Summer Internship

<u>Integration of IoT and Machine Learning to capture operating events of an electric motor-driven fan</u>

PROJECT REPORT

Submitted by

T.NIKHILESH

A V YASWANTH SAI TEJA

SUJITH P K

Under the supervision of

SANJAY KUBERKAR

ABSTRACT

For modern industrial applications, real time diagnosis and monitoring of operating conditions of motor driven machines are of great importance. With the recent advanced developments in machine learning and artificial intelligence techniques and the availability of cloud data streaming and cloud storage, the field of real time diagnosis has taken impetus. Deep learning ,mainly the neural networks have given us ample of scope for research and innovations in this field. In this project we have built a model which is an approach to predict the rotating speed of an electric motor driven fan using machine learning and Google IoT core. This model can be employed for a wide range of applications where real time diagnosis and monitoring of operating condition is required.

Table of contents

INTRODUCTION

Machine learning has gained great popularity in recent times due to the numerous application possibilities in robotics and artificial intelligence. In addition to the conventional applications today machine learning is widely used for the real time monitoring of machines.

Real time monitoring of motor driven machines includes identifying the events on the machine and monitoring if necessary. Analysis of events in real time gives information about the operating condition of the machine. If the identified event is not as per expectation or is crossing the safety limits then either the machine should be monitored for expected operation or needs to be stopped.

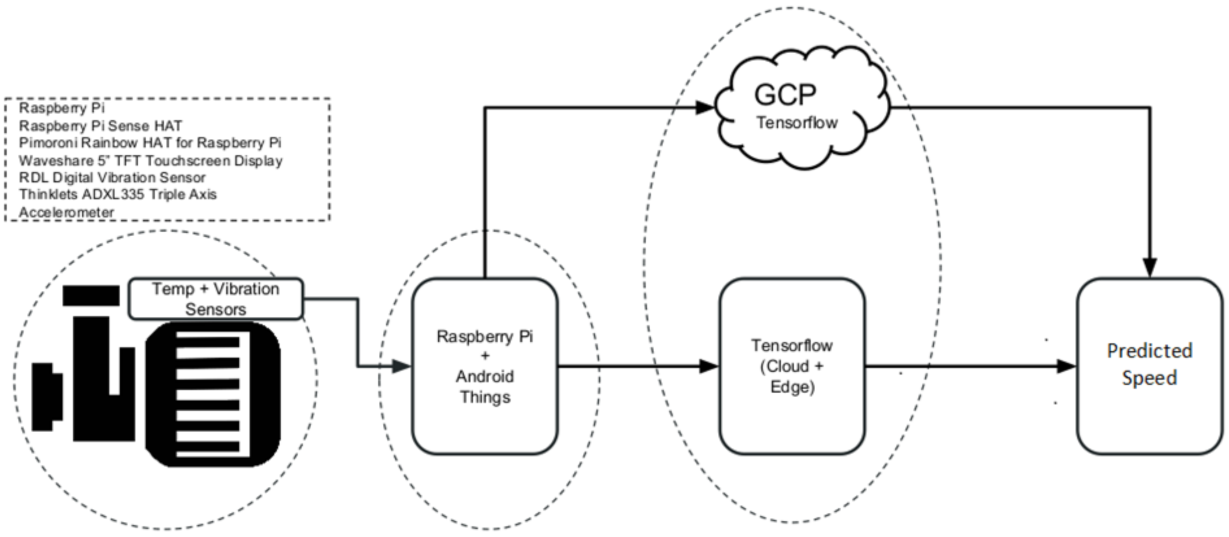
Real time monitoring has been a topic of great importance since any failure with the motors may lead to unexpected consequence of the whole machine. As bearing failures always bring downtime, expensive repair and hidden cost to enterprises, real time monitoring and precise fault diagnosis are critical to avoid catastrophic damages. In applications where motors are expected to operate full time, real time monitoring has got immense importance.

In our project we use Deep machine learning techniques to predict the rotation speed of a motor driven fan. We use Convolutional neural networks (CNNs or ConvNets) to predict the speed. Acceleration, vibration data are used to train and test the CNNs. The convolutional neural networks take the root mean square (RMS) maps from the FFT (Fast Fourier Transformation) features of the time-sampled acceleration signals from accelerometer as inputs. Tensorflow, a symbolic math library widely used to design neural networks using tensor based operations, is used to build the CNN model.

To stream data we use google cloud IoT core involving services such as cloud pubsub, cloud data storage, cloud dataflow.

The main objective of the project is to collect the vibration data from the accelerometer and stream the data into the CNN and predict the rotation speed of the motor.

The below schematic diagram shows the architecture of the project.


Raspberry Pi
Raspberry Pi Sense HAT
Pimoroni Rainbow HAT for Raspberry Pi
Waveshare 5" TFT Touchscreen Display
RDL Digital Vibration Sensor
Thinklets ADXL335 Triple Axis Accelerometer
Temp + Vibration Sensors
Raspberry Pi + Android Things
GCP Tensorflow
Tensorflow (Cloud + Edge)
Predicted Speed

Schematic representation of the project

This architecture is realised in the below four different approaches:

- Collecting data and running machine learning locally on Raspberry Pi (Raspbian OS) to get the prediction.
- Collecting data, streaming data into gcloud and running machine learning on google cloud to get the prediction.
- Collecting data and running machine learning locally on AndroidThings to get the prediction , using Tensorflow Lite.
- Collecting data, streaming data into google cloud and running machine learning on google cloud to get the prediction using Android things and Tensorflow Lite.

The above four approaches are described in the future sections of the report .

INSTALLING OPERATING SYSTEM & SETTING UP RASPBERRY PI

Installing the OS

Requirements:

1. Latest Version of RASPBIAN operating system.
2. An SD card (8GB or more).

Downloading the image:

3. Download the official Raspbian image of the website which is 4GB in size.
4. Unzip the archive (7ZIP)
5. Write the image to an SD Card (Etcher)

Setting up raspberry pi

1. Once the image is written into the SD Card
2. A file named ssh must be created without any file extension

   and the card is inserted into pi, power it up using an adapter which delivers 2Amps
3. Connection between Rpi and working platform (laptop) is made by Ethernet cable
4. Get the IP address of Rpi using any software like advanced ipscanner
5. Now the Rpi can be interacted with laptop with the help of any virtual terminal such as PUTTY

Check for updates and update the OS with the help of commands below.

```
sudo apt-get update
sudo apt-get upgrade
```

SETTING UP DISPLAY FOR RASPBERRY PI

A Wave share 5 inch HDMI LCD model B display is used.

Steps involved in setting up of the display

- In the sd card to which the system image is written ,a text file named config is modified by appending the following lines

```
max_usb_current=1
hdmi_group=2
hdmi_mode=87
hdmi_cvt 800 480 60 6 0 0
hdmi_drive=1
```

- Insert the sd card back into pi and set up the connections between LCD display and RPI using HDMI cable and USB cable for powering
- Check that backlight button on display is ON and calibrate the display once the desktop is seen.

INSTALLING AND INTEGRATING SENSORS WITH RPI

SCHEMATIC:

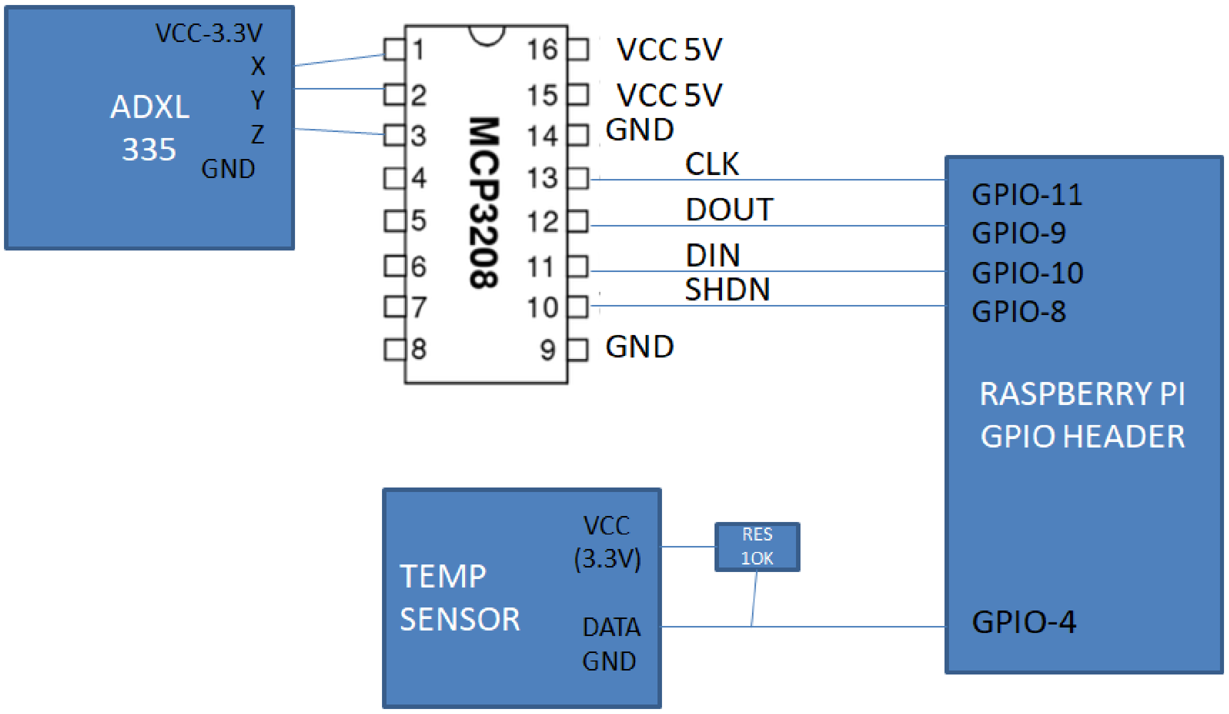


Figure.2

Sensors Introduction and theory:

1. ADXL335 3-axis accelerometer,
2. DS18B20 digital temperature sensor and

3. ADIS16220 digital vibration sensor.

ADXL335 ACCELEROMETER:

The ADXL335 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs.The product measures acceleration with a minimum full-scale range of ±3 g. It can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration. Bandwidth of the accelerometer range from 0.5 Hz to 1600 Hz for the X and Y axes, and a range from 0.5 Hz to 550 Hz for the Z axis.

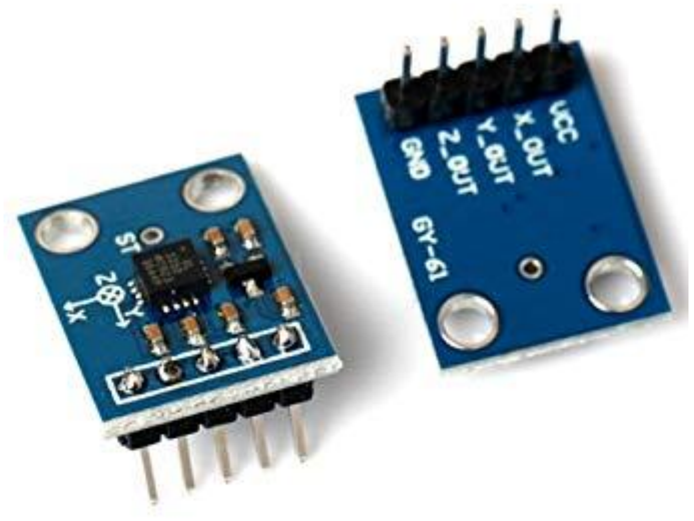


Figure-3 shows the image of ADXL335 Accelerometer.

Pin Configuration:

- ADXL335 has five pins.
- Vcc (3.3v is preferable),
- Ground pin,
- X_OUT, Y_OUT, Z_OUT.

The outputs of these axes give Analog voltage values based on the acceleration of each axes. The ADXL335 output is ratiometric; therefore, the output sensitivity (or scale factor) varies proportionally to the supply voltage (VS).

For VS = 3.6 V, the output sensitivity is typically 360 mV/g.
VS = 2 V, the output sensitivity is typically 195 mV/g.

Here, 3.3v is being used for supply voltage. So the output sensitivity is typically 330mV/g.

The zero g bias output is also ratiometric, thus the zero g output is nominally equal to VS/2 at all supply voltages. The zero g bias voltage is voltage when there is no gravity acting on the axes (technically when axes are horizontal).

