3D point cloud classification using Resnet50

Team No:32

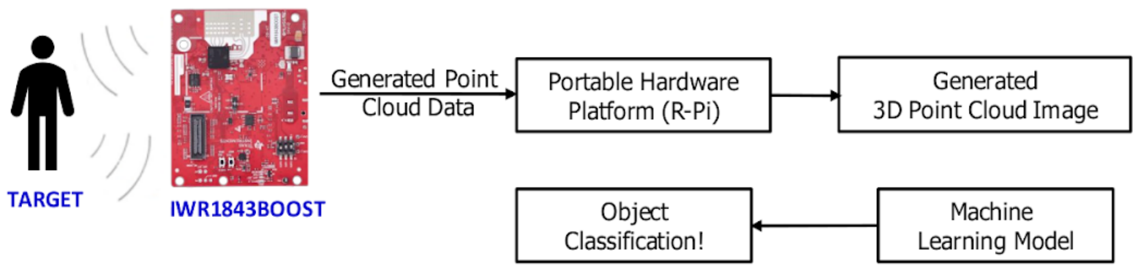
Team Name: Last Alpha

Problem statement:

The aim of the project is to create a Machine learning model using Resnet50 model which able to classify 3D point cloud data.

Block Diagram:

Proposed Solution



TARGET
IWR1843BOOST
Generated Point Cloud Data
Portable Hardware Platform (R-Pi)
Generated 3D Point Cloud Image
Machine Learning Model
Object Classification!

Flow for Object Detection & Classification

Motivation:

During the last years, there has been a rapid and successful expansion of computer vision research. Object detection is one of the key abilities required by most computer and robot vision systems.

Considering that this is the age of Artificial Intelligence, object detection is in high demand. The information from the object detector can be used for obstacle avoidance and other interactions with the environment. It is also useful for self-driving cars, helping decide what to do next; accelerate, apply brakes or turn, providing information about where all the objects are around the car and what those objects are. Object detection will reduce human efforts and provide efficiency. It is of interest as it may help humans to be aware of minute information about particular objects and reduce human tasks.

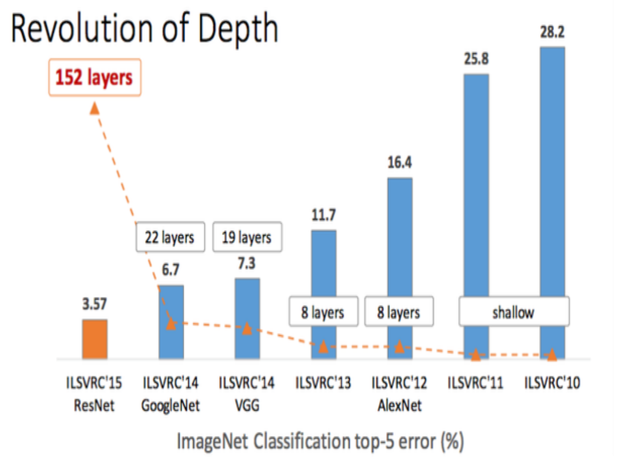
Overview:

A range of sensors, including lidar, radar, and depth cameras, collect point cloud data. These sensors record 3-D location data about the objects in a scene, which is helpful for various augmented reality and autonomous driving applications. For instance, differentiating between cars and people is essential for planning an autonomous vehicle's course. However, due to the sparsity of data per item, object occlusions, and sensor noise, building strong classifiers with point cloud data is difficult. By directly learning strong feature representations from point cloud data, deep learning approaches have been demonstrated to overcome several of these issues. One of the classifying techniques for point cloud is ResNet 50.

Methodology:

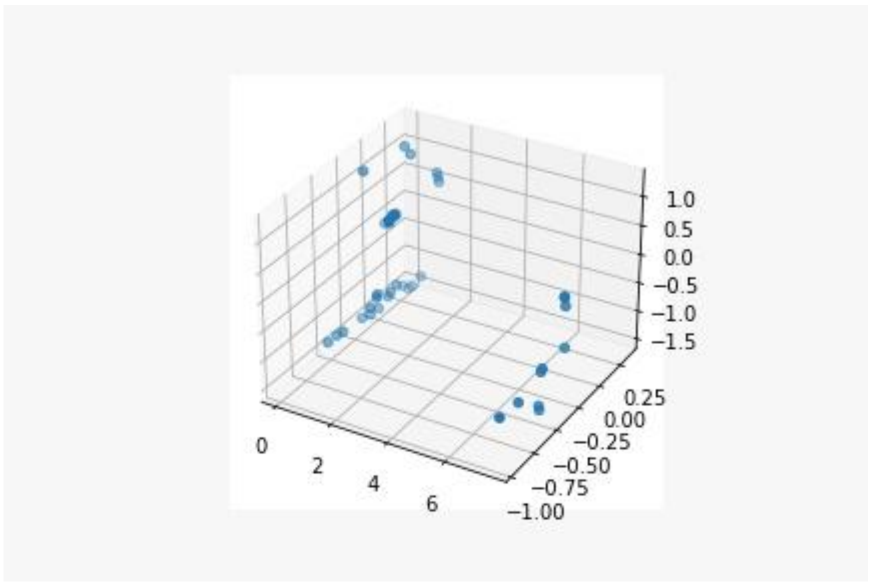
Why we chose ResNet50:

A convolutional neural network with 50 layers is called ResNet-50. The ImageNet database contains a pretrained version of the network that has been trained on more than a million pictures. The pretrained network can categorise photos into 1000 different item categories, including several animals, a keyboard, a mouse, and a pencil. The network has therefore acquired rich feature representations for a variety of pictures. The network accepts images with a resolution of 224 by 224.



Load Data set:

The given dataset contains 3D point cloud dataset for 7 classes namely- car, human car, van, truck, car bike, sedan, human



Downloading and extracting dataset to a directory

```
downloadDirectory = tempdir;
datapath = downloadSydneyUrbanObjects(downloadDirectory);
```

Using the loadSydneyUrbanObjectsData helper function to load the downloaded training and validation data set. Applied the fourth data fold for validation after using the previous three for training.

```
foldsTrain = 1:3;
foldsVal = 4;
dsTrain = loadSydneyUrbanObjectsData(datapath,foldsTrain);
dsVal = loadSydneyUrbanObjectsData(datapath,foldsVal);
```

For a better understanding of the distribution of labels throughout the data set, read the labels and count the points given to each one.

```
dsLabelCounts = transform(dsTrain,(data){data{2} data{1}.Count});
labelCounts = readall(dsLabelCounts);
labels = vertcat(labelCounts{:,1});
counts = vertcat(labelCounts{:,2});
```

utilising the utility function randReplicateFiles to randomly oversample the files to the appropriate number of observations per class, grouping the files by label, and counting the number of observations per class.

```
rng(0)
[G,classes] = findgroups(labels);
numObservations = splitapply(numel,labels,G);
desiredNumObservationsPerClass = max(numObservations);
files =
splitapply((x){randReplicateFiles(x,desiredNumObservationsPerClass)},dsTrain.Files,G
);
files = vertcat(files{:});
dsTrain.Files = files;
```

Data Augmentation:

The probability of overfitting the network increases when the files are duplicated in order to correct class imbalance since a large portion of the training data is the same. Use the transform and augmentPointCloud helper function, which randomly rotates the point cloud, randomly removes points, and randomly jitters points with Gaussian noise, to perform data augmentation to the training data in order to counteract this impact.

```
dsTrain = transform(dsTrain,augmentPointCloud);
```

Data pre-processing:

The point cloud data must go through two pre-processing procedures in order to be ready for training and prediction.

First, choose a predetermined number of points from each point cloud in order to facilitate batch processing during training. The ideal number of points will vary depending on the data collection and the quantity of points necessary to adequately represent the object's form. Calculate the lowest, maximum, and mean points per class to aid in choosing the right number of points.

```
minPointCount = splitapply(min,counts,G);
maxPointCount = splitapply(max,counts,G);
meanPointCount = splitapply((x)round(mean(x)),counts,G);
```

```
stats =
table(classes,numObservations,minPointCount,maxPointCount,meanPointCount)
```

Choosing a value that suits all classes is challenging due to the substantial intra-class and inter-class diversity in the number of points each class. One heuristic is to pick a sufficient number of points so that the geometry of the objects is accurately captured while avoiding processing too many points, which would increase the computing cost. An acceptable trade-off between these two factors is a value of 1024. The ideal number of points may also be chosen through empirical analysis. That, however, is outside the purview of this illustration. Choose 1024 points from the training and validation sets using the transform function.

```
numPoints = 1024;
dsTrain = transform(dsTrain,(data)selectPoints(data,numPoints));
dsVal = transform(dsVal,(data)selectPoints(data,numPoints));
```

To accommodate for significant variances in the range of data values, the point cloud data is normalised between 0 and 1 in the final pre-processing phase. For instance, items nearby the lidar sensor have lower readings than ones farther away. These variations may prevent the network from convergent throughout training. The point cloud data in the training and validation sets can be normalised by using a transform.

```
dsTrain = transform(dsTrain,@preprocessPointCloud);
dsVal = transform(dsVal,@preprocessPointCloud);
```

