**Report**

**Q.1 Problem statement :**

Implement a suitable classical machine learning algorithm to detect characters in these signs and display detected text finally.

Approach:

The provided sample data have too much of noise, irregular shape, and undistinguishable characters within each image.

Proposed pipeline:

1. Train a ML model for classifying individual letters
2. Read image -> Segment out every letter from an image -> Provide this letter to Trained ML (1) model for class detection -> get predicted class of letter -> Update .txt file for each image

To solve this problem, I preferred image processing approach. In this approach I used various combinations of techniques which are listed below.

1. Color conversion: RGB to Gray, RGB to HSV, RGB to LAB
2. Thresholding: Global & Adaptive
3. Filtering: Gaussian, Laplace, Custom
4. Edge detection: Canny
5. Contour detection

Various combinations and experiments were conducted to extract every letter from the input image, but the optimal combination was not achieved.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

To create a test dataset for ML model performance analysis, manually some letters from the provided images were cropped and stored in ‘.png’ format.



**ML model training:**

For classifying identified letter images w.r.t. each representative letter, ML approach was used. Platform used for training the ML model was Kaggle.

Dataset used: <https://www.kaggle.com/datasets/preatcher/standard-ocr-dataset>

Approach:

According to pipeline, user will provide image to ‘Image processing module’ which will return individual letters from the given image. Those individual letters will be provided to trained ML model for class prediction, and the resultant class will be updated in .txt file.

Libraries used: sklearn, numpy, os, opencv, matplotlib

Dataset: The dataset is itself divided into two subfolders ‘data/training\_data’ & ‘data/testing\_data’

Each folder has subfolders of characters: 0-9 & A-Z; total of 36

Preprocess:

Images are imported in test & train variables, individual class distribution is checked; it was found to be equal.

Images were normalized so that processing became faster. The image classes were encoded in the numeric class format.

In the training and analysis,

SVM, KNN and Decision Tree models were trained and experimented with.

Model performance was checked using parameters: Accuracy & Precision

In the performance, every model reported accuracy of 97+ % in each class. But for testing on sample data which was manually used, it did not perform that well.

**Q.2 Problem statement :**

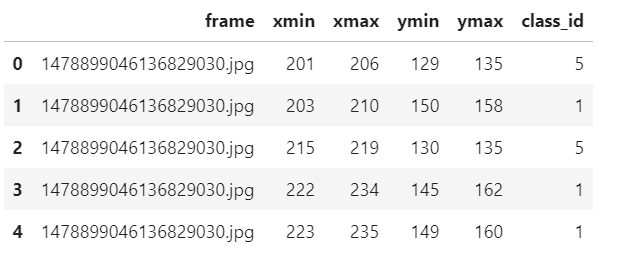
Problem statement :

Design detection, classification and tracking DL framework for Pedestrian, Cyclist, Motorcyclist, Standard Vehicles, Non-Standard Vehicles etc.

Dataset: <https://www.kaggle.com/datasets/alincijov/self-driving-cars>

Dataset is having a folder name ‘images’ and three .csv files ‘labels\_train’, ‘labels\_trainval’, ‘labels\_val’

In the label files,



‘frame’ column indicates image name and class\_id represents label for the bounding box.

EDA:

To check whether the given bounding box coordinates match the respective image, first we load random index from label list and corresponding to that index, image and its bounding box values are plotted for visualization. This process shows single bbox in an image.

To check whether each image has multiple bbox labels within and to find out corresponding label for ‘class\_id’, a visualization cell was developed. This cell randomly identifies a image id, as we get image id, it is cross referenced with all the labels present in the file, wherever image id matches, those bounding boxes are appended in an array. In the last, image is visualized with all the bounding boxes with different colors as per labels. This helped to cross-check and verify which labels belong to what class.

Class\_id Label

1. car
2. truck
3. pedestrian
4. bicyclist
5. light

Data-generator: If we load out dataset as whole into the architecture for training the model, it affects the memory and RAM of the system, there is chance of error (this happed while during initial experiments). To avoid this, data generators are introduced for train and validation data. These data generators load data in batches, normalize images, provide in array format to the DL architecture.

Architecture:

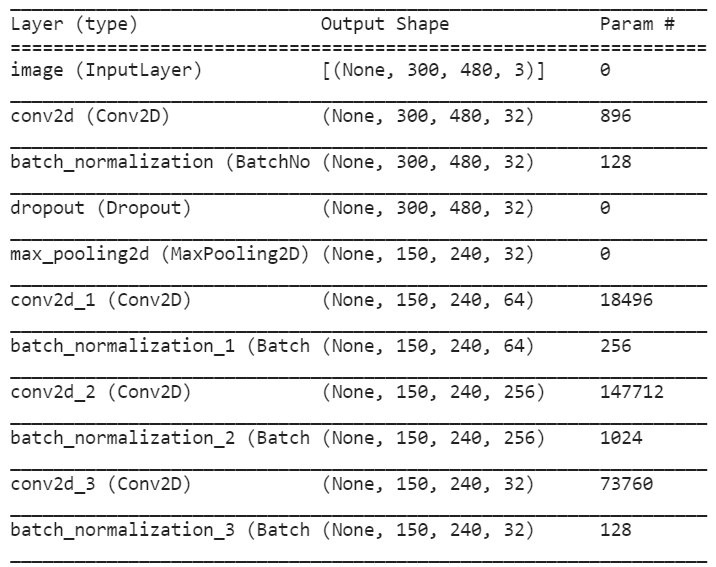
All the images were of size (300,480), the input to the architecture was given with same dimension.

The architecture involves Convolution, Batch-normalization, Dropout, Pooling layers.

The architecture is designed in a way such that it can be interpreted as blocks.

In a block, at 1st layer we use 32 filters, for 2nd layer 64 and 3rd layer 256. After each layer, batch-normalization is performed. Addition to this, after 1st layer there is drop-out and pooling layer introduced. As this block execution is completed, a drop-out and pooling layer is introduced when block number is even (2,4,6,..). This whole block is in the for loop for execution of 5 times.

After this convolution block, flatten layer is introduced. The activation used is ‘relu’, the padding in case of convolution is same.



A picture containing text, receipt, screenshot, document

Description automatically generated

Table

Description automatically generated

Table

Description automatically generatedTable

Description automatically generated

The loss used is ‘MSE’ along with Adam optimizer with learning rate 0.0001.

In the initial observation, loss divergence is starting from a very high point, and diverging slowly. The training accuracy has increased from 10 to 80% in 1st epoch. The model training is facing challenges, after 1st epoch the execution continues to run but does not updates on the execution platform.

REFERENCES

<https://www.tensorflow.org/api_docs/python/tf/keras/Model>

https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/

http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/#download