ASSIGNMENT - 3

Team Members: Avish Khosla, Priyadarshini Venkatesan, Sudhanshu Kaul, Sneha Ramkumar, Ujwal Kasturi

ABSTRACT

An Android application that splits handwritten images of digits into four quadrants and sends each piece to a different mobile client to compute the prediction of the digit and store the image in the highest confidence numbered folder.

TECHNOLOGIES USED

- This application was built on Android API version 30
- Java
- Python
- Tensor Flow
- Flask server
- Convolutional Neural Network using Keras Libraries
- OkHttp API to upload from app to server HTTP
- OpenCV

APPROACH

1. ***Developing a User Interface***

   The first UI page has two buttons, "Yes" and "No," where the user selects whether he wants to capture the image or not. The user is required to choose yes if he wants to capture an image using the phone camera. On clicking this option, an alert screen pops up (Fig. 1.2) and prompts the user to capture the image with a better zoom and avoid shadows for better accuracy. On clicking "No," the user gets exited from the main application.

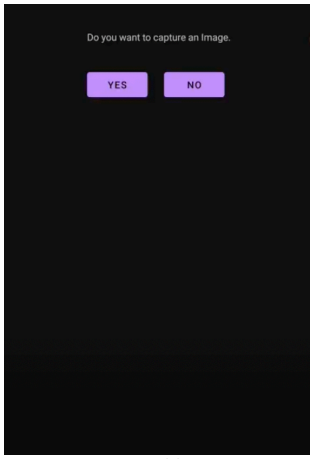

Fig .1.1

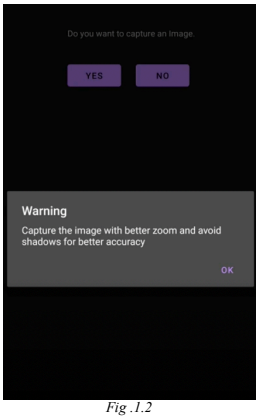

Fig .1.2

Digits from 0-9 are written on a piece of paper and user zooms into a digit without any shadows and captures an image.

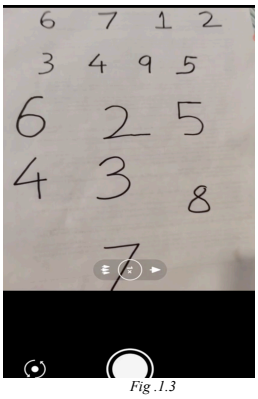

Fig .1.3

The user is prompted with 3 options: Retake, Ok and Cancel. (Fig. 1.4)

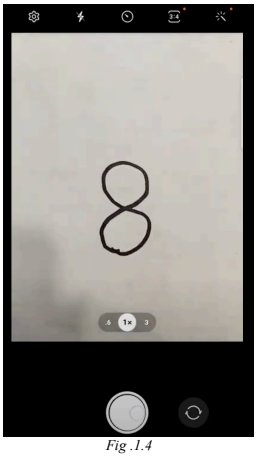

Fig .1.4

On clicking the "Ok" option, the user gets redirected to the preview page of the application. If not satisfied with the image, the user can click on “Retry”. The captured image is sent as a URL to this screen for display (Fig.

1.5). Once the user is satisfied with the image, on clicking the “SPLIT IMAGES" button, the image is split into four quadrants and is displayed on the screen (Fig. 1.6).

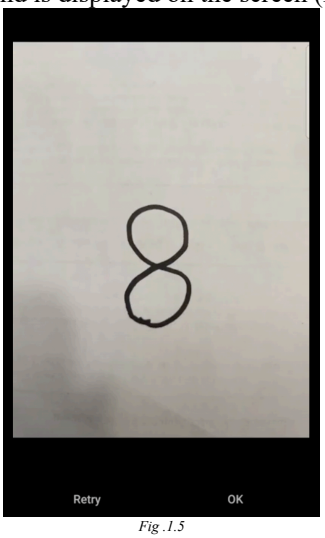

Fig .1.5

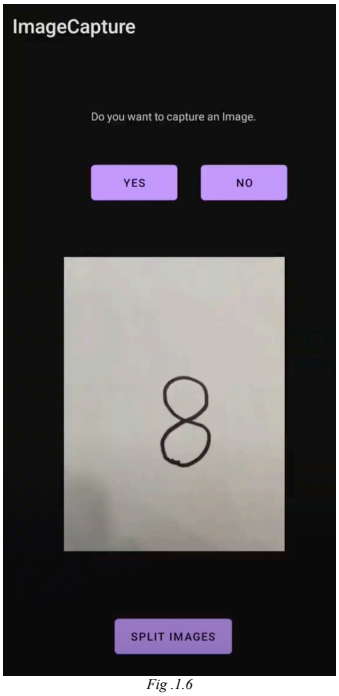

Fig .1.6

On clicking the “UPLOAD” button, the images are offloaded into four different edge devices (Fig. 1.7)