Defining Resnet Model:

Load the pretrained Resnet model.

```
resnet_model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,

input_shape=(180,180,3),

pooling='avg',classes=5,
weights='imagenet')
for layer in pretrained_model.layers:
layer.trainable=False
```

Training:

```
epochs=10
history=resnet_model.fit(
train_ds,
validation_data=val_ds,
epochs=epochs
)
```

Function for processing PointCloudData

In order to normalise the data between 0 and 1, the preprocessPointCloudData function collects the X, Y, and Z point data from the input data. The function returns the X, Y, and Z values normalised.

Data Augmentation Function

The augmentPointCloudData function randomly jitters the point location with Gaussian noise, drops 30% of the points, and rotates a point cloud about the z-axis.

Supporting functions:

function aggregateConfusionMetric

The aggregateConfusionMetric function uses the projected results (YPred) and the expected results (YTest) to progressively populate a confusion matrix.

```
function cmat = aggreateConfusionMetric(cmat,YTest,YPred)
YTest = gather(extractdata(YTest));
YPred = gather(extractdata(YPred));
[m,n] = size(cmat);
cmat = cmat + full(sparse(YTest,YPred,1,m,n));
```

Function to Extract Training Data

The dataset of point cloud images and label data are extracted using the extractTrainingData method.

Function prepareForPrediction

Applying a user-defined function on data from nested structure is done using the prepareForPrediction function. It is employed to transfer model state and parameter data to the GPU.

Output:

The following model able to predict the given 3D point cloud image as one of 7 classes.

Object Classification for Images

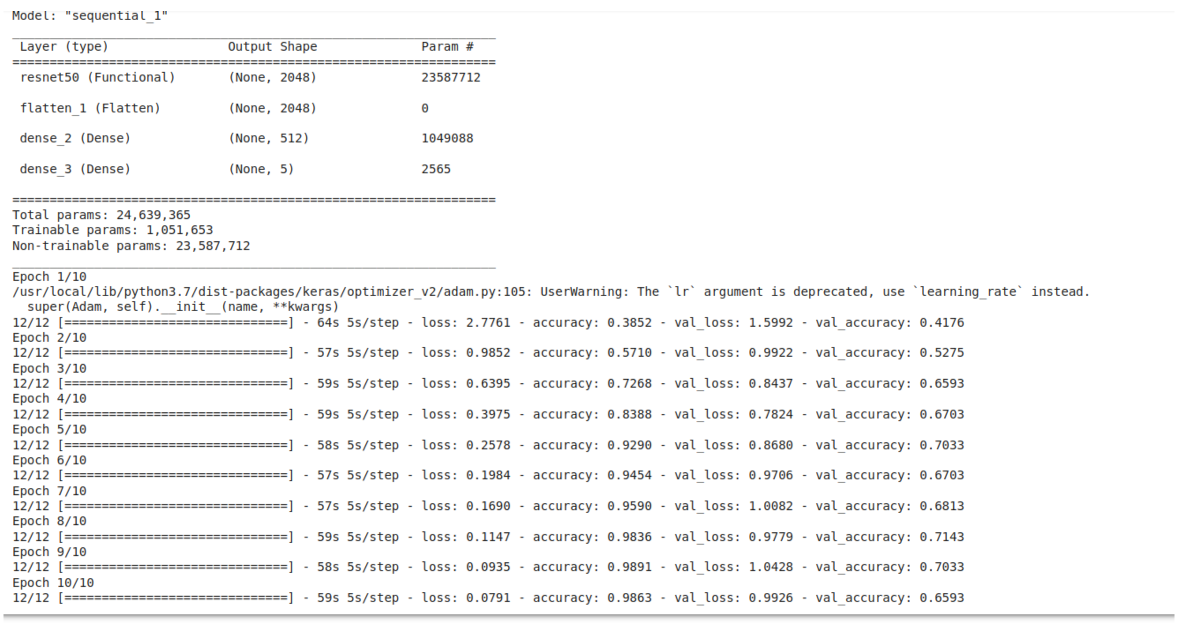
Data validation:

Our picture data was divided into training and validation sets. Our model will be trained on the training subset at the beginning of each epoch while being evaluated on the validation data at the end of each epoch.

