MAKING MONEY WITH MACHINE LEARNING?

Edmundo Arias De Abreu
Economics Department
Universidad de los Andes

Lucía Maldonado
Economics Department
Universidad de los Andes

Juan Diego Heredia
Economics Department
Universidad de los Andes

December 2, 2024

1 Introduction

Real estate has been one of the most profitable economic sectors globally. In the United States, the Gross Domestic Product (GDP) generated by real estate has more than doubled over the last 20 years (of St. Louis (2024)).[1] In China, real estate is considered the most important sector, contributing 23.6% of the annual GDP (Li et al., 2024). Similar trends can be observed in countries with economic patterns comparable to Colombia. For instance, in Mexico, real estate accounts for approximately 5% of the GDP, while this figure rises to nearly 15% in Ecuador and 10% in Paraguay (de Bancos (2022)). In Colombia, the real estate sector contributes 4.3% to the GDP (Guzmán, 2024).

Despite its importance, the transition of real estate from the sale of newly constructed properties to the resale of pre-owned properties has posed challenges for property valuation in the real estate market. This scarcity of reliable information creates market frictions that hinder not only the potential expansion of trade but also household wealth on a global scale. In Spain, for instance, between 1997–2006 and 1998–2005, the inability of households to predict actual property prices fueled a housing bubble. This bubble artificially increased housing prices by 117% above their market value, adversely impacting the wealth of investors in this sector and destabilizing the country's macroeconomic foundation (Bertolín Mora, 2014). The absence of trustworthy price prediction methods also affects the profitability of real estate companies. For example, Sandra Ortega's real estate agency, the largest in Spain, reported losses of 43.3 million euros last year due to housing market price volatility and its inability to predict prices accurately.

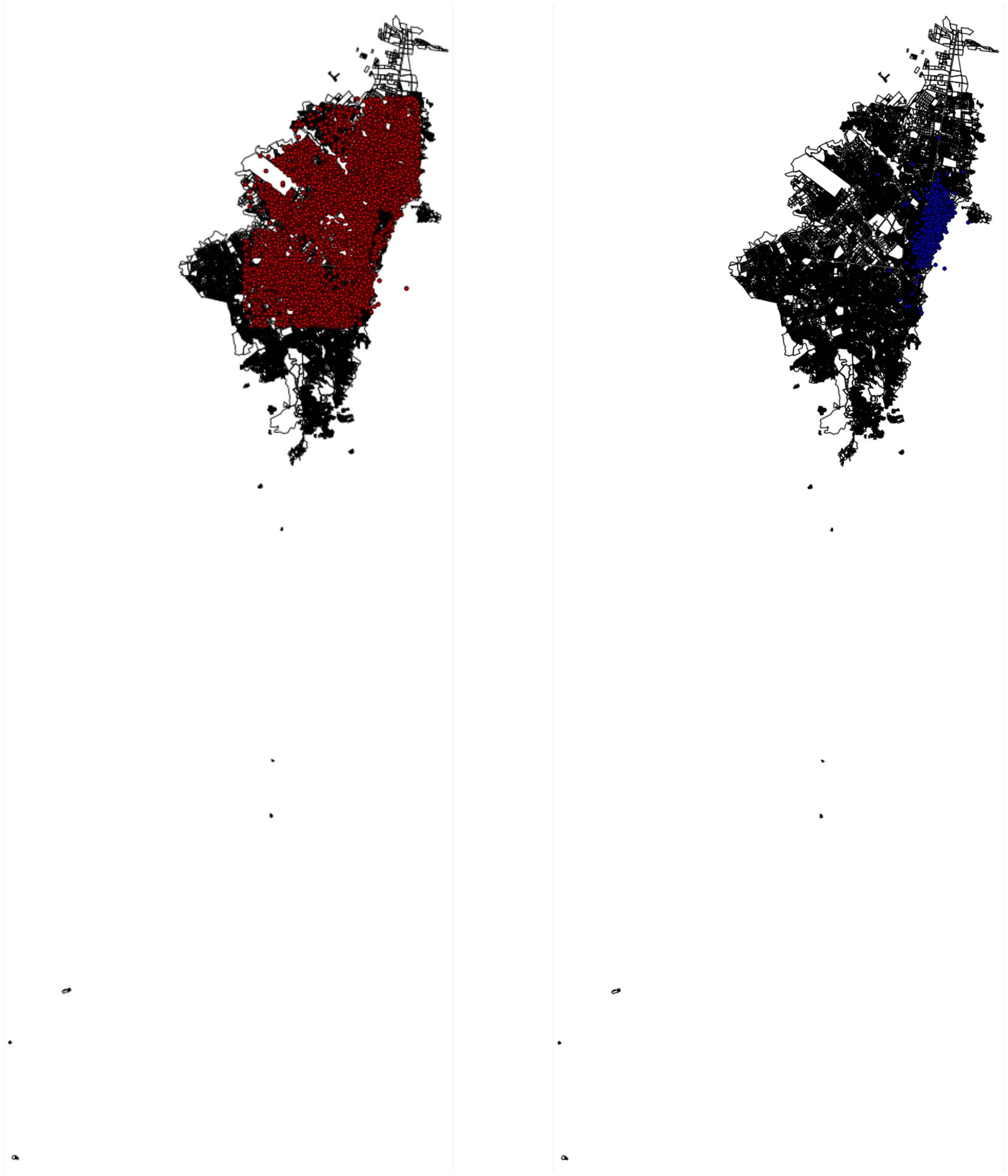
This document presents a predictive model for asking prices based on product differentiation features of housing. The model is trained and tested on a dataset of housing lots located in Chapinero, Bogotá. This study contributes to the extensive body of literature employing machine learning models to predict housing prices, including methods such as XGBoost (Sharma et al., 2024), random forests (Xu & Nguyen, 2022), Elastic Net, and bagging (Mora-Garcia et al., 2022). The document is organized as follows: Section 1 introduces the motivation for the study; Section 2 describes the dataset and the data cleaning process undertaken prior to model training; Section 3 discusses the best-performing prediction model; and Section 4 offers conclusions and recommendations.

