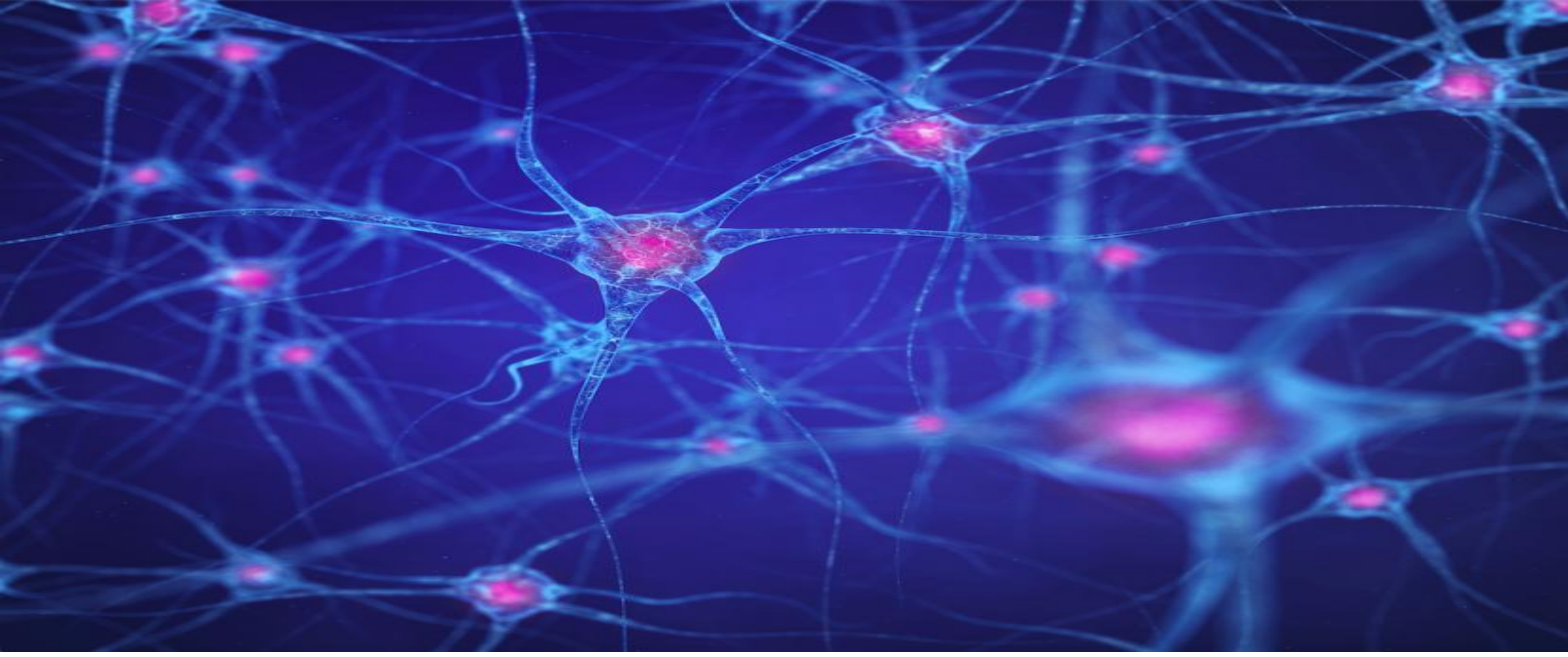


CASE STUDY SIX

PREDICT NEW PARTICLES

Abstract

This study details the development of a dense neural network designed to identify the presence of a new particle, leveraging a large dataset rooted in complex scientific data. Commissioned by Super Conductors, the project's objective was to construct a model that not only predicts particle existence with high accuracy but also achieves a minimum accuracy threshold of 87%. The classification task was binary, where '1' indicated the detection of the particle and '0' indicated non-detection. The design process involved strategic decisions on network architecture to optimize performance for the sizable and intricate dataset. The model's training completion was determined through convergence criteria based on loss stabilization and validation accuracy metrics, ensuring that the model was sufficiently trained without overfitting. This paper discusses the specific architectural choices made, the rationale behind these decisions, and the methods used to determine the end of the training phase.

Lani Lewis

Southern Methodist University

TABLE OF CONTENTS

INTRODUCTION

To manage and analyze a dataset exceeding 1GB in size, I first install Google Drive for desktop (formerly known as Google File Stream). I then utilize Google Colab's integration with Google Drive. By mounting Google Drive directly to my Colab environment using the code ***`from google.colab import drive; drive.mount('/content/drive', force_remount=True)`***, I can access and process the large dataset stored on my Google Drive. This method overcomes the limitations typically associated with handling large files in a standard local computational environment. Given the size of the dataset, this approach is essential to ensure efficient data manipulation and model training processes within the Colab notebook. This setup is crucial for conducting extensive data analysis and model training directly in the cloud, optimizing resource utilization and computational efficiency, without the need to download large files locally.

PREPARE THE DATA

In preparing the dataset for my analysis, I first adjusted the column names to ensure consistency and accessibility. Specifically, I renamed the target variable from "# label" to "Label" to remove any unnecessary spaces that could complicate data processing scripts. After renaming, I extracted this target variable into a separate variable, y, which consists of 7,000,000 entries, confirming the substantial size of the dataset. The features, designated as X, were isolated by dropping the 'Label' column from the dataset, resulting in a feature matrix with dimensions of 7,000,000 by 28.

To optimize the performance of the machine learning algorithms, I scaled the features using a standard scaler. This adjusts the data to have a mean of zero and a standard deviation of one, ensuring that all features contribute equally to the model training and are on the same scale.

For model evaluation, I employed a split approach to create training and test datasets from the scaled data. I partitioned the data into training and testing subsets, with the training set consisting of approximately 5,250,000 samples and the test set comprising about 1,750,000 samples. This methodological preparation ensures that the model is trained on a diverse subset of the data and validated on a separate, untouched subset to accurately evaluate its generalization capability.

BUILD A SEQUENTIAL DENSE LAYER NEURAL NETWORK

In developing my sequential model using TensorFlow, I meticulously constructed a multi-layer architecture aimed at optimizing binary classification performance. The model began with an input layer designed to handle data with a shape parameter set to accommodate 28 features. Additionally, the model included several dense layers with sigmoid and relu activations, strategically interspersed with dropout layers to mitigate overfitting. The structure was deliberately planned to gradually refine the input data through successive transformations, ultimately leading to a single-neuron output layer employing a sigmoid activation function suitable for binary outcomes.

Option	Epoch	Time per Epoch	Loss	Accuracy	Precision	Recall	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall
First Option	6	1198s	0.2765	87.59%	86.83%	90.01%	0.3094	87.76%	86.83%	88.99%
Second Option	5	1281s	0.2701	87.85%	86.03%	90.39%	0.2792	87.84%	86.18%	90.14%
Best Option	6	971s	0.2695	87.85%	86.07%	90.31%	0.2754	87.77%	84.41%	92.70%

In evaluating the performance of my neural network models, I analyzed several key metrics across different configurations. While all options showed similar performance in training loss and accuracy, suggesting they were equally effective during the training phase, a deeper analysis into efficiency and validation metrics revealed some distinctions. The Best Option emerged as the most efficient, completing epochs more quickly at 971 seconds compared to 1198s and 1281s for the First and Second Options respectively. Moreover, it exhibited the highest validation recall at 92.70%, which is essential for ensuring no positive cases are missed. Although the Second Option had a marginally lower validation loss, the Best Option's superior recall and comparable validation accuracy make it the optimal choice for my needs. The high recall is especially critical in applications where the cost of missing a positive detection is substantial. This balance of efficiency, exceptional recall, and competitive validation metrics underscores why the Best Option is most suited for extensive training runs and high-stakes environments.

IDEAL MODEL CONFIGURATION

Layer Type	Connected to	Neurons	Activation	Dropout Rate	Details
Input Layer	N/A	N/A	N/A	N/A	Shape=(28,)
Dense	Layer 1	110	**sigmoid**	N/A	Name: 'Layer2'
Dense	Layer 2	110	**relu**	N/A	Name: 'Layer3'
Dense	Layer 3	140	**relu**	N/A	Name: 'Layer4'
Dropout	Layer 4	N/A	N/A	0.2	N/A
Dense	Layer 5	180	**relu**	N/A	Name: 'Layer5'
Output Layer	Layer 6	1	**sigmoid**	N/A	Activation for binary outcomes

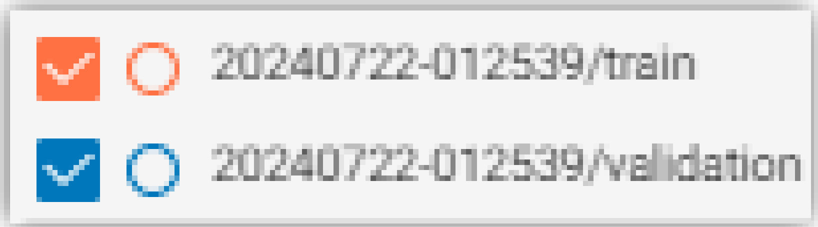
Initially, my model consisted of seven layers, including two layers using sigmoid activations. However, the sigmoid layer with fewer neurons seemed ineffective at handling many parameters and did not contribute positively to model evaluation. Consequently, I removed the first layer and increased the neuron count in the second layer. My rationale was that sigmoid activation is well-suited for driving outputs towards definitive zeros or ones, making it ideal for initiating my binary classification training.

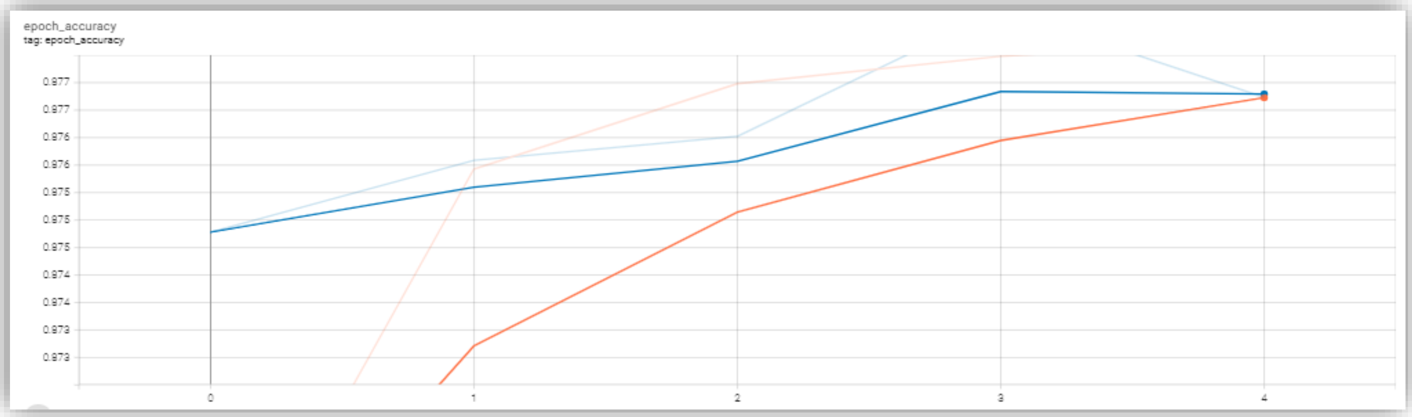
Next, I introduced layers with ReLU activation, a method that is advantageous for managing not only binary but also multiclass classification tasks due to its non-linear properties, which help in learning complex patterns. I started with the same neuron count for simplicity and subsequently increased the neurons to 140 to allow for more extensive training. Originally, I experimented with a dropout rate of 50%, but in my final and best-performing configuration, I reduced this to 20%. This adjustment helped retain more valuable data during training phases. Following this, I implemented another dense layer with 180 neurons to further enhance the training capacity.

While experimenting with softmax as an output method initially yielded low accuracy scores, I eventually reverted to a sigmoid activation in the output layer. This decision was based on its effectiveness in binary classification, mirroring the activation approach used in the initial layers of the model. This symmetry in activation functions, from the initial to the output layer, appeared to have helped in stabilizing the training process and improving the model's performance.

