



Check OCR project implementation

27.09.2022

Aditya Dixit
Sruthi Sreekumar
Bhanu M prakash

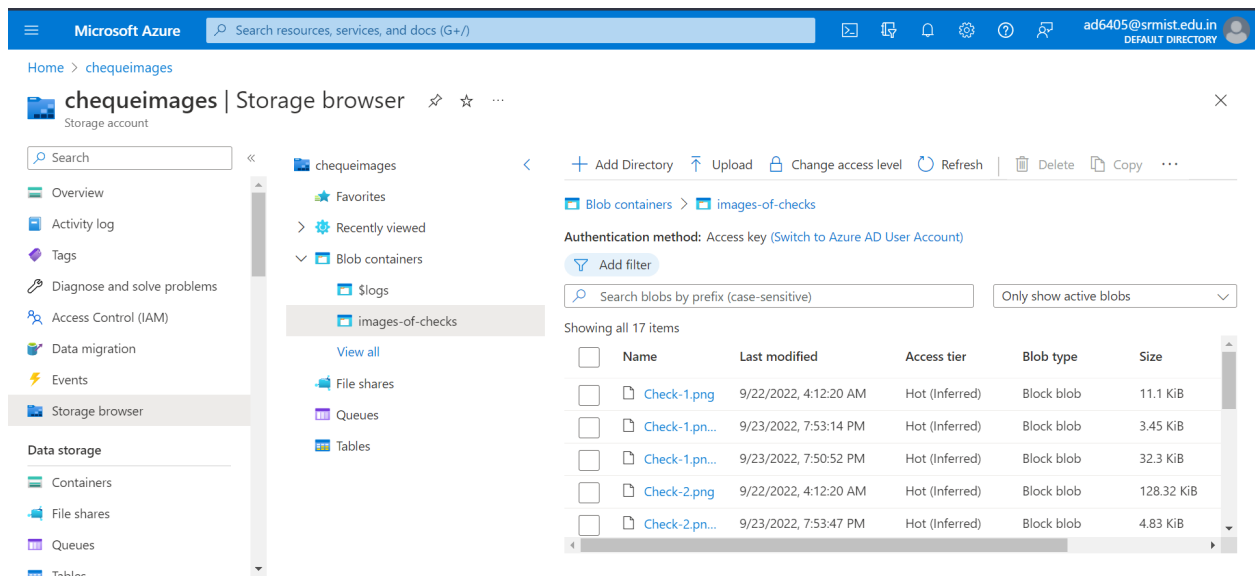
- 1) Create a Microsoft Azure account and subscribe to Form Recognizer service provided by Microsoft Azure.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Microsoft Azure logo, a search bar, and user information (ad6405@srmist.edu.in). The main content area displays the 'CHECK-FORM-RECOGN' resource page for the 'Form recognizer' service. The left sidebar shows navigation options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main content area includes a 'Help us improve Form Recognizer' survey banner, an 'Essentials' section with details like Resource group (CHECK-OCR), Status (Active), Location (Central India), Subscription (Azure subscription 1), and Subscription ID (6dbe80d3-01ac-42cd-9004-34e0d291a6ec). It also shows API type (Form Recognizer), Pricing tier (Free), Endpoint (https://check-form-recogn.cognitiveservices.azure.com/), and a link to manage keys. A 'JSON View' link is visible in the top right corner.

You will get a dashboard like this. From this dashboard you can get your endpoints to provide in the project code.

- 2) Create a Storage blob container provided by azure services and upload the check images to train the custom model on your dataset.

