AUTOMATIC NUMBER PLATE RECOGNITION (ANPR) -- By Mayank Vajpayee

In [32]:
```
#installing Opencv library
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\mayan\anaconda4\envs\tf_env
\lib\site-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in c:\users\mayan\anaconda4\envs\tf_env
\lib\site-packages (from opencv-python) (2.1.3)
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
import cv2
```

its a separate command

import zipfile zip_ref = zipfile.ZipFile('/content/archive.zip', 'r') zip_ref.extractall('/content') zip_ref.close()

In [6]:
```
#importing tensot flow
#earlier was getting error to import tensor fow, so created a new environment "tf_e
#and the running jupyter notebbok and the libraries in the new environment
import tensorflow as tf
print(tf.__version__)
```

```
2.19.0
```

In [7]:
```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropo
```

In [8]:
```
# Loads the data required for detecting the license plates from cascade classifier.
plate_cascade = cv2.CascadeClassifier(r'C:\Users\mayan\Full_Stack_Data_Science_Cour
```

In [9]:
```
def detect_plate(img, text=''): # the function detects and perfors blurring on the
    plate_img = img.copy()
    roi = img.copy()
    plate_rect = plate_cascade.detectMultiScale(plate_img, scaleFactor = 1.2, minNe
    for (x,y,w,h) in plate_rect:
        roi_ = roi[y:y+h, x:x+w, :] # extracting the Region of Interest of license
        plate = roi[y:y+h, x:x+w, :]
        cv2.rectangle(plate_img, (x+2,y), (x+w-3, y+h-5), (51,181,155), 3) # finall
    if text!='':
        plate_img = cv2.putText(plate_img, text, (x-w//2,y-h//2),
                                cv2.FONT_HERSHEY_COMPLEX_SMALL , 0.5, (51,181,155),

    return plate_img, plate # returning the processed image.
```

In [11]:
```python
import matplotlib.pyplot as plt
```

In [12]:
```python
# Testing the above function
def display(img_, title=''):
    img = cv2.cvtColor(img_, cv2.COLOR_BGR2RGB)
    fig = plt.figure(figsize=(10,6))
    ax = plt.subplot(111)
    ax.imshow(img)
    plt.axis('off')
    plt.title(title)
    plt.show()

img = cv2.imread(r'C:\Users\mayan\Full_Stack_Data_Science_Course_IIT_Guwahati\Final
display(img, 'input image')
```

input image

In [13]:
```python
# Getting plate prom the processed image
output_img, plate = detect_plate(img)
```

In [14]:
```python
display(output_img, 'detected license plate in the input image')
```

detected license plate in the input image

In [15]:
```
display(plate, 'extracted license plate from the image')
```

extracted license plate from the image

In [16]:
```
# Match contours to license plate or character template
def find_contours(dimensions, img) :

    # Find all contours in the image
    cntrs, _ = cv2.findContours(img.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    # Retrieve potential dimensions
    lower_width = dimensions[0]
    upper_width = dimensions[1]
    lower_height = dimensions[2]
    upper_height = dimensions[3]

    # Check largest 5 or  15 contours for license plate or character respectively
    cntrs = sorted(cntrs, key=cv2.contourArea, reverse=True)[:15]

    ii = cv2.imread('contour.jpg')
```

```python
    x_cntr_list = []
    img_res = []
    for cntr in cntrs :
        # detects contour in binary image and returns the coordinates of rectangle
        intX, intY, intWidth, intHeight = cv2.boundingRect(cntr)

        # checking the dimensions of the contour to filter out the characters by co
        if intWidth > lower_width and intWidth < upper_width and intHeight > lower_
            x_cntr_list.append(intX) #stores the x coordinate of the character's co

            char_copy = np.zeros((44,24))
            # extracting each character using the enclosing rectangle's coordinates
            char = img[intY:intY+intHeight, intX:intX+intWidth]
            char = cv2.resize(char, (20, 40))

            cv2.rectangle(img, (intX,intY), (intWidth+intX, intY+intHeight), (75,30
            plt.imshow(img, cmap='gray')

            # Make result formatted for classification: invert colors
            char = cv2.subtract(255, char)

            # Resize the image to 24x44 with black border
            char_copy[2:42, 2:22] = char
            char_copy[0:2, :] = 0
            char_copy[:, 0:2] = 0
            char_copy[42:44, :] = 0
            char_copy[:, 22:24] = 0

            img_res.append(char_copy) # List that stores the character's binary ima

    # Return characters on ascending order with respect to the x-coordinate (most-l

    plt.show()
    # arbitrary function that stores sorted list of character indeces
    indices = sorted(range(len(x_cntr_list)), key=lambda k: x_cntr_list[k])
    img_res_copy = []
    for idx in indices:
        img_res_copy.append(img_res[idx])# stores character images according to the
    img_res = np.array(img_res_copy)

    return img_res
```