2 Data

We worked with two datasets containing information about households of interest. The training dataset includes 38,644 households for sale in Bogotá, of which 75.50% are apartments and 24.50% are houses. The test dataset consists of 10,286 households, with 97.33% being apartments and only 2.67% houses. Figure 1 shows the distribution of households in the training and test datasets across Bogotá. While the training data covers households located throughout the city, the test set is concentrated exclusively in the eastern area, specifically in Chapinero.

[1]The U.S. real estate GDP increased from US$1,308,532 million in 2003 to US$3,447,118 million in 2023.

Figure 1: Centroids of Households of Interest and Bogotá's *Manzanas*



The figure shows the coordinates of the centroids for all the *households of interest* in the training and test sets included in the analysis. The grey boundaries represent all *manzanas* in the capital district of Bogotá, Colombia. The polygons were obtained from the National Administrative Department of Statistics of Colombia (DANE) through the *Marco Geoestadístico Nacional*, version 2023.

Given that variables describing household attributes are crucial for predicting their sale price, we utilized the descriptions of households on the website to reconstruct a series of dummy variables capturing household features. A semi-supervised matching algorithm of the Jaro-Winkler type (1990) was employed to identify the presence of amenities such as laundry facilities, clubhouse, furniture, parking, etc. Jaro (1989) proposed a system for determining the degree of similarity between text strings based on comparisons of their length, letter-by-letter matching, and the order of their characters. Accordingly, if S_1 and S_2 are two text strings, the Jaro (1989) similarity coefficient (Φ) maps the pair (S_1, S_2) onto the closed interval $[0,1]$, where one implies that the two text strings are identical, and zero indicates the lowest level of similarity. Mathematically, the Jaro (1989) similarity index is defined as:

$$\Phi = W_1 \left(\frac{c}{d}\right) \times W_2 \left(\frac{c}{r}\right) \times W_t \left(\frac{c-\tau}{d}\right)$$

where[2],W_i, for $i \in \{1,2\}$, represents the weight associated with the characters in each text string (S_1, S_2). W_t represents the weight associated with transposed characters. d and r are the number of words in text strings S_1 and S_2, respectively. τ is the number of shared letters in the wrong position, divided by 2. c is the number of common letters between (S_1, S_2).

The standard weighting factor $W_i \in 1, 2, t$ is equal to 1/3, such that if two text strings are identical, the Jaro similarity score is one.

We set a threshold of 0.99 to allow for the identification of words reflecting the existence of amenities with minor differences or typographical errors. Table 1 presents descriptive statistics for the amenity variables created in this process for both the training and test sets. It is observed that 23% of the properties in the training set were remodeled before being put up for sale, 39% have a balcony, and 24% report having at least one parking space. In contrast, the most observed attributes in the test set are a balcony (41%), laundry facilities (23%), and parking (22%).

Additionally, we used the command `regmatches()` to identify the numbers immediately following words such as 'estrato', 'baños', 'habitaciones', and variants of these words to determine the socioeconomic status, number of bathrooms, and number of bedrooms in the household. This process allowed us to reduce the number of missing values for these variables in the raw files for both the training and test datasets. On average, households in the training dataset have 2.97 rooms, 3.14 bedrooms, and 2.88 bathrooms, while households in the test dataset have slightly lower values for these amenities, with averages of 2.38 rooms, 2.38 bedrooms, and 2.71 bathrooms. As a final step, we utilized *Open Street Maps* data to calculate the distance from each household to the closest Transmilenio station and to the Zona G (Calle 72 with Carrera 7). These variables are relevant from the perspective of the monocentric model, where price increases with proximity to the CBD, and the Rosen-Roback model, where amenities (such as location) can affect price.

Table 1: Summary of Households and Predictors by Set

Variable	*Training set*				*Test set*			
	Count	Mean	Std. Dev.	P50	Count	Mean	Std. Dev.	P50
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Precio (Millones)	38644	654.53	311.42	559.99		.	.	.
Distancia Transmilenio (mts)	38644	851.56	615.06	675.56	10286	784.85	466.30	673.11
Distancia Zona G (mts)	38644	6455.98	2223.53	6151.77	10286	2021.65	999.89	2015.12
Estrato	2907	4.32	1.12	4.00	588	5.05	1.20	5.00
Area	13162	1474.96	4453.05	115.00	3451	1430.32	4395.80	124.00
Ascensor	38644	0.15	0.36	0.00	10286	0.17	0.38	0.00
Remodelado	38644	0.08	0.27	0.00	10286	0.14	0.34	0.00
Lavanderia	38644	0.23	0.42	0.00	10286	0.23	0.42	0.00
Club House	38644	0.06	0.23	0.00	10286	0.03	0.17	0.00
Patio	38644	0.11	0.31	0.00	10286	0.03	0.17	0.00
Balcón	38644	0.39	0.49	0.00	10286	0.41	0.49	0.00
Amoblado	38644	0.01	0.09	0.00	10286	0.02	0.15	0.00
Parqueadero	38644	0.24	0.43	0.00	10286	0.22	0.41	0.00
Baños (Número)	28573	2.88	1.09	3.00	7795	2.71	0.96	3.00
Habitaciones (Número)	38644	3.14	1.53	3.00	10286	2.38	0.96	2.00
Piezas (Número)	22153	2.97	1.35	3.00	6087	2.38	0.93	2.00

The table presents the count of households with available information for all the predictors of interest. The data is at the household level. Treatment is defined as the earliest year in which at least one plot was certified by the SQP. Columns (2)-(5) consider only households in the training set, while columns (6)-(9) focus on households in the test set. Variables without a defined unit of measurement correspond to dichotomous variables set to one if the household has a given amenity.

[2]The similarity percentage is calculated if the text strings do not differ in the number of words by more than $\left(\frac{m}{2} - 1\right)$ with $m = \max\{d, r\}$.