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
data_dir,
validation_split=0.2,
subset="validation",
seed=123,
label_mode='categorical',
image_size=(img_height, img_width),
batch_size=batch_size)
```

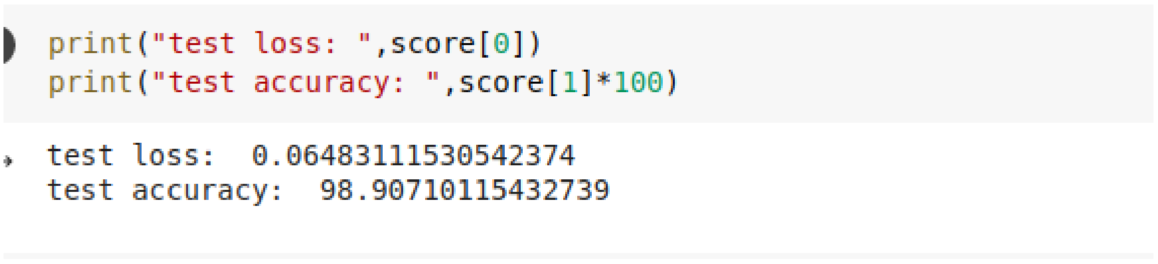
Training:

Below are the training accuracies of ResNet model



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 512)               1049088

 dense_3 (Dense)             (None, 5)                 2565

=================================================================
Total params: 24,639,365
Trainable params: 1,051,653
Non-trainable params: 23,587,712
_________________________________________________________________
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
12/12 [==============================] - 64s 5s/step - loss: 2.7761 - accuracy: 0.3852 - val_loss: 1.5992 - val_accuracy: 0.4176
Epoch 2/10
12/12 [==============================] - 57s 5s/step - loss: 0.9852 - accuracy: 0.5710 - val_loss: 0.9922 - val_accuracy: 0.5275
Epoch 3/10
12/12 [==============================] - 59s 5s/step - loss: 0.6395 - accuracy: 0.7268 - val_loss: 0.8437 - val_accuracy: 0.6593
Epoch 4/10
12/12 [==============================] - 59s 5s/step - loss: 0.3975 - accuracy: 0.8388 - val_loss: 0.7824 - val_accuracy: 0.6703
Epoch 5/10
12/12 [==============================] - 58s 5s/step - loss: 0.2578 - accuracy: 0.9290 - val_loss: 0.8680 - val_accuracy: 0.7033
Epoch 6/10
12/12 [==============================] - 57s 5s/step - loss: 0.1984 - accuracy: 0.9454 - val_loss: 0.9706 - val_accuracy: 0.6703
Epoch 7/10
12/12 [==============================] - 57s 5s/step - loss: 0.1690 - accuracy: 0.9590 - val_loss: 1.0082 - val_accuracy: 0.6813
Epoch 8/10
12/12 [==============================] - 59s 5s/step - loss: 0.1147 - accuracy: 0.9836 - val_loss: 0.9779 - val_accuracy: 0.7143
Epoch 9/10
12/12 [==============================] - 58s 5s/step - loss: 0.0935 - accuracy: 0.9891 - val_loss: 1.0428 - val_accuracy: 0.7033
Epoch 10/10
12/12 [==============================] - 59s 5s/step - loss: 0.0791 - accuracy: 0.9863 - val_loss: 0.9926 - val_accuracy: 0.6593
```

