**License plate detection**

def detect\_plate (img, text = ''):

This function takes the input image and, using the ‘haar cascade’, detects the license plates and returns the coordinates and dimensions of the contours of the detected license plate. The scaleFactor parameter means the value by which the input image can be scaled for better license plate detection. minNeighbors is just a parameter to reduce false positives, if this value is low, the algorithm may be more likely to give incorrectly recognized results.

**Image processing on the license plate**

def segment\_characters (image):

This function takes the image as input and performs the following operations on it:

• Resizes to such a dimension that all characters appear clear and separated from each other

• Convert a color image to a grayscale image in gray, ie instead of 3 channels (BGR), the image has only one 8-bit channel with values from 0 to 255, where 0 corresponds to black and 255 to white. We do this to prepare the image for the next process

• The threshold function now converts the scaled gray image to a binary image, ie each pixel will now have a value of 0 or 1, where 0 corresponds to black and 1 corresponds to white. This is done by applying a threshold of 0 to 255, here a value of 200, which means in the image in grayscale for pixels with a value above 200, in the new binary image of the pixel will be given a value of 1. And for pixels with a value below 200 in the new binary image will be given a value of 0 for this pixel

• The image is now in binary form and ready for the next Eroding process. Eroding is a simple process used to remove unwanted pixels from the boundary of an object, ie pixels that should have a value of 0 but have a value of 1. It works by considering each pixel in the image one by one and then by considering the pixel neighbor (number of neighbors depends on the size of the kernel), the pixel is given a value of 1, only if all its neighboring pixels are equal to 1, otherwise it is given a value of 0

• The image is now clear and borderless, we will now expand the image to fill in the missing pixels, pixels that should be 1 but have a value of 0. The function works similarly to Eroding, but with a small capture, it works by looking at each pixel on image one by one, and then considering the neighbor of the pixel (the number of neighbors depends on the size of the core), the pixel is given a value of 1, if at least one of the neighboring pixels is equal to 1

• The next step is to make the image borders white. This is required to delete any pixel of the frame, if any.

**Segmentation of characters from the license plate**

def find\_contours (dimensions, img):

This function uses the following steps to remove individual characters from the license plate:

• Find all contours in the input image. The cv2.findContours function returns all the contours it finds in the image. The contours can be explained simply as a curve connecting all continuous points (along the border), having the same color or intensity

• Having found all the contours, we consider them in turn and calculate the dimensions of the corresponding bounding box. Now consider a bounding box - the smallest possible rectangle that contains a contour

• Since we have the dimensions of these bounding boxes, all we need to do is adjust the settings and filter out the desired rectangle that contains the required characters. To do this, we will perform some size comparison, taking only those rectangles that have a width in the range 0 - (picture length) / (number of characters) and a length in the range (picture width) / 2 - 4 \* (picture width) / 5

**Segmentation of characters from the license plate**

def find\_contours (dimensions, img):

This function uses the following steps to remove individual characters from the license plate:

• Find all contours in the input image. The cv2.findContours function returns all the contours it finds in the image. The contours can be explained simply as a curve connecting all continuous points (along the border), having the same color or intensity

• Having found all the contours, we consider them in turn and calculate the dimensions of the corresponding bounding box. Now consider a bounding box - the smallest possible rectangle that contains a contour

• Since we have the dimensions of these bounding boxes, all we need to do is adjust the settings and filter out the desired rectangle that contains the required characters. To do this, we will perform some size comparison, taking only those rectangles that have a width in the range 0 - (picture length) / (number of characters) and a length in the range (picture width) / 2 - 4 \* (picture width) / 5

**Creating a model of machine learning and its training**

For simulation, we will use a convolutional neural network with 9 layers. To keep the model simple, we'll start by creating a sequential Sequential model. The first layer will be a convolution layer with 16 output filters, a convolution window size (22.22) and "Relu" as a function of activation. The second layer will be a convolution layer with 32 output filters, a convolution window size (16,16) and "Relu" as a function of activation. The third layer will be a convolution layer with 64 output filters, a convolution window size (8.8) and "Relu" as a function of activation. The fourth layer will be a convolution layer with 64 output filters, a convolution window size (4.4) and "Relu" as a function of activation. Next, we'll add a maximum merge layer with a window size (4.4). Maximum aggregation is a sampling process based on sampling. The aim is to reduce the sample of input data (image, output matrix of the hidden layer, etc.), reduce its dimensionality and allow to make assumptions about the features contained in the subregions involved. Now we will add a drop out to take care of the over-equipment. Drop out is a regularization hyperparameter initialized to prevent over-neural network debugging. Drop out is a technique where randomly selected neurons are ignored during training. We chose a factor of 0.4, which means that 60% of the node will be saved. Add a layer of Flatten. It takes data from the previous layer and presents them in one dimension. Finally, we will add 2 dense layers, one with the size of the output space as 128, the activation function = 'relu' and the other, our last layer with 36 outputs to classify 26 alphabets (AZ) + 10 digits (0-9) and the activation function = 'softmax'.

The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.

Softmax function - normalized exponential function is a generalization of the logistics function into several dimensions. The softmax function takes as input the vector z from K real numbers and normalizes it into a probability distribution consisting of K probabilities proportional to the exponents of the input numbers. That is, before the application of softmax, some components of the vector may be negative or more than one; and cannot sum up to 1; but after applying softmax, each component will be in the range [0,1], and the sum of the components will be 1, so they can be interpreted as a probability.

**CNN model training**

The model was trained in the file SANDBOX.ipynb. Then the model was saved and applied in the logic of the brain application. The data we will use contains images of alphabets (AZ) and numbers (0-9) in size 28x28, and the data is balanced. The data are divided into 2 parts: 20% and 80%. We will use the ImageDataGenerator class, available in keras, to generate additional data using image magnification techniques such as width and height offsets. Width Offset: Takes a floating-point value to indicate how much of the image will be shifted left and right. Height Offset: Takes a floating-point value to indicate how much of the image will be shifted up and down. During 41 epochs, the model reached an accuracy of 98.38%.