Photo Classification App - Group 26

CSE-535 Mobile Computing - Project 2

Bhavani Mahalakshmi Gowri Sankar
Computer Science
ARIZONA STATE UNIVERSITY
Tempe, USA
bgowrisa@asu.edu

Mahidher Duraisamy Krishnan
Computer Science
ARIZONA STATE UNIVERSITY
Tempe, USA
mdurais1@asu.edu

Bitaan Chakrabarti
Computer Science
ARIZONA STATE UNIVERSITY
Tempe, USA
bchakra7@asu.edu

Preetham Sana
Computer Science
ARIZONA STATE UNIVERSITY
Tempe, USA
psana@asu.edu

Avinash Senthil Kumaran
Computer Science
ARIZONA STATE UNIVERSITY
Tempe, USA
asenth10@asu.edu

I. INDEX

II. ABSTRACT

The main of the project is to create an application to allow users to capture an image by accessing the camera application and uploading the image to the server and training a deep-learning framework from basics to classify the handwritten digits and tell what number that is. Then those images will be saved in their respective folder.

III. TECHNOLOGIES USED

Client Side: Expo, base 64 encoding format for images, OkHttp client, React native, Node js.

Server Side: Python3, Flask module, base 64 decoding format for images, Opencv, Tensorflow, keras.

IV. SYSTEM REQUIREMENTS

1. Mobile Phone with IOS or an emulator.
2. Expo application on Laptop.
3. Minimum 8GB RAM on the laptop to ensure that Expo runs perfectly.
4. Python3, pip, flask installed on the system for executing the server side code.

V. IMPLEMENTATION

Steps for implanting project 2:

1. There are two different parts. They are the Mobile Application and Server side of the mobile application.

2. In the Mobile application we need to develop different user interfaces for different pages like Welcome Screen, Camera Screen, and Upload Screen.

3. Firstly, on the welcome screen we have a button to access the camera to take pictures and upload them to the server.

4. Then it directs to the camera page to take the picture by using the TAKE PICTURE button.

5. Then the image will be uploaded to the server and the screen shows as “Picture Uploaded Successfully!”.

6. ***For the Backend*** – For Training, the server has two parts. They are Classification and Application.

7. **In the Classification**,

a) First, we receive the image from the application, and it will be converted to the grayscale format.
b) Then, we use libraries such as NumPy’s expand_dims and cv2’s resize to make the image shape as (1,28,28,1).

c) We normalize this image by dividing it by 255.

d) We send this image as input to our pre-trained model and get the output from it.

e) Finally, we return it to our application

8. **In the Application**:

a) First, we install flask and create a flask application with the host as localhost.

b) Then create a REST API with a POST method that can take in a JSON body as part of the request parameters. The two JSON keys that we pass for this project are 'image data' and 'category'.

c) We have created two routes: Home route ('/') and 'save image route'. The Home route is a route for the initial start-up which shows us the "Hello world" message. The 'save image' route saves the image with the image name in this location when we upload an image from the application.

d) The image is passed as a base64 encoded string between the client and the server. So we have to decode the base64 image data that is passed in the body of the request.

e) We save the converted image data as an image file of the required format.

f) The storage path will be defined with the result obtained from the pre-trained machine learning model. Once the storage path is defined, the image is then saved to that path.

g) To execute we run the python file and the application that is hosted locally.

VI. Technical Approach to the Problem

A. *Client side*

For project 2 the application has several pages; the first one is used to grant access to the camera. When we click the button labeled "TAKE PICTURE," the camera application launches, allowing us to take images. Then it will take you to the following screen, where you can upload the photographs to a local server. The image will then be saved on the server. Then the images will be classified based on the model and give the result. By importing the react-native and react modules, we created the application utilizing the three screens "*welcome screen, upload screen, and camera screen*."

Welcome Screen: The TAKE PICTURE button is on the application's home page. The camera program is launched when users click the button. To access the camera, we utilized the "*expo-camera*" module.