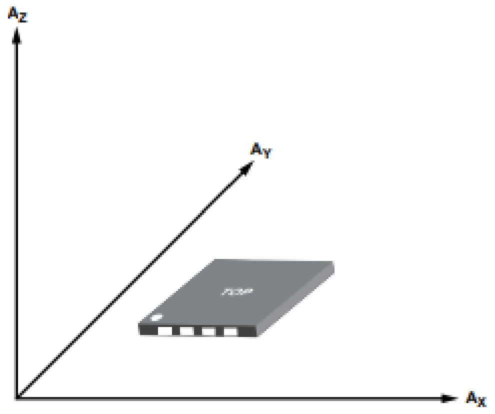
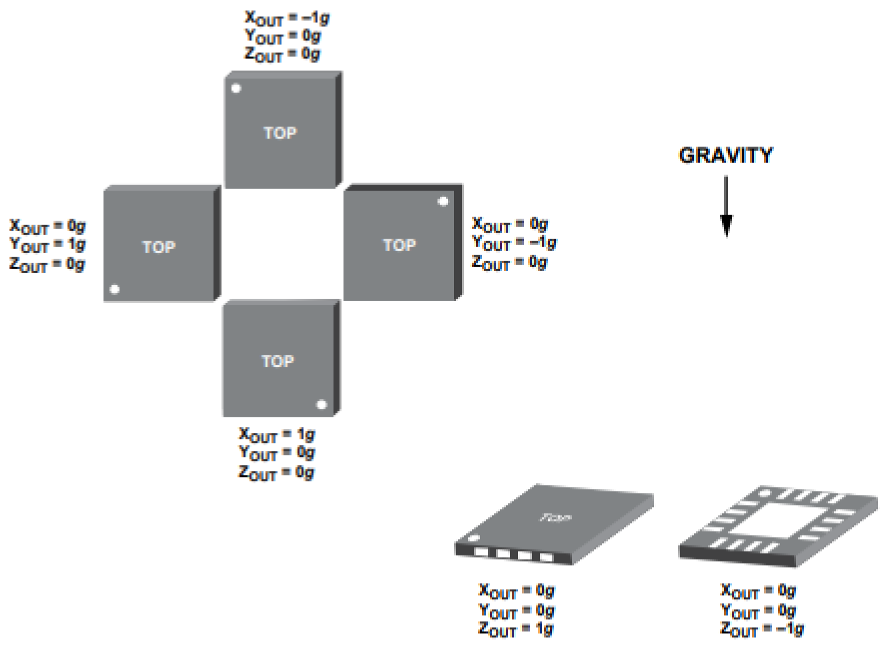


Figure-4 describes axes of Acceleration Sensitivity;

Corresponding Output Voltage Increases When Accelerated along the Sensitive Axis.



$X_{OUT} = -1g$ $Y_{OUT} = 0g$ $Z_{OUT} = 0g$
TOP
GRAVITY
$X_{OUT} = 0g$ $Y_{OUT} = 1g$ $Z_{OUT} = 0g$
TOP
TOP
$X_{OUT} = 0g$ $Y_{OUT} = -1g$ $Z_{OUT} = 0g$
TOP
$X_{OUT} = 1g$ $Y_{OUT} = 0g$ $Z_{OUT} = 0g$
TOP
$X_{OUT} = 0g$ $Y_{OUT} = 0g$ $Z_{OUT} = 1g$
$X_{OUT} = 0g$ $Y_{OUT} = 0g$ $Z_{OUT} = -1g$

Figure 24. Output Response vs. Orientation to Gravity

Figure-5

When there is no motion or vibration accelerometer experiences only acceleration due to gravity. In that case, the axis which is horizontal will be having 0g value and vertically downwards experience +1g acceleration; if the axis is vertically upward then it experiences -1g acceleration.

From Figures 4 & 5, when the Z-axis is vertically downwards it experiences +1g and remaining two axes are horizontal so they experience 0g and if Z-axis is vertically upward then the value is -1g. similarly for the other two cases horizontal Axes have 0g and vertically upward Axes has -1g and vertically downward axes has +1g.So based on orientation, values ranges between -1g and +1g.

Generally, when the power supply is 3.3v the 0g bias voltage is 3.3/2=1.65v.Now,when the axes experience +ve acceleration the voltage increases (>1.65), if it experiences -ve acceleration then voltage decreases (<1.65).

We cannot directly interface these outputs to RPI because these outputs are analog voltage values but RPI supports only digital values. So in order to connect the sensor to RPI, an ADC which converts the analog voltage value to digital value, can be used (MCP3208 12-bit ADC).

ADXL335 CALIBRATION:

Calibration increases the accuracy of any output data as the sensitivity of accelerometer changes for different voltages.

Here we get the data and find the 0g bias voltage and sensitivity of each axis. When the axis(any of X,Y,Z) is vertically downward it gives +1g and gives corresponding voltage value say (X1,Y1,Z1) respectively, flip the sensor upside down then it gives -1g and gives corresponding voltage value say (X2,Y2,Z2) respectively. Then after getting the above data,

Taking average of both gives 0g bias voltage of each axis and

Subtracting the two values and divide by 2 gives sensitivity in V/g.

X0= X1+X2/2 (0g bias voltage of X-axis)

Xs= X1-X2/2 (sensitivity of X-axis in V/g)

Similarly for the other two Axes.

DS18B20 TEMPERATURE SENSOR:

DS18B20 is a digital thermometer that provides 9-bit to 12-bit Celsius temperature measurements. The DS18B20 communicates with the “One-Wire” communication protocol, a proprietary serial communication protocol that uses only one wire to transmit the temperature readings to the microcontroller. In addition, the DS18B20 can derive power directly from the data line (“parasite power”), eliminating the need for an external power supply.

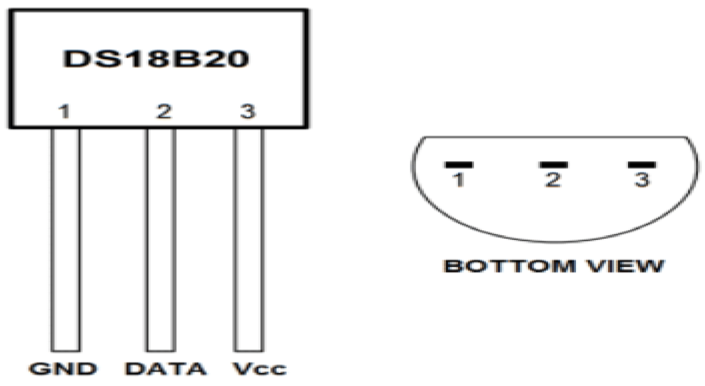


Figure.6

This contains three pins Vcc, ground and data pin. Operating power is between 3v to 5v. A 64 bit ROM stores the device’s unique serial code. This 64 bit address allows a microcontroller to receive temperature data from a virtually unlimited number of sensors at the same pin. The address tells the microcontroller from which sensor a particular temperature value is coming from.

MCP3208 ADC:

MCP3208 is a 12-bit Analog to Digital convertor. Communication with the devices is accomplished using a simple serial interface compatible with the SPI protocol. The devices are capable of conversion rates of up to 100 ksps. The MCP3204/3208 devices operate over a broad voltage range (2.7V - 5.5V).

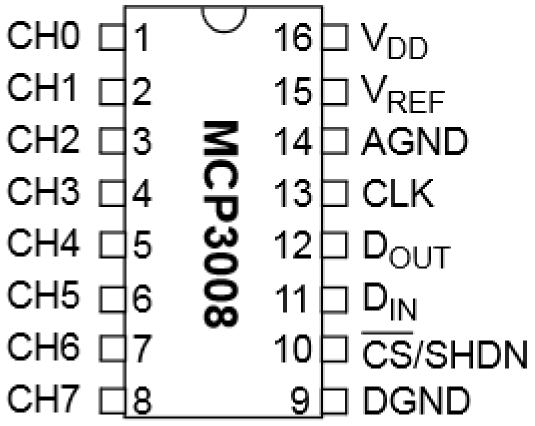


Figure.7

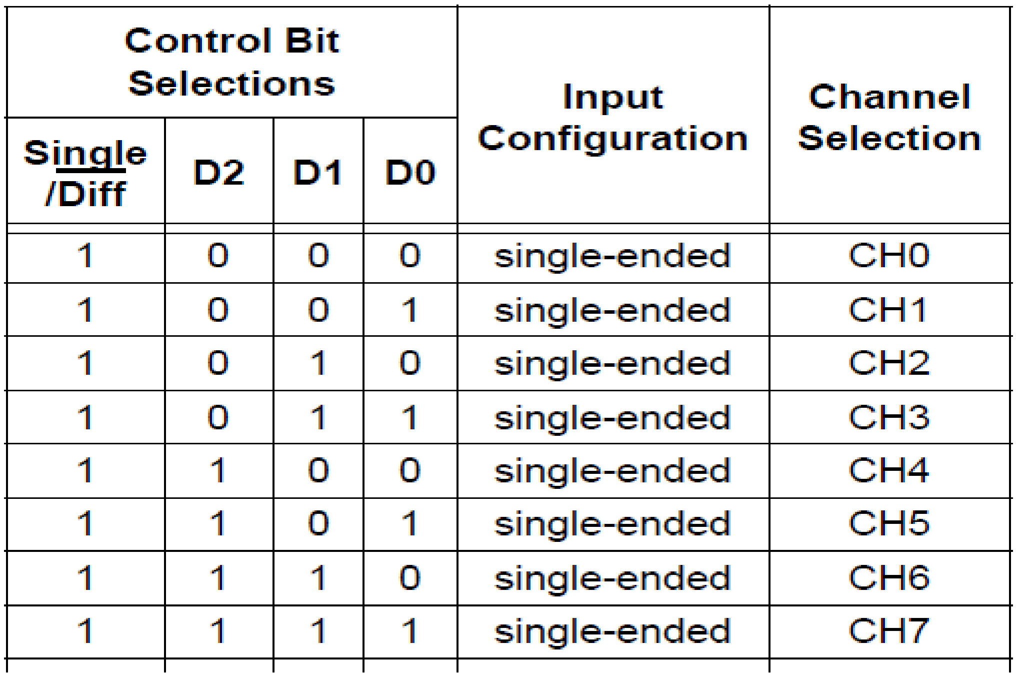
From the above pin diagram, ADC has 8 channels (0-7). A 4 pin SPI protocol is used which includes CLK,DOUT,DIN and SHDN which are connected to the BCM pins of the raspberry pi such as

CLK – SPI SCLK (bcm gpio-11 pin of Rpi)

DOUT – SPI MISO (bcm gpio-9 pin)

DIN – SPI MOSI (bcm gpio-10 pin)

SHDN – SPI CE0 (bcm gpio-8 pin)



Control Bit Selections				Input Configuration	Channel Selection
Single /Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7

Figure.8

Here we used single ended mode as we have only one input channel which takes bit 1. And the bits D2,D1,D0 are used to select the input channel configuration.

<u>INTEGRATING SENSORS</u>:

Communication with the MCP3208 devices is accomplished using a standard 4 pin SPI-compatible serial interface. The Raspberry Pi has a Broadcom BCM 2835 chip allowing it to interface with SPI devices on its GPIO pins.

ADXL335 requires the SPI protocol for interfacing.

Enabling the SPI on Raspberry Pi

1. In Pi terminal ,run

   ```
   sudo raspi-config
   ```

2. Go to Advanced Options > SPI

3. Choose “Yes” for both questions then select Finish to exit raspi-config

4. Either reboot your Pi or run this command to load the kernel module

```
sudo modprobe spi-bcm2708
```

Install Spidev

Spidev is a python module that allows us to interface with the Pi's SPI bus.

1. sudo apt-get update
2. sudo apt-get upgrade
3. sudo apt-get install python-dev python3-dev
4. cd ~
5. git cone https://github.com/doceme/py-spidev.git
6. cd py-spidev
7. make
8. sudo make install

Now the SPI controlled devices like ADXL335 can be interfaced with Raspberry pi.

For the temperature sensor we need to enable the One-Wire interface to receive data from the sensor. Once the DS18B20 sensor is connected, open the terminal and then follow these steps to enable the One-Wire interface:

In the Pi terminal, enter

1.
```
sudo nano/boot/config.txt        # this opens config.txt file. In this file at
the bottom add the line
dtoverlay=w1-gpio
```

2. Exit and reboot the Pi

3. open terminal and enter

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

4. after the above steps open the directory, `cd /sys/bus/w1/devices and` enter `ls`
5. It displays our temperature sensor address () and `w1_bus_master1`
6. now enter cd Address()
7. enter `cat_w1_slave` to get the raw temperature output data
8. enter `cd` and go back to root directory.

