Exploring Depth and Width in Multi-Layer Perceptrons for Credit Risk Prediction

NAME: CYNTHIA CHINENYE UDOYE

STUDENT NO: 22029346

GitHub Repository: https://github.com/Cynthiaudoye/MLP_CreditRisk_Tutorial

Web URL: https://cynthiaudoye.github.io/MLP_CreditRisk_Tutorial

Abstract

This tutorial explores how the architecture of Multi-Layer Perceptrons (MLPs), specifically their depth (number of layers) and width (number of neurons per layer), impacts their performance in predicting credit risk. Using the "Credit-G" dataset, we address class imbalance by applying class weights and test various combinations of layers and neurons. Our findings reveal that while increasing the depth of MLPs enhances their learning capacity, it also increases the risk of overfitting. Similarly, adding more neurons often increases computational time without consistently improving accuracy. The best-performing model achieved an AUC-ROC score of 0.80. This tutorial provides a clear and practical guide for using neural networks to tackle imbalanced data challenges in credit risk prediction.

Introduction

Predicting credit risk is very important in finance. It helps banks and other financial institutions figure out which applicants are likely to repay loans, reducing the risk of losing money and increasing profits (Thomas et al., 2002). Credit risk models make the decision-making process in lending more reliable.

In this tutorial, we explore the use of Multi-Layer Perceptrons (MLPs), a type of neural network, for classifying credit data into 'good' and 'bad' risk categories. MLPs are particularly well-suited for this task due to their ability to model complex nonlinear relationships in data (LeCun et al., 2015). However, datasets like the "Credit-G" dataset often exhibit class imbalance, with significantly more 'good' cases than 'bad' (He and Garcia, 2009). To address this, we apply class weights to ensure equitable learning from both classes.

The objective of this tutorial is to investigate how the architecture of MLPs, specifically the number of layers (depth) and neurons per layer (width), influences model performance. While deeper networks can capture intricate patterns and wider layers enhance representational power, both architectures pose challenges, including increased computational cost and potential overfitting (Hinton et al., 2012). Performance will be assessed using metrics such as **AUC-ROC, accuracy, precision, recall, F1-score, and validation loss**, providing a comprehensive evaluation of these trade-offs.

This tutorial will take you through every step of the process, including understanding the dataset, preparing the data, testing different MLP designs, and looking at the results. By the end, we hope to gain valuable insights into building effective neural networks for credit risk prediction and similar challenges in finance.

Dataset and Preprocessing

The "Credit-G" dataset from OpenML contains financial records labelled as 'good' or 'bad' credit risks. It is commonly used in credit risk modelling because it reflects real-world lending data. However, the dataset is imbalanced, with more 'good' cases than 'bad,' making it harder for models to learn from the minority class.

The dataset includes both categorical and numerical features. The following preprocessing steps were applied:

1. **One-Hot Encoding**: Categorical features were converted into numerical values to make them usable by the model.
2. **Scaling**: Numerical features were standardised using StandardScaler to ensure that no single feature dominated the learning process.
3. **Data Splitting**: The dataset was divided into training, validation, and test sets using stratified sampling to maintain the class distribution across all subsets.
4. **Class Weights**: Class weights were calculated and applied during training to give equal importance to 'good' and 'bad' cases.

Key Considerations:

Reproducibility was ensured by using consistent random seeds for data splitting and preprocessing. Ethical considerations, such as avoiding bias amplification, were kept in mind to ensure fair treatment of both classes in the dataset.

Model Architecture and Design

The Multi-Layer Perceptrons (MLP) is a neural network that learns patterns in data. In this tutorial, the MLP was used to classify credit risks as 'good' or 'bad' based on the features of the "Credit-G" dataset (OpenML, n.d.).

1. Model Overview

i. **Input Layer**:
   - Receives the processed dataset, with the number of input nodes equal to the total number of features.

ii. **Hidden Layers**:
   - Multiple hidden layers were tested, varying in the number of layers (depth) and neurons per layer (width).
   - Each layer uses the ReLU (Rectified Linear Unit) activation function to identify complex patterns (LeCun et al., 2015).

iii. **Output Layer**:

- Contains one neuron with a sigmoid activation function, predicting the probability of a record being a 'bad' credit risk.

