Machine Learning Engineer Nanodegree

Capstone Project

Akash Suresh

February 2nd, 2019

I. Definition

Project Overview

India is one of the world's largest producer of incense and is a healthy exporter to other countries. Burning of incense in India dates back to thousands of years, and India exported the idea to China, Japan and other Asian countries. The primary form of burning incense in India is the incense stick or agarbathi. The basic ingredients of an incense stick are bamboo sticks, paste (generally made of charcoal dust or sawdust and joss/jiggit/gum/tabu powder – an adhesive made from the bark of litsea glutinosa and other trees), and the perfume ingredients.

Incense sticks were traditionally hand-rolled. Of late, through breakthroughs in

technology, incense sticks are manufactured using machinery. These machines are fed with the paste and bamboo sticks as input and mass produce incense sticks. Only downside being, the machine is not always efficient, and might produce subpar incense sticks (around 30-40% of the time). Currently, they use a worker to segregate the good sticks from the bad ones. These workers are paid around INR 500-1000/day. Automating this process could bring down the cost of production by a large margin for the industry.

My dad owns one such industry and this project is to integrate an image processing module for every machine. I propose to build a CNN model which will classify the good sticks from the bad by using the expertise gained in this course in order to help my dad cut down costs. I will be helping him reduce his workforce by 1 worker/machine. (We have around 50 machines). Thus saving him about INR 1.5 million per year.

Problem Statement

The incense stick manufacturing machine, takes the sticks and paste as input, and rolls out the formed incense sticks on a conveyor belt. A Raspberry Pi will be integrated with the machine to automate the segregation process. The problem at hand is to design a CNN that classifies camera procured images of the sticks on the conveyor belt as good or bad with at least 85% accuracy and at the same time is simple enough to run on a Raspberry Pi.

Evaluation Metrics

The evaluation metric or **score** will be a comparison between the accuracy of the model, and also the size of the model. Maybe a combination of both will be the best metric to attain optimal performance since the performance of the model in this kind of setting depends on accuracy directly and size inversely. This also gives a way to normalize the results to compare it with the benchmark.

$$\boldsymbol{score} = \frac{accuracy}{size(model)}$$

II. Analysis

Data Exploration

The dataset for this project is generated manually. I ran a script to capture photos of the sticks on the conveyor belt and manually classified them as good/bad. This dataset is used as training images for my CNN.

Each image is a square image, shot on the Raspberry Pi camera. The bamboo stick used for making the incense stick will be captured in the frame.

Each photo is a 480x480 RGB image. We have around 4500 photos, after cleaning up (discarding the empty frames), and manually tagging them as Good or Bad. Here is how the data distribution looks like

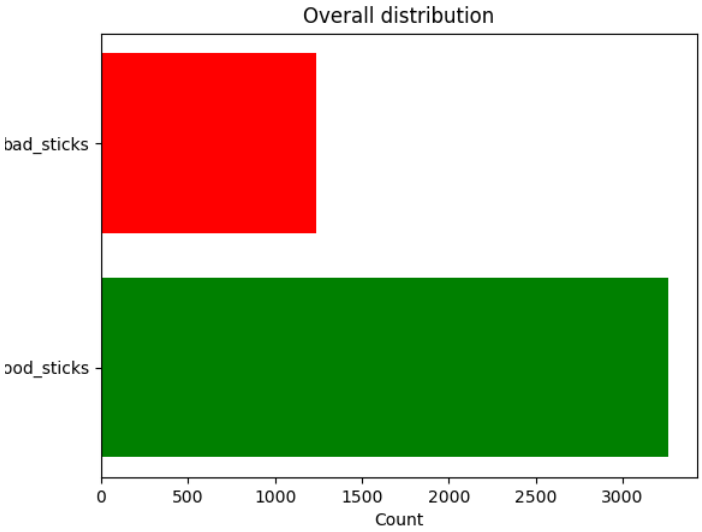
Type	**Number of images**	**Percentage**
Good	3264	71.9%
Bad	1236	28.1%