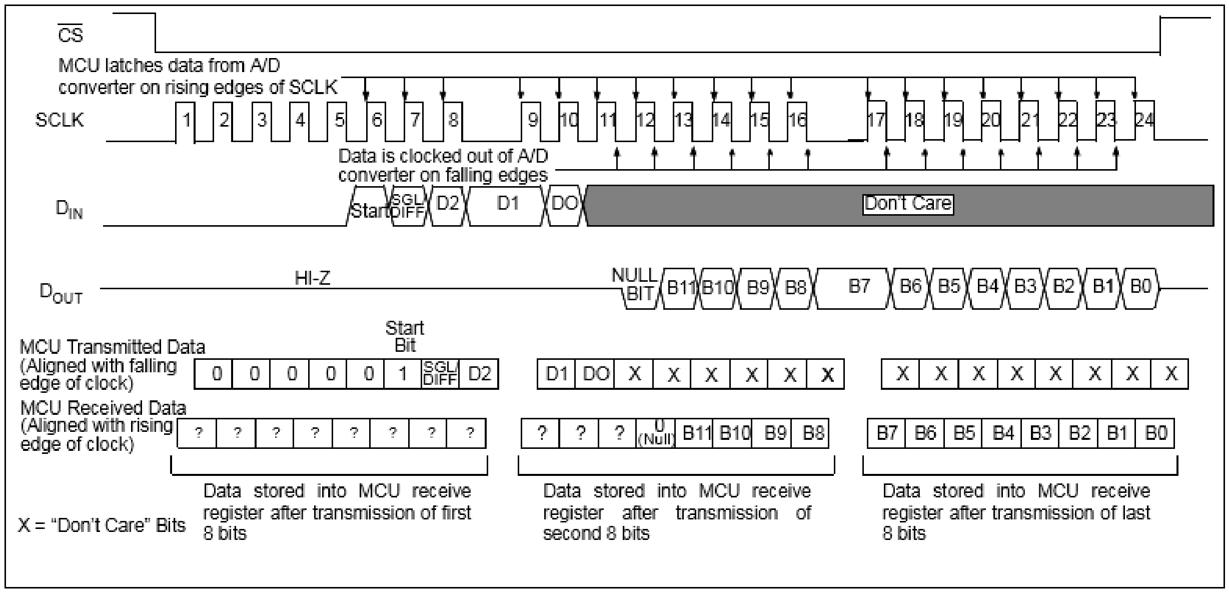
CODE:

```
path = "/home/pi/Downloads/sensor_Data.csv"
import spidev
from time import sleep
import os
import glob
import time
from datetime import datetime


## To open the SPI bus

spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz =7629          ## max driver freq

os.system('modprobe w1-gpio')         ## Enabling GPIO PIN

os.system('modprobe w1-therm')        ## Enabling the temp sensor

base_dir = '/sys/bus/w1/devices/'                ## address location of temperature
sensor
device_folder = glob.glob(base_dir + '28*')[0]   ## retriving the address
device_file = device_folder + '/w1_slave'        ## to display raw data



## Different channels used by MCP3208 Analog To Digital Converter

xchannel = 0
ychannel = 1
zchannel = 2
vchannel = 3
j=0



def getReading(channel):


## SPI Communication with MCP3208

    ## pulling the raw data from the chip
    rawData = spi.xfer([6, (0+channel) << 6, 0])        ###  12 BIT CHANNEL
    #rawData = spi.xfer([4 | 2 | (channel >> 2), (channel & 3) << 6, 0])    ---
OTHER WAY TO WRITE THE ABOVE STATEMENT

    ## Rawdata to Bit value
    processedData = ((rawData[1]&15) << 8) + rawData[2]
    return processedData

def convertVoltage(bitValue, decimalPlaces=2):
    voltage = (bitValue * 3.3) / float(4095)
    voltage = round(voltage, decimalPlaces)
    return voltage
```

```python
def convertGx(voltage, decimalPlaces=2):
    gvaluex = (voltage - 1.65)/float(0.33)
    gvaluex = round(gvaluex, decimalPlaces)
    return gvaluex

def convertGy(voltage, decimalPlaces=2):
    gvaluey = (voltage - 1.64)/float(0.33)
    gvaluey = round(gvaluey, decimalPlaces)
    return gvaluey

def convertGz(voltage, decimalPlaces=2):
    gvaluez = (voltage - 1.700)/float(0.325)
    gvaluez = round(gvaluez, decimalPlaces)
    return gvaluez

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines


def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f


while j<3141:

## reading the data from accelerometer adxl335

    xData = getReading(xchannel)
    yData = getReading(ychannel)
    zData = getReading(zchannel)

## reading the data from vibration sensor ADIS16220

    vData = getReading(vchannel)


    xVoltage = convertVoltage(xData)
    yVoltage = convertVoltage(yData)
    zVoltage = convertVoltage(zData)

## conversion of acceleration data to g's

    xG = convertGx(xVoltage)
    yG = convertGy(yVoltage)
    zG = convertGz(zVoltage)

## reading the data from temperature sensor DS18B20
```

```
    temp_c,temp_f = read_temp()


## outputting the sensor data from all the sensors

    print("x bitValue = {} ; Voltage = {} V ; xGvalue = {} g".format(xData,
xVoltage, xG))
    print("y bitValue = {} ; Voltage = {} V ; yGvalue = {} g".format(yData,
yVoltage, yG))
    print("z bitValue = {} ; Voltage = {} V ; zGvalue = {} g".format(zData,
zVoltage, zG))
    print("v bitValue = {} ".format(vData))
    print("temp_c = {} ".format(temp_c))
    print("temp_f = {} ".format(temp_f))

    sleep(sleepTime)


## writing the retrived data to a csv file which will be created in the path given
in the first line of this code

    file = open(path, "a")
    if os.stat(path).st_size == 0:
            file.write("X1,Y1,Z1\n")
    file.write(str(xG)+","+str(yG)+","+str(zG)+"\n")
    file.flush()
    j=j+1
file.close()
```

SPI Channel:



CS
MCU latches data from A/D converter on rising edges of SCLK
SCLK
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Data is clocked out of A/D converter on falling edges
DIN
Start
SGL/DIFF
D2
D1
DO
Don't Care
DOUT
HI-Z
NULL BIT
B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
MCU Transmitted Data (Aligned with falling edge of clock)
Start Bit
0 0 0 0 0 1 SGL/DIFF D2
D1 DO X X X X X X
X X X X X X X X
MCU Received Data (Aligned with rising edge of clock)
? ? ? ? ? ? ? ?
? ? ? 0 (Null) B11 B10 B9 B8
B7 B6 B5 B4 B3 B2 B1 B0
Data stored into MCU receive register after transmission of first 8 bits
Data stored into MCU receive register after transmission of second 8 bits
Data stored into MCU receive register after transmission of last 8 bits
X = "Don't Care" Bits

FIGURE 6-1: *SPI Communication using 8-bit segments (Mode 0,0: SCLK idles low).*

Figure.9

From the above ADC communication diagram, both the MCU transmitted data and received data has three 8-bit segments.

```
rawData = spi.xfer([6, (0+channel) << 6, 0])
processedData = ((rawData[1]&15) << 8) + rawData[2]
```

First line of above two refers to transmitted data and second line refers to received data.

From the processedData, we get required 12-bit bit values for each channel.

Conversions in the code:

1. `def convertVoltage`

As the MCP3208 gives 12-bit bit values ranging from 0-4095.We have to convert bit value into voltage experienced by each axes.

2. `def convertG`

To convert the voltages experienced by each axes to acceleration experienced by each axes, we have to convert by the calibrated values of each axis.

For example consider z axis,

Let the voltage we get be Zv,

0g bias voltage of z-axis is Z0 and sensitivity Zs.

Then, Gz= (Z-Z0) / (Zs) gives the acceleration value.

MACHINE LEARNING WITH CONVOLUTIONAL NEURAL NETWORKS USING TENSORFLOW:

Introduction and Design:

Machine learning:

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data such as examples, direct experience, or instruction in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Artificial neural networks :

Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analysing example images that have been manually labelled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

Convolutional neural networks:

A convolutional neural network (CNN or ConvNet) is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery.
CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

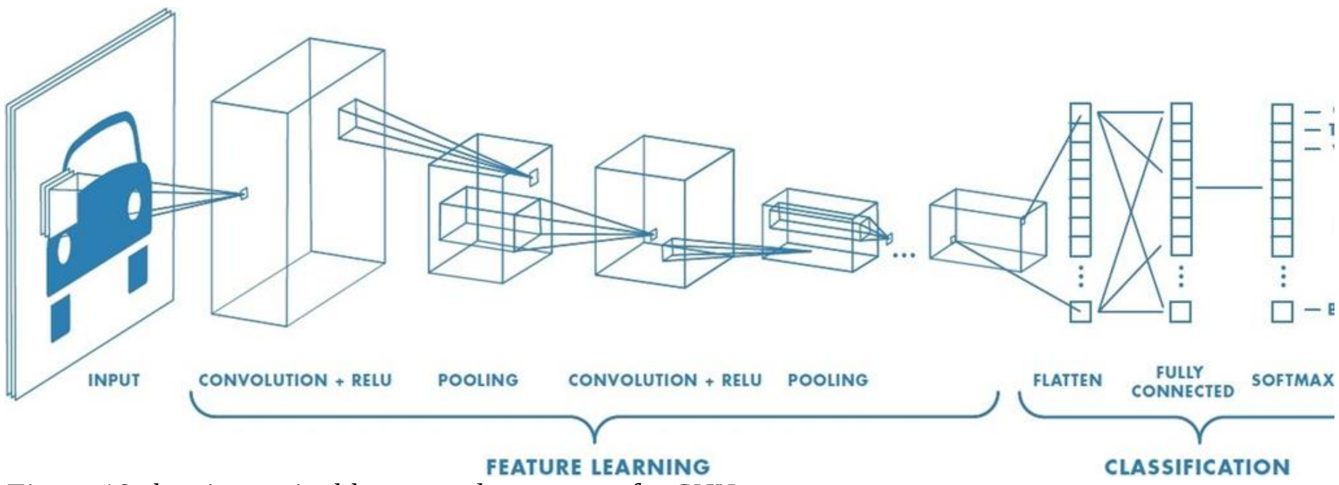


Figure.10 showing typical layers and structure of a CNN

Design of typical CNNs:

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers

Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.

Convolutional layer:

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli.

Each convolutional neuron processes data only for its receptive field.

Although fully connected feed forward neural networkscan be used to learn features as well as classify data, it is not practical to apply this architecture. A very high number of neurons would be necessary, due to the very large input sizes associated with certain applications, where each data is a relevant variable. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters by using backpropagation.

Pooling layer:

Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Another method is average pooling, which uses the average value from each of a cluster of neurons at the prior layer.

Fully connected layer:

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP).

Weights:

CNNs share weights in convolutional layers, which means that the same filter is used for each receptive field in the layer. This reduces memory footprint and improves performance.

Biases:

Bias nodes are added to feed forward neural networks to help these learn patterns. Bias nodes function like an input node that always produces a constant value. Because of this property, they are not connected to the previous layer. The constant here is called the bias activation. Bias neurons allow the output of an activation function to be shifted.

Tensorflow:

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Originally developed by researchers and engineers from the Google Brain team within Google’s AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

The TensorFlow ‘layers’ module provides a high-level API that makes it easy to construct a neural network. It provides methods that facilitate the creation of dense (fully connected) layers and convolutional layers, adding activation functions, and applying dropout regularization.

Tensorflow Lite:

TensorFlow Lite is TensorFlow’s lightweight solution for mobile and embedded devices. It enables on-device machine learning inference with low latency and a small binary size. TensorFlow Lite also supports hardware acceleration with the Android Neural Networks API.

