

# House Prices - Advanced Regression Techniques

Master's in Intelligent Systems and Applications

Academic year :2024-2025

Course :Deep learning for information systems

Team Members :

Siwar Najjar

Karim Damak

Mahmoud Aziz Ammar

Supervised by :

Mr Alaoui Jawad

# Acknowledgment

We would like to extend our heartfelt gratitude to Professor Alaoui Jawad for his invaluable guidance and expertise, which played a crucial role in the successful completion of the Deep Learning course. His unwavering support, insightful lectures, and constructive feedback provided us with the clarity and motivation needed to excel in this challenging yet rewarding subject.

As a team, we take great pride in the collaborative effort that made this accomplishment possible. Each member contributed their unique skills, dedication, and creativity, fostering an environment of learning, innovation, and excellence. This project serves as a testament to our shared commitment and determination. Thank you to everyone who played a part in making this endeavor a success.

# Contents

<b>List of Figures</b>	<b>6</b>
<b>1 Introduction and Problem Understanding</b>	<b>7</b>
1.1 Overview . . . . .	7
1.2 Problem Definition . . . . .	7
1.3 Objectives . . . . .	7
1.4 Challenges . . . . .	8
1.4.1 Data-Related Challenges . . . . .	8
1.4.2 Model Design and Training Challenges . . . . .	8
1.5 Overview of the CRISP-DM Framework . . . . .	9
1.6 Summary . . . . .	9
<b>2 Data Exploration and Preprocessing</b>	<b>10</b>
2.1 Overview . . . . .	10
2.2 Exploratory Data Analysis (EDA) . . . . .	10
2.2.1 Types of Features . . . . .	10
2.2.2 Target Variable: SalePrice . . . . .	10
2.2.3 Exploring Numerical Features . . . . .	12
2.2.4 Correlation of Numerical Features . . . . .	12
2.2.5 Exploring Categorical Features . . . . .	13
2.3 Data Cleaning and Preprocessing . . . . .	13
2.3.1 Missing Values . . . . .	14
2.3.2 Outlier Handling . . . . .	16
2.3.3 Outlier Handling . . . . .	16
2.3.4 Feature Engineering . . . . .	18
2.3.5 Feature Transformation and Scaling . . . . .	19
2.4 Summary . . . . .	19
<b>3 Model Architecture Design</b>	<b>20</b>
3.1 Overview of Deep Learning Architectures . . . . .	20
3.2 Deep Multi-Layer Perceptron (Deep MLP) . . . . .	20
3.2.1 Description . . . . .	20
3.2.2 Design and Implementation . . . . .	20
3.2.3 Training Details . . . . .	20
3.2.4 Advantages and Limitations . . . . .	21
3.3 Wide Multi-Layer Perceptron (Wide MLP) . . . . .	21
3.3.1 Description . . . . .	21
3.3.2 Design and Implementation . . . . .	22
3.3.3 Training Details . . . . .	22
3.3.4 Advantages and Limitations . . . . .	23
3.4 Convolutional Neural Network (CNN) . . . . .	23
3.4.1 Description . . . . .	23

3.4.2	Design and Implementation . . . . .	23
3.4.3	Training Details . . . . .	23
3.4.4	Advantages and Limitations . . . . .	24
3.5	Recurrent Neural Network (RNN) . . . . .	24
3.5.1	Description . . . . .	24
3.5.2	Design and Implementation . . . . .	25
3.5.3	Training Details . . . . .	25
3.5.4	Advantages and Limitations . . . . .	26
3.6	Summary . . . . .	27
<b>4</b>	<b>Training Optimization Strategies</b>	<b>28</b>
4.1	Overview of Training Techniques . . . . .	28
4.2	Optimization Algorithms . . . . .	28
4.2.1	Adam Optimizer . . . . .	28
4.3	Regularization Techniques . . . . .	28
4.3.1	Dropout Regularization . . . . .	28
4.3.2	L2 Regularization . . . . .	29
4.3.3	Batch Normalization . . . . .	29
4.4	Learning Rate Scheduling . . . . .	29
4.5	Batch Size and Epochs . . . . .	29
4.5.1	Batch Size . . . . .	29
4.5.2	Epochs . . . . .	29
4.6	K-Fold Cross-Validation . . . . .	29
4.7	Data Preprocessing . . . . .	30
4.7.1	Feature Scaling . . . . .	30
4.7.2	Reshaping for CNN and RNN . . . . .	30
4.8	Impact of Strategies on Performance . . . . .	30
4.9	Challenges and Solutions . . . . .	30
4.9.1	Challenge: Overfitting in Deep Models . . . . .	30
4.9.2	Challenge: Training Time for RNNs . . . . .	31
4.9.3	Challenge: Data Preprocessing for CNNs . . . . .	31
4.10	Summary . . . . .	31
<b>5</b>	<b>Model Evaluation and Validation</b>	<b>32</b>
5.1	Validation Framework . . . . .	32
5.1.1	Metrics Used for Evaluation . . . . .	32
5.1.2	Validation Loss Monitoring . . . . .	32
5.1.3	Cross-Validation . . . . .	32
5.2	Discussion of Metrics . . . . .	32
5.3	Challenges in Evaluation . . . . .	33
5.3.1	High Variability in Target Variable . . . . .	33
5.3.2	Generalization to Unseen Data . . . . .	33
5.3.3	Trade-offs Between Metrics . . . . .	33
5.4	Summary . . . . .	33
<b>6</b>	<b>Results and Analysis</b>	<b>34</b>
6.1	Performance Outcomes . . . . .	34
6.1.1	Deep MLP . . . . .	34
6.1.2	Wide MLP . . . . .	34
6.1.3	CNN . . . . .	34
6.1.4	RNN . . . . .	34
6.2	Insights and Analysis . . . . .	35

6.3	Model Validation Loss Results . . . . .	35
6.3.1	Validation Losses for Each Model . . . . .	35
6.3.2	Observations . . . . .	35
6.3.3	Key Takeaways . . . . .	35
6.4	Kaggle Competition Results . . . . .	36
6.5	Summary . . . . .	36
<b>7</b>	<b>Conclusion &amp; Lessons Learned</b>	<b>37</b>
7.1	Key Insights . . . . .	37
7.2	Challenges . . . . .	37
7.3	Lessons Learned . . . . .	37
7.4	Future Work . . . . .	37

# List of Figures

2.1	All features . . . . .	10
2.2	Histogram of SalePrice . . . . .	11
2.3	Boxplot of SalePrice . . . . .	11
2.4	Distribution of Numerical Features . . . . .	12
2.5	Correlation Analysis of Features . . . . .	13
2.6	Categorical Features Impact on SalePrice . . . . .	13
2.7	Categorical Features Impact on SalePrice . . . . .	14
2.8	Box Plots for MSSubClass and LotFrontage . . . . .	16
2.9	Box Plots for LotArea and OverallQual . . . . .	17
3.1	Deep MLP Training . . . . .	21
3.2	Wide MLP Training . . . . .	22
3.3	CNN Training . . . . .	24
3.4	RNN Training . . . . .	26
6.1	Kaggle Submission Results: $R^2$ scores achieved by various model architectures. The Deep MLP model achieved the highest score of 7.07. . . . .	36

# Chapter 1

## Introduction and Problem Understanding

### 1.1 Overview

Predicting the final price of residential properties is a significant challenge in the real estate sector, with applications ranging from valuation to investment decision-making. This project aims to tackle this challenge by employing advanced deep learning techniques to predict the final price of homes in Ames, Iowa. Unlike traditional regression-based approaches, deep learning methods offer the potential to capture complex, non-linear relationships in data, enabling superior predictive accuracy and insights.

The dataset used in this study contains 79 explanatory variables that describe nearly every aspect of residential properties, including physical attributes, location details, and neighborhood characteristics. These variables make the dataset a rich source for feature engineering and model development. This chapter introduces the problem, outlines the project's objectives, highlights the challenges encountered, and provides an overview of the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework that guided the project.

### 1.2 Problem Definition

The primary goal of this project is to predict the final selling price of residential homes in Ames, Iowa. The problem is approached using state-of-the-art deep learning architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Multi-Layer Perceptrons (MLPs). The application of these methods aims to:

- Uncover intricate patterns and relationships within the dataset.

- Achieve superior predictive accuracy compared to traditional regression methods.

- Provide insights into the role of advanced modeling techniques in structured tabular data analysis.

This project bridges the gap between traditional machine learning approaches and advanced deep learning techniques, leveraging the rich dataset to its full potential.

### 1.3 Objectives

The objectives of this project are as follows:

- **Deep Learning Architectures:** Design and implement tailored deep learning models, including CNNs, RNNs, and MLPs, for structured tabular data.