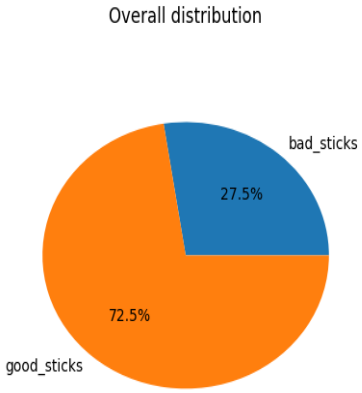
We can understand more about the data, by looking at the visualizations in the next subsection.

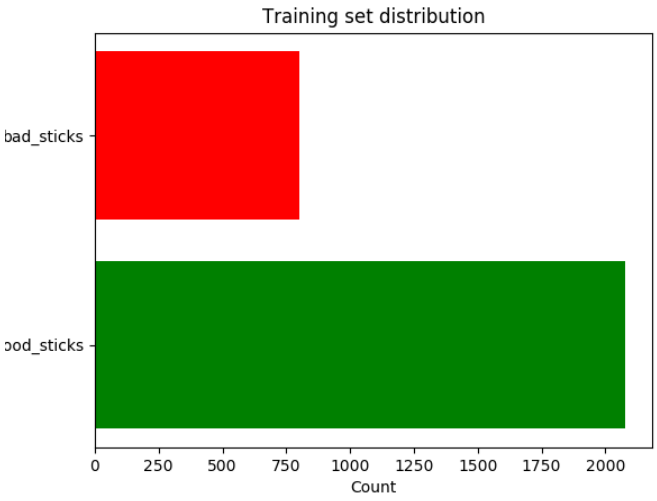
Exploratory Visualization

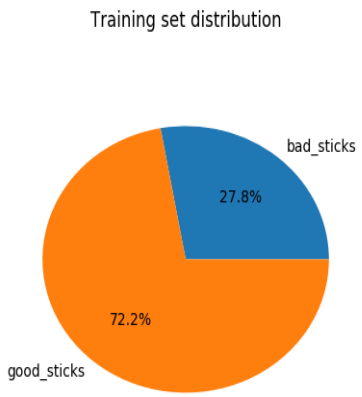
The train test split was done with a ratio of 80:20 from the overall dataset. And the train validation split was done with a ratio of 80:20 from the training dataset.

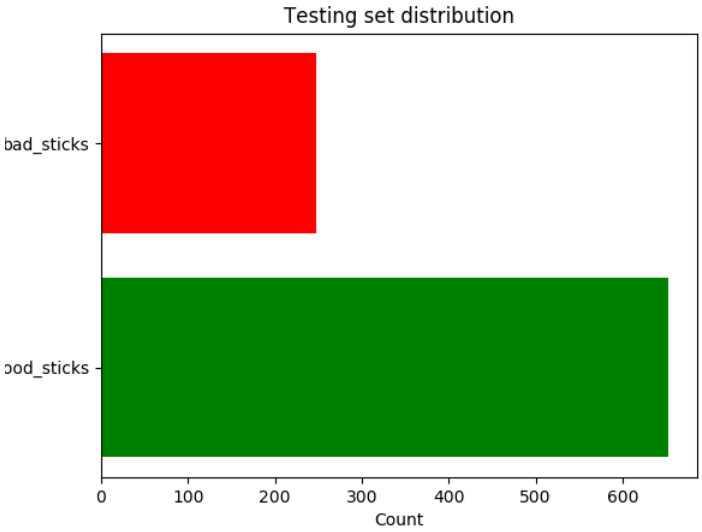
I have used Bar plots and Pie charts for visualizing the data, as these provide information about the distributions for the split made (Training, testing and validation).

