Final Project: Multi-task Learning for Predicting House Prices and House Category

Vinay Govias

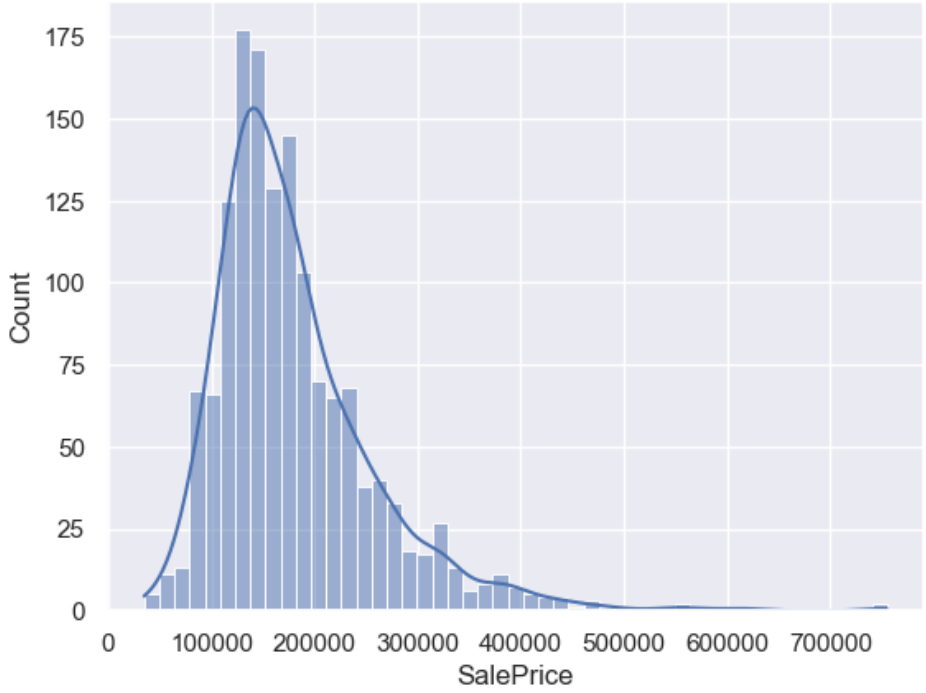
261143063

The objective is to construct a multi-task learning model performing two distinct but related tasks: predicting house prices (regression task) and classifying houses into categories (classification task). PyTorch Lightning is utilized to create the data loaders, model building, logging, early stopping etc. with multiple learning objectives for streamlined model development and training processes.

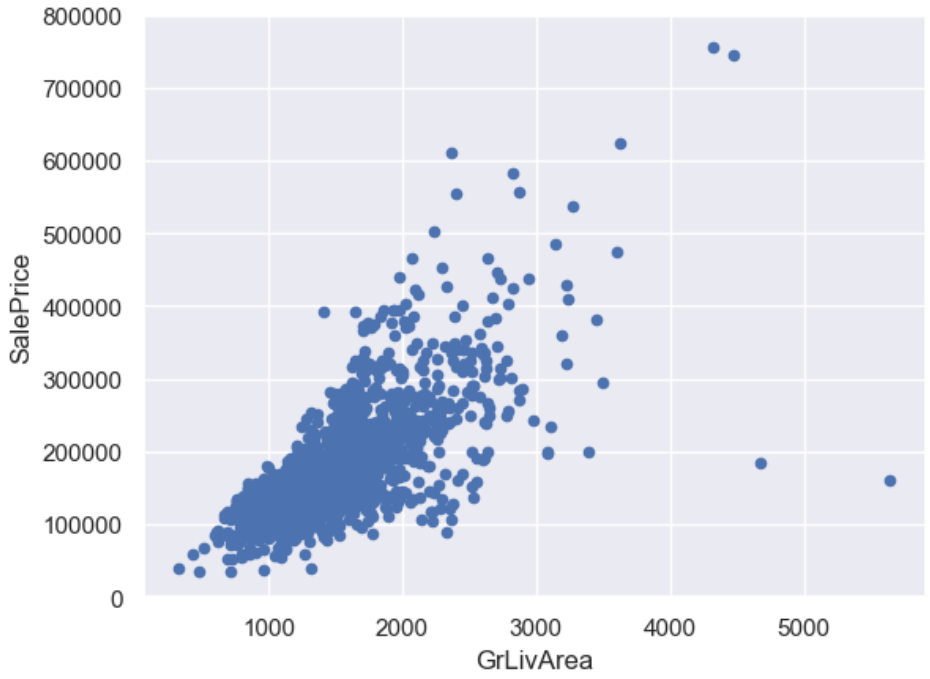
Exploratory Data Analysis:

The dataset is the House Prices - Advanced Regression Techniques Dataset. This dataset captures a variety of aspects of housing data: location, neighborhood, number of garages, fireplaces, Year it was built, remodelled, selling price etc. For the classification task, a new feature named 'House Category' is engineered based on 'House Style', 'Bldg Type', 'Year Built', and 'Year Remod/Add'.

Distribution of Sale Prices: A histogram shows the distribution of sale prices for the houses. The shape of this distribution shows it's right-skewed, most homes are sold at lower prices with a few exceptions at higher prices.



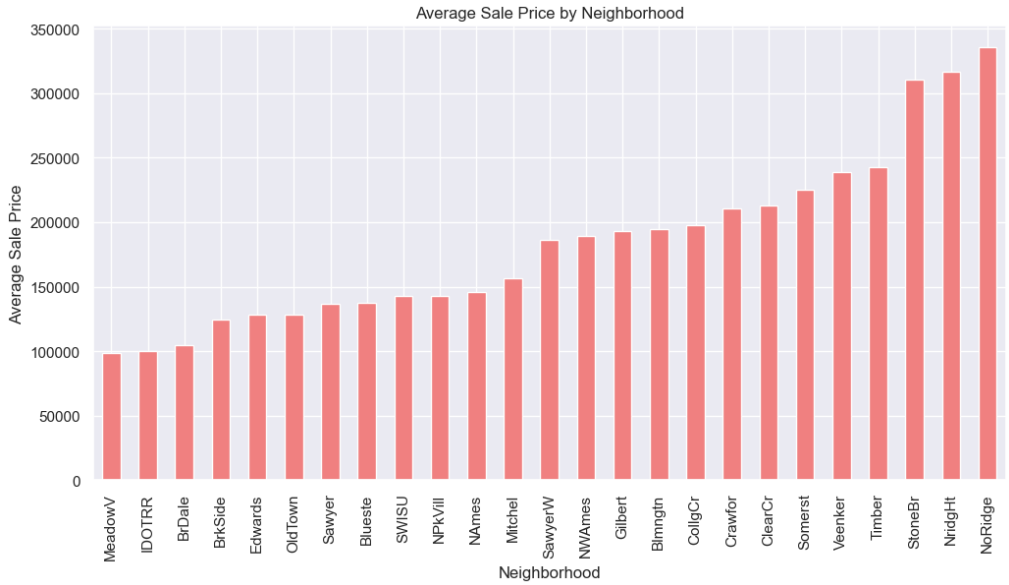
GrLivArea vs. SalePrice Scatter Plot: A scatter plot comparing 'GrLivArea' (above-grade living area) and 'SalePrice' shows a correlation between living area and the sale price of the houses. The upward trend in the plot would suggest that larger homes tend to sell at higher prices.



Average Sale Price by Zoning Classification: A bar chart indicating the average sale price for different zoning classifications (MSZoning) reveals if certain zones like residential, commercial, or agriculture are associated with higher or lower sale prices on average. Floating Village Residential and Low density Residential properties have the highest average sales price.