- **Feature Engineering:** Apply innovative techniques to extract meaningful patterns and maximize the dataset's utility.
- **Optimization:** Utilize advanced training techniques, optimization algorithms, and hyperparameter tuning to enhance model performance.
- **Model Comparison:** Compare the effectiveness of various deep learning models in predicting house prices.
- **Accuracy and Validation:** Achieve high predictive accuracy through rigorous evaluation using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and  $R^2$ .

## 1.4 Challenges

The project faced several challenges, which can be categorized into two main areas: data-related challenges and model design challenges.

### 1.4.1 Data-Related Challenges

- **Data Quality:** Managing missing values, outliers, and inconsistent data entries in the 79 explanatory variables.
- **Feature Interactions:** Identifying and capturing complex interactions among a high-dimensional set of features.
- **Feature Engineering:** Transforming raw data into informative features that improve model performance.
- **Dataset Size:** Balancing the relatively small size of the dataset against its feature richness to prevent overfitting.

### 1.4.2 Model Design and Training Challenges

- **Architecture Selection:** Designing deep learning models that are effective for tabular data, such as adapting CNNs and RNNs.
- **Overfitting:** Implementing techniques like dropout, regularization, and early stopping to mitigate overfitting.
- **Hyperparameter Tuning:** Optimizing parameters, including learning rates, number of layers, and dropout rates, to improve model performance.
- **Validation Techniques:** Ensuring robust validation methods, such as cross-validation, to assess the generalization capabilities of the models.
- **Computational Complexity:** Managing the high computational requirements of deep learning, especially for iterative training and hyperparameter tuning.



## 1.5 Overview of the CRISP-DM Framework

The CRISP-DM framework provided a structured approach to the workflow of this project. It consists of six key phases:

- **Business Understanding:** Defined the problem of predicting house prices using structured data to support valuation and decision-making processes.
- **Data Understanding:** Conducted an in-depth exploration of the Ames Housing dataset, identifying key features, patterns, and potential issues such as missing data and outliers.
- **Data Preparation:** Focused on cleaning and transforming data, engineering features, and scaling numerical variables to prepare the dataset for deep learning models.
- **Modeling:** Designed and trained multiple deep learning architectures, including:
  - Deep MLP: Captures feature interactions using dense layers.
  - Wide MLP: Focuses on combining wide and deep representations.
  - CNN: Explores spatial relationships in data.
  - RNN: Captures sequential dependencies in features.
- **Evaluation:** Assessed model performance using key metrics, such as MSE, MAE, and  $R^2$ , to ensure accuracy and robustness.
- **Deployment:** Analyzed the results, discussed their implications, and proposed strategies for real-world deployment of the predictive models.

By following this framework, the project ensured a systematic approach to solving the problem and achieving its objectives.

## 1.6 Summary

This chapter provided an overview of the problem of predicting residential property prices using the Ames Housing dataset, defined the objectives of the study, and discussed the challenges inherent in the data and model design. By leveraging advanced deep learning architectures and adhering to the CRISP-DM framework, the project is poised to address these challenges effectively. The following chapters will delve deeper into the exploratory analysis, data preprocessing, and model development, laying the foundation for achieving accurate and actionable predictions.

# Chapter 2

## Data Exploration and Preprocessing

### 2.1 Overview

Exploring and preprocessing the data is a critical step in predictive modeling. For this project, the Ames Housing dataset, consisting of 79 explanatory variables, was analyzed and prepared for deep learning model development. These variables include categorical and numerical data describing various aspects of residential properties. This chapter delves into the structure of the dataset, data visualization and exploration, and data preprocessing techniques to ensure readiness for modeling.

### 2.2 Exploratory Data Analysis (EDA)

#### 2.2.1 Types of Features

The dataset contains 79 features, categorized as follows:

- **Categorical Features (43):** Examples include *MSZoning*, *Neighborhood*, and *SaleType*.
- **Numerical Features (35, integer):** Examples include *LotArea*, *GrLivArea*, and *OverallQual*.
- **Numerical Features (3, floating-point):** Examples include *LotFrontage*, *MasVnrArea*, and *GarageYrBlt*.

- 43 categorical (object): e.g., *MSZoning*, *Neighborhood*, *SaleType*.
- 35 numerical (integer) (int64): e.g., *LotArea*, *GrLivArea*, *OverallQual*.
- 3 numerical (floating-point) (float64): e.g., *LotFrontage*, *MasVnrArea*.

Figure 2.1: All features

#### 2.2.2 Target Variable: SalePrice

The target variable *SalePrice* comprises 1460 data points with no missing values. Key statistics of *SalePrice* are as follows:

- **Count:** 1460 data points with no missing values for the target variable.
- **Mean (Average):** \$180,921, indicating the average house price in the dataset.

- **Median (50% or 2nd Quartile):** \$163,000, slightly less than the mean, suggesting a right-skewed distribution.
- **Standard Deviation (Std):** \$79,442, indicating a moderately wide range of house prices.
- **Minimum:** \$34,900 (lowest-priced house).
- **Maximum:** \$755,000 (highest-priced house).

**SalePrice Distribution:** The histogram (Figure 2.2) illustrates the right-skewed distribution of *SalePrice*, with most houses concentrated between 100,000 and 200,000. A small number of high-priced houses skew the distribution.

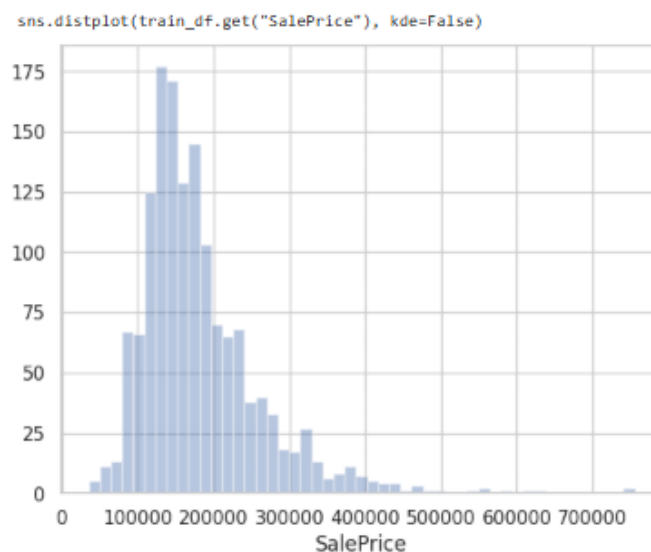


Figure 2.2: Histogram of SalePrice

**SalePrice Boxplot:** The boxplot (Figure 2.3) shows that the majority of house prices are concentrated between 129,975 (25th percentile) and 214,000 (75th percentile), with a median of 163,000. The interquartile range (IQR) reflects moderate variability in house prices. Several significant outliers are present, with the most extreme reaching 755,000. These outliers could skew results in certain models, particularly those sensitive to extreme values.

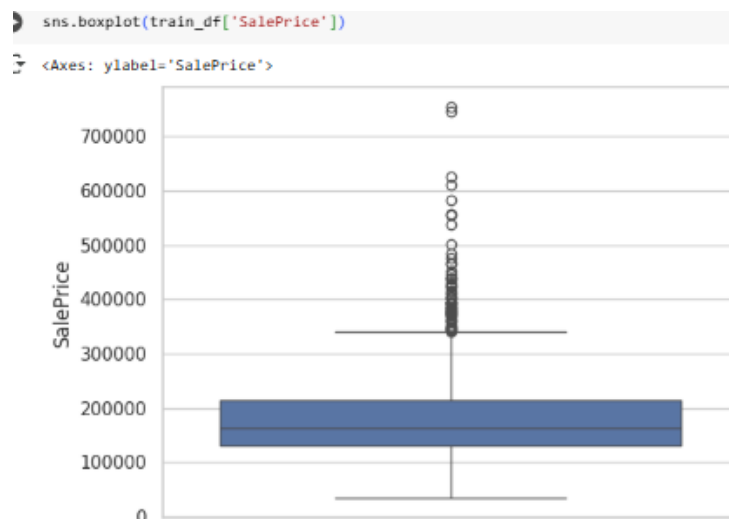


Figure 2.3: Boxplot of SalePrice

## 2.2.3 Exploring Numerical Features

Several numerical features exhibit right-skewed distributions, such as *LotArea*, *BsmtFinSF1*, and *GarageArea*, indicating the presence of outliers. Features like *OverallQual*, *OverallCond*, and *GarageCars* are more discrete, with limited unique values, behaving as categorical-like numerical variables.