3 Model

3.1 Pre-set up for Neural Network Models

Before implementing the neural network models for housing price prediction, a robust pre-set up process was established. This phase included data preprocessing, the definition of a custom dataset class, and the creation of a modular model architecture and training framework. These steps ensured consistency across all models and facilitated a systematic evaluation of their performance.

The preprocessing function was designed to transform raw data into a format suitable for training. First, relevant numerical features were selected, excluding irrelevant columns such as the property identifier and the target variable. These features were normalized using `StandardScaler` to ensure homogeneous distributions and efficient convergence during training. The target variable, representing property prices, was log-transformed to stabilize variance and address its skewed distribution. Subsequently, the dataset was split into training and validation subsets in an 80/20 ratio to ensure reliable evaluation of the models' generalization capacity. To integrate these subsets into PyTorch's workflow, a custom dataset class, `PropertyDataset`, was implemented. This class converted the feature and target arrays into PyTorch-compatible tensors and provided methods for efficient batch retrieval during training.

The architecture of the neural network models was defined through the `PricePredictor` class, which implemented a flexible and modular structure. This network consisted of an input layer, hidden layers, and an output layer specifically designed for regression tasks. The hidden layers employed activation functions, batch normalization, and dropout regularization to improve stability and prevent overfitting. The modular design allowed iterative modifications to optimize the network's performance across experiments.

To optimize the models' parameters and monitor their performance, a dedicated training function was developed. This function utilized PyTorch's training and evaluation modes to calculate gradients during training while disabling them during validation to reduce computational overhead. The loss function selected was the Mean Absolute Error (MAE), which quantified the discrepancy between predicted and actual values. The Adam optimizer was employed to iteratively adjust the models' weights. Additionally, the training function monitored training and validation losses, storing the best-performing model based on validation performance for subsequent evaluation.

This pre-set up phase provided a consistent foundation for evaluating multiple models iteratively. It ensured uniformity in data handling, model architecture, and training procedures, facilitating fair comparisons and reliable results.

3.2 Neural Network Models

3.2.1 Model 1: Baseline Neural Network

The first model implemented was a baseline neural network designed to establish a benchmark for predicting housing prices. The architecture of the model consisted of a fully connected feedforward neural network, characterized by a straightforward yet effective structure. The input layer had a number of neurons equal to the total number of features in the dataset, ensuring all available information was utilized. The network included three hidden layers with 128, 64, and 32 neurons, respectively. Each hidden layer employed the ReLU activation function to introduce non-linearity, followed by batch normalization to stabilize training and dropout regularization (0.3 for the first layer and 0.2 for the second layer) to mitigate overfitting. The final output layer comprised a single neuron designed for regression, predicting the log-transformed property prices.

The training configuration employed for this model was designed to ensure robustness and stability. The loss function was defined as the Mean Absolute Error (MAE), chosen for its resilience to outliers and its interpretability in regression tasks. The Adam optimizer was selected with a learning rate of 0.001, leveraging its adaptive learning capabilities to improve convergence. The model was trained using a batch size of 64 over 50 epochs. The training process included validation at each epoch, allowing the model's generalization capacity to be monitored. Computations were executed on a GPU where available, significantly enhancing the efficiency of the training process. Although early stopping was not applied at this stage, the validation loss provided insight into the model's performance across iterations.

The results of the training process are depicted in the loss curves, which demonstrate the evolution of both training and validation losses over the 50 epochs. These curves reveal that the model converged rapidly, with losses stabilizing within the first 10 epochs. The close alignment between training and validation losses indicates that the baseline model generalized well to unseen data, with minimal signs of overfitting.

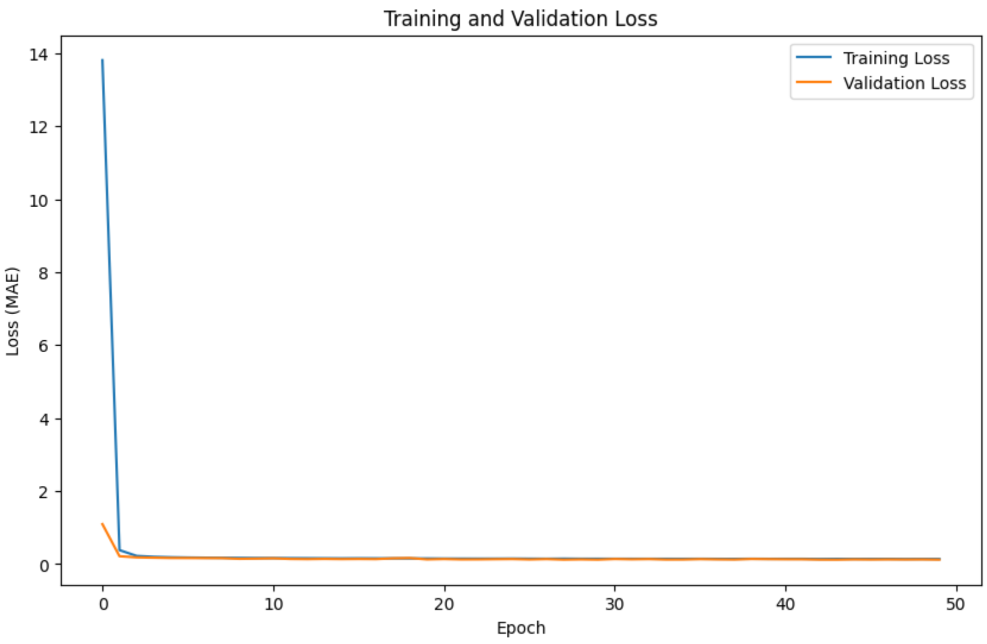


Figure 2: Training and Validation Loss for Model 1: Baseline Neural Network.

3.2.2 Model 2: Enhanced Neural Network with Increased Neurons

In the second model iteration, adjustments were made to the baseline architecture to enhance its capacity for capturing complex relationships within the data. Specifically, the first hidden layer's size was increased from 128 to 256 neurons. This modification aimed to provide the model with a greater representational capacity, enabling it to process a broader range of feature interactions and extract more nuanced patterns. The remainder of the architecture remained consistent with the baseline model, maintaining three hidden layers and the same activation functions, batch normalization, and dropout configurations.