Accuracy of the Model is 98.907%

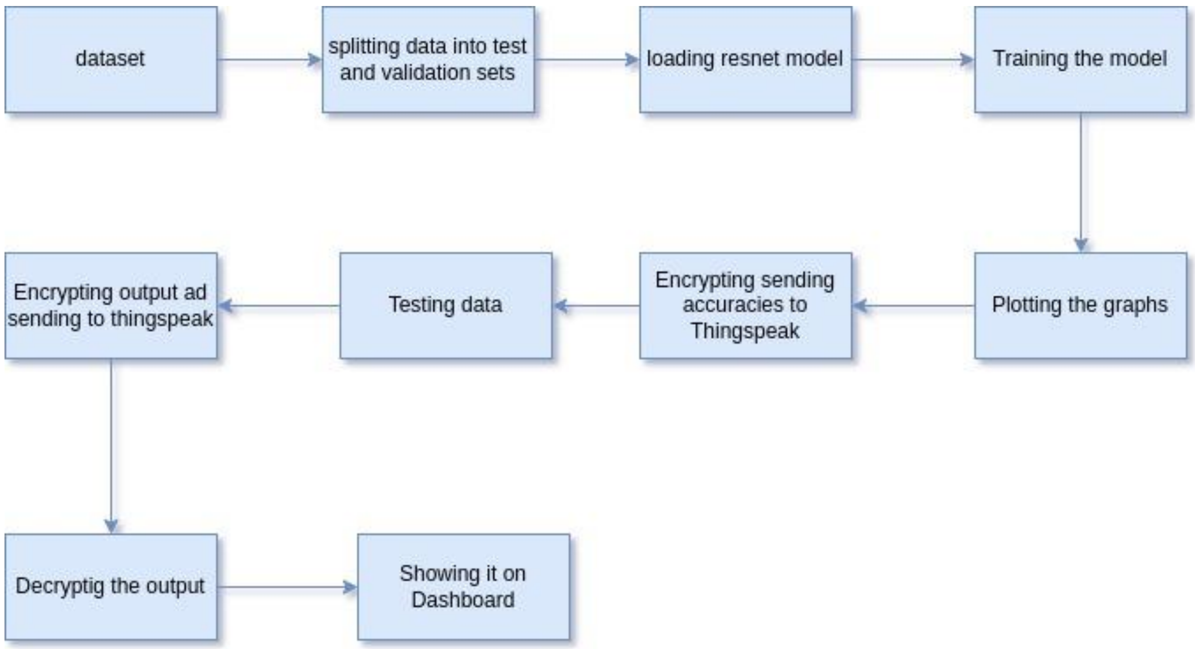


```
print("test loss: ",score[0])
print("test accuracy: ",score[1]*100)
```

```
test loss:  0.06483111530542374
test accuracy:  98.90710115432739
```

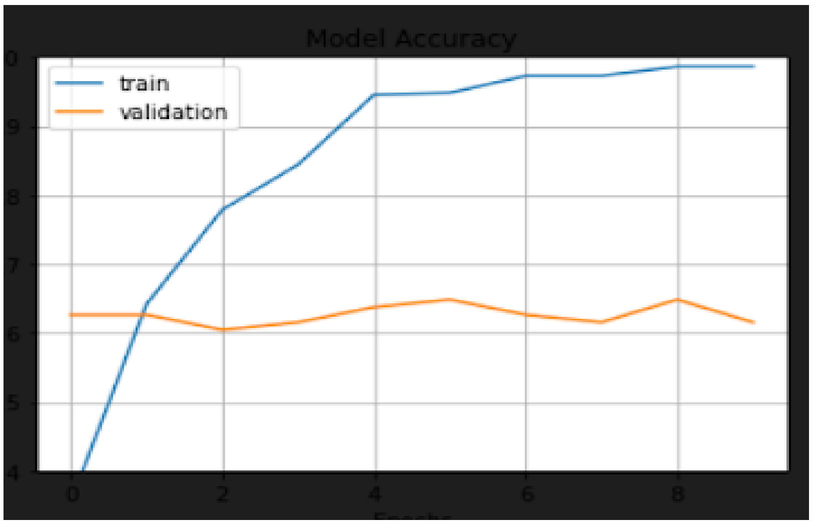
<u>Flowchart:</u>

Our working flow for the 2D model we did is given below.

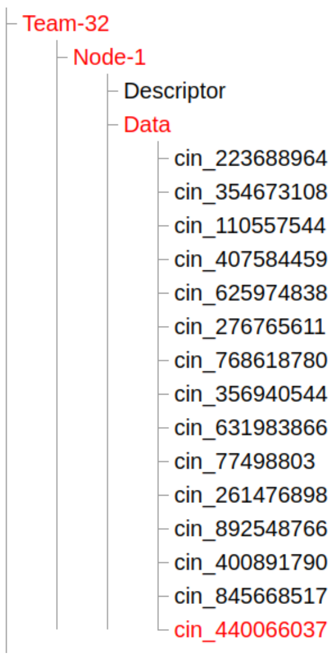

dataset
splitting data into test and validation sets
loading resnet model
Training the model
Plotting the graphs
Encrypting sending accuracies to Thingspeak
Testing data
Encrypting output ad sending to thingspeak
Decryptig the output
Showing it on Dashboard

