

Analysis and Interpretation of Biological Data

Assignment-2

Siddharth Betala

BE19B032



Introduction

A convolutional neural network (CNN or ConvNet) is a network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction.

CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They can also be quite effective for classifying non-image data such as audio, time series, and signal data.

As we'll learn going ahead, I have used simple models for both questions, given that we had to code the CNN from scratch because of computational and time constraints. To accelerate the process, I decided to use CuPy instead of NumPy to leverage the GPUs on Google Colab. We'll discuss the effect of this further.

The fact that the CIFAR-10 dataset has a significant bottleneck in the form of images with a maximum resolution of 32x32 is a positive component of this assignment. This means that, while more powerful networks with higher parameters and fancier connections will perform better, the benefit of increasing complexity will be minor because there is only so much detail in the image to collect.

Question-1

Network architecture

As mentioned before, I kept it quite simple. I used 3 Convolution Layers, each of kernel size (2,2) and stride = 2. Stride is a parameter to modify the amount of movement over the image or video.

The first convolutional layer has 1 input channel and 64 output channels. The second has 64 input channels and 10 output channels, while the last has 10 input channels and 10 output channels, as we need to classify all the objects and map each to one of the 10 labels.

In between the convolutional layers, I used the ReLU activation function and Max Pooling Layers. Each Max Pooling layer has a dimension of (2,2) and stride of 2. At the output layer, I used the Softmax activation function.

The overall architecture:



The final convolutional layer then predicts the logits of the classes (10 in our case) for each image in the batch. Again, for simplicity, there aren't any residual connections, batch normalization, or other such features. One could experiment with other activation functions, but ReLU and SoftMax have been shown to give good results.

Number and type of Layers used

3 convolutional + 2 MaxPool layers = 5 layers

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

Convolutions are extremely effective information extraction layers, but they come with a number of drawbacks. This is particularly true when using a kernel size of 2 and a stride of 2. Convolutions rose to prominence as a result of their capacity to collect local and spatial information, giving them an advantage over their MLP counterparts. But, ironically, they fall short for the same reason, as there are instances of local features dominating the complete picture.

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after the max pooling layer would be a feature map containing the most prominent features of the previous feature map. In CNNs, it helps in recognizing edges.

Neural networks require linear independence of the input features and low dimensionality of the input space. We utilize CNNs instead of other NN designs because the data we commonly analyze with them (audio, image, text, and video) does not usually fit either of these hypotheses.

The hidden layers of a CNN learn various abstract representations over their input by performing convolution and pooling during training, which reduces the dimensionality of the input.

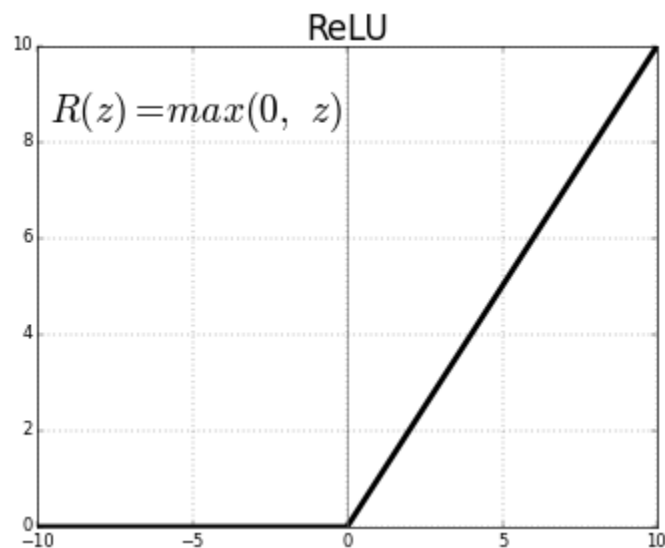
The network then believes that these abstract representations are independent of one another rather than the underlying input data. These abstract representations are usually found in the CNNs hidden layer, and they have a lower dimensionality than the input.