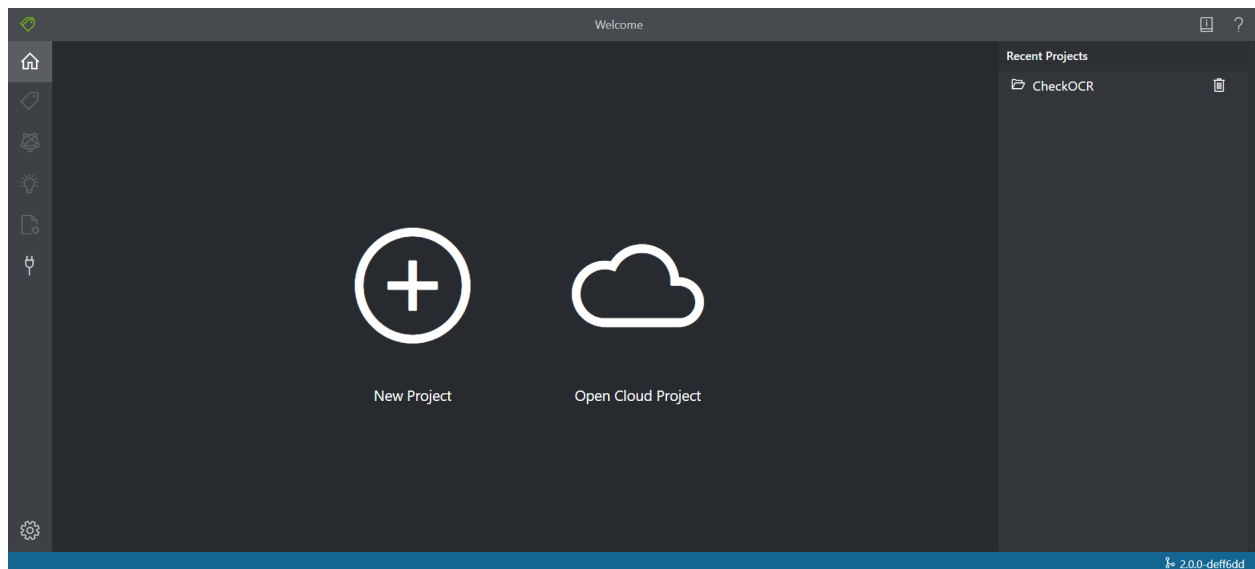
The screenshot shows the Microsoft Azure portal interface for the 'chequeimages' Storage account. The top navigation bar includes the Microsoft Azure logo, a search bar, and user information (ad6405@srmist.edu.in). The main content area displays the 'chequeimages' resource page for the 'Storage account' service. The left sidebar shows navigation options like Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, and Data storage. The main content area includes an 'Essentials' section with details like Resource group (CHECK-OCR), Location (Central India), Primary/Secondary Location (Primary: Central India, Secondary: South...), Subscription (Azure subscription 1), Subscription ID (6dbe80d3-01ac-42cd-9004-34e0d291a6ec), and Disk state (Primary: Available, Secondary: Available). It also shows Performance (Standard), Replication (Read-access geo-redundant storage (RA-GRS)), Account kind (StorageV2 (general purpose v2)), Provisioning state (Succeeded), and Created (9/22/2022, 4:08:55 AM). A 'JSON View' link is visible in the top right corner. Below the Essentials section, there are tabs for Properties, Monitoring, Capabilities (7), Recommendations, Tutorials, and Developer Tools. The 'Properties' tab is active, showing details for Blob service (Hierarchical namespace: Disabled, Default access tier: Hot, Blob public access: Enabled) and Security (Require secure transfer for REST API operations: Enabled, Storage account key access: Enabled).

