Research project – advanced computer architecture

by – Roei Michael 322989666 and Bar Daabul 324079243

Introduction and Motivation

In the modern era of genomic research, the classification of viral lineages has become a pivotal task with considerable clinical implications. The present project encapsulates the development of an algorithm employing Deep Neural Networks (DNNs), particularly Convolutional Neural Networks (CNNs), for classifying lineages of various virus sequences.

The motivation for the endeavor originates from the necessity of a real-time classification system, capable of aiding medical professionals in timely identifying and combating different viruses within a patient. Such an approach could circumvent the need for more complicated and time-consuming examinations.

The urgency and significance of this work were further underscored by the COVID-19 pandemic of 2019-2021, where multiple variants of the SARS-CoV-2 virus required swift identification for effective treatment. Moreover, early detection of the correct viral variant is paramount for managing potential complications such as blood infections.

The project encompasses learning about the structural composition of genes, DNA sequences and the implementation of DNNs for lineage classification. While prioritizing classification accuracy, the approach also emphasizes optimization in terms of learning and prediction time.

Nevertheless, the task is laden with numerous challenges:

- Scale of Data: The dataset is expansive, consisting of numerous lineages with long sequences that aggregate to hundreds of gigabytes of information.
- Complexity of Patterns: The intricacies in DNA sequences necessitate a sophisticated neural network for accurate prediction.
- Speed Requirement: Traditional classification methods are time-intensive, thus accentuating the need for a rapid identification system.

For the construction of the CNN, PyTorch and other associated neural network libraries were deployed, leveraging CUDA for GPU acceleration on NVIDIA hardware.

Preprocessing and Model Design

Data Extraction:

The initial phase of the project involved extracting vital information from a comprehensive MetaData file containing millions of Sequence IDs and lineage correlations. Data were sourced from the COVID-19 Data Portal [1].

A meticulous process was employed to select and organize the data:

- Lineage Selection: All the lineages within the metadata file were iterated and counted. The top 200 lineages were selected to prioritize the classification of the most common and consistent viral variants.
- Data Collection: A directory was created, comprising 250 FASTA file samples for each lineage, culminating in an initial dataset representative of prevalent viral strains.
- Data Encoding and Standardization

The subsequent step entailed the encoding and cleanup of the sequences:

- Base Encoding: The nucleotide bases (A, C, G, T) were encoded using a binary schema (00, 01, 10, 11), while other characters were dealt with using the IUPAC codes [2].
- Sequence Length Adjustment: To maintain uniformity, all encoded sequences were adjusted to an approximate length of 50K characters, determined by the average length across all lineage types.
- Data Structure: The dataset was organized into 200 lineage directories, each containing 250 text files named by sequence ID.

Efficient Data Handling

Considering the immense scale of the data, efficient handling was a paramount concern:

- Dataset Utilization: Utilizing the torch.utils library, each sequence was paired with a file path and label, enabling rapid data extraction.
- Optimization: This approach was favored over creating large CSV files or consolidating sequences into a single file, as it minimized CPU computation and time requirements.

Model Design

The project's model design began with exploratory experimentation. Initially, a simple Neural Network (NN) comprising several fully connected layers (FCL) and an activation layer was constructed. However, this approach was found to be insufficient due to the complexity of the sequences and time-consuming processing.

An intermediate model using a Convolutional Neural Network (CNN) was then implemented, yielding promising results with an accuracy of around 40% on 50 lineage classes. It featured a convolution layer, an activation layer, a max-pooling layer, followed by a second convolution layer, and concluded with an FCL.

Final Model Design - ComplexCNN

The final model, named ComplexCNN, was constructed using PyTorch and encapsulates a series of sequential block layers followed by fully connected layers. Each block layer consists of a Convolutional layer, Batch Normalization layer, ReLU Activation layer, and Max Pooling layer. The architecture is as follows:

Block Layers

- Convolutional Layer: Using a 1D convolution with hidden_size filters, a kernel size of 5, stride of 1, and padding of 2.
- Batch Normalization: Applied to the output of the convolution to stabilize learning.
- ReLU Activation
- Max Pooling: Performed with a kernel size of 2 and a stride of 2.

The subsequent layers (Layer 2 to Layer 5) followed a similar pattern, with each layer doubling the number of filters compared to the previous layer (i.e., hidden_size, hidden_size * 2, hidden_size * 4, etc.). This gradually increased the complexity of features being captured.

Fully Connected Layers

After the block layers, the output was flattened and passed through the fully connected layers:

- First Fully Connected Layer (fc1): Linear transformation with an input size calculated based on the final hidden size and the successive application of max pooling layers.
- Dropout Layer: Applied with a rate of 0.5 to prevent overfitting.
- Second Fully Connected Layer (fc2): Linear transformation leading to the final output with num_classes neurons.

Convolutional Layer: Detects local patterns or motifs in DNA sequences.

Batch Normalization Layer: Stabilizes and accelerates learning by normalizing inputs.

ReLU Activation Layer: Introduces non-linearity, enabling complex mappings from inputs to outputs.

Max Pooling Layer: Reduces dimensionality, summarizing features and reducing overfitting risk.

Fully Connected Layer (FCL): Interprets extracted features for final classification.

Dropout Layer: Prevents overfitting by randomly disabling a fraction of input units during training.

