In [3]:
```
pip install imutils
```

```
Collecting imutils
  Downloading imutils-0.5.4.tar.gz (17 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: imutils
  Building wheel for imutils (setup.py): started
  Building wheel for imutils (setup.py): finished with status 'done'
  Created wheel for imutils: filename=imutils-0.5.4-py3-none-any.whl size=25857 sha256=5023a7d51b33aacb2d3acfbfb1f5af6cffd2
72db109dd80dc0b2d75a7967ac48
  Stored in directory: c:\users\adars\appdata\local\pip\cache\wheels\5b\76\96\ad0c321506837bef578cf3008df3916c23018435a355d
9f6b1
Successfully built imutils
Installing collected packages: imutils
Successfully installed imutils-0.5.4
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
import os
import cv2
import imutils
import numpy as np
import glob
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

CAPTCHA_IMAGE_FOLDER = "captcha_images"
OUTPUT_FOLDER = "extracted_letter_images"

def extract_letters_from_captcha():
    """
    Extracts letters from CAPTCHA images in the folder.
    Each CAPTCHA is assumed to have exactly 4 letters.
    """
    captcha_image_files = glob.glob(os.path.join(CAPTCHA_IMAGE_FOLDER, "*"))
    data = []
    labels = []

    for captcha_image_file in captcha_image_files:
        print(f"[INFO] Processing {captcha_image_file}...")
        # Extract the correct text from the filename
        filename = os.path.basename(captcha_image_file)
        captcha_correct_text = os.path.splitext(filename)[0]

        # Load the image and convert it to grayscale
        image = cv2.imread(captcha_image_file)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Add extra padding around the image
        gray = cv2.copyMakeBorder(gray, 8, 8, 8, 8, cv2.BORDER_REPLICATE)

        # Threshold the image (convert it to pure black and white)
        thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]

        # Find contours (continuous blobs of pixels) in the image
        contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        letter_image_regions = []

        # Loop through each contour and extract the letter inside
        for contour in contours:
            (x, y, w, h) = cv2.boundingRect(contour)

            # Split wide contours (conjoined letters) into two regions
            if w / h > 1.25:
                half_width = int(w / 2)
                letter_image_regions.append((x, y, half_width, h))
                letter_image_regions.append((x + half_width, y, half_width, h))
            else:
                letter_image_regions.append((x, y, w, h))

        # Skip the image if there are not exactly 4 letters detected
        if len(letter_image_regions) != 4:
            continue

        # Sort the detected letter images from left-to-right
        letter_image_regions = sorted(letter_image_regions, key=lambda x: x[0])

        # Extract each letter as an individual image
        for letter_bounding_box, letter_text in zip(letter_image_regions, captcha_correct_text):
            x, y, w, h = letter_bounding_box
            letter_image = gray[y - 2:y + h + 2, x - 2:x + w + 2]

            # Resize the letter image (if resizing is needed)
            letter_image_resized = cv2.resize(letter_image, (20, 20))

            # Append the resized letter image to the data list
            data.append(letter_image_resized)

            # Label the extracted letter
            labels.append(letter_text)

    return np.array(data), np.array(labels)


# Load the images and labels
```

```
# Load the images and labels
data, labels = extract_letters_from_captcha()

# Preprocess the data
# Reshape data for CNN input and normalize the images
data = data.reshape((data.shape[0], 20, 20, 1))  # (num_samples, height, width, channels)
data = data.astype("float32") / 255.0  # Normalize pixel values to [0, 1]

# Convert labels to categorical (one-hot encoding)
# Define custom character set excluding "O" and "I"
char_set = "23456789ABCDEFGHJKLMNPQRSTUVWXYZ"
char_to_label = {char: idx for idx, char in enumerate(char_set)}

# Encode labels to integers based on the custom character set
labels = np.array([char_to_label[char] for char in labels])
labels = to_categorical(labels, num_classes=32)  # 32 unique characters

# Build the CNN model
model = Sequential()

# First convolutional layer with max pooling
model.add(Conv2D(20, (5, 5), padding="same", input_shape=(20, 20, 1), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Second convolutional layer with max pooling
model.add(Conv2D(50, (5, 5), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Hidden layer with 500 nodes
model.add(Flatten())
model.add(Dense(500, activation="relu"))

# Output layer with 32 nodes (one for each possible character)
model.add(Dense(32, activation="softmax"))
```

```
# Compile the model
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

from sklearn.model_selection import train_test_split

# Split the data
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)


# Train the model
model.fit(X_train, y_train, batch_size=32, epochs=5, validation_data=(X_test, y_test), verbose=1)


# Save the trained model
model.save("captcha_model.h5")
```

```
[INFO] Processing captcha_images\22A6.png...
[INFO] Processing captcha_images\22BJ.png...
[INFO] Processing captcha_images\22HS.png...
[INFO] Processing captcha_images\22KD.png...
[INFO] Processing captcha_images\22L9.png...
[INFO] Processing captcha_images\22NR.png...
[INFO] Processing captcha_images\22PL.png...
[INFO] Processing captcha_images\22SS.png...
[INFO] Processing captcha_images\22UX.png...
[INFO] Processing captcha_images\23D7.png...
[INFO] Processing captcha_images\23EZ.png...
[INFO] Processing captcha_images\23FQ.png...
[INFO] Processing captcha_images\23T2.png...
[INFO] Processing captcha_images\23XT.png...
[INFO] Processing captcha_images\243B.png...
[INFO] Processing captcha_images\248C.png...
[INFO] Processing captcha_images\24FJ.png...
[INFO] Processing captcha_images\24FK.png...
[INFO] Processing captcha_images\24HA.png...
[INFO] Processing captcha_images\24HK.png...
[INFO] Processing captcha_images\24XX.png...
[INFO] Processing captcha_images\24Z5.png...
[INFO] Processing captcha_images\24ZK.png...
[INFO] Processing captcha_images\2544.png...
[INFO] Processing captcha_images\2552.png...
[INFO] Processing captcha_images\256Q.png...
[INFO] Processing captcha_images\2582.png...
[INFO] Processing captcha_images\25BG.png...
[INFO] Processing captcha_images\25MZ.png...
[INFO] Processing captcha_images\25Q2.png...
[INFO] Processing captcha_images\25VJ.png...
[INFO] Processing captcha_images\265P.png...
[INFO] Processing captcha_images\267R.png...
[INFO] Processing captcha_images\267X.png...
[INFO] Processing captcha_images\26AT.png...
[INFO] Processing captcha_images\26E4.png...
[INFO] Processing captcha_images\26FL.png...
[INFO] Processing captcha_images\26GM.png...
[INFO] Processing captcha_images\26M3.png...
[INFO] Processing captcha_images\26VL.png...
[INFO] Processing captcha_images\26WU.png...
[INFO] Processing captcha_images\26WY.png...
[INFO] Processing captcha_images\273U.png...
[INFO] Processing captcha_images\276W.png...
[INFO] Processing captcha_images\2798.png...
[INFO] Processing captcha_images\2799.png...
[INFO] Processing captcha_images\27CA.png...
[INFO] Processing captcha_images\27E5.png...
[INFO] Processing captcha_images\27HB.png...
[INFO] Processing captcha_images\27HF.png...
```