ANN classification for geotechnical stability

Paulo Lopes[1]
Hernani Lima[2]

Abstract

This research presents an in-depth exploration of the challenges and benefits of using a neural network model for a complex classification task. Central to the investigation was the process of hyperparameter tuning, which, while ensuring model optimization, also introduced significant computational demands. A strategic transformation from multi-class to binary classification was made, resulting in increased accuracy but limited prediction specificity. The study revealed that while deep learning models demonstrated superior performance in terms of accuracy, their intrinsic complexity posed challenges related to transparency and interpretability. Comparisons between traditional machine learning models and deep learning models illustrated the trade- offs between prediction time and accuracy. Furthermore, a user-friendly application interface was designed and developed using Tkinter, emphasizing intuitive interaction and visual feed- back mechanisms. The deep learning model achieved average accuracies of 95.24% and 98.81% for three-class and binary classifications, respectively. Findings from this study underscore the promise of deep learning in classification tasks, highlighting areas for future research in optimization, interpretability, and user-friendly design.

Keywords

[1] Co-founder - Beyond Mining - paulo@beyondmining.tech
[2] Researcher and Professor - UFOP - hernani.lima@ufop.edu.br

Introduction

Open-pit mining is an essential technique for the extraction of valuable mineral resources. The safety and efficiency of these operations heavily rely on the stability of the excavated slopes. However, as these slopes get steeper, the risk of slope failure increases, resulting in significant economic and operational implications[3][24]. An example of such a significant slope failure can be seen in Figure 1 which showcases a porphyry copper mine in South America. Traditional methodologies for assessing slope stability are challenged by difficulties in obtaining prior information, simplicity of calculation models, and weak fitting abilities[20].

Figure 1: A porphyry copper mine in South America suffered a major slope failure. [21]

To address these limitations, this dissertation aims to apply advanced machine learning techniques, with a focus on Deep Learning and Neural Networks[14]. Our goal is to develop a model that enhances accuracy, efficiency, and user-friendliness of slope stability assessments in open-pit mining. This effort is geared towards improving mine production efficiency, reduc- ing economic risks, and making a valuable contribution to the field of geotechnical engineer- ing.

Existing state-of-the-art solutions, both commercial and academic, present limitations that this project aims to overcome. Despite advancements in the field, these methodologies often incur high computational costs, lack accuracy in complex scenarios, and fall short in user-friendliness, particularly for non-specialist personnel. In our research, we address these challenges by finding the optimal balance between computational complexity and accuracy, developing a user- friendly software interface for practical, in-field usage, and validating the performance of our model against real-world case studies[25].

The project's initial phase is centered around running a grid search to determine the optimal neural network structure for a three-class problem (OF, ST, and FSB) in order to maximize accuracy. It's important to note that in practical applications, the focus is typically on discern- ing between two main states: stable and unstable. With this in mind, the primary emphasis of this project is the development of an optimized model tailored to this binary classification dilemma. This nuanced approach not only offers a more authentic representation of real-world applications but also provides the added benefits of reduced computational complexity and time consumption.

In laying out the objectives of this project, several key areas are addressed. Firstly, there's the endeavor to meticulously review and analyze current methodologies in slope stability assessment specific to open-pit mining. This includes both identifying potential shortcomings in these methods and exploring avenues for enhancement. Following this, the project aims to de- termine the best neural network structure for the aforementioned three-class problem through a grid search. Subsequently, there's a shift to refine the model to specifically cater to the bi- nary classification of stable and unstable states. Another pivotal objective lies in harnessing advanced machine learning methodologies, with a distinct emphasis on deep learning and neural networks. This is to notably augment the precision and efficiency of assessments related to slope stability. In addition, there's the task of curating a software interface that is user-centric. This interface should serve as a bridge, making these intricate machine learning models easily accessible for stability assessments. Lastly, validating the efficacy and accuracy of the model in question is indispensable. This will be achieved through real-world case studies, as supported by Zhang et al. (2021)[34], and by drawing comparisons with pre-existing methodologies.

From a hypothesis standpoint, this research operates on a few foundational beliefs. It posits that advanced machine learning strategies, especially deep learning and neural networks, have the potential to significantly enhance the precision and efficiency of slope stability assessments in the realm of open-pit mining. Another hypothesis is that the transition from a three-class problem to a binary one (stable and unstable) will yield a model that's better optimized for practical uses. Furthermore, this research suggests that a thoughtfully constructed software interface could potentially democratize the access to these high-end techniques,

even for those who aren't experts. This, in turn, could facilitate more informed decision-making in the field.

In sum, this project stands as an original contribution to geotechnical engineering. It carves out a niche by presenting a machine learning-oriented resolution for slope stability assessments, honing in on the pragmatic binary classification, and by introducing a user-friendly software tool for real-world applications. By tackling existing challenges and pushing the boundaries of the current state-of-the-art, this endeavor aims to play a pivotal role in amplifying the safety protocols and economic feasibility of open-pit mining activities on a global scale.

Dataset

The dataset used for this project is relatively small, consisting of only 84 entries. Each entry contains 18 parameters that are essential for the model's operation and falls into one of three class labels: OF(Out of Failure), ST(Stable), and FSB(Factor of Safety Below 1). The compact feature of this dataset presents both challenges and opportunities for our machine learning models[33].

Methods

The methodology for this project involved a multi-step process that includes the comparison be- tween conventional machine learning techniques and a deep learning approach. We also ran a comprehensive grid search to identify optimal hyperparameters for our Neural Network model. The project was implemented in Python using the Jupyter notebook environment, which pro- vided a flexible and interactive platform for model development, training, and evaluation[8].

Model evaluation metric

Given the limited size of our dataset, the division of data into training and testing sets must be handled delicately to prevent overfitting while ensuring that the models are well-trained and that their performance can be accurately assessed. For this study, 12 lines of data have been designated as the test case to keep it the same as in Zare Naghadehi et al., 2013[33], leaving the remaining 72 entries for training.

For model evaluation, we've decided to utilize a 7-fold cross-validation method. This method involves dividing the training dataset into seven subsets. During the evaluation process, the model is trained on six subsets and validated on the one left out. This process is repeated seven times with a different subset used for validation each time. The final model's performance is then given by the average performance across these seven iterations. The rationale behind using 7-fold cross-validation is

that it provides a good balance between computational efficiency and reliable estimation of model performance[32]. In our context, this method is especially effective due to the small size of our dataset.

Furthermore, we've also used a leave-one-out cross-validation method for a more detailed comparison[31]. In this method, a single observation from the original sample is used as the validation data, and the remaining observations as the training data. This is repeated such that each observation in the sample is used once as the validation data. This approach is similar to K-fold cross-validation; however, instead of arbitrarily choosing K subsets, we leave out only one observation at a time from the dataset. This is computationally more intensive than K-fold cross-validation but provides a more comprehensive evaluation of the model by leveraging all but one observation for each training-test iteration.

While these methods may seem rigorous, given the small size of our dataset, it is critical to utilize as much data as possible for training while still retaining a rigorous validation process. The choice of these specific methods ensures a thorough validation of our models and provides a more robust estimate of their performance in practical applications[31].

Conventional machine learning

The machine learning models we used are common classifiers that come with their own strengths:

- Support Vector Machines (SVM): SVM is particularly well-suited for classification of com- plex but small- or medium-sized datasets. It can model non-linear decision boundaries, and there are many kernels to choose from. It is also robust against overfitting, especially in high-dimensional space[17].
- K-Nearest Neighbors (KNN): KNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until func- tion evaluation[16].
- Decision Trees: A decision tree is simple to understand and to interpret. Trees can be visualized, which makes it handy for data exploration. It requires little data preparation and can handle numerical and categorical data.
- Random Forest: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting[35].
- Logistic Regression: Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although

many more complex extensions exist. It is a simple and fast model, suitable as a baseline for classification problems[29].

Deep learning model

The deep learning model used here is a type of feed-forward artificial neural network, where information moves in only one direction—forward—through the layers, from the input nodes, through the hidden nodes, and to the output nodes. There are no cycles or loops in the network, hence the term 'feed-forward'[27].

Neural networks acting like the human brain to interpret data[1], and recognizing the patterns hiding behind the data that are human impossible. The network is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.

We implemented the Neural Network using the Keras library, a high-level API built on top of TensorFlow, a library for machine learning in Python. Keras allows for easy and fast prototyping and supports both convolutional networks and recurrent networks, as well as combinations of the two[12].

There are four adjustable hyperparameters in this case:

- layer_config: This is a list where each element represents the number of neurons in the corresponding hidden layer of the neural network.
- dropout_rate: Dropout is a regularization technique were randomly selected neurons are ignored during training. It is specified as a rate between 0 (no dropout) and 1 (all neurons dropped out)[26].
- learn_rate: This is the learning rate used by the Adam optimizer for weight updates[37].
- activation: This is the activation function used in the neurons. Activation functions intro- duce nonlinearity into the model, enabling it to learn complex patterns.
- regularizer: Regularization is a technique used to prevent overfitting by adding a penalty to the loss function based on the size of the weights.