<u>Communication to thing speak and oneM2M:</u>

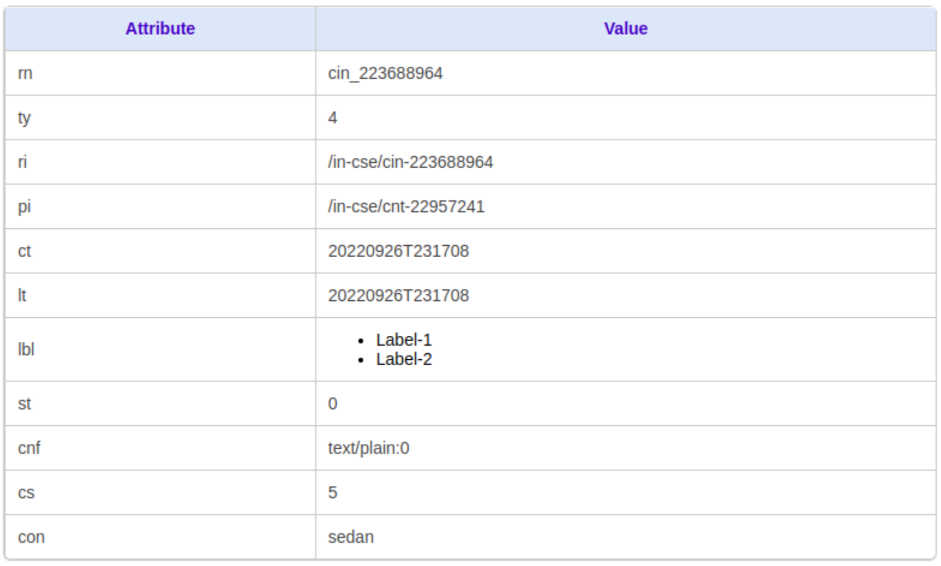
Thing speak:



OneM2M:



```
Team-32
  Node-1
    Descriptor
    Data
      cin_223688964
      cin_354673108
      cin_110557544
      cin_407584459
      cin_625974838
      cin_276765611
      cin_768618780
      cin_356940544
      cin_631983866
      cin_77498803
      cin_261476898
      cin_892548766
      cin_400891790
      cin_845668517
      cin_440066037
```



Attribute	Value
rn	cin_223688964
ty	4
ri	/in-cse/cin-223688964
pi	/in-cse/cnt-22957241
ct	20220926T231708
lt	20220926T231708
lbl	• Label-1 • Label-2
st	0
cnf	text/plain:0
cs	5
con	sedan

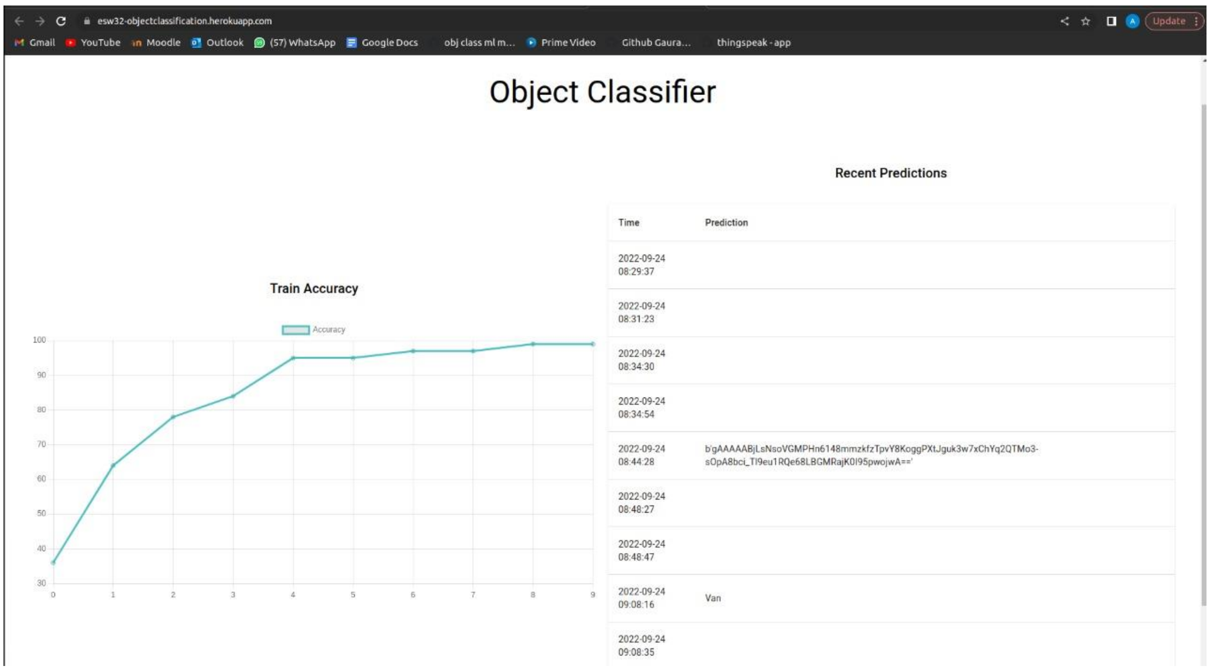
Here you can see the output(sedan) which is sent to onem2m .