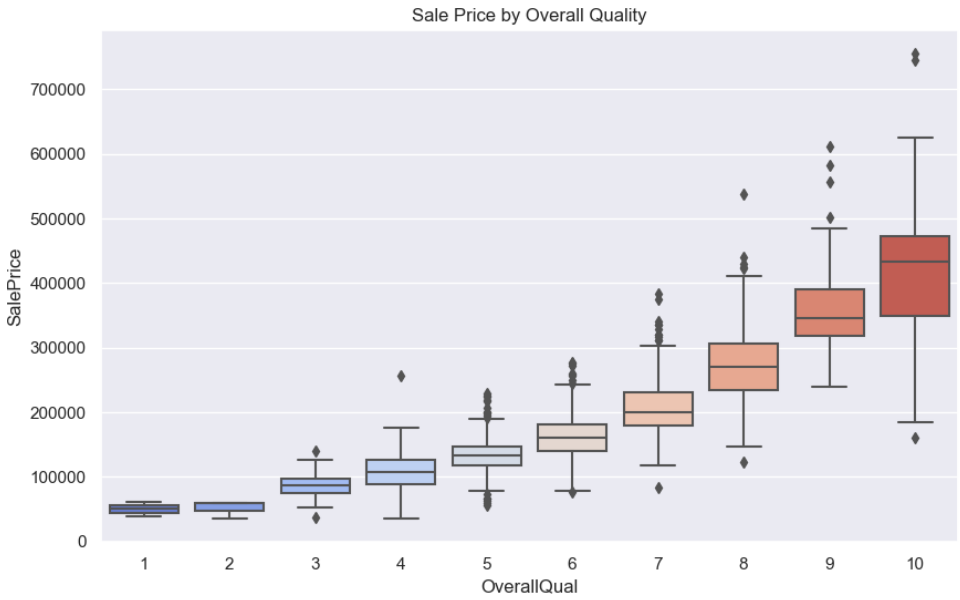
Average Sale Price by Neighborhood: A similar bar chart for neighborhoods shows location-based price trends, which might be useful for understanding the influence of location on house prices. The neighbourhoods with the highest average sale price are

Northridge, Northridge Heights, Stonebrook. Meadow Village and Iowa DOT and Railroad are on the opposite end with the lowest average sale price.

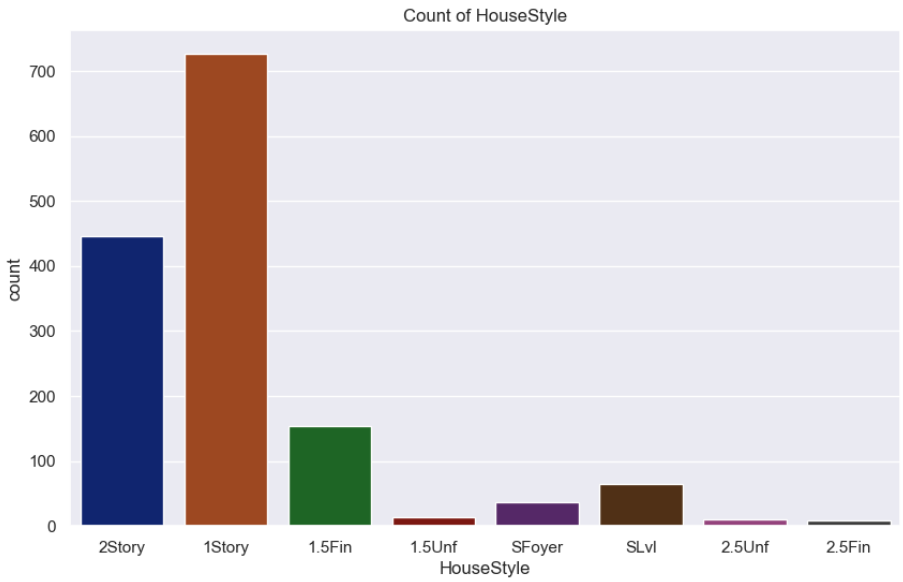


Sale Price vs. Lot Area Scatter Plot: This visualization explores the relationship between the size of the lot and the sale price. A positive correlation would suggest that larger lots of command higher prices. The scatterplot is not definitive.