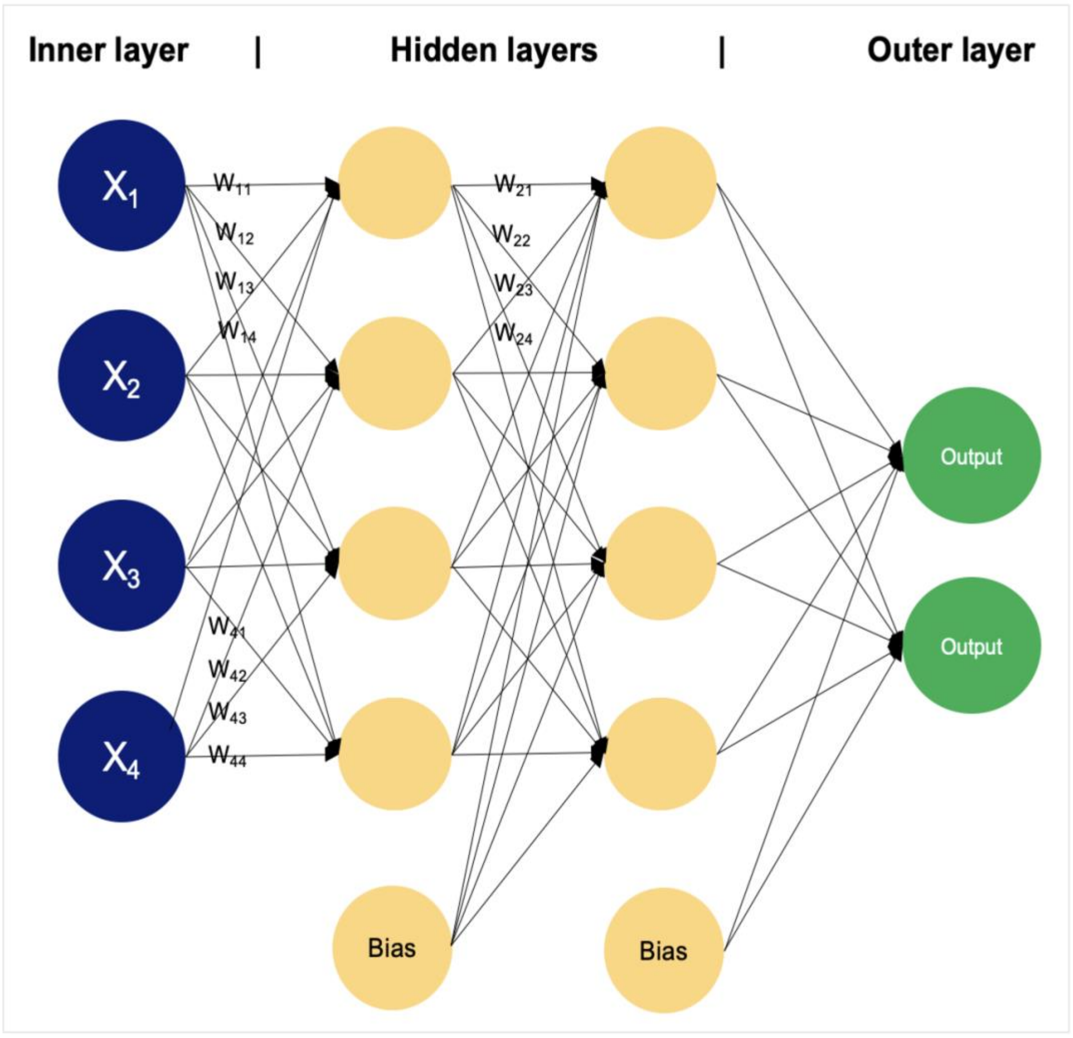
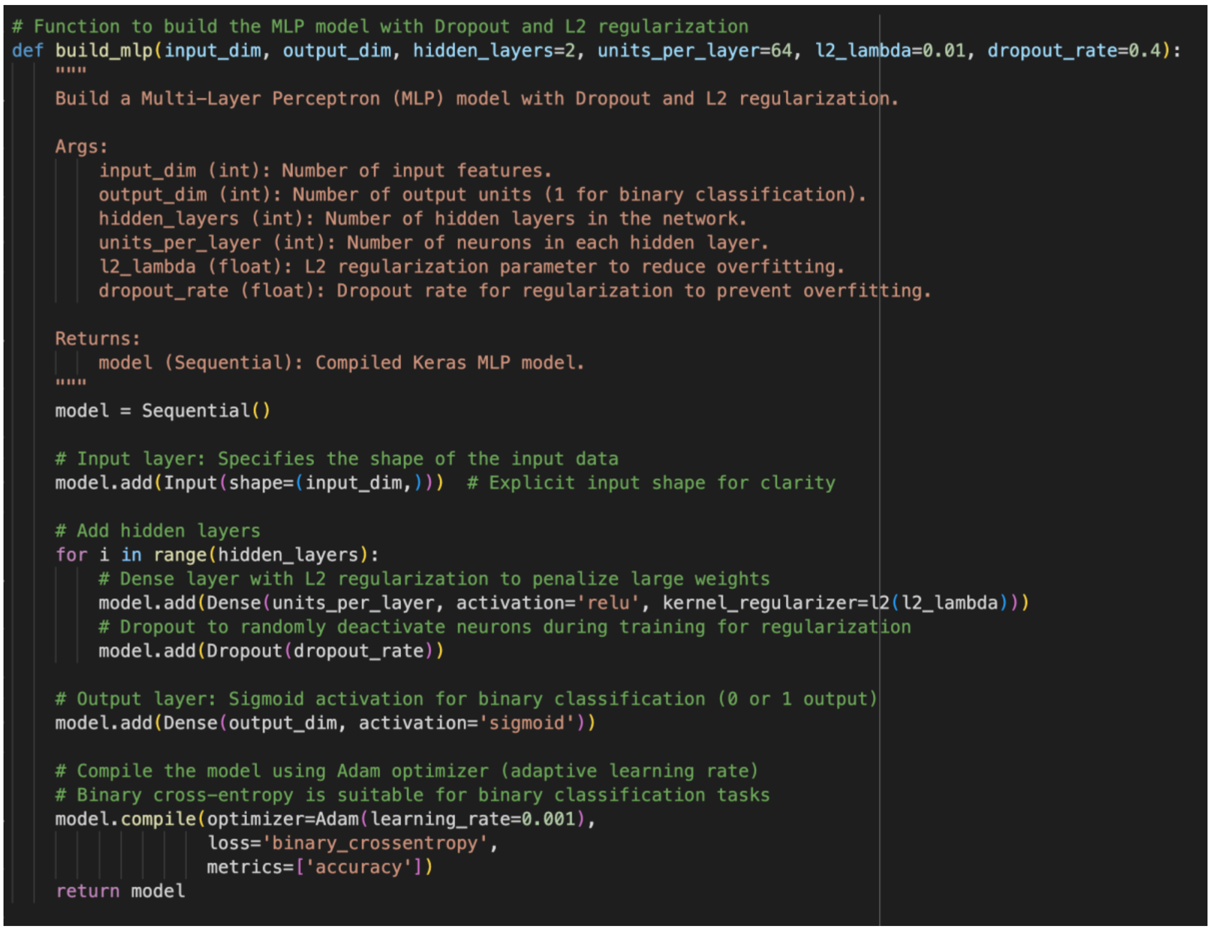


