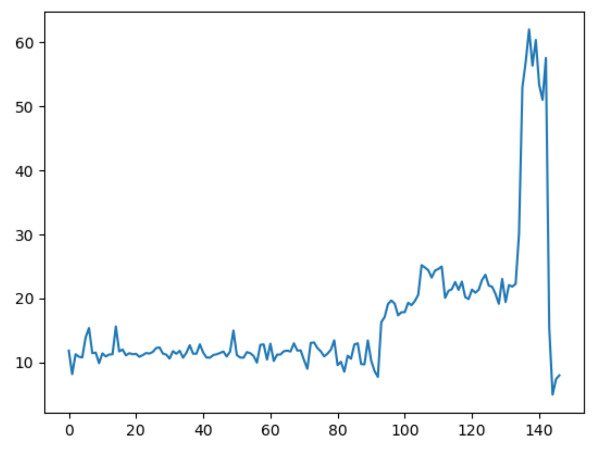


Figure 10 AbsEnergy of Signal reflected back from Car

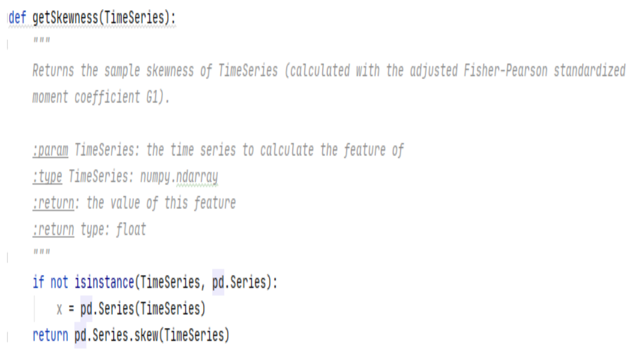
2) *Skewness of TimeSeries*

Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution.

Mathematical Formulae:

$$G_1 = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s} \right)^3$$ **Equation 2**

Adjusted Fisher-Pearson standardized moment coefficient is used to find skewness



```
def getSkewness(TimeSeries):
    """
    Returns the sample skewness of TimeSeries (calculated with the adjusted Fisher-Pearson standardized
    moment coefficient G1).

    :param TimeSeries: the time series to calculate the feature of
    :type TimeSeries: numpy.ndarray
    :return: the value of this feature
    :return type: float
    """
    if not isinstance(TimeSeries, pd.Series):
        x = pd.Series(TimeSeries)
    return pd.Series.skew(TimeSeries)
```

Figure 11 Calculate Skewness Function for TimeSeries

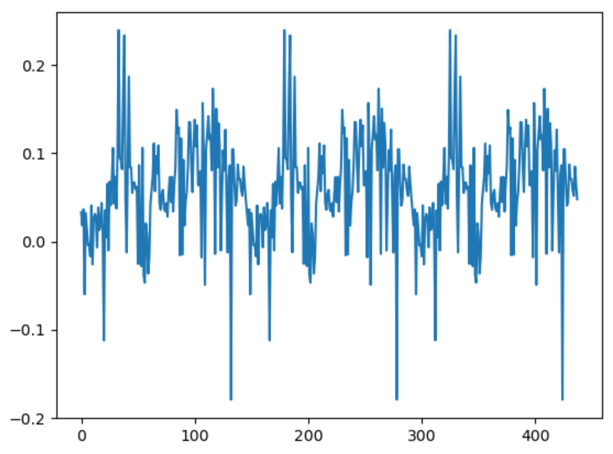
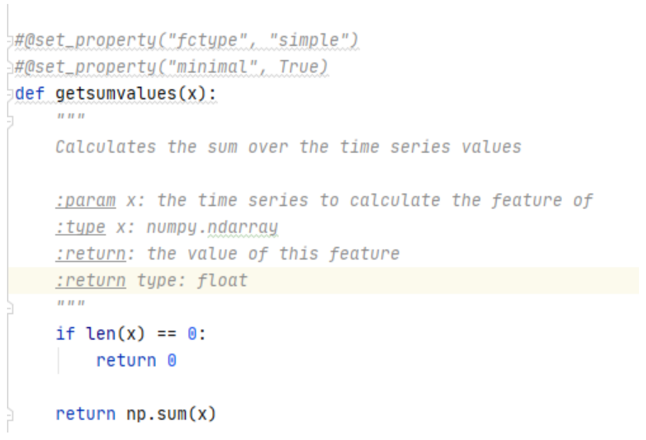


Figure 12 Skewness of Signal reflected from Wall



```
#@set_property("fctype", "simple")
#@set_property("minimal", True)
def getsumvalues(x):
    """
    Calculates the sum over the time series values

    :param x: the time series to calculate the feature of
    :type x: numpy.ndarray
    :return: the value of this feature
    :return type: float
    """
    if len(x) == 0:
        return 0

    return np.sum(x)
```