Sale Price by Overall Quality Box Plot: A box plot provide insights into the relationship between the overall quality of a house and its sale price. Higher quality ratings correlate with higher sale prices.

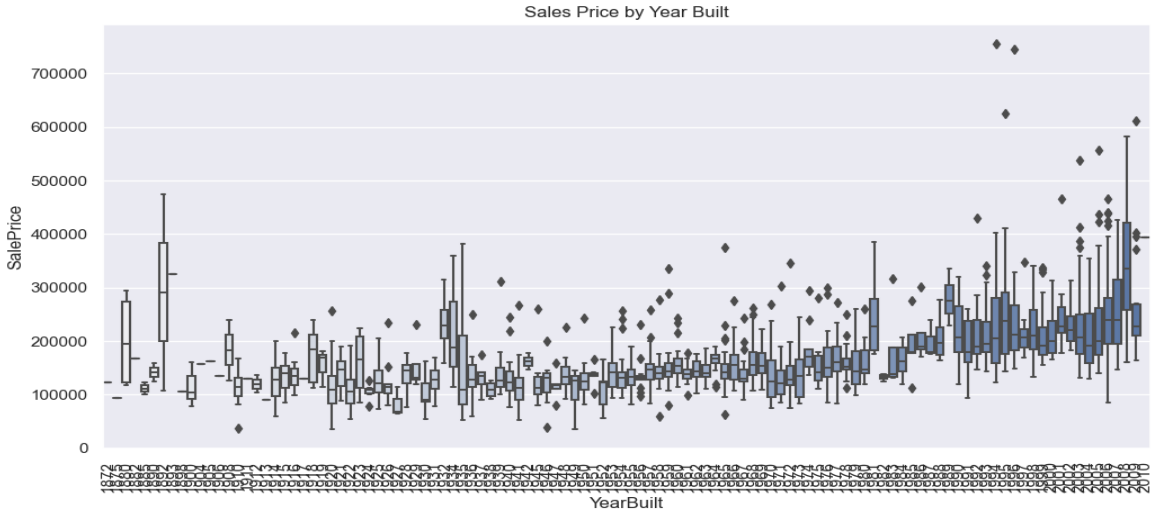


Count of House Style: A bar chart showing the count of different house styles can inform us about the popularity or availability of certain styles within the dataset. The dataset contains the majority of 1 Story and 2 Story buildings. Split level houses are a minority.

