Team Members

- Member 1
  - Model tuning
- **Member 2**
  - Dataset loading
  - Dataset preprocessing
- Member 3
  - Model design
- Member 4
  - Model Evaluation
  - Experiment Design
  - Introduction

Introduction & Background

Our project is going to be based on loan assessment prediction with the use of a credit risk dataset by Dr. Hans Hofmann. Clients will be classified by two categories, whether they have good credit risk or if they have bad credit risk. The credit risk will be based off many attributes, main ones include the age of the client, their credit history and credit amount, as well as their housing situation. The models used on this dataset will include a one-layer logistical regression model, a single hidden layer neural network, and a multi hidden-layered deep neural network.

The reason for choosing specifically loan assessment for our project is because we want to address a real-world issue when it comes to risk assessment for loans. When it comes to financial issues, precision is extremely important as it can directly affect the livelihood of the people being accepted or rejected for loans. As of late, there seems to be a failure of traditional credit scoring methods in capturing complex relationships between applicant attributes and loan outcomes which leads us to look for a more efficient and effective way to automatically assess loan applications.

Our solution to this problem is the Deep Neural Network (DNN), we want to compare the performance of logistic regression and shallow neural networks to the DNN in hopes that it will perform marginally better in providing faster and most importantly, more reliable lending decisions for clients.

Experiment Design

- **Research Question 1:** How does the performance of a deep neural network compare to traditional machine learning models (specifically Logistic Regression)?
- **Research Question 2:** How does the performance of a deep neural network compare to a shallow neural network?

These are the two research questions that this project will answer by the end of the experiment which are related to the problem statement above. We'll measure our results and compare them using the evaluation metrics of accuracy, precision, recall, and F1-score with a focus on rejecting bad credit scores as not being incorrect about loaning money to a client is more important than accepting good credit for loans.

The hardware used for this experiment includes a Tesla T4 GPU, being run on Google Colab with 16GB GDDR6 memory and 2560 CUDA cores.

For software, the experiment will use Python notebook which will be run on cloud via Google Colab. The libraries used with Python include TensorFlow for the training of neural networks, pandas for data structures that it offers as well as data analysis, NumPy for its collection of high-level mathematical functions, scikit-learn for predictive data analysis, and matplotlib to visualize all our data through graphs and charts.

The name of the dataset being used in our experiment is 'German Credit Data', found through OpenML. The source of this dataset is the UCI Machine Learning Repository, created by Dr. Hans Hofmann. It consists of a size of 1,000 entities, 21 columns (20 attributes, 1 label) with a target variable of 2 label classes, the binary outcomes of 'Good vs Bad' credit risk. This dataset is perfect for our experiment as it has a good variety of different information, both financial information as well as personal information all of which will simulate how it would look to review the credit risks of a client to see if they're fit for a loan.

Here are all 21 columns of attributes and labels listed below:

Feature Name	**Type**	**Description**
Status	Categorical	Status of existing checking account
Duration	Numeric	Duration in months
CreditHistory	Categorical	Credit history record
Purpose	Categorical	Purpose of the loan (car, furniture, etc.)
CreditAmount	Numeric	Amount of credit in Deutsche Marks
SavingsAccount	Categorical	Savings account/bonds status

EmploymentSince	Categorical	Present employment since (in years)
InstallmentRate	Numeric	Installment rate as a percentage of disposable income
PersonalStatusSex	Categorical	Personal status and sex (e.g., male single, female divorced)
Debtors	Categorical	Other debtors or guarantors
ResidenceSince	Numeric	Present residence since (in years)
Property	Categorical	Property ownership
Age	Numeric	Age in years
OtherInstallmentPlans	Categorical	Other installment plans (bank, stores, none)
Housing	Categorical	Housing status (own, rent, free)
ExistingCredits	Numeric	Number of existing credits at this bank
Job	Categorical	Job classification (skilled, unskilled, etc.)
LiablePeople	Numeric	Number of people being liable to provide maintenance
Telephone	Categorical	Has a telephone (yes/no)
ForeignWorker	Categorical	Is a foreign worker (yes/no)

Experiment Procedure

The first part of the procedure will be obtaining the dataset from OpenML. It will be downloaded as an ARFF file and converted into a Pandas dataframe using the scipy library.

Once the dataset is procured, it will be run through our dataset preprocessing. This processing will begin by calculating three ratio columns: monthly credit burden (credit amount / duration), installment per credit (installment commitment / credit amount), and dependents per credit (num dependents / credit amount). Next, it will replace all the numerical columns (including the new ratio columns) with their log computed with np.log. The numerical columns will also be standardized with the scikit-learn StandardScaler. After that, the ordered categories and binary categories (including checking status, savings status, employment, own telephone, foreign worker, and class) will be encoded using the scikit-learn OrdinalEncoder. Finally, the remaining categorical columns will be one-hot encoded using the scikit-learn OneHotEncoder.