Figure 16 Get Sum values function

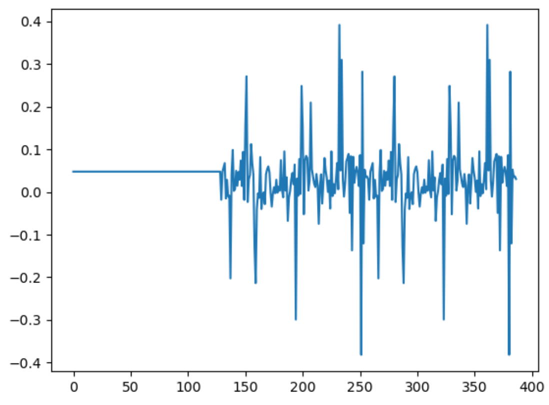


Figure 13 Skewness of Signal reflected from Human

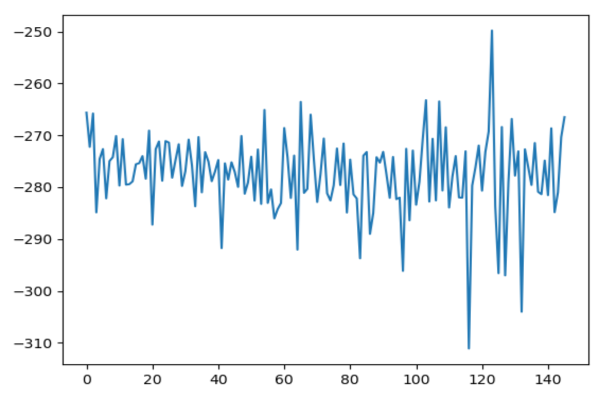


Figure 15 Sum values over Signal reflected from Wall

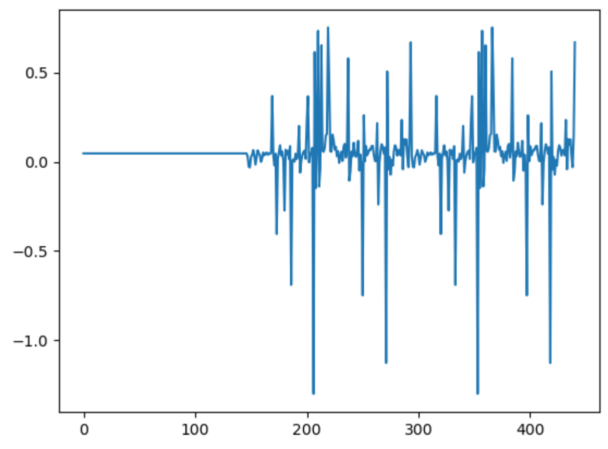


Figure 14 Skewness of Signal reflected from Car

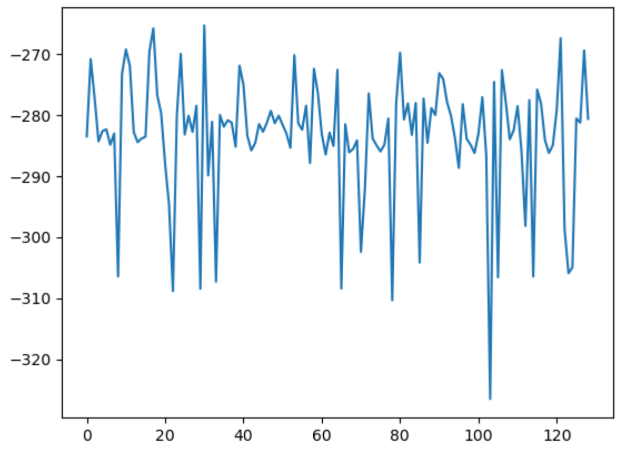


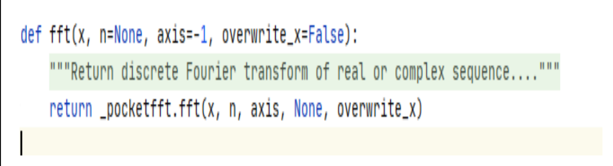
Figure 17 Sum values over Signal reflected from Human

3) Calculate the sum over Timeseries

4) *Fourier Transform of TimeSeries Data:*

From calculating FFT transformation scipy library fft function is used A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa

For Calulating FFT scipy library is used in the following project



```
def fft(x, n=None, axis=-1, overwrite_x=False):
    """Return discrete Fourier transform of real or complex sequence...."""
    return _pocketfft.fft(x, n, axis, None, overwrite_x)
```