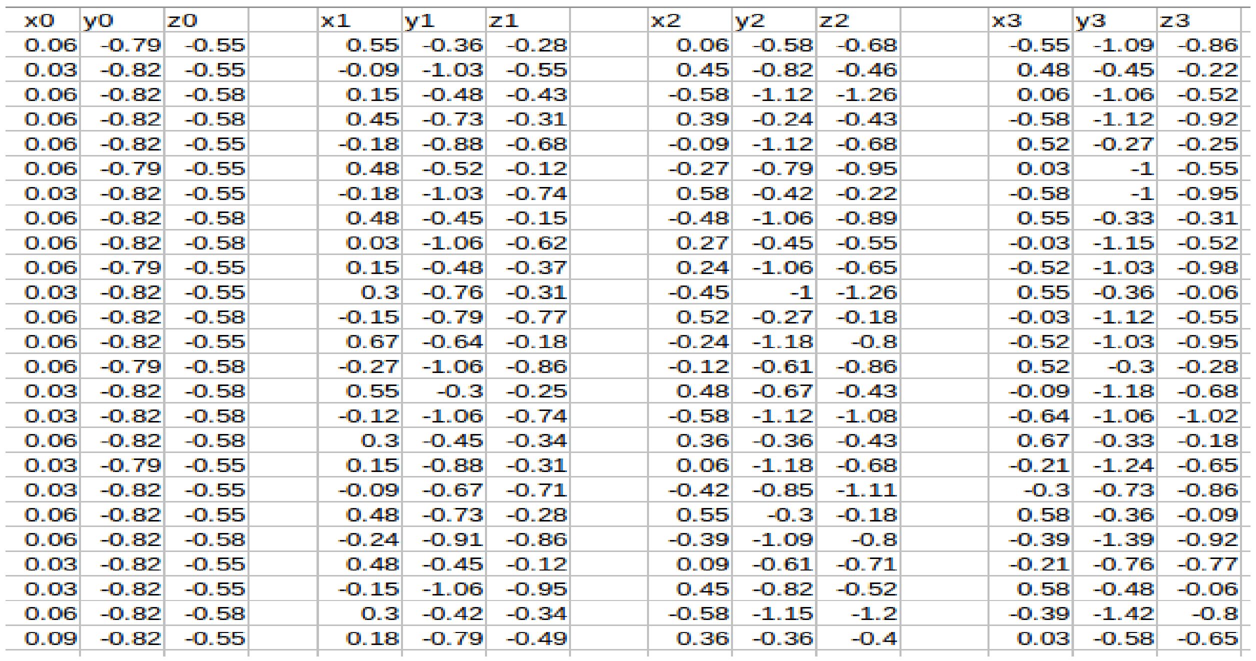
TensorFlow Lite uses many techniques for achieving low latency such as optimizing the kernels for mobile apps, pre-fused activations, and quantized kernels that allow smaller and faster (fixed-point math) models.

Running ML locally:

Steps involved in preparing the ML model for an electric motor driven pump using its acceleration values:

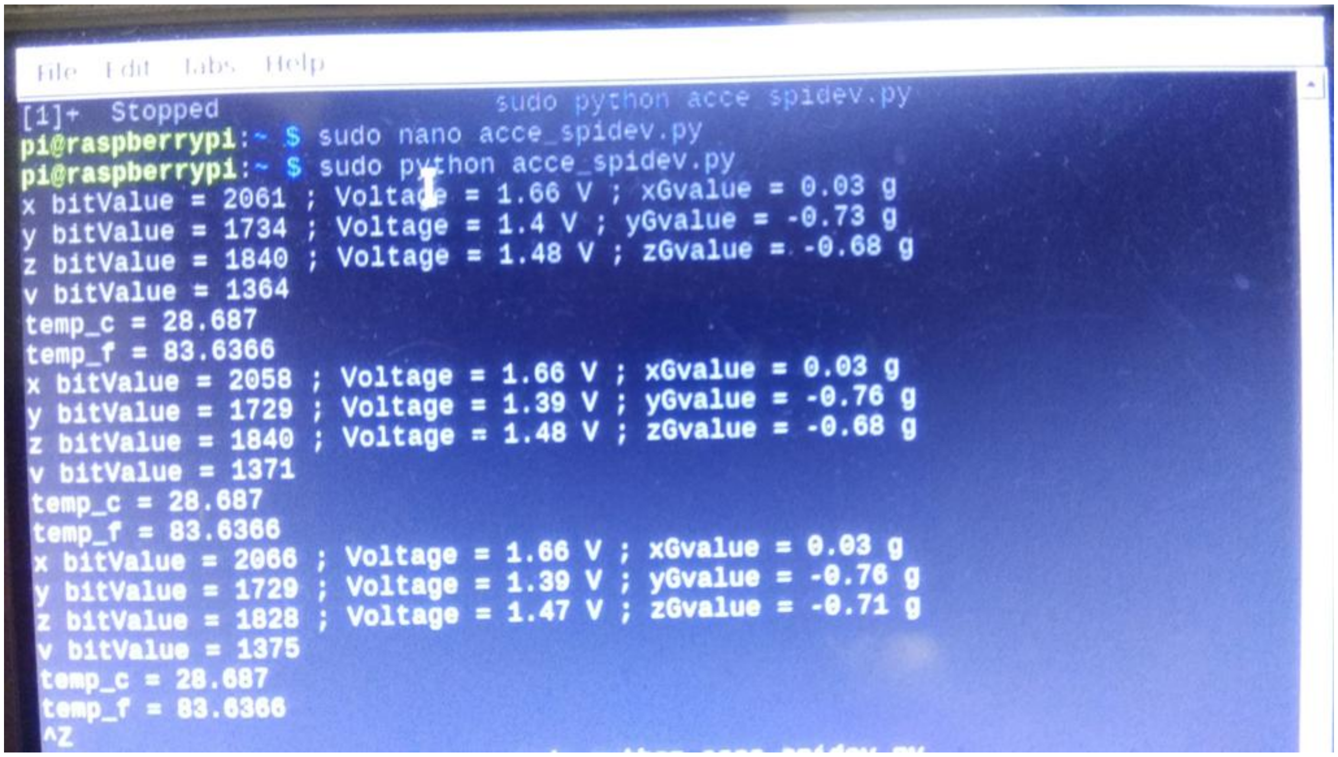
Collecting data for different speeds of fan

Feature extraction from raw sensor data is critical in machine monitoring. Several sets of acceleration values for different speeds of fan are collected and stored. The values stored were acceleration values with unit ‘g’ (9.8 m/s^2) . The values are stored as comma seperated values (.csv format). A sample format of collected data for different speeds is shown below.



x0	y0	z0	x1	y1	z1	x2	y2	z2	x3	y3	z3
0.06	-0.79	-0.55	0.55	-0.36	-0.28	0.06	-0.58	-0.68	-0.55	-1.09	-0.86
0.03	-0.82	-0.55	-0.09	-1.03	-0.55	0.45	-0.82	-0.46	0.48	-0.45	-0.22
0.06	-0.82	-0.58	0.15	-0.48	-0.43	-0.58	-1.12	-1.26	0.06	-1.06	-0.52
0.06	-0.82	-0.58	0.45	-0.73	-0.31	0.39	-0.24	-0.43	-0.58	-1.12	-0.92
0.06	-0.82	-0.55	-0.18	-0.88	-0.68	-0.09	-1.12	-0.68	0.52	-0.27	-0.25
0.06	-0.79	-0.55	0.48	-0.52	-0.12	-0.27	-0.79	-0.95	0.03	-1	-0.55
0.03	-0.82	-0.55	-0.18	-1.03	-0.74	0.58	-0.42	-0.22	-0.58	-1	-0.95
0.06	-0.82	-0.58	0.48	-0.45	-0.15	-0.48	-1.06	-0.89	0.55	-0.33	-0.31
0.06	-0.82	-0.58	0.03	-1.06	-0.62	0.27	-0.45	-0.55	-0.03	-1.15	-0.52
0.06	-0.79	-0.55	0.15	-0.48	-0.37	0.24	-1.06	-0.65	-0.52	-1.03	-0.98
0.03	-0.82	-0.55	0.3	-0.76	-0.31	-0.45	-1	-1.26	0.55	-0.36	-0.06
0.06	-0.82	-0.58	-0.15	-0.79	-0.77	0.52	-0.27	-0.18	-0.03	-1.12	-0.55
0.06	-0.82	-0.55	0.67	-0.64	-0.18	-0.24	-1.18	-0.8	-0.52	-1.03	-0.95
0.06	-0.79	-0.58	-0.27	-1.06	-0.86	-0.12	-0.61	-0.86	0.52	-0.3	-0.28
0.03	-0.82	-0.58	0.55	-0.3	-0.25	0.48	-0.67	-0.43	-0.09	-1.18	-0.68
0.03	-0.82	-0.58	-0.12	-1.06	-0.74	-0.58	-1.12	-1.08	-0.64	-1.06	-1.02
0.06	-0.82	-0.58	0.3	-0.45	-0.34	0.36	-0.36	-0.43	0.67	-0.33	-0.18
0.03	-0.79	-0.55	0.15	-0.88	-0.31	0.06	-1.18	-0.68	-0.21	-1.24	-0.65
0.03	-0.82	-0.55	-0.09	-0.67	-0.71	-0.42	-0.85	-1.11	-0.3	-0.73	-0.86
0.06	-0.82	-0.55	0.48	-0.73	-0.28	0.55	-0.3	-0.18	0.58	-0.36	-0.09
0.06	-0.82	-0.58	-0.24	-0.91	-0.86	-0.39	-1.09	-0.8	-0.39	-1.39	-0.92
0.03	-0.82	-0.55	0.48	-0.45	-0.12	0.09	-0.61	-0.71	-0.21	-0.76	-0.77
0.03	-0.82	-0.55	-0.15	-1.06	-0.95	0.45	-0.82	-0.52	0.58	-0.48	-0.06
0.06	-0.82	-0.58	0.3	-0.42	-0.34	-0.58	-1.15	-1.2	-0.39	-1.42	-0.8
0.09	-0.82	-0.55	0.18	-0.79	-0.49	0.36	-0.36	-0.4	0.03	-0.58	-0.65

Figure.11



```
File  Edit  Tabs  Help
[1]+  Stopped                 sudo python acce_spidev.py
pi@raspberrypi:~ $ sudo nano acce_spidev.py
pi@raspberrypi:~ $ sudo python acce_spidev.py
x bitValue = 2061 ; Voltage = 1.66 V ; xGvalue = 0.03 g
y bitValue = 1734 ; Voltage = 1.4 V ; yGvalue = -0.73 g
z bitValue = 1840 ; Voltage = 1.48 V ; zGvalue = -0.68 g
v bitValue = 1364
temp_c = 28.687
temp_f = 83.6366
x bitValue = 2058 ; Voltage = 1.66 V ; xGvalue = 0.03 g
y bitValue = 1729 ; Voltage = 1.39 V ; yGvalue = -0.76 g
z bitValue = 1840 ; Voltage = 1.48 V ; zGvalue = -0.68 g
v bitValue = 1371
temp_c = 28.687
temp_f = 83.6366
x bitValue = 2066 ; Voltage = 1.66 V ; xGvalue = 0.03 g
y bitValue = 1729 ; Voltage = 1.39 V ; yGvalue = -0.76 g
z bitValue = 1828 ; Voltage = 1.47 V ; zGvalue = -0.71 g
v bitValue = 1375
temp_c = 28.687
temp_f = 83.6366
^Z
```

Figure.12

Preprocessing the data to get training & testing data sets:

The raw data is processed to get training and testing data in order to increase the accuracy of the network.

Sampling:

Each raw data sample taken is of length 3140. A sliding window of length 5 and step size 2 is used to sample the data. Mean of the data in the given interval is given out as the sampled data containing 1568 data points.

Windowing using Hanning window and Fourier Transform:

To avoid spectrum leakage time domain signal should be windowed with a Hanning window. The samlpled signal is multiplied with Hanning window. Then the Fast Fourier Transform (FFT) of the resulting spectrum is taken.

Getting RMS of the frequency bands:

We use root mean square (RMS) over a sub band of the frequency spectrum as feature for our CNN model, which has the advantage of maintaining the energy shape at the spectrum peaks. The FFT spectrum obtained from the previous step is divided into 784 bands (28*28) and RMS of each band is calculated and stored , which is the final feature used to train or test the CNN model.

Classification of data:

The final data is classified as training and testing data (approximtely 80% and 20% respectively). 10% of the training data is used for validation.

Building the network model

The neural netwok consisits of the following layers:

Convolutional layer 1 :

The input data is assigned to variable x and is reshaped into [batch_size, 28,28,1]. x is convolved using 32 filters and a kernel of size 5. Stride size is taken 1 by default. The resulting layer has dimension [batch_size, 24,24,32].

Max pooling layer 1 :

Max- pooling is performed with kernel of size 2. Stride size is taken 2. Now the dimension becomes [batch_size, 12,12,32]

Convolutional layer 2 :

The max pooled layer is again convolved using 64 filters and a kernel of size 3. Stride size is taken 1 by default. The resulting layer has dimension [batch_size, 10,10,64].

Max pooling layer 2 :

Max- pooling is performed with kernel of size 2. Stride size is taken 2.. Dimensssion of conv2 becomes [batch_size, 5,5,64]

Flattened layer:

The resulting layer after max-pooling 2 is flattened to get a 1 dimensional flattened layer of dimension [batch_size, 1600]

Fully connected layer 1:

A dense layer of size 1024 with the previous flattened layer as input is created.