Figure 1. The resulting dataset after preprocessing

```
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   checking_status                               1000 non-null   float64
 1   duration                                      1000 non-null   float64
 2   credit_amount                                 1000 non-null   float64
 3   savings_status                                1000 non-null   float64
 4   employment                                    1000 non-null   float64
 5   installment_commitment                        1000 non-null   float64
 6   residence_since                               1000 non-null   float64
 7   age                                           1000 non-null   float64
 8   existing_credits                              1000 non-null   float64
 9   num_dependents                                1000 non-null   float64
 10  own_telephone                                 1000 non-null   float64
 11  foreign_worker                                1000 non-null   float64
 12  class                                         1000 non-null   float64
 13  monthly_credit_burden                         1000 non-null   float64
 14  installment_per_credit                        1000 non-null   float64
 15  dependents_per_credit                         1000 non-null   float64
 16  job_high qualif/self emp/mgmt                 1000 non-null   float64
 17  job_skilled                                   1000 non-null   float64
 18  job_unemp/unskilled non res                   1000 non-null   float64
 19  job_unskilled resident                        1000 non-null   float64
 20  housing_for free                              1000 non-null   float64
 21  housing_own                                   1000 non-null   float64
 22  housing_rent                                  1000 non-null   float64
 23  other_payment_plans_bank                      1000 non-null   float64
 24  other_payment_plans_none                      1000 non-null   float64
 25  other_payment_plans_stores                    1000 non-null   float64
 26  property_magnitude_car                        1000 non-null   float64
 27  property_magnitude_life insurance             1000 non-null   float64
 28  property_magnitude_no known property          1000 non-null   float64
 29  property_magnitude_real estate                1000 non-null   float64
 30  other_parties_co applicant                    1000 non-null   float64
 31  other_parties_guarantor                       1000 non-null   float64
 32  other_parties_none                            1000 non-null   float64
 33  personal_status_female div/dep/mar            1000 non-null   float64
 34  personal_status_male div/sep                  1000 non-null   float64
 35  personal_status_male mar/wid                  1000 non-null   float64
 36  personal_status_male single                   1000 non-null   float64
 37  credit_history_all paid                       1000 non-null   float64
 38  credit_history_critical/other existing credit 1000 non-null   float64
 39  credit_history_delayed previously             1000 non-null   float64
 40  credit_history_existing paid                  1000 non-null   float64
 41  credit_history_no credits/all paid            1000 non-null   float64
 42  purpose_business                              1000 non-null   float64
 43  purpose_domestic appliance                    1000 non-null   float64
 44  purpose_education                             1000 non-null   float64
 45  purpose_furniture/equipment                   1000 non-null   float64
 46  purpose_new car                               1000 non-null   float64
 47  purpose_other                                 1000 non-null   float64
 48  purpose_radio/tv                              1000 non-null   float64
 49  purpose_repairs                               1000 non-null   float64
 50  purpose_retraining                            1000 non-null   float64
 51  purpose_used car                              1000 non-null   float64
dtypes: float64(52)
```

Once the dataset is processed, we will construct one tuner for each type of model we plan to compare (logistic regression, shallow neural network, and deep neural network). The logistic regression model will come from the scikit-learn library and will use the GridSearchCV tuner. The neural networks will come from TensorFlow and use the RandomSearch tuner from Keras instead. The logistic regression model's tuner will search for the optimal solver and C value. Meanwhile, both neural network tuners will search for the optimal activation functions, optimizers, and neurons per layer. Additionally, the deep neural network tuner will search for the optimal number of layers ranging from 2 to 6. The number of neurons per layer will be determined by the number of neurons selected for the top layer. Each subsequent layer in the deep model will have half the neurons of the preceding layer.

Finally, once the best hyperparameters for each model are selected, they will be tested with their respective fit methods with the test set as input. Confusion matrices will be generated for each model by providing the test set labels and predictions to the confusion_matrix function from scikit-learn. The resulting object will be fed into a ConfusionMatrixDisplay object.After that, the object will run its plot method and be displayed to the screen using MatPlotLib. Finally, the evaluation metrics for each model (accuracy, precision, recall, and f1-score) will be calculated and displayed using the classification_report function from scikit-learn.