The revised architecture began with an input layer matching the dimensionality of the dataset's features. The first hidden layer, now expanded to 256 neurons, was followed by a ReLU activation function, batch normalization, and a dropout rate of 0.3. The second hidden layer retained its configuration of 64 neurons, ReLU activation, batch normalization, and a dropout rate of 0.2. Similarly, the third hidden layer included 32 neurons with batch normalization and a ReLU activation function. The final output layer comprised a single neuron designed for regression tasks, predicting the log-transformed property prices.

The training process for this model employed the same configuration as the baseline model, with a Mean Absolute Error (MAE) loss function and the Adam optimizer using a learning rate of 0.001. The model was trained over 50 epochs with a batch size of 64, and validation loss was monitored throughout the training process to evaluate the generalization of the model.

The training and validation loss curves for this enhanced model are presented in the figure below. Compared to the baseline model, the increased capacity in the first hidden layer resulted in slightly faster convergence during the initial epochs. Both training and validation losses stabilized within 10 epochs, demonstrating consistent performance and effective learning. The close alignment between training and validation losses highlights the model's ability to generalize well to unseen data without significant overfitting.

3.2.3 Model 3: Neural Network with Leaky ReLU Activation

In the third model iteration, a significant change was introduced by replacing the ReLU activation function with Leaky ReLU across all layers. This modification addressed the "dying neuron" problem commonly encountered with standard ReLU, where neurons can become inactive during training and fail to learn. By allowing a small, non-zero gradient (defined by a slope of 0.1 in this implementation) for negative inputs, Leaky ReLU improved the flow of gradients through the network, particularly for sparse or complex datasets. This change was expected to enhance the model's learning dynamics and stability during training.

The architecture of the model retained the overall structure from the baseline, consisting of three hidden layers with 128, 64, and 32 neurons, respectively. Each hidden layer incorporated Leaky ReLU as the activation function, followed

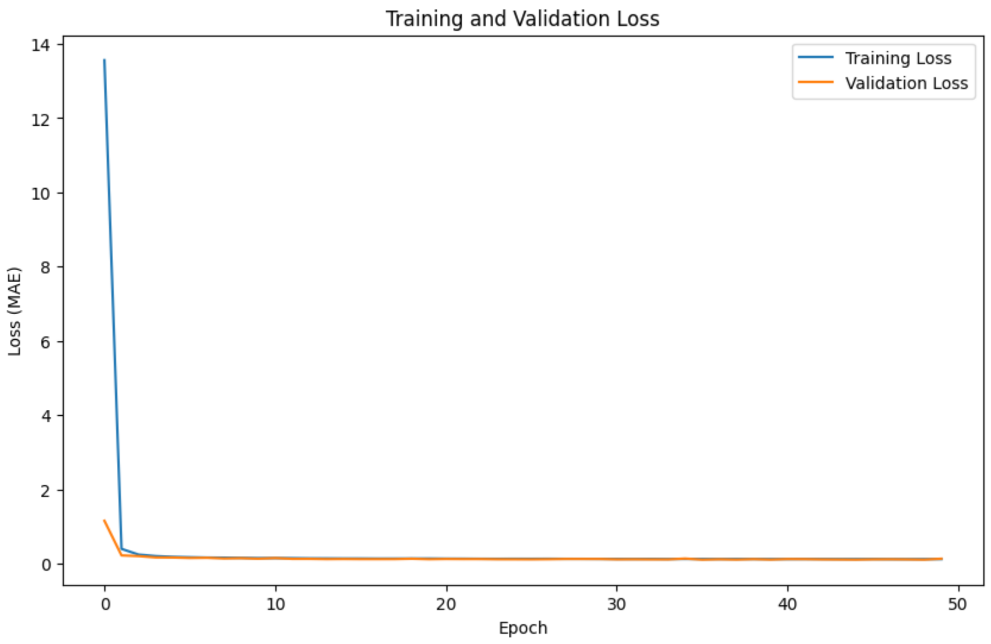


Figure 3: Training and Validation Loss for Model 2: Enhanced Neural Network with Increased Neurons.

by batch normalization to stabilize intermediate outputs and dropout to mitigate overfitting. Dropout rates of 0.3 and 0.2 were applied in the first and second layers, respectively, while the third layer included batch normalization without dropout. The output layer remained unchanged, containing a single neuron designed for regression tasks and predicting the log-transformed property prices.

The training configuration remained consistent with the prior models, using the Mean Absolute Error (MAE) as the loss function and the Adam optimizer with a learning rate of 0.001. The model was trained over 50 epochs with a batch size of 64. This setup ensured a fair comparison with previous models while focusing on evaluating the impact of the new activation function.

The training and validation loss curves for this model demonstrated improved stability compared to earlier iterations. The Leaky ReLU activation facilitated better gradient propagation, which was particularly evident in the smooth convergence of the loss curves. The training loss decreased steadily during the early epochs, while the validation loss remained closely aligned, indicating that the model generalized effectively to unseen data without significant overfitting.

The inclusion of Leaky ReLU across all layers proved to be a meaningful enhancement, enabling the network to learn more efficiently and reducing the risk of neurons becoming inactive. This design iteration underscores the importance of activation function choice in optimizing neural network performance, particularly for complex regression tasks such as housing price prediction.

3.2.4 Model 4: Neural Network with Residual Connections

The fourth iteration of the neural network introduced a significant architectural innovation by incorporating residual connections. Residual connections, initially proposed in ResNet architectures, aim to address the vanishing gradient problem, particularly in deeper networks. By allowing the input of a layer to bypass transformations and be added back to the output, residual connections enable better gradient flow and facilitate the learning of identity mappings, improving model convergence and stability.

