FINAL CAPSTONE PROJECT :- 3 :- (RELATED TO CONVOLUTIONAL NEURAL NETWORK(CNN))

FINAL CAPSTONE PROJECT NAME :- AUTOMATIC NUMBER PLATE RECOGNITION

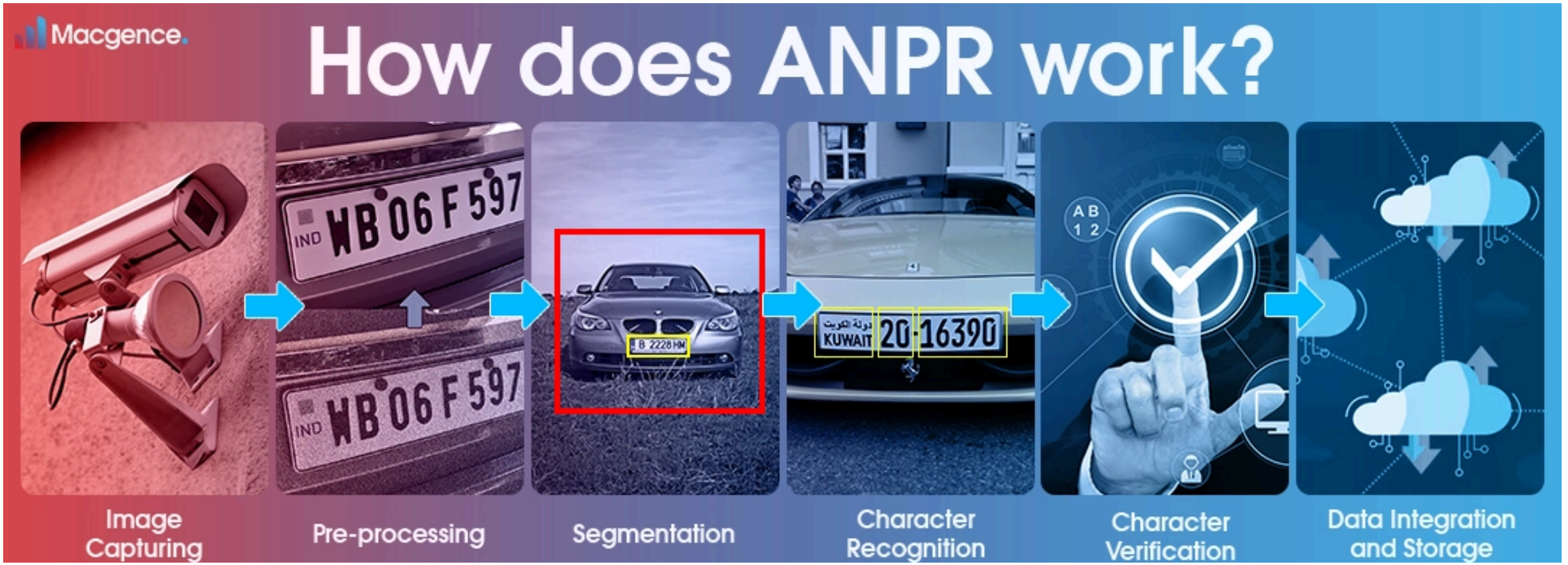
Start coding or generate with AI.

CREATED BY :- JAYASHRI PACHARANE, DATE :- 10TH MARCH 2025

Start coding or generate with AI.

IMAGE:-

```
from IPython import display
display.Image('/content/Automatic_number_plate_recognition.png')
```



Start coding or generate with AI.

```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (2.0.2)
```

Start coding or generate with AI.

```
import cv2
```

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

Start coding or generate with AI.

IMPORTING LIBRARIES AND DATA :-

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import tensorflow as tf
from sklearn.metrics import f1_score
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten, MaxPooling2D, Dropout, Conv2D
```

Start coding or generate with AI.

Load the data required for detecting the license plates:-

```
# Loads the data required for detecting the license plates from cascade classifier.
plate_cascade = cv2.CascadeClassifier('/content/indian_license_plate.xml')
# add the path to 'india_license_plate.xml' file.
```

Start coding or generate with AI.

The Function detects and perfors on the number plate:-

```
def detect_plate(img, text=''): # the function detects and perfors blurring on the number plate.
    plate_img = img.copy()
    roi = img.copy()
    plate_rect = plate_cascade.detectMultiScale(plate_img, scaleFactor = 1.2, minNeighbors = 7) # detects numberplates and returns the coordinates
    for (x,y,w,h) in plate_rect:
        roi_ = roi[y:y+h, x:x+w, :] # extracting the Region of Interest of license plate for blurring.
        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) # finally representing the detected contours by drawing rectangles arou
    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), 1, cv2.LINE_AA)

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

Start coding or generate with AI.

```
import matplotlib.pyplot as plt
```

Testing the above function:-

```
# 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('/content/car.jpg')
display(img, 'Input image')
```

Start coding or generate with AI.