The model architecture consists of an input layer, multiple hidden layers with dropout applied, and an output layer with three units (corresponding to the three classes). The 'softmax' activation function is used in the output layer to provide a probability distribution over the classes, and the 'categorical cross entropy' loss function is used for multi-class classification[30].

Hyperparameter optimization with grid search

To find the optimal set of hyperparameters that yield the best performance of the neural network, a grid search strategy is employed. This is a brute-force approach to hyperparameter tuning that involves training and evaluating a model for each combination of specified hyperparameter values[15].

Once this optimal set is identified, a more focused search is conducted around these parameter values. This second, narrower search allows for fine-tuning of the hyperparameters to further enhance the performance of the model.

The model associated with the highest average accuracy over the k-fold cross-validation is selected as the best model[28]. This combination of broad and focused searching enables a robust and effective optimization of the model's hyperparameters. However, the grid search approach may be computationally expensive when the hyperparameter space is large and com- plex.

Sensitivity analysis

To understand the influence of individual features on our model's performance, we conducted a sensitivity analysis. In this context, sensitivity analysis helps us identify how sensitive our model's predictions are to changes in the input features[10]. Two different methods were em- ployed for this analysis: Feature masking and permutation importance.

Feature masking

In the first method, we evaluated the model's performance by masking each feature in turn and measuring the accuracy of the model's predictions without that feature. The idea here is that if a feature is important, masking (or setting it to zero) would significantly degrade the model's performance[23].

The code first lists all the feature names used in the model. It then initializes an array to hold the accuracy values for each feature across all models. The loop iterates through each model, masking each feature one at a time, predicting the class labels, calculating the accuracy of these predictions, and storing these accuracies in the array. After this, we compute the average accuracy for each feature across all models.

Here, the masked feature is replaced with zeros in the test set, essentially rendering it 'invisible' to the model. The model then makes predictions based on this altered input. The accuracy of these predictions (compared to the true labels) gives us an idea of how important that particular feature is — if accuracy drops significantly when a feature is masked, we can infer that the feature is of high importance.

Permutation importance

The second method used for sensitivity analysis is permutation importance, aided by the eli5 library[4]. Permutation importance is a robust and straightforward way of computing feature importances. It works by permuting the values of each feature and measuring how much the performance decreases.

The function permutation importance first calculates a baseline score, which is the model's accuracy on the test data. Then, for each feature, the function shuffles the values of that feature (thus destroying any meaningful relationship it has with the target), and measures the decrease in the model's accuracy.

The logic behind this method is that permuting the values of an important feature will significantly degrade the model's performance, as the model relies on that feature to make accurate predictions[13]. By repeating this process multiple times and averaging the score decreases, we get a reliable measure of the feature's importance.

The function returns a dictionary where the keys are the feature names and the values are the feature importance scores, which represent the decrease in accuracy caused by permuting each feature.

In conclusion, sensitivity analysis, through feature masking and permutation importance, gives us insights into the relative importance of different features used in the model. It provides a basis for potential model refinement by identifying less important features which could be removed to simplify the model without significantly impacting performance.

Adaptation to real-world situation and model modification

In practical applications, particularly in the context of open-pit mining, slope stability essentially boils down to two conditions - stable and unstable. Reflecting this reality, we have decided to modify our classification task, originally framed as a three-class problem (OF, ST, and FSB), into a binary one. This simplification is accomplished by combining the 'OF' and 'FSB' categories into a single class, representative of an 'unstable' state. The 'ST' category, naturally, represents the 'stable' state.

A significant advantage of this conversion is the simplification it brings to our problem. Bi- nary classification problems are generally easier to model and solve compared to multi-class problems, and the performance of binary classification algorithms is often superior due to the inherent simplicity of the problem structure[22].

This adjustment necessitates a change in the output layer of our neural network model. Instead of using model.add(Dense(3, activation='softmax')) — which suits a three-class problem — we transition to a binary output layer with the code model.add(Dense(2, activation='sigmoid')). This updated layer now consists of two units (corresponding to the two classes of 'stable' and 'unstable') with the 'sigmoid' activation function.

The sigmoid function is chosen as it maps the output to a range between 0 and 1, which can be interpreted as probabilities. This makes it an ideal choice for binary classification problems[9]. It's important to note that while 'softmax' can also be used for binary classification, 'sigmoid' is often preferred due to its simplicity and ease of interpretation in a binary setting[11].

Through these modifications — merging classes and adjusting the output layer of our neural network model — we have successfully tailored our model to better represent the real-world conditions of open-pit mining and enhanced its capacity to make accurate and interpretable slope stability predictions.

It's also worth noting that apart from these changes, the rest of the model development process remains the same as in the original three-class problem. The pre-processing of data, the structure and tuning of the hidden layers, the training procedure, and the validation methods (includ- ing the 7-fold cross-validation and leave-one-out method) all retain their original configurations. This consistency ensures that we maintain a methodological continuity, and any changes in the model performance can be attributed solely to our adaptations in class representation and output layer configuration.

Tkinter usage

Our chosen user interface employs Tkinter, a built-in Python package, known for its simplicity and cross-platform compatibility. This tool enables the creation of a minimalist graphic user interface (GUI) that is operable across various operating systems such as Mac, Windows, and Linux. The software's design prioritizes user-friendliness and efficiency. Upon launching the software, it loads the pretrained model seamlessly. The interface has been equipped with intu- itive controls to enable users to input all 18 parameters required for the model. For enhanced user flexibility, two input options are provided: an easy-to-use slider for swift value adjustments, and a text box for manual entry. Upon input completion, the 'predict' button initiates the model's inference process. To enhance user comprehension, we represent the model's prediction re- sults using a pie chart, which graphically breaks down the probabilities of each class. This visualization offers users a clear and immediate understanding of the model's confidence level regarding its prediction.

Technical requirements and deployment

The technical requirements for this project are not intensive. As it is written in Python, it can be executed on any operating system that supports Python. For the heavy computation needed for training the model, a computer with a multi-core processor and a reasonable amount of RAM (at least 8 GB, but more preferred) would be needed.

The model is built and trained using open-source libraries, including scikit-learn, Keras, Tensor- Flow, numpy and pandas. These can be installed via pip or conda. The Python version used for this project is 3.7, but any version from 3.5 onwards should work fine.

Once the model is trained, it can be saved and loaded for prediction using the joblib library in scikit-learn. The saved model can be loaded into a web or desktop application for practical use. This requires integration work with a software development team.

Results

Comparison of traditional ML methods

Various machine learning and deep learning algorithms were utilized in this study to construct and evaluate different models. The derived average accuracies are as follows:

TABLE 1 Average results and error ranges for various machine learning methods.

Method	Average result (%)	Error range (%)
Support Vector Machine (SVM)	89.26	± 5.80
K-Nearest Neighbors (KNN)	74.93	± 4.84
Decision Tree	67.87	± 5.90
Random Forest	83.38	± 8.57
Logistic Regression	91.69	± 5.97

Among the traditional machine learning techniques, Logistic Regression achieved the highest accuracy at 91.69%, closely followed by SVM with 89.26%. Both methods demonstrated stan- dard deviations within a 6% range, suggesting stable performance. Conversely, Decision Trees recorded the lowest accuracy, at 67.87%.

The diverse accuracies underline the significance of algorithm selection based on data attributes and the nature of the problem.

Neural network optimization results

For the Neural Network, we performed a grid search to determine the most effective hyperpa- rameters. The results are as follows:

- Layer Configuration: [16, 64, 64]
- Dropout Rate: 0.0
- Learning Rate: 0.0005
- Activation Function: ReLU
- Regularizer: L2

Implementing the optimal hyperparameters resulted in an average accuracy of 95.24% across the seven cross-validation folds. The model's performance was observed to stabilize after roughly 230 epochs as shown in Figure 2. Two examples of the model's confusion matrix are also provided in Figure 3.