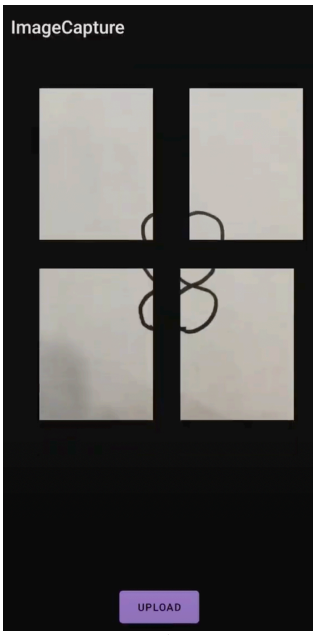

Fig .1.7

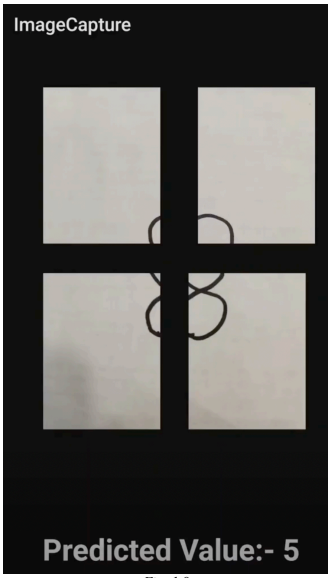

Fig .1.8

The edge devices return a consolidated predicted value, (5 in this case) (Fig. 1.8)

2 .Server side

Fig.2.1 and Fig. 2.2 are the source codes for splitting the image into four quadrants before offloading the image onto multiple edge devices.

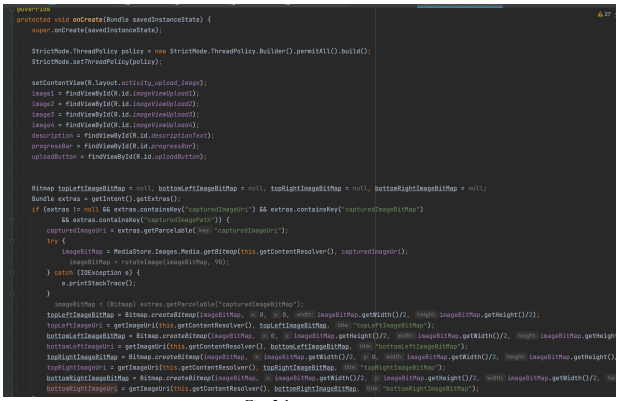

Fig .2.1

```
    image1.setImageURI(topLeftImageUri);
    image2.setImageURI(topRightImageUri);
    image3.setImageURI(bottomLeftImageUri);
    image4.setImageURI(bottomRightImageUri);
    uploadButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { preparePostRequest(); }
    });
      capturedImage.setImageBitmap(topRightImageBitMap);
      capturedImage.setImageURI(capturedImageUri);
    //Conversion of bitmap to byte array to send it to server
}

private Bitmap rotateImage(Bitmap source, float angle) {
    Matrix matrix = new Matrix();
    matrix.postRotate(angle);
    return Bitmap.createBitmap(source,  x: 0,  y: 0, source.getWidth(), source.getHeight(),
            matrix,  filter: true);
}


private Uri getImageUri(ContentResolver contentResolver, Bitmap inImage, String title) {
    ByteArrayOutputStream bytes = new ByteArrayOutputStream();
    inImage.compress(Bitmap.CompressFormat.PNG,  quality: 100, bytes);
    String path = MediaStore.Images.Media.insertImage(contentResolver, inImage, title,  description: null);
    return Uri.parse(path);
}
```