Figure 1: The structure of Multi-Layer Perceptrons (MLP) consists of interconnected layers facilitating complex computations (Datacamp, n.d.).

2. Building the MLP

The build_mlp_model() function creates a multi-layer perceptron (MLP) model using TensorFlow and Keras. It accepts parameters for the number of layers, neurons per layer, activation functions, and optimizer settings. The model is compiled for binary classification with metrics like accuracy and AUC. This design allows flexibility in architecture exploration, enabling customisation of depth and width to optimise performance on the dataset.



```python
# Function to build the MLP model with Dropout and L2 regularization
def build_mlp(input_dim, output_dim, hidden_layers=2, units_per_layer=64, l2_lambda=0.01, dropout_rate=0.4):
    """
    Build a Multi-Layer Perceptron (MLP) model with Dropout and L2 regularization.

    Args:
        input_dim (int): Number of input features.
        output_dim (int): Number of output units (1 for binary classification).
        hidden_layers (int): Number of hidden layers in the network.
        units_per_layer (int): Number of neurons in each hidden layer.
        l2_lambda (float): L2 regularization parameter to reduce overfitting.
        dropout_rate (float): Dropout rate for regularization to prevent overfitting.

    Returns:
        model (Sequential): Compiled Keras MLP model.
    """
    model = Sequential()

    # Input layer: Specifies the shape of the input data
    model.add(Input(shape=(input_dim,)))  # Explicit input shape for clarity

    # Add hidden layers
    for i in range(hidden_layers):
        # Dense layer with L2 regularization to penalize large weights
        model.add(Dense(units_per_layer, activation='relu', kernel_regularizer=l2(l2_lambda)))
        # Dropout to randomly deactivate neurons during training for regularization
        model.add(Dropout(dropout_rate))

    # Output layer: Sigmoid activation for binary classification (0 or 1 output)
    model.add(Dense(output_dim, activation='sigmoid'))

    # Compile the model using Adam optimizer (adaptive learning rate)
    # Binary cross-entropy is suitable for binary classification tasks
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
```

Figure 2: Building the MLP

3. Training and Evaluation with Class Weights

The train_and_evaluate_with_class_weights function trains and evaluates the MLP model while addressing class imbalance by applying class weights. It takes the model, training and validation datasets, class weights, and training parameters such as epochs and batch size. The function outputs the training history and evaluation metrics, including accuracy and AUC, to measure the model's performance effectively.

4. Experimentation and Hyperparameter Tuning

The run_experiments function automates the process of exploring various configurations of the MLP model. It iterates through combinations of depth (number of layers), width (neurons per layer). The results of each configuration are logged to facilitate comparison and identify the optimal architecture for the dataset. This function simplifies architecture exploration and supports effective model optimisation.