The architecture began with an input projection layer that reduced the dimensionality of the input features to 128 neurons, preparing the data for the subsequent layers. The core of the network included three main layers: The first layer, consisting of 128 neurons, used a Leaky ReLU activation function with a slope of 0.1, followed by batch normalization to stabilize the activations and a dropout rate of 0.3 to mitigate overfitting. This layer included a residual connection, where the input to the layer was directly added back to its output. The second layer, with 64 neurons, also employed Leaky ReLU activation, batch normalization, and a dropout rate of 0.2. Finally, the third layer consisted of 32 neurons with Leaky ReLU activation and batch normalization. The network concluded with an output layer containing a single neuron to predict the log-transformed housing prices. This sequential yet interconnected design allowed the network to efficiently capture complex feature interactions while benefiting from the stability provided by the residual connections.

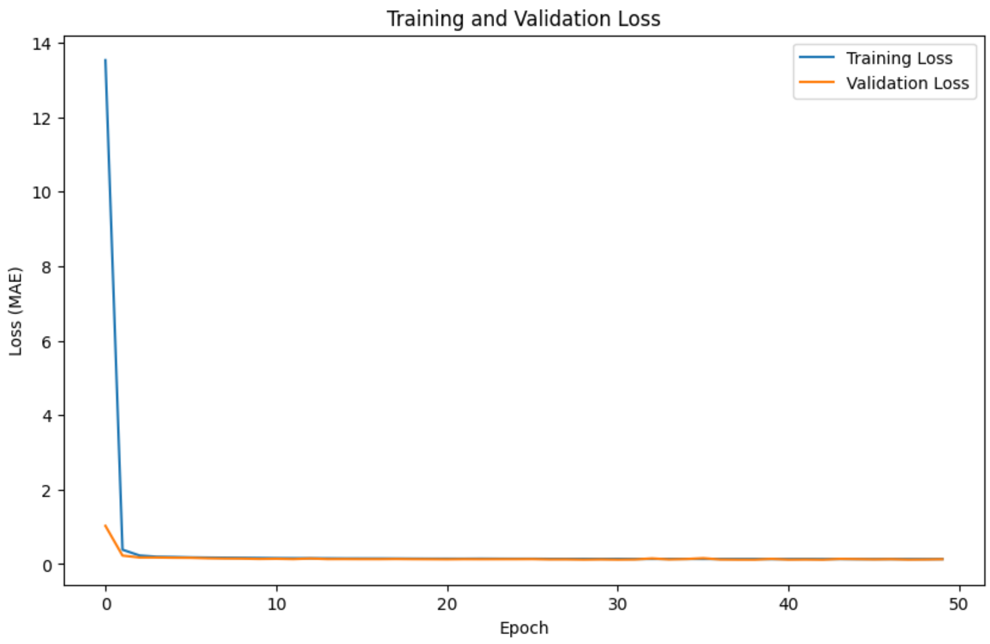


Figure 4: Training and Validation Loss for Model 3: Neural Network with Leaky ReLU Activation.

The training configuration remained consistent with prior models, using the Mean Absolute Error (MAE) loss function and the Adam optimizer with a learning rate of 0.001. The model was trained over 50 epochs with a batch size of 64. The inclusion of residual connections facilitated faster convergence and improved gradient propagation, as evident in the loss curves. These connections also enabled the network to learn deeper representations without the risk of degradation, a common issue in deeper architectures without such enhancements.

The results demonstrated improved stability in training, with validation losses slightly lower than those of the previous models. The model's ability to generalize was maintained, as indicated by the minimal gap between training and validation loss curves. This design marked a pivotal advancement in the iterative development process, showcasing the effectiveness of integrating modern deep learning strategies into regression models.

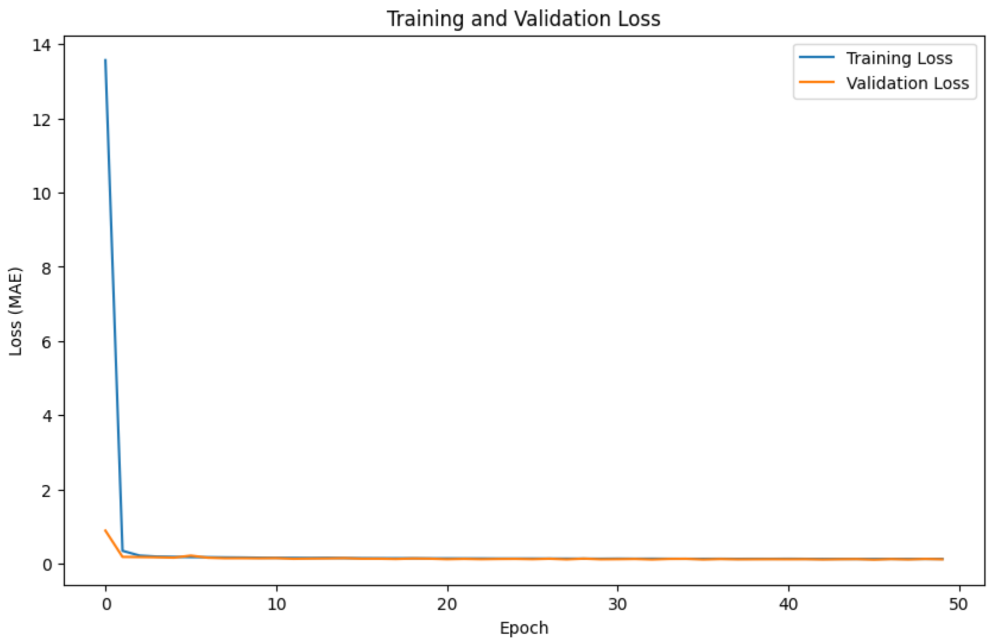


Figure 5: Training and Validation Loss for Model 4: Neural Network with Residual Connections.