In [18]:

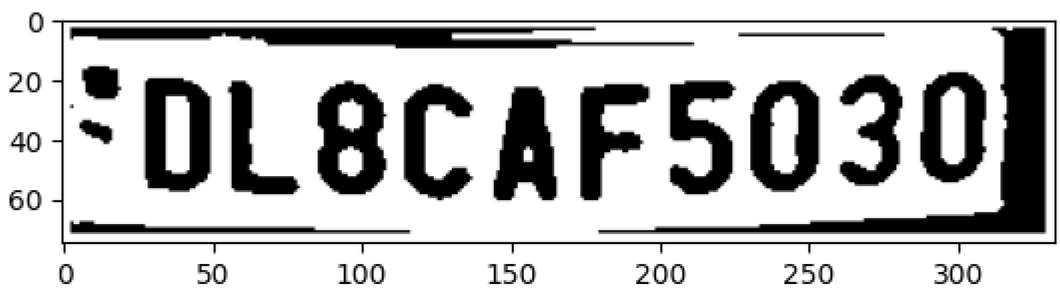
```python
import numpy as np
```

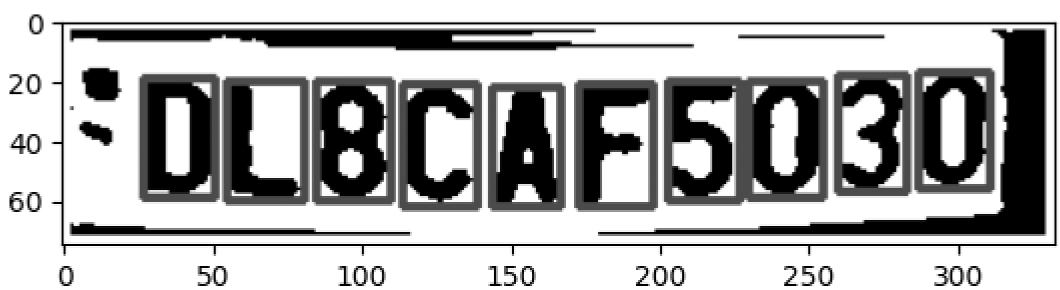
In [19]:

```python
# Find characters in the resulting images
def segment_characters(image) :

    # Preprocess cropped license plate image
    img_lp = cv2.resize(image, (333, 75))
    img_gray_lp = cv2.cvtColor(img_lp, cv2.COLOR_BGR2GRAY)
    _, img_binary_lp = cv2.threshold(img_gray_lp, 200, 255, cv2.THRESH_BINARY+cv2.T
    img_binary_lp = cv2.erode(img_binary_lp, (3,3))
    img_binary_lp = cv2.dilate(img_binary_lp, (3,3))
```