A CNN thus helps solve the so-called “Curse of Dimensionality” problem, which refers to the exponential increase in the amount of computation required to perform task in relation to the unitary increase in the dimensionality of the input.

Activation functions used

- 1) **ReLU (Rectified Linear Unit) Activation:** The Rectified Linear Unit is best described as:

$$\text{ReLU}(x) = \max\{0, x\}$$



Hidden layers of neurons in a trained CNN correspond to possible abstract representations of input information. When faced with an unknown input, a CNN has no way of knowing which of the abstract representations it has learnt would be useful.

There are two conceivable (fuzzy) situations for any given neuron in the buried layer that represents a learnt abstract representation: it is relevant, or it isn't.

If the neuron is irrelevant, this does not rule out the possibility of other abstract representations. If we employed a function whose image encompasses the negative area of the real field, this indicates that the output of a neuron would contribute adversely to the output of the NN for certain values of the input.

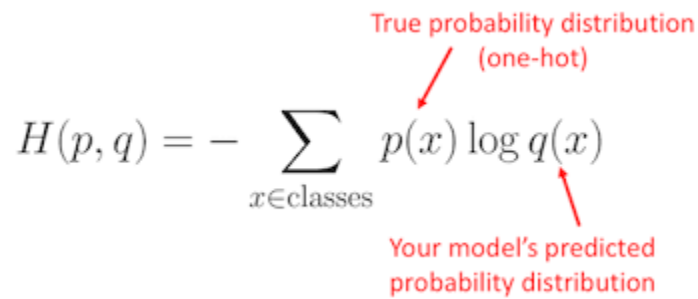
The usage of ReLU helps prevent the exponential growth in the computation required to operate the neural network. It also fixes the vanishing-gradient problem, which other activation functions like sigmoid can't do. ReLU also has biological significance, as it is similar to threshold-based activation of neurons.

- 2) **SoftMax Function:** The softmax function transforms a vector of values into a vector of probabilities, with the probability of each value proportional to the vector value. This may transform the output vector to the class label (based on which index has the highest probability), which is useful for classification models.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Loss Function

The loss is computed via the **Categorical Cross-Entropy Loss** method, which first converts logits to probabilities via the softmax function and then computes the KL-Divergence distance between the predicted and the ground truth probability distributions.



The diagram shows the formula for Kullback-Leibler divergence: $H(p, q) = - \sum_{x \in \text{classes}} p(x) \log q(x)$. A red arrow points from the text "True probability distribution (one-hot)" to $p(x)$. Another red arrow points from the text "Your model's predicted probability distribution" to $q(x)$.

Learning Algorithm

The learning algorithm used for this function is **Stochastic Gradient Descent**. **SGD** is an iterative method for optimizing an objective function. Because it replaces the actual gradient (derived from the complete data set) with an estimate, it can be considered a stochastic approximation of gradient descent optimization (calculated from a randomly selected subset of the data).

Learning Rate

Learning Rate is 0.001. But I realize that a decaying learning rate would give better results.

Loss and Accuracy

Batch size = 64, Epochs = 50, Parameters = 1860

Calculation of Parameters:

Product of shapes = $1 \times 32 \times 2 \times 2 = 128$

Bias1 = 32

Second Weight = $32 \times 10 \times 2 \times 2$

Bias2 = 10

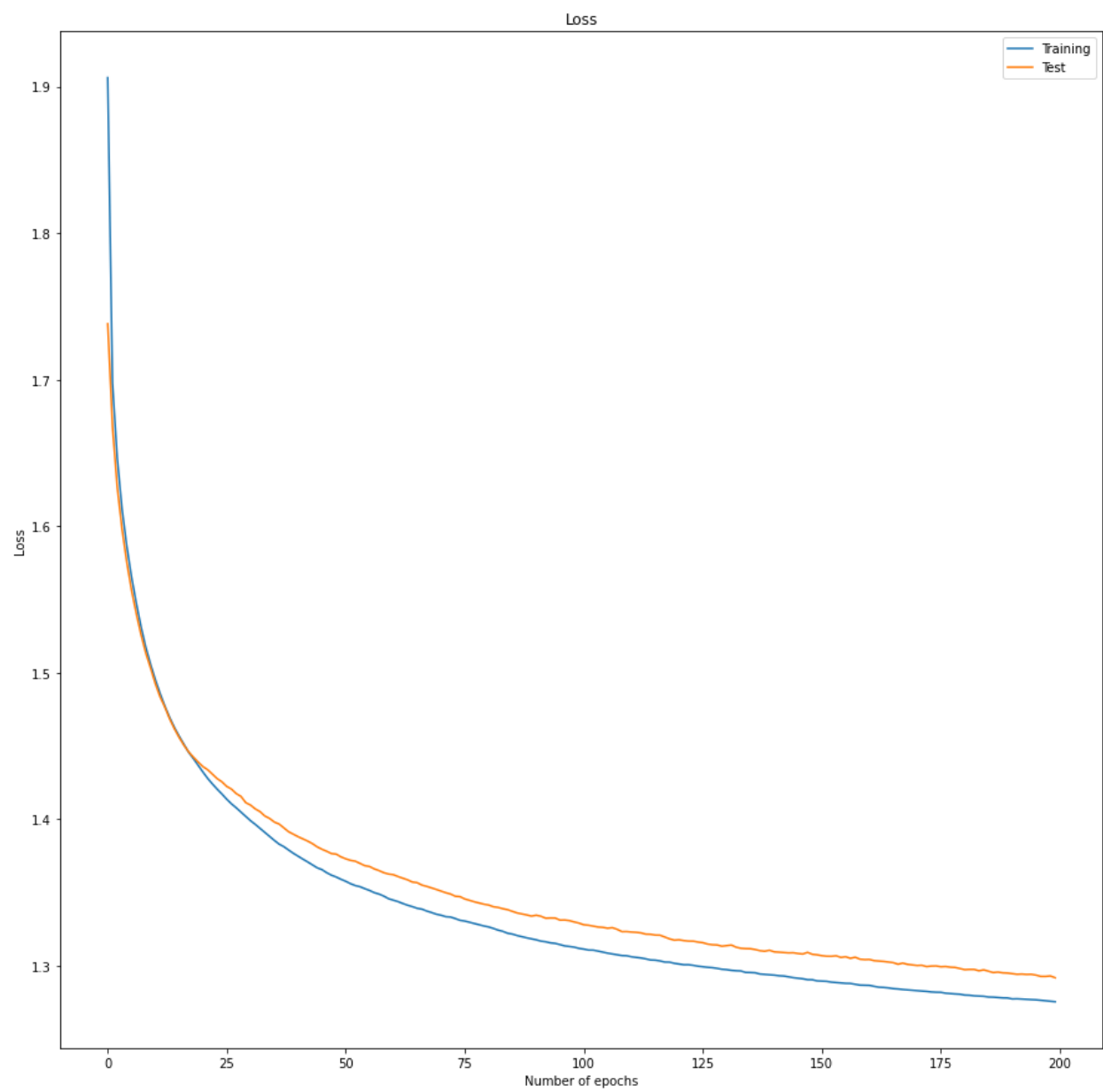
Third Weight = $10 \times 10 \times 2 \times 2$

Bias3 = 10

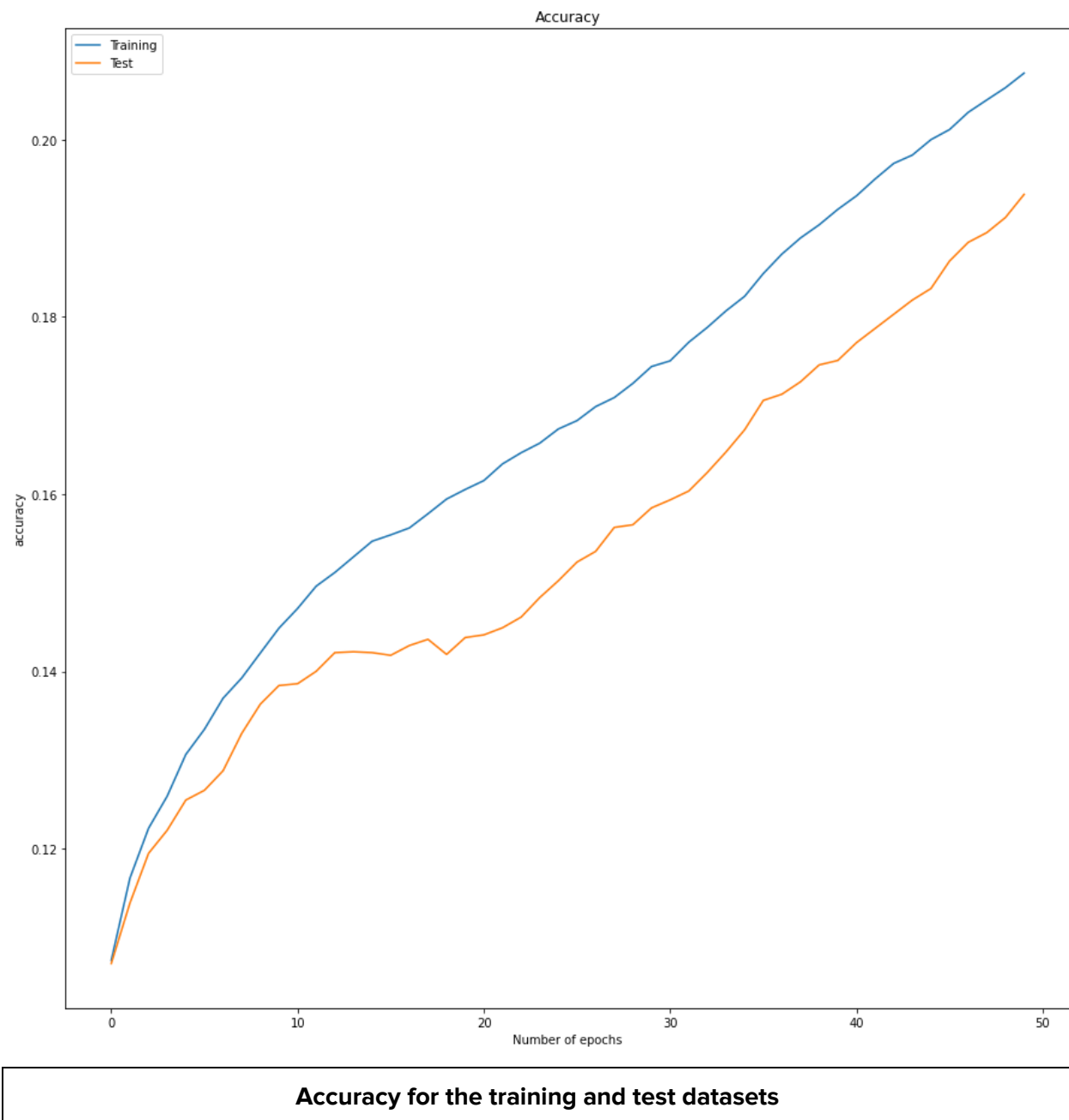
Total = $128 + 32 + 1280 + 10 + 400 + 10 = 1860$

A **loss of 2.0081336**, **accuracy = 20%** was obtained when the model was run over the entire dataset.

Possible reasons for low accuracy: Very few parameters, a small number of epochs, simple architecture that fails to extract features and global aspects, untuned parameters, etc.



Loss Function for the training and test datasets



Why use CuPy and CUDA?

Time taken to run 3 epochs and taking 1000 random images as training dataset from the training batches with NumPy and CPU= 24 minutes

Time taken to run 50 epochs on the entire dataset with CuPy, CUDA aided by GPU = 24 minutes

Speed-up Factor = (time taken without CuPy per epoch)/(time taken with CuPy per epoch)
*(Scaling factor of dataset size)

$$\begin{aligned} &= ((24/3)*60)/((24/50)*60) * (50000/1000) \\ &= (480/28.8) * (50) \\ &= 833.33 \end{aligned}$$

The use of CuPy with CUDA to leverage GPUs can accelerate the process quite effectively as seen.

[Colab Link for Question-1](#) (Please take care to change hardware accelerator to GPU in Notebook Settings and mount data on drive for convenience)

Question-2

3D convolutions were designed primarily for use with voxels. Voxels are simply pixels, but instead of being flat squares, they are a cuboid, adding depth and "volume" to the image. The most common use for 3D convolution is scene reconstruction and autonomous vehicle driving.

Network architecture

This network, like the last one, is made up of **three 3D Convolutional Layers, two Max Pooling Layers, and the ReLU Activation Function.**

Kernel size (3,3,3), stride (2), and padding (1) were used in the convolutional layers.

The **kernel size of the Max Pooling layers was (2,2,1), and the stride was (2,2,1).**


The **third dimension was chosen as 1** since the third dimension for our image (depth) doesn't carry much information, thus it was retained as 1 during the convolution process to conserve the dimension.

Once again, I used CUDA to leverage the GPUs on Google Colab.

Number and type of Layers used

3 Convolutional + 2 MaxPool layers = 5 layers

We can also add Batch Normalisation for speeding up the training and better results.



Let's start with a definition of voxels before thinking about 3D convolution. Consider a multicolored cuboid. Let's say each of these cuboids is divided into n groups, each of which represents a different color gradient. We get RGB for images and now we think of voxels. Convolution is the next step. Let's imagine we have I channels in our input and O channels in our output in a two-dimensional instance. We have one convolution kernel per I kernel for each O channel. As a result, we have a total of $O \times I$ kernels, each of which convolves over its own channel before aggregating the results across all input channels to form the output channel.

Activation functions used

ReLU was used again and it has been explained in the section for Q1.

Loss Function

Categorical Cross-Entropy Loss was used again and it has been explained in the section for Q1. It was implemented using PyTorch this time around.

Learning Algorithm and Rate

The optimization algorithm used is **Adam**. Adam can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. Adam improves on the stochastic gradient method by employing:

- 1) **Adaptive Gradient Algorithm (AdaGrad)**, which maintains a per-parameter learning rate that improves performance on problems with sparse gradients (like the one here)
- 2) **Root Mean Square Propagation (RMSProp)**, which also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing)

In addition to point (2), where the parameter learning rates are adapted based on the mean, Adam also makes use of the variance of the gradients, to give good results.

Here, the **learning rate was initialized to 0.001** and the optimizer took care of the rest.

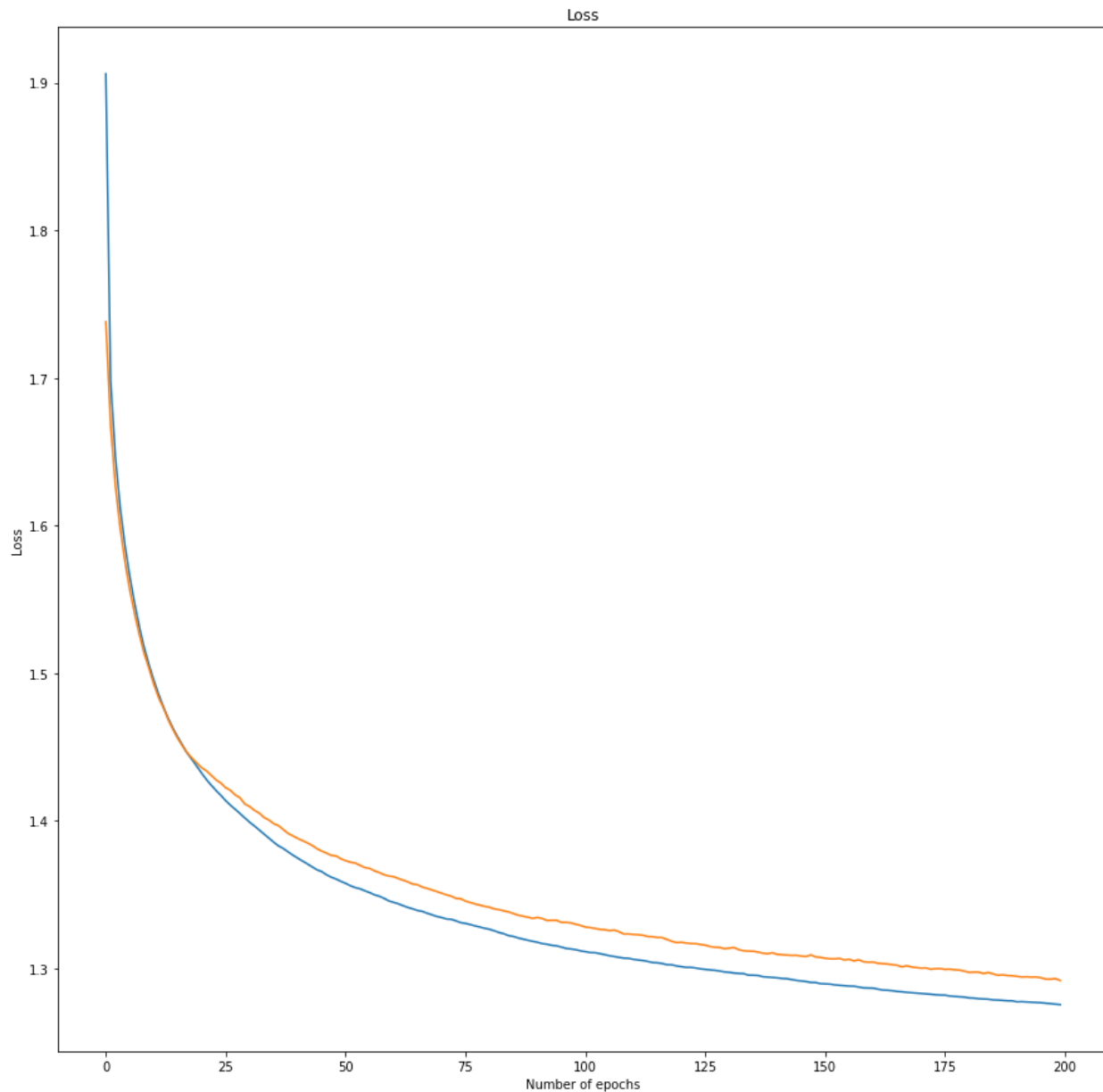
Loss and Accuracy

Batch size = 32, epochs = 200

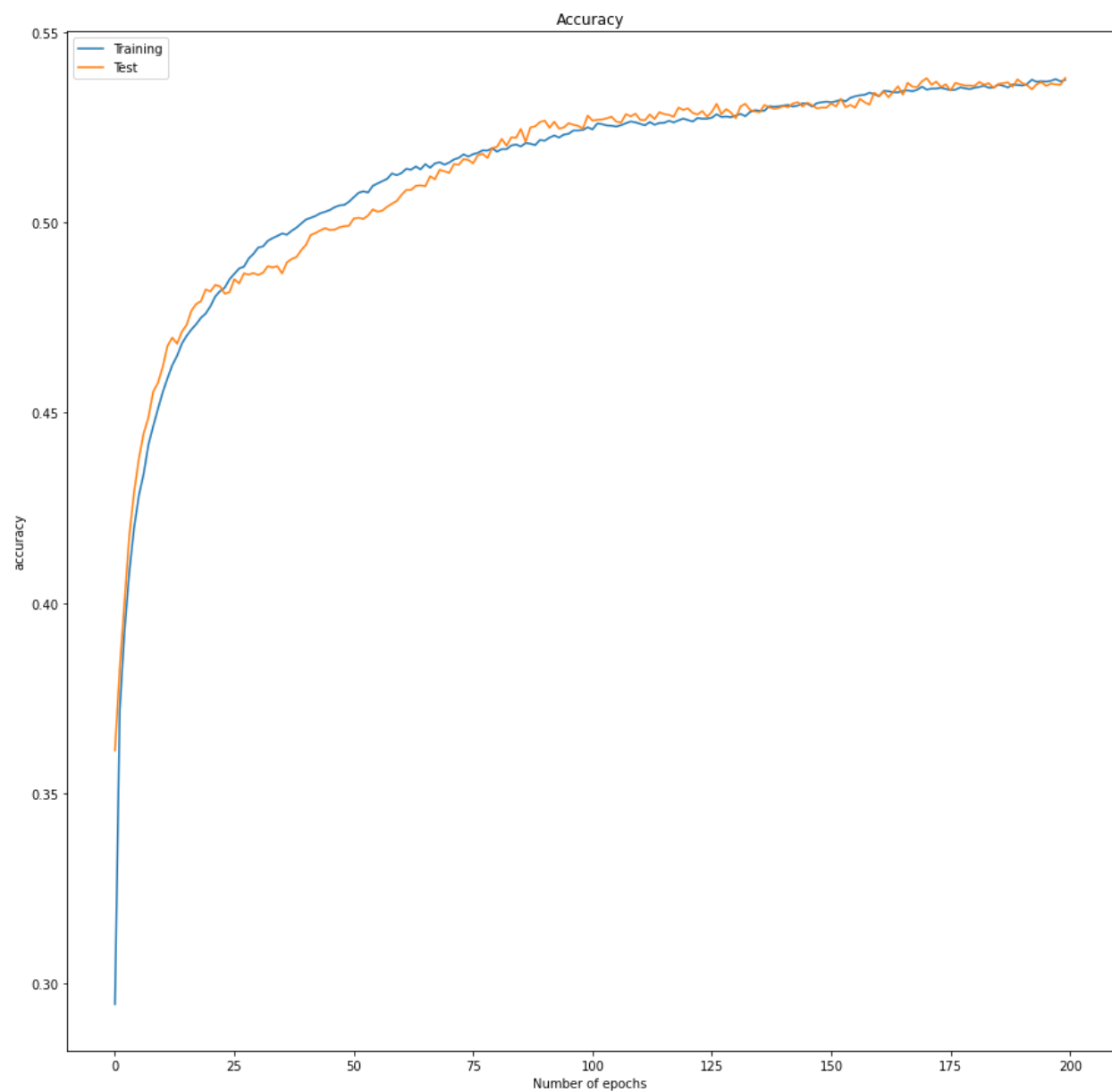
The use of CUDA and GPU ensured that the 200 epochs were run on the entire model in just 34 minutes. Each epoch took around 10-11 seconds to run.

A **mean test loss of 1.252586187362671**, and **mean test accuracy = 54.54%** were obtained when the model was run over the entire dataset.


The improved result might be attributed to the use of a larger number of parameters (around 1 million) to extract features more efficiently from the images and map them with the inputs correctly. However, it must be noted that over-parameterization can lead to over-fitting in some cases.



Loss Function for the training and test datasets



Accuracy for the training and test datasets



[Colab Link for Question-2](#) (Please take care to change hardware accelerator to GPU in Notebook Settings and mount data on drive for convenience)

References

- 1) Course Content of BT3041: Analysis and Interpretation of Biological Data by [Professor V Srinivasa Chakravarthy](#)
- 2) Course Content of [CS6023: GPU Programming](#) by [Dr. Rupesh Nasre](#)
- 3) [CuPy Documentation](#)