International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 04 Issue: 11 | Nov-2020 ISSN: 2582-3930

DETECTION AND RECOGNITION OF NUMBER PLATES ON VEHICLES USING MACHINE LEARNING

Er. Amaan, Department of Computer Science and Engineering, Punjabi University Patiala

Er. Sikander Singh Cheema, Assistant Professor, Department of Computer Science and Engineering, Punjabi University Patiala

---***---

Abstract- License Plate Detection and number recognition is a very powerful and advanced application that needs Machine learning, Deep Learning algorithms integrated with the Image processing . This paper deals with both number plate detection and recognition using Python, Numpy, Deep Learning and Machine Learning algorithms. The work is divided into three different parts, first, the number plate is detected using OpenCV, Numpy and Keras libraries. Secondly, we performed character segmentation of license plates using python and machine learning algorithms. Finally, after parsing the characters in the previous section, we have performed character recognition of license plates using deep learning algorithms.

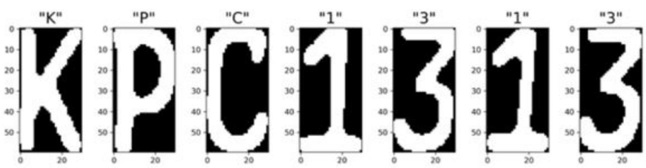
Keywords: OpenCV2, ALPR, Keras, Numpy, CNN, ML, DNN

1.INTRODUCTION

These days there is constant advancement in Machine Learning, particularly with Neural Networks and Open-source Machine Learning libraries such as Keras, Pytorch and so on. There are several other ways to deploy such technologies, however, this paper defines the setup rules for ALPR systems in association with previously trained model using Wpod-Net and Computer Vision along with OpenCV and Character Recognition using Neural Networks. The research has been classified into 3 broad categories. In the first stage, the data has been taken from different countries to implement the pre-trained data model for license plate recognition system using Wpod-Net.

And, in second part, characters of license plate are segmented with OpenCV.

Lastly, Neural Network is trained to predict the character generated in the previous section.



1.1. Library and Tools Used during the experiment

- Python 3.7
- Keras 2.3.1
- Tensorflow 1.14.0
- Jupyter Notebook
- Numpy 1.17.4
- Matplotlib 3.2.1

- OpenCV 4.1.0

2. Detection of License Plate using Wpod-Net

After installing the all the required libraries and loading the pre-trained model, description of packages and functions have been discussed as follows:

- **CV2**: CV2 or Computer Vision library, which is also known as OpenCV. It is used to perform image processing techniques.
- **numpy**: It is a library that supports multi-dimensional arrays and matrix operations.
- **matplotlib**: This library supports plotting and visualize the data.
- **local_utils**: This python script contains some functions which is used to process the data from Wpod-Net.
- **os.path / glob:** Operating system interface package/library for python programming. We will for directories and file system handling.
- **keras.models**: This package contains *model_from_json* to load the model data architecture in JSON format.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from local_utils import detect_lp
from os.path import splitext, basename
from keras.models import model_from_json
import glob
%matplotlib.inline
```

Next, the data model is loaded from the pre-trained model.

```python
def load_model(path):
    try:
        path = splitext(path)[0]
        with open('%s.json' % path, 'r') as json_file:
            model_json = json_file.read()
        model = model_from_json(model_json, custom_objects={})
        model.load_weights('%s.h5' % path)
        print("Loading model successfully...")
        return model
    except Exception as e:
        print(e)

wpod_net_path = "wpod-net.json"
wpod_net = load_model(wpod_net_path)

```

Similarly, preprocess_image function needs to be created in order to read and process the images of license plate. By using this function, we can parse the image and then convert into RGB format. After this, the data needs to be normalized in the range between 0-1. This will make the data compatible with matplotlib. Apart from this, we will set resize=true because we need to resize all the images for similar dimension i.e. width is 224 and height is 224 as well.

