CSE 535: Mobile Computing
Project 2 Report - DigitClassifier

Keenan Rahman
1222316745
krahman2@asu.edu

Ashutosh Garg
1222340795
agarg69@asu.edu

Akshay Malhotra
1220233742
amalho23@asu.edu

Himanshu Pahadia
1222425139
hpahadia@asu.edu

Ninad Bharat Gund
1222336947
ngund@asu.edu

Responsbilities

Name	Responsibility
Ashutosh Garg	Developed source code related to camera component (CameraX). Ensuring that data is stored safely based on control flow.
Akshay Malhotra	Developed Neural Network for image classification. Used MNIST dataset for CNN based training and testing
Himanshu Pahadia	Developed source code related to backend-server. Contributed with app-side development to take images and send to server.
Keenan Rahman	Responsible for drafting the report and ensuring it meets submission guidelines. Responsible for recording demo video.
Ninad Bharat Gund	Responsible for increasing the accuracy of the deep neural network. Responsible for preprocessing of the captured image. Ensured that the digit pictures are identified with high accuracy.

Git URL

All source code is mentioned in the below-mentioned GIT URL. It contains application source code (android app) and server source code. Please refer to the README for more details.
GIT URL: https://github.com/pahadiahimanshu/Flaskdroid/tree/assignment2

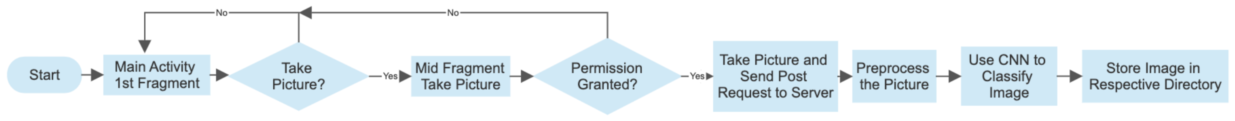
Technologies & Data Used

1. Flask - For developing a backend application server that accepts REST requests.
2. Postman - For testing and integration purposes.
3. Android Studio - Software used for developing the Android Application.
4. Kotlin - The language used for developing the Android Application.
5. Volley - The HTTP library used for sending REST requests to the backend application server.
6. CameraX - A camera-based library used in the android application.
7. Custom Neural Network - A neural network to classify the uploaded image w.r.t digit.
8. MNIST Dataset - To classify and train the neural network.

Introduction

In this project, we were responsible for developing an Android Application that lets users take a picture of digits and upload it on the server. The server uses a neural network to identify the digit and upload it in the respective directory in the server. There are two components to the project - Android Application and Flask Backend Server.

