CSE 535: Mobile Computing

Assignment 2

Group Members: Radhika Kulkarni, Sandhya Balaji, Ravali Badri, Venkata Sai Gowtham Koniki, Viranchi Jayeshbhai Badheka

Overview

This project aimed to develop a mobile application that can capture an image of a handwritten digit, perform recognition and return the recognized digit as the response to the user. The application utilizes the back camera of the mobile phone to capture the image. This is similar to what we implemented in Project 1. Then, the image is sent to the server for recognition. The server uses a pretrained model to perform the recognition and the returned label from the image is used to place the image in the respective folder. The mobile application has been developed using Android and the server-side code has been developed using Flask. For training the model, we have used the MNIST handwritten digits dataset. The tech stack used for model training is Python, Tensorflow, and Keras.

Application Implementation

To make the app run without any permission issues, we added the required permissions for accessing the camera, storage and backend communication. These permissions are added to the Android Manifest file. We have used an intent action type of MediaStore.ACTION_IMAGE_CAPTURE to launch an existing camera application to capture the image. For communicating with the backend, we have used OkHttpClient. To create a request, the URL is the machine's IP address; we run the application using the POST method. POST call consists of a payload containing the image. The response from the server, which contains the recognized digit, is displayed.

Server Implementation

We have used Flask for server development. In the python script, @app.route("/uploader") is associated with the handle_request() method, which is being called from the Android application. app.run(host=" 0.0.0.0") will host the server on the machine's IP address. We have used port 10001 for the application. The pre-trained model weights are stored in the file mnist.h5, which the server uses. Finally, the method predict_digit() is called, which loads the model weights and performs pre-processing on the captured image before feeding it into the model.

Model Development and Workflow

Dataset:

We used a CNN model to train the MNIST handwritten digits dataset. We used Keras to build the CNN model and also to work on the data preprocessing. The MNIST dataset consists of 60000 training images and 10000 test images. At first, we worked on the data preprocessing part of these images. This involved reshaping the images from 28x28 to 28x28x1. 1 here signifies that the image is grayscale.

Model Architecture:

We have used the Sequential model type to build our model. The simplest technique to build a model in Keras is sequential. It enables layer-by-layer model construction. To add additional layers, we used the add() function. Conv2D layers make up the first two layers. These layers of convolution will handle our input photos, which are represented as 2D matrices. In this case, the first layer has 32 nodes and the second layer has 64 nodes. The filter matrix for our convolution has a kernel size. Consequently, a 3x3 filter matrix will be employed if the kernel size is 3. Rectified Linear Activation, also known as ReLU, is the activation function employed in this case for the top two layers. If the input is a negative integer, this function outputs 0, and if the input is a positive number, it outputs the same input. The flow of the model is as follows:

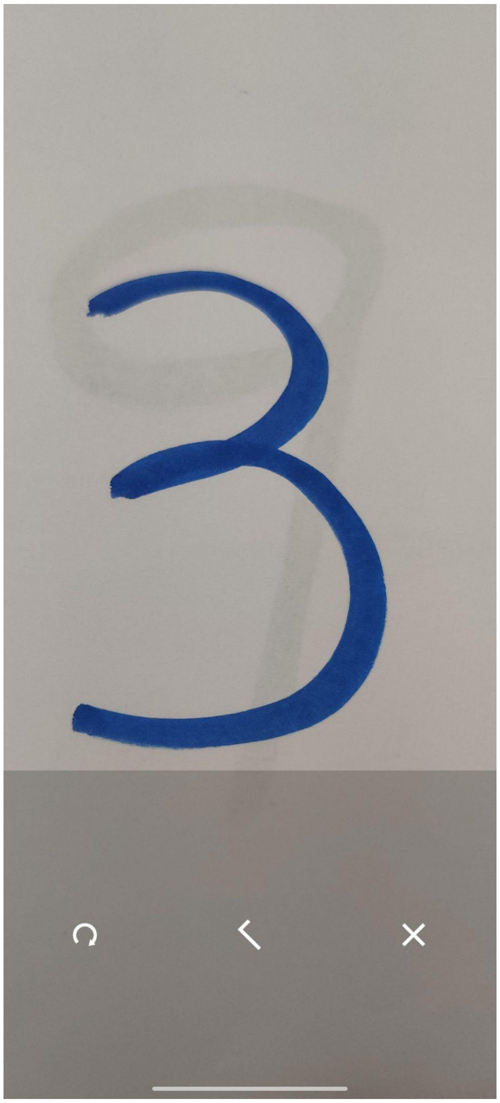
- The first layer takes in the input image (28x28x1).
- There is a "Flatten" layer sandwiched between the Conv2D layers and the dense layer. Between dense and convolutional layers, flatten acts as a link.
- “Dense” layer is used for the output layer of the model with “Softmax” activation function which outputs probability values. The model predicts the one with the highest probability.