5. Improving the Model

- **Dropout**:
  - Randomly disables some neurons during training to prevent the model from relying too heavily on specific patterns (Srivastava et al., 2014).
- **L2 Regularisation**:
  - Adds a penalty for large weights, helping reduce overfitting and improve the model's ability to generalise to new data (Hinton et al., 2012).

6. Impact of Depth and Width:

To explore how the architecture of Multi-Layer Perceptrons (MLPs) impacts credit risk prediction, different combinations of **depths** (number of hidden layers) and **widths** (number of neurons per layer) were examined.

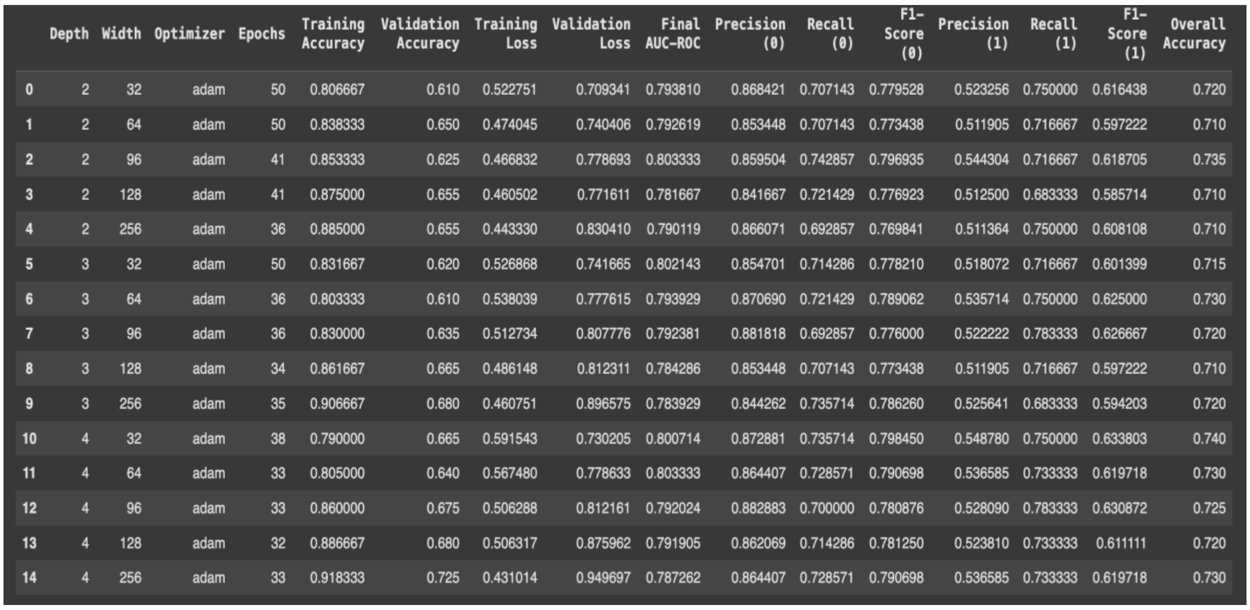
- **Depth (Layers)**:
  - Adding more layers helps the model learn more detailed patterns but can lead to overfitting and slower training if overdone.
- **Width (Neurons per Layer)**:
  - Increasing neurons gives the model more capacity to learn but also increases training time without always improving performance.
- **Optimal Design**:
  - A model with 4 layers and 64 neurons per layer balanced performance and complexity best. Wider models (256 neurons) had higher validation accuracy but didn’t consistently improve AUC-ROC.

7. Using Class Weights

- The "Credit-G" dataset is imbalanced, with more 'good' cases than 'bad.' Class weights were applied to make the model focus more on the minority class ('bad'). This improved the recall for 'bad' cases, balancing the model’s predictions (He & Garcia, 2009).

Evaluation and Results

This table represents the performance of the models with varying configurations of depth, width, and epochs, using the Adam optimiser:



	Depth	Width	Optimizer	Epochs	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss	Final AUC-ROC	Precision (0)	Recall (0)	F1-Score (0)	Precision (1)	Recall (1)	F1-Score (1)	Overall Accuracy
0	2	32	adam	50	0.806667	0.610	0.522751	0.709341	0.793810	0.868421	0.707143	0.779528	0.523256	0.750000	0.616438	0.720
1	2	64	adam	50	0.838333	0.650	0.474045	0.740406	0.792619	0.853448	0.707143	0.773438	0.511905	0.716667	0.597222	0.710
2	2	96	adam	41	0.853333	0.625	0.466832	0.778693	0.803333	0.859504	0.742857	0.796935	0.544304	0.716667	0.618705	0.735
3	2	128	adam	41	0.875000	0.655	0.460502	0.771611	0.781667	0.841667	0.721429	0.776923	0.512500	0.683333	0.585714	0.710
4	2	256	adam	36	0.885000	0.655	0.443330	0.830410	0.790119	0.866071	0.692857	0.769841	0.511364	0.750000	0.608108	0.710
5	3	32	adam	50	0.831667	0.620	0.526868	0.741665	0.802143	0.854701	0.714286	0.778210	0.518072	0.716667	0.601399	0.715
6	3	64	adam	36	0.803333	0.610	0.538039	0.777615	0.793929	0.870690	0.721429	0.789062	0.535714	0.750000	0.625000	0.730
7	3	96	adam	36	0.830000	0.635	0.512734	0.807776	0.792381	0.881818	0.692857	0.776000	0.522222	0.783333	0.626667	0.720
8	3	128	adam	34	0.861667	0.665	0.486148	0.812311	0.784286	0.853448	0.707143	0.773438	0.511905	0.716667	0.597222	0.710
9	3	256	adam	35	0.906667	0.680	0.460751	0.896575	0.783929	0.844262	0.735714	0.786260	0.525641	0.683333	0.594203	0.720
10	4	32	adam	38	0.790000	0.665	0.591543	0.730205	0.800714	0.872881	0.735714	0.798450	0.548780	0.750000	0.633803	0.740
11	4	64	adam	33	0.805000	0.640	0.567480	0.778633	0.803333	0.864407	0.728571	0.790698	0.536585	0.733333	0.619718	0.730
12	4	96	adam	33	0.860000	0.675	0.506288	0.812161	0.792024	0.882883	0.700000	0.780876	0.528090	0.783333	0.630872	0.725
13	4	128	adam	32	0.886667	0.680	0.506317	0.875962	0.791905	0.862069	0.714286	0.781250	0.523810	0.733333	0.611111	0.720
14	4	256	adam	33	0.918333	0.725	0.431014	0.949697	0.787262	0.864407	0.728571	0.790698	0.536585	0.733333	0.619718	0.730

Figure 3: Figure 2: Impact of Depth and Width on Model Performance Metrics

The following metrics were used to evaluate model performance:

- **AUC-ROC**: Measures the model's ability to distinguish between 'good' and 'bad' credit risks.
- **Precision**: Indicates how many of the predicted 'bad' cases were correct.
- **Recall**: Measures how well the model identifies 'bad' credit cases.
- **F1-Score**: Balances precision and recall.
- **Validation Accuracy**: Reflects the overall correctness on the validation set.

2. Presentation of Results

The results are summarised using heatmaps and line plots to show how performance changes with different depths and widths:

I. **Heatmaps**:

   - AUC-ROC, validation accuracy and validation loss scores for all tested configurations were visualised. The best-performing model had **4 layers and 64 neurons per layer**, achieving an AUC-ROC score of **0.80** and a validation accuracy of **75%**.

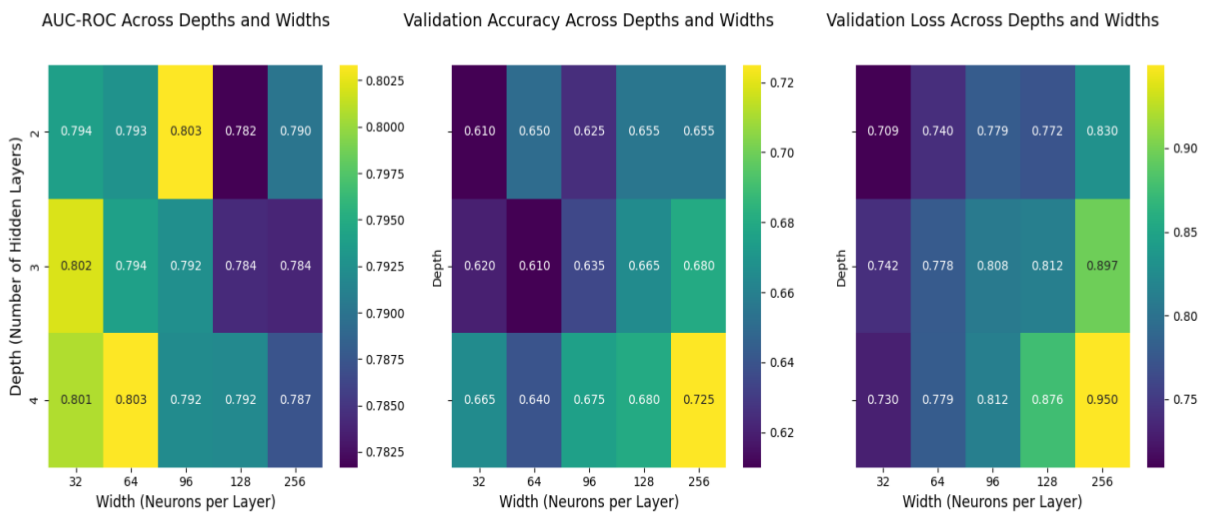


Figure 4: Model Performance Across Depths and Widths

II. Line Plots:

- Trends in AUC-ROC scores across widths for each depth showed that increasing the width generally improved performance up to a certain point (64 neurons) but slightly dropped beyond that.