Figure 19 Scipy Library FFT function

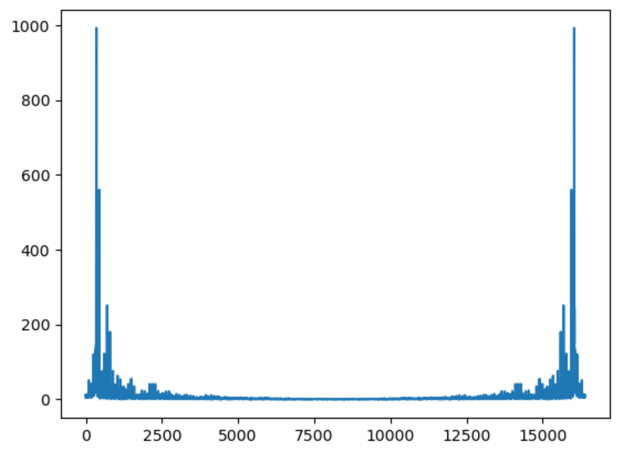


Figure 18 FFT of signal reflected from Car

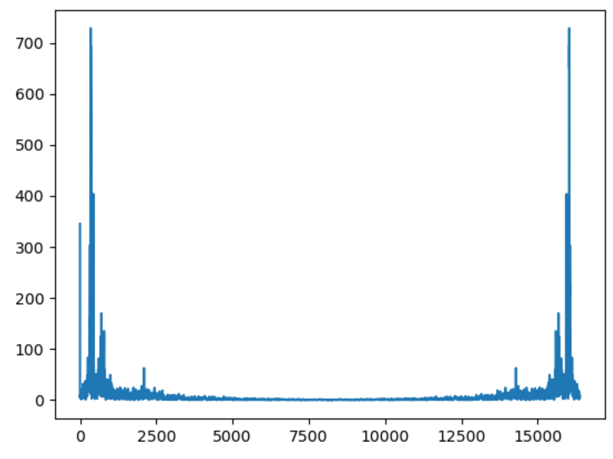


Figure 20 FFT of signal reflected from Wall

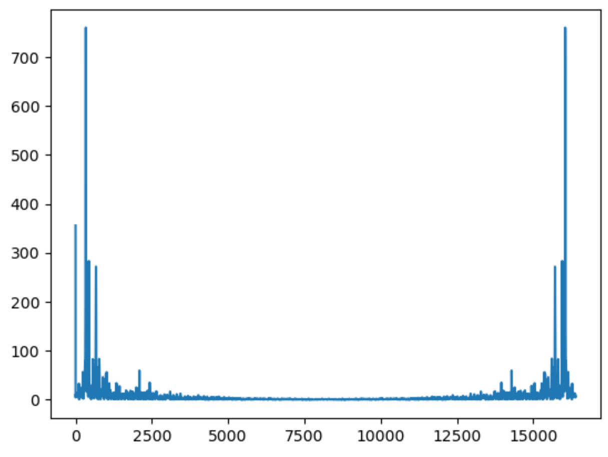


Figure 21 FFT of signal reflected from Human

IV. Long Short Term Memory

A. *Introduction*

Recurrent neural networks with Long Short-Term Memory (which we will concisely refer to as LSTMs) have emerged as
an effective and scalable model for several learning problems related to sequential data. Earlier methods for attacking these
problems have either been tailored towards a specific problems or did not scale to long time dependencies. LSTMs on the
other hand are both general and effective at capturing longterm temporal dependencies. They do not suffer from the
optimization hurdles that plague simple recurrent networks [1] [2]

B. *History of LSTM*

LSTM was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber. [3] By introducing Constant Error Carousel (CEC) units, LSTM deals with the vanishing gradient problem. The initial version of LSTM block included cells, input and output gates. By introducing Constant Error Carousel (CEC) units, LSTM deals with the vanishing gradient problem. The initial version of LSTM block included cells, input and output gates [4]. In 1999, Felix Gers and his advisor Jürgen Schmidhuber and Fred Cummins introduced the forget gate (also called “keep gate”) into LSTM architecture,[5] enabling the LSTM to reset its own state. [4] In 2000, Gers & Schmidhuber & Cummins added peephole connections (connections from the cell to the gates) into the architecture.[4] Additionally, the output activation function was omitted

C. *Idea*

In theory, classic (or "vanilla") RNNs can keep track of arbitrary long-term dependencies in the input sequences.

The problem of vanilla RNNs is computational (or practical) in nature: when training a vanilla RNN using back-propagation, the gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the process, which use finite-precision numbers. RNNs using LSTM units partially solve the vanishing gradient problem, because LSTM units allow gradients to also flow unchanged. However, LSTM networks can still suffer from the exploding gradient problem.

D. Architecture

There are several architectures of LSTM units. A common architecture is composed of a cell (the memory part of the LSTM unit) and three "regulators", usually called gates, of the flow of information inside the LSTM unit: an input gate, an output gate and a forget gate. Some variations of the LSTM unit do not have one or more of these gates or maybe have other gates [7] Briefly, the cell is responsible for keeping track of the dependencies between the elements in the input sequence. The input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function of the LSTM gates is often the logistic sigmoid function.

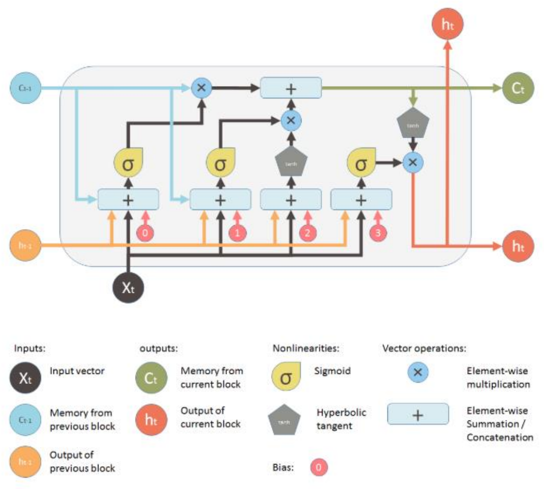


Figure 22 LSTM cell

Figure 23 is referenced from [25]

Brief explanation about LSTM cell gates:

1) Forget Gate:

This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

Forget factor f is calculated as

$$f_t = \sigma(W_f[STM_{t-1}, E_t] + b_f)$$ **Equation 3**

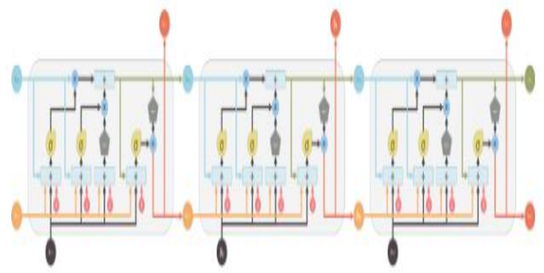


Figure 23 LSTM network

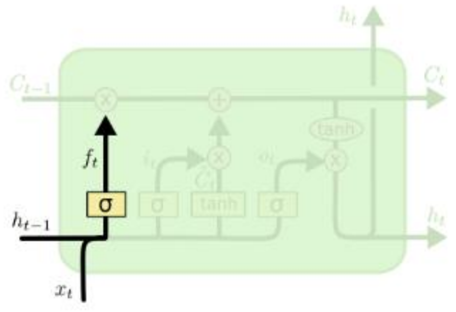


Figure 24 Forget Gate

2) Input Gate:

To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. We also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then we multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output

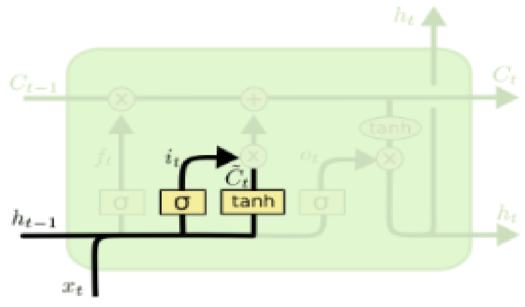


Figure 25 Input Gate

Input Gate equation:

$i_t = \sigma(W_i[h_{t-1} x_t] + b_i)$**Equation 4**

Input Gate Modulation Equation:

$c_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$ **Equation 5**

3) Output Gate:

The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

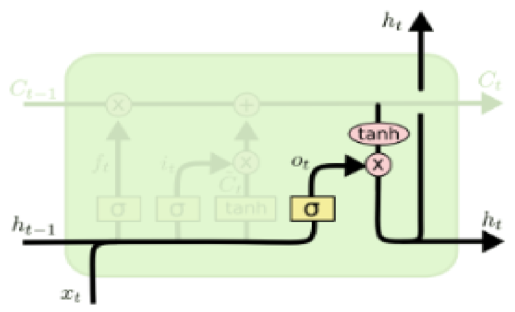


Figure 26 Output Gate

Output Gate Equation:

$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$**Equation 6**

$h_t = o_t * \tanh(C_t)$ **Equation 7**

4) Different topologies of LSTM-RNN:

Bidirectional LSTM, Grid LSTM, Stacked LSTM, Multi Dimensional LSTM, Grid LSTM blocks, Gated Recurrent Unit.

E. Training of LSTM network

In order to preserve the CEC in LSTM memory block cells, the original formulation of LSTM used a combination of two learning algorithms: BPTT to train network components located after cells, and RTRL to train network components located before and including cells. The latter units work with RTRL because there are some partial derivatives (related to the state of the cell) that need to be computed during every step, no matter if a target value is given or not at that step. For now, we only allow the gradient of the cell to be propagated through time, truncating the rest of the gradients for the other recurrent connections. We define discrete time steps in the form ts = 1; 2; 3; :::. Each step has a forward pass and a backward pass; in the forward pass the output/activation of all units are calculated, whereas in the backward pass, the calculation of the error signals for all weights is performed [3]

1) Forward Pass

Let M be the set of memory blocks. Let mc be the c-th memory cell in the memory block m, and W[u;v] be a weight connecting unit u to unit v. In the original formulation of LSTM, each memory block m is associated with one input gate inm and one output gate outm. The internal state of a memory cell mc at time + 1 is updated according to its state smc ( ) and according to the weighted input zmc (+ 1) multiplied by the activation of the input gate yinm(+ 1). Then, we use the activation of the output gate zoutm(+ 1) to calculate the activation of the cell ymc (+ 1).The activation yinm of the input gate inm is computed as

$y_{inm}(\tau + 1) = f_{inm}(z_{inm}(\tau + 1))$**Equation 8**

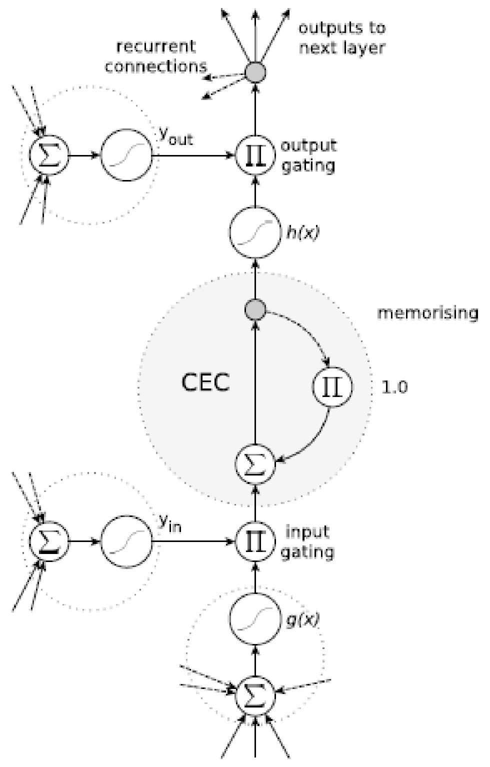


Figure 27 LSTM cell

Fig 28 describes A standard LSTM memory block. The block contains (at least) one cell with a recurrent self-connection (CEC) and weight of `1'. The state of the cell is denoted as sc. Read and write access is regulated by the input gate, yin, and the output gate, y(out). The internal cell state is calculated by multiplying the result of the squashed input, g, by the result of the input gate, y(in), and then adding the state of the last time step, Sc(t - 1). Finally, the cell output is calculated by multiplying the cell state, Sc, by the activation of the output gate, y(out). [5]