Camera Screen: Users can take a picture by clicking the "round shaped button," or "take picture" button, while the camera application is activated. The image is captured, the information is sent to the following page, and the image appears there. The "*upload screen*" is the following page.

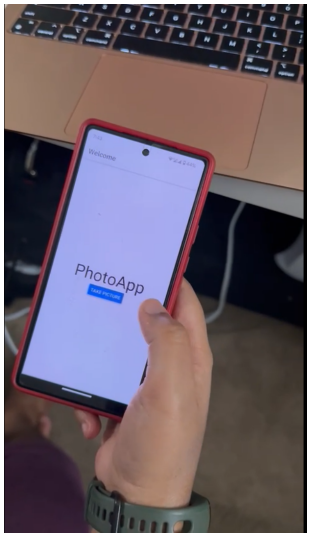

Fig. 1. Welcome Screen

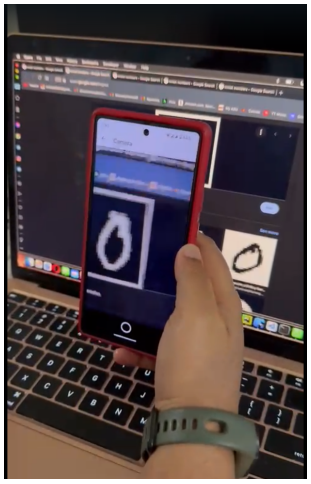

Fig. 2. Camera Screen

Upload Screen: When a user clicks the "TAKE PICTURE" button the image is added to that category on the local server. Then a push notification stating, "Picture Uploaded Successfully!" shows on the screen. To upload an image to the local server, the "upload Handler" method will be invoked. We utilized the "*react-native-dropdown-picker*" module for the drop-down.

Finally, after classification of the image in the upload screen, we will be able to see the number which was in the image that we have uploaded as "*The number is* _" in a dialogue box.

Additionally, the useState function was employed. You can incorporate state variables in functional components by using the "useState" Hook. In exchange for an initial state parameter, this method returns a variable containing the value of the

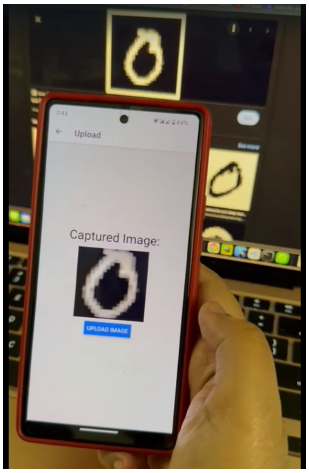

Fig. 3. Upload Screen

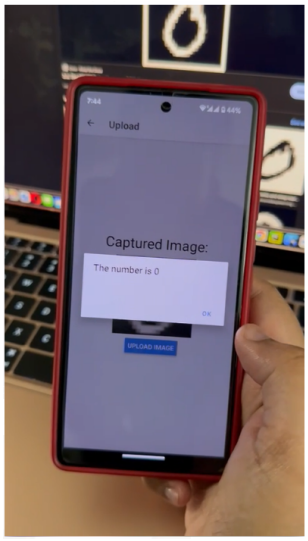

Fig. 4. Screen showing the value that is classified

current state (which need not be the starting state) and another function to update this value.

B. Server Side:

Training the Model:

1. First, we load the MNIST dataset, and then split it to training and test datasets.
2. After that, we check if the dataset is balanced (whether each category has similar number of images).
3. Then, we normalize both training and test datasets.
4. Since MNIST dataset is grayscale, we add an extra channel to the elements in the dataset. So, the images will now have dimensions of (28,28,1). Thus, the preprocessing is done.
5. Then we create a simple deep learning model to classify image to 10 categories.