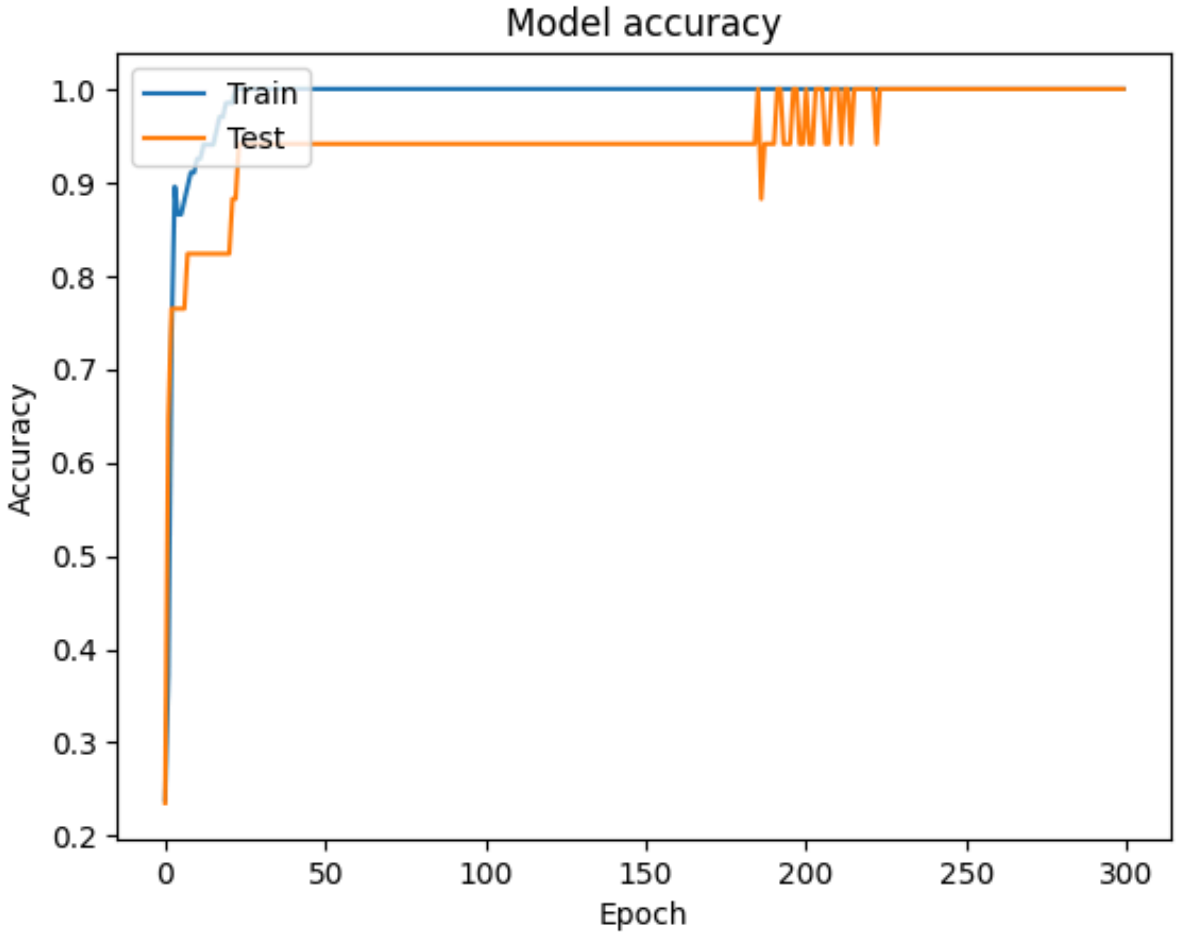
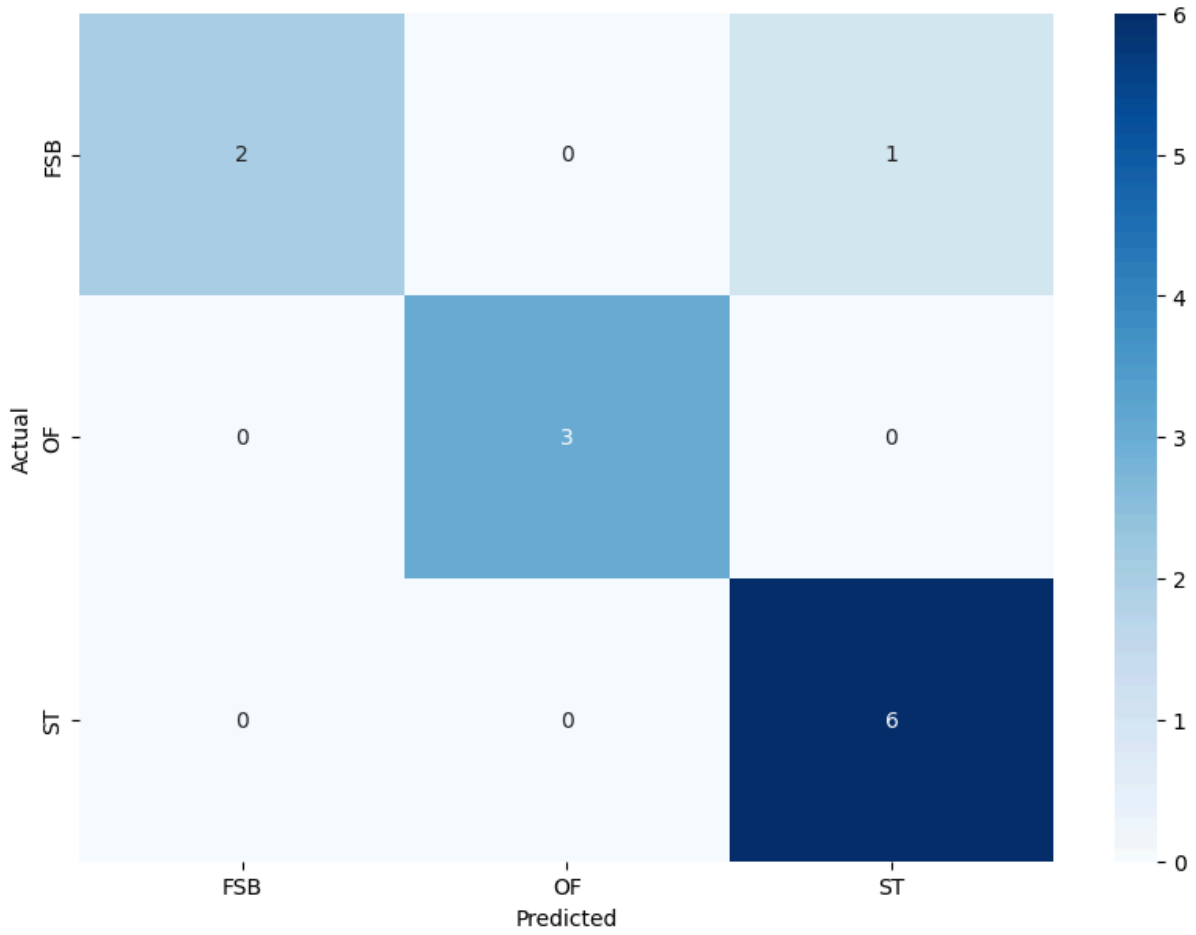
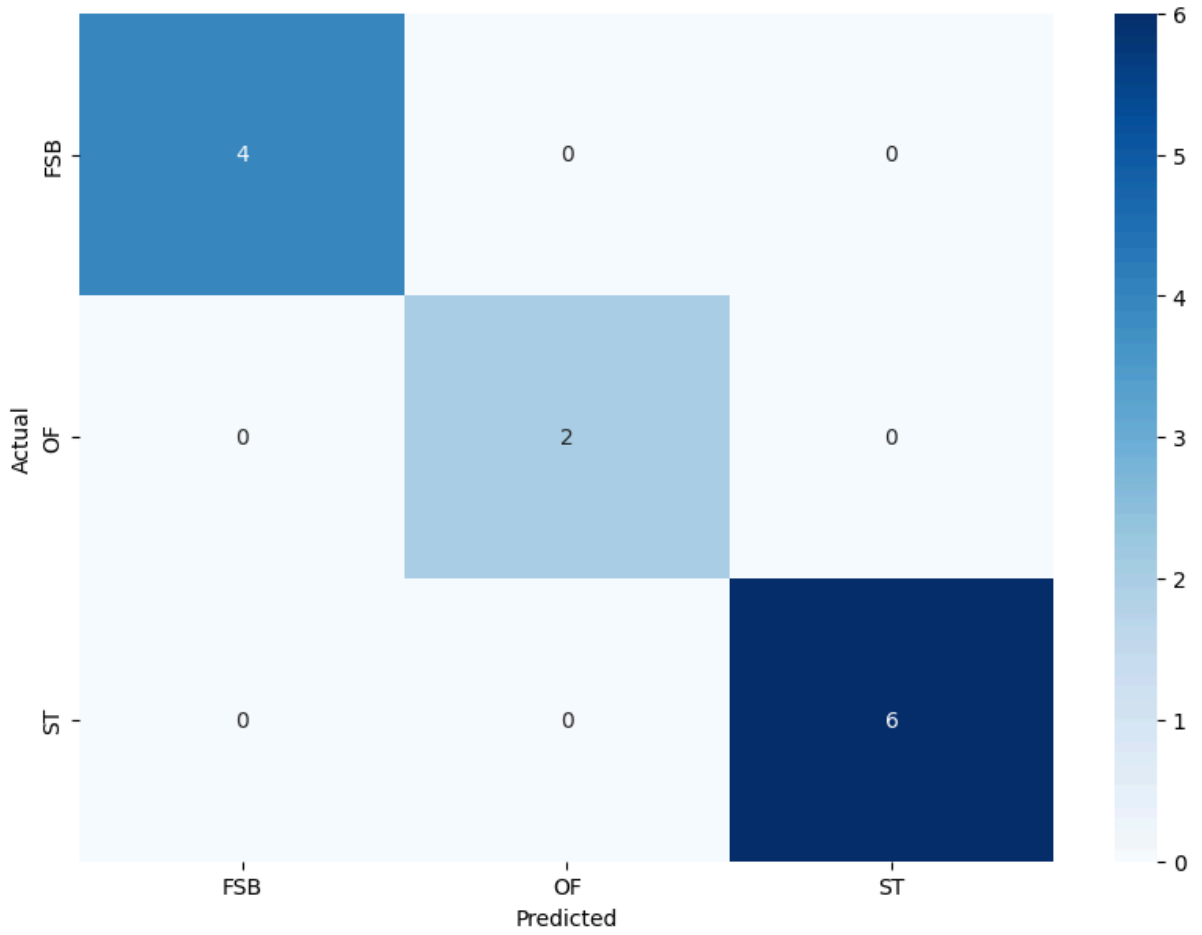


Figure 2: Illustration of Epochs versus Accuracy for one of the best performing Neural Network models.