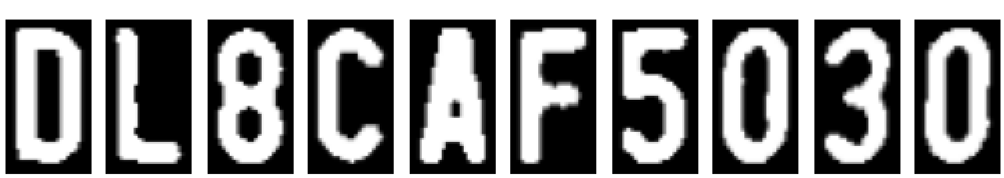
```
    LP_WIDTH = img_binary_lp.shape[0]
    LP_HEIGHT = img_binary_lp.shape[1]

    # Make borders white
    img_binary_lp[0:3,:] = 255
    img_binary_lp[:,0:3] = 255
    img_binary_lp[72:75,:] = 255
    img_binary_lp[:,330:333] = 255

    # Estimations of character contours sizes of cropped license plates
    dimensions = [LP_WIDTH/6,
                       LP_WIDTH/2,
                       LP_HEIGHT/10,
                       2*LP_HEIGHT/3]
    plt.imshow(img_binary_lp, cmap='gray')
    plt.show()
    cv2.imwrite('contour.jpg',img_binary_lp)

    # Get contours within cropped license plate
    char_list = find_contours(dimensions, img_binary_lp)

    return char_list
```

In [20]:
```
char = segment_characters(plate)
```





In [21]:
```
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(char[i], cmap='gray')
    plt.axis('off')
```



In [22]:
```
import tensorflow.keras.backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.1, height_sh
path = 'C:/Users/mayan/Full_Stack_Data_Science_Course_IIT_Guwahati/Final_Notebooks/
train_generator = train_datagen.flow_from_directory(
        path+'/train',  # this is the target directory
        target_size=(28,28),  # all images will be resized to 28x28
        batch_size=1,
        class_mode='sparse')

validation_generator = train_datagen.flow_from_directory(
        path+'/val',  # this is the target directory
        target_size=(28,28),  # all images will be resized to 28x28 batch_size=1,
        class_mode='sparse')
```

```
Found 864 images belonging to 36 classes.
Found 216 images belonging to 36 classes.
```

In [23]:
```
from tensorflow.keras import optimizers
K.clear_session()
model = Sequential()
model.add(Conv2D(16,kernel_size=(22,22),padding='same',activation='relu',input_shap
model.add(Conv2D(32,kernel_size=(16,16),padding='same',activation='relu'))
model.add(Conv2D(64,kernel_size=(8,8),padding='same',activation='relu'))
model.add(Conv2D(64,kernel_size=(4,4),padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(36, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizers.Adam(lea
```

```
WARNING:tensorflow:From C:\Users\mayan\anaconda4\envs\tf_env\lib\site-packages\keras
\src\backend\common\global_state.py:82: The name tf.reset_default_graph is deprecate
d. Please use tf.compat.v1.reset_default_graph instead.
```

```
C:\Users\mayan\anaconda4\envs\tf_env\lib\site-packages\keras\src\layers\convolutiona
l\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument t
o a layer. When using Sequential models, prefer using an `Input(shape)` object as th
e first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [24]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	
conv2d (Conv2D)	(None, 28, 28, 16)	
conv2d_1 (Conv2D)	(None, 28, 28, 32)	
conv2d_2 (Conv2D)	(None, 28, 28, 64)	
conv2d_3 (Conv2D)	(None, 28, 28, 64)	
max_pooling2d (MaxPooling2D)	(None, 7, 7, 64)	
dropout (Dropout)	(None, 7, 7, 64)	
flatten (Flatten)	(None, 3136)	
dense (Dense)	(None, 128)	
dense_1 (Dense)	(None, 36)	

Total params: 757,268 (2.89 MB)

Trainable params: 757,268 (2.89 MB)

Non-trainable params: 0 (0.00 B)

In [25]:
```
class stop_training_callback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_accuracy') > 0.999):
      self.model.stop_training = True
```

In [1]:
```
pip install Pillow
```

```
Requirement already satisfied: Pillow in c:\users\mayan\anaconda4\envs\tf_env\lib\si
te-packages (11.2.1)
Note: you may need to restart the kernel to use updated packages.
```

In [26]:
```
from PIL import Image

img = Image.new('RGB', (100, 100), color='red')
img.show()
```

In [28]:
```
pip install scipy
```