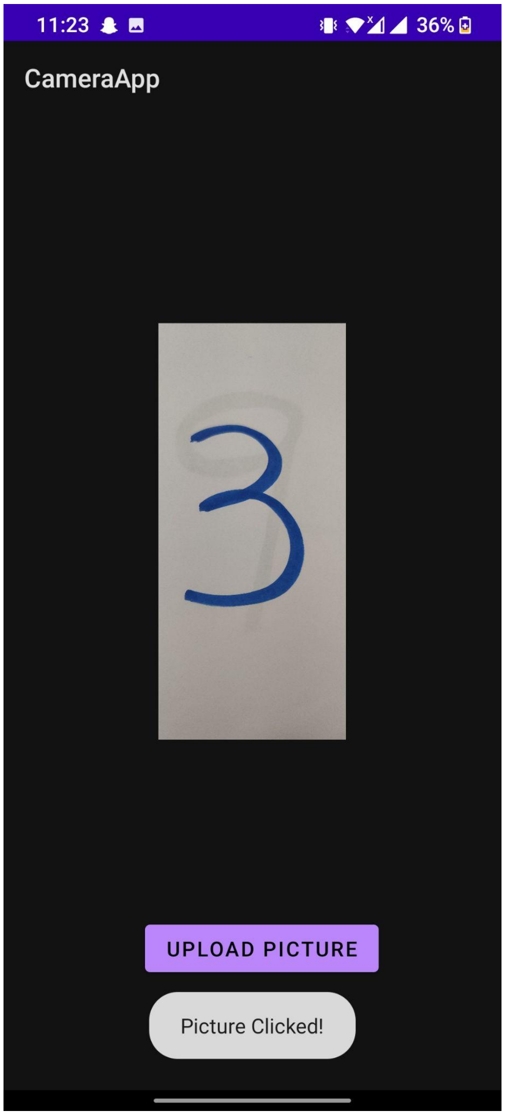
In order to compile the model, we used “adam” optimizer with “categorical_crossentropy” loss function.

Prediction of Real Images

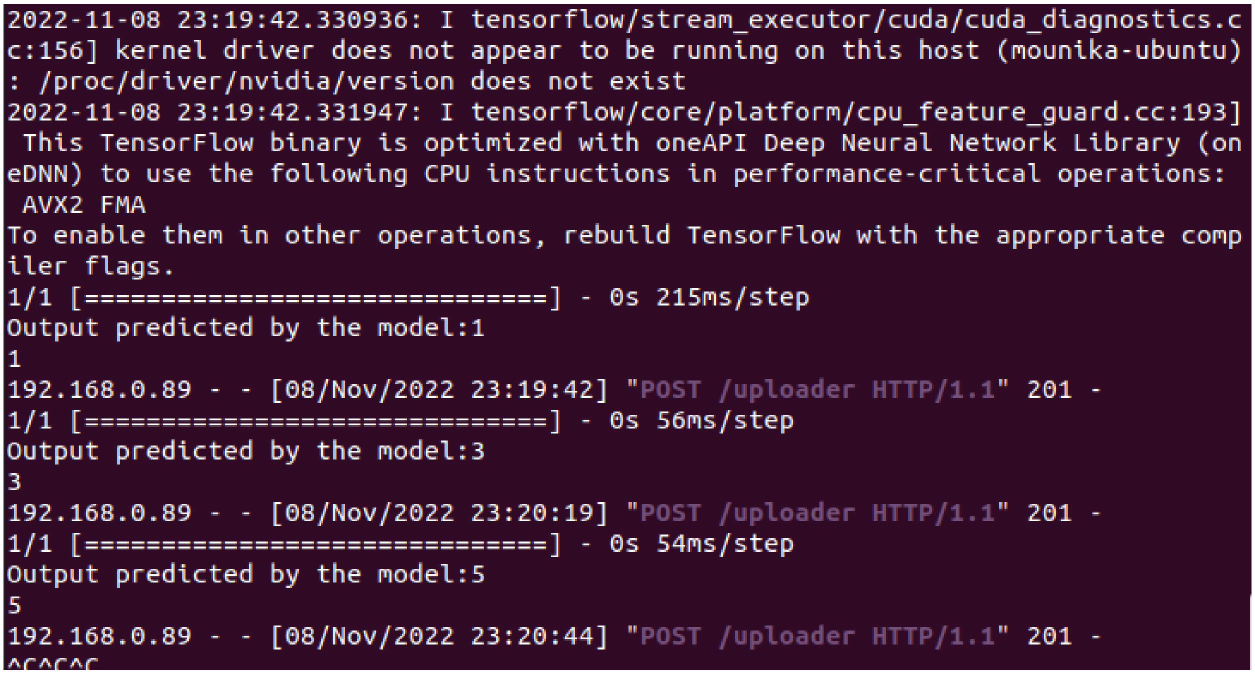
In order to predict digits from real life images - we first made the picture grayscale. Grayscale image is then binarized (thresholded) so that just the digits are white and the rest is black. The contours are found in the image using the binarized version. Here, the image's contours will give us each digit individually. We now have the numbers. However, it needs to be further altered such that it resembles the images in the training dataset much more closely. We may infer that the image must have the shape (28, 28), that the digit should be white, that the backdrop should be black, and that the digit should not be stretched to the image's edges. Instead, there should be a 5 pixel zone of black around the digit on each of its four sides. In order to change our image, we will now resize it to (18,18). Next, a 5 pixel-wide padding of zeros (in black) will be added in both directions (top, bottom, left, right). The final padded image will therefore be of the desired size (5+18+5, 5+18+5) = (28, 28).