```python
def preprocess_image(image_path,resize=False):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img / 255
    if resize:
        img = cv2.resize(img, (224,224))
    return img
```

In this, we will see the dataset of vehicle. The dataset has images of 20 different vehicles along with plate data that is gathered from different countries. The data is collected from Germany, Japan, Vietnam, Thailand, Saudi Arabia, Russia, United States of America, Korea, India and China. In the following code, the information is displayed in 5 column and 4 rows.

```python
# Create a list of image paths
image_paths = glob.glob("Plate_examples/*.jpg")
print("Found %i images..."%(len(image_paths)))

# Visualize data in subplot
fig = plt.figure(figsize=(12,8))
cols = 5
rows = 4
fig_list = []
for i in range(cols*rows):
    fig_list.append(fig.add_subplot(rows,cols,i+1))
    title = splitext(basename(image_paths[i]))[0]
    fig_list[-1].set_title(title)
    img = preprocess_image(image_paths[i],True)
    plt.axis(False)
    plt.imshow(img)

plt.tight_layout(True)
plt.show()
```

Now, we will use get_plate function, which will process the raw data. After this, the data is sent to the model and return the plate image (LpImg) along with co-ordinates. In case of null data, warning is shown. Therefore, we need to change the Dimension values i.e. Dmin to increase the overall dimension. The Wpod-Net only parse the data having character in black color with white background. Therefore, there might be failure in the prediction if the image is not clear or there is any kind of obstacle.

```
def get_plate(image_path, Dmax=608, Dmin=256):
    vehicle = preprocess_image(image_path)
    ratio = float(max(vehicle.shape[:2])) / min(vehicle.shape[:2])
    side = int(ratio * Dmin)
    bound_dim = min(side, Dmax)
    _ , LpImg, _, cor = detect_lp(wpod_net, vehicle, bound_dim, lp_threshold=0.5)
    return LpImg, cor

# Obtain plate image and its coordinates from an image
test_image = image_paths[0]
LpImg,cor = get_plate(test_image)
print("Detect %i plate(s) in"%len(LpImg),splitext(basename(test_image))[0])
print("Coordinate of plate(s) in image: \n", cor)

# Visualize our result
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.axis(False)
plt.imshow(preprocess_image(test_image))
plt.subplot(1,2,2)
plt.axis(False)
plt.imshow(LpImg[0])
```

```
Detect 1 plate(s) in china_car_plate
Coordinate of plate(s) in image:
 [array([[253.34864037, 508.28028195, 497.12277873, 242.19113715],
       [471.69134945, 526.6670509 , 635.22632653, 580.25062507],
       [  1.        ,   1.        ,   1.        ,   1.        ]])]
```

Next, we will draw bounding box using co-ordinates generated previously as shown in the Figure.

```
def draw_box(image_path, cor, thickness=3):
    pts=[]
    x_coordinates=cor[0][0]
    y_coordinates=cor[0][1]
    # store the top-left, top-right, bottom-left, bottom-right
    # of the plate license respectively
    for i in range(4):
        pts.append([int(x_coordinates[i]),int(y_coordinates[i])])

    pts = np.array(pts, np.int32)
    pts = pts.reshape((-1,1,2))
    vehicle_image = preprocess_image(image_path)

    cv2.polylines(vehicle_image,[pts],True,(0,255,0),thickness)
    return vehicle_image

plt.figure(figsize=(8,8))
plt.axis(False)
plt.imshow(draw_box(test_image,cor))
```

At last, we will use get_plate function for all vehicles images and then return the plate images to the model.

```
# Viualize all obtained plate images
fig = plt.figure(figsize=(12,6))
cols = 5
rows = 4
fig_list = []

for i in range(cols*rows):
    fig_list.append(fig.add_subplot(rows,cols,i+1))
    title = splitext(basename(image_paths[i]))[0]
    fig_list[-1].set_title(title)
    LpImg,_ = get_plate(image_paths[i])
    plt.axis(False)
    plt.imshow(LpImg[0])