(a) 91% accuracy case



(b) 100% accuracy case

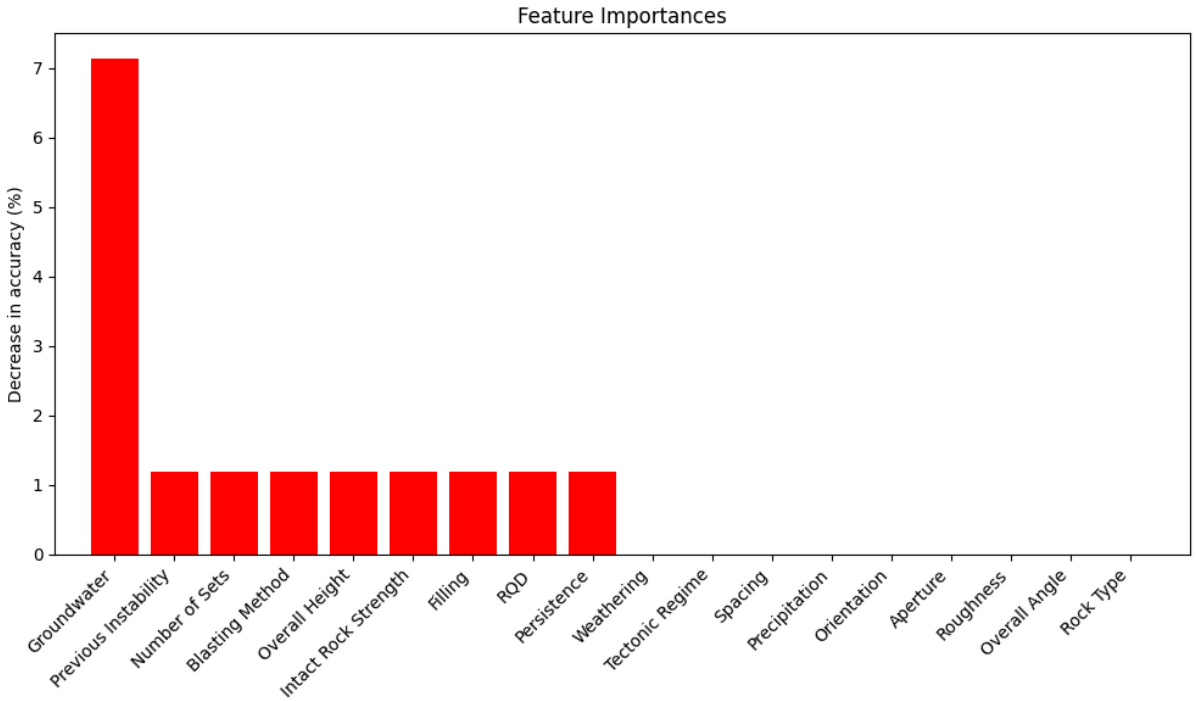
Figure 3: Confusion matrices from the highest-performing model in three classes, show- casing (a) a 91% accuracy instance with only one error, and (b) a 100% accuracy instance.

Feature importance in three-class scenario

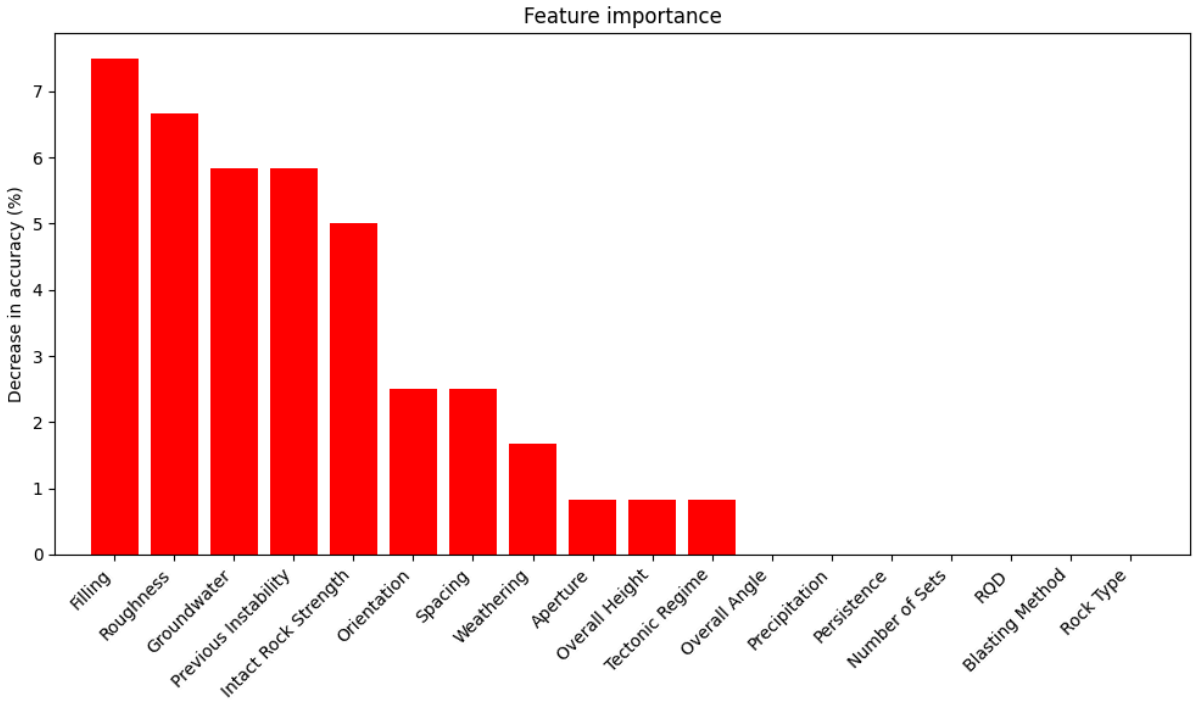
From the feature masking results in Figure 4 (a), the following key observations were made:

- Groundwater: A significant accuracy decrease is observed when this feature is removed, with a drop of 7.14%, underscoring its essential role in class determination within this context.
- Parameters with Marginal Influence: Features including 'Previous Instability', 'Number of Sets', 'Blasting Method', 'Overall Height', 'Intact Rock Strength', 'Filling', 'RQD', and 'Persistence' contribute to an accuracy decline of 1.19% when masked, suggesting their moderate relevance.

- Features with no Impact on Accuracy: Attributes like ’Weathering’, ’Tectonic Regime’, ’Spacing’, ’Precipitation’, ’Orientation’, ’Aperture’, ’Roughness’, ’Overall Angle’, and ’Rock



(a) Feature Masking



(b) Permutation Importance

Figure 4: Sensitivity for each parameter in three classes case. Each figure was obtained from the following method: (a)Feature Masking. (b)Permutation Importance.

Types do not affect the accuracy when masked, indicating either their negligible influence on model decisions or potential data redundancies.

From Figure 4 (b), using the permutation importance method, we discerned:

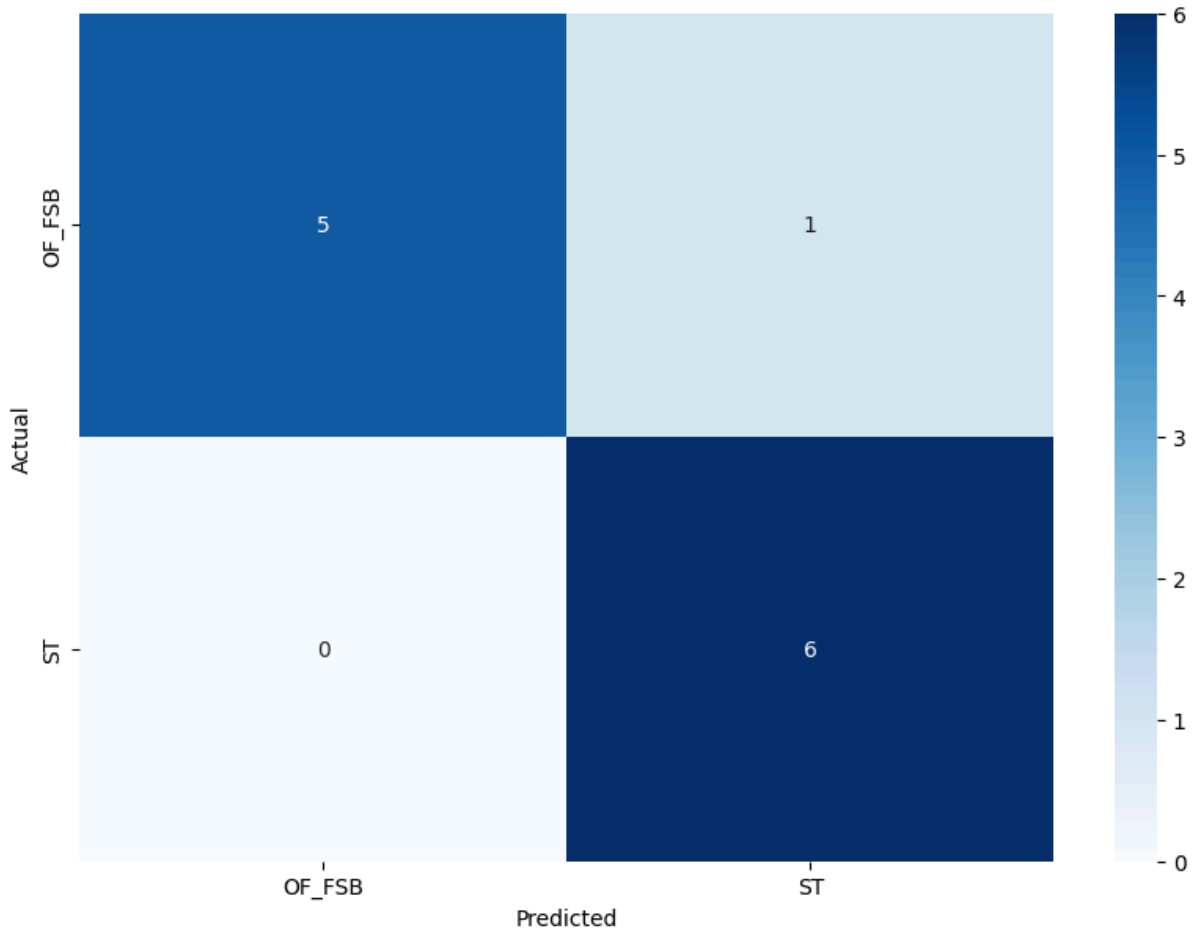
- Filling and Roughness: Both features have a significant permutation importance score exceeding 6.67%, indicating their essential role in model performance.
- Groundwater and Previous Instability: Each of these features registers a score of 5.83%, supporting their influential status identified by the feature masking method.
- Intact Rock Strength: This feature registers a score of around 0.05, denoting its moderate impact on the model's decisions.
- Features of Negligible Significance: Attributes such as 'Precipitation', 'Rock Type', 'RQD', 'Number of Sets', 'Persistence', 'Overall Angle', 'Blasting Method', and another instance of 'Precipitation' show no permutation importance, reinforcing their potential insignificance or redundancy in the given scenario.

Results for binary classification scenario

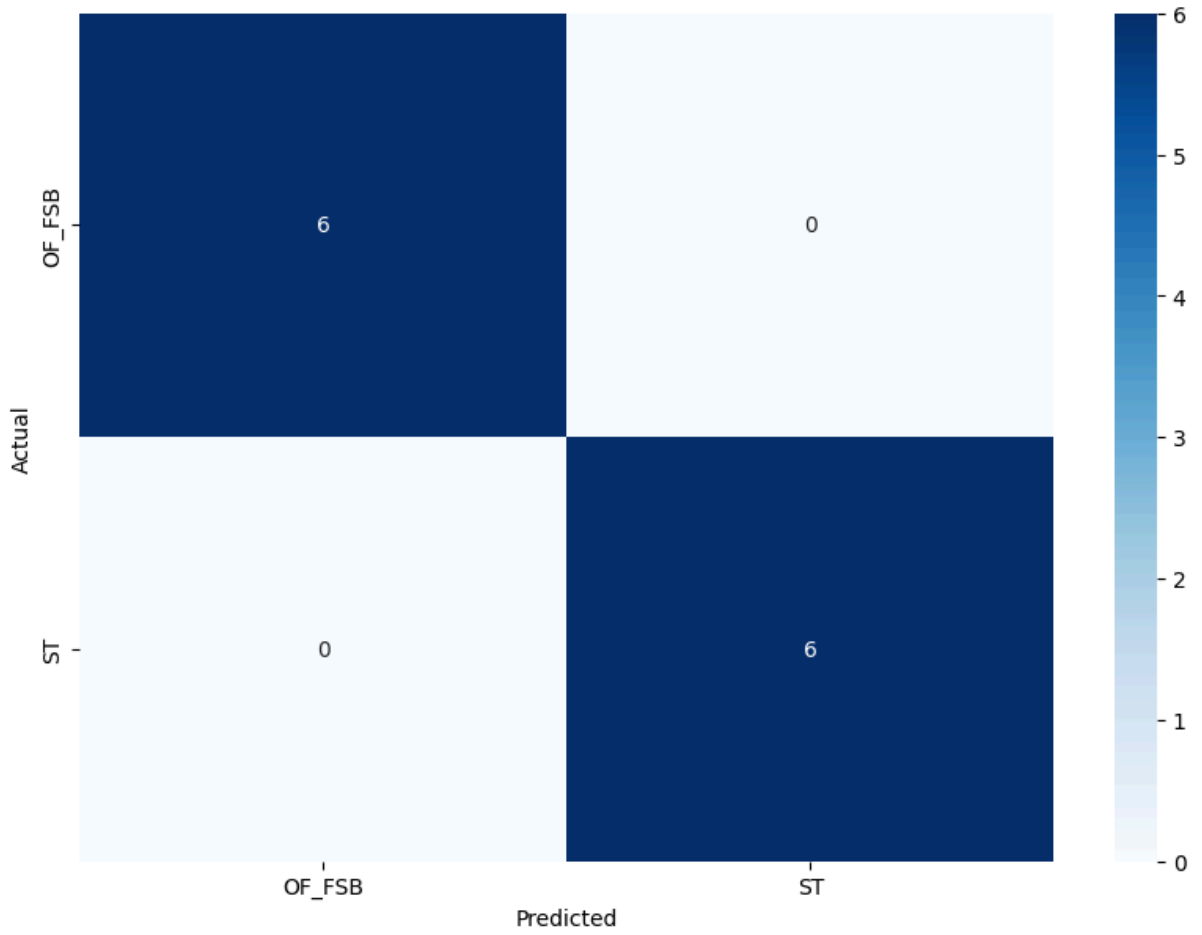
For a more real-world representation, we amalgamated the 'OF' and 'FSB' categories into a singular 'unstable' class, transforming the problem into binary classification. A subsequent grid search yielded an optimal model structure:

- Layer Configuration: [16, 64]
- Dropout Rate: 0.0
- Learning Rate: 0.001
- Activation Function: ReLU
- Regularizer: L2

This modified model, also tested over seven folds, achieved an average accuracy of 98.81% and can reach 100% for some random cases. The corresponding confusion matrix is illustrated in Figure 5 and the feature importance for each parameter was demonstrated in Figure 6.