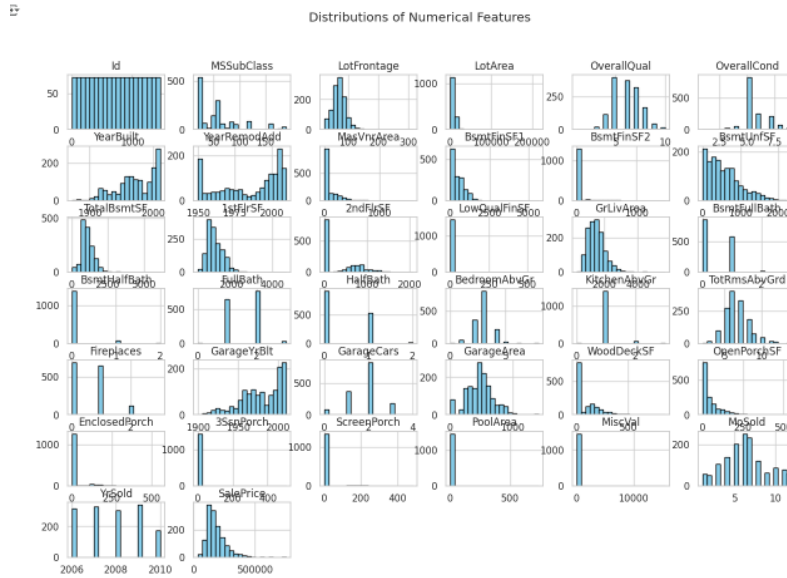


Figure 2.4: Distribution of Numerical Features

## 2.2.4 Correlation of Numerical Features

Top predictors of *SalePrice* based on correlation analysis include:

- *OverallQual* (0.79)
- *GrLivArea* (0.71)
- *GarageCars* (0.64)
- *GarageArea* (0.62)
- *TotalBsmntSF* (0.61)

Moderately impactful features include *1stFlrSF*, *FullBath*, and *TotRmsAbvGrd*, while weak predictors like *MiscVal* (0.02) and *YrSold* (0.03) have little or no impact on *SalePrice*. These findings guide feature selection and prioritization during modeling.

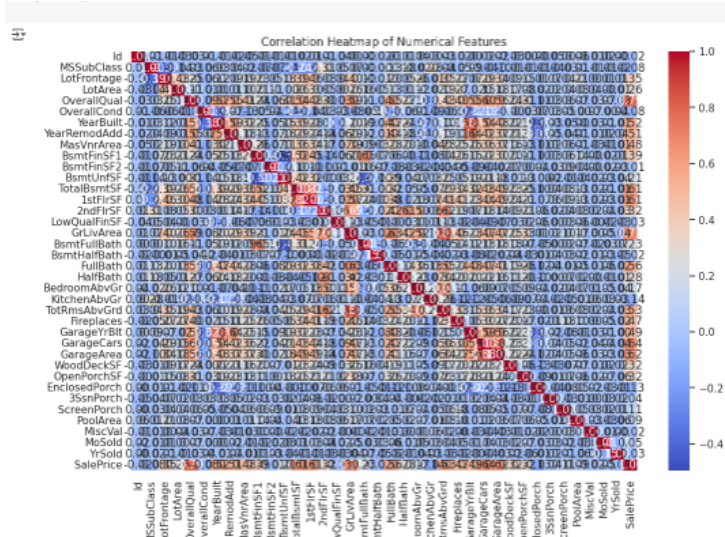


Figure 2.5: Correlation Analysis of Features

## 2.2.5 Exploring Categorical Features

Categorical features such as *Neighborhood*, *MSZoning*, and *SaleType* provide significant insights into patterns of *SalePrice*. These features often indicate location, zoning, or transaction type, which can significantly impact house prices. Features like *Utilities* and *Street* might have limited predictive power and could be excluded or transformed.

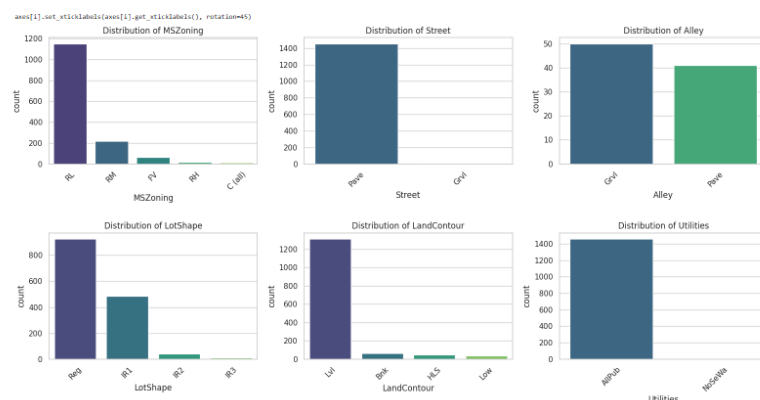


Figure 2.6: Categorical Features Impact on SalePrice

## 2.3 Data Cleaning and Preprocessing

We will Merge train and test data for data preprocessing to ensure uniform processing, We will Handle categorical and numerical missing values by imputing missing values based on feature types and relationships.

### 2.3.1 Missing Values

Numerical columns with missing values:

	Count	Percentage
SalePrice	1459	49.982871
LotFrontage	486	16.649538
GarageYrBlt	159	5.447071
MasVnrArea	23	0.787941
BsmtFullBath	2	0.068517
BsmtHalfBath	2	0.068517
BsmtFinSF1	1	0.034258
BsmtFinSF2	1	0.034258
BsmtUnfSF	1	0.034258
TotalBsmtSF	1	0.034258
GarageCars	1	0.034258
GarageArea	1	0.034258

Categorical columns with missing values:

	Count	Percentage
PoolQC	2909	99.657417
MiscFeature	2814	96.402878
Alley	2721	93.216855
Fence	2348	80.438506
MasVnrType	1766	60.500171
FireplaceQu	1420	48.646797
GarageQual	159	5.447071
GarageFinish	159	5.447071
GarageCond	159	5.447071
GarageType	157	5.378554
BsmtCond	82	2.809181
BsmtExposure	82	2.809181
BsmtQual	81	2.774923
BsmtFinType2	80	2.740665
BsmtFinType1	79	2.706406
MSZoning	4	0.137033
Functional	2	0.068517
Utilities	2	0.068517
Electrical	1	0.034258
KitchenQual	1	0.034258
Exterior2nd	1	0.034258
Exterior1st	1	0.034258
SaleType	1	0.034258

Figure 2.7: Categorical Features Impact on SalePrice

### 2.3.1.1 Imputing Categorical Values

Before imputing categorical values, it is essential to verify the conditions for missing values using the original numerical data from the raw dataset. This ensures that missing values are treated appropriately, either as legitimate data inputs or as actual missing values. The following steps outline the process for categorical value imputation:

- **PoolQC and PoolArea:** If both *PoolQC* (pool quality) and *PoolArea* (pool area in square feet) are missing, these values are treated as legitimate, indicating the absence of a pool.
- **MiscFeature and Fence:** *MiscFeature* and *Fence* are treated as features that can have missing values as legitimate inputs because these attributes may not exist for certain houses.
- **Essential Categorical Columns:** Columns such as *MSZoning*, *Functional*, *Utilities*, *Exterior2nd*, *Exterior1st*, *KitchenQual*, *Electrical*, and *SaleType* cannot have missing values as legitimate inputs. These were imputed using the mode, as this method ensures the most likely value is assigned based on the dataset.
- **FireplaceQu:** Missing values in *FireplaceQu* are considered legitimate only when *Fireplaces* equals 0. If *Fireplaces* is greater than 0, missing values were imputed using the mode.
- **Garage Columns:** For garage-related features such as *GarageType*, *GarageYrBlt*, *GarageFinish*, *GarageCars*, *GarageArea*, *GarageQual*, and *GarageCond*, missing values were treated as legitimate if *GarageArea* equals 0. Otherwise, they were imputed using the mode.
- **Basement Columns:** For basement-related categorical columns (*BsmtQual*, *BsmtCond*, *BsmtExposure*, *BsmtFinType1*, *BsmtFinType2*), missing values were handled as follows:
  - If all basement-related columns, including numerical ones (*BsmtFinSF1*, *BsmtFinSF2*, *BsmtUnfSF*, *TotalBsmtSF*), are missing, the categorical columns were set to *Missing*.
  - Remaining missing values in categorical columns were filled using the mode.

### 2.3.1.2 Imputing Numerical Values: KNN Imputation

For numerical columns with missing values, K-Nearest Neighbors (KNN) Imputation was applied. This imputation method uses the values of the closest rows (neighbors) in the dataset to estimate missing values. The closeness of rows is determined using a distance metric such as Euclidean distance, calculated on features without missing values. The missing value is then estimated as the average (or weighted average) of the corresponding values from the selected nearest neighbors.