3.2.5 Model 5: Neural Network with Learning Rate Scheduler

In the fifth iteration of the neural network, a significant enhancement was introduced to optimize the training process: the integration of a learning rate scheduler. This model reused the architecture from Model 3, retaining its three hidden layers with 128, 64, and 32 neurons, respectively. The hidden layers employed the Leaky ReLU activation function (with a slope of 0.1) to improve gradient flow and mitigate the "dying neuron" problem. Batch normalization was applied after each hidden layer to stabilize intermediate outputs, and dropout regularization was included in the first two layers, with rates of 0.3 and 0.2, respectively. The output layer consisted of a single neuron for predicting the log-transformed property prices.

The primary innovation in this model was the addition of a learning rate scheduler to dynamically adjust the learning rate during training. Specifically, the `ReduceLROnPlateau` scheduler was employed to reduce the learning rate by a factor of 0.5 whenever the validation loss plateaued for five consecutive epochs. This approach allowed the optimizer to make larger updates during the initial phases of training and finer adjustments as convergence was approached, effectively improving model performance. The minimum learning rate was set to 1×10^{-6}, ensuring that updates remained meaningful even in later stages of training.

The training configuration was consistent with prior iterations, utilizing the Mean Absolute Error (MAE) as the loss function and the Adam optimizer with an initial learning rate of 0.001. The model was trained for 50 epochs with a batch size of 64. The training loop incorporated the scheduler, which adjusted the learning rate based on the validation loss at the end of each epoch. This mechanism ensured that the learning rate was reduced only when necessary, avoiding unnecessary stagnation in training progress.

The results highlighted the effectiveness of integrating the learning rate scheduler. The training and validation loss curves demonstrated smooth convergence, with the validation loss exhibiting a more consistent decline compared to previous models. The scheduler's adjustments facilitated efficient learning by focusing optimization efforts on challenging regions of the loss landscape. Additionally, the use of the scheduler helped avoid overfitting by preventing the optimizer from overadjusting to the training data in later epochs.

The introduction of a learning rate scheduler marked a pivotal advancement in the iterative development process. This enhancement underscored the importance of adaptive optimization strategies in achieving efficient convergence and robust generalization for regression tasks such as housing price prediction.

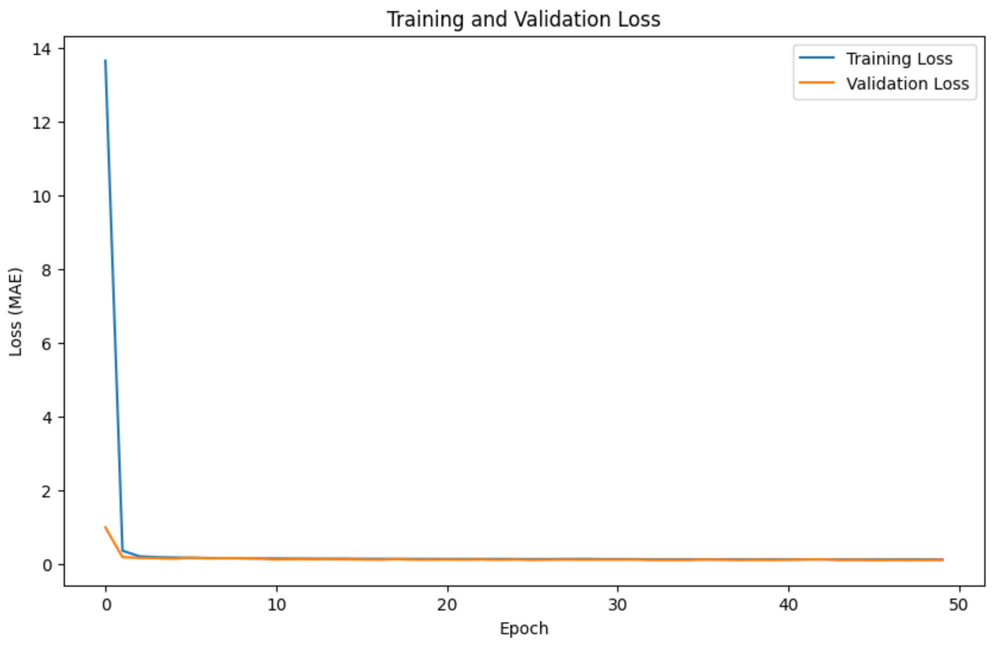


Figure 6: Training and Validation Loss for Model 5: Neural Network with Learning Rate Scheduler.

3.2.6 Model 6: Neural Network with Extended Training Duration

The sixth iteration of the neural network retained the same architecture as Model 3 but focused on the impact of extending the training duration. The model consisted of three hidden layers with 128, 64, and 32 neurons, respectively. Each hidden layer used the Leaky ReLU activation function with a slope of 0.1 to address the vanishing gradient problem and ensure consistent gradient flow. Batch normalization followed each hidden layer to stabilize intermediate outputs, and dropout was applied in the first two layers with rates of 0.3 and 0.2, respectively, to prevent overfitting. The output layer contained a single neuron tasked with predicting the log-transformed property prices.

The primary modification in this iteration was extending the training duration to 200 epochs, allowing the model to explore more of the parameter space and achieve potentially better convergence. The training configuration retained the use of the Mean Absolute Error (MAE) as the loss function and the Adam optimizer with an initial learning rate of 0.001. The batch size was maintained at 64, ensuring consistent training stability across iterations. This extended training setup aimed to evaluate whether the model could capture additional patterns in the data with a prolonged training process and whether this would lead to improvements in generalization.

The longer training duration provided the model with more opportunities to refine its weights and reduce the loss function. However, it also increased the risk of overfitting, making validation loss monitoring crucial throughout the process. By carefully observing the training and validation loss curves, the model's ability to generalize could be assessed.

The results of this iteration revealed that extending the training duration led to stable and gradual improvements in both training and validation loss. The model benefited from additional epochs, showing a consistent reduction in validation loss without significant signs of overfitting. This highlighted the potential advantages of prolonged training for capturing more intricate relationships in the dataset while reinforcing the importance of regularization techniques such as dropout and batch normalization in maintaining generalization.