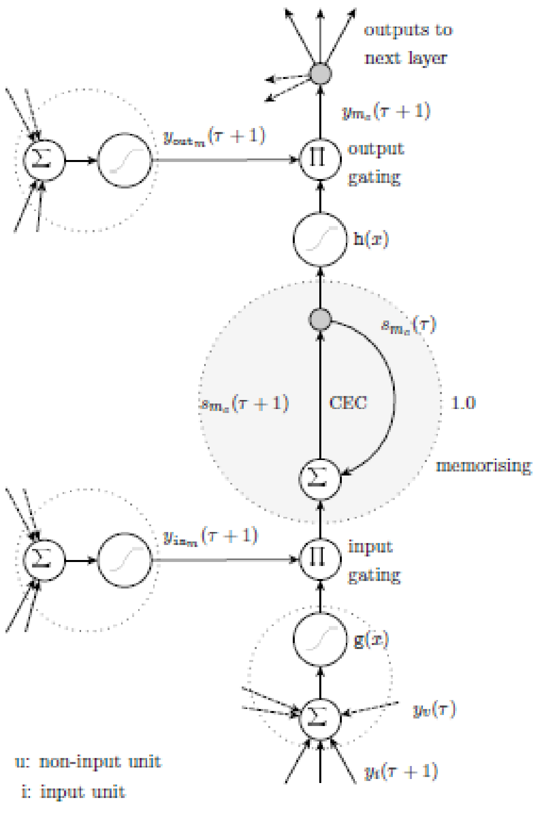


Figure 29 Forward Pass second phase

Fig 29 describes A standard LSTM memory block. The block contains (at least) one cell with a recurrent self-connection (CEC) and weight of `1'. The state of the cell is denoted as sc . Read and write access is regulated by the input gate, yin , and the output gate, yout . The internal cell state is calculated by multiplying the result of the squashed input, g(x), by the result of the input gate and then adding the state of the current time step, smc (r), to the next, smc (r + 1). Finally, the cell output is calculated by multiplying the cell state by the activation of the output gate.

With the input gate input

$$z_{inm}(\tau+1) = \sum_{u} W_{[inm,u]} X_{[u,inm]}(\tau+1), with\ u \in Pre(in_m)$$

$$= \sum_{v \in U} W_{[inm,v]} y_v(\tau) + \sum_{i \in I} W_{[inm,i]} y_i(\tau+1)$$

Equation 9

The activation of the output gate outm is

$$y_{outm}(\tau+1) = f_{outm}(z_{outm}(\tau+1))$$ **Equation 10**

with the output gate input

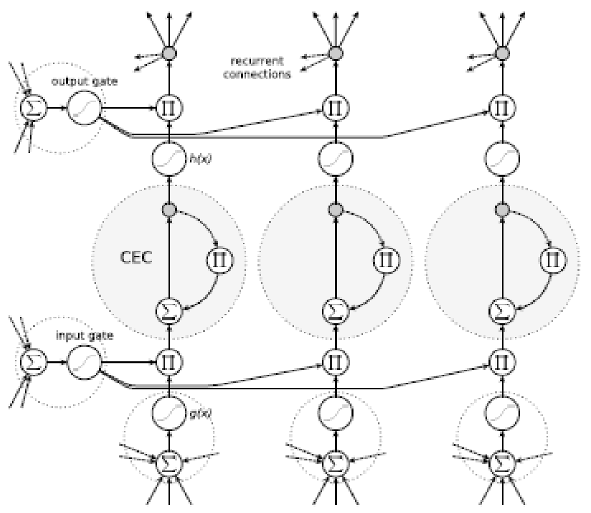


Figure 28 A three cell LSTM memory block with recurrent self-connections

$$z_{outm}(\tau+1) = \sum_{u} W_{[outm,u]} X_{[u,outm]}(\tau+1), with\ u \in Pre(out_m)$$

$$= \sum_{v \in U} W_{[outm,v]} y_v(\tau) + \sum_{i \in I} W_{[outm,i]} y_i(\tau+1)$$

Equation 11

The results of the gates are scaled using the non-linear squashing function

f(in) = f(outm)=f defined by

$$f(s) = \frac{1}{1+e^{-s}}$$ **Equation 12**

So that they will be in range[0,1].Thus the input for the memory cell will only be able to pass if the signal at the input gate is sufficiently close to ‘1’.

For a memory cell mc in the memory block m , the weight input zmc(t+1) is defined by

$$z_{mc}(\tau+1) = \sum_{u} W_{[mc,u]} X_{[u,mc]}(\tau+1), \qquad with\ u \in Pre(m_c)$$

$$= \sum_{v \in U} W_{[mc,v]} y_v(\tau) + \sum_{i \in I} W_{[mc,i]} y_i(\tau+1)$$

Equation 13

As we mentioned before, the internal state sm(t+1) of the unit in the memory cell at time (t+1) is computed differently;the weighted input is squashed and then multiplied by the activation of the input gate, and then the state of the last timestep is added. The corresponding eq is

$$s_{mc}(\tau+1) = s_{mc}(\tau) + y_{inm}(\tau+1) g(z_{mc}(\tau+1))$$ **Equation 14**

With smc = 0 and the non linear squashing function for the cell input

$$g(z) = \frac{4}{1+e^{-z}} - 2$$ **Equation 15**

Which in this case the result is in range [-2,2]

Then output ym is calculated by squashing and multiplying cell state smc by the activation of the output gate yout:

$$y_{mc}(\tau+1) = y_{outm}(\tau+1)h(s_m(\tau+1))$$ **Equation 16**

With the non linear squashing function with range [-1,1]

$$h(z) = \frac{2}{1+e^{-z}} - 1$$ **Equation 17**

Assuming a layered recurrent neural network with standard input,output and hidden layer consisting of memory blocks, the activation of the output unit o is computed as

$$y_o(\tau+1) = \sum_{u \in U-G} W_{[o,u]} y_u(\tau+1)$$ **Equation 18**

Where G is the set of Gate units, and we can again use the logistic sigmoid in Equation 12 as squashing function fo