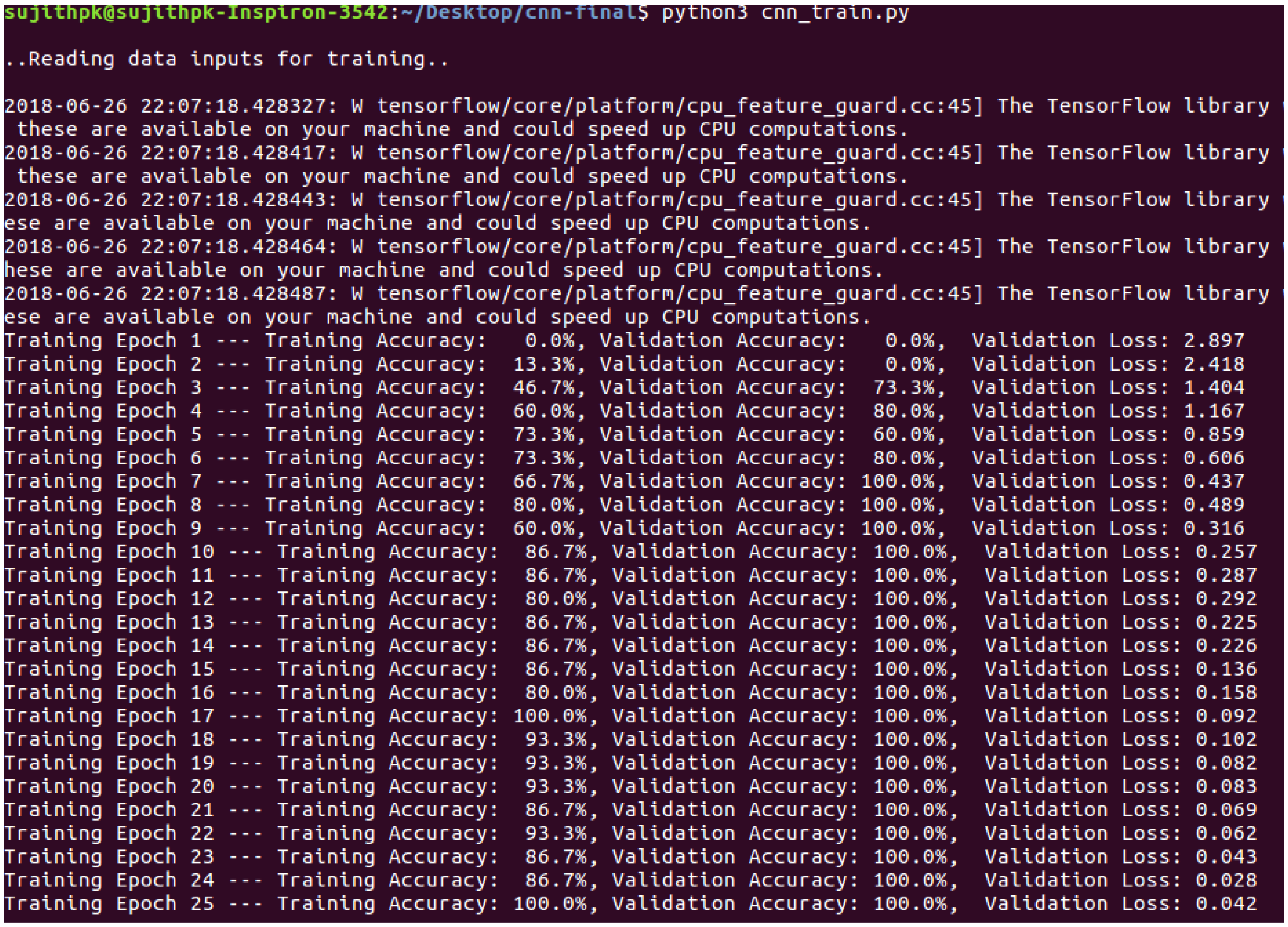
Fully connected layer 2:

Another dense layer of size n_classes is created. The outputs of this layer consists of predicted probabilities.

Training and validating the model and storing the graph

The constructed network is trained and validated with a batch size of 15 for 1600 iterations. During training, dropout is applied on the flattened layer. The weights are optimised using Adam optimiser. A learning rate of 10^-4 is employed. Training and validation progress is displayed at regular intervals during the process. Training and validation accuracy are displayed on the terminal. After training, the graph is saved in a .meta file and the assignments are stored in checkpoint files. Before tarining, the user should specify the number of training datasets being used, number of classes and the number of training steps.

Command to train the model:
python3 cnn_train.py



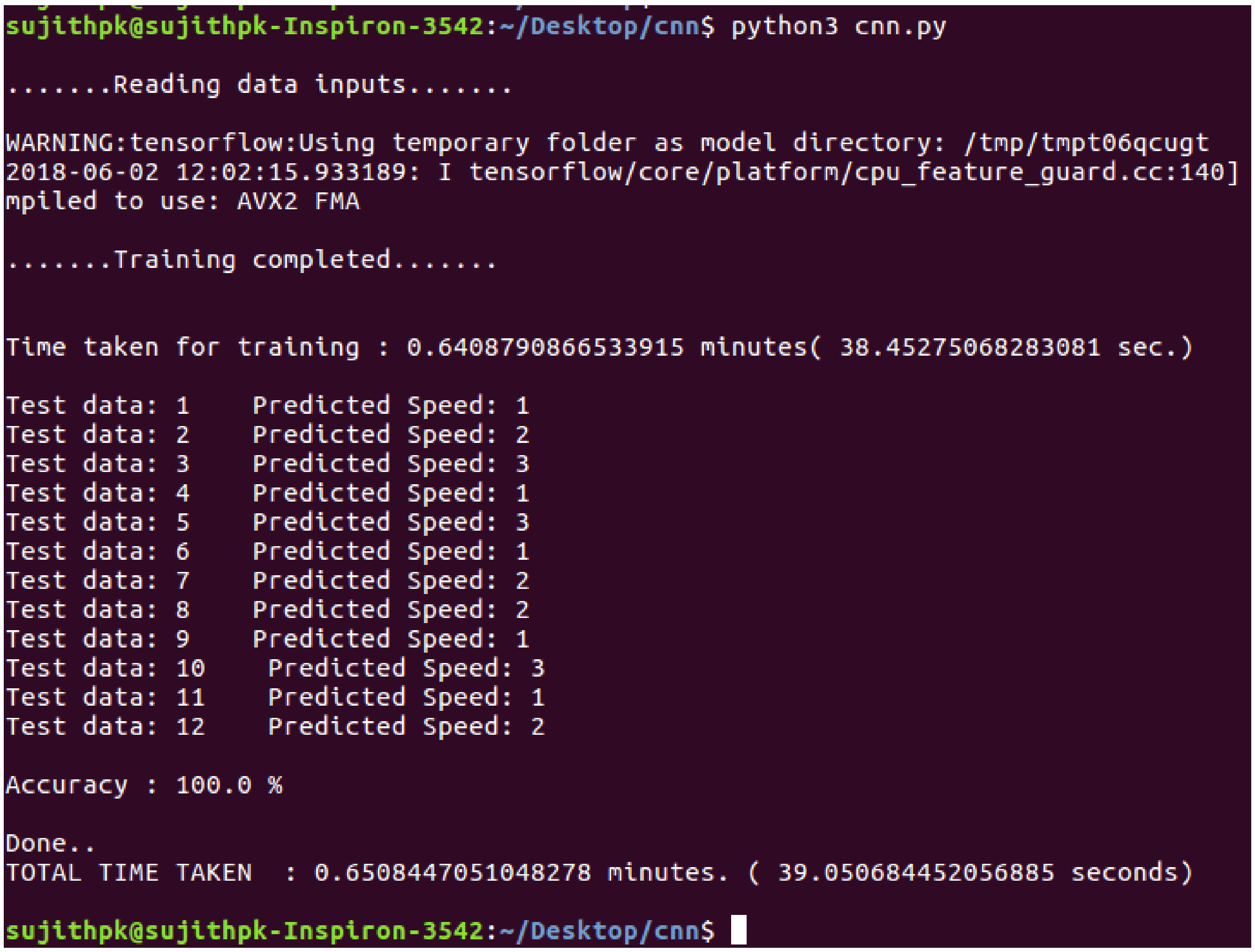
```
sujithpk@sujithpk-Inspiron-3542:~/Desktop/cnn-final$ python3 cnn_train.py

..Reading data inputs for training..

2018-06-26 22:07:18.428327: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
 these are available on your machine and could speed up CPU computations.
2018-06-26 22:07:18.428417: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
 these are available on your machine and could speed up CPU computations.
2018-06-26 22:07:18.428443: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
ese are available on your machine and could speed up CPU computations.
2018-06-26 22:07:18.428464: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
hese are available on your machine and could speed up CPU computations.
2018-06-26 22:07:18.428487: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
ese are available on your machine and could speed up CPU computations.
Training Epoch 1 --- Training Accuracy:   0.0%, Validation Accuracy:   0.0%,  Validation Loss: 2.897
Training Epoch 2 --- Training Accuracy:  13.3%, Validation Accuracy:   0.0%,  Validation Loss: 2.418
Training Epoch 3 --- Training Accuracy:  46.7%, Validation Accuracy:  73.3%,  Validation Loss: 1.404
Training Epoch 4 --- Training Accuracy:  60.0%, Validation Accuracy:  80.0%,  Validation Loss: 1.167
Training Epoch 5 --- Training Accuracy:  73.3%, Validation Accuracy:  60.0%,  Validation Loss: 0.859
Training Epoch 6 --- Training Accuracy:  73.3%, Validation Accuracy:  80.0%,  Validation Loss: 0.606
Training Epoch 7 --- Training Accuracy:  66.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.437
Training Epoch 8 --- Training Accuracy:  80.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.489
Training Epoch 9 --- Training Accuracy:  60.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.316
Training Epoch 10 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.257
Training Epoch 11 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.287
Training Epoch 12 --- Training Accuracy:  80.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.292
Training Epoch 13 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.225
Training Epoch 14 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.226
Training Epoch 15 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.136
Training Epoch 16 --- Training Accuracy:  80.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.158
Training Epoch 17 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.092
Training Epoch 18 --- Training Accuracy:  93.3%, Validation Accuracy: 100.0%,  Validation Loss: 0.102
Training Epoch 19 --- Training Accuracy:  93.3%, Validation Accuracy: 100.0%,  Validation Loss: 0.082
Training Epoch 20 --- Training Accuracy:  93.3%, Validation Accuracy: 100.0%,  Validation Loss: 0.083
Training Epoch 21 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.069
Training Epoch 22 --- Training Accuracy:  93.3%, Validation Accuracy: 100.0%,  Validation Loss: 0.062
Training Epoch 23 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.043
Training Epoch 24 --- Training Accuracy:  86.7%, Validation Accuracy: 100.0%,  Validation Loss: 0.028
Training Epoch 25 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.042
```

Figure.13

Testing the model

The stored model graph is restored and the network graph is recreated. After loading the stored weights, the restored graph is accessed. The test inputs are fed into the input placeholders using a feed dictionary and a tensorflow session is run. Output probabilities for all classes are obtained from the output tensor node. The class with the maximum probability is output as the result(predicted class). Predicted class for each test dataset is displayed against the corresponding dataset on the terminal. Accuracy of testing is also displayed. Before testing, the user should specify the number of test datasets being used, number of classes and batch size to be used.
Accuracy of testing can be increased by finding the right combination of batch size and the number of iterations in training process.