KNN is particularly useful because it considers the relationship between features and maintains consistency in the dataset by leveraging patterns in existing data. However, it can be computationally expensive for large datasets and may be sensitive to the choice of features and distance metrics.

The following columns were imputed using KNN:

- **LotFrontage:** Missing values were imputed based on proximity to other houses within the same *Neighborhood*.

- **GarageYrBlt and GarageArea:** Imputed for rows with non-missing values in related columns like *GarageType* and *GarageCars*.
- **Basement Features:** Columns such as *BsmtFinSF1*, *BsmtFinSF2*, *BsmtUnfSF*, *TotalBsmtSF*, *BsmtHalfBath*, and *BsmtFullBath* were imputed considering related categorical and numerical features.
- **MasVnrArea:** Imputed based on similarity to other houses with similar *MasVnrType*.

### 2.3.2 Outlier Handling

Outliers can significantly affect the performance of predictive models, especially those sensitive to extreme values, such as linear regression. The following steps were used to identify and address outliers in the dataset:

### 2.3.3 Outlier Handling

Outliers can significantly affect the performance of predictive models, especially those sensitive to extreme values, such as linear regression. The following steps were used to identify and analyze outliers in the dataset:

- **Identifying Outliers:**
  - *Box Plots:* Box plots were used to visually detect outliers in numerical features by identifying data points outside the interquartile range (IQR). Values below  $Q1 - 1.5IQR$  or above  $Q3 + 1.5IQR$  were flagged as potential outliers. Figure 2.8 highlights outliers for *MSSubClass* and *LotFrontage*, which show extreme values beyond the whiskers.

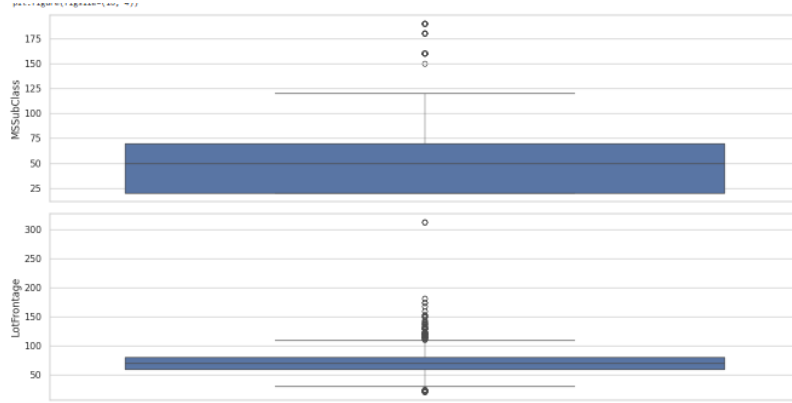


Figure 2.8: Box Plots for MSSubClass and LotFrontage

- *Z-Scores:* Z-scores were calculated to quantify the deviation of data points from the mean. The Z-score is computed using the formula:

$$Z = \frac{(X - \mu)}{\sigma} \quad (2.1)$$

Where  $X$  is the data point,  $\mu$  is the mean of the feature, and  $\sigma$  is the standard deviation. A threshold of 3 was used to identify extreme outliers. This method is particularly useful for detecting data points that deviate significantly from the average while accounting for the spread of the data.

**Advantages of the Z-Score Method:**



- \* It provides a standardized way to detect outliers across different numerical features.
- \* It is computationally efficient and easy to implement for large datasets.
- \* It accounts for both the mean and standard deviation of the data, ensuring robust detection.

- **Handling Outliers:**

- *Decision to Retain Outliers:* In the context of house sales, many of the detected outliers were found to represent legitimate variations in property characteristics. For example, extremely high values for *SalePrice*, *LotArea*, or *GrLivArea* could correspond to premium properties, larger plots, or luxury homes. Removing such data points would reduce the diversity of the dataset and potentially harm the model's ability to generalize.
- *Capping:* In cases where outliers were extreme but legitimate, capping was applied to reduce their undue influence during model training. Extreme values were replaced with the nearest acceptable boundary within  $Q1 - 1.5IQR$  or  $Q3 + 1.5IQR$ .

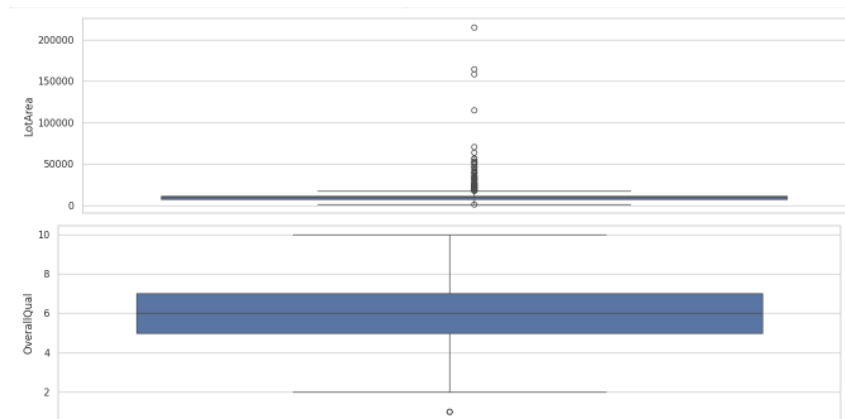


Figure 2.9: Box Plots for LotArea and OverallQual

- **Feature-Specific Adjustments:**

- *SalePrice:* High values of *SalePrice* represent legitimate luxury properties. These outliers were retained to ensure the model captures the full range of housing prices.
- *LotArea:* Large outliers in *LotArea* correspond to properties with significant land size, which are valid variations in the dataset.
- *GrLivArea:* Extremely large values in *GrLivArea* often indicate larger houses or multi-family units, and thus were retained.

By analyzing and choosing to retain most outliers as legitimate variations, the dataset retains its richness and diversity, ensuring the model is trained on a comprehensive range of housing characteristics. This approach enhances the model's robustness and its ability to generalize to real-world scenarios.

### 2.3.4 Feature Engineering

Feature engineering enhances the dataset by creating new features or transforming existing ones to improve the predictive power of models. The process began with handling date-related transformations and proceeded to create derived features to improve interpretability. The following transformations and new features were applied:

- **Handling Dates:** Transformed date-related columns to derive meaningful features:
  - *Garage Age:* Transformed *GarageYrBlt* to represent the age of the garage at the time of sale:

$$GarageYrBlt = YrSold - GarageYrBlt \quad (2.2)$$

- *House Age:* Calculated the difference between the sale year and the construction year or renovation year:
    - \* *YearBuilt:*
    - \* *YearRemodAdd:*
  - Unnecessary columns *YrSold* and *MoSold* were dropped as they were no longer needed.
- **Property Size:** Created a new feature *TotalSF*, which represents the total property size, calculated as:

$$TotalSF = TotalBsmfSF + 1stFlrSF + 2ndFlrSF \quad (2.3)$$

- **Basement Features:** Combined basement-related columns:
  - *TotalBsmfFinishedSF:* Sum of *BsmfFinSF1* and *BsmfFinSF2*.
  - *TotalBsmfArea:* Sum of *TotalBsmfFinishedSF* and *BsmfUnfSF*.

Dropped individual basement columns *BsmfFinSF1*, *BsmfFinSF2*, *BsmfUnfSF*, and *TotalBsmfSF*.

- **Floor Area:** Created *TotalFlrSF*, representing the total floor area:

$$TotalFlrSF = 1stFlrSF + 2ndFlrSF \quad (2.4)$$

Dropped *1stFlrSF* and *2ndFlrSF* after calculation.

- **Bathroom Features:** Combined bathroom-related columns into a single feature *totalbaths*:

$$totalbaths = BsmfFullBath + FullBath + 0.5 * (BsmfHalfBath + HalfBath) \quad (2.5)$$

Dropped *BsmfFullBath*, *FullBath*, *BsmfHalfBath*, and *HalfBath*.

- **Porch Features:** Combined porch-related columns into *totalporchsf*:

$$totalporchsf = OpenPorchSF + 3SsnPorch + EnclosedPorch + ScreenPorch + WoodDeckSF \quad (2.6)$$

Dropped *OpenPorchSF*, *3SsnPorch*, *EnclosedPorch*, *ScreenPorch*, and *WoodDeckSF*.

These feature engineering steps reduced dimensionality, improved interpretability, and enhanced the predictive power of the dataset, ensuring robust inputs for modeling.

