Search Term Classification For Google Ads Using AI

# Objective

The objective of this experience is to find potential solutions to decrease the time spent classifying search terms on Google Ads, giving the manager of the accounts more time to focus on deep optimizations such as copy writing and keyword research.

# Challenge Context

Depending on the location and budget of an account, it might take a PPC manager over 1 hour per account to fully check the search terms in an account.

In order to decrease the time of the task, it is common practice to limit the search terms analyzed only to those that received clicks during a given time period. However, this approach is not comprehensive enough to achieve tight targeting and might allow future irrelevant searches to spend money in the future. Additionally, this approach might hide opportunities to add keywords that could increase the lead flow of an account if actively targeted.

By addressing the issue, over 15 work hours per month could be unlocked, allowing to increase the accounts under management per team member and ultimately resulting on a revenue increase thanks to the extra capacity.

# The experiments

We considered multiple AI models to solve this issue, starting at the least technically difficult and most computationally inexpensive techniques. By the end, I had tested Decision Tree Models, Shallow Neural Networks and the Hugging Face implementation of the roBERTa model.

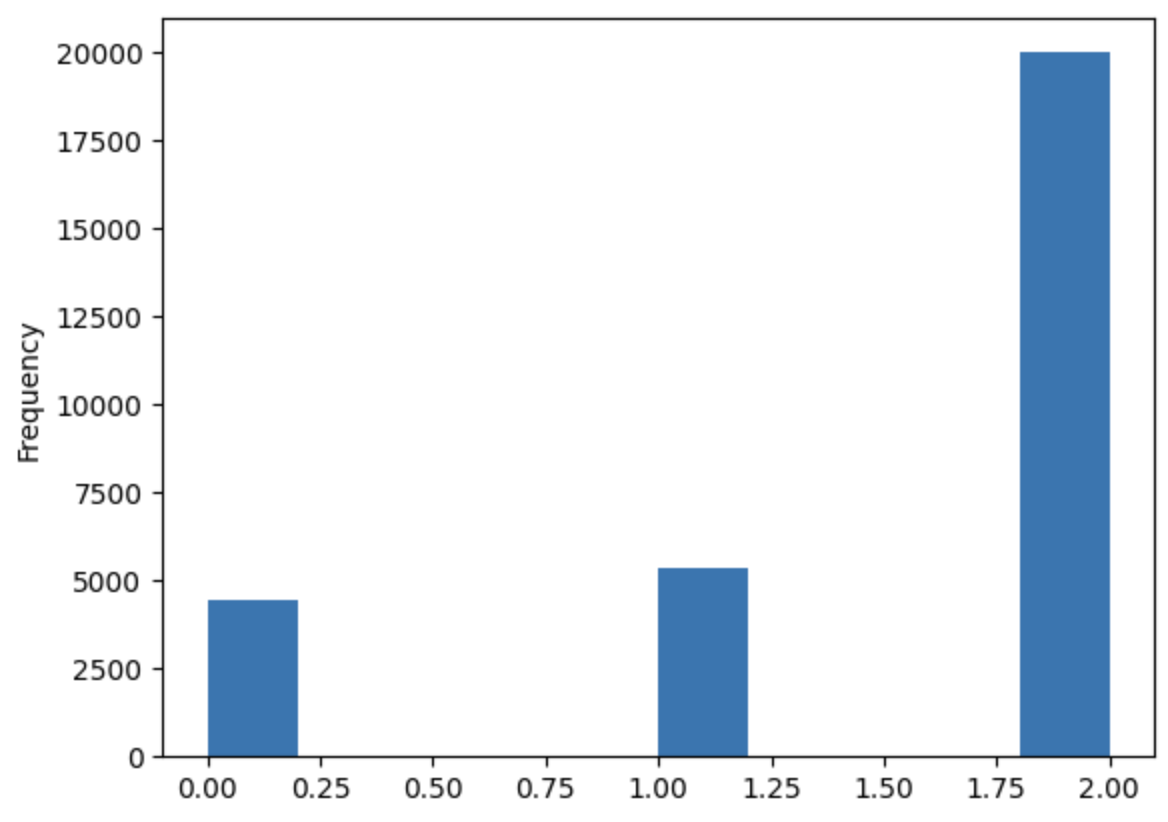
## Decision Trees and Shallow Neural Networks

In order to use these techniques, we needed to create a dataset that would allow us to compare the semantic similarity between the keyword and the search term (Similarity Score). To get this measurement, we used sentence embeddings and applied cosine similarity to get the similarity between both terms. We implemented the [SentenceTransformers library](https://www.sbert.net/) to get the required transformers and other mathematical tools.

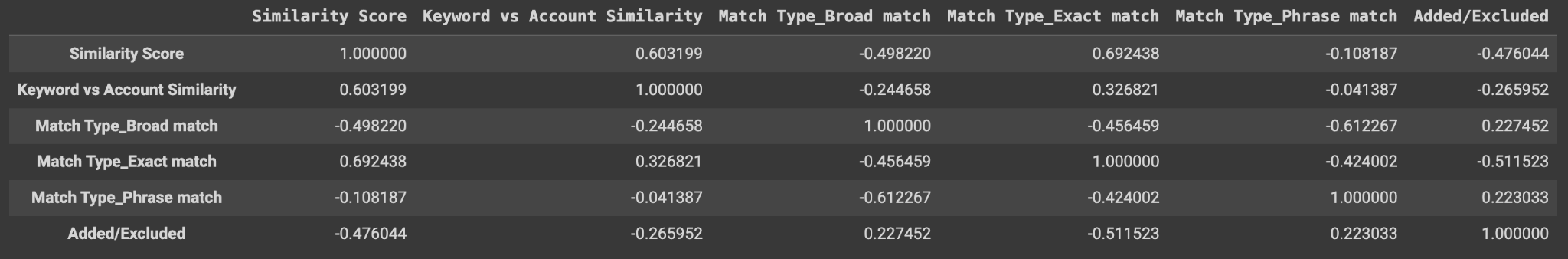
Additionally, we used hot-one encoding to turn into numerical information the match type of a keyword (Exact, Phrase and Broad). This information was essential as different match types are more likely to attract out-of-target searches than others.

As an additional data point, we also got the average similarity of a term with all the keywords in an account (Keyword vs Account Similarity). This should act as a proxy for the contextual relevance of a search term, that might not be directly related to the keyword but which is still relevant based on the advertised service of the account.

The dataset was not balanced, with close to 20,000 rows labeled as “None” (2) and merely 10000 labeled as either “Excluded” (1) or “Added” (0).

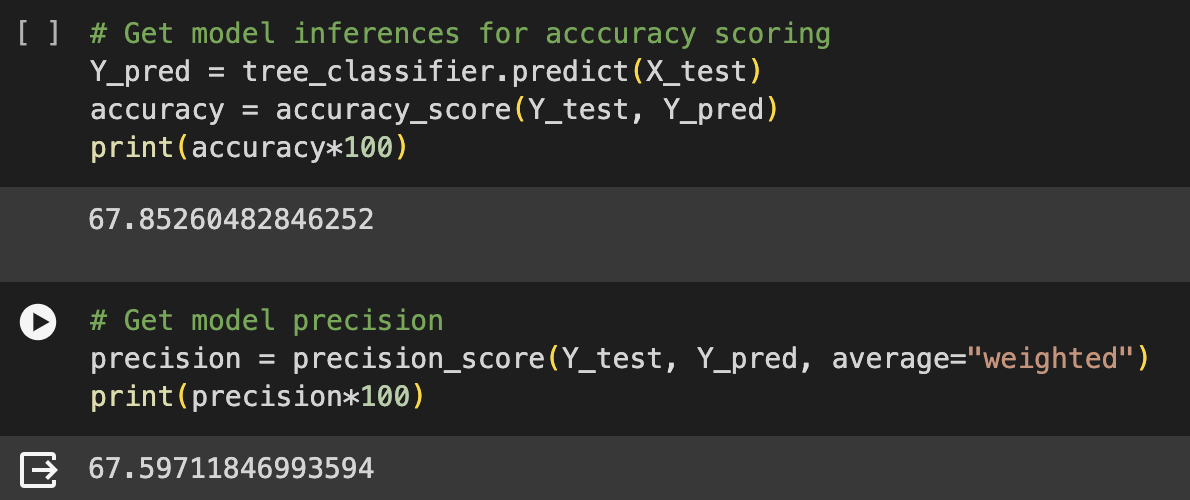


Moreover, we analyzed the correlations between the variables to check if the data points were indeed relevant and made sense based on empirical experience.

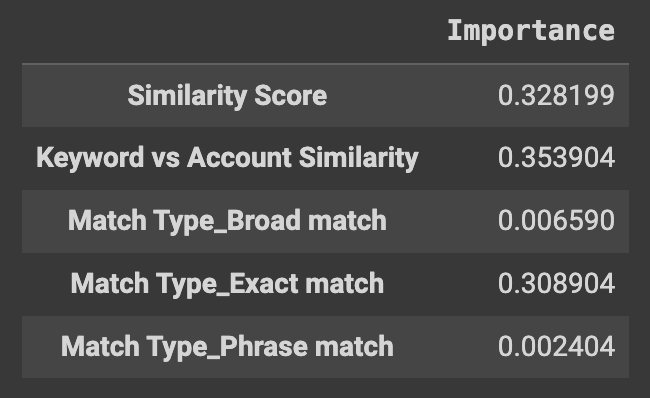


Once the dataset was created, we trained a Scikit Learn DecisionTreeClassifier model. As measurements for performance, we used accuracy and precision, since we need the model to predict as many correct labels as possible while doing so consistently.