Command to test:
python3 cnn_test.py



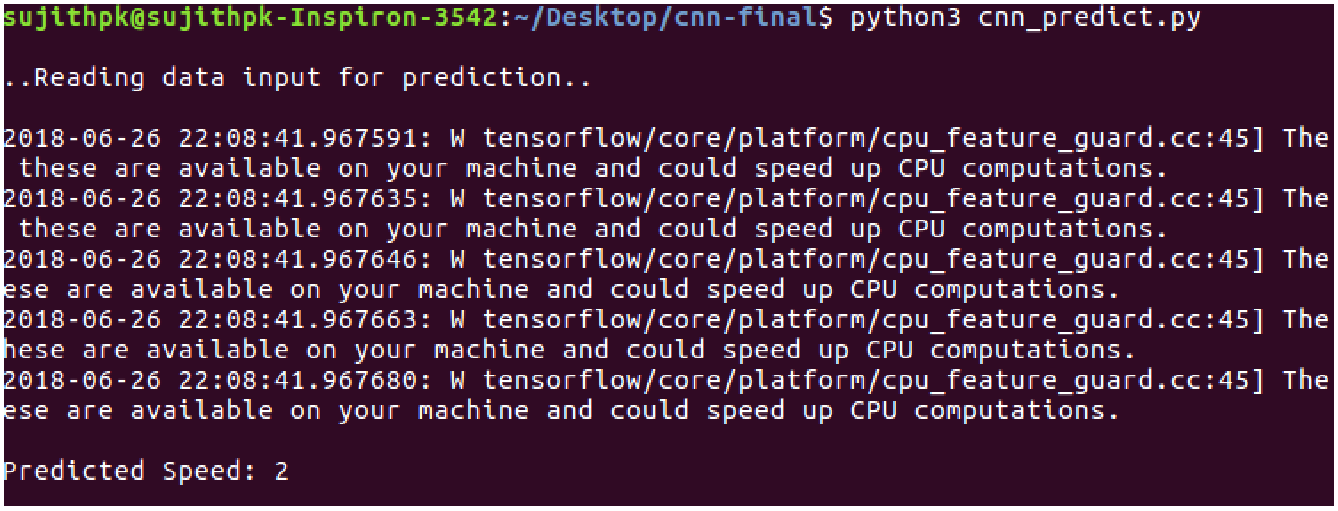
```
sujithpk@sujithpk-Inspiron-3542:~/Desktop/cnn$ python3 cnn.py

.......Reading data inputs.......

WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmpt06qcugt
2018-06-02 12:02:15.933189: I tensorflow/core/platform/cpu_feature_guard.cc:140]
mpiled to use: AVX2 FMA

.......Training completed.......


Time taken for training : 0.6408790866533915 minutes( 38.45275068283081 sec.)

Test data: 1    Predicted Speed: 1
Test data: 2    Predicted Speed: 2
Test data: 3    Predicted Speed: 3
Test data: 4    Predicted Speed: 1
Test data: 5    Predicted Speed: 3
Test data: 6    Predicted Speed: 1
Test data: 7    Predicted Speed: 2
Test data: 8    Predicted Speed: 2
Test data: 9    Predicted Speed: 1
Test data: 10    Predicted Speed: 3
Test data: 11    Predicted Speed: 1
Test data: 12    Predicted Speed: 2

Accuracy : 100.0 %

Done..
TOTAL TIME TAKEN  : 0.6508447051048278 minutes. ( 39.050684452056885 seconds)

sujithpk@sujithpk-Inspiron-3542:~/Desktop/cnn$
```

Figure.14

Running ML to predict the results

During prediction of results, the stored model graph is restored and the network graph is recreated. After loading the stored weights, the restored graph is accessed. Live data obtained from the fan for prediction is fed into the input placeholder using a feed dictionary and a tensorflow session is run. Output probabilities for all classes are obtained from the output tensor node. The class with the maximum probability is output as the predicted class.

Command to predict:
python3 cnn_predict.py



```
sujithpk@sujithpk-Inspiron-3542:~/Desktop/cnn-final$ python3 cnn_predict.py

..Reading data input for prediction..

2018-06-26 22:08:41.967591: W tensorflow/core/platform/cpu_feature_guard.cc:45] The
 these are available on your machine and could speed up CPU computations.
2018-06-26 22:08:41.967635: W tensorflow/core/platform/cpu_feature_guard.cc:45] The
 these are available on your machine and could speed up CPU computations.
2018-06-26 22:08:41.967646: W tensorflow/core/platform/cpu_feature_guard.cc:45] The
ese are available on your machine and could speed up CPU computations.
2018-06-26 22:08:41.967663: W tensorflow/core/platform/cpu_feature_guard.cc:45] The
hese are available on your machine and could speed up CPU computations.
2018-06-26 22:08:41.967680: W tensorflow/core/platform/cpu_feature_guard.cc:45] The
ese are available on your machine and could speed up CPU computations.

Predicted Speed: 2
```

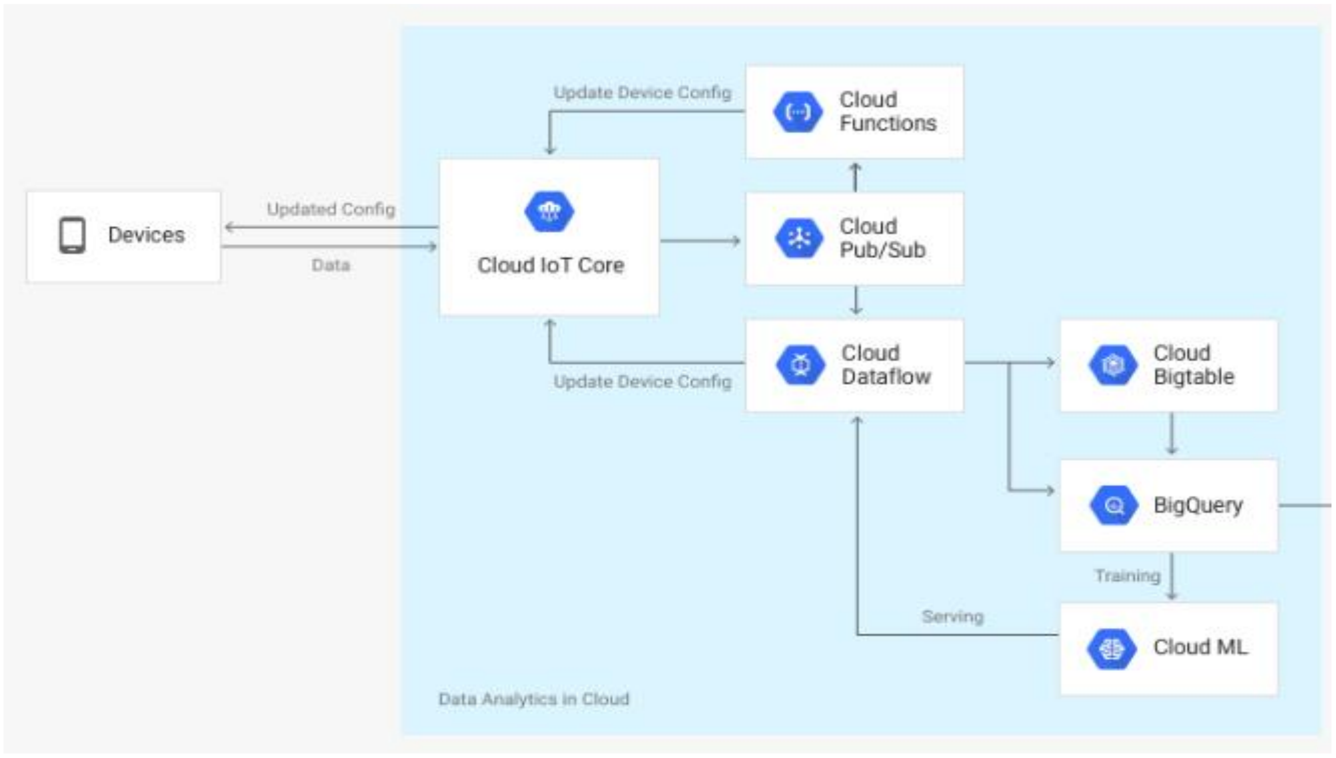
Figure.15

Running ML on Google Cloud Platform (GCP):

Introduction and Architecture of GCP:

Google Cloud Platform, offered by Google, is a suite of cloud computing services. Google Cloud Platform offers services for compute, storage, networking, big data, machine learning and the internet of things (IoT), as well as cloud management, security and developer tools.

Figure.16 *Google Cloud platform architecture*



Google Cloud Platform offers application development and integration services. For example, Google Cloud Pub/Sub is a managed and real-time messaging service that allows messages to be exchanged between applications.

The below figure shows the architecture of the project.

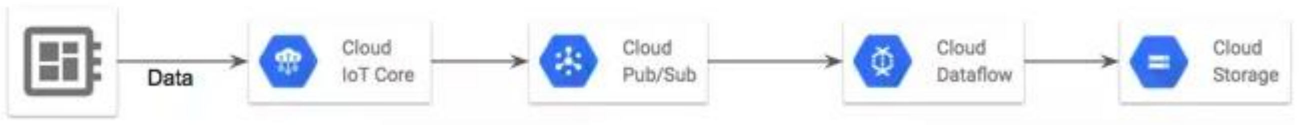


Figure.17

The main aim of using GCP is getting the live streaming data from the fan and storing the data in cloud and running Machine learning to predict the speed of rotation of fan. The data that we get from the sensors installed on the fan (raspberry pi terminal) are published to a PubSub topic using cloud IoT core.

Using Cloud dataflow pipelining the published live streaming data is stored in cloud storage as a file.

The file is accessed by cloud ML which is fed into the stored CNN network to predict the speed.

Setting up Cloud IoT Core:

Cloud IoT Core is a fully managed service that allows users to easily and securely connect, manage, and ingest data from millions of globally dispersed devices. Cloud IoT Core, in combination with other services on Google Cloud IoT platform, provides a complete solution for collecting, processing, analyzing, and visualizing IoT data in real time to support improved operational efficiency.

Steps in setting up the cloud IoT core:

- Create a new project In the Google cloud console
- Enable billing for the project.
- Enable APIs for Cloud IoT core.

  In the local environment (RPi terminal) ,Google cloud SDK is installed and Initialised as follows:

- Google cloud SDK currently doesn't support Python3. Hence check the Python2 version installed. The version should be 2.7.9 or later.

  Python2 --version

- Download the Cloud SDK package for Linux.
- Extract the package and open a new terminal window.

- To initialize the SDK:

  Run the following at a command prompt:

  gcloud init

- accept the option to log in by entering 'y' when prompted.
- In a new browser window log in to Google user account when prompted and click Allow to grant permission to access Google Cloud Platform resources .
- Choose the cloud project to be used.
- Choose the compute zone (us-central1 is preferred).

The above procedure authenticates the user to use Google Cloud SDK.

Registering the Device (RPi)

Now in the cloud Iot Core page open up google cloud console and create a new device registry as follows:

- Enter Registry ID, Cloud region and select MQTT for the protocol.
- In the Telemetry Topic, create a topic and then click create topic
- click create on cloud iot page.

Now add device to the registry

- on the registry details page click add device and enter Device ID.
- select Allow for communication.
- click Add

Add public key to the device:

- open the terminal and run the following to create RS256 key

  ```
  openssl req -x509 -newkey rsa:2048 -keyout rsa_private.pem -nodes \
  -out rsa_cert.pem -subj "/CN=unused"
  ```
- copy the contents of rsa_cert.pem file
- on device details page click add public key,select RS256_X509 for public key format.
- Paste the value in Public key value box.
- click Add

Setting up PubSub:

Cloud Pub/Sub is a fully-managed real-time messaging service that allows users to send and receive messages between independent applications. Cloud Pub/Sub brings the scalability, flexibility, and reliability of enterprise message-oriented middle ware to the cloud. By providing many-to-many, asynchronous messaging that decouples senders and receivers, it allows for secure and highly available communication between independently written applications. Cloud Pub/Sub delivers low-latency, durable messaging that helps developers quickly integrate systems hosted on the Google Cloud Platform and externally.

- Go to cloud pubsub topics page
- Create a Topic

- Open menu of the created topic
- Create new subscription by selecting the delivery type ‘Pull’

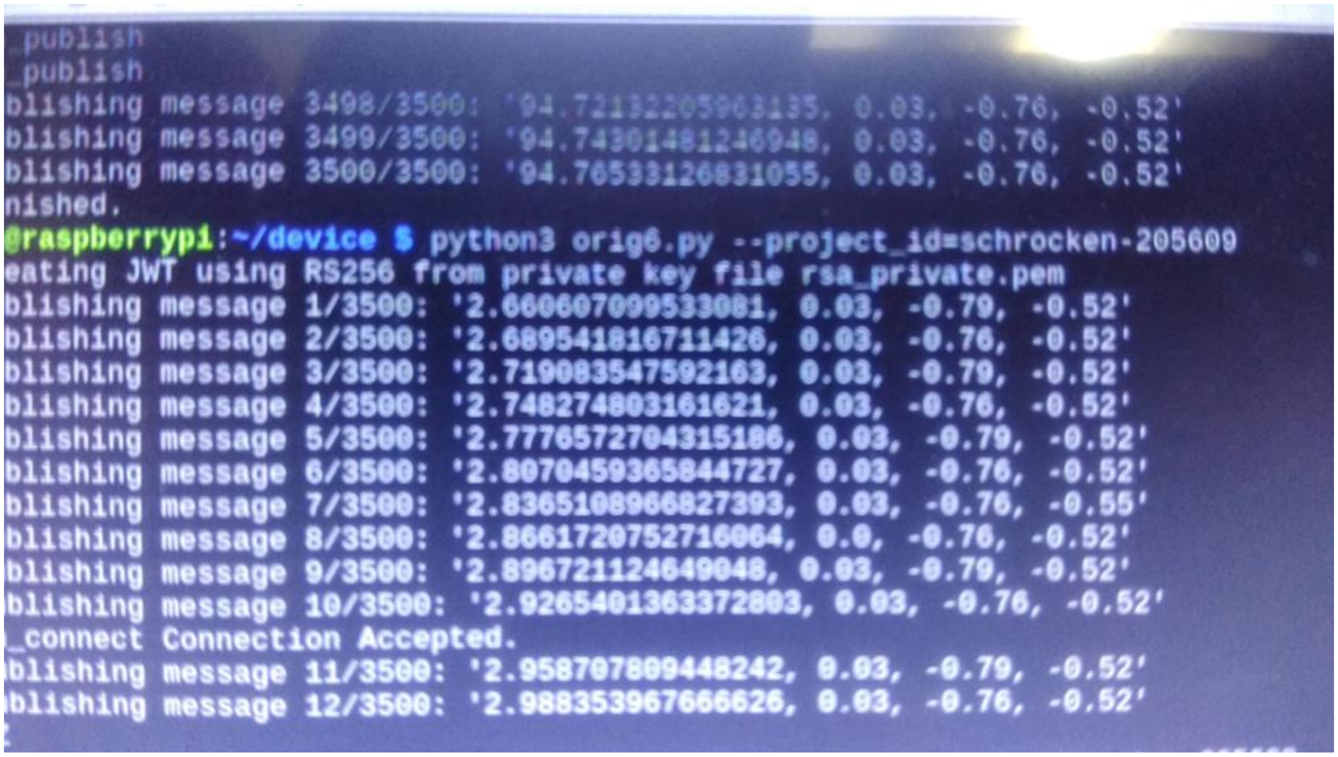
Streaming data from device to Pubsub topic:

As we need to send data from local device to cloud platform, we send it to the specific PubSub topic and run a dataflow job to store the data in cloud storage. So, to send data from RPi to PubSub topic the previous code is modified such that it uses MQTT client to publish data from RPi to the PubSub topic.This MQTT client bridges between RPi and pubsub topic.