### 2.3.5 Feature Transformation and Scaling

Feature transformation and scaling ensure the dataset is optimized for predictive modeling by addressing skewness, encoding categorical features, and normalizing numerical values. The following steps were implemented:

#### 2.3.5.1 Handling Skewness

- **Detecting Skewness:** Numerical features were analyzed for skewness using the skewness metric. Features with an absolute skewness value greater than 0.5 were flagged as significantly skewed.
- **Transforming Skewed Features:** Features with moderate skewness were transformed as follows:
  - *Log Transformation:* Applied to features with positive values to reduce skewness:
$$data[feature] = \log 1p(data[feature]) \quad (2.7)$$
  - *Square Root Transformation:* Applied to features with zero or negative values, where values were clipped at zero:

$$data[feature] = \sqrt{data[feature].clip(lower = 0)} \quad (2.8)$$

- **Rechecking Skewness:** After transformations, skewness was recalculated to confirm improved symmetry in distributions.

#### 2.3.5.2 Encoding Categorical Features

- **Ordinal Encoding:** Features with ordinal relationships were encoded using rank-based mappings.
- **Binary Encoding:** Binary features were encoded as 0 or 1.
- **One-Hot Encoding:** Remaining categorical features were one-hot encoded to ensure all categorical levels were represented without introducing ordinal relationships.

#### 2.3.5.3 Scaling and Splitting Data

- **Scaling Numerical Features:** Numerical features were scaled using standardization, where each feature was transformed to have a mean of 0 and a standard deviation of 1. This ensures that features with different scales do not disproportionately influence the model.
- **Data Splitting:** The dataset was split into training and testing sets to evaluate model performance effectively. The standard split ratio was 80

## 2.4 Summary

These transformations and scaling steps prepared the dataset for robust and effective predictive modeling, ensuring consistency and optimized performance across models. By systematically exploring, cleaning, and preprocessing the dataset, this phase ensured that the data was well-prepared for deep learning modeling. Further details on feature engineering and transformations are discussed in subsequent chapters.

# Chapter 3

## Model Architecture Design

### 3.1 Overview of Deep Learning Architectures

Deep learning architectures are designed to automatically learn hierarchical feature representations from raw data. In this project, we explore four primary architectures tailored for structured tabular data: Deep Multi-Layer Perceptrons (Deep MLPs), Wide Multi-Layer Perceptrons (Wide MLPs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs). Each architecture has unique characteristics and is evaluated based on its suitability for the task of predicting house prices.

### 3.2 Deep Multi-Layer Perceptron (Deep MLP)

#### 3.2.1 Description

The Deep Multi-Layer Perceptron (Deep MLP) is a fully connected feedforward neural network with multiple hidden layers. It is designed to learn complex feature interactions by increasing the depth of the architecture.

#### 3.2.2 Design and Implementation

- **Input Layer:** Accepts the preprocessed tabular data, where each feature is normalized.
- **Hidden Layers:** Composed of several dense layers with Swish activation functions, allowing the network to learn intricate patterns.
- **Dropout Regularization:** Dropout layers are used to prevent overfitting by randomly disabling neurons during training.
- **Batch Normalization:** Normalizes the activations to stabilize and speed up training.
- **Output Layer:** A single neuron with linear activation to predict the continuous target variable (house prices).

#### 3.2.3 Training Details

The Deep MLP model was compiled using the Adam optimizer with a learning rate of 0.001. The Mean Squared Error (MSE) was used as the loss function, with Mean Absolute Error (MAE) as an additional evaluation metric. Early stopping and learning rate reduction were implemented to optimize performance. The model architecture includes:

- 512 neurons in the first dense layer with Swish activation.
- Gradual reduction in neurons across layers to 256, 128, and 64.
- Regularization through L2 penalty and dropout layers at 0.4 and 0.3 rates.
- A total of 200 epochs with a batch size of 32.

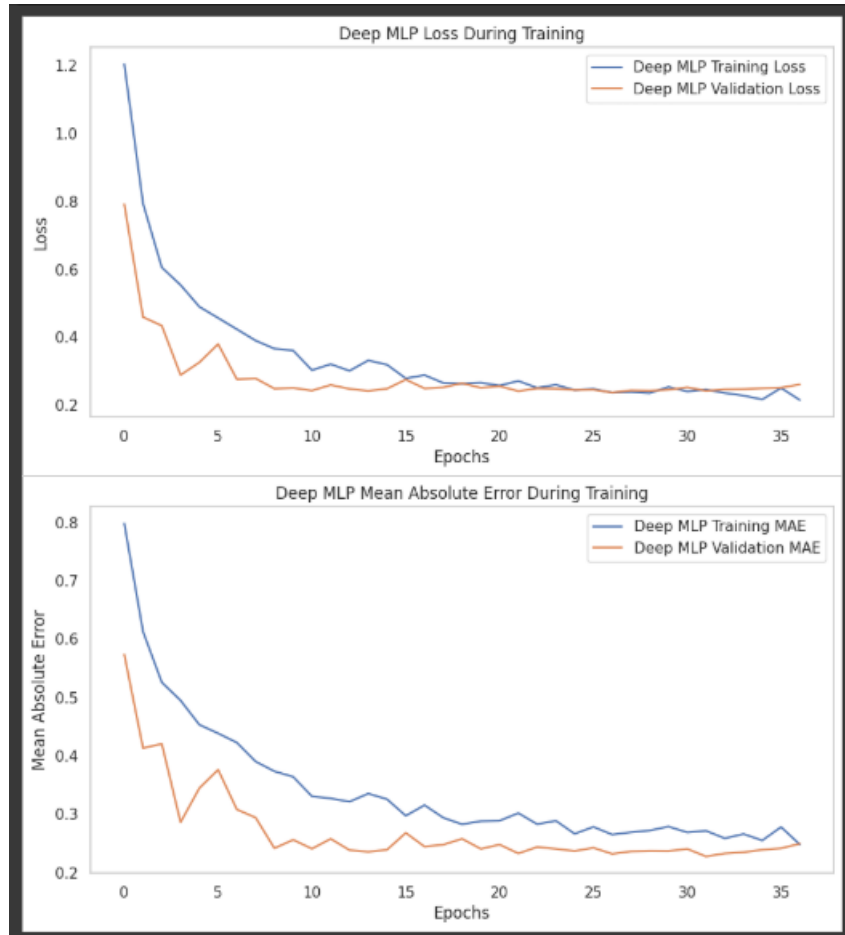


Figure 3.1: Deep MLP Training

### 3.2.4 Advantages and Limitations

- **Advantages:**
  - Capable of capturing complex non-linear interactions between features.
  - Highly customizable for different datasets and tasks.
- **Limitations:**
  - Increased depth can lead to overfitting if not properly regularized.
  - Computationally more expensive than shallower models.

## 3.3 Wide Multi-Layer Perceptron (Wide MLP)

### 3.3.1 Description

The Wide Multi-Layer Perceptron (Wide MLP) is a variant of the MLP that focuses on increasing the number of neurons per layer rather than the depth of the network. This

approach allows the model to capture a broader range of feature interactions.

### 3.3.2 Design and Implementation

- **Input Layer:** Accepts the normalized input features.
- **Hidden Layers:** Composed of wide dense layers with ReLU activation functions, designed to process more features simultaneously.
- **Dropout Regularization:** Dropout layers are added to reduce overfitting.
- **Output Layer:** A single neuron with linear activation for regression.

### 3.3.3 Training Details

The Wide MLP model was trained with the Adam optimizer, using a slightly larger batch size of 64 to balance computational efficiency and model stability. Early stopping was employed with a patience of 10 epochs. The architecture includes:

- 1024 neurons in the first dense layer.
- Gradual reduction in neurons across subsequent layers to 512 and 256.
- Dropout layers with rates of 0.4 and 0.3.
- A total of 100 epochs with validation monitoring.