Getting plate prom the processed image:-

```
# Getting plate prom the processed image
output_img, plate = detect_plate(img)
```

Start coding or generate with AI.

Detected License Plate in the Input Image:-

```
display(output_img, 'Detected License Plate in the Input Image')
```

Start coding or generate with AI.

Extracted License Plate From The Image:-

```
display(plate, 'Extracted License Plate from the Image')
```

Start coding or generate with AI.

Match Contours to License Plate:-

```
# 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')

    x_cntr_list = []
    target_contours = []
    img_res = []
    for cntr in cntrs :
        # detects contour in binary image and returns the coordinates of rectangle enclosing it
```

```
        intX, intY, intWidth, intHeight = cv2.boundingRect(cntr)

        # checking the dimensions of the contour to filter out the characters by contour's size
        if intWidth > lower_width and intWidth < upper_width and intHeight > lower_height and intHeight < upper_height :
            x_cntr_list.append(intX) #stores the x coordinate of the character's contour, to used later for indexing the contours

            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(ii, (intX,intY), (intWidth+intX, intY+intHeight), (50,21,200), 2)
            plt.imshow(ii, 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 image (unsorted)

    # Return characters on ascending order with respect to the x-coordinate (most-left character first)

    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 their index
    img_res = np.array(img_res_copy)

    return img_res
```

Start coding or generate with AI.

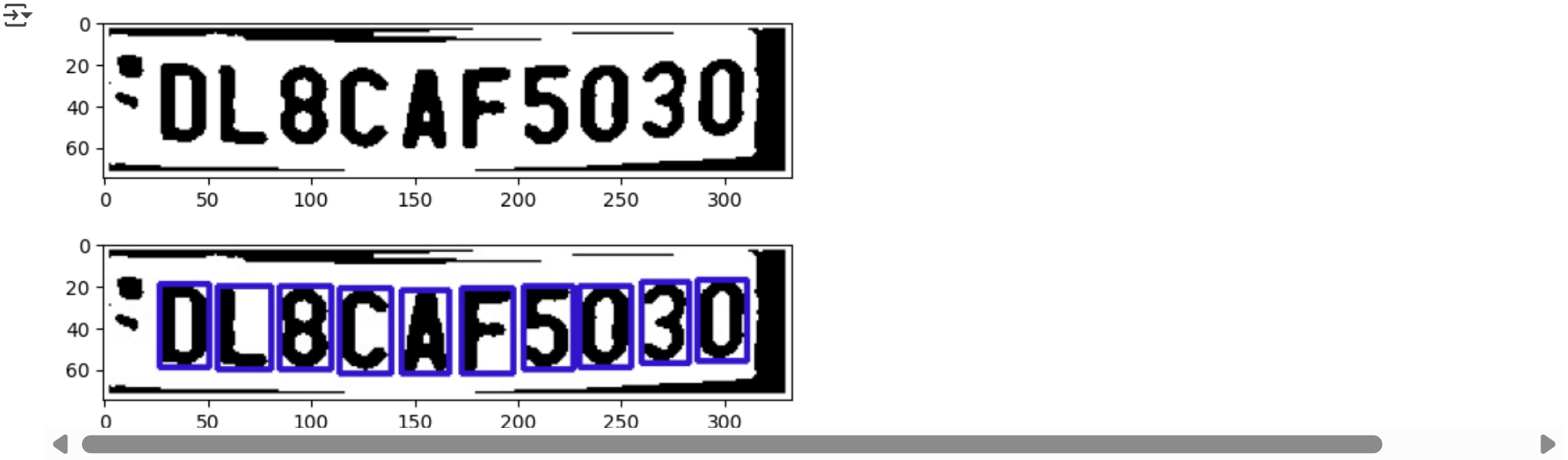
Find Characters in the Resulting Images:-

```
# 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.THRESH_OTSU)
    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
```

Start coding or generate with AI.

Let's see the Segmented Characters:-

```
# Let's see the segmented characters
char = segment_characters(plate)
```



Start coding or generate with AI.

Using For Loop:-

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

Start coding or generate with AI.

Model for characters:-

```
import tensorflow.keras.backend as K
train_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.1, height_shift_range=0.1)
path = '/content/data'
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.
```

Start coding or generate with AI.