By running the code **sensor_data_publishing.py** on terminal by providing the details of
project_id,
device_id,
registry_id,
private key file,
algorithm,
cloud region,
ca_certs,
MQTT bridge details we can get the local device data into pubsub topic mentioned previously.

- Command to run the code is
  *python3 sensor_data_publishing.py *
  --project_id=<project id of the project>

Get the ca_certs file **roots.pem** from https://pki.google.com/roots.pem
The given mqtt_bridge_hostname as mqtt.googleapis.com and bridge port as 8883 are default sources.



```
_publish
_publish
blishing message 3498/3500: '94.7213220596313 5, 0.03, -0.76, -0.52'
blishing message 3499/3500: '94.74301481246948, 0.03, -0.76, -0.52'
blishing message 3500/3500: '94.76533126831055, 0.03, -0.76, -0.52'
nished.
@raspberrypi:~/device $ python3 orig6.py --project_id=schrocken-205609
eating JWT using RS256 from private key file rsa_private.pem
blishing message 1/3500: '2.660607099533081, 0.03, -0.79, -0.52'
blishing message 2/3500: '2.689541816711426, 0.03, -0.76, -0.52'
blishing message 3/3500: '2.719083547592163, 0.03, -0.79, -0.52'
blishing message 4/3500: '2.748274803161621, 0.03, -0.76, -0.52'
blishing message 5/3500: '2.7776572704315186, 0.03, -0.79, -0.52'
blishing message 6/3500: '2.8070459365844727, 0.03, -0.76, -0.52'
blishing message 7/3500: '2.8365108966827393, 0.03, -0.76, -0.55'
blishing message 8/3500: '2.8661720752716064, 0.0, -0.76, -0.52'
blishing message 9/3500: '2.896721124649048, 0.03, -0.79, -0.52'
blishing message 10/3500: '2.9265401363372803, 0.03, -0.76, -0.52'
_connect Connection Accepted.
blishing message 11/3500: '2.958707809448242, 0.03, -0.79, -0.52'
blishing message 12/3500: '2.988353967666626, 0.03, -0.76, -0.52'
```

Figure.18

Setting up GOOGLE CLOUD STORAGE:

Google Cloud Storage is a unified object storage for developers and enterprises. Cloud Storage allows world-wide storage and retrieval of any amount of data at any time. Users can use Cloud Storage for a range of scenarios including serving website content, storing data for archival and disaster recovery, or distributing large data objects to users via direct download.

- Go to Cloud storage page in Google cloud console
- Create a storage bucket by selecting appropriate region and location.

Setting up CLOUD DATAFLOW pipelining:

Cloud Dataflow is a unified programming model and a managed service for developing and executing a wide variety of data processing patterns. Cloud Dataflow includes SDKs for defining data processing workflows, and a Cloud Platform managed service to run those workflows on Google Cloud Platform resources such as Compute Engine, BigQuery, and more.

It is a fully-managed service for transforming and enriching data in stream (real time) and batch (historical) modes with equal reliability. Its serverless approach to resource provisioning and management, gives user access to virtually limitless capacity to solve big data processing challenges.

Setting up authentication :

- Enable the Cloud Dataflow, Compute Engine, Stackdriver Logging, Google Cloud Storage, Google Cloud Storage JSON, BigQuery, and Google Cloud Resource Manager APIs.

- Go to the Create service account key page in the GCP Console.
- create new service account and select Role as Project>owner.
- click create.

This downloads a JSON file that contains the service account key to computer storage.

1. Set the environment variable GOOGLE_APPLICATION_CREDENTIALS to the file path of the Json file by the below command.

   export GOOGLE_APPLICATION_CREDENTIALS="[PATH]"

Live streaming of data from PubSub to Cloud Storage

By running **PubSubReader.java** the Dataflow job can be created and after the creation of job run **sensor_data_publishing.py** code on terminal to send data to pubsub topic. This dataflow job collects the data from pubsub topic using dataflow pipelining.

To run **PubSubReader.java** code on cloud shell, install apache.beam library. In the code the pubsub topic from which the data is to be read and the cloud storage location to where the output file has to be created are mentioned.

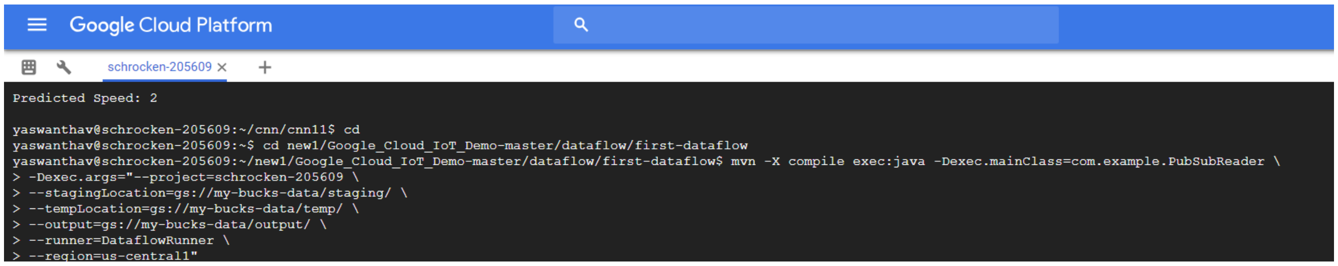
The main part of the code which creates the data flow job is

```
p.apply(PubsubIO.readStrings().fromTopic(options.getPubSubTopic()))
 .apply(Window.<String>into(FixedWindows.of(Duration.standardMinutes(3))))
 .apply(TextIO.write().to(options.getOutput()).withWindowedWrites().withNumShards(1));
```

The first line of code opens the PubSub MQTT Topic as an input source of the Pipeline. In the second line we apply Windowing which segments the incoming data-stream. This is needed as we are reading from a stream oriented input-source and are writing to a batch oriented output-source. The third line writes the received data to the Cloud Storage as an Object, while combining 3-minute of collected data into one Object.

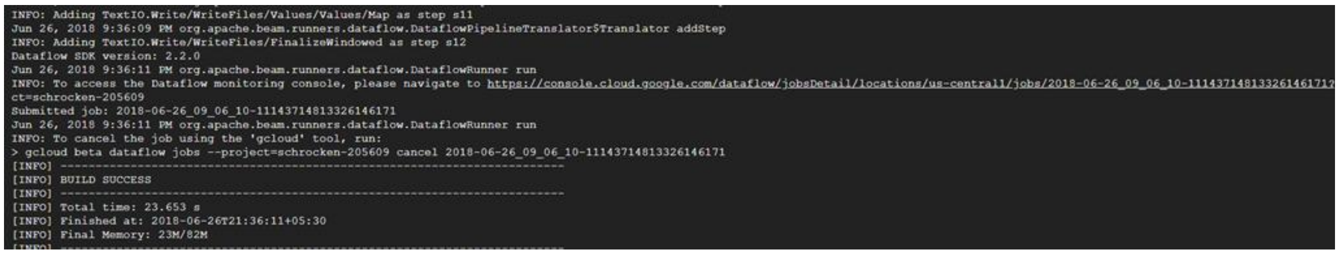
Compiling and running the code:
we use the following commands to compile and run the dataflow job code

```
$ mvn compile exec:java -Dexec.mainClass=com.example.PubSubReader \
                        -Dexec.args="--project=<Project Name> \
                        --stagingLocation=<Bucket Name>/staging/ \
                        --tempLocation=<Bucket Name>/temp \
                        --output=<Bucket Name>/output/ \
                        --pubSubTopic=<Name of created topic> \
                        --runner=DataflowRunner \
                        --region=<cloud region>"
```



```
Predicted Speed: 2

yaswanthav@schrocken-205609:~/cnn/cnn11$ cd
yaswanthav@schrocken-205609:~$ cd new1/Google_Cloud_IoT_Demo-master/dataflow/first-dataflow
yaswanthav@schrocken-205609:~/new1/Google_Cloud_IoT_Demo-master/dataflow/first-dataflow$ mvn -X compile exec:java -Dexec.mainClass=com.example.PubSubReader \
> -Dexec.args="--project=schrocken-205609 \
> --stagingLocation=gs://my-bucks-data/staging/ \
> --tempLocation=gs://my-bucks-data/temp/ \
> --output=gs://my-bucks-data/output/ \
> --runner=DataflowRunner \
> --region=us-central1"
```

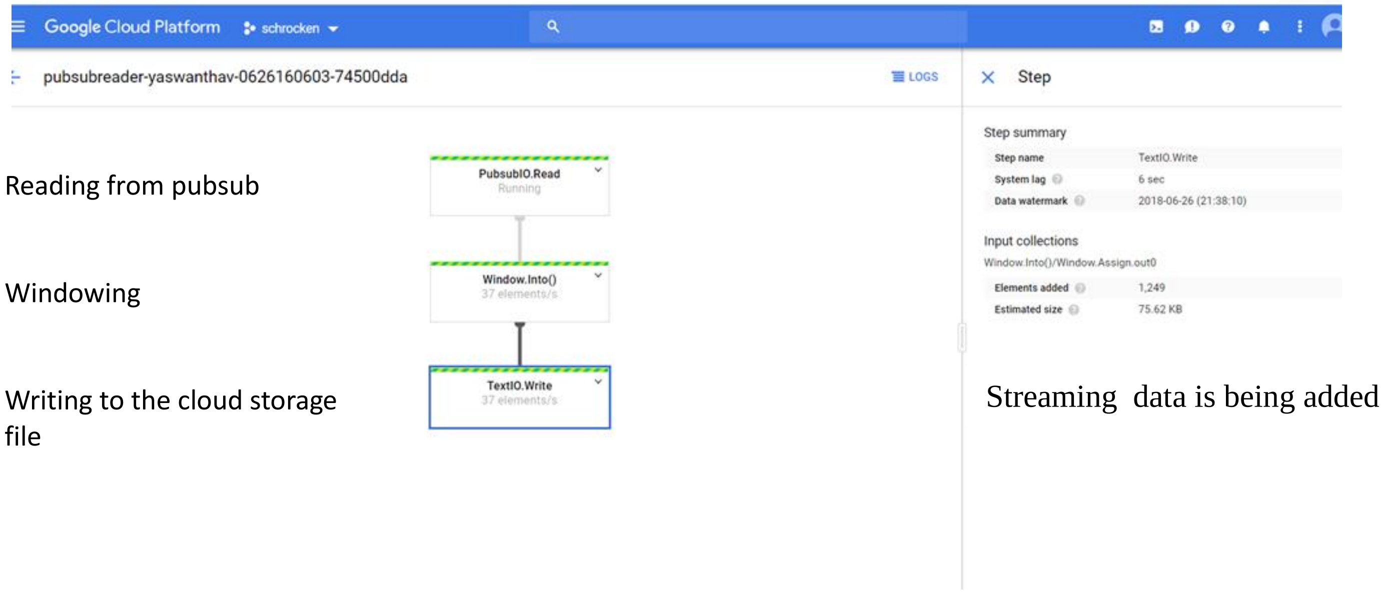
Figure.19 Running dataflow



```
INFO: Adding TextIO.Write/WriteFiles/Values/Values/Map as step s11
Jun 26, 2018 9:36:09 PM org.apache.beam.runners.dataflow.DataflowPipelineTranslator$Translator addStep
INFO: Adding TextIO.Write/WriteFiles/FinalizeWindowed as step s12
Dataflow SDK version: 2.2.0
Jun 26, 2018 9:36:11 PM org.apache.beam.runners.dataflow.DataflowRunner run
INFO: To access the Dataflow monitoring console, please navigate to https://console.cloud.google.com/dataflow/jobsDetail/locations/us-central1/jobs/2018-06-26_09_06_10-11143714813326146171?
ct=schrocken-205609
Submitted job: 2018-06-26_09_06_10-11143714813326146171
Jun 26, 2018 9:36:11 PM org.apache.beam.runners.dataflow.DataflowRunner run
INFO: To cancel the job using the 'gcloud' tool, run:
> gcloud beta dataflow jobs --project=schrocken-205609 cancel 2018-06-26_09_06_10-11143714813326146171
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 23.653 s
[INFO] Finished at: 2018-06-26T21:36:11+05:30
[INFO] Final Memory: 23M/82M
```