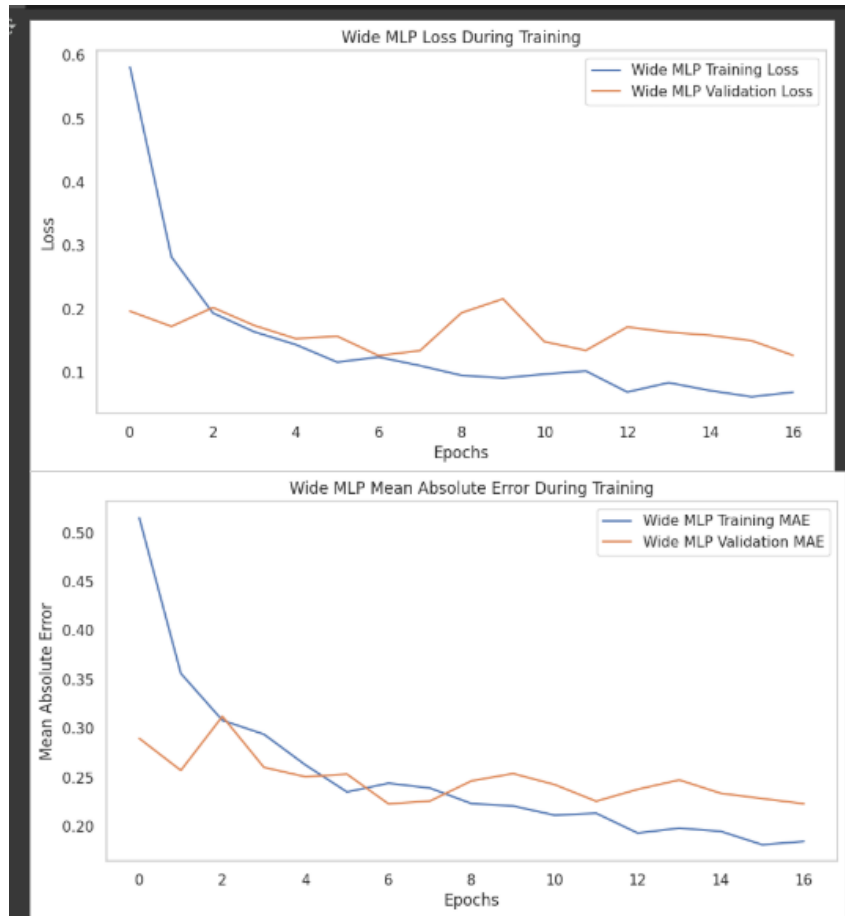


Figure 3.2: Wide MLP Training

### 3.3.4 Advantages and Limitations

- **Advantages:**
  - Handles a larger number of features effectively.
  - Simpler training process compared to deeper networks.
- **Limitations:**
  - May not capture deep feature interactions as effectively as deep architectures.
  - Larger layers require more computational resources.

## 3.4 Convolutional Neural Network (CNN)

### 3.4.1 Description

Convolutional Neural Networks (CNNs) are typically used for image data but can be adapted for tabular data by reshaping features into a grid-like structure. CNNs excel at capturing spatial patterns and local dependencies.

### 3.4.2 Design and Implementation

- **Input Layer:** The tabular data is reshaped into a 2D grid to mimic the structure of image data.
- **Convolutional Layers:** Extract local patterns from the reshaped data using small convolutional kernels.
- **Pooling Layers:** Reduce the spatial dimensions of feature maps, retaining only the most significant features.
- **Flatten and Dense Layers:** Convert the feature maps into a format suitable for regression tasks.
- **Dropout Regularization:** Dropout layers are added to prevent overfitting by reducing over-reliance on specific neurons.
- **Output Layer:** A single neuron with linear activation for continuous target prediction.

### 3.4.3 Training Details

The CNN model was trained on reshaped data using 1D convolutional layers with ReLU activation. The model parameters included:

- Two convolutional layers with 32 and 64 filters, kernel size 2.
- Max-pooling layers to downsample feature maps.
- Dense layers with 128 neurons, followed by a dropout layer with a rate of 0.3.
- 100 epochs with a batch size of 32 and early stopping.

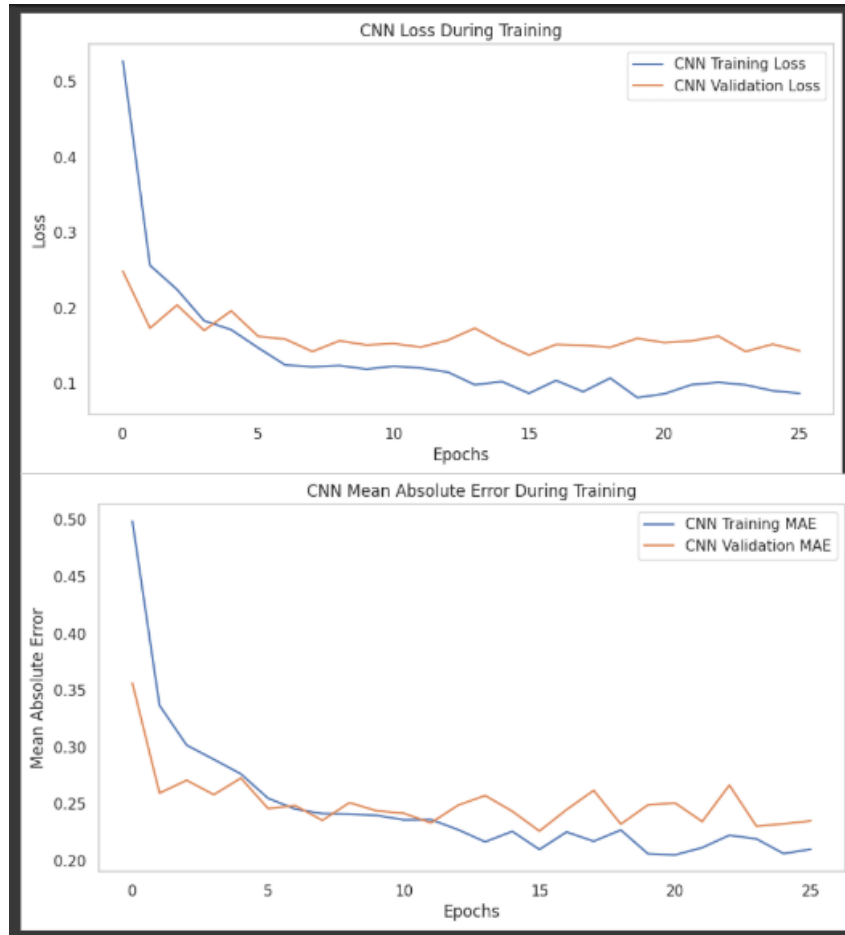


Figure 3.3: CNN Training

### 3.4.4 Advantages and Limitations

- **Advantages:**
  - Captures spatial relationships between features effectively.
  - Robust against noisy or redundant features.
  - Can handle high-dimensional feature spaces effectively.
- **Limitations:**
  - Requires additional preprocessing to reshape data.
  - Computationally more expensive compared to MLPs.
  - Interpretation of learned spatial patterns can be challenging for tabular data.

## 3.5 Recurrent Neural Network (RNN)

### 3.5.1 Description

Recurrent Neural Networks (RNNs) are designed to process sequential data. In this project, RNNs are adapted to model relationships between features by treating them as sequences.



### 3.5.2 Design and Implementation

- **Input Layer:** The tabular data is reshaped into a sequence format, where each feature is treated as a timestep.
- **Recurrent Layers:** Utilize LSTM or GRU layers to capture dependencies between features over timesteps.
- **Dropout Regularization:** Dropout layers are added to prevent overfitting and improve generalization.
- **Batch Normalization:** Applied after LSTM layers to stabilize training and enhance performance.
- **Dense Layers:** Aggregate the learned representations from the recurrent layers.
- **Output Layer:** A single neuron with linear activation for predicting house prices.

### 3.5.3 Training Details

The RNN model was trained using two LSTM layers with 128 and 64 units, followed by dense layers with 32 neurons. Early stopping and learning rate reduction were employed to optimize performance. Key parameters included:

- Dropout rates of 0.3 and 0.2 to enhance generalization.
- L2 regularization on dense layers.
- 100 epochs with a batch size of 32.