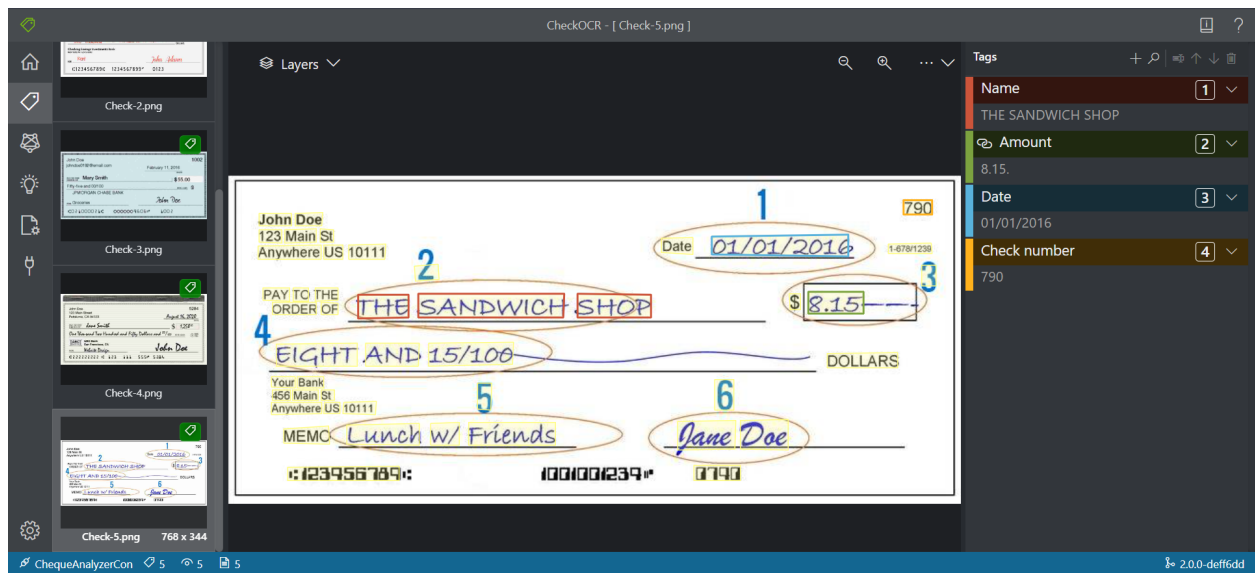


You will get a dashboard similar to this.

Now to train your custom model go to [fott.azure](https://fott.azure.com) and make a new project.

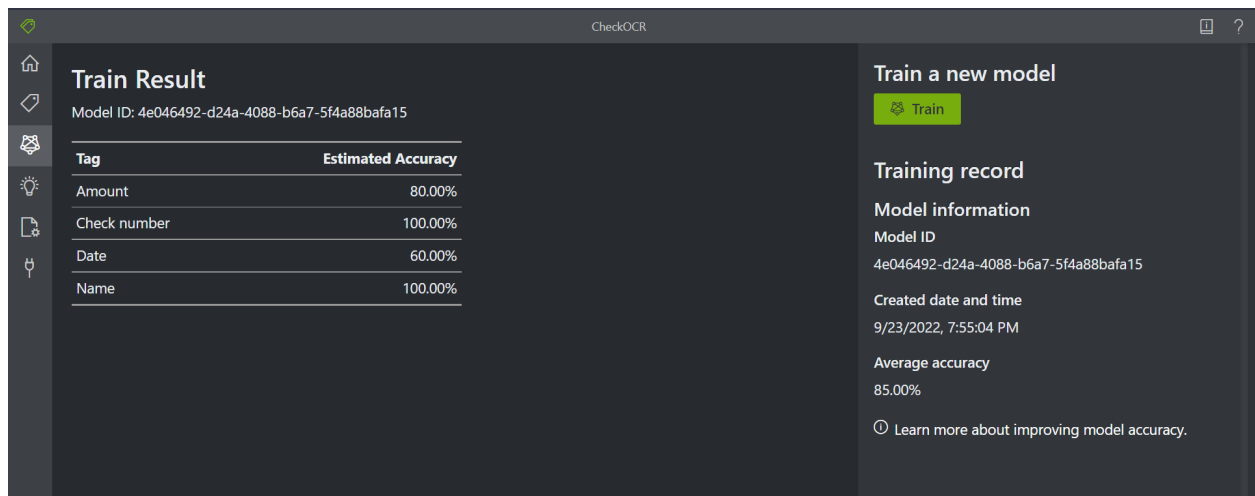
Connect with your Storage account of azure and start labeling the data.





After labeling all the images, train your model in the training tab.

After training the model you will get a model-id which you use in the API call code to execute.