Figure.20 build

After dataflow job is built, a link ,which is the link to the built dataflow log page,is shown on the cloud shell. (underlined portion in figure.build)



Step summary

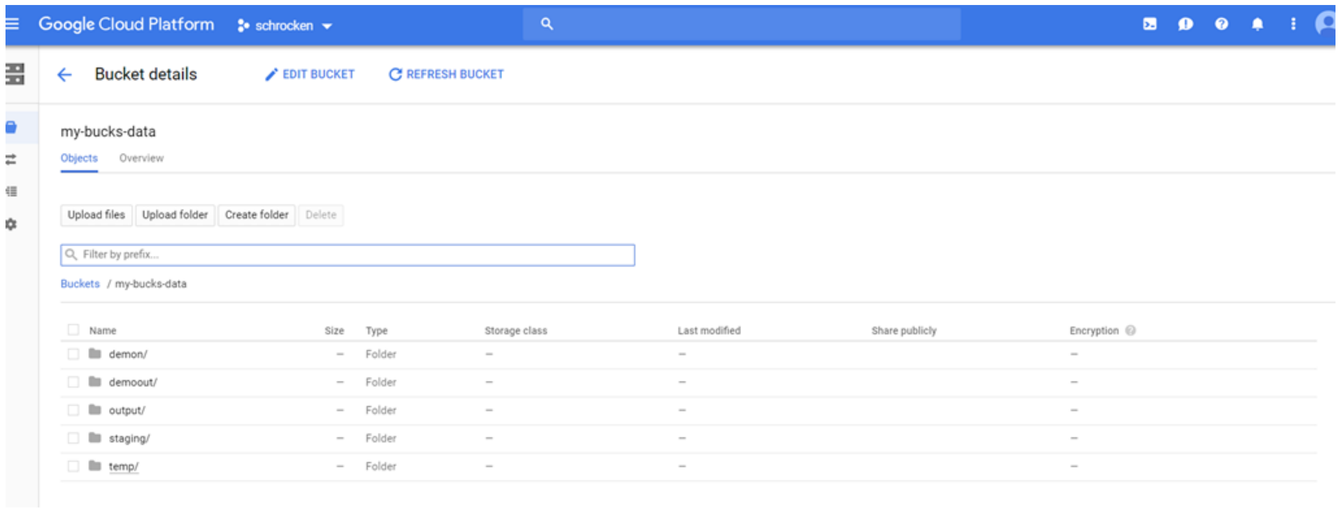
Step name	TextIO.Write
System lag	6 sec
Data watermark	2018-06-26 (21:38:10)

Input collections

Window.Into()/Window.Assign.out0

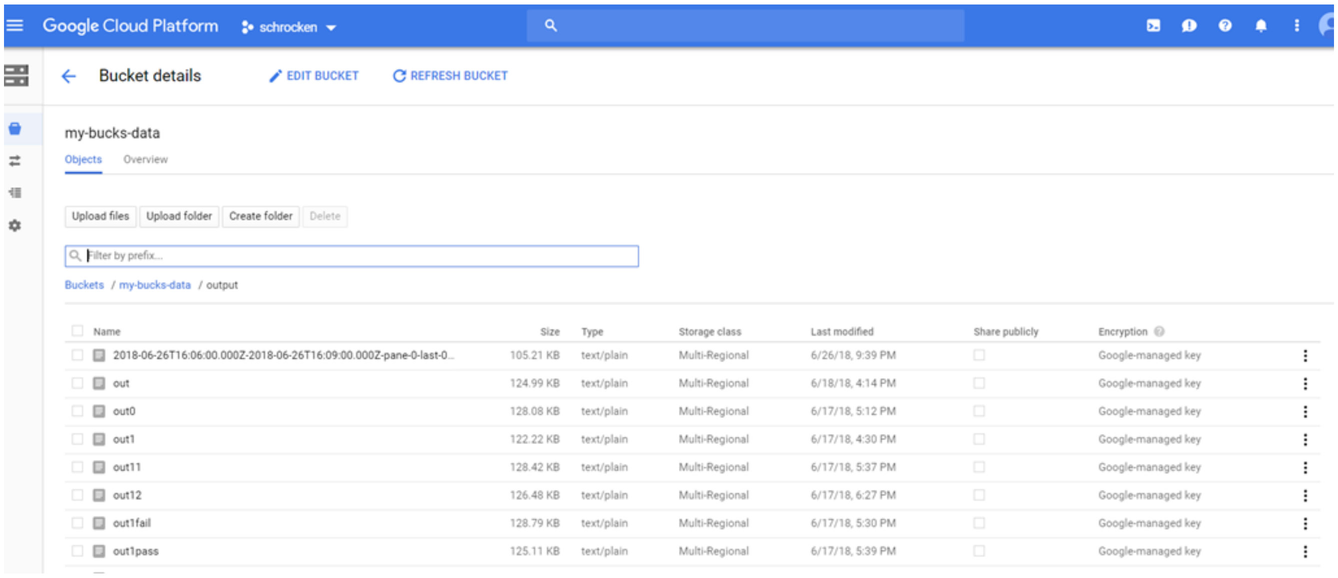
Elements added	1,249
Estimated size	75.62 KB

Figure.21 Dataflow log Page



Name	Size	Type	Storage class	Last modified	Share publicly	Encryption
demon/	–	Folder	–	–		–
demoout/	–	Folder	–	–		–
output/	–	Folder	–	–		–
staging/	–	Folder	–	–		–
temp/	–	Folder	–	–		–

Figure.22 This shows the folders created after running the dataflow



Name	Size	Type	Storage class	Last modified	Share publicly	Encryption
2018-06-26T16:06:00.000Z-2018-06-26T16:09:00.000Z-pane-0-last-0...	105.21 KB	text/plain	Multi-Regional	6/26/18, 9:39 PM	☐	Google-managed key
out	124.99 KB	text/plain	Multi-Regional	6/18/18, 4:14 PM	☐	Google-managed key
out0	128.08 KB	text/plain	Multi-Regional	6/17/18, 5:12 PM	☐	Google-managed key
out1	122.22 KB	text/plain	Multi-Regional	6/17/18, 4:30 PM	☐	Google-managed key
out11	128.42 KB	text/plain	Multi-Regional	6/17/18, 5:37 PM	☐	Google-managed key
out12	126.48 KB	text/plain	Multi-Regional	6/17/18, 6:27 PM	☐	Google-managed key
out1fail	128.79 KB	text/plain	Multi-Regional	6/17/18, 5:30 PM	☐	Google-managed key
out1pass	125.11 KB	text/plain	Multi-Regional	6/17/18, 5:39 PM	☐	Google-managed key

Figure.23 This shows the file created (first file)

Caution: Running the above code generates Compute engine instances and dataflow pipelines for which google charges money.Better use if free trail is available.

Running ML to predict the results

In the google cloud shell (gcloud) navigate to the directory containing the stored model and checkpoint files.
Required libraries like Tensorflow, Numpy, Scipy, google_cloud_storage, etc need to be installed (if not installed already) before running machine learning.

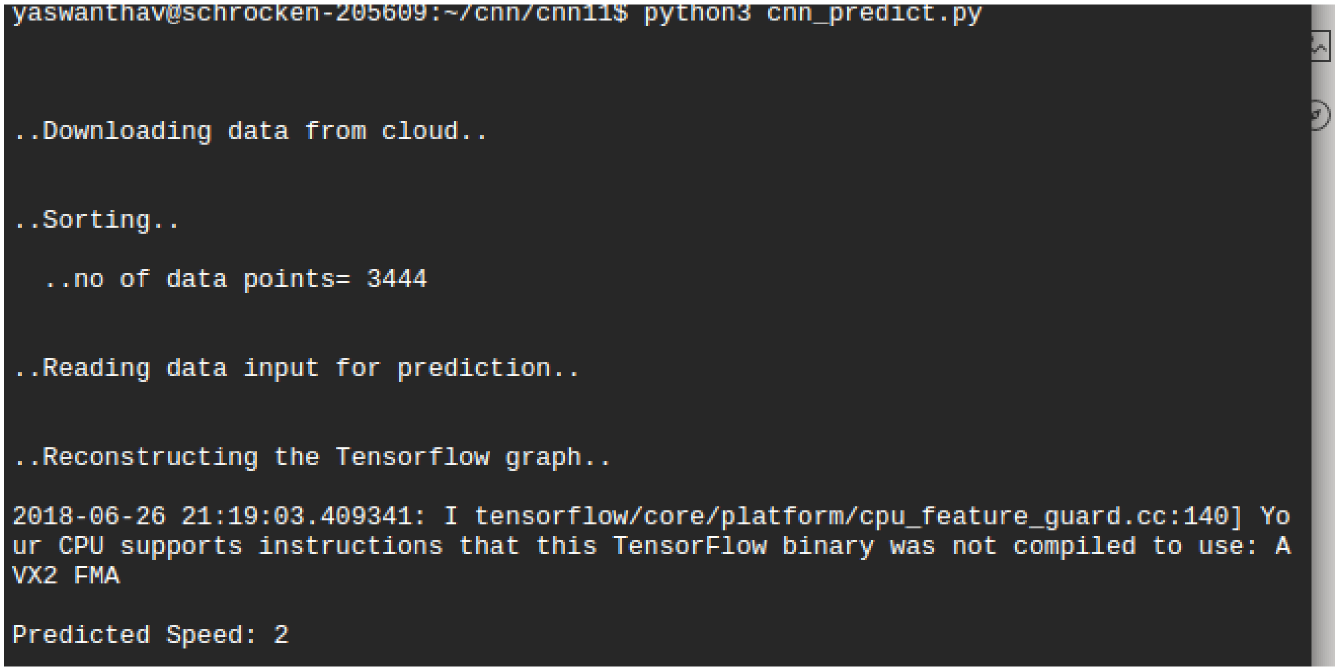
The stored model graph is restored and the network graph is recreated. After loading the stored weights, the restored graph is accessed by running *cnn_predict.py* file which uses *inp_file_pred.py* as the input file to get the data.

Live data obtained from the fan for prediction ,which is stored in the google cloud storage is downloaded into a local file in the gcloud shell (in csv format). To download from cloud storage the user should authenticate the session with valid license file that contains service account key and specify the storage location from which data needs to be retrieved.

The data downloaded need not be in order , as the data pulled from any PubSub topic don’t follow any order. Hence, the data is sorted based on the time of publishing. This input is fed into the input placeholder using a feed dictionary, and then a tensorflow session is run. Output probabilities for all classes are obtained from the output tensor node. The class with the maximum probability is output as the predicted class.

Command to predict the speed:
python3 cnn_predict.py

During prediction, the number of data points downloaded from cloud storage is displayed in the gcloud shell.



```
yaswanthav@schrocken-205609:~/cnn/cnn11$ python3 cnn_predict.py



..Downloading data from cloud..


..Sorting..

  ..no of data points= 3444


..Reading data input for prediction..


..Reconstructing the Tensorflow graph..

2018-06-26 21:19:03.409341: I tensorflow/core/platform/cpu_feature_guard.cc:140] Yo
ur CPU supports instructions that this TensorFlow binary was not compiled to use: A
VX2 FMA

Predicted Speed: 2
```

Figure.24 result

References :

- **Raspbian OS**
- **understanding ADC**
- **ADC to Raspberry Pi**
- **LCD display**
- **controlling an SPI device**
- **Temperature sensor**
- **Tensorflow**
- **convolutional neural networks**
- **CNN example reference**
- **cloud iot core , cloud dataflow , cloud pubsub**
- **cloud Dataflow**

Along with the above references the datasheets of the sensors, google cloud documentations and Wikipedia etc. are also referred.