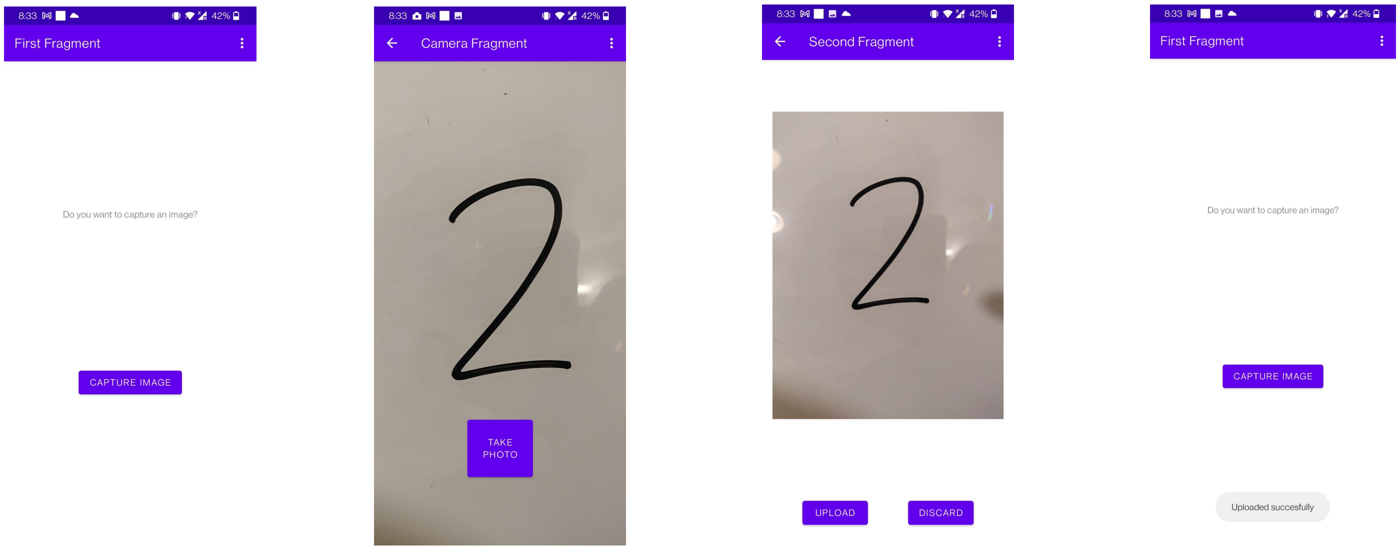
Application Flow


Start
Main Activity 1st Fragment
Take Picture?
No
Yes
Mid Fragment Take Picture
Permission Granted?
No
Yes
Take Picture and Send Post Request to Server
Preprocess the Picture
Use CNN to Classify Image
Store Image in Respective Directory

<u>Android Application</u>

Android applications are developed using fragments. Fragments are the reusable components in the application UI and it helps us in modular application design. The application asks the user to take a picture of a digit. This image is then uploaded to the server which then uses a neural network to classify the image and store them in the respective directory based on the classified digit. Android application is developed using various frameworks -

- **Camera System -** A Jetpack library called **CameraX** is used to make the use of cameras in the application simpler. CameraX gives us the ability to create these use cases, add listeners to them, and then connect them to the activity lifecycle since it is a use-case-based approach.
- **Network System -** An HTTP library called **Volley** makes networking for Android apps quicker and easier. We have implemented our HTTP communication of mobile apps to the flask backend using this library.



<u>Flask Server</u>

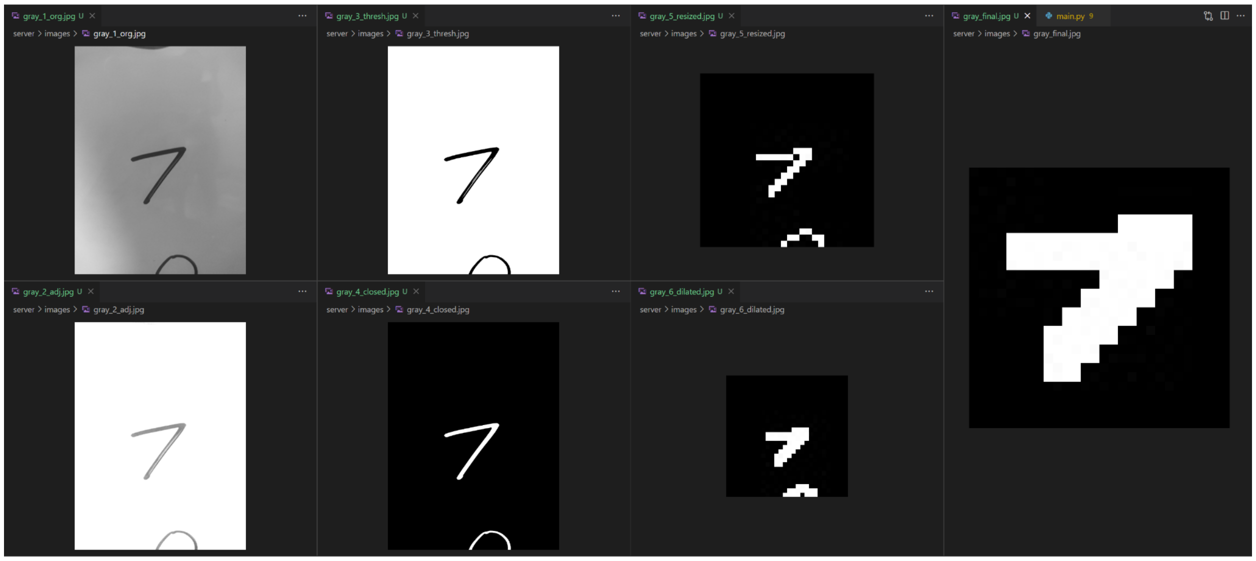
This backend Flask application contains one POST API. This API takes an image as a multipart form request. Once the request is received the image is sent to the neural network for classification and based on the digit classified the image gets saved locally into the folder.