```
Collecting scipy
  Downloading scipy-1.15.3-cp310-cp310-win_amd64.whl.metadata (60 kB)
Requirement already satisfied: numpy<2.5,>=1.23.5 in c:\users\mayan\anaconda4\envs\t
f_env\lib\site-packages (from scipy) (2.1.3)
Downloading scipy-1.15.3-cp310-cp310-win_amd64.whl (41.3 MB)
   ---------------------------------------- 0.0/41.3 MB ? eta -:--:--
   --- ------------------------------------ 3.4/41.3 MB 16.7 MB/s eta 0:00:03
   --- ------------------------------------ 3.9/41.3 MB 16.8 MB/s eta 0:00:03
   ---- ----------------------------------- 4.5/41.3 MB 7.4 MB/s eta 0:00:05
   ---- ----------------------------------- 5.0/41.3 MB 6.6 MB/s eta 0:00:06
   ----- ---------------------------------- 6.0/41.3 MB 5.8 MB/s eta 0:00:07
   ------- -------------------------------- 7.3/41.3 MB 5.6 MB/s eta 0:00:07
   ------- -------------------------------- 7.9/41.3 MB 5.5 MB/s eta 0:00:07
   -------- ------------------------------- 8.9/41.3 MB 5.1 MB/s eta 0:00:07
   --------- ------------------------------ 10.0/41.3 MB 5.2 MB/s eta 0:00:07
   ---------- ----------------------------- 10.7/41.3 MB 5.0 MB/s eta 0:00:07
   ----------- ---------------------------- 12.1/41.3 MB 5.1 MB/s eta 0:00:06
   ------------ --------------------------- 12.8/41.3 MB 5.0 MB/s eta 0:00:06
   ------------- -------------------------- 14.2/41.3 MB 5.1 MB/s eta 0:00:06
   -------------- ------------------------- 14.9/41.3 MB 5.1 MB/s eta 0:00:06
   --------------- ------------------------ 16.0/41.3 MB 5.0 MB/s eta 0:00:06
   ---------------- ----------------------- 17.0/41.3 MB 5.0 MB/s eta 0:00:05
   ----------------- ---------------------- 18.1/41.3 MB 5.0 MB/s eta 0:00:05
   ------------------ --------------------- 18.9/41.3 MB 5.0 MB/s eta 0:00:05
   ------------------- -------------------- 20.2/41.3 MB 5.0 MB/s eta 0:00:05
   -------------------- ------------------- 21.2/41.3 MB 5.0 MB/s eta 0:00:04
   --------------------- ------------------ 22.3/41.3 MB 5.0 MB/s eta 0:00:04
   ---------------------- ----------------- 23.3/41.3 MB 5.0 MB/s eta 0:00:04
   ----------------------- ---------------- 24.1/41.3 MB 5.0 MB/s eta 0:00:04
   ------------------------ --------------- 25.2/41.3 MB 5.0 MB/s eta 0:00:04
   ------------------------- -------------- 26.2/41.3 MB 5.0 MB/s eta 0:00:04
   -------------------------- ------------- 27.3/41.3 MB 5.0 MB/s eta 0:00:03
   --------------------------- ------------ 28.3/41.3 MB 5.0 MB/s eta 0:00:03
   ---------------------------- ----------- 29.4/41.3 MB 5.0 MB/s eta 0:00:03
   ----------------------------- ---------- 30.1/41.3 MB 5.0 MB/s eta 0:00:03
   ------------------------------ --------- 31.2/41.3 MB 5.0 MB/s eta 0:00:03
   ------------------------------- -------- 32.2/41.3 MB 5.0 MB/s eta 0:00:02
   -------------------------------- ------- 33.3/41.3 MB 5.0 MB/s eta 0:00:02
   --------------------------------- ------ 34.3/41.3 MB 5.0 MB/s eta 0:00:02
   ---------------------------------- ----- 35.4/41.3 MB 5.0 MB/s eta 0:00:02
   ----------------------------------- ---- 36.4/41.3 MB 5.0 MB/s eta 0:00:01
   ------------------------------------ --- 37.5/41.3 MB 5.0 MB/s eta 0:00:01
   ------------------------------------- -- 38.5/41.3 MB 4.9 MB/s eta 0:00:01
   -------------------------------------- - 39.6/41.3 MB 5.0 MB/s eta 0:00:01
   ---------------------------------------  40.6/41.3 MB 5.0 MB/s eta 0:00:01
   ---------------------------------------  41.2/41.3 MB 4.9 MB/s eta 0:00:01
   ---------------------------------------- 41.3/41.3 MB 4.9 MB/s eta 0:00:00
Installing collected packages: scipy
Successfully installed scipy-1.15.3
Note: you may need to restart the kernel to use updated packages.
```

In [29]:

```
batch_size = 1
callbacks = [stop_training_callback()]
model.fit(
      train_generator,
      steps_per_epoch = train_generator.samples // batch_size,
```

```
    validation_data = validation_generator,
    epochs = 80, verbose=1, callbacks=callbacks)
```

```
Epoch 1/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.0728 - loss: 3.4810
```

```
C:\Users\mayan\anaconda4\envs\tf_env\lib\site-packages\keras\src\trainers\data_adapt
ers\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `supe
r().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_m
ultiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they w
ill be ignored.
  self._warn_if_super_not_called()
```

```
864/864 ━━━━━━━━━━━━━━━━━━━━ 34s 36ms/step - accuracy: 0.0729 - loss: 3.4805 - val_a
ccuracy: 0.4954 - val_loss: 1.7115
Epoch 2/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 27s 32ms/step - accuracy: 0.4947 - loss: 1.6275 - val_a
ccuracy: 0.6944 - val_loss: 0.8638
Epoch 3/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 26ms/step - accuracy: 0.7226 - loss: 0.8709 - val_a
ccuracy: 0.8657 - val_loss: 0.4835
Epoch 4/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.8645 - loss: 0.4502 - val_a
ccuracy: 0.8981 - val_loss: 0.3588
Epoch 5/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 26ms/step - accuracy: 0.8960 - loss: 0.3751 - val_a
ccuracy: 0.9444 - val_loss: 0.1943
Epoch 6/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9132 - loss: 0.2895 - val_a
ccuracy: 0.9167 - val_loss: 0.2671
Epoch 7/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9366 - loss: 0.1950 - val_a
ccuracy: 0.9583 - val_loss: 0.2184
Epoch 8/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9412 - loss: 0.1914 - val_a
ccuracy: 0.9398 - val_loss: 0.1827
Epoch 9/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 27ms/step - accuracy: 0.9527 - loss: 0.1308 - val_a
ccuracy: 0.9537 - val_loss: 0.1273
Epoch 10/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 26ms/step - accuracy: 0.9607 - loss: 0.1048 - val_a
ccuracy: 0.9444 - val_loss: 0.1178
Epoch 11/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9561 - loss: 0.1805 - val_a
ccuracy: 0.9676 - val_loss: 0.1172
Epoch 12/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9589 - loss: 0.1162 - val_a
ccuracy: 0.9722 - val_loss: 0.1254
Epoch 13/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9522 - loss: 0.1545 - val_a
ccuracy: 0.9815 - val_loss: 0.0601
Epoch 14/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9727 - loss: 0.0769 - val_a
ccuracy: 0.9861 - val_loss: 0.0499
Epoch 15/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 27s 31ms/step - accuracy: 0.9685 - loss: 0.0947 - val_a
ccuracy: 0.9861 - val_loss: 0.0362
Epoch 16/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9662 - loss: 0.1125 - val_a
ccuracy: 0.9676 - val_loss: 0.1380
Epoch 17/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9580 - loss: 0.1296 - val_a
ccuracy: 0.9722 - val_loss: 0.0931
Epoch 18/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 28ms/step - accuracy: 0.9719 - loss: 0.0770 - val_a
ccuracy: 0.9815 - val_loss: 0.0694
Epoch 19/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 28ms/step - accuracy: 0.9820 - loss: 0.0502 - val_a
ccuracy: 0.9954 - val_loss: 0.0236
```