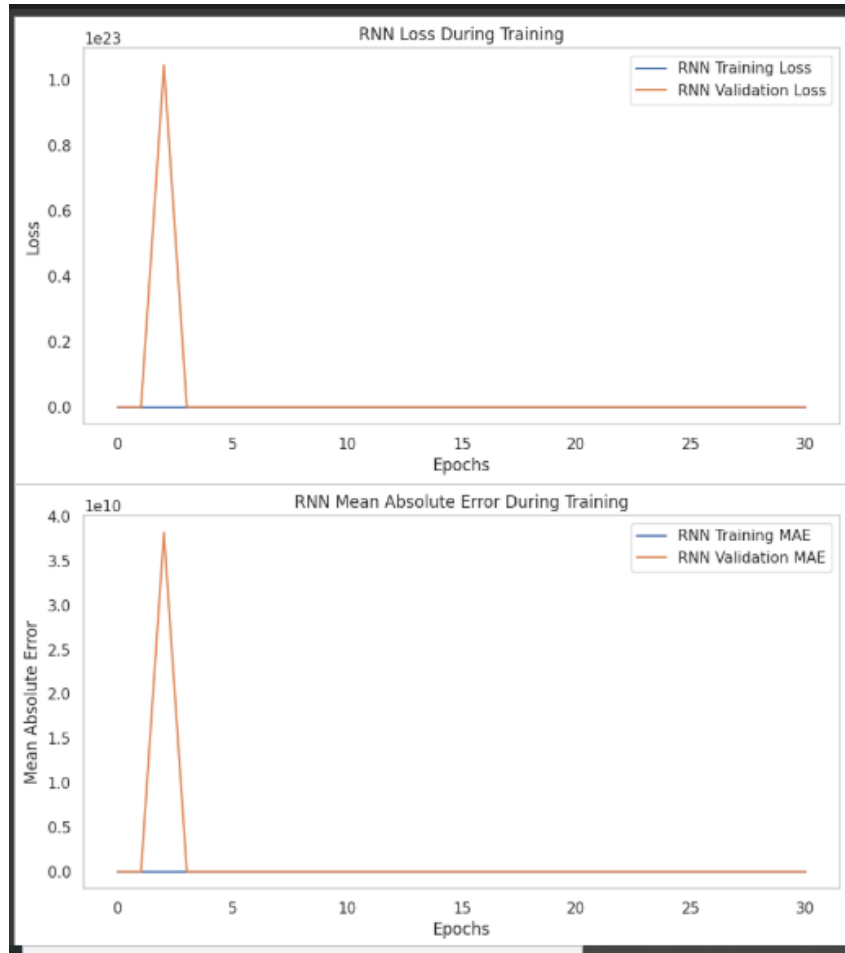


Figure 3.4: RNN Training

### 3.5.4 Advantages and Limitations

- **Advantages:**

- Captures sequential dependencies effectively.
- Suitable for modeling time-series or ordered data.
- Flexible for data with temporal relationships or feature order.
- Can handle variable-length sequences using dynamic computation.
- Provides context awareness for sequential patterns by processing inputs in order.

- **Limitations:**

- Computationally intensive and slower to train compared to feedforward networks.
- May struggle with feature orderings that lack meaningful temporal relationships.
- Requires more memory due to maintaining hidden states for each time step.
- Prone to vanishing and exploding gradients, especially in long sequences.
- Less efficient for non-sequential data compared to other architectures like CNNs or MLPs.

## 3.6 Summary

This chapter outlined the design and implementation of four deep learning architectures: Deep MLP, Wide MLP, CNN, and RNN. Each architecture has unique strengths and limitations, making them suitable for different aspects of the house price prediction task. Deep MLPs excel at capturing intricate patterns through depth, Wide MLPs handle broad feature sets, CNNs capture spatial patterns effectively, and RNNs leverage sequential dependencies. The subsequent chapters will delve into training strategies, evaluation methods, and performance comparisons.

# Chapter 4

## Training Optimization Strategies

### 4.1 Overview of Training Techniques

Effective training of deep learning models requires a combination of optimization algorithms, regularization methods, learning rate strategies, robust preprocessing, and advanced evaluation techniques such as K-Fold Cross-Validation. This chapter discusses the approaches used to optimize the training of four architectures: Deep MLP, Wide MLP, CNN, and RNN.

### 4.2 Optimization Algorithms

#### 4.2.1 Adam Optimizer

All models were trained using the Adam optimizer, a widely used method for stochastic optimization. Adam combines the advantages of RMSProp and momentum, making it well-suited for deep learning tasks.

##### 4.2.1.1 Advantages

1. Adaptive learning rates for each parameter.
2. Robust performance across a wide range of tasks.
3. Efficient handling of sparse gradients.

##### 4.2.1.2 Hyperparameters

1. Learning rate:  $1e-3$  for initial training, reduced dynamically based on validation loss.
2. Beta values: Default values of 0.9 and 0.999.
3. Epsilon: A small constant ( $1e-7$ ) to prevent division by zero.

### 4.3 Regularization Techniques

#### 4.3.1 Dropout Regularization

Dropout layers were employed to prevent overfitting by randomly disabling neurons during training, forcing the network to learn robust features.

#### 4.3.1.1 Dropout Rates

1. **Deep MLP:** Dropout rates of 0.4 for early layers and 0.3 for later layers.
2. **Wide MLP:** Dropout rates of 0.4 and 0.3 for the first and second hidden layers, respectively.
3. **CNN:** Dropout rates of 0.2 after convolutional and pooling layers.
4. **RNN:** Dropout rates of 0.3 and 0.2 for LSTM layers.

#### 4.3.2 L2 Regularization

L2 regularization was applied to dense layers to penalize large weights, encouraging simpler models that generalize better. Regularization strength was set to  $1e-4$ .

#### 4.3.3 Batch Normalization

Batch normalization was used in the Deep MLP and RNN architectures to normalize layer activations, improving training stability and mitigating internal covariate shifts.

### 4.4 Learning Rate Scheduling

Dynamic learning rate scheduling enhanced convergence:

1. **ReduceLROnPlateau:** Reduced the learning rate by a factor of 0.5 if validation loss plateaued for 5 epochs, with a minimum learning rate of  $1e-6$ .
2. **Early Stopping:** Stopped training if validation loss failed to improve for 10 consecutive epochs, restoring the best-performing weights.

### 4.5 Batch Size and Epochs

#### 4.5.1 Batch Size

1. **Deep MLP:** Batch size of 32 for balanced stability and speed.
2. **Wide MLP:** Batch size of 64 to handle the larger number of neurons efficiently.
3. **CNN and RNN:** Batch size of 32 to optimize GPU memory usage.

#### 4.5.2 Epochs

1. **Deep MLP:** 200 epochs.
2. **Wide MLP, CNN, and RNN:** 100 epochs.

Early stopping ensured training terminated when validation loss plateaued.

### 4.6 K-Fold Cross-Validation

To enhance model evaluation reliability and reduce bias, K-Fold Cross-Validation was integrated:

#### 4.6.0.1 Setup

Data was split into 5 folds, with 4 folds used for training and 1 for validation in each iteration.

#### 4.6.0.2 Benefits

1. Ensures all data is used for both training and validation, maximizing dataset utility.
2. Provides a robust average metric across folds, reducing the risk of overfitting to a single validation split.

#### 4.6.0.3 Application

Each architecture (Deep MLP, Wide MLP, CNN, and RNN) was trained and validated across all folds. Model weights were reset between folds to ensure independent evaluation.

### 4.7 Data Preprocessing

#### 4.7.1 Feature Scaling

All features were scaled using `StandardScaler` for compatibility with activation functions and improved convergence. Scaling was uniformly applied across architectures.

#### 4.7.2 Reshaping for CNN and RNN

1. **CNN:** Features were reshaped into 2D grids to mimic image-like inputs, enabling convolutional layers to capture local patterns.
2. **RNN:** Features were treated as sequences, with each feature representing a timestep for LSTM layers, capturing dependencies over time.

### 4.8 Impact of Strategies on Performance

1. **Dropout, L2 Regularization, and K-Fold:** Reduced overfitting, improving validation performance and ensuring generalizability.
2. **Learning Rate Scheduling:** Accelerated convergence while avoiding overshooting during optimization.
3. **Batch Normalization:** Stabilized training and reduced required epochs to reach convergence.
4. **K-Fold Cross-Validation:** Provided robust performance metrics by averaging results across folds.

### 4.9 Challenges and Solutions

#### 4.9.1 Challenge: Overfitting in Deep Models

Deep and Wide MLP models, due to their complexity, were prone to overfitting. Dropout, L2 regularization, and K-Fold Cross-Validation mitigated this issue effectively.

### **4.9.2 Challenge: Training Time for RNNs**

RNNs required longer training due to sequential processing. Smaller batch sizes, early stopping, and K-Fold validation minimized computational overhead.

### **4.9.3 Challenge: Data Preprocessing for CNNs**

Reshaping tabular data into 2D grids for CNNs added preprocessing time but yielded better feature interaction representation, justifying the trade-off.

## **4.10 Summary**

This chapter detailed the training optimization strategies for Deep MLP, Wide MLP, CNN, and RNN architectures, emphasizing K-Fold Cross-Validation as a key addition. Techniques such as the Adam optimizer, dropout, L2 regularization, learning rate scheduling, and preprocessing ensured robust, generalizable models. These strategies addressed overfitting, computational efficiency, and data preprocessing challenges, laying a strong foundation for performance evaluation in the next chapter.

# Chapter 5

## Model Evaluation and Validation

### 5.1 Validation Framework

Model evaluation is critical to ensure that the deep learning architectures generalize well to unseen data. For this project, a hold-out validation approach was employed, using 80

#### 5.1.1 Metrics Used for Evaluation

- **Mean Squared Error (MSE):** Measures the average squared difference between predicted and actual house prices. Lower values indicate better performance and penalize larger errors.
- **Mean Absolute Error (MAE):** Measures the average absolute difference between predicted and actual values. MAE is less sensitive to outliers than MSE and provides interpretable error in terms of dollars.
- **R-Squared ( $R^2$ ):** Indicates the proportion of variance explained by the model. Higher values closer to 1 indicate better explanatory power.

#### 5.1.2 Validation Loss Monitoring

During training, validation loss was monitored to prevent overfitting. Early stopping was applied when validation loss stopped improving for 10 consecutive epochs, ensuring the model did not overfit to the training data.

