Mobile Offloading to Neural Networks in a Cloudlet Architecture for Handwritten Digits Recognition

Mahesh Chandra Yayi  
Arizona State University  
ASU ID: 1224141347  
myayi@asu.edu

*Abstract*— Mobile devices have become an integral part of our daily lives, but they are often limited by resources in terms of computational power, battery life, etc. Mobile offloading is a promising solution to this problem, as it allows mobile devices to transfer compute-intensive tasks to more powerful servers or clouds. This report investigates the use of mobile offloading to improve the performance of handwritten digit recognition using artificial neural networks (ANNs). Utilizing the provided offloading schema, we use a cloudlet architecture, which consists of a network of nearby servers that are connected to the mobile devices. The results provide valuable insights into the benefits and challenges of mobile offloading for handwritten digit recognition using ANNs trained on the MNIST dataset.

* Keywords—Mobile Offloading, Mobile Computing, Computation offloading, Cloudlet Architecture, Image Recognition, Artificial neural networks (ANNs), Real-time processing

# Introduction

Mobile devices have become a natural extension to us with the advent of smartphones which has only accelerated with more miniaturized devices such as smartwatches and BCIs. However, these devices are often limited by their computational power and battery life, which hinder their performance and usability. One solution to this problem is mobile offloading, which involves transferring computationally intensive tasks from mobile devices to more powerful servers or clouds to achieve turnaround time suitable for real-time or near real-time applications.

In this paper, we explore the use of mobile offloading to improve the performance of handwritten digit recognition using artificial neural networks (ANNs) trained on the MNIST dataset. By offloading the computation to a cloudlet architecture, we aim to reduce the burden on mobile devices and improve the speed and accuracy of the recognition process.

Mobile offloading can be done in many ways to reduce compute-time using application-specific hardware or vastly higher compute-capable devices on local networks. One approach is to use a cloudlet architecture, which consists of a network of nearby servers that are located close to the mobile devices and connected to the internet via high-bandwidth links.

The main advantage of using a cloudlet architecture for mobile offloading is that it allows mobile devices to offload tasks to servers that are physically close by, which can significantly reduce the average latency of the offloading process. This can be especially useful for tasks that require real-time processing or have strict time constraints, as it allows mobile devices to receive the results of the offloaded tasks more quickly. In addition, a cloudlet architecture can also provide additional computational resources and storage capacity to mobile devices, which can improve the performance and capabilities of the devices. However, there are many challenges to solve such as shared memory bandwidth and capacity, scope for parallelized optimization of the task, network latencies, etc.

This project was completed as a group project for the CSE 535 Mobile Computing course instructed by Dr. Ayan Banerjee. The rest of this project report is structured as follows: Section II outlines the three-step solution to the task and describes my contributions to the project. Section III presents the results of the implementation, and potential areas for future investigation are discussed which is followed by the References section.

The team members of the group apart from me are as follows.

* Fenny Zalavadia
* Shilpitha Gandla
* Ajay Kannan
* Sai Krishna Reddy Cheruku

# Solution

* 1. *Step 1: Creating the mobile application along with a Flask server.*

Initially, an android application was created that lets the user capture a photo which is uploaded to a server. The server saves the received data on the host device (a laptop in this instance). This step mainly consists of two parts: the android application and the Flask server.

The Android app was developed using Android Studio and Java. It features a two-page design, with a prompt on the first page asking the user if they want to capture a photo. The user can choose between two buttons: "Yes" and "No." If "Yes" is selected, the device's camera is triggered to open using an intent, where they can take a photo. The user can then confirm or retake the photo. If "No" is selected, the app closes. These actions are implemented via listeners that are initialized during the application launch. These listeners trigger corresponding methods upon button clicks. Once the user confirms their photo, it is displayed on the second page using an ImageView instantiation. The user can then click an "Upload" button to send the image and selected category to the server using a multipart/form-data POST request. If the image is successfully saved, the user is redirected to the first page. (The pages in the android application design are shown in the image below).

Graphical user interface, application

Description automatically generated

Moving on to the backend of the application, it was developed by me using Flask and Python. Flask is a microframework written in Python that enables handling HTTP requests on the internet, LANs, and WANs. Upon receiving a POST request from the mobile app, the server stores the image in the specified category folder. If the folder does not exist, it is created, and the image saves within it.

## Step 2: Performing mobile offloading to a standard server for image recognition.

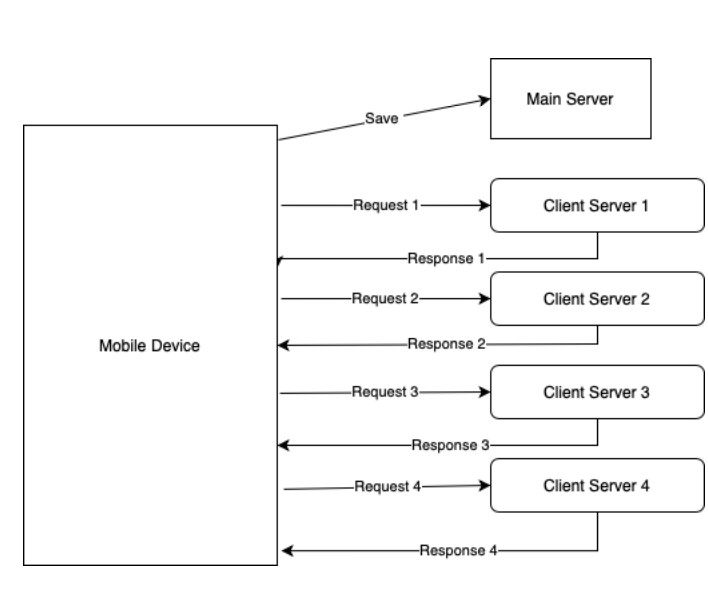
We selected ANNs as the model for image recognition in our project owing to the simplicity of the greyscale MNIST dataset. After testing various models with different hyperparameters, we finalized on the following architecture: 3 hidden layers, each with 256 neurons, connected through a ReLu activation function in the first and second layers, with 45% dropout during training. SoftMax layer is placed after the third layer for giving the prediction as the output. The model uses categorical cross entropy as the loss function and ADAM for gradient optimization. The model was trained for 20 epochs with a batch size of 128 and achieved an accuracy of 96.4% in our tests.

After finalizing the model architecture (based on performance after the training phase), it was saved in Flask server to handle the digit prediction. The incoming images were converted into greyscale and the trained model is used to make predictions on handwritten digits. The server creates a folder (if it does not already exist) with the name of the prediction and stores the image within it.

## Step 3: Performing mobile offloading to a cloudlet server for image recognition.

The offloading schema (that was provided) was to divide the image into four quadrants and predict the class of each quadrant using one server from a cloudlet server cluster consisting of four laptops running four independent Flask servers (each containing the ANN model trained on the quadrants of the MNIST images to consequently accommodate for the offloading schema), generating confidence values for each digit class. The resulting output is a 4x10 matrix where the columns represent the digit class, and each row consists of the confidence values of the corresponding server's prediction. The confidence values are summed across each class (i.e., column-wise summation), and the digit with the highest cumulative confidence value is chosen as the prediction.

Modifications were done to the android application also. After the conversion of the image into greyscale, it is divided into four quadrants and sent to each server for prediction. Upon receiving outputs from all the four servers, confidence values are summed to give the final output as previously mentioned and sent to a fifth server that creates a folder (if not previously present) and saves the prediction. The architecture of the trained deep learning model that is stored in each of the cloudlet servers is as follows: The model architecture comprises a dense layer of 256 hidden units followed by another dense layer of 100 units and a final layer with 10 neurons. ReLu activation layers are placed after the first two hidden layers and a SoftMax layer is placed after the last layer. We train the model for 20 epochs with batches of size 512. The loss function and the optimizer are unchanged from the model used in Step 2. The resulting test accuracy is 81.8%. The model was then saved in each client to predict the handwritten image received. The overview of the functionality is detailed in the following image.



## My contributions to the project.

In Step 1, I added the "Yes" and "No" buttons to the first page's xml and created corresponding listeners in Java. In addition, I participated in the group effort to find the right combination of hyperparameters for the ANN in Step 2 and Step 3. Finally, I also wrote the code for splitting the image into four quadrants and calculating the combined result after receiving the servers' outputs.

# Results and Future Focus Areas

The offloading done in Step 2 yielded real-time predictions. On-device learning has not been implemented and hence the performance gains achieved due to offloading cannot be quantified. We did not see any performance gains after switching to a cloudlet server. Latency and the accuracy of the predictions are significantly better than the results from Step 3. This is due to the incompatibility between the offloading schema and the nature of the computation involved in the mobile application. Splitting the images not only reduced the prediction accuracy, but also the increased the latency due to the increase in turnover time caused by the mobile application continuously polling four separate servers for results. These problems can be resolved with further study. Improvements can be made to the offloading schema and reduce latency through simpler approaches to communicate with the server. The results from this project indicate that compute-intensive applications that can truly utilize the parallelism and the `asynchrony of the cloudlet server architecture truly stand to gain vast performance improvements. Applications designed this way ensure consistent performance for all the users of the application. There are more benefits than shortcomings to mobile offloading and thus becomes essential for industry-grade applications. Insert <usecases for cloudlet architecture>. I believe that hierarchical offloading machine learning systems will become essential for inventing AGI.

##### References

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**