3) Install all the required libraries for the code like streamlit, azure etc.

```
from email.mime import image
from tabnanny import check
import streamlit as st
import json
import time
```

```

from requests import get, post

from azure.core.credentials import AzureKeyCredential

from azure.ai.formrecognizer import DocumentAnalysisClient

import matplotlib.image as mpimg

from matplotlib import pyplot as plt

from PIL import Image

import numpy as np

import PIL

import os

import cv2

#####

def my_func(image):

    # Endpoint URL

    endpoint = r"https://check-form-recogn.cognitiveservices.azure.com/"

    apim_key = "cae738e5045c4533b4a172f0d80ea137"

    model_id = "4e046492-d24a-4088-b6a7-5f4a88bafa15"

    post_url = endpoint + "/formrecognizer/v2.1/custom/models/%s/analyze" % model_id

    #source = r"abc_bank.png"

    params = {

        "includeTextDetails": True

    }

    headers = {

        # Request headers

        'Content-Type': 'image/png',

        'Ocp-Apim-Subscription-Key': apim_key,

    }

    # with open(source, "rb") as f:

    #     data_bytes = f.read()

    try:

        resp = post(url = post_url, data = image, headers = headers, params = params)

        if resp.status_code != 202:

            print("POST analyze failed:\n%s" % json.dumps(resp.json()))

            quit()

        print("POST analyze succeeded:\n%s" % resp.headers)

        get_url = resp.headers["operation-location"]

    except Exception as e:

```

```

        print("POST analyze failed:\n%s" % str(e))

        quit()

#####

n_tries = 3
n_try = 0
wait_sec = 2
max_wait_sec = 60
while n_try < n_tries:

    try:

        resp = get(url = get_url, headers = {"Ocp-Apim-Subscription-Key": apim_key})

        resp_json = resp.json()

        if resp.status_code != 200:

            print("GET analyze results failed:\n%s" % json.dumps(resp_json))

            quit()

        status = resp_json["status"]

        if status == "succeeded":

            print("Analysis succeeded:\n%s" % json.dumps(resp_json))

            #quit()

        if status == "failed":

            print("Analysis failed:\n%s" % json.dumps(resp_json))

            quit()

        # Analysis still running. Wait and retry.

        time.sleep(wait_sec)

        n_try += 1

        wait_sec = min(2*wait_sec, max_wait_sec)

    except Exception as e:

        msg = "GET analyze results failed:\n%s" % str(e)

        print(msg)

        #quit()

#####

data=resp.text
parse_data=json.loads(data)

#####

name=parse_data['analyzeResult']['documentResults'][0]['fields']['Name']['valueString']
amount=parse_data['analyzeResult']['documentResults'][0]['fields']['Amount']['valueNumber']

```

```

date=parse_data['analyzeResult']['documentResults'][0]['fields']['Date']['valueString']

checkNumber=parse_data['analyzeResult']['documentResults'][0]['fields']['Check number']['valueString']

# st.write("Name:- "+str(name))

# st.write("Amount:- "+str(amount))

# st.write("Date:- "+str(date))

# st.write("Check Number:- "+str(checkNumber))

newDate=""

for ch in date:

    if ch.isalpha()==0:

        newDate+=str(ch)

st.text_input(label="Name", value=name)

st.text_input(label="Amount", value=amount)

st.text_input(label="Check Number", value=checkNumber)

st.text_input(label="Date", value=date)

#####
####

def readModel():

    image=plt.imread("test.png")

    crop_position= image.shape[0] // 5

    half_image = image[image.shape[0] - crop_position,: ]

    fin_img=Image.fromarray((half_image* 255).astype(np. uint8))

    fin_img.save('cropped_img.png')

    endpoint = "https://check-form-recogn.cognitiveservices.azure.com/"

    key ="cae738e5045c4533b4a172f0d80ea137"

    document_analysis_client = DocumentAnalysisClient(

        endpoint=endpoint, credential=AzureKeyCredential(key)

    )

    # Make sure your document's type is included in the list of document types the custom model can analyze

    with open(r'cropped_img.png', "rb") as f:

        poller = document_analysis_client.begin_analyze_document(