Fig .2.2

```
private void preparePostRequest(){
    progressBar.setVisibility(View.VISIBLE);
    String[] res = new String[4];
    String[] links = {"http://192.168.0.106:5000/","http://192.168.0.252:5000/","http://192.168.0.106:5000/","http://192.168.0.252:5000/"};
    res[0] = callServer(topLeftImageUri,links[0]);
    res[1] = callServer(bottomLeftImageUri,links[1]);
    res[2] = callServer(topRightImageUri,links[2]);
    res[3] = callServer(bottomRightImageUri,links[3]);
    float confidence=-1;
    int num = -1;
    for(String line:res)
    {
        float val = Float.parseFloat(line.split( regex: ":")[0]);
        if(val>confidence)
        {
            confidence=val;
            num = Integer.parseInt(line.split( regex: ":")[1]);
        }
    }
    imageCategory=String.valueOf(num);
    if(imageCategory != null){
        String text = "Predicted Value:- "+ imageCategory;
        description.setText(text);
        description.setTypeface( tf: null, Typeface.BOLD);
    }
    progressBar.setVisibility(View.GONE);
    uploadButton.setVisibility(View.GONE);
    description.setVisibility(View.VISIBLE);
    storeImage(num);
}
```

Fig .2.3

The 4 servers are setup, and each split image is sent to four different devices.

```
private String callServer(Uri ImageUri, String url){

    InputStream iStream;
    try {
        iStream = getContentResolver().openInputStream(ImageUri);
        byteArray = getBytes(iStream);
    } catch (IOException e) {
        e.printStackTrace();
    }
    OkHttpClient client = new OkHttpClient();
    String timeStamp = new SimpleDateFormat( pattern: "yyyyMMdd_HHmmss").format(new Date());
    String filename = "Image_"+timeStamp+".jpeg";
    RequestBody formBody = new MultipartBody.Builder()
            .setType(MultipartBody.FORM)
            .addFormDataPart( name: "image", filename,RequestBody.create(MediaType.parse("image/jpg"), byteArray))
            .build();

    Request request = new Request.Builder().url(url).post(formBody).build();

    Call call = client.newCall(request);
    Response response;
    try {
        response = call.execute();
        imageCategory = response.body().string();
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println("Category Selected: "+ imageCategory);
      testMethod(imageCategory);
    return imageCategory;
}
```

Fig .2.4

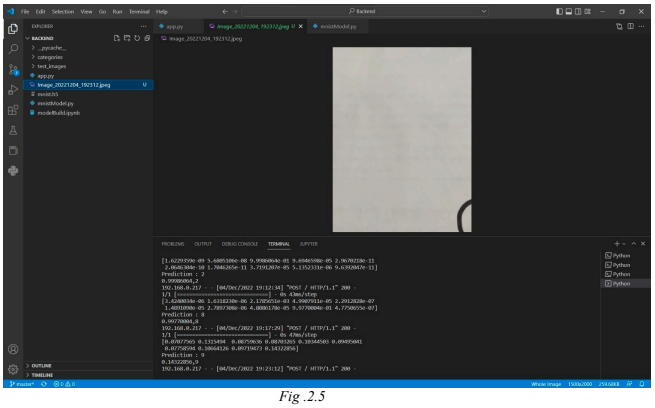

Fig .2.5

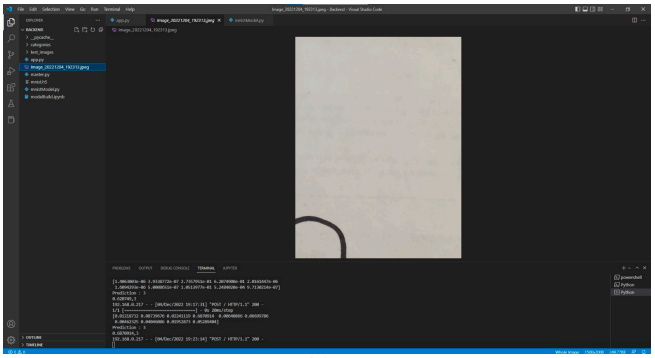

Fig .2.6

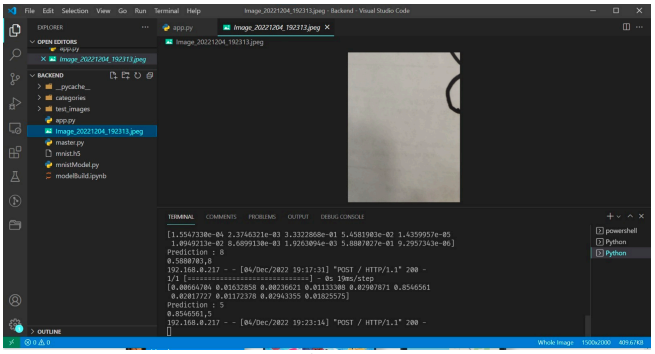

Fig .2.7

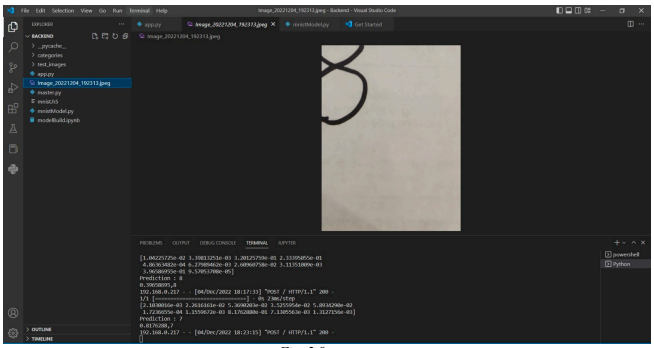

Fig .2.8

The split images are shown on each offloaded device (Fig. 2.5 to Fig. 2.8)

Once the slave application receives the image, it runs the image through a prediction model to predict the value of the digit. In this case, we are using a CNN model

```
def detect_num(img):
    img_proc = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    (thresh, blackAndWhiteImage) = cv2.threshold(img_proc, 127, 255, cv2.THRESH_BINARY)
    img_proc = cv2.resize(blackAndWhiteImage, (28, 28))

    img_proc = preprocess_images(img_proc)

    img_proc = 1 - img_proc # inverse since training dataset is white text with black background


    net_in = np.expand_dims(img_proc, axis=0)
    y_pred = model.predict(net_in)[0]
    print(y_pred)
    print("Prediction :",np.argmax(y_pred))
    return np.argmax(y_pred)
```

Fig .2.9

A Convolution Neural Network Model is used as a Deep-learning framework on the MNIST dataset to classify different handwritten digits. The CNN model is trained on the dataset and this trained model is used for prediction.

The layers of CNN model (Fig. 2.10)

```
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
model = keras.Sequential()
# 32 convolution filters used each of size 3x3
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
# 64 convolution filters used each of size 3x3
model.add(Conv2D(64, (3, 3), activation='relu'))
# choose the best features via pooling
model.add(MaxPooling2D(pool_size=(2, 2)))
# randomly turn neurons on and off to improve convergence
model.add(Dropout(0.25))
# flatten since too many dimensions, we only want a classification output
model.add(Flatten())
# fully connected to get all relevant data
model.add(Dense(128, activation='relu'))
# one more dropout
model.add(Dropout(0.5))
# output a softmax to squash the matrix into output probabilities
model.add(Dense(10, activation='softmax'))
```

Fig .2.10

As seen from Fig.2.5 to Fig. 2.8, the prediction model is run on each edge device and each device returns a prediction. The consolidated predicted value that has the highest confidence score is returned as the predicted value.

Here, the image captured is stored in folder **'5'**.

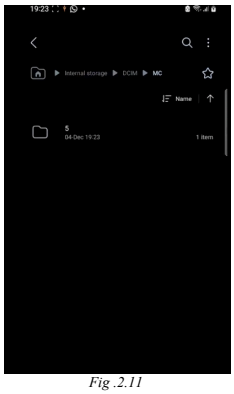

Fig .2.11

Fig .2.12

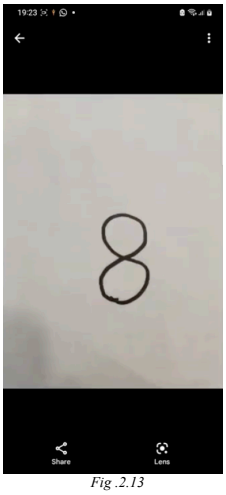

Fig .2.13

CONCLUSION

This android application successfully achieves the concept of Edge Computing. The application captures a handwritten digit image (0-9), splits the image into four quadrants and offloads the images to four different edge devices using a flask server to predict the value of the digit in the image and store it in the main application. A Convolutional Neural Network is trained on the MNIST dataset to classify and predict the handwritten

digit. Since we are splitting the image into many parts, the prediction is not always accurate.

REFERENCES

1. https://developer.android.com/studio
2. https://flask.palletsprojects.com/en/2.2.x/
3. https://www.tensorflow.org/datasets/catalog/mnist
4. https://www.kaggle.com/code/arunkumarramanan/awesome-ml-frameworks-and-mnist-classification
5. https://techtutorialsx.com/2019/04/13/python-opencv-converting-image-to-black-and-white/
6. https://learnopencv.com/opencv-threshold-python-cpp/
7. https://developer.android.com/training/data-storage