2) *Backward Pass*

LSTM incorporates elements from both BPTT and RTRL. Thus, we separate units into two types: those units whose weight changes are computed using a variation of BPTT (i.e, output units, hidden units, and the output gates), and those whose weight changes are computed using a variation of RTRL (i.e., the input gates, the forget gates and the cells). Following the notation used in previous section overall network error can be defined at timestep t is

$$E(\tau) = \frac{1}{2} \sum_{\sigma \in O} \left(d_o(\tau) - y_o(\tau)\right)^2$$ **Equation 19**

Let us consider units that work with BPTT.We define the notion of individual error of a unit u at time t by

$$v_u(\tau) = -\frac{\delta E(\tau)}{\delta z_u(\tau)}$$ **Equation 20**

Where zu is the weight input of the unit.We can expand the notion of weight contribution as follows

$$\Delta W_{[u,v]}(\tau) = -\frac{\eta \delta E(\tau)}{\delta W_{[u,v]}}$$

$$= -\eta \frac{\partial E(\tau)}{\partial W_{[u,v]}}$$

$$= -\eta \frac{\partial E(\tau)}{\partial z_u(\tau)} \frac{\partial z_u(\tau)}{\partial W_{[u,v]}}$$ **Equation 21**

V. Applications of LSTM-RNN

Brief examples of application of LSTM given here with the projects that have been completed by the respective authors.

1) *Speech Recognition*

In 2003 good results applying standard LSTM-RNN networks with a mix of LSTM and sigmoidal units to speech recognition tasks were obtained by [23, 24]

2) *Handwriting Recognition*

In 2007 [20] introduced BLSTM-CTC and applied it to online handwriting recognition, with results later outperforming Hidden-Markov-based recognition systems presented by [10]. [11] combined BLSTM-CTC with a probabilistic language
model and by this developed a system capable of directly transcribing raw online handwriting data

3) *Machine Translation*

The RNN Encoder-Decoder architecture is based on an approach communicated by [48]. A very similar deep LSTM architecture, referred to as sequence-to-sequence learning, was investigated by [68] con_rming these results [53] addressed the rare word problem using sequence-to-sequence, which improves the ability to translate words not in the vocabulary

4) *Image Processing*

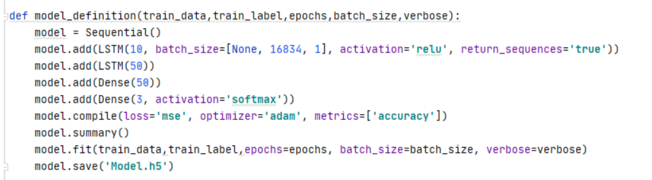
In 2015 the more recent LSTM variant using the Sequence-to-Sequence framework was successfully trained by [73, 79] to generate natural sentences in plain English describing images. Also in 2015 [14] the authors combined LSTMs with a deep hierarchical visual feature extractor and applied the model to image interpretation and lassi_cation tasks, like activity recognition and image/video description.

5) *TimeSeries Forecasting*
6) *TimeSereies classification*
7) *Feature Extraction*
8) *Predicion Modelling for sequential data*

VI. LSTM Network and Model used in Project

A. *LSTM network*

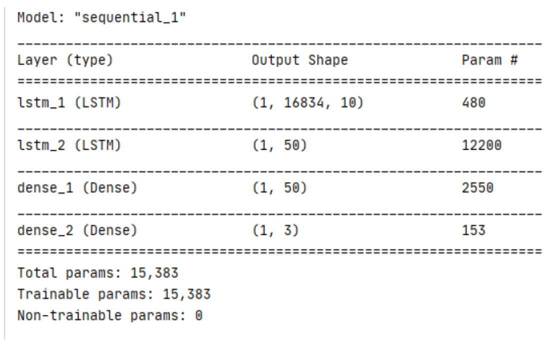
For the implenation of LSTM network I have used Keras Library.
Our model has 4 layers:LSTM,LSTM,Dense,Dense
On the first layer we have used ‘Relu’ as input function and on the last layer we have used softmax as we are developing the model for classification.



```
def model_definition(train_data,train_label,epochs,batch_size,verbose):
    model = Sequential()
    model.add(LSTM(10, batch_size=[None, 16834, 1], activation='relu', return_sequences='true'))
    model.add(LSTM(50))
    model.add(Dense(50))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
    model.summary()
    model.fit(train_data,train_label,epochs=epochs, batch_size=batch_size, verbose=verbose)
    model.save('Model.h5')
```

Figure 30 Model Architecture

Number of layers in the model with the proper activation function plays a crucial role for any specific problem.
For the following time sequence classification problem 2 LSTM and 2 Dense are providing accuracy within 85-99 %.
Over addition of layers can also make model to overfir which should also be taken into account.
In Fig 31 we can see the output shape of each layer with the number of parameters which will be trained within the network

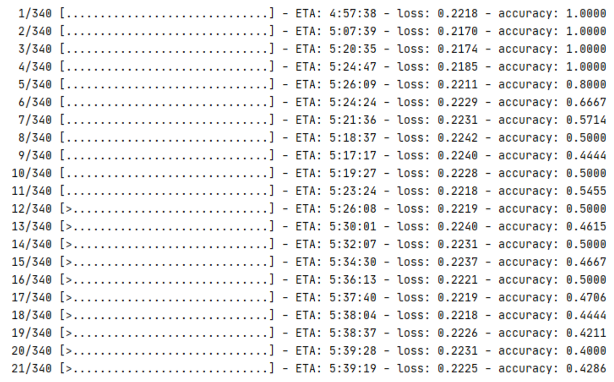


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (1, 16834, 10)            480
_________________________________________________________________
lstm_2 (LSTM)                (1, 50)                   12200
_________________________________________________________________
dense_1 (Dense)              (1, 50)                   2550
_________________________________________________________________
dense_2 (Dense)              (1, 3)                    153
=================================================================
Total params: 15,383
Trainable params: 15,383
Non-trainable params: 0
```

Figure 31 Model Summary

B. *Training of Model*

During the training we have noticed that the accuracy of model increases with the size of training dataset. Epoch also have effect on the accuracy till certain extent. Our training dataset is having one feature per timestep i.e. timeseries data received form the redpitaya so that faster classification can be done within the project and less computation time is required so that project can be used in real life scenarios.