```

```

        model_id="prebuilt-read", document=f
    )

result = poller.result()

final_String=""

for idx, document in enumerate(result.documents):

    print("-----Analyzing document #{0}-----".format(idx + 1))

    print("Document has type {}".format(document.doc_type))

    print("Document has confidence {}".format(document.confidence))

    print("Document was analyzed by model with ID {}".format(result.model_id))

    for name, field in document.fields.items():

        field_value = field.value if field.value else field.content

        print(".....found field of type '{}' with value '{}' and with confidence {}".format(field.value_type,
field_value, field.confidence))

# iterate over tables, lines, and selection marks on each page
for page in result.pages:

    #print("\nLines found on page {}".format(page.page_number))

    for line in page.lines:

        #print("...Line '{}'".format(line.content))

        final_String+=line.content

    #for word in page.words:

    #    print( "...Word '{}' has a confidence of {}".format(word.content, word.confidence))

    for selection_mark in page.selection_marks:

        print(

            "...Selection mark is '{}' and has a confidence of {}".format(

                selection_mark.state, selection_mark.confidence

            )

        )

)

for i, table in enumerate(result.tables):

    print("\nTable {} can be found on page:".format(i + 1))

    for region in table.bounding_regions:

        print("...{}".format(i + 1, region.page_number))

    for cell in table.cells:

        print(

            "...Cell[{}][{}] has content {}".format(

                cell.row_index, cell.column_index, cell.content

            )

        )

)

```



```

print("-----")

if final_String.find("#") == 0:

    Transit_number=final_String[final_String.find("#")+1:final_String.find(".")]

    Bank_Code=final_String[final_String.find(".") + 1:final_String.find("#", final_String.find("#")+1)]

Designation_Number=final_String[final_String.find("#", final_String.find("#")+1) + 1:final_String.find(".", final_String.find("#")+1)]

Account_number=final_String[final_String.find(".", final_String.find(".") + 1) + 1:final_String.find("#", final_String.find("#")+1)]

    # st.write("Transit Number:- " + str(Transit_number))

    # st.write("Bank Code:- " + str(Bank_Code))

    # st.write("Designation Number:- " + str(Designation_Number))

    # st.write("Account Number:- " + str(Account_number))

    st.text_input(label="Transit Number:- ", value=Transit_number)

    st.text_input(label="Bank Code", value=Bank_Code)

    st.text_input(label="Designation Number", value=Designation_Number)

    st.text_input(label="MICR", value=Account_number)

else:

    Routing_number=final_String[final_String.find("#")+1:final_String.find("#", final_String.find("#")+1)]

    Account_number=final_String[final_String.find("#", final_String.find("#")+1) + 1:final_String.find("#")]

    accnum=""

    for ch in Account_number:

        if ch.isnumeric():

            accnum+=str(ch)

    # st.write("Routing Number:- " + str(Routing_number))

    # st.write("Account Number:- " + str(accnum))

    st.text_input(label="Routing Number", value=Routing_number)

    st.text_input(label="Account Number", value=accnum)

#####

st.title("Cheque Recognition Software")

st.header("Upload your check here:-")

```

```

uploaded_file = st.file_uploader("Choose a image file",type='png')

if uploaded_file is not None:

    image = uploaded_file.read()

    img = st.image(image, caption='Your Bank check', use_column_width=True)

if uploaded_file is not None:

    with open(os.path.join("C:/Users/Aditya Dixit/Desktop/CHECK-OCR/Streamlit Tutorial","test.png"),"wb") as f:

        f.write(uploaded_file.getbuffer())

if uploaded_file is not None:

    my_func(image)

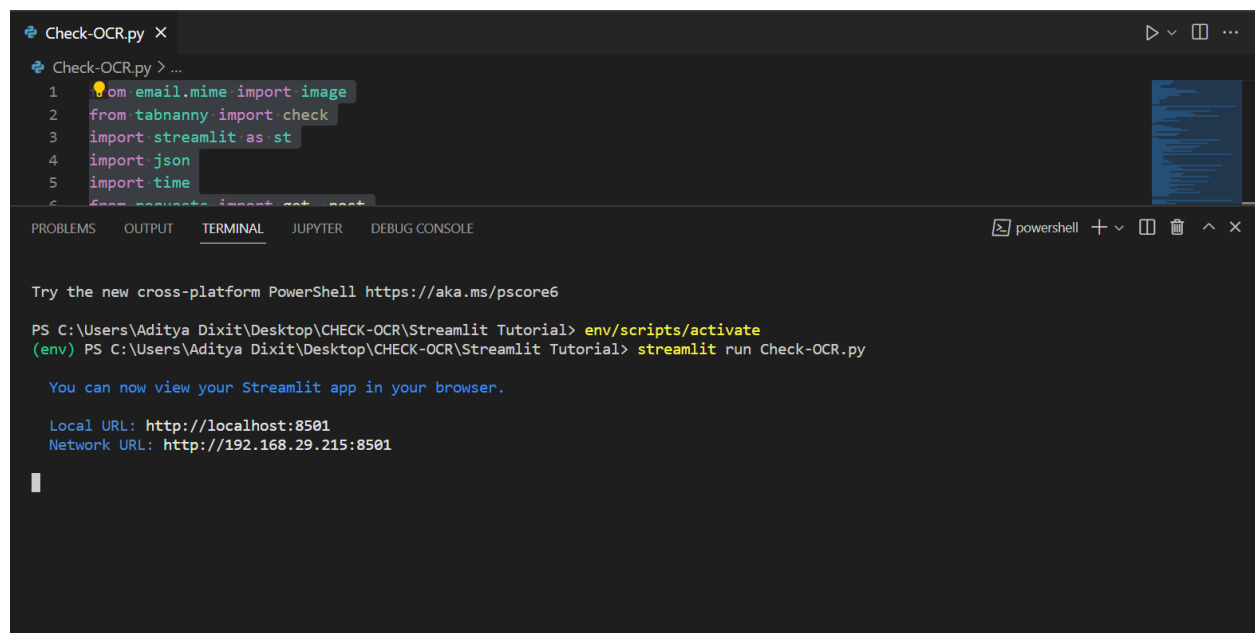
if uploaded_file is not None:

    readModel()

#####