```
Epoch 20/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 28ms/step - accuracy: 0.9739 - loss: 0.0696 - val_a
ccuracy: 0.9815 - val_loss: 0.0888
Epoch 21/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 28ms/step - accuracy: 0.9885 - loss: 0.0403 - val_a
ccuracy: 0.9769 - val_loss: 0.0741
Epoch 22/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 28ms/step - accuracy: 0.9455 - loss: 0.1863 - val_a
ccuracy: 0.9815 - val_loss: 0.0653
Epoch 23/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9769 - loss: 0.0710 - val_a
ccuracy: 0.9722 - val_loss: 0.0516
Epoch 24/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 27ms/step - accuracy: 0.9808 - loss: 0.0693 - val_a
ccuracy: 0.9722 - val_loss: 0.1093
Epoch 25/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9778 - loss: 0.0726 - val_a
ccuracy: 0.9583 - val_loss: 0.1501
Epoch 26/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9652 - loss: 0.0859 - val_a
ccuracy: 0.9630 - val_loss: 0.0551
Epoch 27/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 27s 31ms/step - accuracy: 0.9852 - loss: 0.0370 - val_a
ccuracy: 0.9815 - val_loss: 0.0650
Epoch 28/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9813 - loss: 0.0529 - val_a
ccuracy: 0.9444 - val_loss: 0.1772
Epoch 29/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9666 - loss: 0.0799 - val_a
ccuracy: 0.9722 - val_loss: 0.0854
Epoch 30/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9690 - loss: 0.1111 - val_a
ccuracy: 0.9722 - val_loss: 0.0691
Epoch 31/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9701 - loss: 0.0649 - val_a
ccuracy: 0.9769 - val_loss: 0.0457
Epoch 32/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9911 - loss: 0.0237 - val_a
ccuracy: 0.9815 - val_loss: 0.0452
Epoch 33/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9816 - loss: 0.0407 - val_a
ccuracy: 0.9815 - val_loss: 0.0687
Epoch 34/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9631 - loss: 0.1492 - val_a
ccuracy: 0.9907 - val_loss: 0.0550
Epoch 35/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 28ms/step - accuracy: 0.9641 - loss: 0.1375 - val_a
ccuracy: 0.9861 - val_loss: 0.0301
Epoch 36/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9816 - loss: 0.0431 - val_a
ccuracy: 0.9769 - val_loss: 0.0361
Epoch 37/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9672 - loss: 0.0837 - val_a
ccuracy: 0.9722 - val_loss: 0.0481
Epoch 38/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9873 - loss: 0.0378 - val_a
```

```
ccuracy: 0.9907 - val_loss: 0.0235
Epoch 39/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9706 - loss: 0.0889 - val_a
ccuracy: 0.9815 - val_loss: 0.0462
Epoch 40/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9936 - loss: 0.0166 - val_a
ccuracy: 0.9861 - val_loss: 0.0307
Epoch 41/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9922 - loss: 0.0407 - val_a
ccuracy: 0.9491 - val_loss: 0.1452
Epoch 42/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 28ms/step - accuracy: 0.9809 - loss: 0.0520 - val_a
ccuracy: 0.9861 - val_loss: 0.0315
Epoch 43/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9904 - loss: 0.0328 - val_a
ccuracy: 0.9491 - val_loss: 0.1898
Epoch 44/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9616 - loss: 0.1107 - val_a
ccuracy: 0.9861 - val_loss: 0.0703
Epoch 45/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9878 - loss: 0.0290 - val_a
ccuracy: 0.9769 - val_loss: 0.0994
Epoch 46/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9883 - loss: 0.0261 - val_a
ccuracy: 0.9815 - val_loss: 0.0604
Epoch 47/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9737 - loss: 0.0606 - val_a
ccuracy: 0.9537 - val_loss: 0.2130
Epoch 48/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9725 - loss: 0.0991 - val_a
ccuracy: 0.9630 - val_loss: 0.0908
Epoch 49/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9806 - loss: 0.0684 - val_a
ccuracy: 0.9815 - val_loss: 0.0758
Epoch 50/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9869 - loss: 0.0248 - val_a
ccuracy: 0.9815 - val_loss: 0.0421
Epoch 51/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9896 - loss: 0.0264 - val_a
ccuracy: 0.9815 - val_loss: 0.0482
Epoch 52/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9825 - loss: 0.0418 - val_a
ccuracy: 0.9954 - val_loss: 0.0251
Epoch 53/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9863 - loss: 0.0350 - val_a
ccuracy: 0.9861 - val_loss: 0.0442
Epoch 54/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9954 - loss: 0.0206 - val_a
ccuracy: 0.9954 - val_loss: 0.0307
Epoch 55/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9919 - loss: 0.0267 - val_a
ccuracy: 0.9954 - val_loss: 0.0212
Epoch 56/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9731 - loss: 0.0626 - val_a
ccuracy: 0.9676 - val_loss: 0.0668
Epoch 57/80
```