We used multiple iterations of the decision tree, comparing the performance using different criterions (gini, log\_loss, entropy) and class weights. The gini criterion without class weights seemed to perform better than any other iteration, however the performance difference was not big enough to consider it statistically significant. Regardless of the depth of the tree and the number of leaves, the model always achieved roughly 67% accuracy and precision.

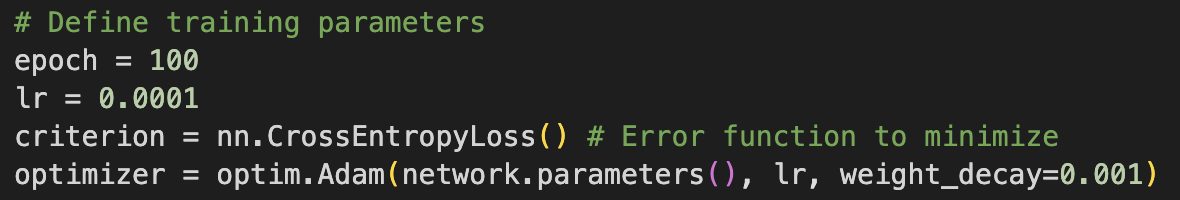


However, it is important to highlight that the model gave reasonable importance to the variables in the dataset. The model is correct in understanding that a keyword that has a higher Similarity Score and Keyword vs Account Similarity, as well as being matched to an Exact Match keyword, is more likely to be included.

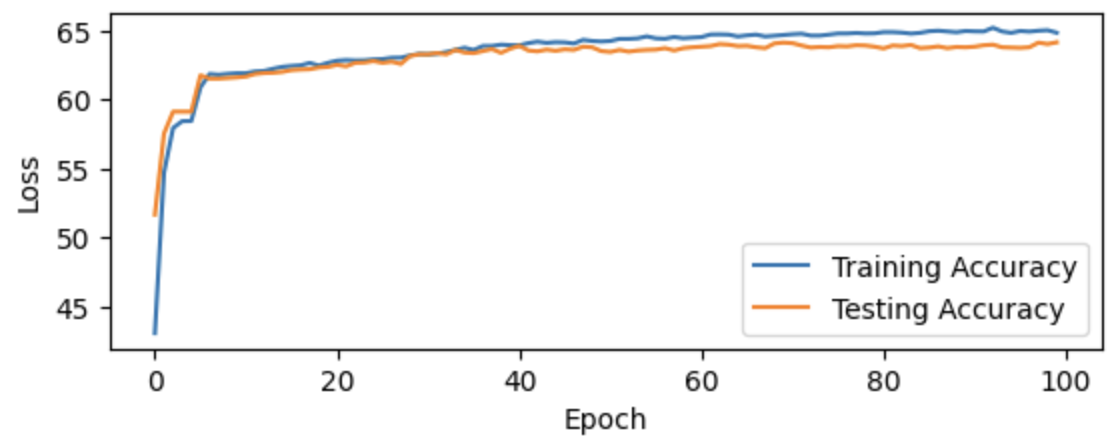


Considering that the accuracy and the precision of the model were not high enough to reliably deploy a decision tree in production, we then proceeded to train a neural network on the same dataset.

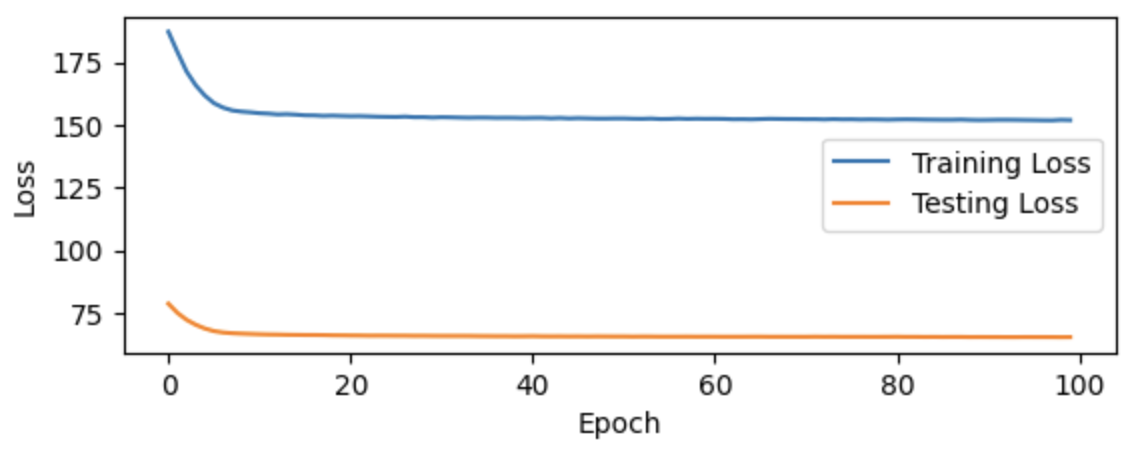
Using Pytorch, we created a neural network with 5 layers with ReLU activations for the outputs of the input and hidden layers and softmax for the output layer. We then proceed to train the model over 100 epochs with the following hyperparameters:



Once the model was trained, we got performance similar to the decision tree, with accuracy at about 64% in both training and validation sets.



The model did seem to get stuck early in the training process, meaning either low learning or hitting a local minima.



Analyzing the plateau in accuracy and similar performance from the decision tree, it is very possible that the model was not able to learn from the data either due to the imbalance of the dataset classes or simply because the data does not show significant enough correlations to accurately predict the results.

With the above considerations in mind, we decided this approach was not scalable due to the amount of preprocessing it required. For each term, we needed to turn every term-keyword pair into a transformer and compute the cosine similarity between the pair and between the term and all the keywords in the account, making it a very slow and expensive process.

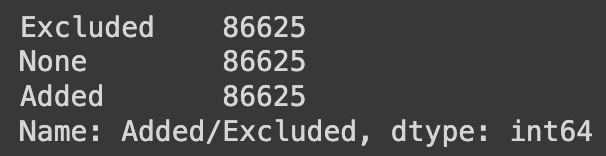
You can check the relevant Jupyter Notebook [here](https://colab.research.google.com/drive/1KRBubHIilbGzL4kibrtRh5p2w_zDmKYJ?usp=sharing).

## Hugging Face’s roBERTa Implementation

Understanding that we needed to keep preprocessing to a minimum, we then decided to use a BERT implementation that would allow us to simply input the term-keyword pair and get a label.

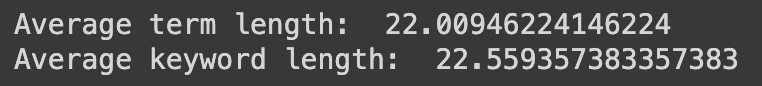
To achieve this, we turned to [roBERTa for Sequence Classification](https://huggingface.co/docs/transformers/model_doc/roberta#transformers.RobertaForSequenceClassification). With a relatively easy implementation, this was the most likely model to perform well in production for a language processing task.

Since roBERTa is a deep model, we needed to improve the quality of our training data. For that, we improved the frequency balance of the dataset and increased the number of samples. In total, our new dataset has 259,875 samples with exactly 86,625 samples for each of the classes (Added, Excluded, None).



Secondly, because roBERTa uses tokenization to process input data, we also removed all special characters that keywords might contain. By removing the [], +, and “” from all the keywords, this allows the model to focus exclusively on the semantics of the keyword and not its match type.

Thirdly, considering that the tokenization process needs to cap the length of the tokens for better results, we had to calculate the average keyword and search term length to choose a limit.

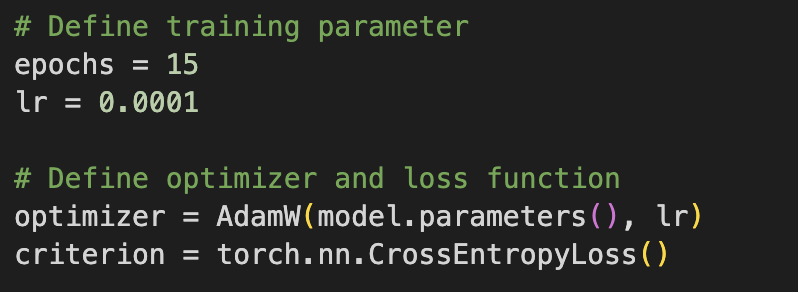


To create the required DataLoaders for the process, we created term-keyword pairs and concatenated both tokenized inputs with a special character to let the model understand that the tensors contained more than one piece of information. We also added a special CLS token, necessary for the model to gather sentence-wide data and classify the pair appropriately.