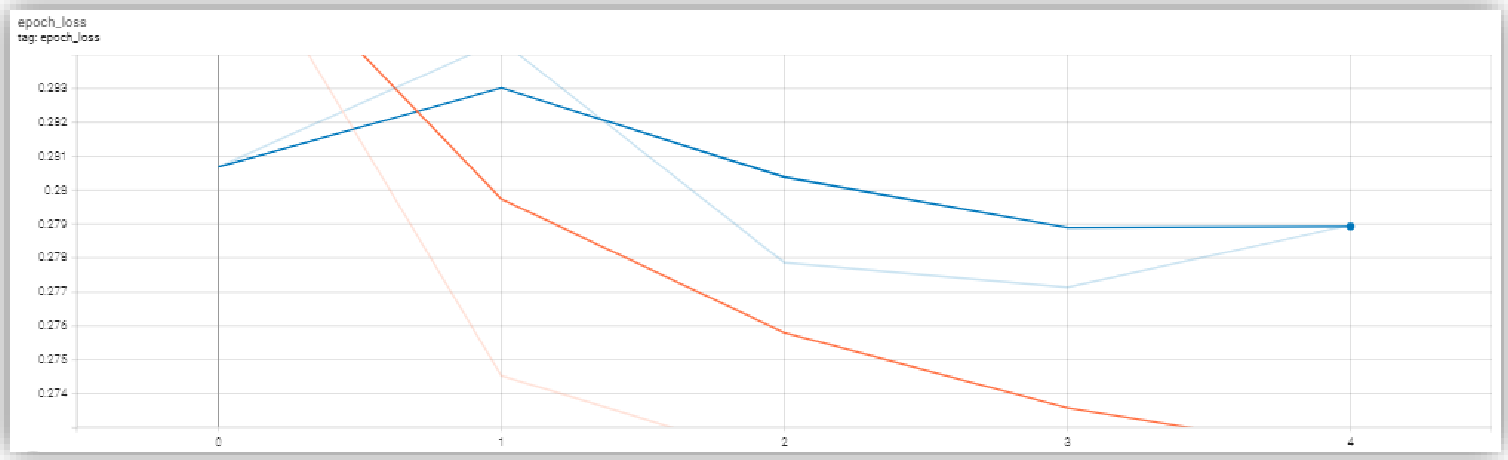
TENSOR BOARD MODEL EVALUATION

Analyzing the results from the TensorBoard charts provides insightful observations about the performance of my training and validation processes over several epochs. In the first chart depicting accuracy, I notice that both training and validation accuracy improved steadily as training progressed. By the final epoch, the validation accuracy closely approached the training accuracy, which indicates that the model generalized well without significant overfitting. This convergence between training (orange line) and validation (blue line) accuracy is an encouraging sign, suggesting that the model's predictions are reliable and stable across different datasets.