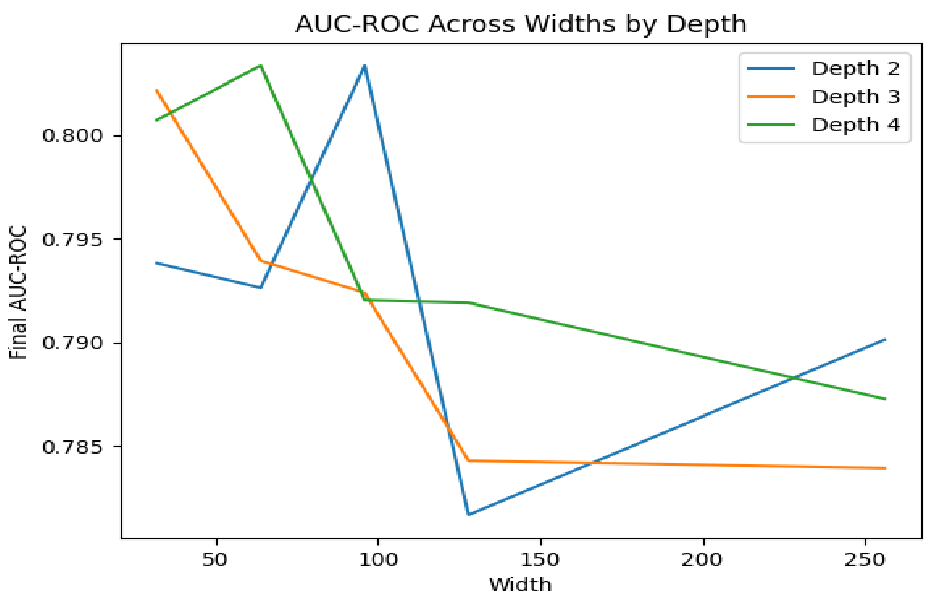


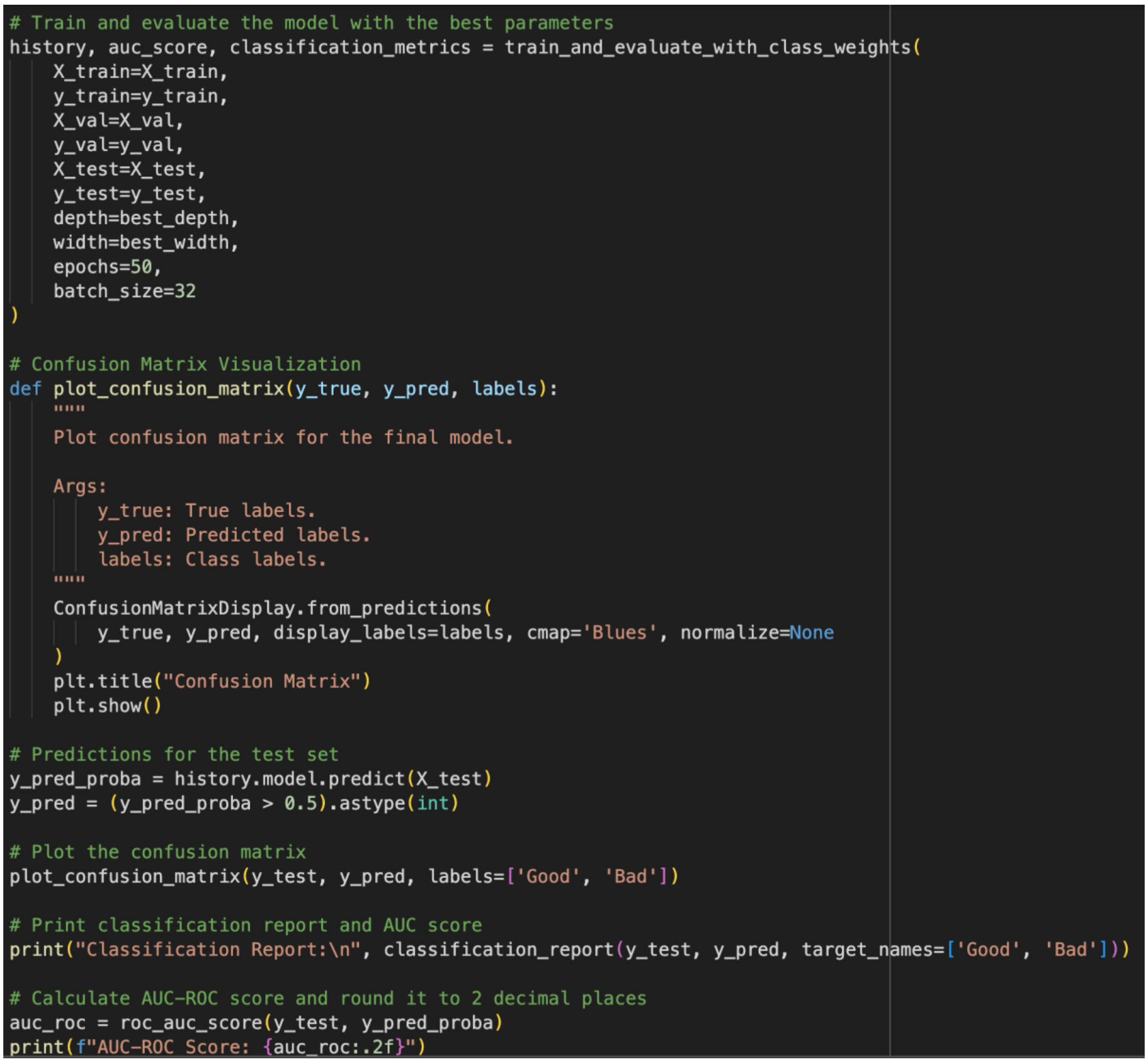
Figure 5: AUC-ROC Trends Across Widths and Depths