Hyperparameter Tuning:

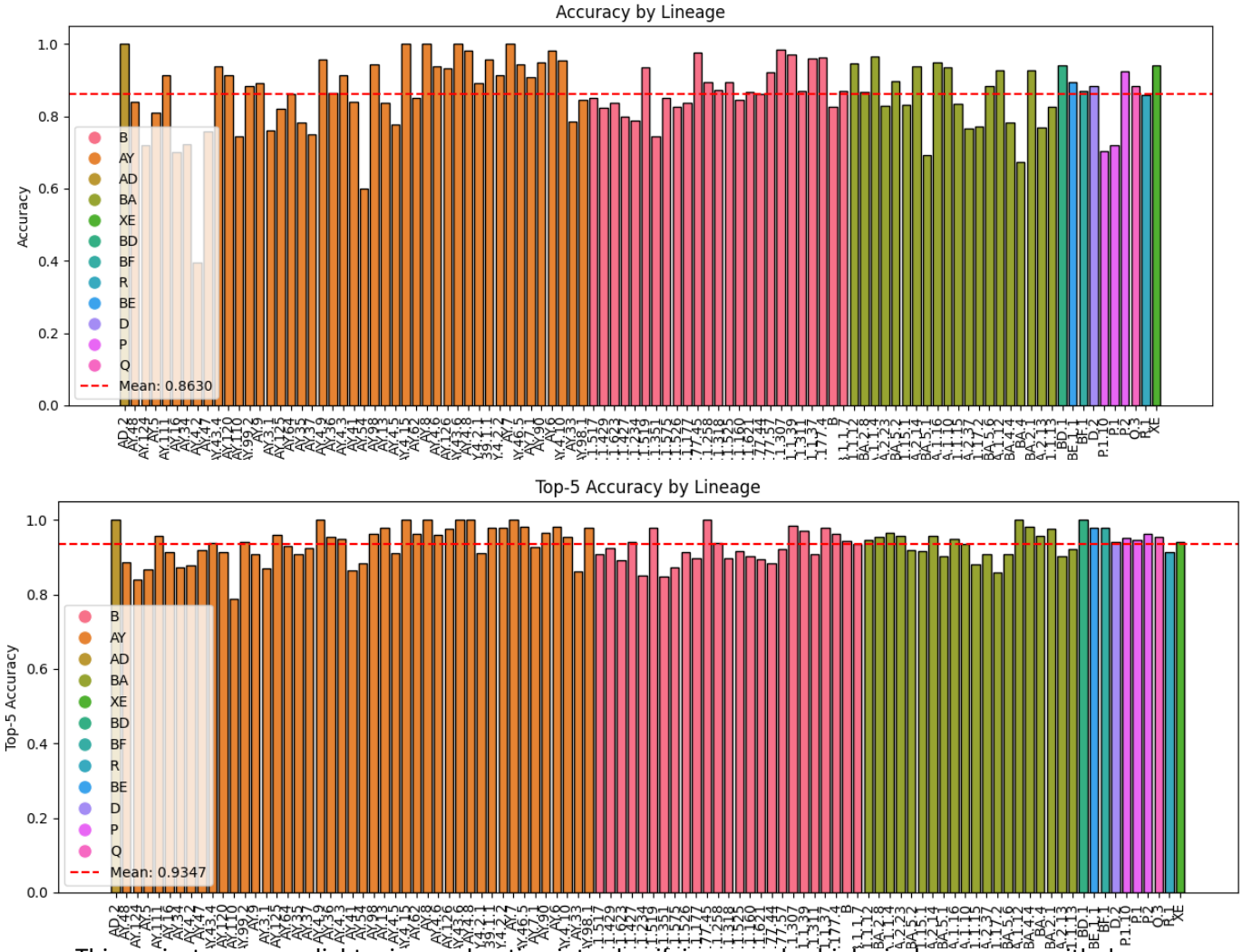
The final model was carefully optimized through extensive hyperparameter tuning, experimenting with variables such as learning rate, dropout rate, hidden size, number of epochs, weight decay, and batch size.
The gradual increase in complexity through the block layers, combined with the thoughtful arrangement of fully connected layers, exemplifies a design suited to the intricate and nuanced task at hand.

Results and Conclusion

Utilizing the complex convolutional neural network model described previously, in addition to taking 100 classes with 250 samples for each class being split into 20-80 ratio of train and test data - the system was rigorously trained using a specific set of hyperparameters. The selected parameters were a batch size of 400, a hidden size of 32, 500 epochs, the cross-entropy loss function as the criterion, and the Adam optimizer with a learning rate of 10^{-5} and weight decay of 10^{-5}

Upon testing the model against the designated dataset, a promising accuracy of 86% was achieved, and top-5 accuracy of about 93%. To provide a comprehensive evaluation of the model, the results were detailed into a summary file encompassing metrics such as accuracy, precision, the number of correct predictions, top-3 accuracy, and top-5 accuracy for each of the 100 lineages assessed.

Certain constraints related to the computational resources, specifically GPU memory size and runtime, necessitated adaptations in the selection of hyperparameters and the number of lineages. Initially trained on local computers, the project was later migrated to the university servers equipped with 4 NVIDIA 1080 Ti GPUs, enabling faster processing but still took quite some time to train the models.



This project was an enlightening exploration into classifying complex biological sequences with deep neural networks. Key findings include:

- Computational limitations: The complexity of the sequences and constraints on computing resources limited the quality of the results, despite a pretty good accuracy.
- Optimization success: Through the effective use of multithreading, optimization libraries, and multi-GPU computing, we achieved drastic reductions in processing time.
- Learning experience: The project provided valuable insights into neural networks, optimization, and computational challenges, laying a solid foundation for future exploration in these domains.