```
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9811 - loss: 0.0605 - val_a
ccuracy: 0.9907 - val_loss: 0.0332
Epoch 58/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9788 - loss: 0.0622 - val_a
ccuracy: 0.9583 - val_loss: 0.2134
Epoch 59/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9810 - loss: 0.0659 - val_a
ccuracy: 0.9815 - val_loss: 0.0403
Epoch 60/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9922 - loss: 0.0312 - val_a
ccuracy: 0.9815 - val_loss: 0.1006
Epoch 61/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9878 - loss: 0.0355 - val_a
ccuracy: 0.9722 - val_loss: 0.0631
Epoch 62/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9819 - loss: 0.0428 - val_a
ccuracy: 0.9769 - val_loss: 0.0679
Epoch 63/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 28ms/step - accuracy: 0.9889 - loss: 0.0319 - val_a
ccuracy: 0.9769 - val_loss: 0.0352
Epoch 64/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 23s 27ms/step - accuracy: 0.9862 - loss: 0.0345 - val_a
ccuracy: 0.9954 - val_loss: 0.0185
Epoch 65/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9877 - loss: 0.0593 - val_a
ccuracy: 0.9769 - val_loss: 0.1113
Epoch 66/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9795 - loss: 0.0365 - val_a
ccuracy: 0.9861 - val_loss: 0.0479
Epoch 67/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 25ms/step - accuracy: 0.9840 - loss: 0.0311 - val_a
ccuracy: 0.9907 - val_loss: 0.0188
Epoch 68/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9869 - loss: 0.0276 - val_a
ccuracy: 0.9954 - val_loss: 0.0194
Epoch 69/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9947 - loss: 0.0259 - val_a
ccuracy: 0.9907 - val_loss: 0.0162
Epoch 70/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 28ms/step - accuracy: 0.9908 - loss: 0.0214 - val_a
ccuracy: 0.9954 - val_loss: 0.0147
Epoch 71/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9569 - loss: 0.1041 - val_a
ccuracy: 0.9861 - val_loss: 0.0862
Epoch 72/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9884 - loss: 0.0306 - val_a
ccuracy: 0.9954 - val_loss: 0.0160
Epoch 73/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9812 - loss: 0.0409 - val_a
ccuracy: 0.9769 - val_loss: 0.0379
Epoch 74/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9823 - loss: 0.0505 - val_a
ccuracy: 0.9815 - val_loss: 0.0866
Epoch 75/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9768 - loss: 0.0483 - val_a
ccuracy: 0.9954 - val_loss: 0.0108
```

```
Epoch 76/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9858 - loss: 0.0259 - val_a
ccuracy: 0.9907 - val_loss: 0.0191
Epoch 77/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9932 - loss: 0.0170 - val_a
ccuracy: 0.9722 - val_loss: 0.2631
Epoch 78/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 22s 26ms/step - accuracy: 0.9794 - loss: 0.0597 - val_a
ccuracy: 0.9769 - val_loss: 0.0514
Epoch 79/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 26s 30ms/step - accuracy: 0.9837 - loss: 0.0566 - val_a
ccuracy: 0.9954 - val_loss: 0.0134
Epoch 80/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 24s 28ms/step - accuracy: 0.9898 - loss: 0.0197 - val_a
ccuracy: 0.9907 - val_loss: 0.0192
```

Out[29]:
```
<keras.src.callbacks.history.History at 0x14428d0d0c0>
```

In [30]:
```
# Predicting the output
def fix_dimension(img):
  new_img = np.zeros((28,28,3))
  for i in range(3):
    new_img[:,:,i] = img
  return new_img

def show_results():
    dic = {}
    characters = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    for i,c in enumerate(characters):
        dic[i] = c