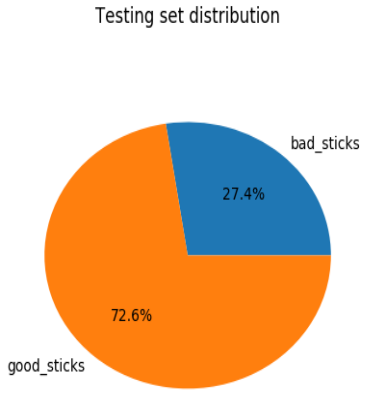

Overall distribution
bad_sticks
ood_sticks
0
500
1000
1500
2000
2500
3000
Count

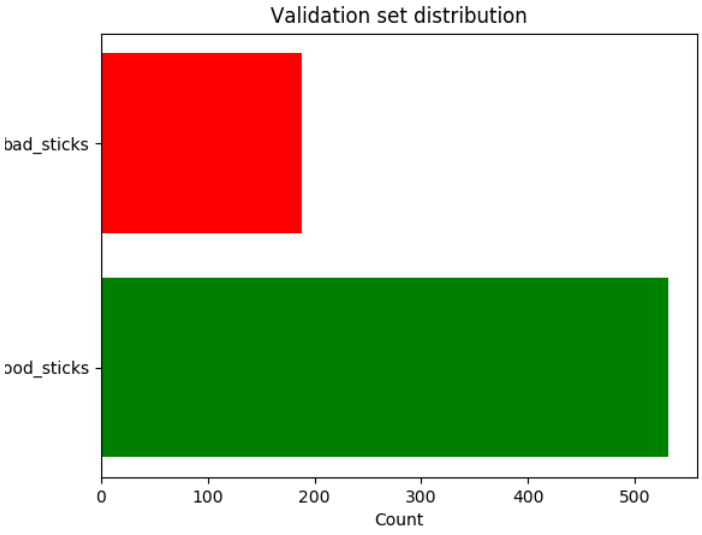

Overall distribution
bad_sticks
27.5%
72.5%
good_sticks

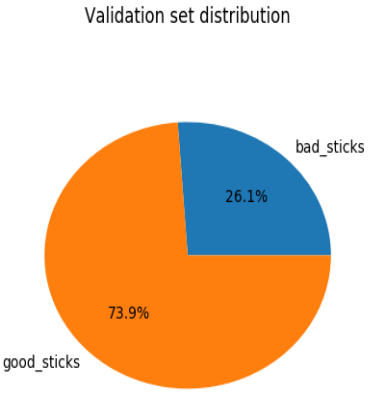

Training set distribution
bad_sticks
ood_sticks
0
250
500
750
1000
1250
1500
1750
2000
Count


Training set distribution
bad_sticks
27.8%
72.2%
good_sticks









We can see that the data has been split well, keeping the distribution consistent across the splits. This will help avoid inherent bias in our model.

Algorithms and Techniques

To solve this problem of identifying the bad sticks from the good sticks. We will be using a Convolutional Neural Network (CNN) to classify the sticks into good or bad, based on the images. The key aspect being that the model, should be compact enough to work real-time with the machine. So the size of the model becomes very crucial.