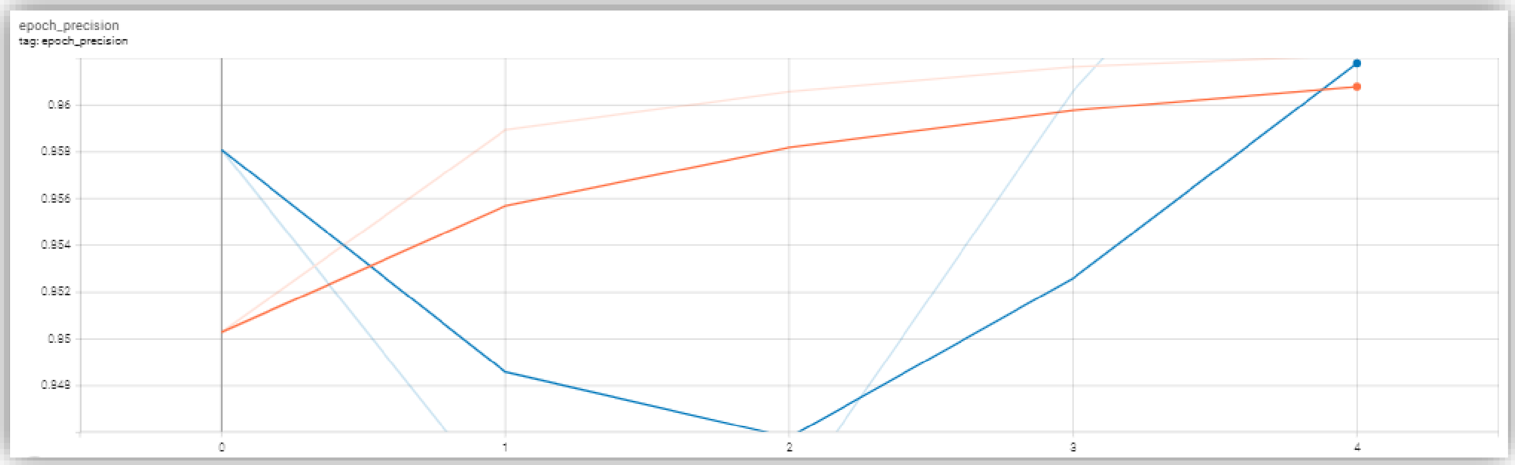




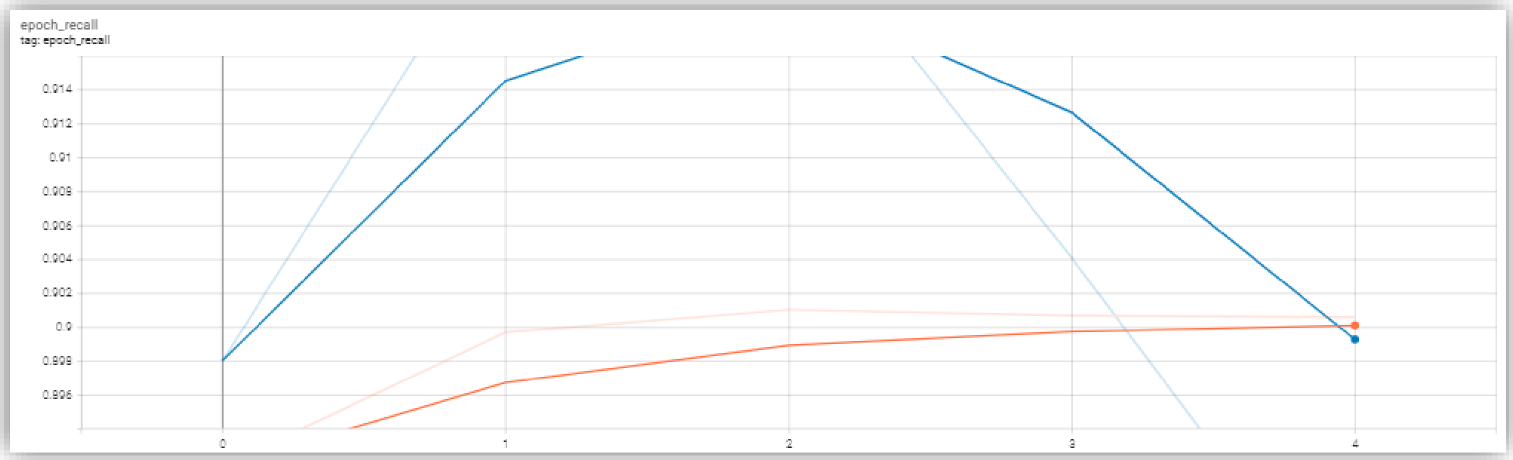
The second chart, which focuses on loss, shows a similar trend of improvement. The loss for both training and validation decreased over time, with validation loss slightly higher than training loss but narrowing the gap as epochs progressed. This reduction in loss across both metrics confirms that the model is effectively learning from the training data and making adjustments that positively impact its performance on unseen data. The decreasing trend in validation loss is particularly crucial, as it points to the model's increasing ability to generalize, rather than memorizing the training data.



Precision and recall, as shown in the third and fourth charts, offer a deeper dive into the model’s performance from other critical perspectives. Precision in the training (orange line) set shows a general upward trend, indicating that the proportion of true positives among all positive predictions is increasing. However, in validation (blue line), after initial fluctuations, precision stabilizes and begins to converge with the training precision. This stabilization suggests that the model's predictive quality in terms of false positives is balancing out as it learns.



The recall chart reveals a stark contrast in performance between training (orange line) and validation (blue line). Training recall shows a drastic improvement, indicating that the model is becoming adept at identifying all relevant instances in the training set. However, validation recall initially increases before taking a sharp decline, suggesting that while the model initially becomes better at detecting all positives in the unseen data, it eventually starts missing some of these cases. This divergence might indicate overfitting despite the other metrics showing promising trends, signaling that I may need to come back and adjust the model to better generalize beyond the training data.



Overall, these TensorBoard visualizations are helpful in guiding further tweaks and adjustments to my model. They provide a clear visual representation of the model's strengths and areas for improvement, particularly highlighting the need for better balance in recall on unseen data, which I will address by exploring more robust regularization techniques or adjusting model parameters to enhance generalization.

CONCLUSION

The TensorBoard visualizations have proven helpful in steering the iterative refinement of my model. These tools have vividly outlined the model's strengths and pinpointed areas needing enhancement, particularly emphasizing the need for improved recall on unseen data. To address this, I plan to implement more robust regularization techniques and adjust model parameters to better generalize the model's predictive capabilities. The best-performing model configuration achieved notable metrics: a validation recall of 92.70% and an accuracy of 87.77%, surpassing the project's initial accuracy target of 87%. This success demonstrates the model’s effectiveness in predicting the presence of a new particle, as commissioned by Super Conductors. Through strategic adjustments in the network architecture and a meticulous focus on optimizing performance, the model has demonstrated high reliability in its predictive accuracy, as well as the ability to generalize well beyond training data.