```
  1/340 [..............................] - ETA: 4:57:38 - loss: 0.2218 - accuracy: 1.0000
  2/340 [..............................] - ETA: 5:07:39 - loss: 0.2170 - accuracy: 1.0000
  3/340 [..............................] - ETA: 5:20:35 - loss: 0.2174 - accuracy: 1.0000
  4/340 [..............................] - ETA: 5:24:47 - loss: 0.2185 - accuracy: 1.0000
  5/340 [..............................] - ETA: 5:26:09 - loss: 0.2211 - accuracy: 0.8000
  6/340 [..............................] - ETA: 5:24:24 - loss: 0.2229 - accuracy: 0.6667
  7/340 [..............................] - ETA: 5:21:36 - loss: 0.2231 - accuracy: 0.5714
  8/340 [..............................] - ETA: 5:18:37 - loss: 0.2242 - accuracy: 0.5000
  9/340 [..............................] - ETA: 5:17:17 - loss: 0.2240 - accuracy: 0.4444
 10/340 [..............................] - ETA: 5:19:27 - loss: 0.2228 - accuracy: 0.5000
 11/340 [..............................] - ETA: 5:23:24 - loss: 0.2218 - accuracy: 0.5455
 12/340 [>.............................] - ETA: 5:26:08 - loss: 0.2219 - accuracy: 0.5000
 13/340 [>.............................] - ETA: 5:30:01 - loss: 0.2240 - accuracy: 0.4615
 14/340 [>.............................] - ETA: 5:32:07 - loss: 0.2231 - accuracy: 0.5000
 15/340 [>.............................] - ETA: 5:34:30 - loss: 0.2237 - accuracy: 0.4667
 16/340 [>.............................] - ETA: 5:36:13 - loss: 0.2221 - accuracy: 0.5000
 17/340 [>.............................] - ETA: 5:37:40 - loss: 0.2219 - accuracy: 0.4706
 18/340 [>.............................] - ETA: 5:38:04 - loss: 0.2218 - accuracy: 0.4444
 19/340 [>.............................] - ETA: 5:38:37 - loss: 0.2226 - accuracy: 0.4211
 20/340 [>.............................] - ETA: 5:39:28 - loss: 0.2231 - accuracy: 0.4000
 21/340 [>.............................] - ETA: 5:39:19 - loss: 0.2225 - accuracy: 0.4286
```

Figure 33 Model Training

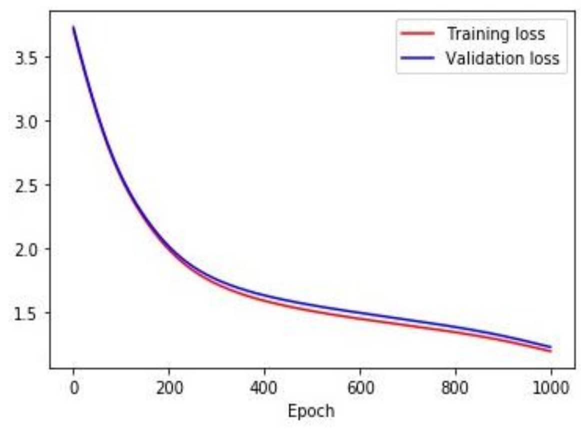
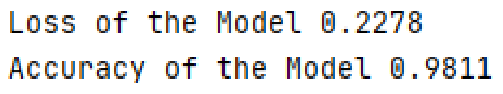


Figure 32 Training Loss vs Epoch

C. *Testing of Model and Accuracy*

For testing of the model we have used total 144 timeseries data which includes data of all three objects i.e. Wall ,Car, Human. Following solution does not depend on distance of the object from the sensor as we are directly using timeseries data for the modelling of the network. Here I have used one feature for the training of the model but the model can be upgraded by using multiple feature for the single time series data which will require higher computational power. Model accuracy increases by increment of neurons in the hidden layers of the network. Activation functions softmax is used in the final layer because following model was tailored for the specific classification problem and softmax gives the final class of the timeseries data