Screenshots of the Working App

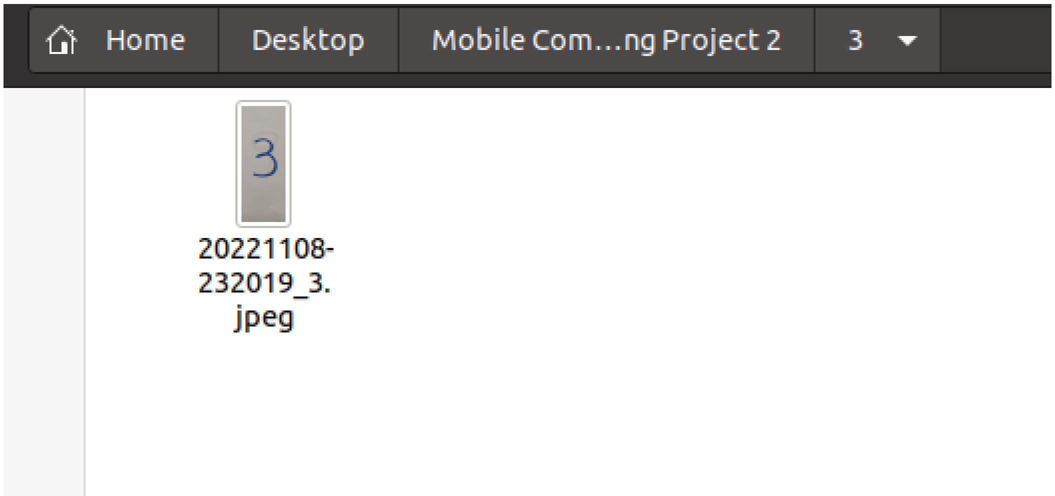




Picture Clicked!



```
2022-11-08 23:19:42.330936: I tensorflow/stream_executor/cuda/cuda_diagnostics.c
c:156] kernel driver does not appear to be running on this host (mounika-ubuntu)
: /proc/driver/nvidia/version does not exist
2022-11-08 23:19:42.331947: I tensorflow/core/platform/cpu_feature_guard.cc:193]
 This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (on
eDNN) to use the following CPU instructions in performance-critical operations:
 AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate comp
iler flags.
1/1 [==============================] - 0s 215ms/step
Output predicted by the model:1
1
192.168.0.89 - - [08/Nov/2022 23:19:42] "POST /uploader HTTP/1.1" 201 -
1/1 [==============================] - 0s 56ms/step
Output predicted by the model:3
3
192.168.0.89 - - [08/Nov/2022 23:20:19] "POST /uploader HTTP/1.1" 201 -
1/1 [==============================] - 0s 54ms/step
Output predicted by the model:5
5
192.168.0.89 - - [08/Nov/2022 23:20:44] "POST /uploader HTTP/1.1" 201 -
^C^C^C
```



The above screenshots show the workflow of the application and how the mobile application classifies a handwritten image of the digit ‘3’.