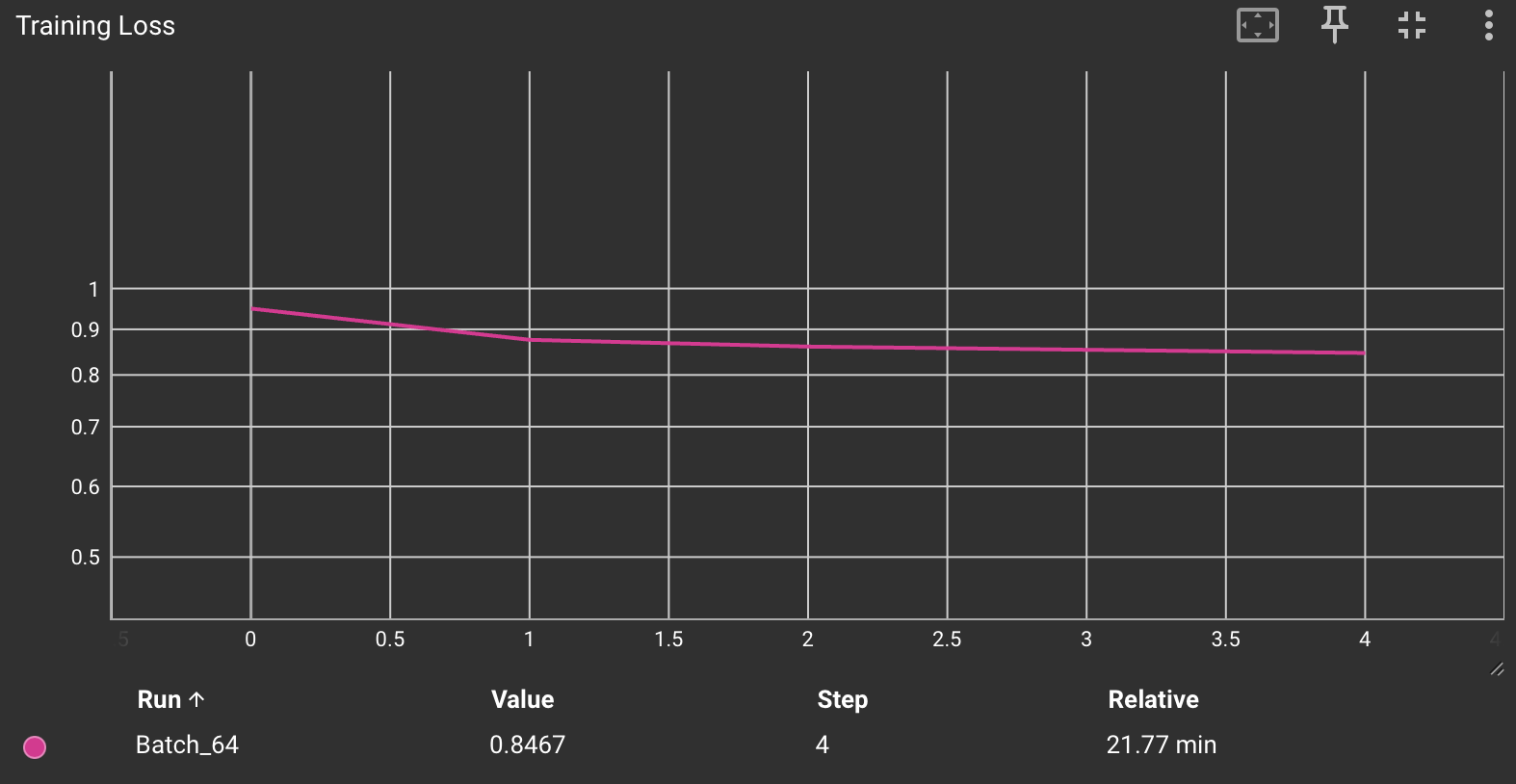
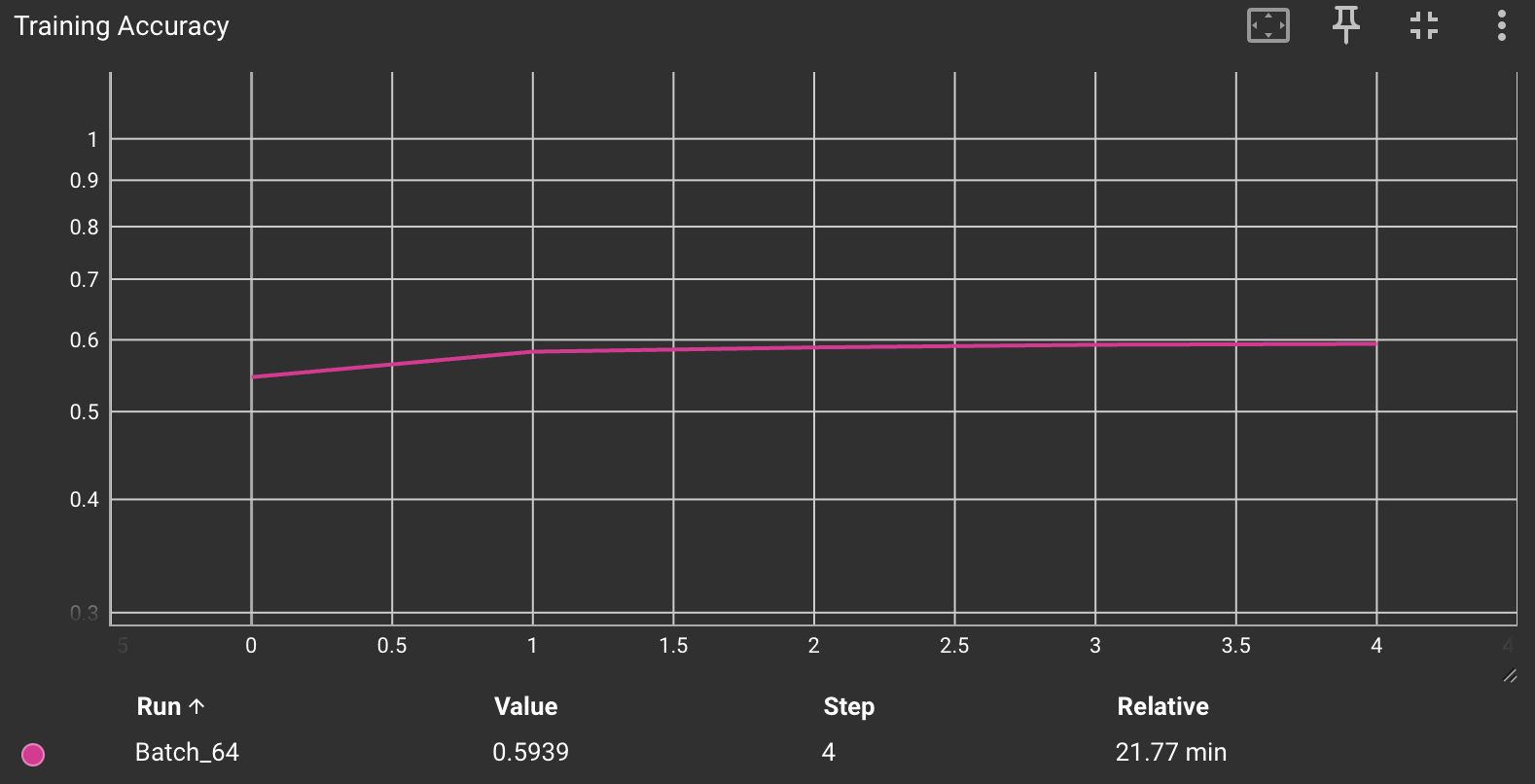
Once the tokenized data was ready, we downloaded a roberta-base model with the pretrained parameters. In order to improve training speed, we froze all parameters except those in the classifier head. This allows for the training to be much faster as only 1 dense and a simple output layer need to be trained.