```
Loss of the Model 0.2278
Accuracy of the Model 0.9811
```

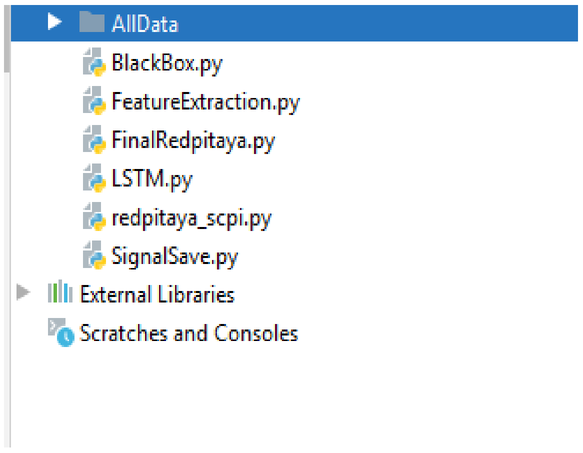
Figure 34 Final Loss and Accuracy of the Model

VII. PROJECT SPECIFICATIONS

1)Libararies Used and specifications

Python 3.7
Keras: Models[sequential], Layers[LSTM,Dense], Utils,
Tensorflow,
Numpy
Scipy

2)Project File Structure



AllData
BlackBox.py
FeatureExtraction.py
FinalRedpitaya.py
LSTM.py
redpitaya_scpi.py
SignalSave.py
External Libraries
Scratches and Consoles

Figure 35 File Structure

3)Feature List:Absolute_Energy,Skewness,FFT,Sum_over_timeseries

VIII. Manual for Using Project

1. To establish ssh connection between Red Pitaya and Raspberry Pi :

1.1. Create a shell script “{AIS-SL2-1266490}.sh” on Raspberry Pi using “ssh” command.

1.2. Use username, password and IP address of Red Pitaya in “ssh” command.

1.3. Enable executable rights to the user for this script file.

1.4. Execute this shell script. This will establish the required ssh connection.

2. To send SCPI commands from Raspberry Pi for data acquisition, feature extraction and classification :

2.1. Establish ssh connection to Red Pitaya.

2.2. Execute “{AIS-SL2-1266490}.py” python code from Raspberry Pi.

2.3. This “{AIS-SL2-1266490}.py” is using “redpitaya_scpi.py” to access Red Pitaya over the IP network using SCPI commands.

2.4. Further SCPI commands written in “{proj-name}.py” is used to plot the real time signal data coming from Red Pitaya.

2.5. For feature extraction, execute “{FeatureExtraction}.py” code to save and plot both time and feature representation of the real time signal data coming from Red Pitaya.

2.6. Execute “{FinalRedpitaya}.py” code for real time classification of 3 objects namely wall, car and human. This classification is displayed by 3 different LEDs on Raspberry Pi.

3. To enable plug-n-play :

3.1. Create a shell script “{AIS-SL2-1266490}.sh” on Raspberry Pi that will execute shell script “{AIS-SL2-1266490-ssh-script}.sh” for ssh connection and then, will execute python code for real time classification.

3.2. Create a service daemon description describing this shell script to be called after boot-up network initialization.

3.3. Move the service daemon file to “/etc/systemd/system”. Save it as “{AIS-SL2-1266490}.service”.

3.4. Enable executable rights to the user for the service file.

3.5. Execute “systemctl start {AIS-SL2-1266490}.service”.

3.6. Make sure there are no errors in the resulting log file. If there are any fix them.

3.7. Execute “systemctl enable {AIS-SL2-1266490}.service” to enable daemon startup at boot-up.

IX. Conclusion

We have provided the result in the following report which conclude that sonar echos can be used efficiently for object recognition LSTM neural network provides high accuracy for the classification of time series data. With the proper trained model with efficient layers for the feature extraction can be used in the automotive field for object recognition in real time which can in turn help for the betterment of security

References

A novel connectionist system for unconstrained handwriting recognition. (2009).

A. Krizhevsky, I. S. (2012). Advances in Neural Information Processing Systems,.

Alex Graves, N. B. (2003). A comparison between spiking and differential recurrent neural networks on spoken digit recognition. Grindelwald,.

Alex Graves, N. B. (2005). Rapid restraing of speech data with LSTM neural networks.

Artificial Neural Network. (n.d.). Retrieved from WIkipedia: https://en.wikipedia.org/wiki/Artificial_neural_network

Gers, F. (1999). *"Learning to forget: Continual prediction with LSTM".*

Graves, A. (2014). Generating Sequences With Recurrent Neural Networks.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2015). *LSTM: A Search Space Odyssey". IEEE Transactions on Neural Networks and Learning Systems.*

Hinton, G. E. (2006). “Reducing the dimensionality of data with neural networks.”.

Hinton, G. E. (2009). Deep belief networks.

Hochreiter, S. (1991.). *Untersuchungen zu dynamischen neuronalen.*

Hochreiter, S. a. (1997). Long Short-Term Memory.” Neural Computation,.

Ilya Sutskever, O. V. (2014). Advances in Neural Information Processing.

K. Hornik, M. S. (1989). *Multilayer feedforward networks are universal.* San Diego: Pcrgamon Press plc.

K. Jarrett, K. K. (2009). In Computer Vision, International Conference.

Karpathy, A. (2015). *Neural Networks for Visual Recognition,.* Retrieved from http://cs231n.github.io/neural-networks-1/

Kelvin Xu, J. B. (2015). Neural image caption generation with visual attention.

LSTM wikipedia. (n.d.). Retrieved from wikipedia.com

M. D. Zeiler, D. K. (2010). Deconvolutional networks. In Computer Vision and Pattern Recognition, Conference.

Minh-Thang Luong, I. S. (2014). Addressing the rare word problem in neural machine translation.

Olshausen, B. A. (1996.). Emergence of simple-cell receptive field properties by learning.

Oriol Vinyals, A. T. (2015). A Neural Image Caption Generator.

Ponce, D. F. (2002). Computer Vision: A Modern Approach. Prentice Hall Professional. New Jersey.

Schmidhuber, J. (2015.). Recurrent Neural Networks.

Sepp Hochreite, J. S. (1997). LONG SHORT-TERM MEMORY.

Sepp Hochreiter, J. S. (1997). *"Long short-term memory". Neural Computation.*

Stallkamp, J. M. (2011). Man Vs. Computer: Benchmarking Machine Learning Algorithms For Traffic Sign Application.”.

Unconstrained online handwriting recognition using neural network. (2007).

UnderStanding LSTM and its diagram. (n.d.). Retrieved from https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714

Y. LeCun, B. B. (1989.). Backpropagation applied to handwritten zip code recognition. Neural Computation.