Deployment of Dashboard:

The dashboard displays the testing accuracy of different classes.

The last 10 objects identified by the model can be accessed from the table.

Dashboard has been deployed using Heroku and was created using React Native.



Object Classifier

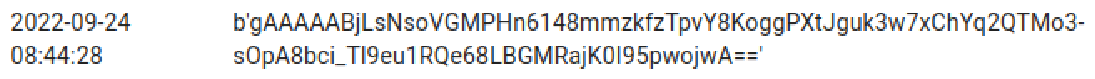
Train Accuracy

Recent Predictions

Time	Prediction
2022-09-24 08:29:37	
2022-09-24 08:31:23	
2022-09-24 08:34:30	
2022-09-24 08:34:54	
2022-09-24 08:44:28	b'gAAAAABjLsNsoVGMPHn6148mmzkfzTpvY8KoggPXtJguk3w7xChYq2QTMo3-sOpA8bci_Tl9eu1RQe68LBGMRajK0l95pwojwA=='
2022-09-24 08:48:27	
2022-09-24 08:48:47	
2022-09-24 09:08:16	Van
2022-09-24 09:08:35	

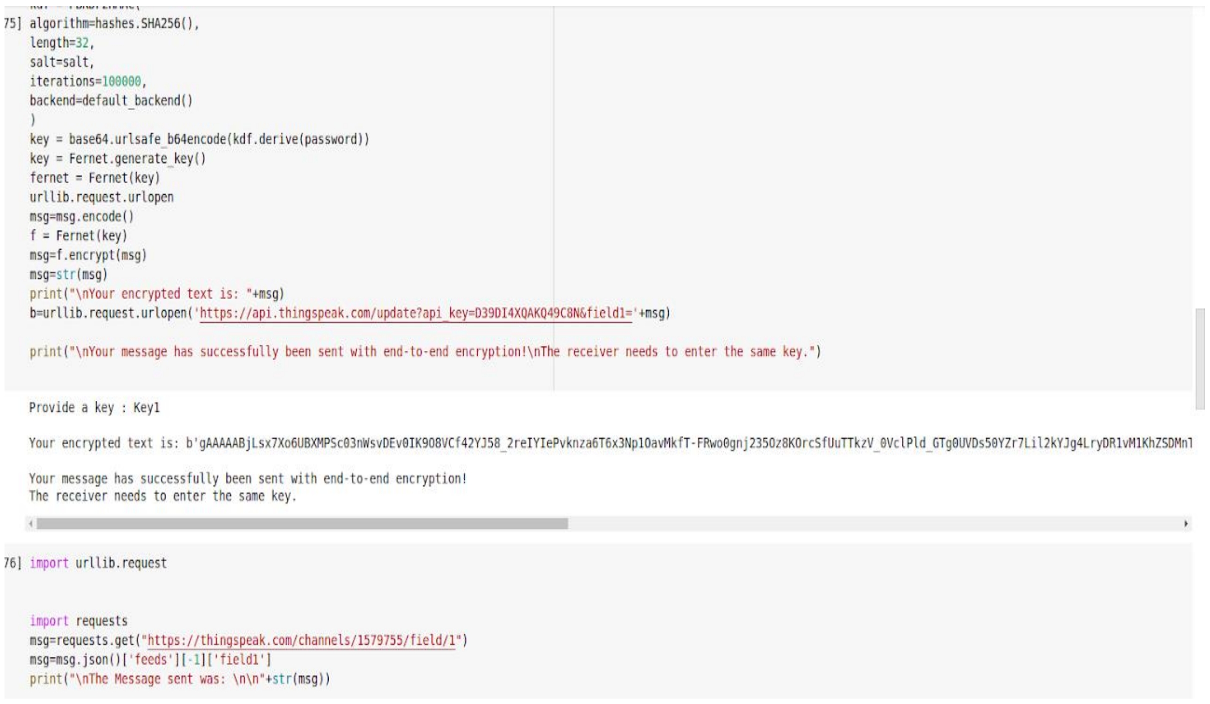
Use of IoT security:

Encryption is used by the model to encrypt the data and is sent to Thing speak. This is then decrypted in the end and stored in a container.