### First Training Iteration - Batch Size 64

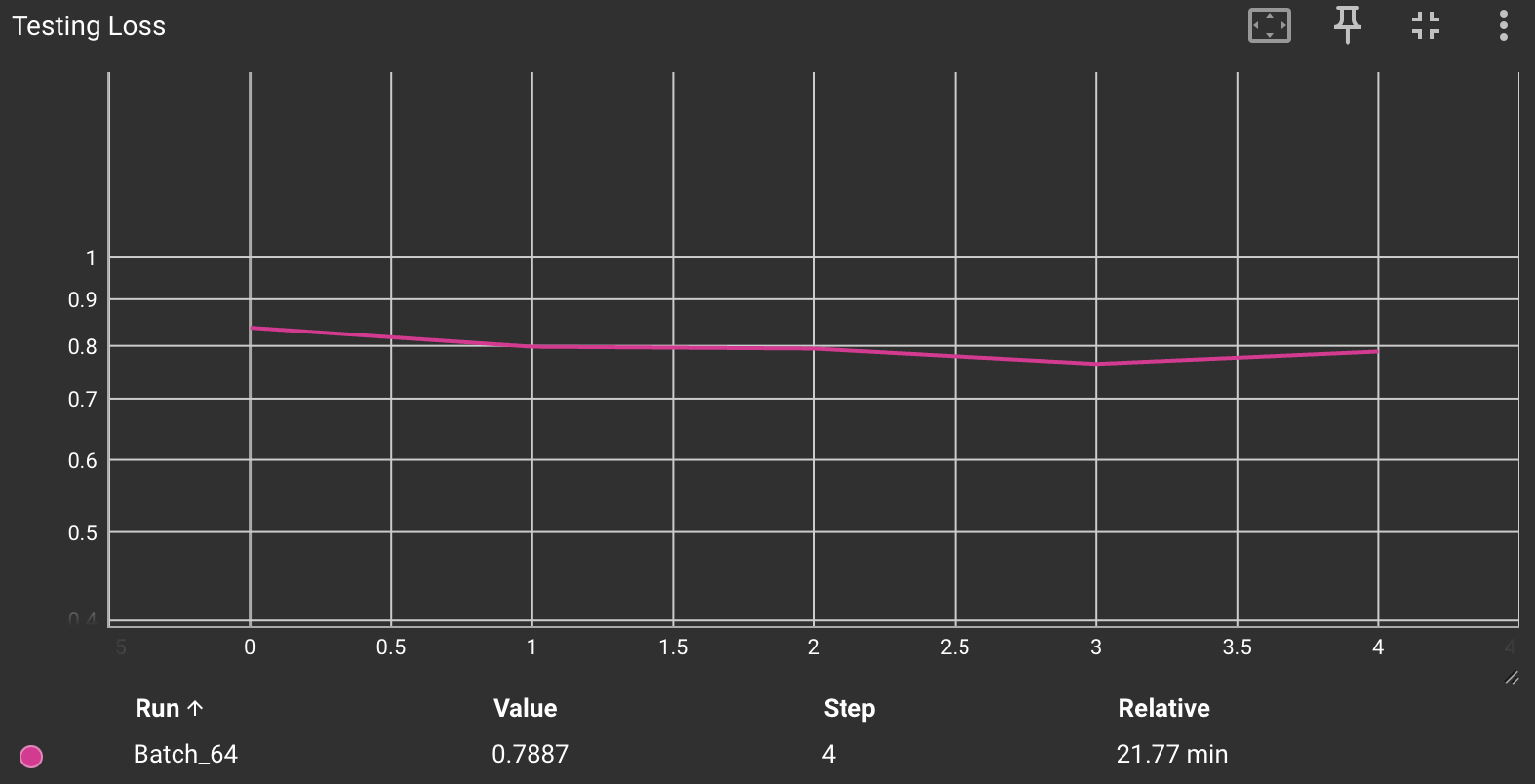
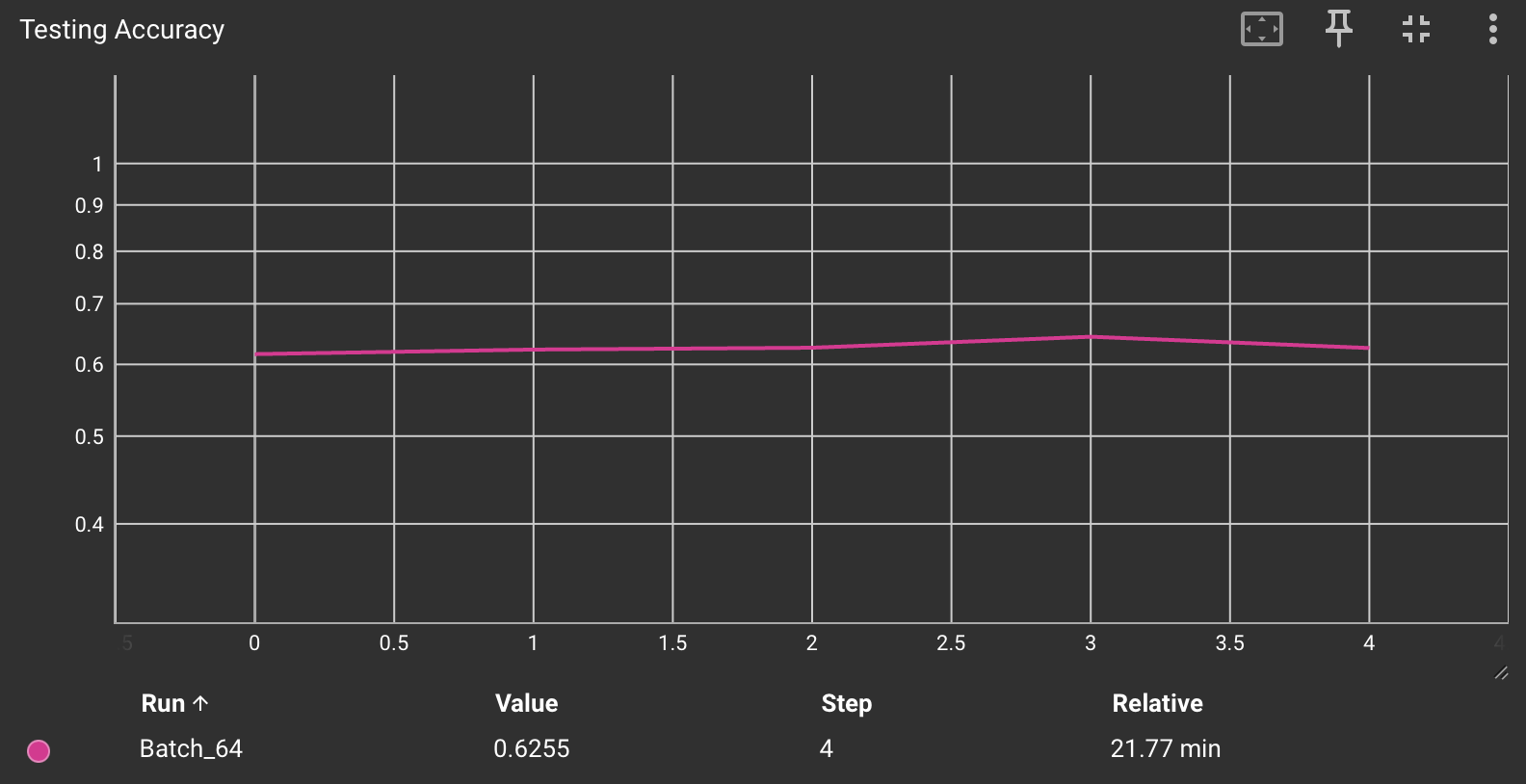
For this iteration, we used a batch size of 64. The following hyperparameters were used for training the model:



After the initial training, the results were not as accurate as the decision tree or the neural network. However, keep in mind that the results of these models were calculated using a much smaller dataset and might not be fully comparable.



Training Loss and Accuracy (Max Accuracy=59%)



Testing Loss and Accuracy (Max Accuracy=64%)

In order to improve accuracy, we decided to test using different batch sizes and changing the model classifier to provide more depth to the architecture. The results of these variations can be seen below.

### Second Training Iteration - Smaller & Bigger Batch Size

For the second iteration of the model, we compared both a smaller and larger batch size (16 and 512). With this we wanted to find a batch size that allows the model to generalize the data more effectively and see what approach would yield higher accuracy in the least number of training epochs.

Even though the accuracy was higher in both testing and training for the model using a batch size of 16, there was a steeper improvement in performance for the model with a batch size of 512. Considering this finding, we will use a larger batch size with a custom classifier head and longer training time for the next iteration.

### 

Training Loss and Accuracy (Max Accuracy=65% - Batch 16)

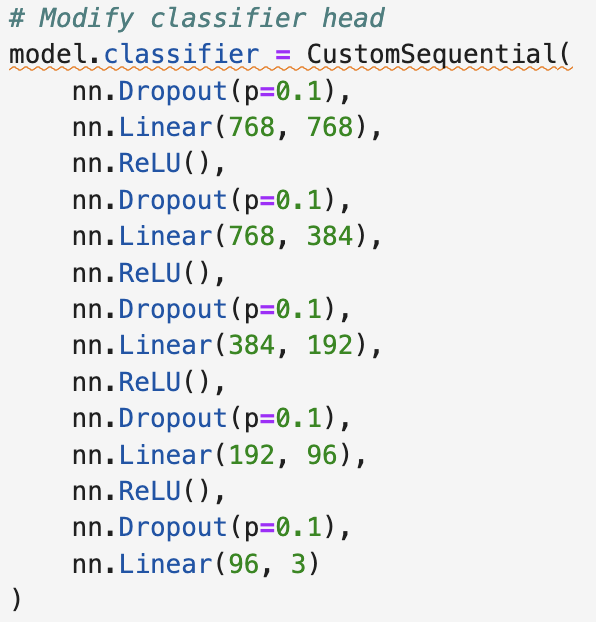
### 

Testing Loss and Accuracy (Max Accuracy=59% - Batch 16)

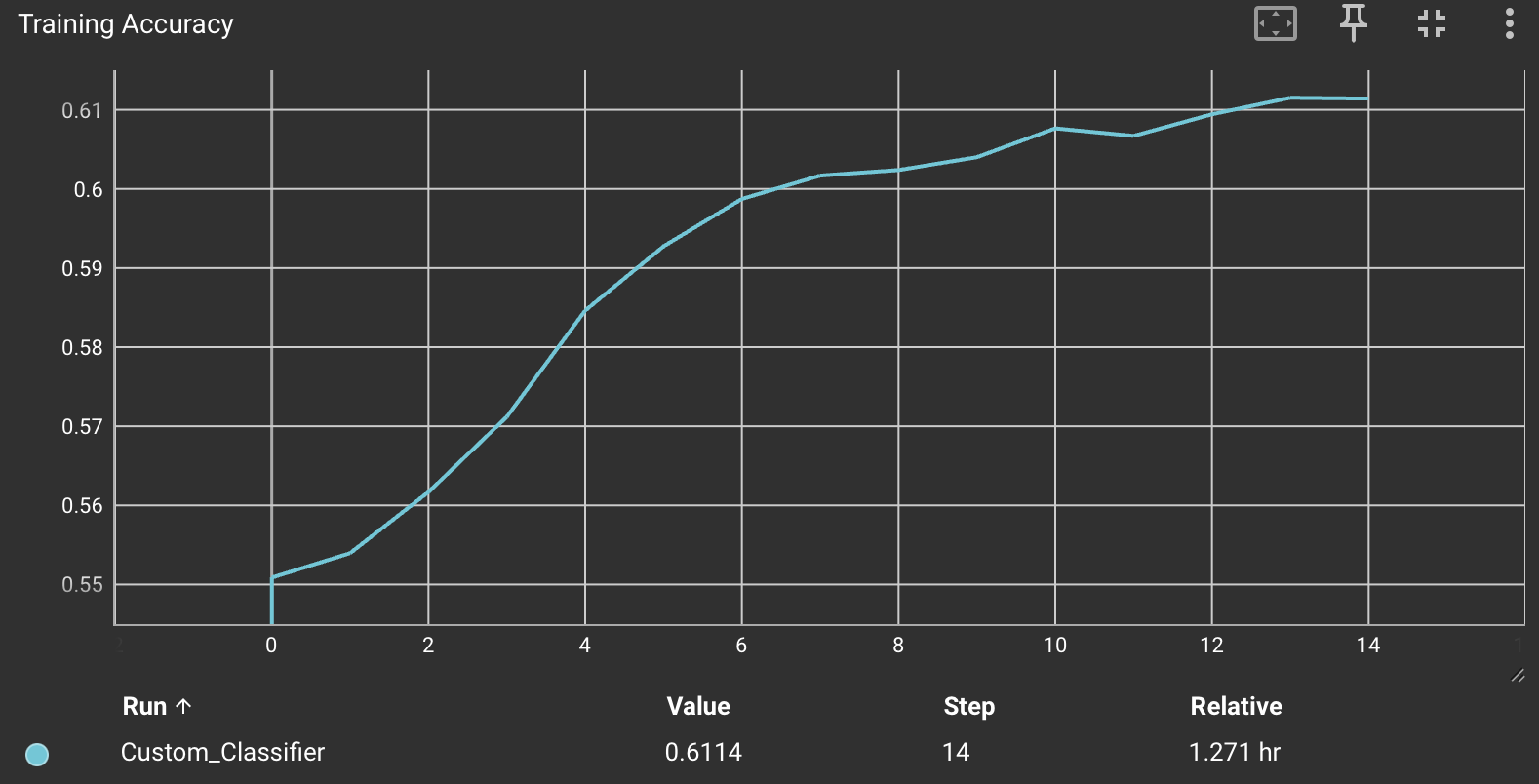
### Third Training Iteration - Custom Classifier Head