CNN MODEL:-

```
from tensorflow.keras import optimizers
K.clear_session()
model = Sequential()
model.add(Conv2D(16,kernel_size=(22,22),padding='same',activation='relu',input_shape=(28,28,3)))
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(learning_rate=0.0001), metrics=["accuracy"])
```

Start coding or generate with AI.

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	23,248
conv2d_1 (Conv2D)	(None, 28, 28, 32)	131,104
conv2d_2 (Conv2D)	(None, 28, 28, 64)	131,136
conv2d_3 (Conv2D)	(None, 28, 28, 64)	65,600
max_pooling2d (MaxPooling2D)	(None, 7, 7, 64)	0
dropout (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 128)	401,536
dense_1 (Dense)	(None, 36)	4,644

Total params: 757,268 (2.89 MB)

Start coding or generate with AI.

```
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
```

Start coding or generate with AI.

```
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 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9818 - loss: 0.0473 - val_accuracy: 0.9815 - val_loss: 0.0445
Epoch 2/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9841 - loss: 0.0408 - val_accuracy: 0.9676 - val_loss: 0.1315
Epoch 3/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9694 - loss: 0.0880 - val_accuracy: 0.9861 - val_loss: 0.0687
Epoch 4/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9802 - loss: 0.0374 - val_accuracy: 0.9444 - val_loss: 0.1764
Epoch 5/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9538 - loss: 0.1566 - val_accuracy: 0.9907 - val_loss: 0.0220
Epoch 6/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9844 - loss: 0.0334 - val_accuracy: 0.9861 - val_loss: 0.0252
Epoch 7/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9794 - loss: 0.0550 - val_accuracy: 0.9907 - val_loss: 0.0281
Epoch 8/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9751 - loss: 0.0767 - val_accuracy: 0.9907 - val_loss: 0.0376
Epoch 9/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9704 - loss: 0.0542 - val_accuracy: 0.9815 - val_loss: 0.0862
Epoch 10/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9710 - loss: 0.0742 - val_accuracy: 0.9954 - val_loss: 0.0151
Epoch 11/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9925 - loss: 0.0318 - val_accuracy: 0.9769 - val_loss: 0.0596
Epoch 12/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9747 - loss: 0.0889 - val_accuracy: 0.9954 - val_loss: 0.0304
Epoch 13/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9915 - loss: 0.0231 - val_accuracy: 0.9815 - val_loss: 0.0480
Epoch 14/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9772 - loss: 0.0595 - val_accuracy: 0.9954 - val_loss: 0.0172
Epoch 15/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9878 - loss: 0.0433 - val_accuracy: 0.9907 - val_loss: 0.0266
Epoch 16/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9905 - loss: 0.0261 - val_accuracy: 0.9583 - val_loss: 0.1743
Epoch 17/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9870 - loss: 0.0441 - val_accuracy: 0.9769 - val_loss: 0.1166
Epoch 18/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9846 - loss: 0.0593 - val_accuracy: 0.9722 - val_loss: 0.0854
Epoch 19/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9836 - loss: 0.0362 - val_accuracy: 0.9861 - val_loss: 0.0843
Epoch 20/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9771 - loss: 0.0521 - val_accuracy: 0.9630 - val_loss: 0.1457
```

```
Epoch 21/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9882 - loss: 0.0428 - val_accuracy: 0.9861 - val_loss: 0.0326
Epoch 22/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9806 - loss: 0.0599 - val_accuracy: 0.9861 - val_loss: 0.0349
Epoch 23/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9904 - loss: 0.0263 - val_accuracy: 0.9769 - val_loss: 0.0665
Epoch 24/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9833 - loss: 0.0541 - val_accuracy: 0.9861 - val_loss: 0.0240
Epoch 25/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9849 - loss: 0.0297 - val_accuracy: 0.9861 - val_loss: 0.0679
Epoch 26/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9839 - loss: 0.0341 - val_accuracy: 0.9676 - val_loss: 0.0891
Epoch 27/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9863 - loss: 0.0536 - val_accuracy: 0.9907 - val_loss: 0.0355
Epoch 28/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9852 - loss: 0.0447 - val_accuracy: 0.9861 - val_loss: 0.0233
Epoch 29/80
864/864 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9725 - loss: 0.1012 - val_accuracy: 0.9861 - val_loss: 0.0748
```

Start coding or generate with AI.

Predicting the Output:-

```
# 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 dictionary
        output.append(character) #storing the result in a list

    plate_number = ''.join(output)

    return plate_number

print(show_results())
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
DL8CAF5030
```

Start coding or generate with AI.

Segmented Characters and their Predicted Value:-

```
# 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 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```