This model demonstrated that extended training could be an effective strategy for improving performance, provided that overfitting is carefully managed. The extended training duration also emphasized the importance of hyperparameter tuning to balance computational cost with performance gains in regression tasks like housing price prediction.

3.2.7 Model 7: Neural Network with Adjustable Leaky ReLU Slope

The seventh iteration of the neural network introduced an adjustable parameter for the slope of the Leaky ReLU activation function, allowing flexibility in how negative inputs are handled during training. This modification aimed to explore the effect of different slopes on the model's performance, particularly in addressing the vanishing gradient problem while maintaining robust gradient flow.

The architecture remained consistent with prior models, comprising three hidden layers with 128, 64, and 32 neurons, respectively. Each layer utilized Leaky ReLU as the activation function, with the negative slope specified as a parameter during model initialization. Batch normalization was applied after each hidden layer to stabilize the training process, and dropout regularization with rates of 0.3 and 0.2 was included in the first two layers to prevent overfitting. The output layer consisted of a single neuron tasked with predicting the log-transformed property prices.

The primary innovation in this iteration was the introduction of a customizable Leaky ReLU slope. For this model, the slope was set to 0.2, differing from the default value of 0.1 used in previous iterations. This adjustment provided an opportunity to examine whether a steeper slope could enhance the model's ability to capture subtle patterns in the data by allowing larger gradients for negative inputs.

The training configuration remained consistent with previous models, utilizing the Mean Absolute Error (MAE) as the loss function and the Adam optimizer with an initial learning rate of 0.001. The model was trained for 50 epochs with a batch size of 64, and the loss values were monitored across both training and validation datasets.

The results demonstrated that the adjusted Leaky ReLU slope contributed to improved performance. Validation loss showed consistent reductions during training, particularly in the early epochs, reflecting the model's ability to learn effectively from the data. The flexibility in defining the Leaky ReLU slope provided a meaningful enhancement, underscoring the importance of tailoring activation functions to the specific characteristics of the dataset.

This model emphasized the potential benefits of optimizing activation function parameters as part of the iterative model development process. By fine-tuning components such as the Leaky ReLU slope, the network demonstrated improved convergence and generalization, making it a valuable addition to the sequence of experiments in housing price prediction.

3.2.8 Model 8: Parallel Neural Network with Multiple Activation Paths and Early Stopping

In the eighth iteration of the neural network, a novel approach was introduced by incorporating multiple activation paths into the architecture. This design aimed to evaluate the impact of different activation functions—Leaky ReLU, ELU, and GELU—on model performance by processing the input data through parallel paths. Each activation path independently extracted features, which were later combined into a unified output for final prediction. Additionally, early stopping and gradient clipping were implemented to improve training stability and prevent overfitting.

The architecture consisted of three parallel paths. The first path used Leaky ReLU activation with a slope of 0.1, maintaining a consistent configuration with previous models. The second path incorporated ELU activation, which is known for its smooth gradient properties and ability to avoid dead neurons. The third path utilized GELU activation, which combines smoothness and non-linearity, often improving gradient flow in deep networks. Each path consisted of three layers, with 128, 64, and 32 neurons, respectively, followed by batch normalization and dropout with rates of 0.3 and 0.2. The outputs of these paths were concatenated and processed through additional dense layers, ultimately producing a single output neuron for regression.

The training process introduced several advanced techniques. Early stopping was implemented with a patience of 10 epochs, halting training when the validation loss plateaued, and ensuring the best model state was preserved. Gradient clipping was used to prevent exploding gradients, capping the norm of gradients to a maximum of 1.0. Additionally, a learning rate scheduler, `ReduceLROnPlateau`, was employed to dynamically adjust the learning rate when validation loss improvements slowed.

The training configuration included the Mean Absolute Error (MAE) as the loss function and the Adam optimizer with a learning rate of 0.001 and a weight decay of 1×10^{-5}. The model was trained for up to 50 epochs with a batch size of 64, although early stopping often reduced the number of actual training epochs.

The results demonstrated the effectiveness of combining multiple activation functions and advanced training strategies. The parallel paths allowed the model to extract diverse feature representations, while early stopping and gradient clipping ensured stable training. The validation loss decreased consistently, highlighting the model's ability to generalize effectively to unseen data. This iteration represents a significant advancement in the modeling process by integrating multiple state-of-the-art techniques.