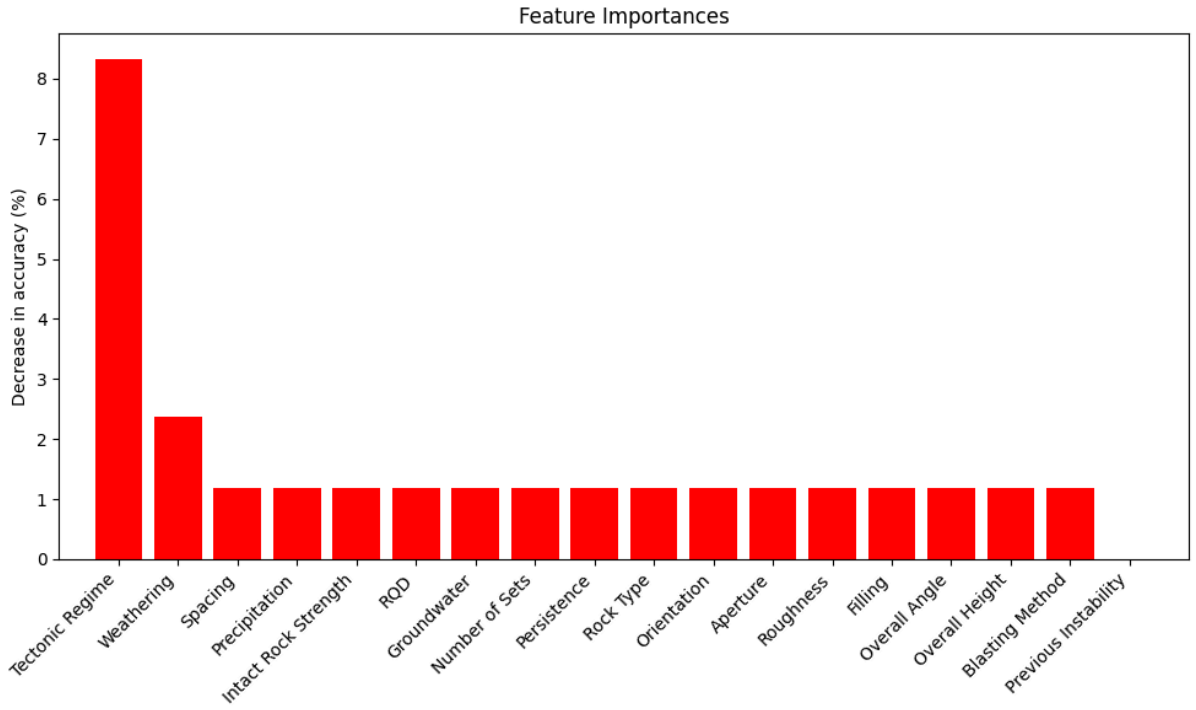


(a) 91% accuracy case

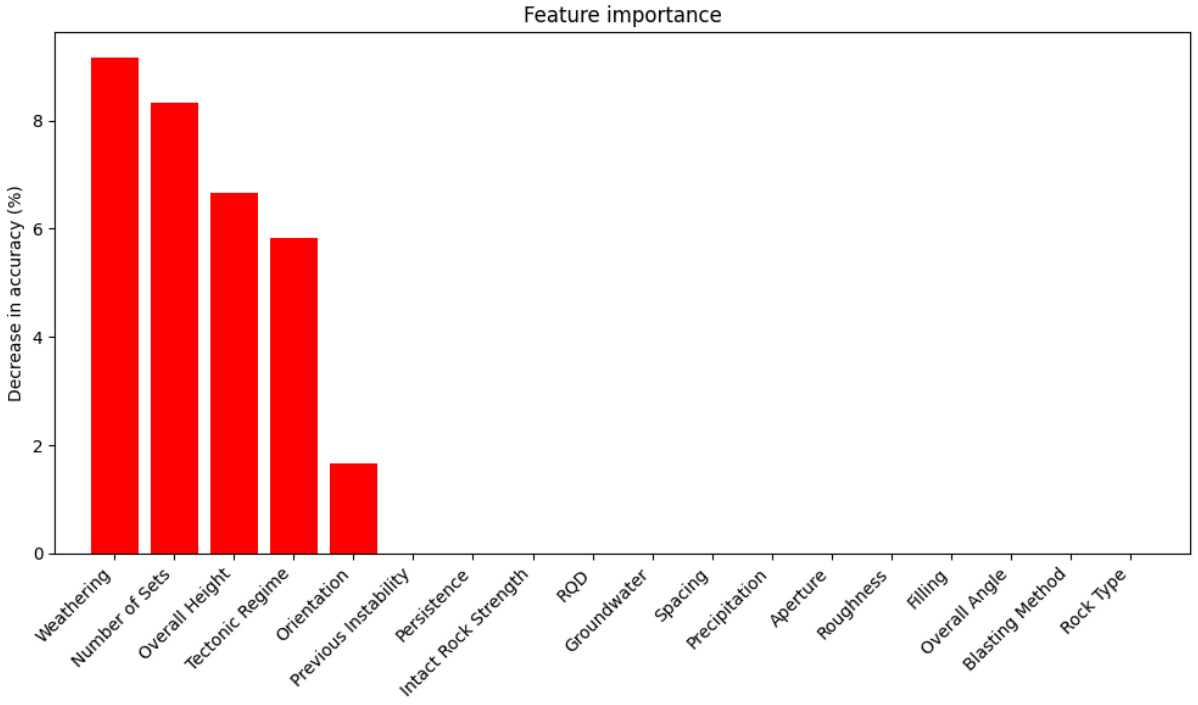


(b) 100% accuracy case

Figure 5: Confusion matrices from the highest-performing model in two classes, showcas- ing (a) a 91% accuracy instance with only one error, and (b) a 100% accuracy instance.



(a) Feature Masking



(b) Permutation Importance

Figure 6: Sensitivity for each parameter in two classes. Each figure was obtained from the following method: (a)Feature Masking. (b)Permutation Importance.

The feature masking results, as illustrated in Figure 6 (a), revealed:

- Groundwater: Its removal consistently results in a significant accuracy decrease, echoing findings from the three-class scenario.
- Features Impacting Accuracy by 1.19%: Attributes like 'Intact Rock Strength', 'RQD', 'Number of Sets', 'Persistence', 'Filling', 'Overall Height', 'Blasting Method', and 'Previous Instability' reduce model accuracy by 1.19% when masked, suggesting their secondary importance.
- Features with Minimal Influence: Masking 'Previous Instability' doesn't impact the accu- racy, hinting at its lesser relevance or redundancy in this binary scenario.

In contrast, the permutation importance results highlighted:

- Features Causing Over 5% Accuracy Reduction: 'Weathering', 'Number of Sets', 'Overall Height', and 'Tectonic Regime' decrease model accuracy by more than 5
- Orientation: This feature affects accuracy by a mere 1.19
- Features without Significant Influence: Most other features, when altered, have no discernible impact on the accuracy, indicating their minor importance or potential redundancy in the binary classification context.

In summary, the binary classification Neural Network exhibited an average accuracy of 98.81% with a standard deviation of roughly 2.92%. This remarkable performance accentuates the model's capability for binary classification endeavors.

These empirical findings validate not only the effectiveness of the adopted algorithms but also elucidate the significance of specific features, facilitating deeper insights from the data. The dis- parities between the accuracies of traditional machine learning techniques and neural networks underscore the potential of deep learning in this specific domain, given appropriate hyperpa- rameters.

User interface presentation

One of the pivotal achievements of this project is the development of a user-friendly interface for our predictive model. Figure 7 showcases the design and layout of the software developed using the Tkinter package.

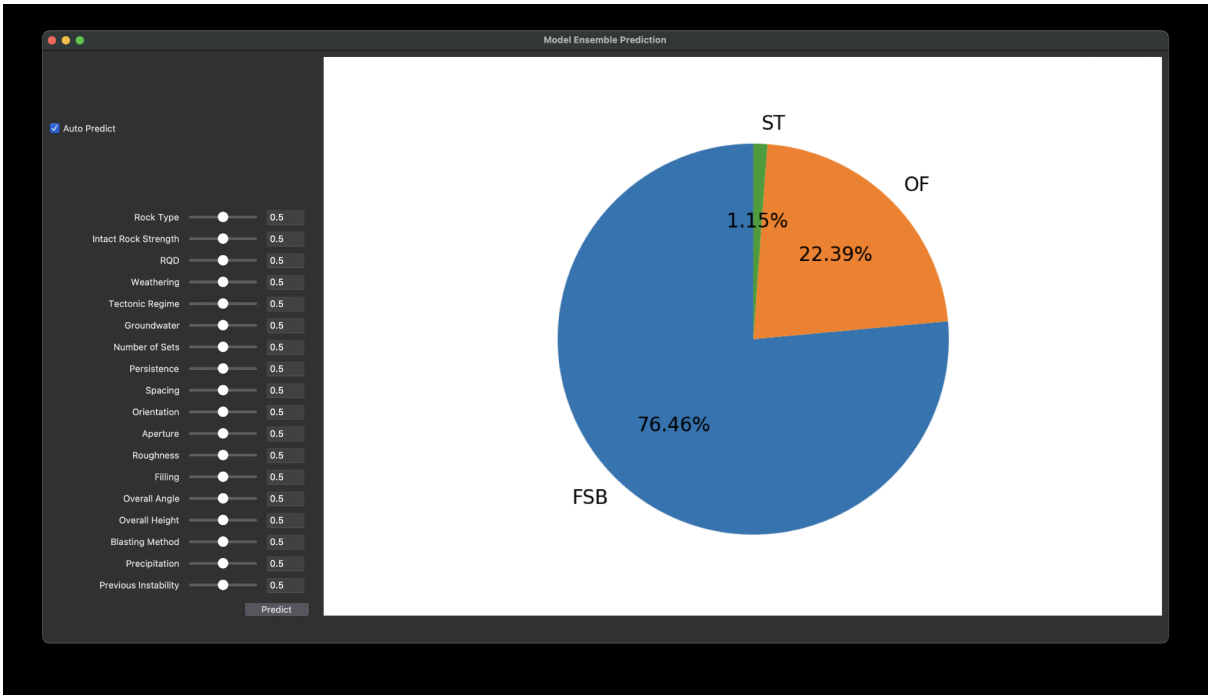


Figure 7: The user interface developed for the predictive model.

The interface, as seen, offers intuitive controls for inputting the required parameters. The pie chart representation at the center provides a graphical breakdown of the prediction results, ensuring clarity for the user.