plt.tight_layout(True)
plt.show()
```

3. Detect and Recognize Vehicle's License Plate with ML and Python: Plate character segmentation with OpenCV

In this part, license plate recognition system is discussed with different magnitude of colors. We will segregate characters out of License Plate through Python and OpenCV. After parsing the data, the data will be parsed again using Convolutional Neural Network (CNN) in part 3. For parsing the data, we will require python 3.7, Jupyter Notebook, Matplotlib 3.2.1, Numpy and OpenCV 4.1.

3.1. Processing of Image

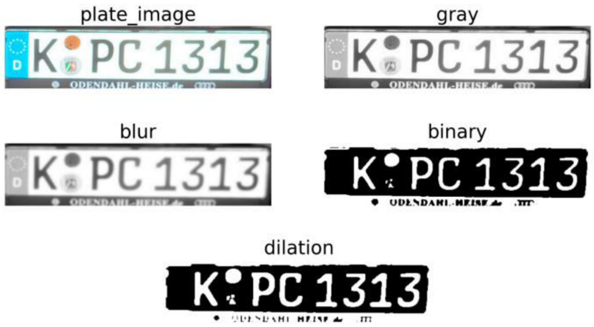
For this process, first we are required to implement different processing technique to mitigate the noise level and enhance the features of character recognition. For experiment, we will consider the image from Plates_example/germany_car_plate.jpg

- **Convert the image to 255 Scale:** The extracted image of license from Wpod-Net is translated as 0-1 range. Hence, we are required to convert this into 8-bit scale.
- **Convert into grayscale:** we need to remove the colors from the license plate. By doing so, we can increase the efficacy of the system.
- **Image Blur:** In order to reduce the noise and other unrequired data, image blurring technique is taken into account. We will use Gaussian Blur method for this.
- **Image Thresholding:** In this, we set a minimum threshold value for computation of image. Below the threshold value, the data is converted into 255. This technique is also called inverse binary thresholding.
- **Dilation:** By using this methodology, we can raise the white region area of the image.

```
if (len(LpImg)): #check if there is at least one license image
    # Scales, calculates absolute values, and converts the result to 8-bit.
    plate_image = cv2.convertScaleAbs(LpImg[0], alpha=(255.0))

    # convert to grayscale and blur the image
    gray = cv2.cvtColor(plate_image, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(7,7),0)

    # Applied inversed thresh_binary
    binary = cv2.threshold(blur, 180, 255,
                         cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
    ## Applied dilation
    kernel3 = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    thre_mor = cv2.morphologyEx(binary, cv2.MORPH_DILATE, kernel3)
```



3.2. Determining the contour of License Plate characters

Further to this, we will implement findContours function of OpenCV technique to collect the co-ordinates of characters. Contours is a collection of continuous points having same intensity and color. The sort_contours sorts the contours from left to right position. It is imperative for ordering the sequence of character. The ratio is equivalent to the height divided by contour width. Besides this, we can filter out the irrelevant information of image. So, we will set the ratio value between 1 to 3.5. Since we already know the minimum height of the image, we can use additional filters for better performance. Next to this, we will be drawing bounding box having the contour passes through the filters. Also, we will apply binary thresholding techniques and attach them to crop_characters.

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 04 Issue: 11 | Nov-2020 ISSN: 2582-3930

```python
# Create sort_contours() function to grab the contour of each digit from left to right
def sort_contours(cnts,reverse = False):
    i = 0
    boundingBoxes = [cv2.boundingRect(c) for c in cnts]
    (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes),
                                        key=lambda b: b[1][i], reverse=reverse))
    return cnts

cont, _  = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# creat a copy version "test_roi" of plat_image to draw bounding box
test_roi = plate_image.copy()

# Initialize a list which will be used to append charater image
crop_characters = []

# define standard width and height of character
digit_w, digit_h = 30, 60

for c in sort_contours(cont):
    (x, y, w, h) = cv2.boundingRect(c)
    ratio = h/w
    if 1<=ratio<=3.5: # Only select contour with defined ratio
        if h/plate_image.shape[0]>=0.5: # Select contour which has the height larger than 50% of th
            # Draw bounding box arroung digit number
            cv2.rectangle(test_roi, (x, y), (x + w, y + h), (0, 255,0), 2)

            # Sperate number and gibe prediction
            curr_num = thre_mor[y:y+h,x:x+w]
            curr_num = cv2.resize(curr_num, dsize=(digit_w, digit_h))
            _, curr_num = cv2.threshold(curr_num, 220, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
            crop_characters.append(curr_num)

print("Detect {} letters...".format(len(crop_characters)))
```

3.3. Visualizing the segmented characters

The crop_characters now have the segemented character value. We can see this using a library matplotlib.