6. We designed a Convolutional Neural network of 9 layers to perform the classification. The image is shown below:

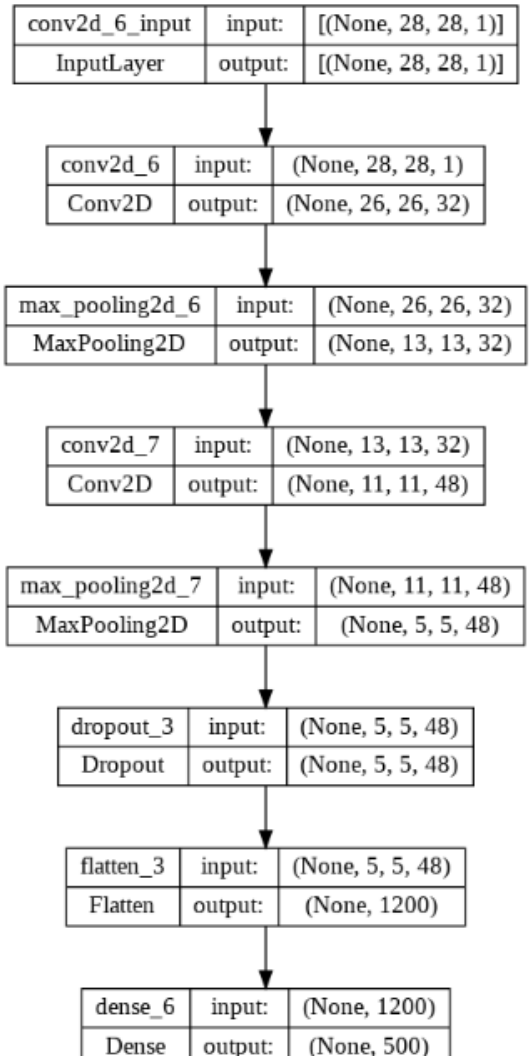

Fig. 5. Convolutional Neural network

7.We train the model using Adam optimizer, with 'categorical_crossentropy' loss.

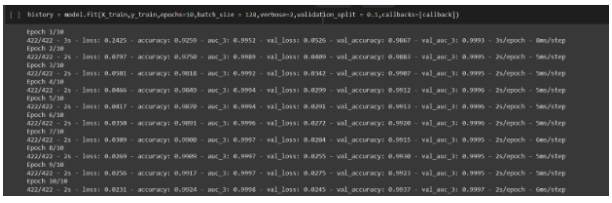

Fig. 6. Training the model

8. The plot of **loss vs epochs** is shown below:

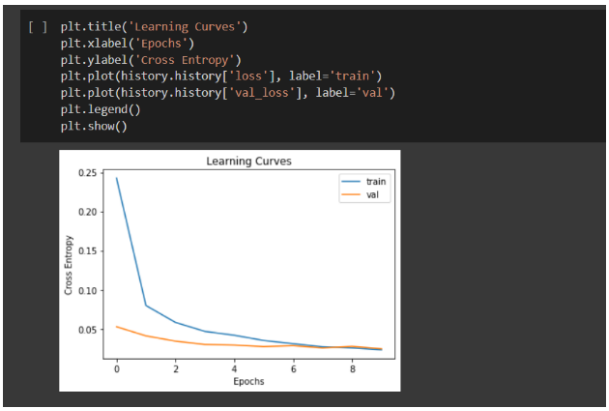

Fig. 7. Loss vs Epochs

9. We achieved a test **accuracy** of 99.37

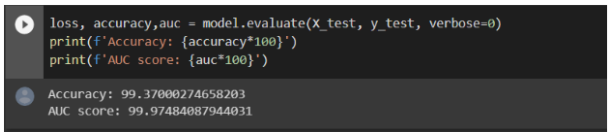

Fig. 8. Accuracy

10. Since the model takes a lot of time to load on the server side, we created a python library for the pre trained model. We import this library on our server side to do classification of the images. **The library can be found at: https://test.pypi.org/project/mnist-mentiumClassifier/**