<u>Image processing</u>

Currently we are expecting the captured images to be handwritten digits on a clear background without any major noise. We have tested with white on black background and black on white background. The MNIST dataset uses white digits on black background images. We are preprocessing the captured images to make the input of the classification network better.

Steps taken to process the image (can be seen in demo video) are -

- Conversion to grayscale
- Adjusting contrast and brightness
- Thresholding into binary pixels
- Closing (morphology) operator
- Resizing into 28x28
- Dilating thinner lines
- Detecting digit contour
- Removing rest of the background



Neural Network

The neural network is built using Tensorflow and Keras Library. This neural network is built using 7 layers, each performing a specific task. We have 10 classes in this neural network as we have 10 different digits to classify.

- **Conv2d -** This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.

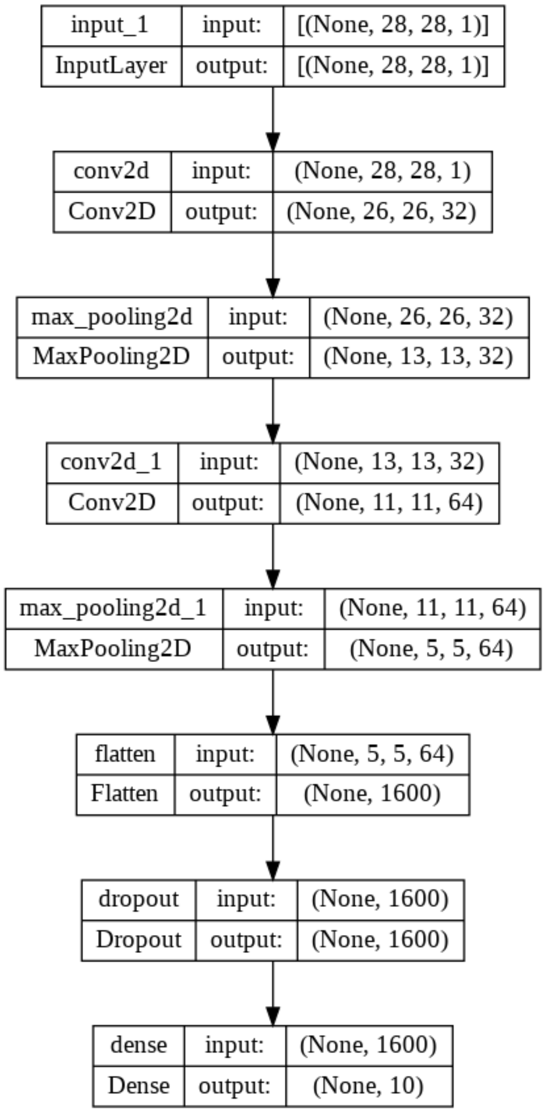
- **MaxPooling2d -** Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool_size) for each channel of the input.

- **Flatten -** Flatten is the function used for flattening the inputs and also at the same time keeping the size of the batch the same.

- **Dropout -** The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.

- **Dense -** Dense Layer is a Neural Network that has deep connection, meaning that each *neuron* in a dense layer receives input from all neurons of its previous layer. Dense Layer performs a matrix-vector multiplication, and the values used in the matrix are parameters that can be trained and updated with the help of backpropagation.

- Model Architecture -


input_1 | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 32)
max_pooling2d | input: | (None, 26, 26, 32)
MaxPooling2D | output: | (None, 13, 13, 32)
conv2d_1 | input: | (None, 13, 13, 32)
Conv2D | output: | (None, 11, 11, 64)
max_pooling2d_1 | input: | (None, 11, 11, 64)
MaxPooling2D | output: | (None, 5, 5, 64)
flatten | input: | (None, 5, 5, 64)
Flatten | output: | (None, 1600)
dropout | input: | (None, 1600)
Dropout | output: | (None, 1600)
dense | input: | (None, 1600)
Dense | output: | (None, 10)