```python
fig = plt.figure(figsize=(14,4))
grid = gridspec.GridSpec(ncols=len(crop_characters),nrows=1,figure=fig)

for i in range(len(crop_characters)):
    fig.add_subplot(grid[i])
    plt.axis(False)
    plt.imshow(crop_characters[i],cmap="gray")
```

4. Detect and Recognize Vehicle's License Plate with ML and Python: Recognize plate license characters with OpenCV and Deep Learning

After parsing the characters in 2nd and 3rd section of this paper, we extracted the License plate information which was easy to perceive. However, it contains black and white characters which is not optimal for digital optimization. Therefore, in this portion, we will parse the data with Neural Network model. There are plethora of eminent Neural Network architectures are available readily such as ResNet, Inception, DenseNet and many more. For this experiment, we will use MobileNets. It is light in weight with higher accuracy.

For computation, we will use python 3.7, Jupyter Notebook, OpenCV, Numpy, Keras, sklearn and Matplotlib. The model dataset contains 34,575 different images, which are further classified into 36 classes.

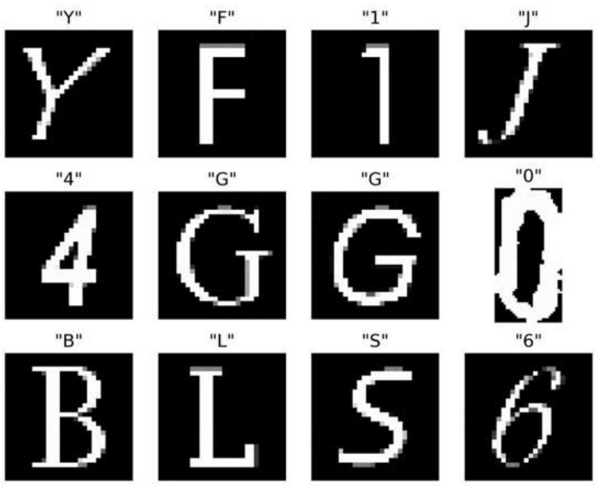
```python
import glob
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np

dataset_paths = glob.glob("dataset_characters/**/*.jpg")

cols=4
rows=3
fig = plt.figure(figsize=(10,8))
plt.rcParams.update({"font.size":14})
grid = gridspec.GridSpec(ncols=cols,nrows=rows,figure=fig)

# create a random list of images will be displayed
np.random.seed(45)
rand = np.random.randint(0,len(dataset_paths),size=(cols*rows))

# Plot example images
for i in range(cols*rows):
    fig.add_subplot(grid[i])
    image = load_img(dataset_paths[rand[i]])
    label = dataset_paths[rand[i]].split(os.path.sep)[-2]
    plt.title('"{:s}"'.format(label))
    plt.axis(False)
    plt.imshow(image)
```



4.1. Pre-processing of Data

Now, we are required to process the data:

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 04 Issue: 11 | Nov-2020 ISSN: 2582-3930

- **Line 2 ~ 14:** Input data is arranged along with their corresponding labels. The size of the original image 224* 224 for MobileNets. But we were required to change the dimensions to 80*80. With such configuration setting, the accuracy improves significantly to 98%.

- **Line 20 ~ 26**: we will consider converting the labels as first-dimensional array to one-hot encoding labels. This enables the representation of labels in more optimal way. The classes of labels are stored locally to improve performance and inverse transformation.

- **Line 29**: Split the dataset into two parts i.e. training set (90%) and validation set (10%). It allows us to monitor accuracy and avoiding overfitting.

- **Line 33**: Using basic transforming techniques, we create data augmenting such as shifting, rotating, zooming and so on. This technique needs to be handled with utmost precision otherwise data may be manipulated.

```
1   # Arange input data and corresponding labels
2   X=[]
3   labels=[]
4
5   for image_path in dataset_paths:
6     label = image_path.split(os.path.sep)[-2]
7     image=load_img(image_path,target_size=(80,80))
8     image=img_to_array(image)
9
10    X.append(image)
11    labels.append(label)
12
13  X = np.array(X,dtype="float16")
14  labels = np.array(labels)
15
16  print("[INFO] Find {:d} images with {:d} classes".format(len(X),len(set(labels))))
17
18  # perform one-hot encoding on the labels
19  lb = LabelEncoder()
20  lb.fit(labels)
21  labels = lb.transform(labels)
22  y = to_categorical(labels)
23
24  # save label file so we can use in another script
25  np.save('license_character_classes.npy', lb.classes_)
26
27  # split 10% of data as validation set
28  (trainX, testX, trainY, testY) = train_test_split(X, y, test_size=0.10, stratify=y, random_state=42)
29
30
31  # generate data augumentation method
32  image_gen = ImageDataGenerator(rotation_range=10,
33                                 width_shift_range=0.1,
34                                 height_shift_range=0.1,
35                                 shear_range=0.1,
36                                 zoom_range=0.1,
37                                 fill_mode="nearest"
38                                 )
```

4.3. Initializing the MobileNet architecture along with pre-trained weights

In this section, the MobileNets architecture is being constructed in association with pre-trained datasets. We can import the data directly from Keras packages.

- **Line 4~5:** Output layers are being ignored in MobileNets model, and we replace those layers with our output according to our needs. In the output layer, there are 36 nodes which are associated with 36 different characters. Also, we need to configure the input layer.

- ***Line 16 ~ 23*****:** Once thetraining datasets are set to true, base_model needs to be defined in each layer along with decay values, learning rate, metrices and precision.

```python
# Create our model with pre-trained MobileNetV2 architecture from imagenet
def create_model(lr=1e-4,decay=1e-4/25, training=False,output_shape=y.shape[1]):
    baseModel = MobileNetV2(weights="imagenet",
                            include_top=False,
                            input_tensor=Input(shape=(80, 80, 3)))

    headModel = baseModel.output
    headModel = AveragePooling2D(pool_size=(3, 3))(headModel)
    headModel = Flatten(name="flatten")(headModel)
    headModel = Dense(128, activation="relu")(headModel)
    headModel = Dropout(0.5)(headModel)
    headModel = Dense(output_shape, activation="softmax")(headModel)

    model = Model(inputs=baseModel.input, outputs=headModel)

    if training:
        # define trainable lalyer
        for layer in baseModel.layers:
            layer.trainable = True

        # compile model
        optimizer = Adam(lr=lr, decay = decay)
        model.compile(loss="categorical_crossentropy", optimizer=optimizer,metrics=["accuracy"])

    return model

# initilaize initial hyperparameter
INIT_LR = 1e-4
EPOCHS = 30

model = create_model(lr=INIT_LR, decay=INIT_LR/EPOCHS,training=True)
```

4.4. Training and evaluation of model

The BATCH_SIZE value can be adjusted according the specification of the system. The larger is the BATCH_SIZE, the fewer samples are trained. Thus, the overall performance of the model declines sharply. However, the computation power increases with the increase in BATCH_SIZE.

There are callback functions, which are programmed to better utilization of the resources. EarlyStopping function can halt the training process if the val_loss fails after 5 epochs. The ModelCheckpoint saves the model weight. It is analyzed that after 5 epochs, the model successfully attained more than 95% of accuracy rate. By the completion, 99% of accuracy has been achieved,

```python
BATCH_SIZE = 64

my_checkpointer = [
                EarlyStopping(monitor='val_loss', patience=5, verbose=0),
                ModelCheckpoint(filepath="License_character_recognition.h5", verbose=1, save_weights_only=True)
                ]

result = model.fit(image_gen.flow(trainX, trainY, batch_size=BATCH_SIZE),
                   steps_per_epoch=len(trainX) // BATCH_SIZE,
                   validation_data=(testX, testY),
                   validation_steps=len(testX) // BATCH_SIZE,
                   epochs=EPOCHS, callbacks=my_checkpointer)
```

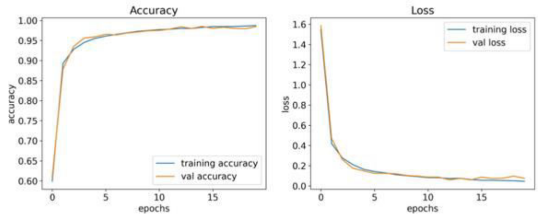
```
Epoch 1/30
486/486 [==============================] - 147s 303ms/step - loss:
1.5441 - accuracy: 0.5989 - val_loss: 1.5866 - val_accuracy: 0.6090

Epoch 00001: saving model to License_character_recognition.h5
Epoch 2/30
486/486 [==============================] - 124s 256ms/step - loss:
0.4201 - accuracy: 0.8930 - val_loss: 0.4698 - val_accuracy: 0.8788

Epoch 00002: saving model to License_character_recognition.h5
Epoch 3/30
486/486 [==============================] - 125s 258ms/step - loss:
0.2769 - accuracy: 0.9281 - val_loss: 0.2617 - val_accuracy: 0.9352

Epoch 00003: saving model to License_character_recognition.h5
Epoch 4/30
486/486 [==============================] - 125s 257ms/step - loss:
0.2103 - accuracy: 0.9454 - val_loss: 0.1744 - val_accuracy: 0.9563

Epoch 00004: saving model to License_character_recognition.h5
Epoch 5/30
486/486 [==============================] - 126s 259ms/step - loss:
0.1642 - accuracy: 0.9554 - val_loss: 0.1491 - val_accuracy: 0.9592
```



The model architecture is stored and saved to avoid reconstruction of base architecture.

```python
# save model architectur as json file
model_json = model.to_json()
with open("MobileNets_character_recognition.json", "w") as json_file:
  json_file.write(model_json)
```

4.5. Collaborating the model

In order to merge the outcomes of Part 1,2 and 3, reconstruction of the model is necessary as in the initial phase the data was extracted and segmented in the next stage.

```python
# Load model architecture, weight and labels
json_file = open('MobileNets_character_recognition.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
model.load_weights("License_character_recognition_weight.h5")
print("[INFO] Model loaded successfully...")

labels = LabelEncoder()
labels.classes_ = np.load('license_character_classes.npy')
print("[INFO] Labels loaded successfully...")
```

```
[INFO] Model loaded successfully...
[INFO] Labels loaded successfully...
```

Previously, we configured the input layer to accept images having the shape sizes (80,80,3). Therefore, we again need to reconvert the image data to the original size along with channel. There is a loop for every image character, which is stored in crop_characters. The final_result predicts and plots every image according to the co-relative prediction.

```python
# pre-processing input images and pedict with model
def predict_from_model(image,model,labels):
    image = cv2.resize(image,(80,80))
    image = np.stack((image,)*3, axis=-1)
    prediction = labels.inverse_transform([np.argmax(model.predict(image[np.newaxis,:]))])
    return prediction

fig = plt.figure(figsize=(15,3))
cols = len(crop_characters)
grid = gridspec.GridSpec(ncols=cols,nrows=1,figure=fig)

final_string = ''
for i,character in enumerate(crop_characters):
    fig.add_subplot(grid[i])
    title = np.array2string(predict_from_model(character,model,labels))
    plt.title('{}'.format(title.strip("'[]"),fontsize=20))
    final_string+=title.strip("'[]")
    plt.axis(False)
    plt.imshow(character,cmap='gray')

print("Achieved result: ", final_string)
#plt.savefig('final_result.png', dpi=300)
```

3. CONCLUSIONS

In this paper, we conducted an experiment which is capable of detecting and recognizing the license plate of vehicles. While experimenting this, there were some obstacles we encountered. Firstly, Wpod-Net system may detect and recognize panels of advertising along with license plate, which could be a cumbersome situation. Further to this, in the segmentation stage, conventional system can be affected by certain climatic circumstances such as illumination, angular perspective, object obstacles and so on. The model is capable to parse the information written or present in Latin language. There are countries like China, Korea, Japan and Saudi Arabia, where Non-Latin characters of license plate are used. However, this problem can be curtailed down by simply adding the additional data into training dataset and re-training the model. Overall, the system has much higher accuracy with greater computational speed which enable this system as optimal.

REFERENCES

1) Polishetty, R., Roopaei, M., Rad, P.: ‘A next-generation secure cloud-based deep learning license plate recognition for smart cities’. IEEE Int. Conf. on Machine Learning and Applications, Anaheim, United States of America, 2016, pp. 286–294.

2) Caltech: ‘Computational vision: archive’, Available at http://www.vision.caltech.edu/html-files/archive.html, accessed August 2018.

3) Selmi, Z., Halima, M., Alimi, A.: ‘Deep learning system for automatic license plate detection and recognition’. Int. Conf. on Document Analysis and Recognition, Kyoto, Japan, 2017, pp. 1132–1138.

4) Liu, W., Anguelov, D., Erhan, D., et al: ‘SSD: single shot MultiBox detector’, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Amsterdam, Netherlands, 2016, vol. 9905, pp. 21–37.

5) Yépez, J., Ko, S.: ‘Improved license plate localization algorithm based on morphological operations’, IET Intell. Transp. Syst., 2018, 12, (6), pp. 542–549.

6) Laroca, R., Severo, E., Zanlorensi, L., et al: ‘A robust real-time automatic license plate recognition based on the YOLO detector’. Int. Joint Conf. on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018.

7) Laroca, R., Zanlorensi, L., Gonçalves, G., et al: ‘An efficient and layout-independent automatic license plate recognition system based on the YOLO detector’, arXiv.org: 1909.01754v2, accessed October 2018.

8) Karthikeyan, V., Vijayalakshmi, V., Jeyakumar, P.: ‘License plate detection using morphological operators’, Available at http://www.academia.edu/2554097/License_Plate_Segmentation_Based_on_Connected_Component_Analysis, accessed October 2018.

9) Jeffrey, Z., Ramalingam, S., Bekooy, N.: ‘Real-time DSP-based license plate character segmentation algorithm using 2D haar wavelet transform’, in Baleanu, D. (Ed.): ‘Advances in wavelet theory and their applications in engineering, physics and technology’ (IntechOpen, Welwyn Garden City, UK, 2012, 1st edn.), pp. 3–22.

10) Kasaei, S., Kasaei, S.: ‘Extraction and recognition of the vehicle license plate for passing under outside environment’. Proc. European Intelligence and Security Informatics Conf., Athens, Greece, 2011, pp. 234–237.

11) Abderaouf, Z., Nadjia, B., Saliha, O.: ‘License plate character segmentation based on horizontal projection and connected component analysis’. World Symp. on Computer Applications & Research, Sousse, Tunisia, 2014, pp. 1–5.

12) Bulan, O., Kozitsky, V., Ramesh, P., et al: ‘Segmentation- and annotation-free license plate recognition with deep localization and failure identification’, IEEE Trans. Intell. Transp. Syst., 2017, 18, (9), pp. 2351–2363.

13) Yépez, J., Castro-Zunti, R., Ko, S.: ‘Deep learning-based embedded license plate localisation system’, IET Intell. Transp. Syst., 2019, 13, (10), pp. 1569–1578.

14) Bengio, Y.: ‘Learning deep architectures for AI, now foundations and trends’ (Now Publishers, Montreal, Canada, 2009, 1st edn.).

15) Montazzolli, S., Jung, C.: ‘Real-time Brazilian license plate detection and recognition using deep convolutional neural networks’. Conf. on Graphics, Patterns and Images, Niteroi, Brazil, 2017, pp. 55–62.

16) How, D., Sahari, K.: ‘Character recognition of Malaysian vehicle license plate with deep convolutional neural networks’. IEEE Int. Symp. on Robotics and Intelligent Sensors, Tokyo, Japan, 2016, pp. 1–5.

BIOGRAPHIES (Optional not mandatory)

1'st Author Photo	Description about the author1 (in 5-6 lines)