Here is the CNN architecture I started out with. I took the input image, and converted it into a 4D tensor suitable for Keras CNN.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 480, 480, 16)      448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 240, 240, 16)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 240, 240, 32)      2080
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 120, 120, 32)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 120, 120, 64)      8256
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 60, 60, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 60, 60, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 230400)            0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               115200500
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 1002
=================================================================
Total params: 115,212,286.0
Trainable params: 115,212,286.0
Non-trainable params: 0.0
_________________________________________________________________
```

I chose this model to begin with, since we need to identify the features from the images using the Convolutional layer and then we need to classify whether the stick is good or bad.

From this, I went on removing layers and reducing the parameters to make the model lightweight and compact to make predictions real-time with the machine.

Benchmark

I wanted to use the Xception model, as the benchmark model. But, I had a lot of difficulty in training this model on my laptop. I wasted a lot of time trying to run this, but my efforts were futile. Since, my main use case was the real-time prediction of the sticks, and not the accuracy of the model, I shifted my focus to work on my model, and get it to work with the Incense stick machine.

Since there is no research or open source models built for this application purpose, there is no standardized comparison benchmark for the same.

III. Methodology

Data Preprocessing

For this project, I had to run a script on the Raspberry Pi to capture images of the sticks on the conveyor belt. This was challenging since the conveyor belt runs at a very high speed. I had to change the configuration of the Raspberry Pi camera to capture photos at 90 frames per second, so that we won't get blurred images. Timing the camera to get photos of the sticks was the second challenge, I opted a high interval rate, and got a lot of empty frames.

To clean-up the data, I had to remove all the empty frames amongst the captured photos. Then I had to manually go through each photo and classify them to make up the training dataset. This was a tedious process.

Implementation

Part 1 - Setting up the Raspberry Pi

For this project, I integrated a Raspberry Pi 2 directly with the machine. The Raspberry Pi used a camera module, to capture photos. This was used to generate the dataset, as well as predict the quality of the sticks in realtime. I had to install and setup Keras, TensorFlow, OpenCV, and the supporting packages on the Raspberry Pi in the best way to optimize performance.

Part 2 - Generating Dataset

I wrote a python script, to shoot photos of the sticks on the conveyor belt, at

a fixed interval and save them to the disk.

```
def datasetGen(count, interval, width):
    # initialize the video stream and allow the camera sensor to warm up
    print("[INFO] starting video stream...")
    vs = VideoStream(usePiCamera=True, resolution=(480,360), framerate=90).start()
    time.sleep(2.0)
    counter = 10000
    # loop over the frames from the video stream
    for i in range(count):
        frame = vs.read()
        frame = imutils.resize(frame, width=480)
        fileName = str(counter+i) + '.png'
        image = cv2.resize(frame, (width, width))
        cv2.imwrite(fileName, image)
        time.sleep(interval)

    # do a bit of cleanup
    print("[INFO] cleaning up...")
    cv2.destroyAllWindows()
    vs.stop()
```

Around 6000 photos were shot, out of which we had a lot of empty frames, which were removed while cleaning up. And then these images, had to be classified manually as good or bad. The tough part was getting a clear image, as the conveyor belt was moving at a high speed. I had to change the camera configuration to record frames at 90 frames per second. This helped. Another challenge, was to get the interval right. This took some trial and error. And I was able to figure out the interval.

Part 3 - Building a CNN

To build a model to classify these images, I started out with a basic CNN architecture. (As mentioned in the Algorithms and Techniques subsection).

This model gave an accuracy of 86.4%. This was a great start. But the downside was that the model size was 700 MB, this took a lot of time to predict, which was not favourable. To get a decent realtime performance, the model had to be somewhere around 20-25 MB. There was a lot of experimentation done here, to reduce the model size. (more in the Refinement section)

Part 4 - Realtime evaluation

The model built in Part 3, was run on the Raspberry Pi, to classify the sticks in realtime. I wrote a python script for the predicting, which snaps a photo in intervals, and predicts the quality. The interval being very small, the model had to be very fast to do all the computations quickly to predict the output. This output

was fed to a GPIO pin, which activated a mechanical apparatus to remove the stick from the conveyor.

Refinement

I started out with a basic CNN architecture, shown below

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 480, 480, 16)      448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 240, 240, 16)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 240, 240, 32)      2080
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 120, 120, 32)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 120, 120, 64)      8256
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 60, 60, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 60, 60, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 230400)            0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               115200500
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 1002
=================================================================
Total params: 115,212,286.0
Trainable params: 115,212,286.0
Non-trainable params: 0.0
_________________________________________________________________
```

Here are a couple of things I tried -

- Remove the third Convolutional layer. Since, we were dealing with images with simple features we don't need three stacked Convolutional layers.
- Reduced the input image size, from 480x480 to 224x224 pixels. Since, we were dealing with simple images, there weren't any details we were missing out by reducing the size.
- Convert the images, from RGB to grey scale. Since, our images don't have any information based on colours, we can as well convert to grey scale. This will help reduce parameters.