2022-09-24 08:44:28	b'gAAAAABjLsNsoVGMPHn6148mmzkfzTpvY8KoggPXtJguk3w7xChYq2QTMo3-sOpA8bci_Tl9eu1RQe68LBGMRajK0l95pwojwA=='

Encryption and Decryption:

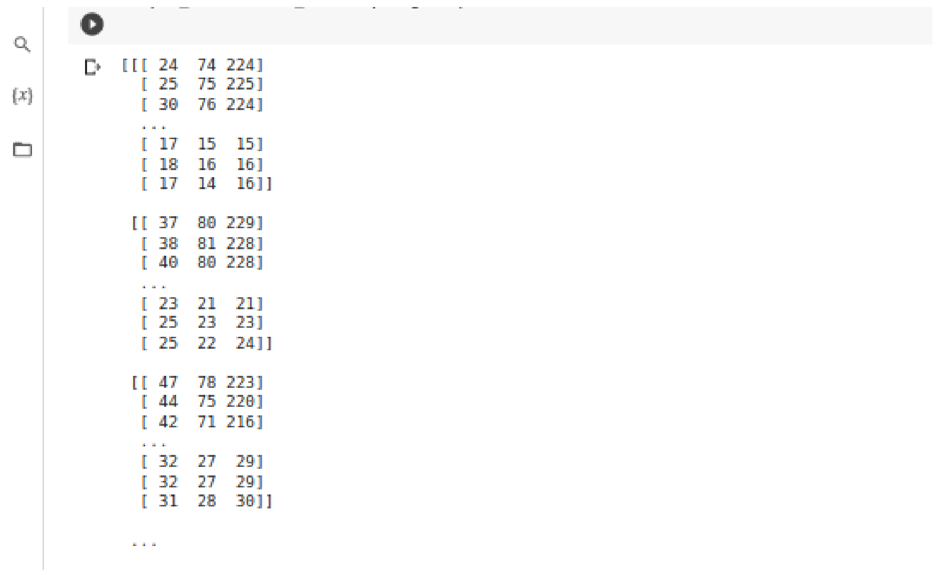


```
75] algorithm=hashes.SHA256(),
    length=32,
    salt=salt,
    iterations=100000,
    backend=default_backend()
    )
    key = base64.urlsafe_b64encode(kdf.derive(password))
    key = Fernet.generate_key()
    fernet = Fernet(key)
    urllib.request.urlopen
    msg=msg.encode()
    f = Fernet(key)
    msg=f.encrypt(msg)
    msg=str(msg)
    print("\nYour encrypted text is: "+msg)
    b=urllib.request.urlopen('https://api.thingspeak.com/update?api_key=D39DI4XQAKQ49C8N&field1='+msg)

    print("\nYour message has successfully been sent with end-to-end encryption!\nThe receiver needs to enter the same key.")

    Provide a key : Key1

    Your encrypted text is: b'gAAAAABjLsx7Xo6UBXMPSc03nWsvDEv0IK908VCf42YJ58_2reIYIePvknza6T6x3Np1OavMkfT-FRwo0gnj2350z8KOrcSfUuTTkzV_0VclPld_GTg0UVDs50YZr7Lil2kYJg4LryDR1vM1KhZSDMn1

    Your message has successfully been sent with end-to-end encryption!
    The receiver needs to enter the same key.
```

```
76] import urllib.request


    import requests
    msg=requests.get("https://thingspeak.com/channels/1579755/field/1")
    msg=msg.json()['feeds'][-1]['field1']
    print("\nThe Message sent was: \n\n"+str(msg))
```

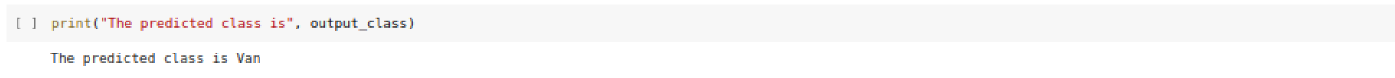
Data Visualisation and analysis:

The image data after converting



```
[[[ 24  74 224]
  [ 25  75 225]
  [ 30  76 224]
  ...
  [ 17  15  15]
  [ 18  16  16]
  [ 17  14  16]]

 [[ 37  80 229]
  [ 38  81 228]
  [ 40  80 228]
  ...
  [ 23  21  21]
  [ 25  23  23]
  [ 25  22  24]]

 [[ 47  78 223]
  [ 44  75 220]
  [ 42  71 216]
  ...
  [ 32  27  29]
  [ 32  27  29]
  [ 31  28  30]]

 ...
```

Final Result



```
[ ] print("The predicted class is", output_class)

    The predicted class is Van
```

Here you can see the result which is predicted by the model as van.

Conclusion:

The Machine learning model which is trained using ResNet50 can predict the given image with 98.9% accuracy.

Git link: https://github.com/AmeyaSharma/ObjectClassification