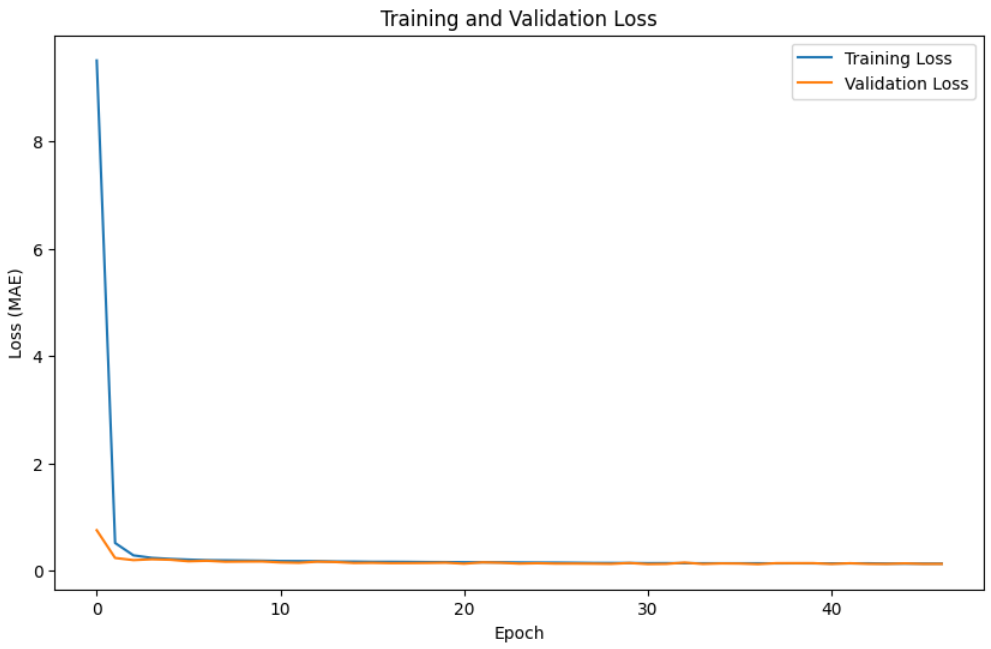


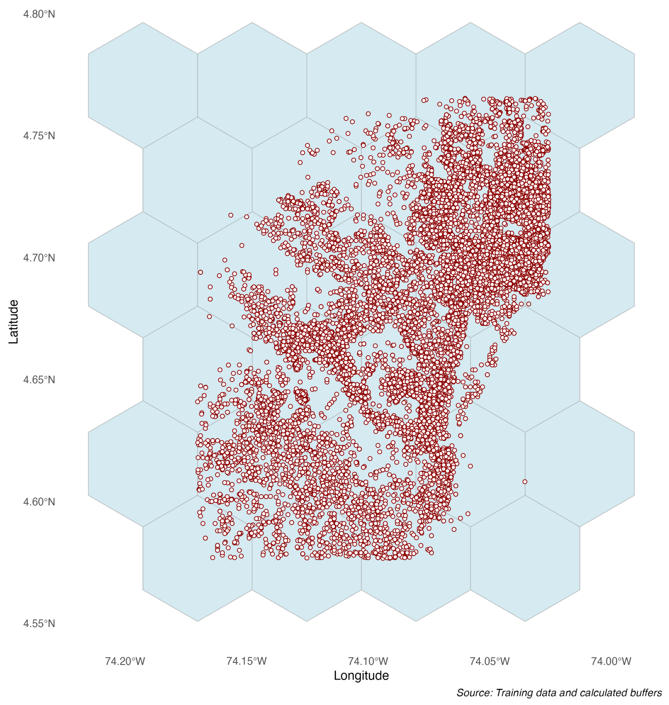
Figure 7: Training and Validation Loss for Model 8: Parallel Neural Network with Multiple Activation Paths and Early Stopping.

This model illustrates the value of combining innovative architectural designs with robust training techniques, contributing to improved convergence and predictive accuracy in the context of housing price prediction.

3.3 Elastic Net

Our model integrates spatial information and property-specific attributes to predict property prices effectively while accounting for spatial heterogeneity. We began by organizing the training data into spatial clusters using a grid of 5-kilometer hexagonal buffers. These clusters were designed to capture localized effects, such as proximity to specific amenities or regions, which are crucial for understanding variations in property prices. The spatial clusters served as a framework for grouping observations, which informed our model's cross-validation strategy and helped ensure robust evaluation. Figure 8 displays the geographical division into areas of 5 square kilometers.

Figure 8: Spatial Clusters, 5 km Buffers Around Training Points



We employed an Elastic Net regression model, chosen for its ability to handle multicollinearity and perform feature selection simultaneously. The predictors included both spatial and non-spatial variables: distances to landmarks, socioeconomic indicators, and property characteristics such as the number of rooms, bathrooms, and the presence of amenities. Spatial clusters were used to define cross-validation folds, allowing us to validate the model in a way that respects the underlying spatial structure. This approach ensures that the model generalizes well to new locations by preventing overfitting to localized patterns. This model achieved an RMSE of \$249,892,967 in the training data and an RMSE of \$258,779,634.88 in the test data. We implemented a 10-fold validation in the training sample. RMSE was used to select the optimal model by identifying the smallest value. The final parameters used for the model were alpha = 0.7 and lambda = 366,467.9.

References

Bertolín Mora, J. (2014, November). La burbuja inmobiliaria española: Causas y consecuencias [Convocatoria: Noviembre/Diciembre 2014].

de Bancos, S. (2022, March). Sistema de banca privada y pública: Informe del sector actividades inmobiliarias [Accedido el 26 de noviembre de 2024]. https://superbancos.gob.ec/estadisticas/portalestudios/wp-content/uploads/sites/4/downloads/2022/05/estudio-sectorial-inmobiliarias-mar-22.pdf

Guzmán, M. F. C. (2024). En 2023 el sector de construcción fue de 4,3% del pib en total y vivienda con 1.88% [Accedido el 26 de noviembre de 2024]. https://www.larepublica.co/

Jaro, M. A. (1989). Advances in record linkage methodology as applied to the 1985 census of tampa florida. *Journal of the American Statistical Association*, *84*(406), 414–20. https://doi.org/10.1080/01621459.1989.10478785

Li, P., Wang, Y., Zhang, L., & Zhang, X. (2024). China's gdp at risk: The role of housing prices. *The Journal of Finance and Data Science*, *10*, 100140. https://doi.org/10.1016/j.jfds.2024.100140

Mora-Garcia, R.-T., Cespedes-Lopez, M.-F., & Perez-Sanchez, V. R. (2022). Housing price prediction using machine learning algorithms in covid-19 times [Submission received: 17 October 2022 / Revised: 8 November 2022 / Accepted: 14 November 2022 / Published: 21 November 2022]. *Land*, *11*(11), 2100. https://doi.org/10.3390/land11112100

of St. Louis, F. R. B. (2024). Gross domestic product: Real estate (531) in the united states [Accessed: November 26, 2024]. https://fred.stlouisfed.org/series/USREALNGSP

Sharma, H., Harsora, H., & Ogunleye, B. (2024). An optimal house price prediction algorithm: Xgboost [Submission received: 21 November 2023 / Revised: 20 December 2023 / Accepted: 29 December 2023 / Published: 2 January 2024]. *Analytics*, *3*(1), 30–45. https://doi.org/10.3390/analytics3010003

Xu, K., & Nguyen, H. (2022). Predicting housing prices and analyzing real estate market in the chicago suburbs using machine learning [Submitted on 12 Oct 2022]. *arXiv preprint arXiv:2210.06261*. https://doi.org/10.48550/arXiv.2210.06261