Using combinations of all this I was able to iteratively, through trial and error, get the size of the model down to 20 MB.

Here is the final model I used –

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 224, 224, 16)      160
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 66
=================================================================
Total params: 2,306.0
Trainable params: 2,306.0
Non-trainable params: 0.0
_________________________________________________________________
```

IV. Results

Model Evaluation and Validation

This is the final model I used –

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 224, 224, 16)      160
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 66
=================================================================
Total params: 2,306.0
Trainable params: 2,306.0
Non-trainable params: 0.0
_________________________________________________________________
```

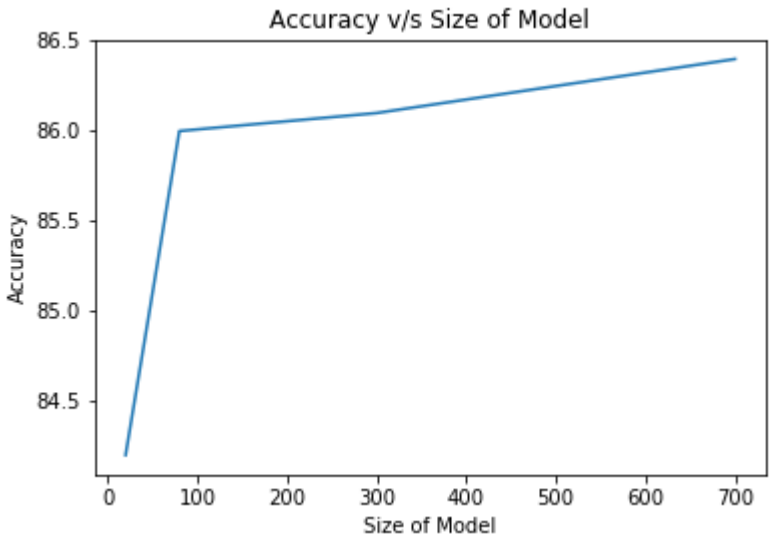
As we can see, this is a very light model, with 2 convolutional layers and one Global Average Pooling layer. And we are feeding in a grey scale image of 224x224. This model seemed to do the job pretty well, while not compromising on accuracy.

I tested out the model, by running the prediction script on the Raspberry Pi, directly on the machine. It was able to classify sticks quickly, in real time situation. Despite the time being very less between two consecutive sticks on the conveyor belt, it was able to figure out and classify them very fast. This is the goal I set initially, and I am glad to have achieved it.