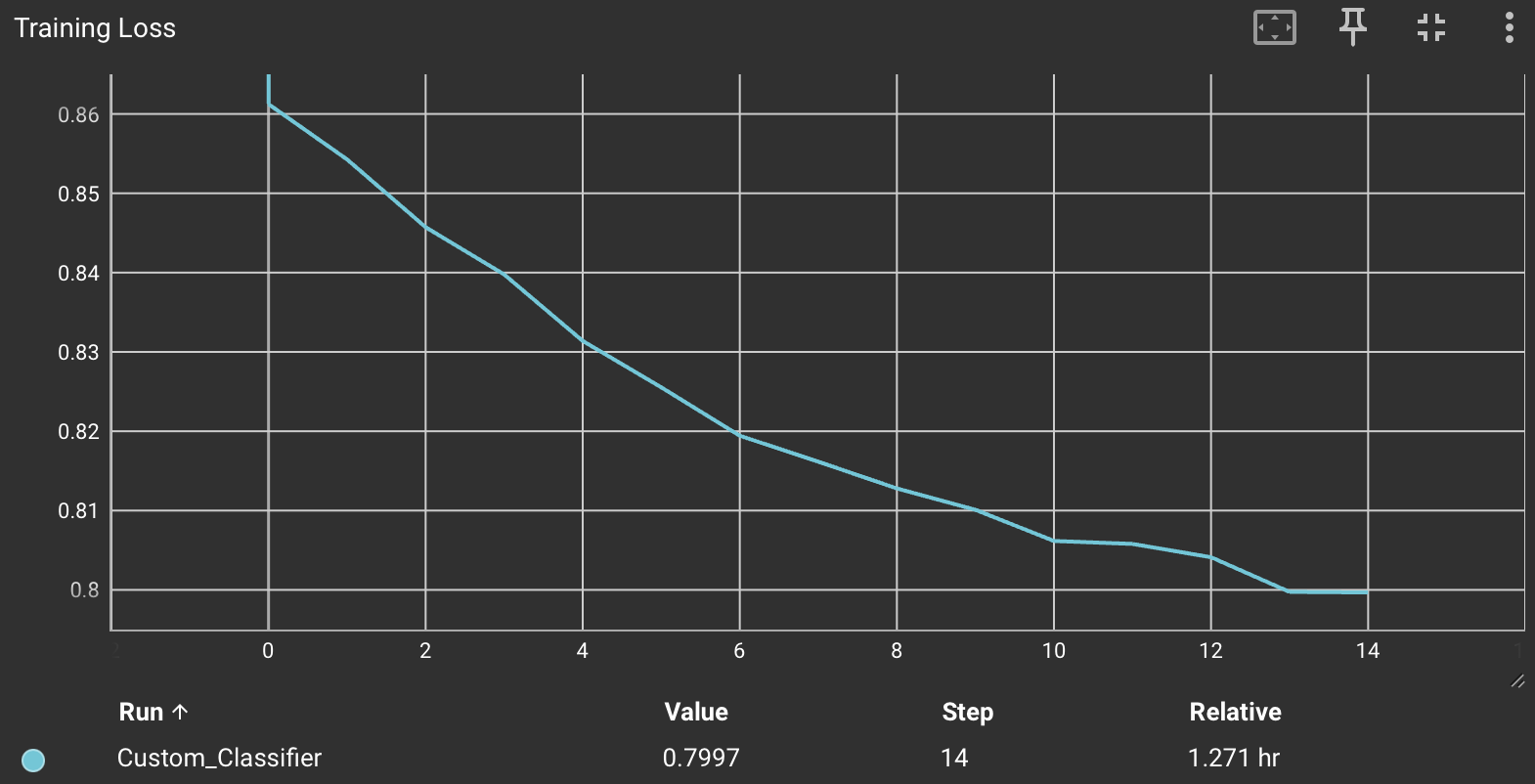
For the third model iteration, I added a custom classification head to see if I could improve the accuracy with additional architecture depth.

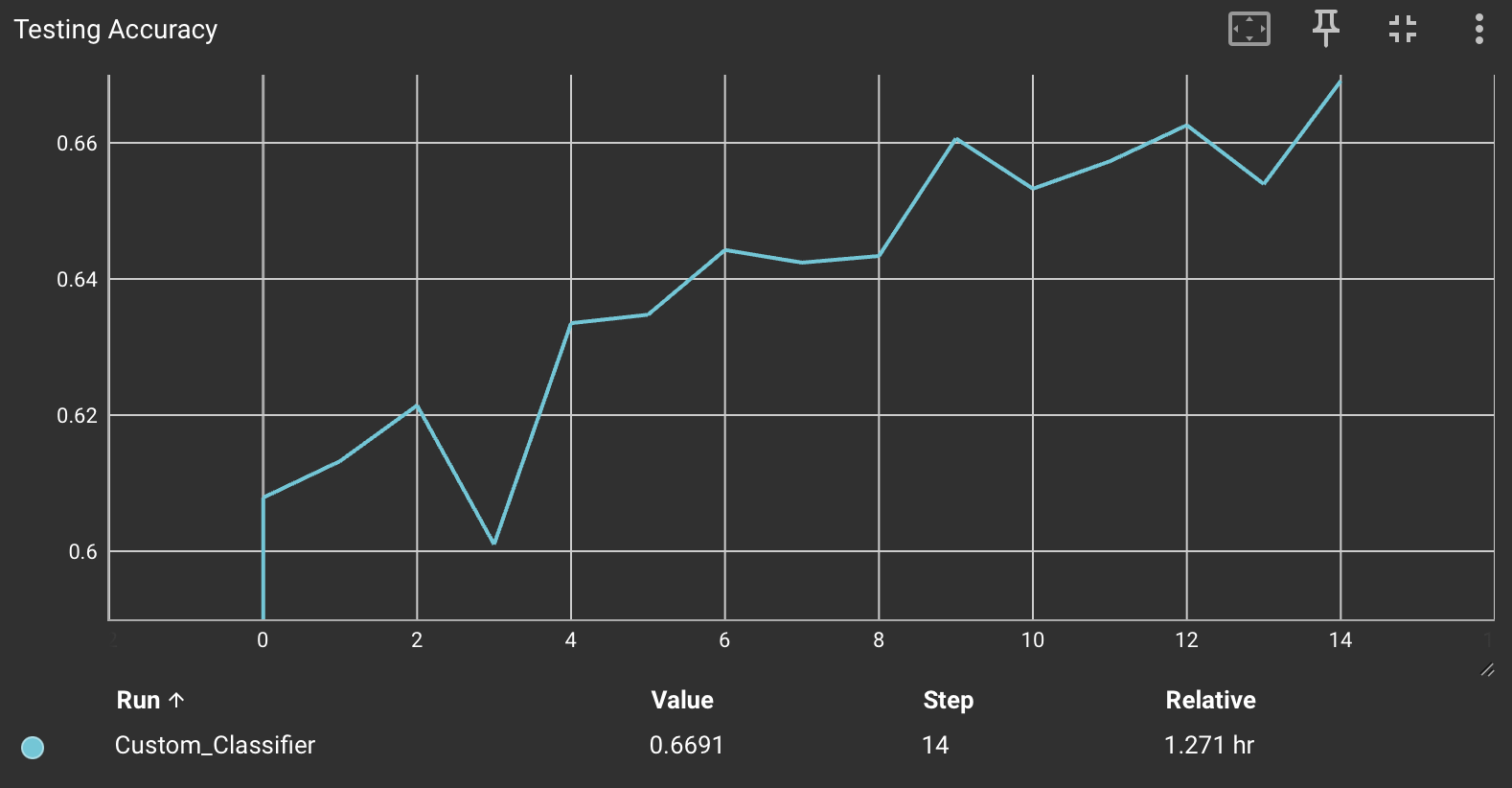
This was the custom classifier implemented:

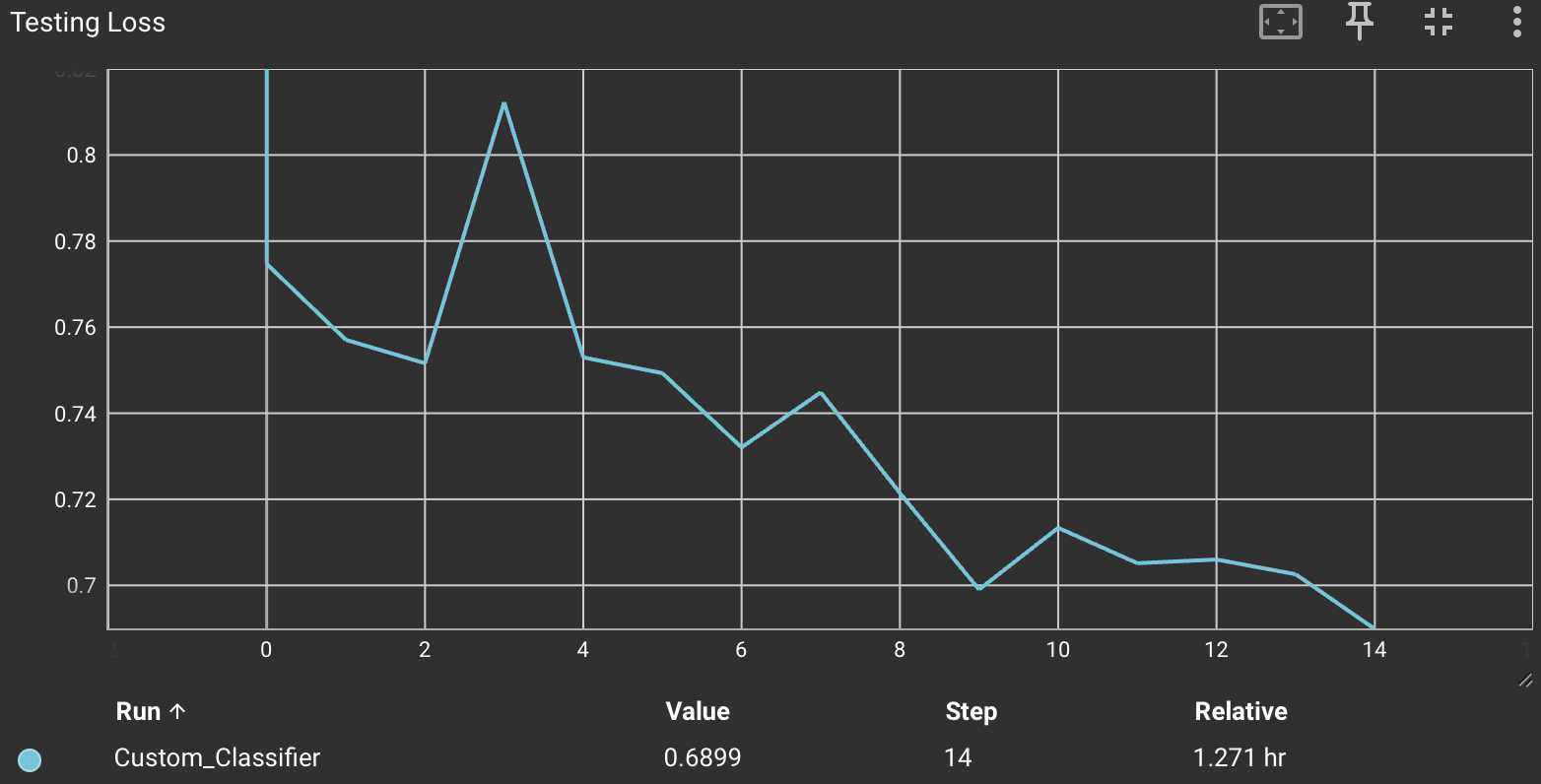


The classifier was able to achieve higher accuracy than the base model architecture, although not by a large margin. For future iterations out of the scope of this experience, we could consider adding even more depth, increasing the number of nodes per layer or using different layer activations.



Training Loss & Accuracy (Max Accuracy=61%)



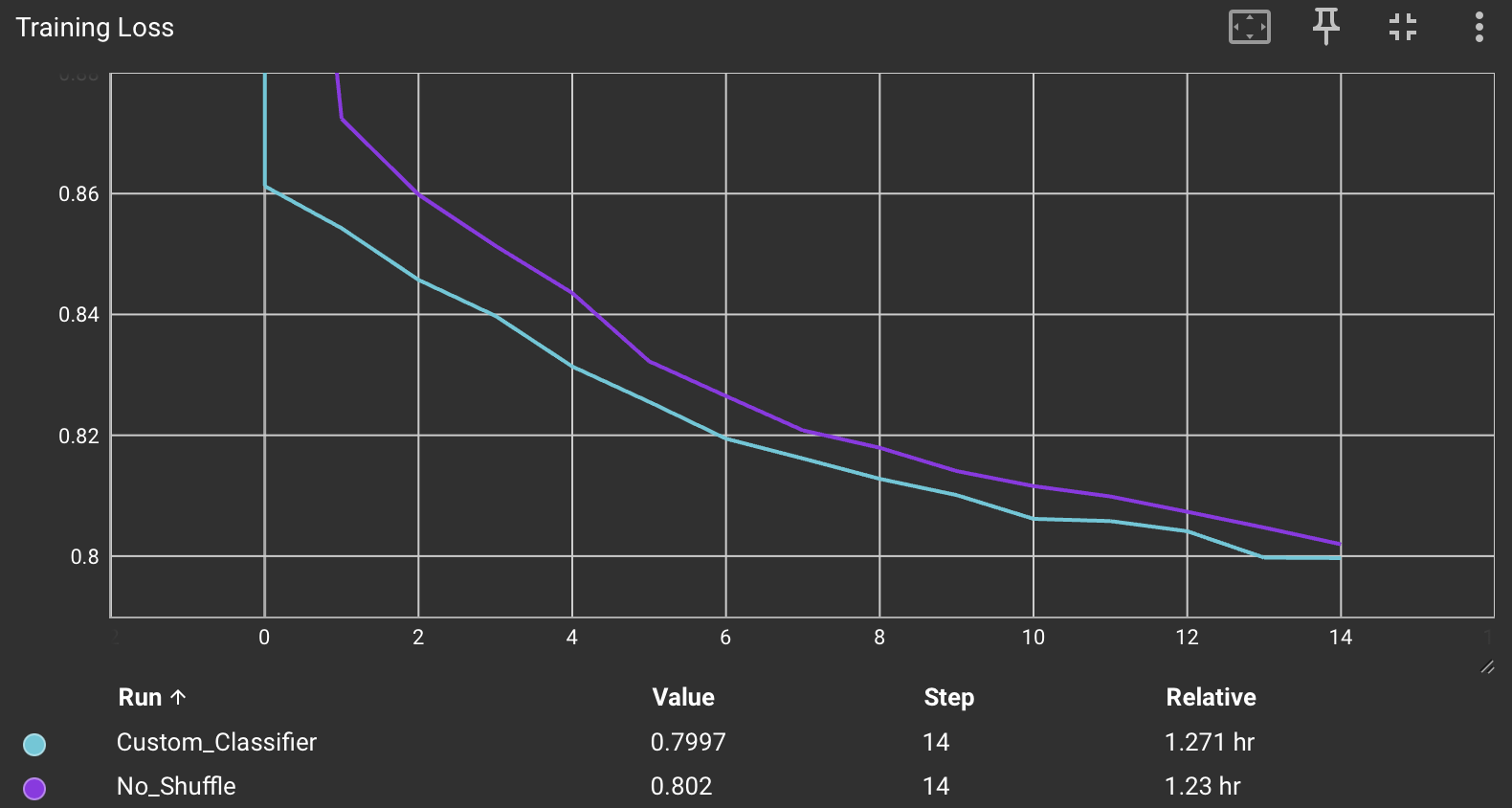
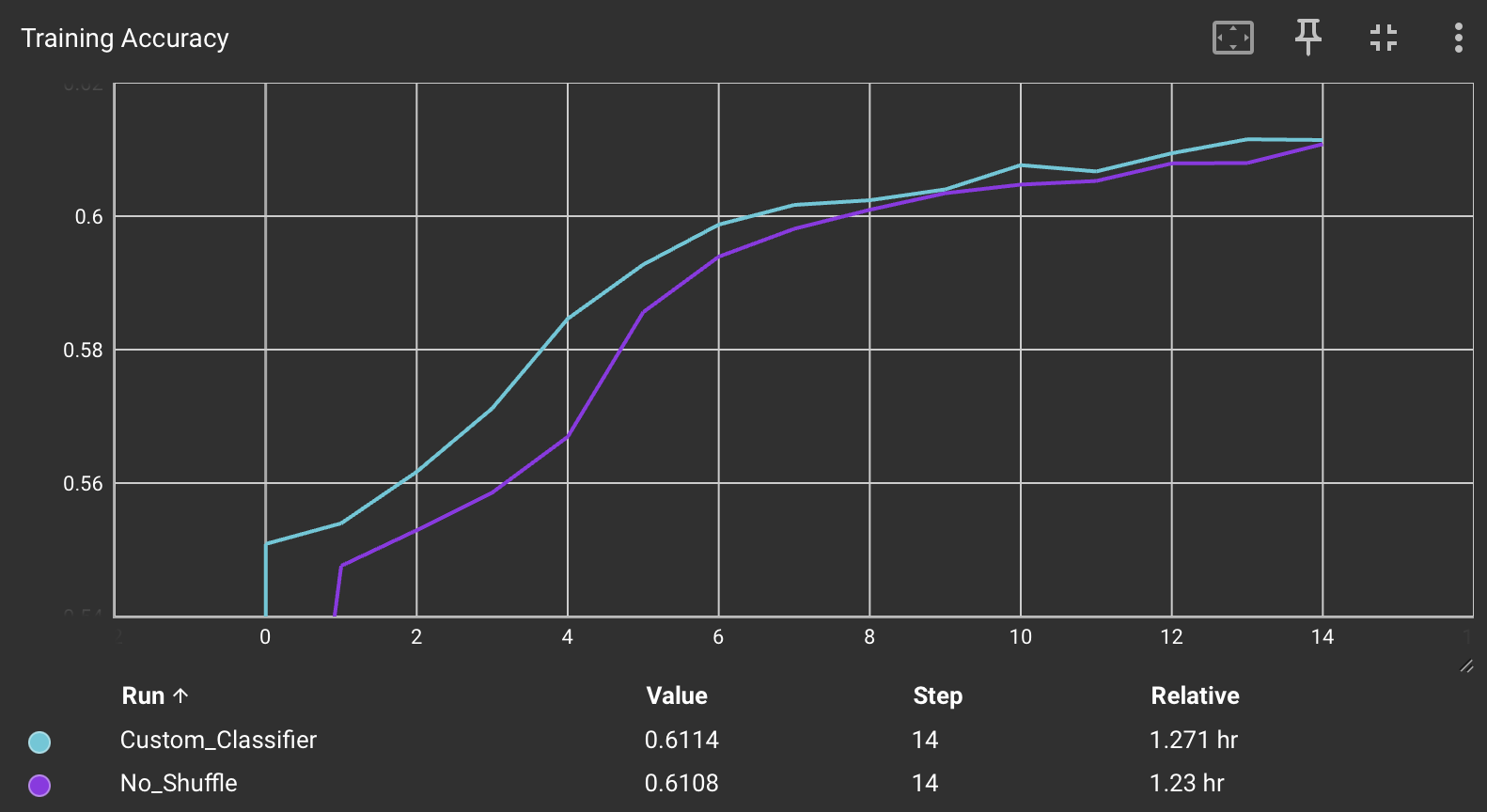