Procedure summary and pseudocode:

1. Getting the dataset
    - Download dataset from an OpenML URL
    - Convert the saved ARFF file using scipy
    - Convert the resulting object into a Pandas dataframe
2. Dataset preprocessing
    - Copy the dataset
    - Add new ratio columns by dividing one column by another
    - Compute the log of each numerical column using np.log
    - Standardize the logs of each numerical column
    - Replace the original numerical columns with the new versions
    - Replace the ordered categories from strings into numbers using OrdinalEncoder
    - Generate one-hot columns for the independent categories using OneHotEncoder
    - Delete the original independent category columns
3. Model definition and tuning
    - Define a LogisticRegression object
    - Create a hyperparameter search grid
    - Define and fit a GridSearchCV object
    - Define a function for building a shallow model with definable hyperparameters
    - Use the function inside a RandomSearch tuner
    - Define another function for building a deep model
    - Use the new function inside another RandomSearch tuner
4. Model evaluation
    - Use the resulting models from tuning to make predictions on the test set
    - Feed the predictions and test labels into the scikit-learn confusion _matrix function
    - Display the resulting matrix with matplotlib
    - Show the evaluation metrics for each model by feeding the predictions and test labels into the classification_report function

Results

Description:
The results address the following research questions:

- **RQ-1:** How does the performance of a deep neural network compare to traditional machine learning models (specifically Logistic Regression)?
- **RQ-2:** How does the performance of a deep neural network compare to a shallow neural network?

The dataset consisted of 1,000 credit cases, with a class imbalance of approximately 70% labeled as “good” credit and 30% as “bad.” A stratified 80:20 train-test split was used to ensure that the class distribution remained consistent in both training and testing sets.

Three models were evaluated:

1. **Logistic Regression** (baseline)
2. **Shallow Neural Network (SNN)** — single hidden layer
3. **Deep Neural Network (DNN)** — multiple hidden layers

Each model was evaluated on the basis of accuracy, precision, recall, and F1-score. Particular emphasis was placed on each model's ability to correctly identify "bad" credit, which represents the minority class and is more critical in real-world applications.

Table 1. Confusion Matrices

Model	True Neg (Bad)	False Pos	False Neg	True Pos (Good)
Logistic Regression	22	38	13	127
Shallow Neural Network	23	37	15	125
Deep Neural Network	31	29	18	122

Table 2. Evaluation Metrics (Test Set)

Metric	Logistic Regression	Shallow NN	Deep NN
Accuracy	0.74	0.74	0.77
Precision (Bad)	0.37	0.38	0.52
Recall (Bad)	0.37	0.38	0.52
F1-score (Bad)	0.37	0.38	0.52
Recall (Good)	0.91	0.89	0.87
F1-score (Overall)	0.71	0.71	0.76

Discussion of Results

This section discusses the findings related to each research question based on the results presented above.

RQ-1: How does the performance of a deep neural network compare to logistic regression?

Observation 1 (Accuracy):
The deep neural network achieved an accuracy of 77%, outperforming logistic regression (74%). While both models showed strong ability to predict the majority class (“good” credit), the DNN displayed a better balance between detecting both good and bad credit.

Observation 2 (Bad Class Detection):
Logistic regression correctly identified only 37% of bad credit cases (recall = 0.37), while the DNN improved this significantly to 52% (recall = 0.52). This is important in credit modeling, where detecting high-risk individuals is a top priority.

Observation 3 (F1-Score and Trade-offs):
Although logistic regression had a slightly higher recall for "good" credit (0.91 vs. 0.87), the DNN had the highest overall F1-score (0.76), indicating better balance and robustness.

Implication:
The DNN is more applicable in real-world scenarios where false negatives (i.e., failing to detect bad credit) are costly. It provides better recall for the minority class without sacrificing overall performance.

RQ-2: How does the performance of a deep neural network compare to a shallow neural network?

Observation 1 (Overall Performance):
Both neural networks had similar performance in terms of accuracy (74% for SNN vs. 77% for DNN), but the DNN performed significantly better in detecting bad credit (recall = 0.52 vs. 0.38).

Observation 2 (Learning Complexity):
The shallow network provided a modest improvement over logistic regression, but it could not capture the deeper patterns in the data that the DNN was able to leverage due to its additional layers.

Observation 3 (False Positive and Negative Rates):
The DNN more effectively reduced both false positives and false negatives for the minority class, enhancing its suitability for applications requiring balanced and reliable classification.

Implication:
The DNN justifies its increased complexity with superior results. However, if model simplicity and lower training time are priorities, the shallow neural network still offers acceptable performance and is easier to implement.

Final Remarks:
While logistic regression offered strong baseline accuracy, it underperformed in identifying bad credit risks. The shallow neural network introduced modest improvements, but the deep neural network provided the best overall balance between accuracy and recall. Given the importance of identifying high-risk individuals in credit decision-making, the deep neural network is the most practical and effective model among the three evaluated in this study.