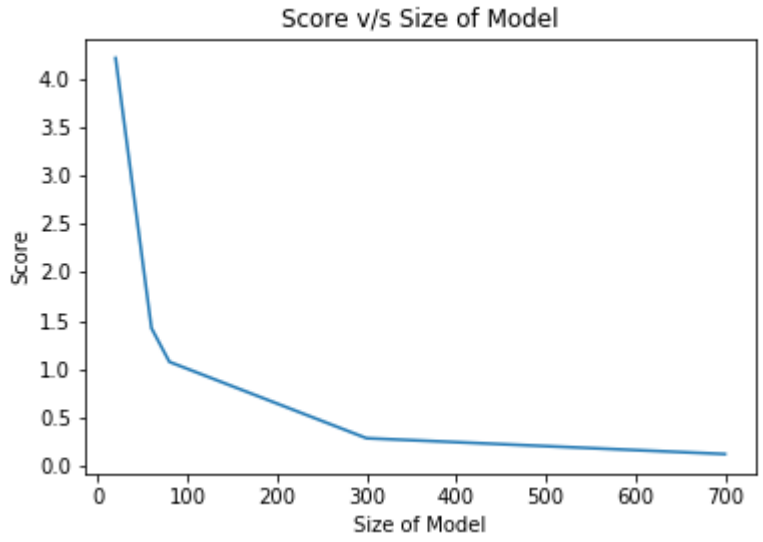
Justification

I set out to build a CNN model, which will classify the good sticks from the bad sticks, on the Incense stick machine. The main problem was, that the model should be able to predict quickly, since the interval between two sticks on the belt was very small. This forced us to reduce the size of the model while not compromising on quality. I think we have achieved this goal.

We were able to build a model, which not only gives an accuracy of 84%, but also performs really well real-time.

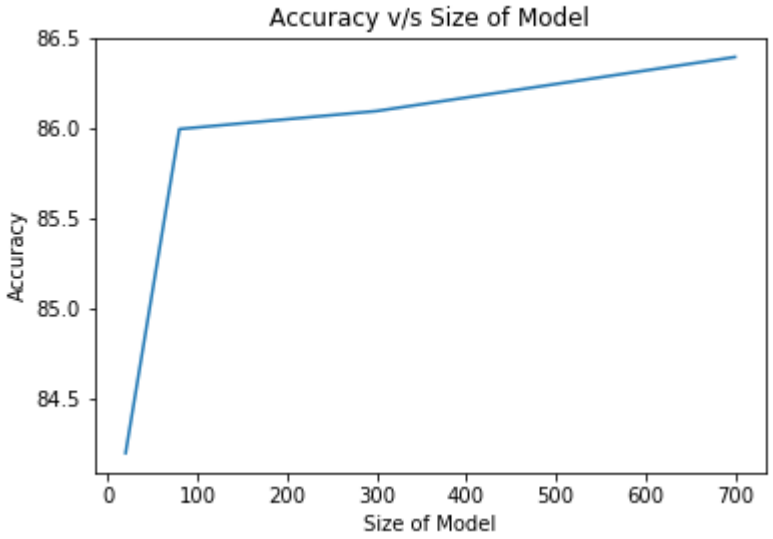


In the graph above, we can observe that despite the size of the model decreasing by a huge amount, from 700 to ~20, the accuracy has not dropped much. In the graph below, we can see how the score changed with different models, based on the size and accuracy. Based on these two factors, and how well it classified the sticks real-time, this is fantastic.



V. Conclusion

Free-Form Visualization



The most important part of this project was to build a classifier model, which would be able to run on the Raspberry Pi in real-time. This graph sums it up. I put

a lot of effort analysing each layer in the network, and reducing the number of parameters, so that the size of the model reduces. And we can see how the model went from 700MB to 20MB. This is the key part of the project.

Reflection

I set out to build an image processing module for every machine in my dad's incense stick industry. I started out by building a Convolutional Neural Network to classify the good sticks from the bad sticks and figuring out how to run it on the Raspberry Pi at real time. The key to the project, was building a lightweight model, which could compute and predict very quickly. This project was very interesting to me since I was solving a real world problem unlike working with some random dataset to just achieve better performance. It was challenging, and I had to get my hands dirty to figure out where the camera and raspberry pi will fit into the machine and how to actually integrate it.

It was challenging and frustrating, because my initial models were a failure, while running it on the Raspberry Pi as they took a lot of time. But, I somehow was able to get it down to the size, which was suitable for the Raspberry Pi. I have integrated the Raspberry Pi with one machine, and it is working. This project is very close to my heart as my dad was proud of me for getting this done.

Improvement

Since, we are limited by the processing capability of the Raspberry Pi, using a more powerful processor will help us have a much better model. This will improve the quality of the real-time classification done. Also, by having a faster camera to shoot photos, we can generate better datasets and ensure greater accuracy.