3. Key Observations:

- **Impact of Depth**: Adding more layers improved the model's ability to capture patterns, but too many layers (more than 4) could lead to overfitting.
- **Impact of Width**: Wider layers (up to 64 neurons) enhanced learning capacity, but widths beyond this provided diminishing returns.

Discussion & Evaluation of the Best-Performing Model

The best-performing model used 4 layers and 64 neurons per layer. This setup balanced learning complex patterns and avoiding overfitting or unnecessary computational cost. It effectively predicted both "good" and "bad" credit risks, even with an imbalanced dataset.



```python
# Train and evaluate the model with the best parameters
history, auc_score, classification_metrics = train_and_evaluate_with_class_weights(
    X_train=X_train,
    y_train=y_train,
    X_val=X_val,
    y_val=y_val,
    X_test=X_test,
    y_test=y_test,
    depth=best_depth,
    width=best_width,
    epochs=50,
    batch_size=32
)

# Confusion Matrix Visualization
def plot_confusion_matrix(y_true, y_pred, labels):
    """
    Plot confusion matrix for the final model.

    Args:
        y_true: True labels.
        y_pred: Predicted labels.
        labels: Class labels.
    """
    ConfusionMatrixDisplay.from_predictions(
        y_true, y_pred, display_labels=labels, cmap='Blues', normalize=None
    )
    plt.title("Confusion Matrix")
    plt.show()

# Predictions for the test set
y_pred_proba = history.model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)

# Plot the confusion matrix
plot_confusion_matrix(y_test, y_pred, labels=['Good', 'Bad'])

# Print classification report and AUC score
print("Classification Report:\n", classification_report(y_test, y_pred, target_names=['Good', 'Bad']))

# Calculate AUC-ROC score and round it to 2 decimal places
auc_roc = roc_auc_score(y_test, y_pred_proba)
print(f"AUC-ROC Score: {auc_roc:.2f}")
```

Figure 6:Model for the best depth and width

1. Key metrics on the test dataset:

- **Precision (Good):** 0.87
- **Recall (Good):** 0.76
- **F1-Score (Good):** 0.81
- **Precision (Bad):** 0.56
- **Recall (Bad):** 0.73
- **F1-Score (Bad):** 0.64
- **Overall Accuracy:** 75%
- **AUC-ROC:** 0.80

This model captured complex patterns in the data while avoiding overfitting, thanks to dropout and regularization techniques. The use of class weights addressed the imbalanced dataset, ensuring that "bad" credit risks were not ignored.

While the precision for 'bad' cases was lower, indicating some false positives, the high recall ensured the model successfully identified 'bad' credit risks. This balance makes the model highly useful in scenarios where identifying risky credit cases is more critical than avoiding occasional false alarms.

By balancing depth, width, and regularization, this architecture demonstrated strong performance and reliability for credit risk prediction.

2. Confusion Matrix

The confusion matrix for the best model shows:

- **True Positives (Bad predicted as Bad)**: High recall for 'bad' credit indicates the model’s effectiveness in identifying these cases.
- **True Negatives (Good predicted as Good)**: Strong precision for 'good' credit highlights its reliability in identifying trustworthy applicants.

3.Visualisation:

The confusion matrix confirms that while the model performs well overall, it still occasionally misclassifies 'bad' credit as 'good.' This misclassification may be influenced by the inherent imbalance in the dataset. However, the application of class weights during training helps mitigate this issue by emphasising the minority class, thereby balancing the trade-off between precision and recall.

