CSE 535 Mobile Computing Project 3

Project Group 12: Lakshman Yadav Patakula, Ruthvik Pentareddy, Saivikas Nulu, Srivatsa Pantangi, Uday Yalamanchili

Mobile Application:

We are using the same mobile application from the previous project with minor modifications.

1. The mobile application asks the user to click an image. Upon selecting the image, it is split into 4 quadrants. Using the height and width of the bitmap, we are splitting it into 4 parts. First we create a scaledBitmap which we later split into 4 chunks by passing the scaled bitmap, the x coordinate, the y coordinate, the chunk height and the chunk width.
2. Each quadrant is then sent to a client device running the flask server connected to the network. Thus, 4 different devices(slaves with different IP's) receive 4 parts of the image in the bitmap format, where classification is done on each device and prediction along with the confidence score are sent back to the master device (mobile).
3. The mobile application will receive the results as strings. The string contains the predicted digit and the confidence score. We use the String.parseDouble() function to extract the confidence score and then compare the confidence scores received from the 4 devices. The result with the highest confidence score is selected and we store the image in the respective folder using ***saveImageToGallery()*** method. Using URI, ***contentValues.put()*** and ***Mediastore*** we save the image on the android device.

Server side:

We have built 4 backend flask servers, one on each client device. Every device receives a quadrant of the image as input from the android application. Each of the 4 servers has a pre-trained model on split images of the MNIST dataset.

To establish offloading, we are using 4 laptops, each having a flask server to process the request.

Deep learning model trained on MNIST dataset:

1. Neural network

We used Convolutional Neural Networks (CNN) architecture to build our model. We will add the layers using the model.add() function. Our model uses three convolution layers and two maxpool layers. The sizes of the three Conv2D layers have filters of size 128, 64, and 64 respectively and they all have kernels of size 3x3. Rectified Linear Activation(ReLU) is used as the activation function. The convolution layers take inputs (i.e. images of MNIST dataset) of dimensions (28, 28, 1). Used a 2x2 MaxPool2D layer after the first convolutional layer and a 1x1 MaxPool2D

layer after the second convolutional layer to reduce the number of parameters to learn and the amount of computation performed in the network.

2. Dataset Preprocessing

We then obtained the training and test data for MNIST from the tensorflow library. We built a preprocessing function to split the image dataset into 4 quadrants, resulting in 240000 images in the train dataset and 40000 in the test dataset. . The data is then preprocessed where we reshaped the dimensions from (n,14,14) to (n,14,14,1), normalized the X values and converted Y values into categorical variables using one-hot vector encoding. To compile our model, we decided to use Root Mean Squared Propagation with a learning rate of 0.001 as our optimizer and applied categorical cross-entropy function to evaluate loss. We then trained the model using model.fit() method, where we trained the model for 20 epochs and the validation split set to 0.1 to evaluate validation loss and validation accuracy. Using model.evaluate() function over our test set, we have obtained an accuracy of 81.84%. Then the model was saved as a .h5 file using the model.save() function, which we would later deploy on our flask server.

Flask server code:

Here we used the model we trained to perform classification. The POST call made from the app calls the ***handle_request()*** function that we use to process the image. The image obtained is sent to the ***predictImg()*** function to preprocess the image and then run prediction. The response from the predicted model sends a string of probabilities, that are parsed to store as a list of float probabilities. We then pick the best probability from this list, obtain the digit related to it and then concatenate the string and confidence value. This concatenated string is then sent via POST Response to the mobile application.

Preprocessing at predictImg(): This function is used to preprocess the input image before the contouring is applied. Before we apply any preprocessing, we first convert our image from RGB to grayscale using the ***cv2.cvtColor()*** method. A median 3x3 filter using ***cv2.blur()*** method is used to smoothen the image and remove noise. To fix missing edges between the pixels in the input image and connect them together, we use dilation and erosion morphological processing. We defined a 3x3 structuring element to be used for dilation and erosion. Using dilation over a single iteration, we managed to connect any missing pixels existing in the digit pixels. Using erosion, we removed the excess boundaries obtained by dilation. To obtain a binary image, we used adaptive thresholding, and the method used mean thresholding to obtain the mean of the neighborhood area. We used this thresholded image, reshaped to (1,14,14,1) to obtain a prediction from the tensorflow model.

Prediction at predictImg(): We load our trained model using ***keras.models.load_model().*** We use the rmsprop_model.h5 file we saved after training. Then we call the model.predict() function and send the reshaped image to match the size of the input function for the trained CNN model.

Contributions:

1. MNIST model - Saivikas Nulu, Ruthvik Pentareddy
2. Image preprocessing and new flask servers- Srivatsa Pantangi, Lakshman Yadav Patakula
3. Android development and flask Integration - Uday Yalamanchili

References:

1. https://developer.android.com/
2. https://flask.palletsprojects.com/en/2.2.x/
3. https://www.tensorflow.org/resources/models-datasets
4. https://www.geeksforgeeks.org/dividing-images-into-equal-parts-using-opencv-in-python
5. https://oleksandrg.medium.com/how-to-divide-the-image-into-4-parts-using-opencv-c0afb5cab10c
6. https://www.cse.scu.edu/~m1wang/projects/IoT_offLoadingMobileEdgeComputing_17s.pdf