**Advarisk OCR Captcha Checker**

1. **Overview:**
   1. Introduction: The “advarisk\_captcha\_solver” project is a robust Optical Character Recognition (OCR) system designed to detect and extract digits from images containing CAPTCHA-like challenges. This Flask-based web application seamlessly integrates an OCR model, allowing users to submit accurate character predictions up-to accuracy of ~96%.
      1. Key Features: Accurate digit recognition from images.
      2. User-Friendly web interface powered by Flask.
      3. Seamless integration of a pre-trained OCR model.
2. **Installations and launching the application:** 
   1. **Prerequisites:** Before getting started, ensure that you have “Python3”, and “pip” installed in your system.
   2. **Setup:** Different setup files (setup.bat and setup.sh) are provided in the project for Windows based systems and Unix based systems.
      1. Put the project directory at your specified place in your system.
      2. Open a new terminal and cd to “advarisk\_captcha\_solver/” (project directory).
   3. **Launch using setup file:**
      1. **Windows Based Systems:** for Windows system, run the following command: “setup.bat” and the complete setup will be done. The command, performs the following operations in sequence:
         1. Creating a virtual environment by name “virtenv”: **“python3 -m venv virtenv”**.
         2. Activates the virtual environment: **“call .\virtenv\Scrips\activate”**.
         3. Upgrades “pip” if it isn’t up-to date with current version: **“python3 -m pip install –upgrade pip”**.
         4. Installing the dependencies: **“pip install -r requirements.txt”**.
         5. Setting up the Flask app:
            1. **“set Flask\_APP=app.py”**
            2. **“set Flask\_APP=development”**
         6. Running the Flask App: **“Flask app”. (**Use –-debug after Flask app if you want to see the logs in the cli.)
         7. A localhost server will be started with following web address: [**http://127.0.0.1:5000/**](http://127.0.0.1:5000/). **ctrl + click** on the address to access the web application. Alternatively, you can copy and paste the address in the browser tab to access the web application.
      2. **Unix Based Systems:** Run the following command: **“setup.sh”** for setting up the application in Unix Systems.
         1. The setup commands are almost the same as of Windows system, except the environment activation command: **“source virtenv/bin/activate”**.
         2. The rest of the setup, starting Flask server and accessing the webpage is same.
   4. **Create setup manually and launch the application:** The same set of commands are used for manual setup as well. You just need to follow the commands given in Windows setup sequentially to do the complete setup and run the application. Please watch out for the virtual environment activation command for different systems.
3. **Available Endpoints:**
   1. **“POST /upload”**: Upload an image for processing.
   2. **“GET /”**: Access the main web interface.
4. **Submitting a job:**
   1. Click on **“Choose Files”** and select the image/images from your local system and click “Upload and Extract Text”. In about ~1-2 seconds, all the images will be displayed on the webpage along with the extracted digits at the bottom of each image.
   2. At the left bottom of the webpage displaying the results, a button name **“Go to Homepage”** is there which will take you to homepage to submit another job.
5. **Model Architecture:** The OCR model is built following a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) hybrid architecture. Below are the details of the model architecture:
   1. **Input Layer:**
      1. Input Shape: (190, 80, 1)
      2. Data Types: Grayscale Image (Single Channel).
   2. Convolutional Layers:
      1. Conv1:
         1. Filters: 32
         2. Kernel Size: (3, 3)
         3. Activation: ReLu
      2. **MaxPooling1:**
         1. Pool Size: (2, 2)
      3. Conv2:
         1. Filters: 64
         2. Kernel Size: (3, 3)
         3. Activation: ReLu
      4. MaxPooling2:
         1. Pool Size: (2, 2)
      5. Conv3:
         1. Filters: 128
         2. Kernel Size: (3, 3)
         3. Activation: ReLu
      6. MaxPooling3:
         1. Pool Size: (2, 2)
      7. Conv3:
         1. Filters: 64
         2. Kernel Size: (3, 3)
         3. Activation: ReLu
      8. MaxPooling4:
         1. Pool Size: (2, 2)
      9. Flattening Layer:
         1. Flattening 2D feature maps to a 1D vector.
      10. LSTM Layer:
          1. Returns Sequence: True
          2. LSTM Layer to capture sequential pattern in the data.
      11. TimeDistributed Dense Layer:
          1. Units: 10 (output layer)
          2. Activation: Softmax
          3. Applies a dense layer to each time step independently.
   3. Model Summary:
      1. Total Params: 675274
      2. Total Trainable Params: 675274
   4. Training Configuration:
      1. Optimizer: “Adam”
      2. Learning Rate: 0.001
      3. Loss Function: “categorical\_crossentropy”
      4. Metrices: “accuracy”
   5. Training Process:
      1. Dataset:
         1. Total Dataset Images: 1774
         2. Training Dataset Images: 1419 (80%)
         3. Validation Dataset Images: 177 (10%)
         4. Testing Dataset Images: 177 (10%)
      2. Training Parameters:
         1. Epochs: 100
         2. Batch Size: 16
   6. Training Results:
      1. Final Training Loss: 0.0673
      2. Final Training Accuracy: 98.24%
      3. Final Validation Loss: .1431
      4. Final Validation Accuracy: 95.88%
   7. Testing Prediction Results:
      1. Test Loss: 0.154
      2. Test Accuracy: 96.1%
6. Model Performance Metrices:
   1. Confusion Matrix:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Predicted | Actual | | | | | | | | | | |
| Labels | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 168 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 328 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 64 | 5 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 62 | 0 | 0 | 0 | 2 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 67 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 74 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 77 | 0 | 0 | 0 |
| 7 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 67 | 0 | 0 |
| 8 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 0 |
| 9 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 69 |

* 1. Accuracy Score: 98.49%
  2. Precision, Recall and F1 Score:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Labels | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Precision | 0.994 | 0.994 | 0.914 | 0.954 | 1 | 1 | 1 | 0.971 | 0.986 | 0.986 |
| Recall | 0.994 | 0.994 | 0.97 | 0.925 | 0.957 | 1 | 1 | 0.957 | 1 | 1 |
| F1 Score | 0.994 | 0.994 | 0.941 | 0.939 | 0.978 | 1 | 1 | 0.964 | 0.993 | 0.993 |

1. Result Analysis: From the analysis of metrices, model is performing very well. Looking at the confusion matric, digit “3” is predicted “2” 5 times which is the most incorrect prediction for a particular class. Apart from that, the model is doing great.
   1. Possible reason for false prediction of 3: I looked at some of the images visually and found out that some of the images had a portion of digit “3” cut at the bottom causing the model to be confused by “2”.
      1. Image ID: **100123** is predicted as 100122.
2. Conclusion: The system can predict the digits in the captcha images with a very high accuracy at a very fast rate (~300 images/second) making it a good and robust system.
3. Limitations: Although, the model is doing great on the similar kind of captcha, the model is failing with high rate if it encounters the image having high data drift.
4. Future Work: Using more images of different kind for training, the limitation can be fixed by some extent.
5. Resources:
   1. ChatGPT.
   2. <https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa>
   3. <https://medium.com/analytics-vidhya/what-is-ocr-f67c9ab218bf>
6. Good luck with the document. Have a great day 😊.

Rahul Maurya

25/12/2023