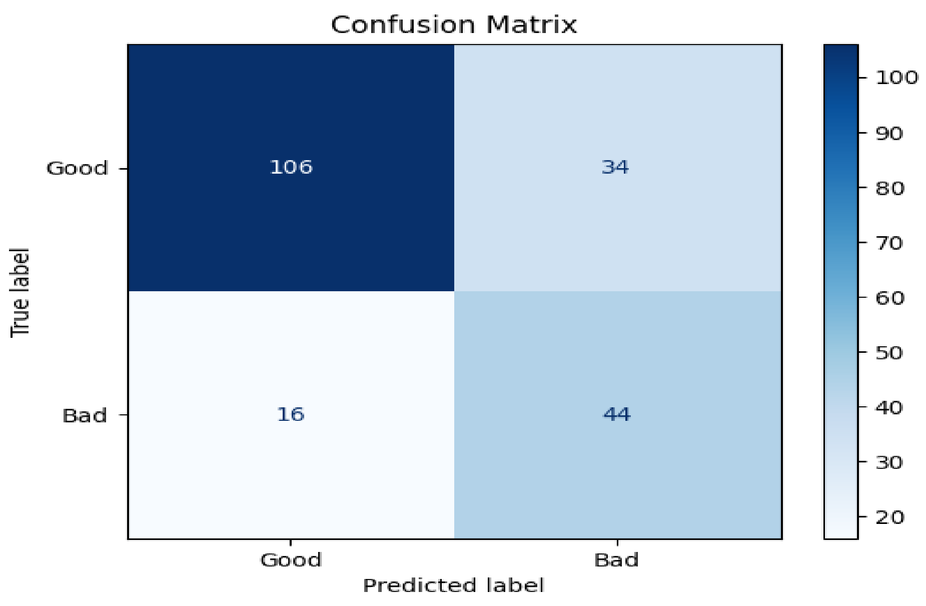


Figure 7: Confusion Matrix for Model Performance Evaluation (Best Layer and Neurons Configuration)

4. Insights

I. **AUC-ROC**:
   a. An AUC-ROC score of **0.80** indicates the model can reliably distinguish between 'good' and 'bad' credit risks. This metric reflects the model's ability to handle imbalanced data effectively.

II. **Precision-Recall Trade-Off**:
   a. The model's precision for 'bad' credit (56%) is lower than its recall (73%), meaning it is more likely to catch potential bad credit cases but at the cost of some false positives.

III. **Class Weights**:

   ○ The use of class weights during training played a key role in ensuring the minority class ('bad') was not overlooked, improving its recall significantly.

Conclusion and Recommendations

This tutorial explored how the architecture of Multi-Layer Perceptrons (MLPs), specifically the number of layers (depth) and neurons per layer (width), affects their performance in predicting credit risk. The best model, with 4 layers and 64 neurons per layer, achieved a good balance between complexity and accuracy. It handled class imbalance effectively using class weights and achieved an AUC-ROC score of 0.80, along with strong precision and recall for both 'good' and 'bad' credit risks.

Practical Applications

The findings demonstrate that MLPs can be a valuable tool for credit risk prediction, helping financial institutions assess loan applications more reliably. By identifying patterns in credit data, these models can reduce default risks and improve decision-making in lending processes.

Recommendations for Future Exploration

1. **Testing on Additional Datasets**: Applying model to other credit datasets could validate its generalisability and robustness.
2. **Hyperparameter Tuning**: Further tuning of learning rates, batch sizes, and regularisation parameters could optimise performance.
3. **Feature Engineering**: Incorporating domain-specific knowledge to create new features might improve prediction accuracy.

In conclusion, this tutorial highlights how thoughtful design choices in neural network architecture can lead to efficient and reliable credit risk models. Future work could extend these insights to other financial applications, offering even broader benefits.

References

Datacamp, n.d., *Multilayer perceptrons in machine learning*. Available at: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning [Accessed 9 Dec. 2024].

Dua, D. & Graff, C., 2019. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. Available at: https://www.openml.org/d/31 [Accessed 9 Dec. 2024].

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep Learning*. Cambridge, MA: MIT Press. Available at: https://www.deeplearningbook.org [Accessed 9 Dec. 2024].

He, H. & Garcia, E.A., 2009. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), pp.1263–1284. Available at: https://doi.org/10.1109/TKDE.2008.239.

Hinton, G., Srivastava, N. & Krizhevsky, A., 2012. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *arXiv preprint*, arXiv:1207.0580. Available at: https://arxiv.org/abs/1207.0580.

LeCun, Y., Bengio, Y. & Hinton, G., 2015. Deep Learning. *Nature*, 521(7553), pp.436–444. Available at: https://doi.org/10.1038/nature14539.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1), pp.1929–1958. Available at: http://jmlr.org/papers/v15/srivastava14a.html.