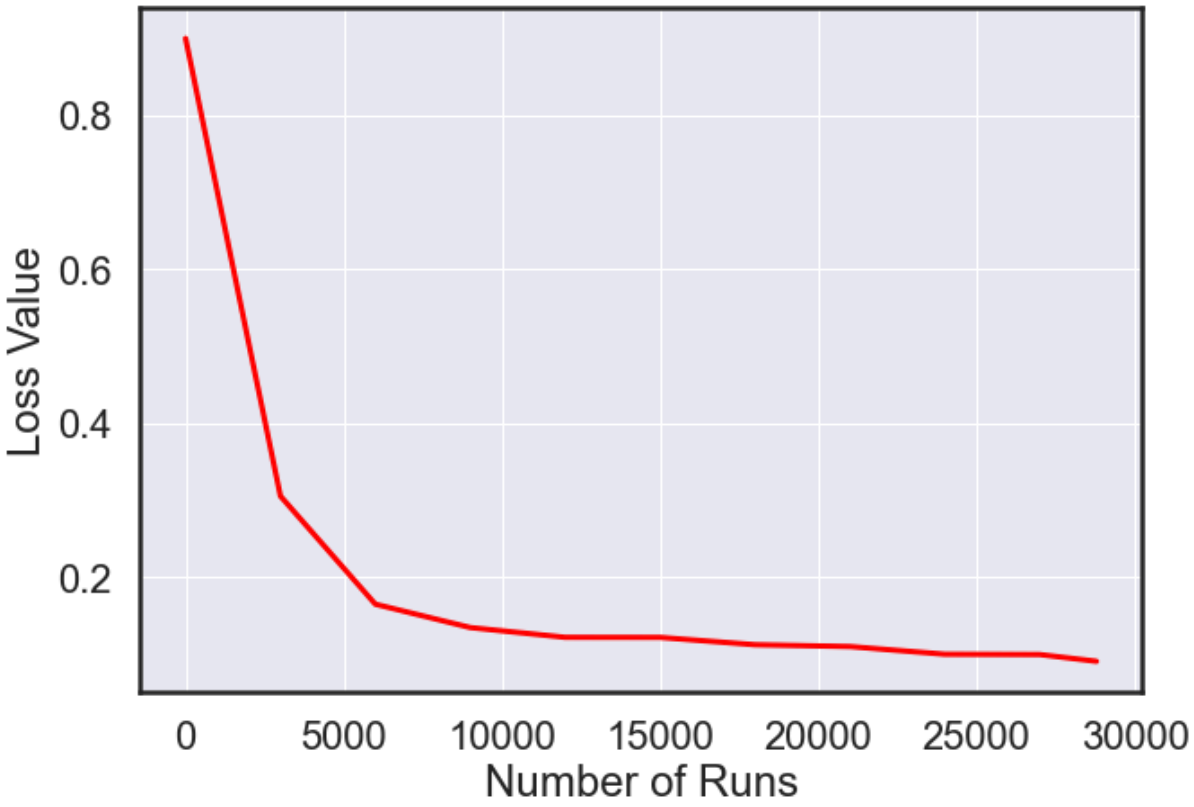
Discussion

Challenges of Hyperparameter Tuning

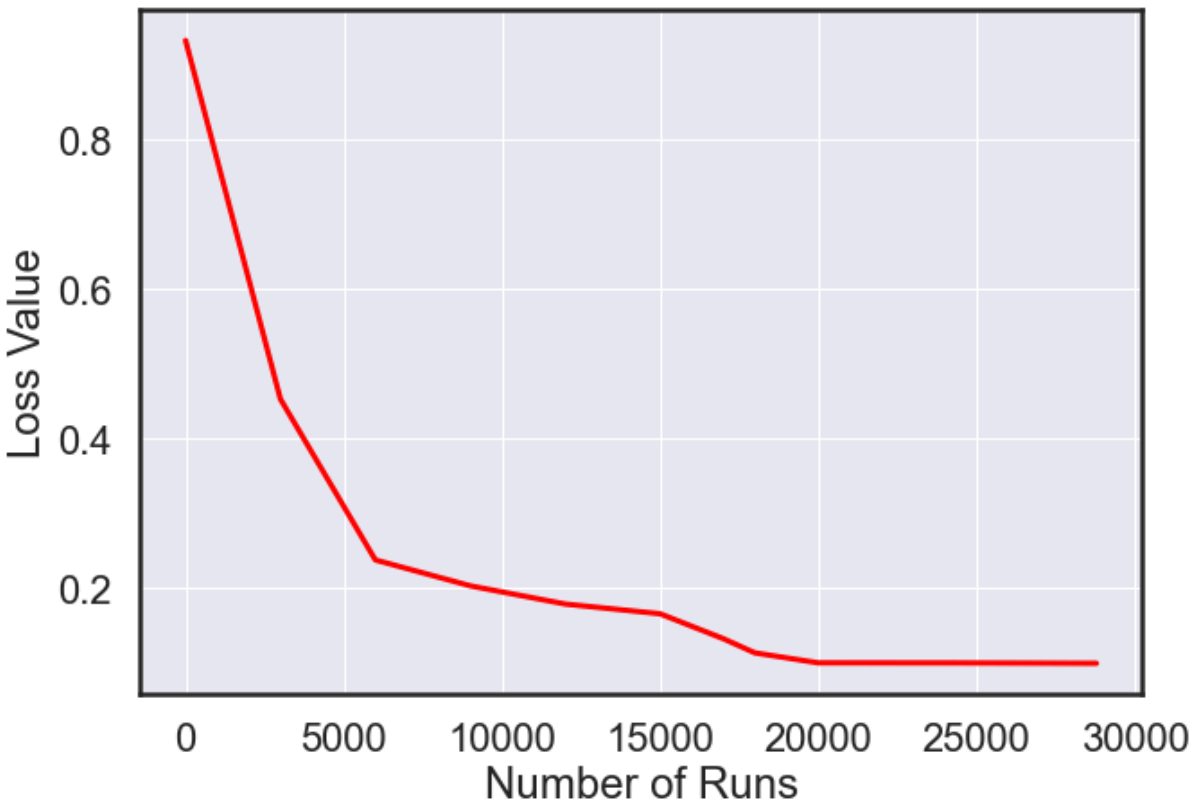
Hyperparameter tuning, specifically determining the most effective configuration for the neural network model, proved to be one of the most formidable challenges faced during the execution of this project[19]. Given the multitude of hyperparameters involved in the construction of a neu- ral network, including the configuration of layers, dropout rate, learning rate, activation function, and regularizer, a vast array of combinations were feasible. The implemented grid search was a robust method for hyperparameter optimization, as it meticulously examined every possible combination to ensure the identification of the best model. However, this exhaustive method comes with significant computational costs, and the temporal demands can be prohibitive, particularly when dealing with larger datasets or more complex models[5]. Furthermore, there's a risk of overfitting with exhaustive grid search methods, as they may lead to a model too specifically tailored to the training data.

These challenges highlight a limitation in the use of traditional hyperparameter tuning meth- ods like grid search for deep learning models. This limitation stimulates

the discussion about the necessity for more efficient optimization methods that can balance between accuracy and practicality. Bayesian optimization, for instance, builds a probability model[2] of the objective function and uses it to select the most promising hyperparameters to evaluate in the true objec- tive function. Genetic algorithms, on the other hand, use principles of natural evolution, such as selection, crossover, and mutation, to determine the optimal hyperparameters[2]. A comparison of the time taken for hyperparameter tuning using various methods can be seen in Figure 8. Future work in this area could explore these more efficient hyperparameter tuning methods to reduce computational burden without significantly compromising model accuracy.



(a) Loss of Grid Search



(b) Loss of Bayesian Algorithm

Figure 8: A comparison chart showing loops taken for hyperparameter tuning with (a) Grid Search, versus (b) Bayesian optimization[2].

From multi-class to binary classification

An essential step in this study was the transformation of the problem from a multi-class to a binary classification, which enabled our model to better reflect real-world scenarios. The amal- gamation of 'OF' and 'FSB' categories into a single 'unstable' class created a model that was more generalized and could distinguish between 'stable' and 'unstable' situations. While this simplified the problem space and improved the model's accuracy, it also introduced a limitation: the model can no longer predict specific types of instability[7]. This trade-off emphasizes the importance of problem formulation, which needs to be carefully constructed depending on the specific application requirements. In some scenarios, binary classification may be sufficient; however, in others, more detailed prediction categories might be necessary.

Deep learning vs. traditional models

The comparison of deep learning and traditional machine learning models revealed the superior ability of the former to handle complex classification tasks, provided they are appropriately tuned[18]. The deep learning model implemented in this study achieved remarkably high ac- curacy of 95.24% and 98.81% for three-class and binary classification settings, respectively. However, the high accuracy achieved by the model does not come without its own set of challenges, particularly regarding interpretability. The intrinsic complexity of deep learning models often results in them being described as ”black boxes,” primarily because the process through which they arrive at a prediction is not transparent or easily comprehensible[36]. Although techniques like feature masking and permutation importance provided some understanding of the model’s decision-making process, these methods were insufficient in providing a comprehensive and detailed explanation.

This challenge related to model interpretability raises an important discussion point for future work. As machine learning models become more prevalent and start making increasingly significant decisions, their transparency and explainability become critical. Various techniques have been proposed to address this challenge, including Local Interpretable Model-Agnostic Explanations (LIME) and Shapley Additive Explanations (SHAP), which provide more detailed insights into the decision-making process of complex models. Future research should consider integrating such techniques to ensure the model’s reliability and transparency.

User Interface Design and Functionality

The design of the user interface, showcased in Figure 7, reflects a deliberate emphasis on ensuring ease-of-use and transparency. Using Tkinter allowed for a cross-platform solution, essential for broad accessibility. The choice of input methods, sliders and manual text entry, was designed to cater to both novice and advanced users.

Moreover, the use of a pie chart to represent prediction probabilities offers several advantages. Firstly, it provides an immediate visual cue on the model’s confidence, enabling users to quickly gauge the level of trust they might place on the prediction. Secondly, by providing a breakdown of probabilities for each class, it offers a nuanced understanding, rather than a binary or singular output. This choice aims to empower users to make informed decisions based on the model’s output.