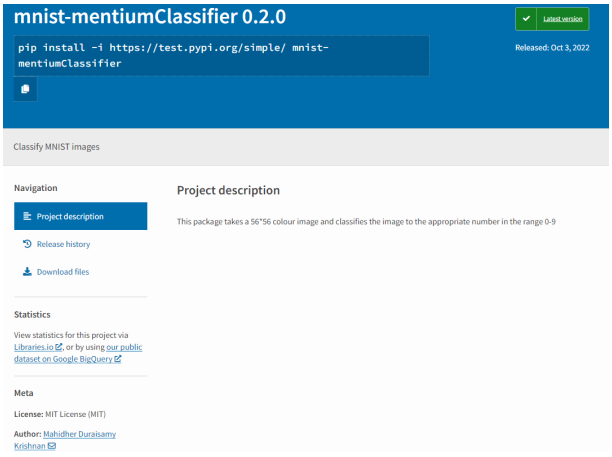

Fig. 9. Library

11. We have achieved the task of classified images are being stored in the respective folders. For the first time we took a picture of "**zero**". So, a folder has been created with the name as "**0**". Then we took a picture of "**four**". Then, again a folder has been created with the name as "**4**". You can see the all of this in the below image.

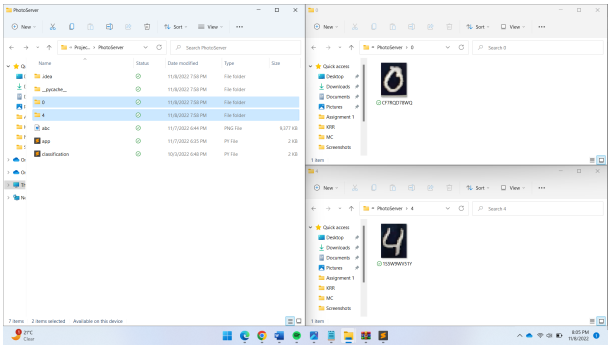

Fig. 10. Storing the images in respective folders

There are 2 routes we have set up for our app that are:

1. The Home route ('/') - which shows the initial start-up message
2. The 'save image' route - which saves the image with a random name in this location when the user uploads an image from the app. The image is then passed to our pre-trained model and classification result is obtained. Based on the classification result, a folder is created in the result's name and the image is saved in that folder.

VII. Conclusion

We have developed an android application, used to capture images and upload them to the server. We have used a machine learning pipeline to train the model and classify the handwritten digit images and store them in their respective folders. To accomplish all these, we have used Python3, React native, Flask, OpenCV, TensorFlow, and Keras. So, users can take a picture and upload it to the server and be able to see what handwritten digit image they have captured and uploaded. This final application, which we believe should satisfy the needs of the project criteria, thus serves as our conclusion.

VIII. Video Link

Video Link

References

[1] Mnist-mentiumclassifier. PyPI. (n.d.). Retrieved November 8, 2022, from https://test.pypi.org/project/mnist-mentiumClassifier/
[2] Tensorflow. TensorFlow. (n.d.). Retrieved November 8, 2022, from https://www.tensorflow.org/
[3] Mollick, S. A. (2019, April 26). Convert image to base64 string in Python. CodeSpeedy. Retrieved September 26, 2022, from https://www.codespeedy.com/convert-image-to-base64-string-in-python/
[4] Baeldung. (2022, July 5). A guide to OkHttp. Baeldung. Retrieved September 26, 2022, from https://www.baeldung.com/guide-to-okhttp

[5] Flask. PyPI. (n.d.). Retrieved September 26, 2022, from https://pypi.org/project/Flask/
[6] Ali, A.-R. (2022, April 11). Base64 encoding and decoding using Python. Code Envato Tuts+. Retrieved September 26, 2022, from https://code.tutsplus.com/tutorials/base64-encoding-and-decoding-using-python–cms-25588
[7] Fetch. The Modern JavaScript Tutorial. (2022, April 14). Retrieved September 26, 2022, from https://javascript.info/fetch