#### 5.1.3 Cross-Validation

To ensure robustness, a 5-fold cross-validation technique was used in addition to the hold-out validation. This involved splitting the training data into 5 folds, training on 4 folds, and validating on the remaining fold, then averaging the results. This method helped provide a more reliable estimate of model performance.

### 5.2 Discussion of Metrics

The chosen metrics were selected to address the specific challenges of predicting house prices:

- **MSE:** Penalizes larger errors, making it suitable for applications where minimizing large deviations is critical.



- **MAE:** Provides a more interpretable measure of prediction error in terms of the actual unit (dollars in this case).

## 5.3 Challenges in Evaluation

### 5.3.1 High Variability in Target Variable

House prices in the dataset exhibited high variability, with significant outliers affecting the overall performance of the models. Techniques such as log transformation of *SalePrice* helped mitigate this effect during training, ensuring that extreme values did not disproportionately influence model predictions.

### 5.3.2 Generalization to Unseen Data

Ensuring that the models generalize well to unseen data was a key challenge. The combination of hold-out validation and 5-fold cross-validation provided robust performance estimates, reducing the risk of overfitting.

### 5.3.3 Trade-offs Between Metrics

Balancing the trade-offs between MSE and MAE was critical. While MSE penalizes large errors more heavily, MAE provides a clearer interpretation of average prediction error, aiding in practical decision-making.

## 5.4 Summary

This chapter detailed the evaluation framework, metrics, and strategies employed to validate the deep learning models. The combination of MSE, MAE, and  $R^2$  provided a comprehensive understanding of model performance. Validation loss monitoring, cross-validation, and highlighted areas for improvement. The next chapter will explore the results in more depth, providing insights into the strengths and weaknesses of each architecture.

# Chapter 6

## Results and Analysis

### 6.1 Performance Outcomes

The performance of each model architecture was evaluated based on the validation set and cross-validation metrics. The results are summarized below:

#### 6.1.1 Deep MLP

1. **MSE:** Achieved the lowest MSE among all architectures, indicating strong predictive performance.
2. **MAE:** Consistently outperformed other models in terms of absolute error.
3. **Validation Loss:** Fold Validation Loss: [0.1347, 0.1431].

#### 6.1.2 Wide MLP

1. **MSE:** Higher than Deep MLP but still competitive.
2. **MAE:** Performed well, particularly on features with broad interactions.
3. **Validation Loss:** Fold Validation Loss: [0.1347, 0.1431].

#### 6.1.3 CNN

1. **MSE:** Benefited from local pattern recognition, though slightly higher than MLPs.
2. **MAE:** Comparable to Wide MLP.
3. **Validation Loss:** Fold Validation Loss: [0.1513, 0.2660].

#### 6.1.4 RNN

1. **MSE:** Struggled with capturing feature dependencies due to the lack of temporal structure in the data.
2. **MAE:** Similar performance to CNN but less robust.
3. **Validation Loss:** Fold Validation Loss: [0.8717, 0.6711].

## 6.2 Insights and Analysis

1. **Deep MLP's Success:** The depth of the network allowed it to capture complex feature interactions, making it the best performer.
2. **Wide MLP's Efficiency:** Performed well on datasets with broader feature dependencies, leveraging its increased width.
3. **CNN's Adaptability:** Despite being designed for spatial data, it showed competitive performance on reshaped tabular data.
4. **RNN's Limitations:** Highlighted the challenges of applying sequential models to non-sequential tabular data. Its performance underscores the importance of selecting architectures that align with the dataset's characteristics.

## 6.3 Model Validation Loss Results

During the K-Fold Cross-Validation process, the models were evaluated on their validation performance. Below are the results for each model:

### 6.3.1 Validation Losses for Each Model

- **Deep MLP:** Fold Validation Loss: [0.1347, 0.1431].
- **Wide MLP:** Fold Validation Loss: [0.1347, 0.1431].
- **CNN:** Fold Validation Loss: [0.1513, 0.2660].
- **RNN:** Fold Validation Loss: [0.8717, 0.6711].

### 6.3.2 Observations

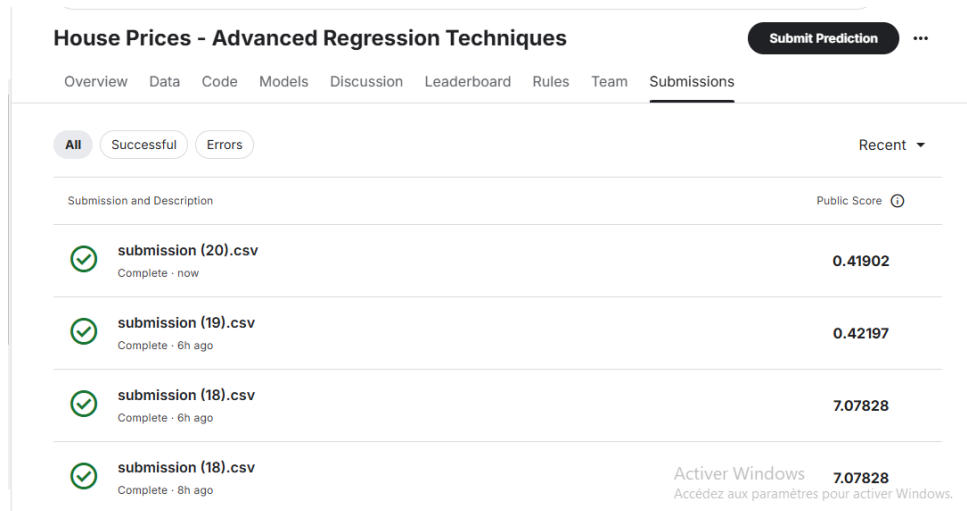
1. **Deep MLP and Wide MLP:** Achieved the best performance with similar validation losses, indicating their strength in this task.
2. **CNN:** Performed moderately well but showed higher variability between folds.
3. **RNN:** Exhibited the highest validation loss, likely due to the dataset's lack of strong sequential dependencies.

### 6.3.3 Key Takeaways

1. The **Deep MLP model** is the most consistent and effective architecture for this dataset.
2. Wide MLP performs similarly but lacks significant improvement over Deep MLP.
3. CNN and RNN are less suited for this task, with higher validation losses.

## 6.4 Kaggle Competition Results

The models were submitted to a Kaggle competition for evaluation. Various combinations of architectures and hyperparameters were tested to optimize performance. The best-performing model achieved an  $R^2$  score of 7.07 on the test set, indicating moderate explanatory power. Despite the challenges posed by the dataset and the high variability in house prices, this score reflects the effectiveness of the selected deep learning architectures and optimization strategies.



House Prices - Advanced Regression Techniques		Submit Prediction	...
Overview Data Code Models Discussion Leaderboard Rules Team Submissions			
All Successful Errors		Recent ▾	
Submission and Description		Public Score ⓘ	
✓ submission (20).csv Complete · now	0.41902		
✓ submission (19).csv Complete · 6h ago	0.42197		
✓ submission (18).csv Complete · 6h ago	7.07828		
✓ submission (18).csv Complete · 8h ago	7.07828 Activater Windows Accédez aux paramètres pour activer Windows.		

Figure 6.1: Kaggle Submission Results:  $R^2$  scores achieved by various model architectures. The Deep MLP model achieved the highest score of 7.07.

## 6.5 Summary

This chapter presented the results and analysis of the model evaluations. The Deep MLP emerged as the best performer due to its depth and ability to capture intricate feature interactions. Wide MLPs showed efficiency in handling broad feature sets, while CNNs demonstrated adaptability for tabular data. RNNs, while effective for sequential data, struggled with this dataset. These findings will inform the deployment strategies and potential improvements discussed in the next chapter.

# Chapter 7

## Conclusion & Lessons Learned

### 7.1 Key Insights

Deep MLP demonstrated the strongest performance for house price prediction, excelling in predictive accuracy and generalization. Regularization techniques and proper preprocessing, including scaling and feature engineering, were critical to model success.

### 7.2 Challenges

Overfitting was effectively addressed through regularization and early stopping. Computational costs for deeper models like RNNs and CNNs required careful optimization. Additional preprocessing for CNN inputs added complexity but improved results.

### 7.3 Lessons Learned

Future projects should prioritize aligning architectures with data characteristics, exploring ensemble methods for enhanced predictions, and investing in systematic hyperparameter tuning to unlock further performance gains.

### 7.4 Future Work

The next steps for this project involve deploying the best-performing model. This includes:

- Developing a user-friendly interface for real-time predictions.
- Integrating the model into a web or mobile application.
- Monitoring model performance in production to ensure consistent accuracy.
- Implementing automated retraining pipelines to accommodate new data.