Potential improvements could include providing tooltips or help sections for users unfamiliar with specific parameters or adding interactive elements to the pie chart for deeper insights into individual class probabilities.

Model prediction times

Lastly, a significant part of our discussion focused on the prediction times of different models. Although traditional machine learning methods like logistic regression exhibited faster prediction times, the study revealed that the advantage of these methods becomes negligible in real- world scenarios. For individual predictions, the difference in prediction time between machine learning models and deep learning models is minimal, leading to similar user experiences in practical use-cases[6]. In situations where speed is not a primary concern, such as in non-real-time applications, this study showed that a higher prediction accuracy should be prioritized, highlighting the benefit of the deep learning model.

Conclusion

In conclusion, this study successfully implemented and optimized a neural network model for a complex classification task, achieving high average accuracies of 95.24% and 98.81% for three- class and binary classifications, respectively. The significant challenges encountered during the execution of the project involved hyperparameter tuning, ensuring the model's applicability to real-world scenarios, and addressing the issue of model interpretability.

A highlight of this project is the user-friendly interface developed using Tkinter, showcasing the feasibility of embedding complex models into user-centric applications. The interface, designed with intuitive controls, not only enables easy input of the necessary parameters but also pro- vides visual feedback via a pie chart, facilitating an immediate understanding of the model's predictions. Such a design accentuates the importance of bridging the gap between intricate machine learning models and end-users, emphasizing the role of effective UI/UX in the practical application of research findings.

Our results reaffirm the capabilities of deep learning in handling complex classification prob- lems, while also illustrating their limitations in terms of interpretability and computational de- mands. Moving forward, future research can build on these findings by investigating more efficient hyperparameter tuning methods, enhancing model interpretability, and tailoring problem formulations to the specific requirements of the application at hand. By focusing on these areas, researchers can further refine and improve the practical application of deep learning models, ensuring they continue to provide valuable and insightful predictions. The positive results achieved in this project, coupled with its user-centric design, provide a strong foundation and direction for these future research endeavors.

References

1. Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. 4(11):e00938.
2. Hussain Alibrahim and Simone A. Ludwig. Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization. In 2021 IEEE Congress on Evolutionary Computation (CEC), pages 1551–1559. IEEE.
3. Awwad H. Altiti, Rami O. Alrawashdeh, Hani M. Alnawafleh, Awwad H. Altiti, Rami O. Alrawashdeh, and Hani M. Alnawafleh. Open Pit Mining. In Mining Techniques - Past, Present and Future. IntechOpen.
4. Andre´ Altmann, Laura Tolos¸i, Oliver Sander, and Thomas Lengauer. Permutation importance: A corrected feature importance measure. 26(10):1340–1347.
5. Shekar B H and Guesh Dagnew. Grid Search-Based Hyperparameter Tuning and Classification of Microarray Cancer Data.
6. Raghavendra Chalapathy, Nguyen Lu Dang Khoa, and Subbu Sethuvenkatraman. Com- paring multi-step ahead building cooling load prediction using shallow machine learning and deep learning models. 28:100543.
7. Sungmoon Cheong, Hoon Sang, Sang-Hoon Oh, and Soo-Young Lee. Support vector machines with binary tree architecture for multi-class classification. 2.
8. Renato L. F. Cunha, Lucas C. Villa Real, Renan Souza, Bruno Silva, and Marco A. S. Netto. Context-aware Execution Migration Tool for Data Science Jupyter Notebooks on Hybrid Clouds. In 2021 IEEE 17th International Conference on eScience (eScience), pages 30– 39.
9. Gao Daqi and Ji Yan. Classification methodologies of multilayer perceptrons with sigmoid activation functions. 38(10):1469–1482.
10. A. P. Engelbrecht, I. Cloete, and J. M. Zurada. Determining the significance of input parameters using sensitivity analysis. In Jose´ Mira and Francisco Sandoval, editors, From Natural to Artificial Neural Computation, Lecture Notes in Computer Science, pages 382–388. Springer.
11. Fatih Ertam and Galip Aydın. Data classification with deep learning using Tensorflow. In 2017 International Conference on Computer Science and Engineering (UBMK), pages 755–758.
12. Antonio Gulli and Sujit Pal. Deep Learning with Keras. Packt Publishing Ltd. Nantian Huang, Guobo Lu, and Dianguo Xu. A Permutation Importance-Based Feature Selection Method for Short-Term Electricity Load Forecasting Using Random Forest. 9(10):767.
13. Mark Jaksa and Zhongqiang Liu. Editorial for Special Issue “Applications of Artificial Intelligence and Machine Learning in Geotechnical Engineering”. 11(10):399.
14. Malik Ali Judge, Asif Khan, Awais Manzoor, and Hasan Ali Khattak. Overview of smart grid implementation: Frameworks, impact, performance and challenges. 49:104056.
15. Gideon Koech and Jean Vincent Fonou-Dombeu. K-Nearest Neighbors Classification of Semantic Web Ontologies. In Christian Attiogbe´ and Sadok Ben Yahia, editors, Model and Data Engineering, Lecture Notes in Computer Science, pages 241–248. Springer International Publishing.
16. Bartosz Krawczyk and Alberto Cano. Locally Linear Support Vector Machines for Imbal- anced Data Classification. In Kamal Karlapalem, Hong Cheng, Naren Ramakrishnan, R. K. Agrawal, P. Krishna Reddy, Jaideep Srivastava, and Tanmoy Chakraborty, editors, Advances in Knowledge Discovery and Data Mining, Lecture Notes in Computer Science, pages 616–628. Springer International Publishing.
17. Dhevan S. Lau and Ritesh Ajoodha. Music Genre Classification: A Comparative Study Between Deep Learning and Traditional Machine Learning Approaches. In Xin-She Yang, Simon

Sherratt, Nilanjan Dey, and Amit Joshi, editors, Proceedings of Sixth International Congress on Information and Communication Technology, Lecture Notes in Networks and Systems, pages 239–247. Springer.
18. Lizhi Liao, Heng Li, Weiyi Shang, and Lei Ma. An Empirical Study of the Impact of Hyper-parameter Tuning and Model Optimization on the Performance Properties of Deep Neural Networks. 31(3):1–40.
19. Shan Lin, Hong Zheng, Chao Han, Bei Han, and Wei Li. Evaluation and prediction of slope stability using machine learning approaches. 15(4):821–833.
20. Tianrui Liu. Applied Machine Learning for Geotechnical Stability.
21. Elena Montan´s, Jose Barranquero, Jorge D´ıez, and Juan Jose´ directed binary trees for multi-class classification. 223:42–55. del Coz. Enhancing
22. M.L. Raymer, W.F. Punch, E.D. Goodman, L.A. Kuhn, and A.K. Jain. Dimensionality re- duction using genetic algorithms. 4(2):164–171.
23. Feng Shaojie, Gao Chen, and Liu Wenbo. Study on failure law of rock mass and slope stability by open-pit combined mining. 248:03013.
24. Arthur Sluyters, Jean Vanderdonckt, and Radu-Daniel Vatavu. Engineering Slidable Graphical User Interfaces with Slime. 5:200:1–200:29.
25. Nitish Srivastava. Improving Neural Networks with Dropout.
26. Daniel Svozil, Vladim´ır Kvasnicka, and Jir´ı Pospichal. Introduction to multi-layer feed-forward neural networks. 39(1):43–62.
27. Hoang Lan Vu, Kelvin Tsun Wai Ng, Amy Richter, and Chunjiang An. Analysis of input set characteristics and variances on k-fold cross validation for a Recurrent Neural Network model on waste disposal rate estimation. 311:114869.
28. Haiying Wang. Logistic Regression for Massive Data with Rare Events. In Proceedings of the 37th International Conference on Machine Learning, pages 9829–9836. PMLR.
29. Zhe Wang, Zonghai Zhu, and Dongdong Li. Collaborative and geometric multi-kernel learning for multi-class classification. 99:107050.
30. Tzu-Tsung Wong. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. 48(9):2839–2846.
31. Tzu-Tsung Wong and Po-Yang Yeh. Reliable Accuracy Estimates from k-Fold Cross Validation. 32(8):1586–1594.
32. Masoud Zare Naghadehi, Rafael Jimenez, Reza KhaloKakaie, and Seyed-Mohammad Esmaeil Jalali. A new open-pit mine slope instability index defined using the improved rock engineering systems approach. 61:1–14.
33. Fei Zhang, Tianhong Yang, Lianchong Li, Jianqing Bu, Tianliang Wang, and Ping Xiao. Assessment of the rock slope stability of Fushun West Open-pit Mine. 14(15):1459.
34. Lifeng Zhang, Hongyan Cui, and Roy E. Welsch. A Study on Multidimensional Medical Data Processing Based on Random Forest. In 2020 5th International Conference on Universal Village (UV), pages 1–5.
35. Quanshi Zhang and Song-Chun Zhu. Visual Interpretability for Deep Learning: A Survey.
36. Zijun Zhang. Improved Adam Optimizer for Deep Neural Networks. In 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), pages 1–2.