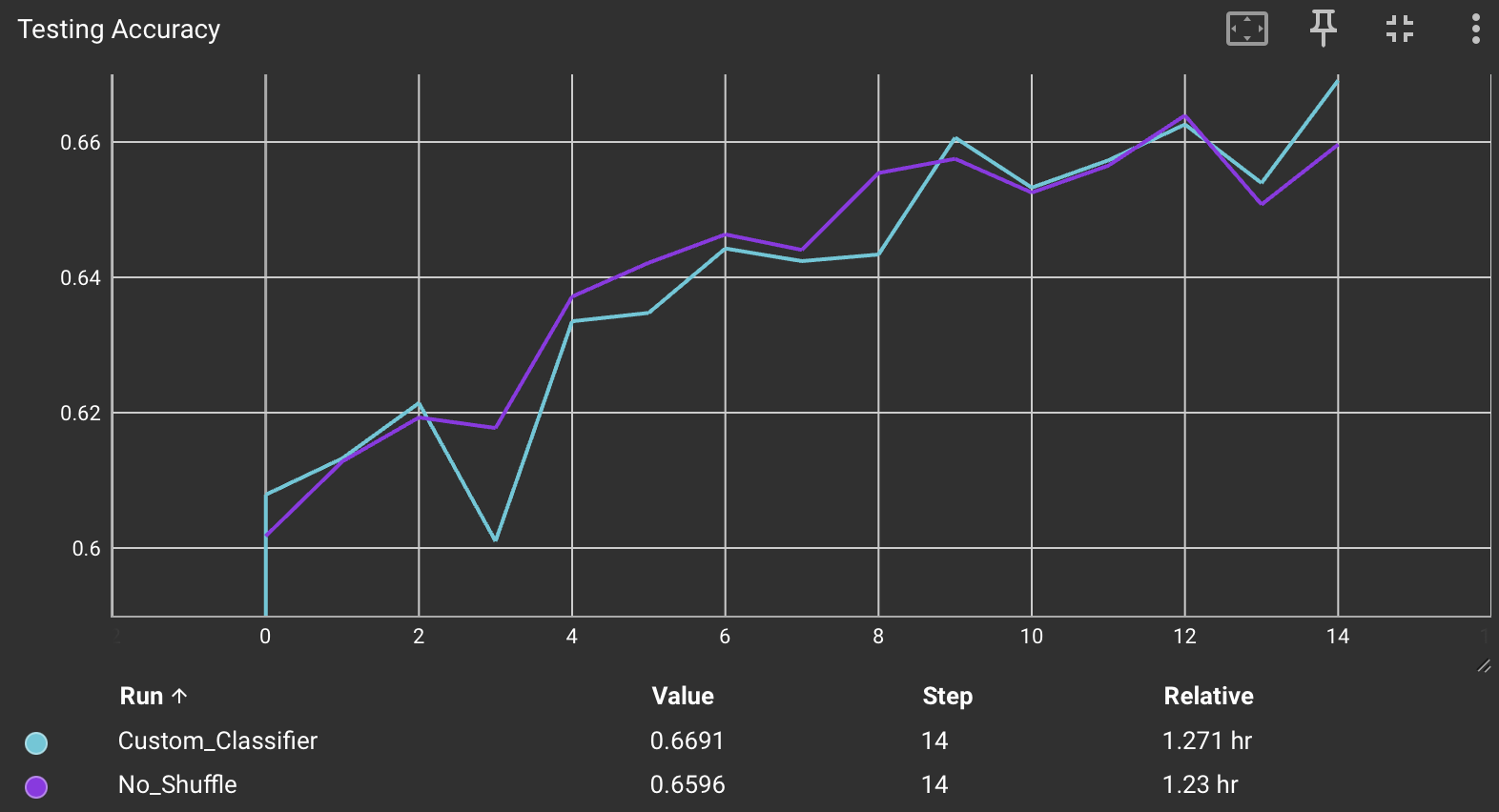
Testing Loss & Accuracy (Max Accuracy=67%)

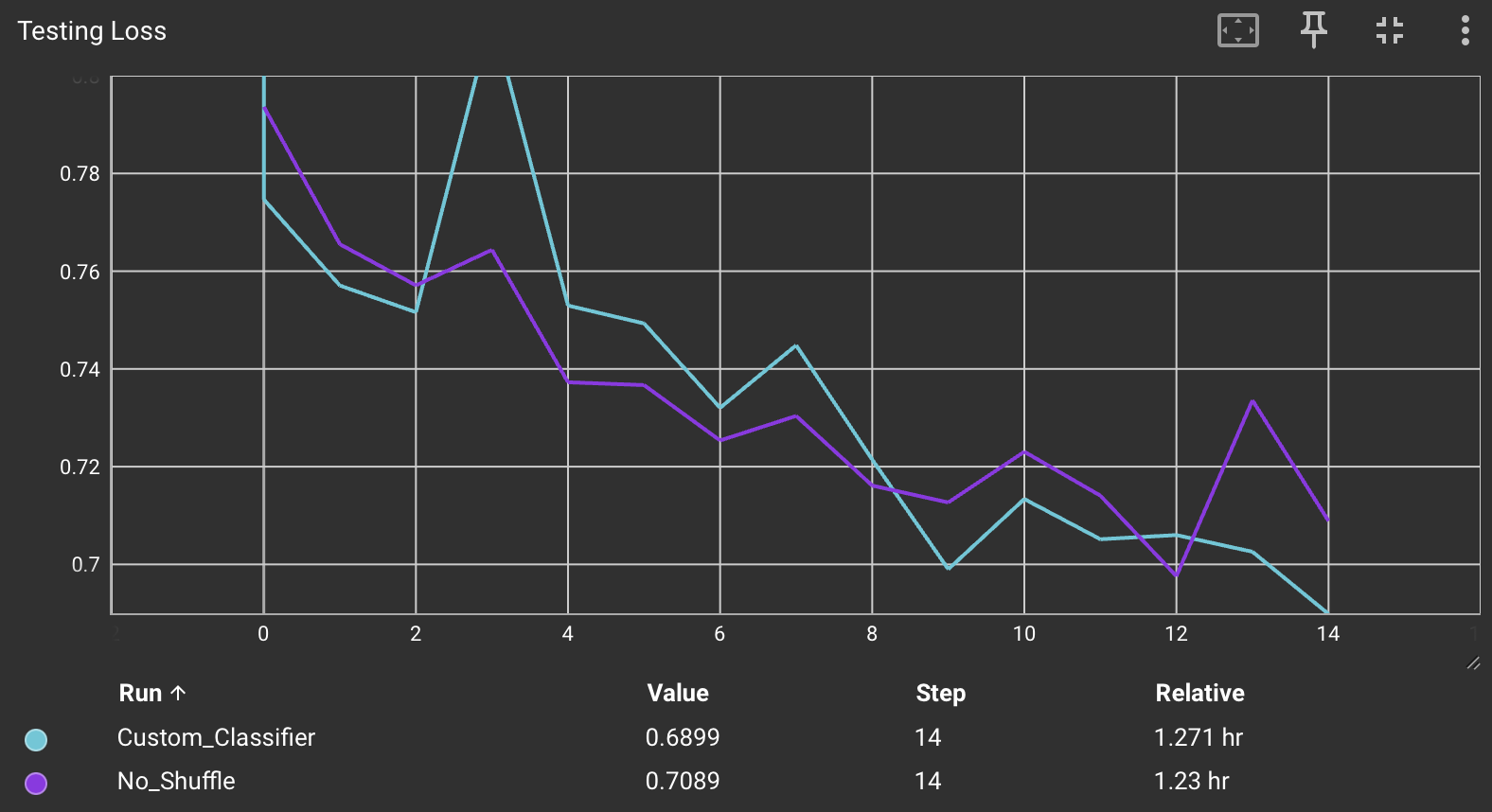
### Fourth Training Iteration - No DataSet Shuffle & Full Model Training

For the final iteration of the model, I decided to test the impact of removing shuffling at the dataset level. Although not recommended because this can cause overfitting to the specific dataset, I still wanted to see if there would be any significant impact.