```

4) Now execute the code with command streamlit run Check-OCR.py



```

Check-OCR.py X
Check-OCR.py > ...
1 from email.mime import image
2 from tabnanny import check
3 import streamlit as st
4 import json
5 import time
6 from requests import get, post

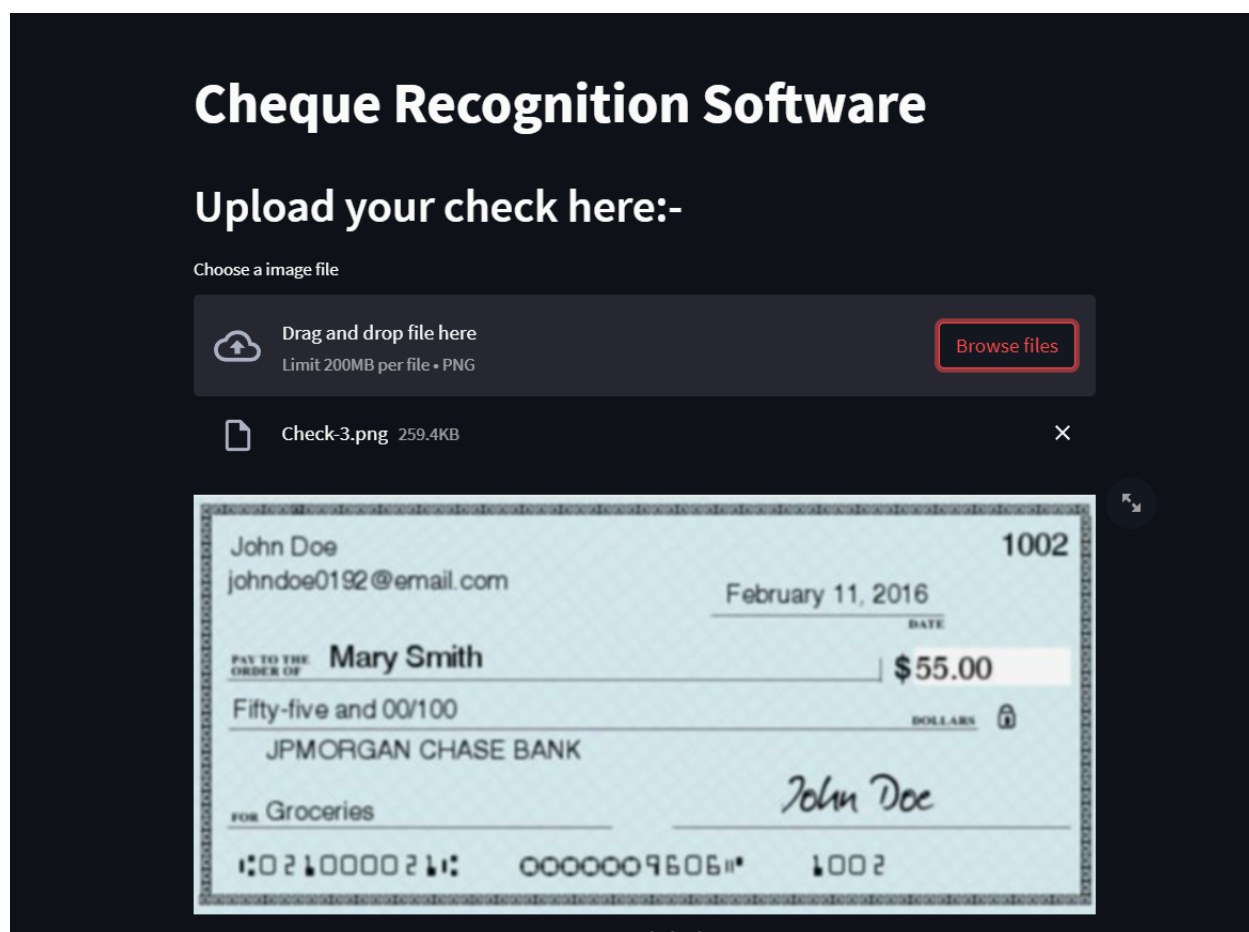
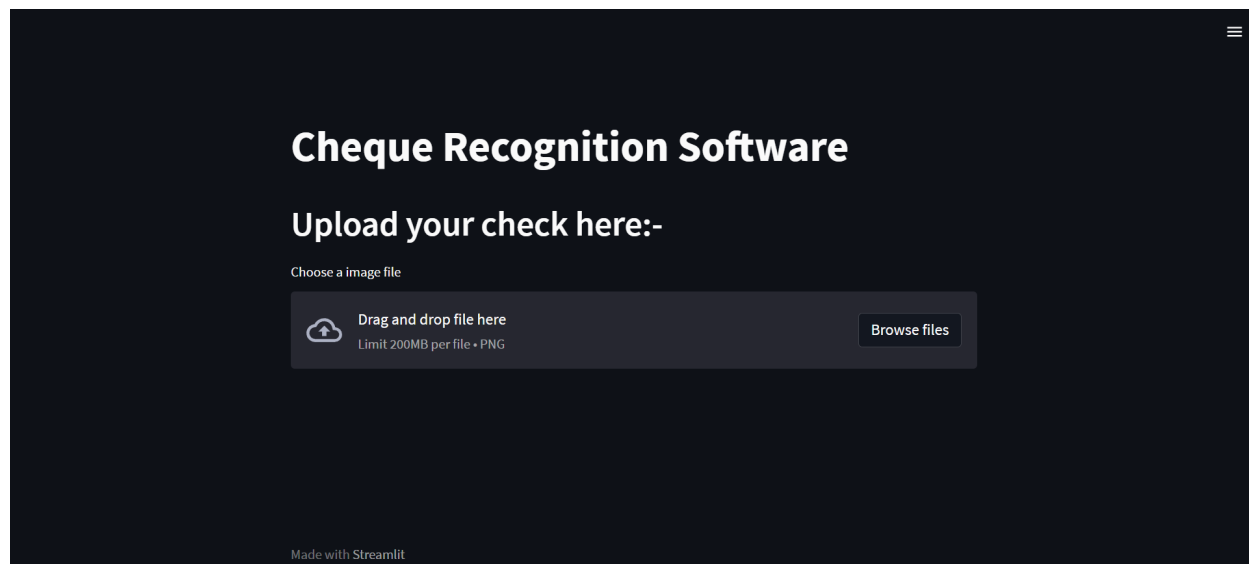
PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\Aditya Dixit\Desktop\CHECK-OCR\Streamlit Tutorial> env/scripts/activate
(env) PS C:\Users\Aditya Dixit\Desktop\CHECK-OCR\Streamlit Tutorial> streamlit run Check-OCR.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.29.215:8501

```

OUTPUT:-



Your Bank check

Name

Mary Smith

Amount

55.0

Check Number

1002

Date

February 11, 2016

Routing Number

021000021

Account Number

0000009606