    output = []
    for i,ch in enumerate(char): #iterating over the characters
        img_ = cv2.resize(ch, (28,28), interpolation=cv2.INTER_AREA)
        img = fix_dimension(img_)
        img = img.reshape(1,28,28,3) #preparing image for the model
        y_ = model.predict(img)[0] #predicting the class
        # Get the index of the predicted class (class with highest probability)
        character_index = np.argmax(y_)
        character = dic[character_index] # Use character_index as key to access dic
        output.append(character) #storing the result in a list

    plate_number = ''.join(output)

    return plate_number

print(show_results())
```

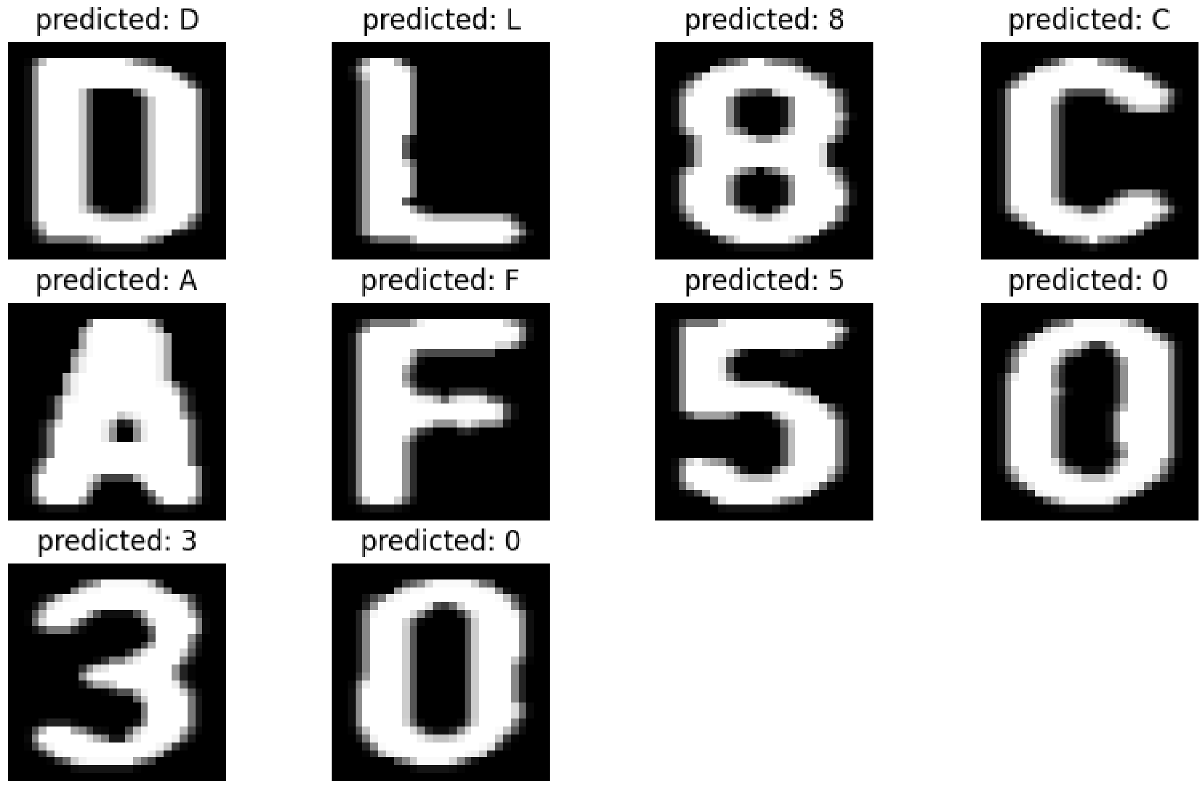
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
DL8CAF5030
```

In [31]:

```
# Segmented characters and their predicted value.
plt.figure(figsize=(10,6))
for i,ch in enumerate(char):
    img = cv2.resize(ch, (28,28), interpolation=cv2.INTER_AREA)
    plt.subplot(3,4,i+1)
    plt.imshow(img,cmap='gray')
    plt.title(f'predicted: {show_results()[i]}')
    plt.axis('off')
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```


predicted: D
predicted: L
predicted: 8
predicted: C
predicted: A
predicted: F
predicted: 5
predicted: 0
predicted: 3
predicted: 0

In [ ]: