**IMAGE CLASSIFICATION USING NEURAL NETWORKS**

# 

Table of Contents

# 

[**INTRODUCTION**](#_jhosz9fyrc6) **1**

[**THE CLIENT SIDE**](#_atwoemhxx1qq) **1**

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **INTRODUCTION**

We have seen how to perform a dot product operation in the Client-Server model. On making additional changes, it is possible to make use of tensor functions such as matrix multiplication and addition to create neural network layers by chaining the processes together.

In the following document, we use the Client-Server setup by treating one of the clients as an inputs provider who would provide the matrices containing the matrices for images from the MNIST dataset, and the second client would provide the weights and biases of the underlying neural network. The servers would continue doing the necessary operations and broadcast the result.

# **THE CLIENT SIDE**

In the case of the image provider, there is not much difference in the way we send a normal matrix explained in the Client-Server setup documentation. A single file containing the values of pixels of all the images is read by the client and shares are generated using the Generation engine and Twister algorithm, and this is then sent to the server upon establishing a tcp connection. The only difference here is the fact that we read from a csv file instead of a regular text file. This means that there will be additional reading helper functions such as **lineStream** to get the correct set of values stored in the right place.

Additional to this, the weights\_provider file requires a matrix class that allows it to read large matrices from csv files. Since it is a tedious process, a separate class addition was a necessity. The method of shares generation remains the same. To execute the client code, we must ensure that we have the servers up and running as discussed later, and upon entering the build\_ folder of the DP CS repository, the following codes are executed for each client:

**Client 0:**

***cd neural\_net\_dp***

***./image\_provider --compute-server0-port 1234 --compute-server1-port 1235 --dp-id 0 --fractional-bits 18***

**Client 1:**

***cd neural\_net\_dp/***

***./weights\_provider --compute-server0-port 1234 --compute-server1-port 1235 --dp-id 0 --fractional-bits 18***

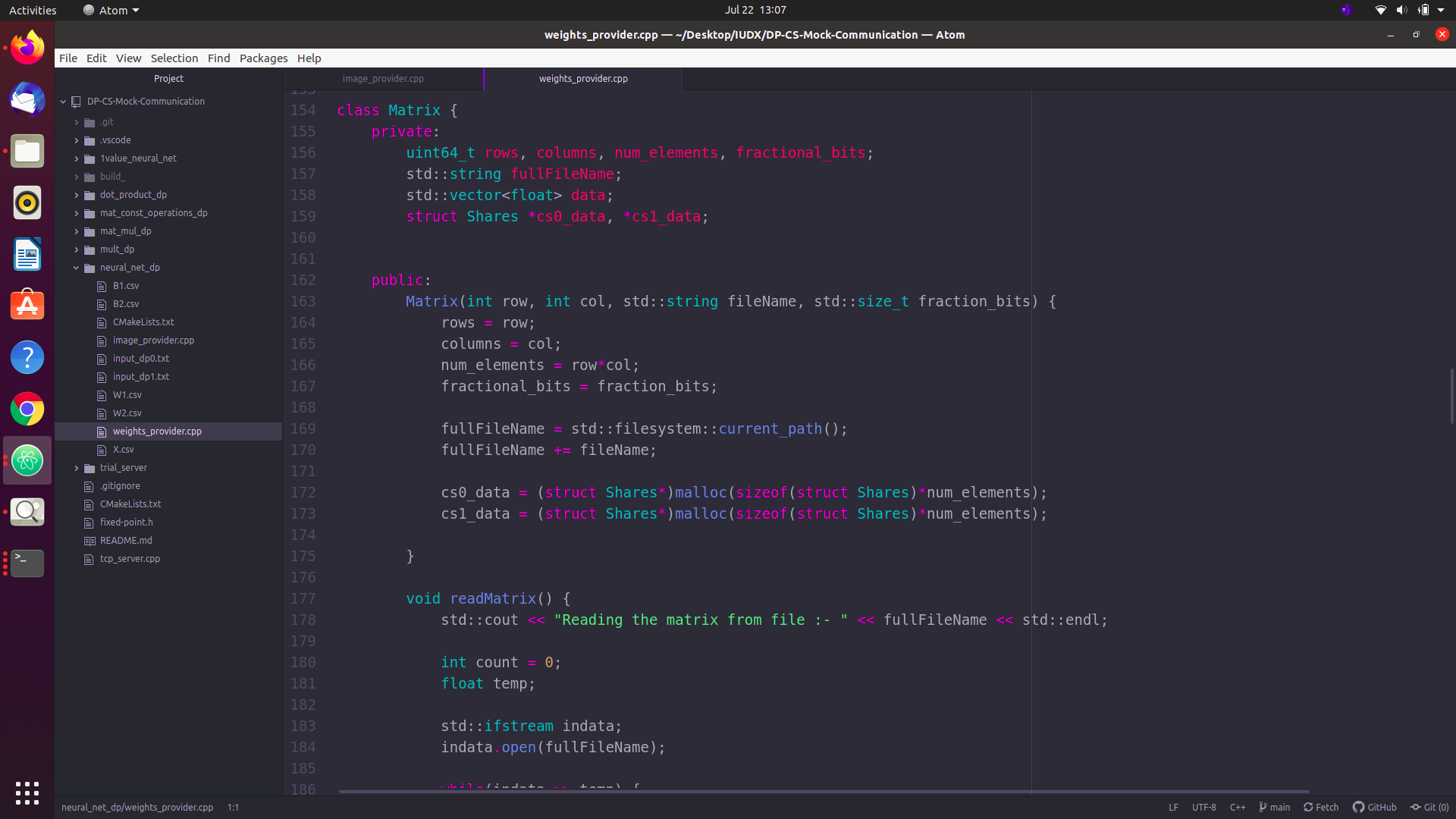


Fig 1: The Matrix class

# **THE SERVER SIDE**

First we shall finish the execution steps. As mentioned before, we must start up the servers before we can execute the client code. For this, we go into the build repository of MOTION2NX, and for each terminal we enter the following:

**Server 0:**

***./bin/tensor\_gt --my-id 0 --party 0,::1,1234 --party 1,::1,1235 --arithmetic-protocol beavy --boolean-protocol beavy --fractional-bits 18***

**Server 1:**

***./bin/tensor\_gt --my-id 1 --party 0,::1,1234 --party 1,::1,1235 --arithmetic-protocol beavy --boolean-protocol beavy --fractional-bits 18***

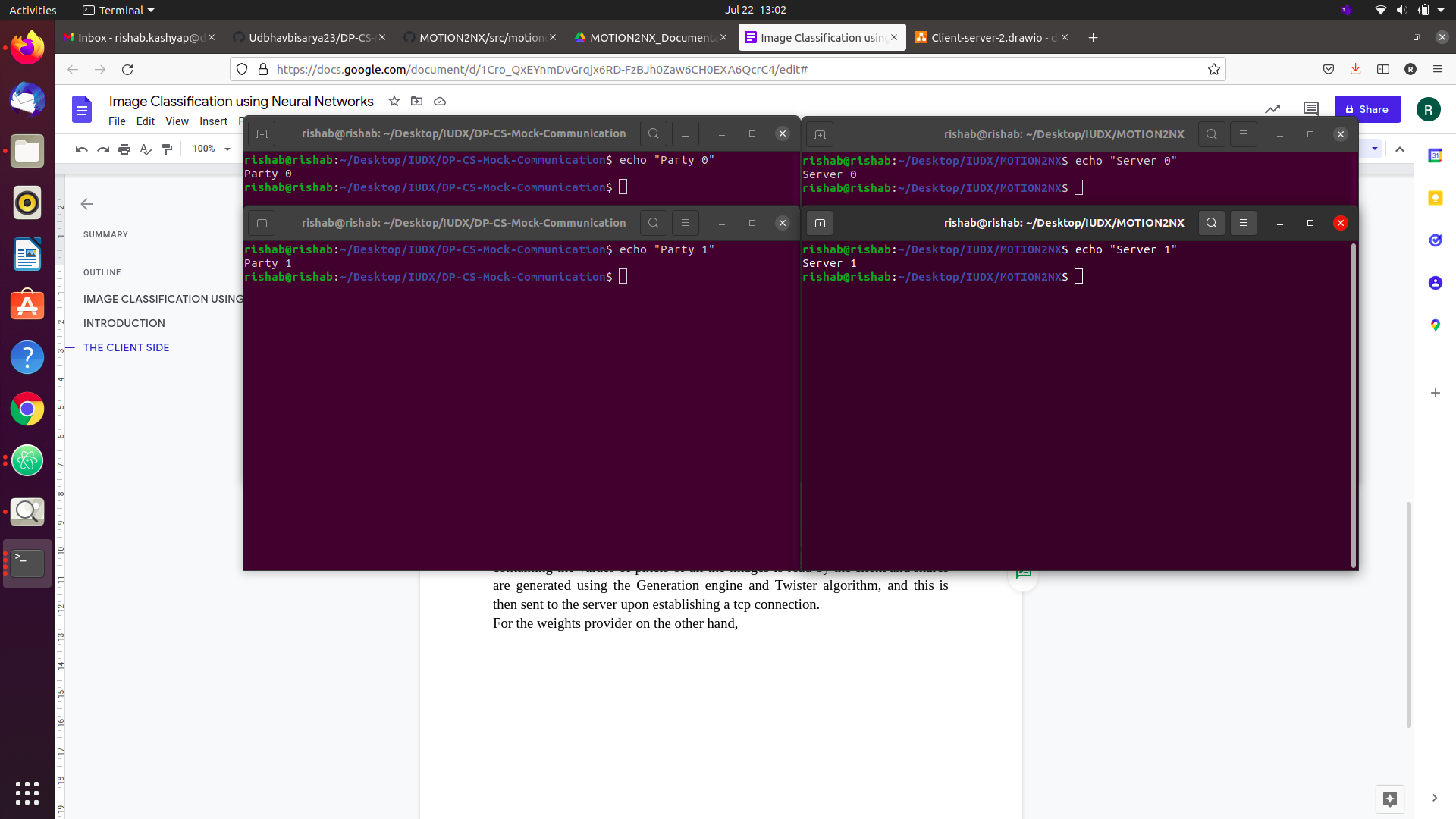
****

Fig 2: Setting up the 4-way architecture

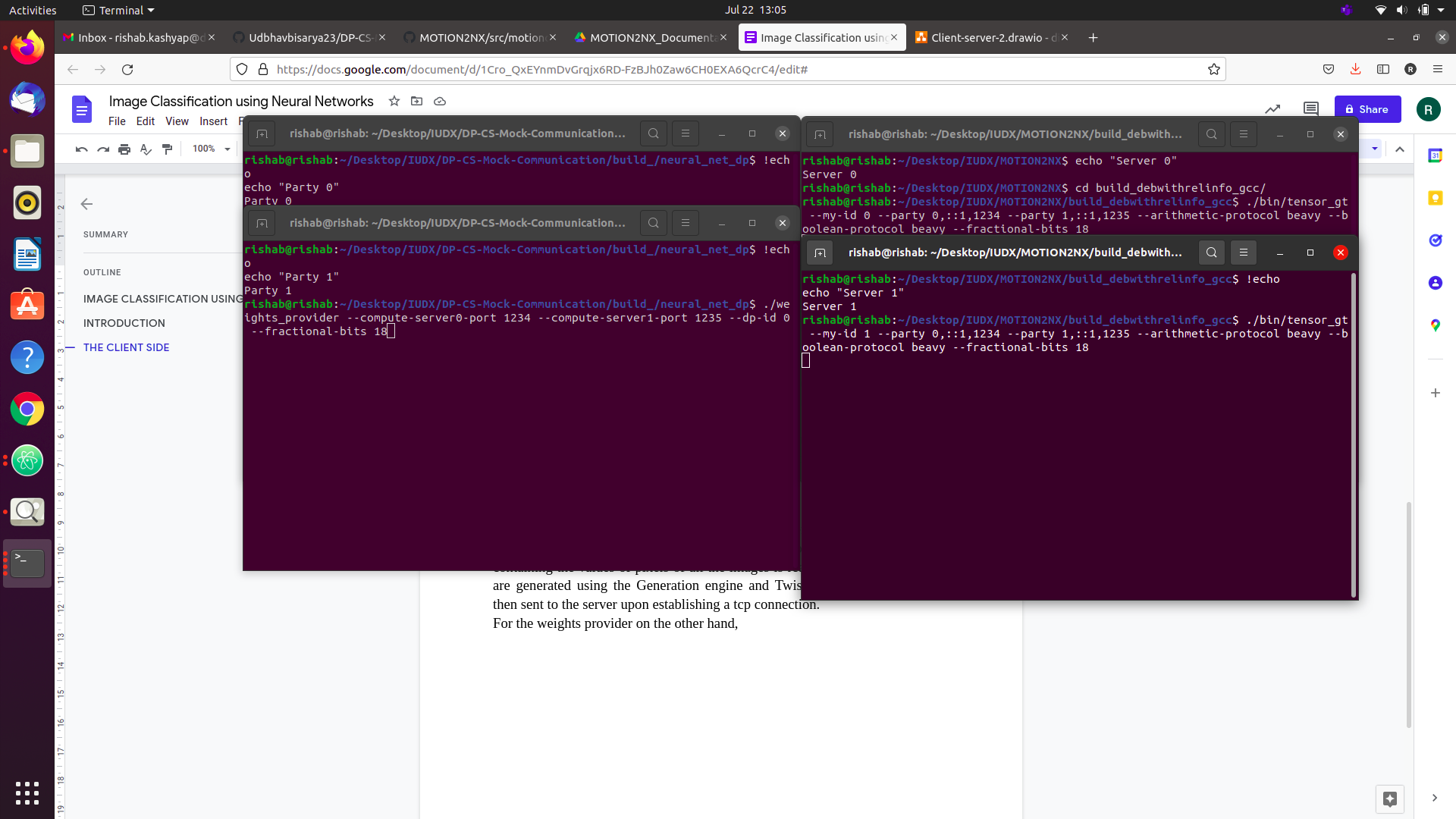


Fig 3: Executing the server codes then the client codes

Once we have the servers running, the client codes as mentioned above are executed. It is important that the image provider is executed first, then the weights provider is executed. Additionally, for the weights provider, each time the message “Sending to Server” appears, we must hit the enter button to send the weights and biases at each step. This is a minor precaution taken to prevent memory overloading due to the sheer size of the matrices.

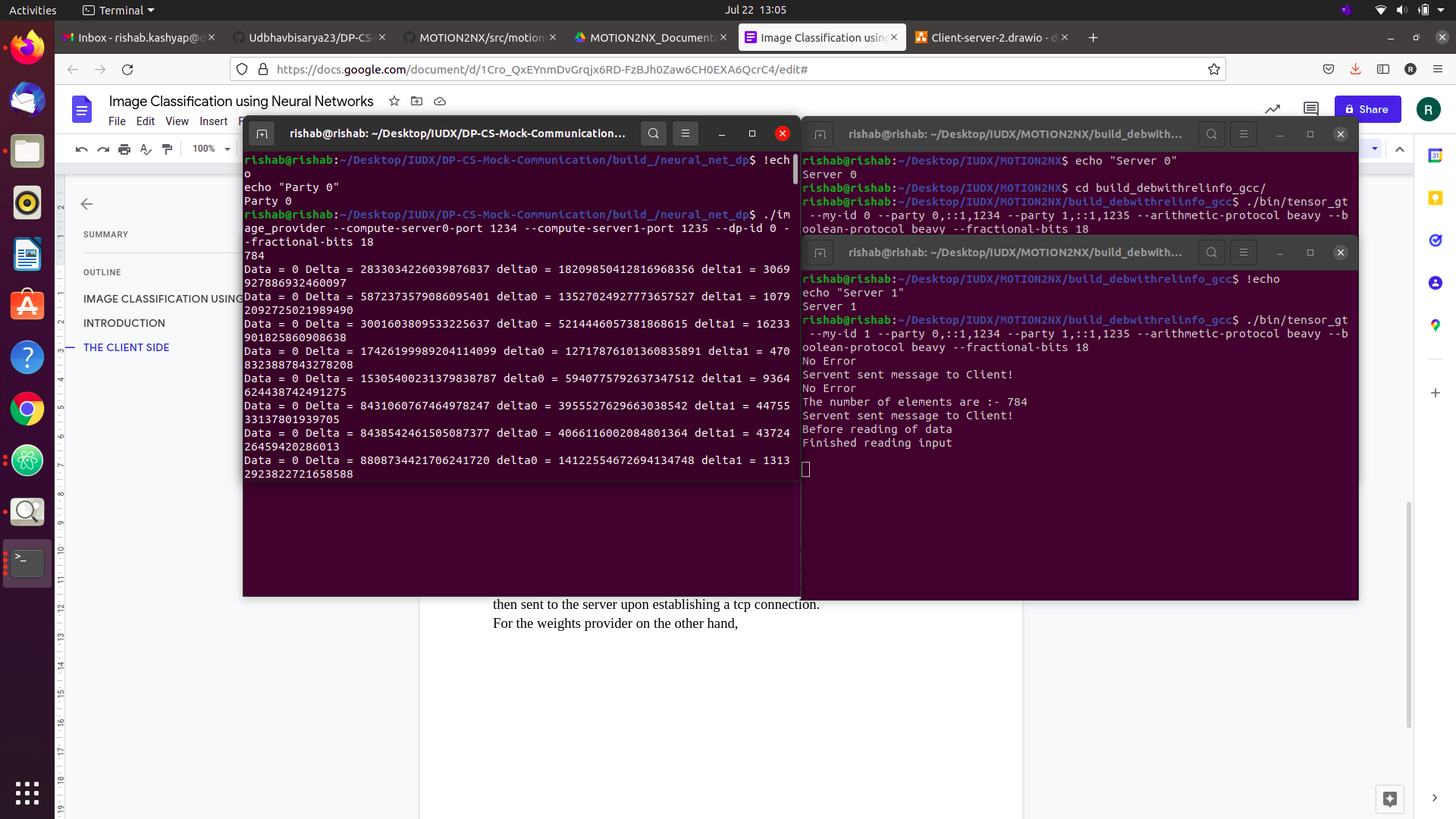


Fig 4: Executing the image provider

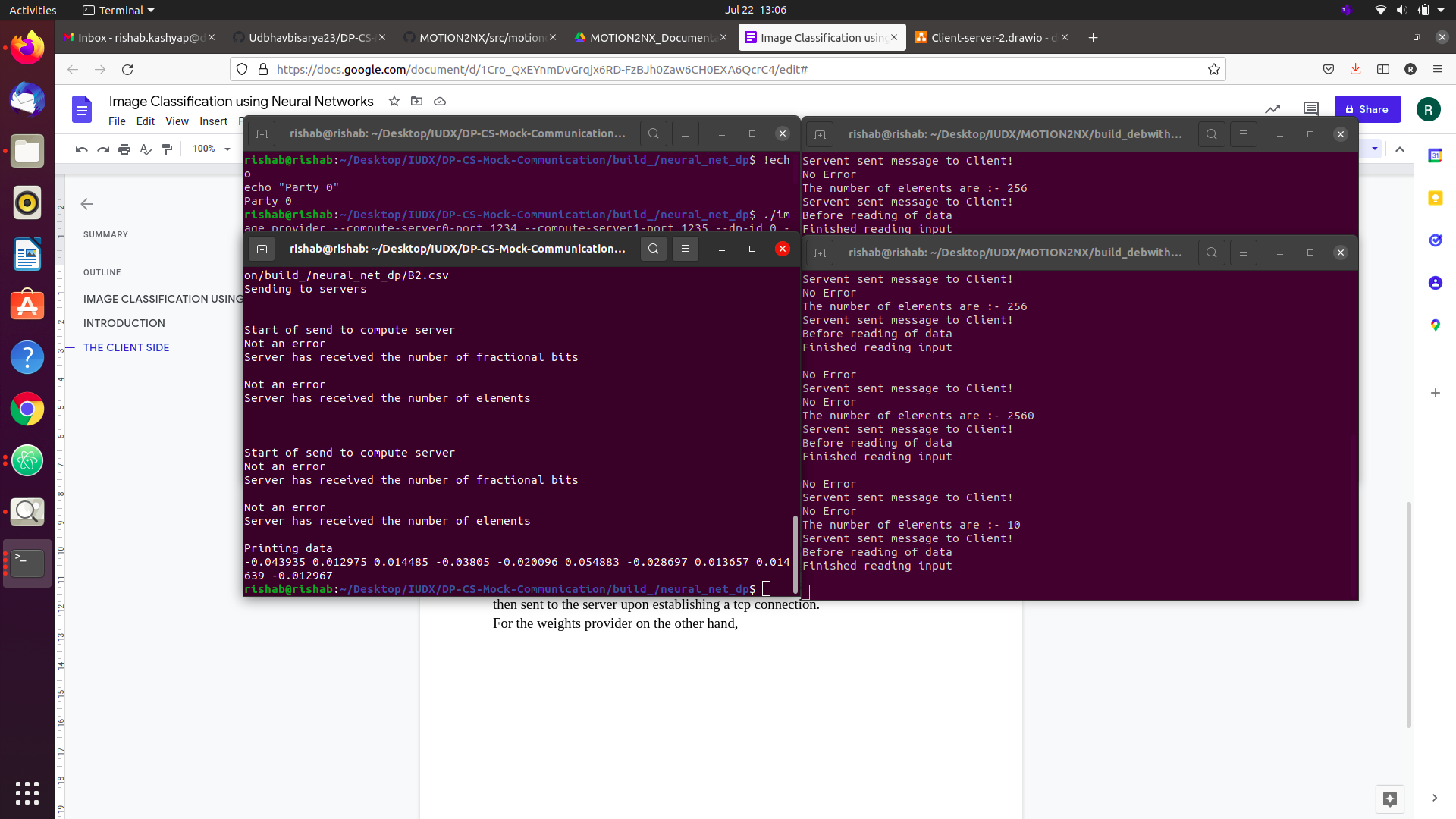


Fig 5: Execution of the weights provider after 7 enters

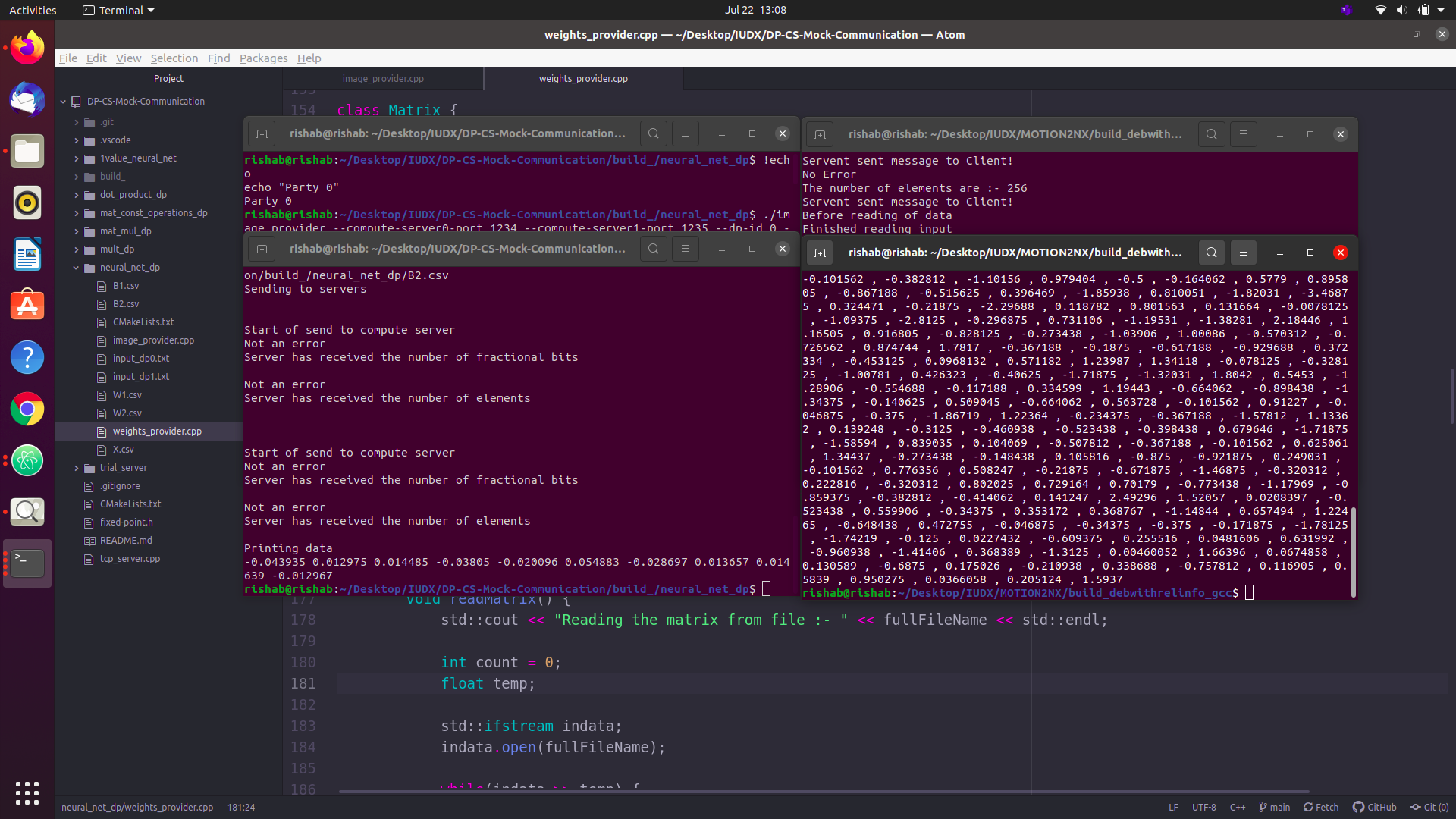


Fig 6: Server intermediate output

We can see in the above images the servers take a couple of minutes to perform the operation. The intermediate results of the first and second matrices are displayed in this case to show the operation, but ideally, we must get the final output value on the terminal instead.