To my surprise, the model performed similarly, reaching almost identical performance on the 15th epoch. However, the loss decreased more in the model that had shuffling active, which might indicate a better ability to generalize the weights enough for a wider variety of scenarios.





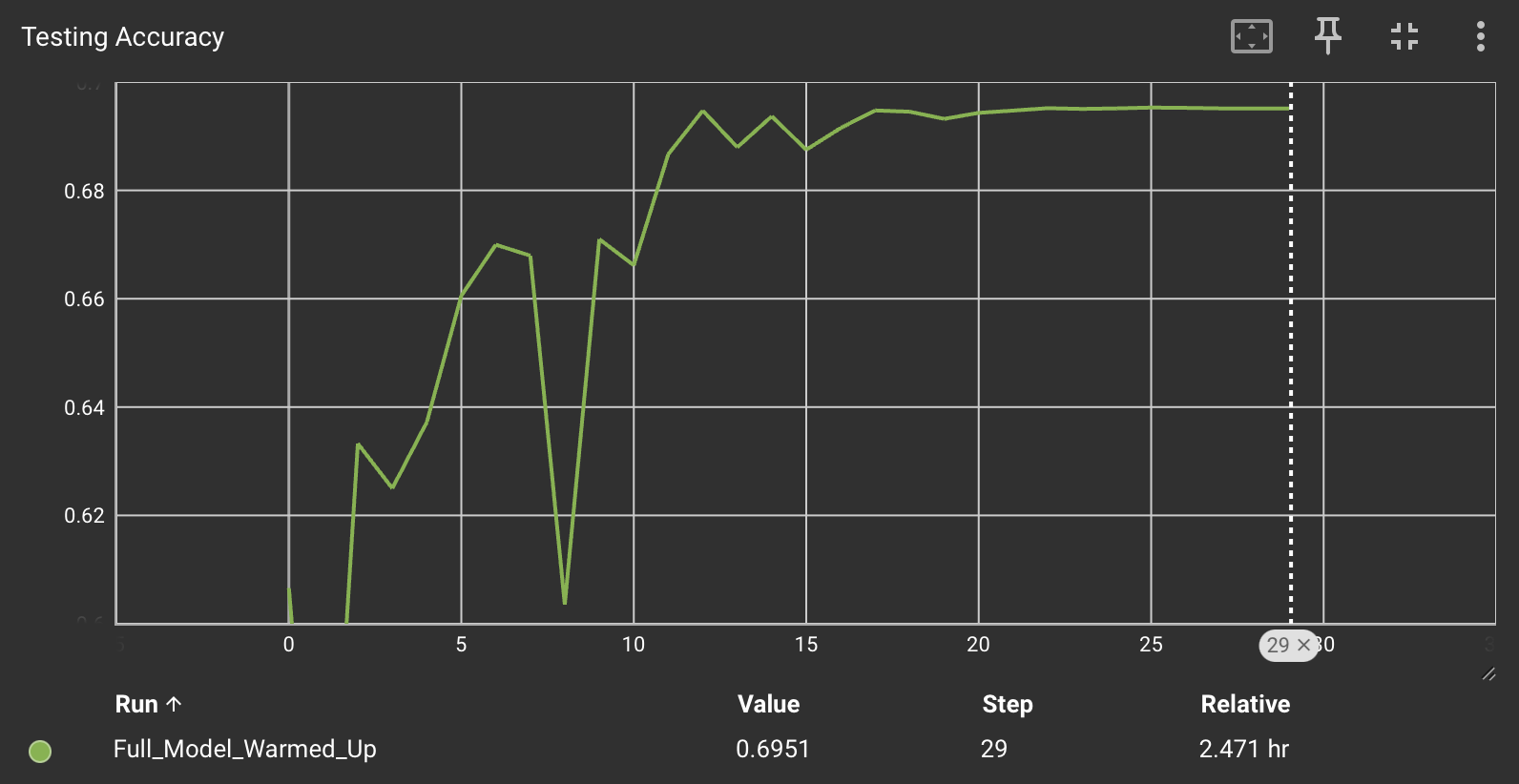


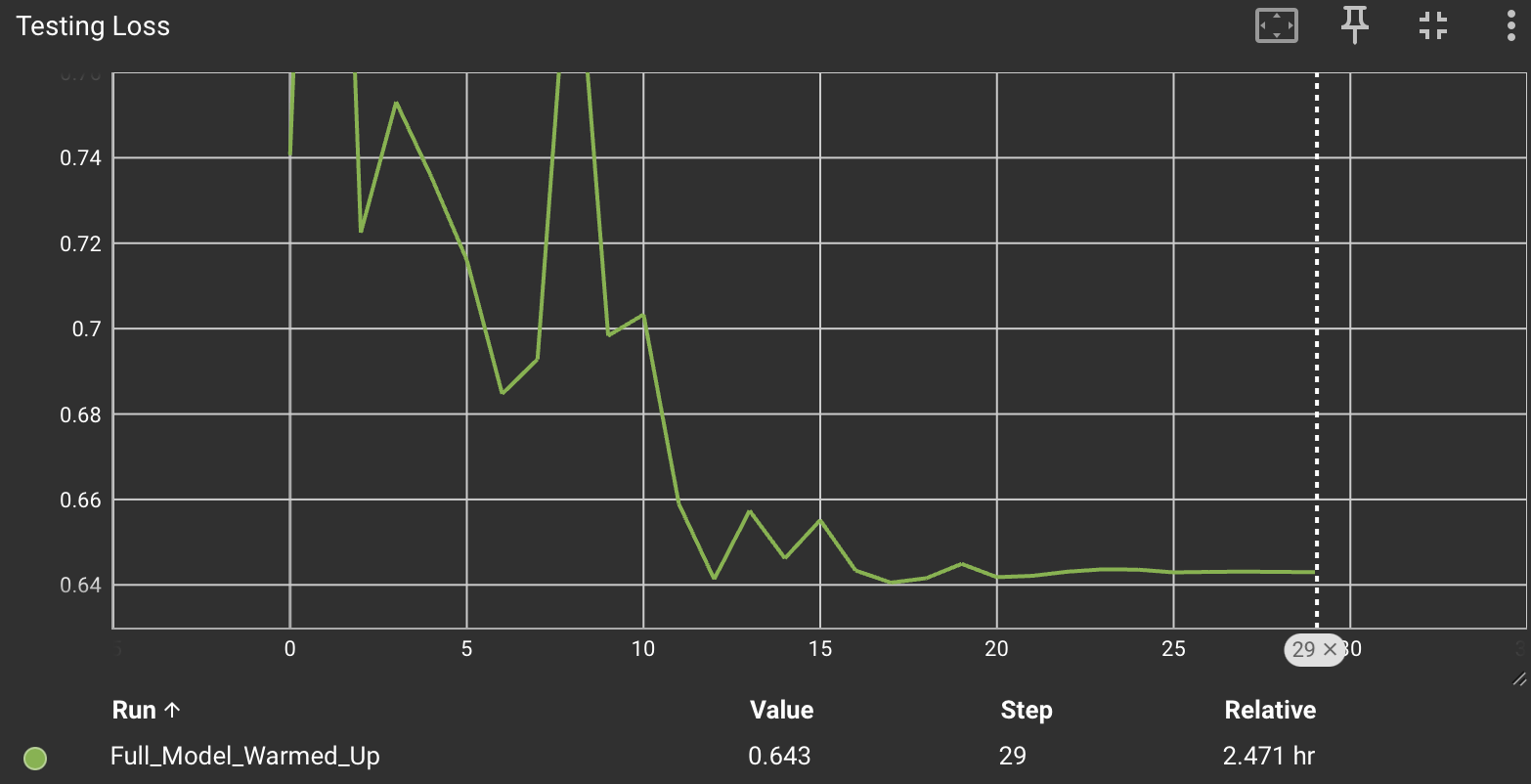
For these reasons, I decided to use the pretrained model that used shuffling and train it for 30 epochs.

### Final Iteration - Learning Rate Scheduler (With pre-trained classifier)

Since the model had already been trained, I decided to implement a learning rate scheduler to avoid overfitting and better handle local minima. The learning rate scheduler was implemented using ReduceLROnPlateau and decreasing the rate by 0.1 after 3 epochs with a decrease in testing loss of less than 1e-4.

The results of the final model can be seen on the graph:





A final accuracy of 70% was achieved on the testing dataset, which surpases the performance of the decision tree and the neural network. This was a positive outcome considering the amount of preprocessing required is limited to the tokenization of the sentence pairs, as opposed to the more complicated cosine difference calculated for the previous approach.

You can check the notebook for this model [here](https://drive.google.com/file/d/1sbIPAA7gG5OUSoicUBuaRBRaoXsu77y8/view?usp=sharing).

# Conclusions

The results from this experience show that automating this task is possible with a transformer model and is very achievable.

For future iterations, I am planning to simplify the training data by removing the “Added” class. Based on the needs of PPC managers, the most time-consuming and sensitive aspect of this task is finding all the irrelevant searches and making sure that relevant searches are not excluded by mistake.

By focusing only on two categories (None and Excluded), we can ensure that the accuracy for the excluded category is higher and make the model less prone to cause labelling issues that could negatively impact the job of a PPC manager. Additionally, our database also contains more sentence pairs labelled as None and Excluded, enabling me to train the model with significantly more data.