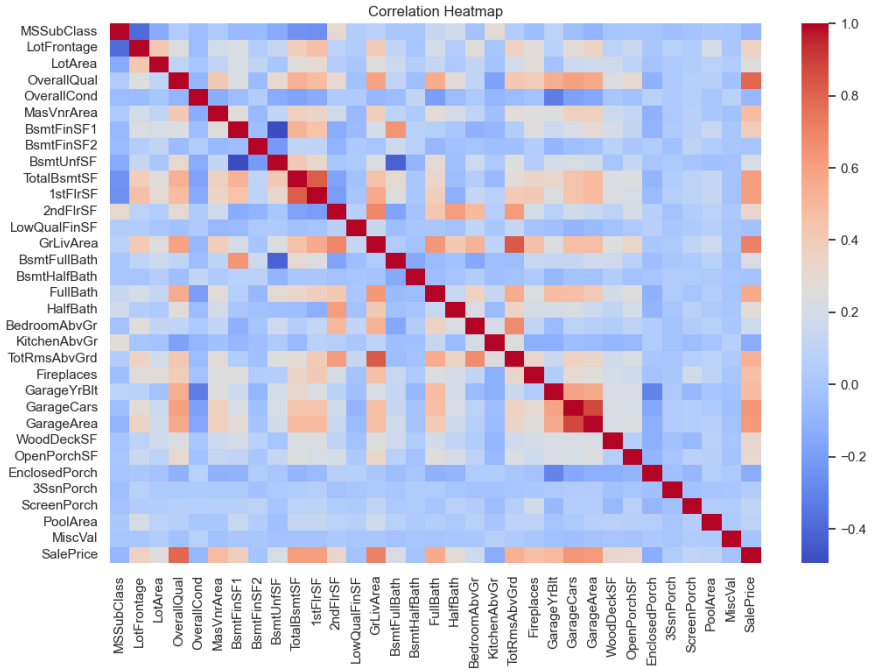


Sales Price by Year Built: This box plot shows the range and median sale price of houses according to their year built. The box plot indicates that newer homes tend to have higher median sale prices, but there is a lot of variability in prices for homes built in each period. Additionally, older homes (built before the 1900s) also show a wider range of sale prices,

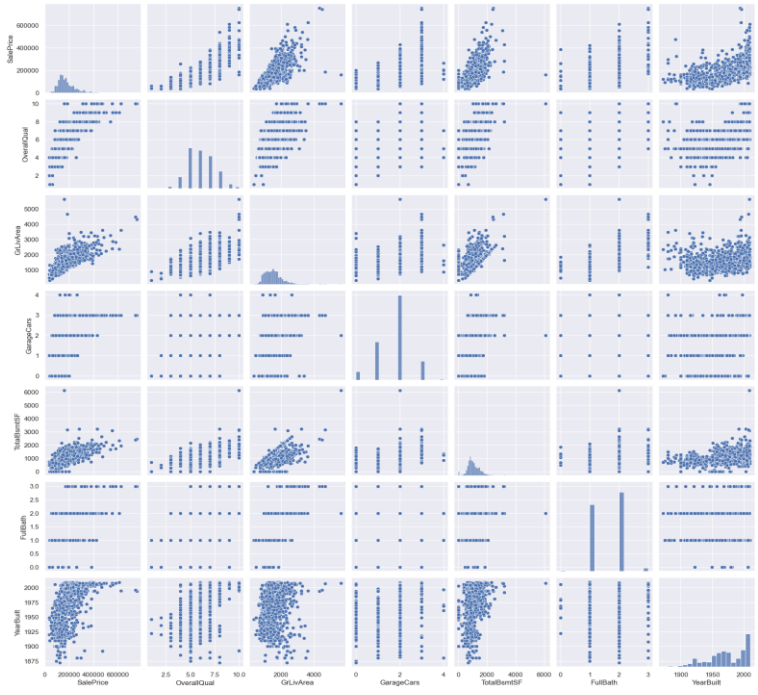
possibly reflecting historic value or renovations. We also see a large number of outliers in terms of sales price.



<u>Correlation Heatmap:</u> A heatmap illustrates the correlation between different variables. This can help identify which features are most related to sale price or to each other. The heatmap shows that certain variables like 'GrLivArea', 'TotalBsmtSF', and 'OverallQual' have a strong positive correlation with 'SalePrice', indicating these are important factors in determining the sale price of a house. 'OverallCond' and 'Enclosed Porch' seems to have a lower correlation with 'SalePrice', suggesting that the condition of the house is less of a price determinant than its size or quality.



Pair Plot: A grid of scatter plots for several variables can help quickly identify the relationships between pairs of variables and spot trends, clusters, and outliers. From the grid of scatter plots, we can observe relationships between variables. For instance, 'TotalBsmtSF' and 'GrLivArea' and 'Sales Price' have a strong positive correlation, which is logical as the basement size can impact the size of the living area and also the price. Also, 'YearBuilt' and 'OverallQual' appear to have some degree of correlation, suggesting that newer homes may generally have better materials and finishes.



Missing Values:

The percentage of missing values in each column was calculated ant the top 5 columns were as follows:

Column	Percent of Nulls
PoolQC	0.995205
MiscFeature	0.963014
Alley	0.937671
Fence	0.807534
FireplaceQu	0.472603

Data Preprocessing:

To prepare a dataset for a regression and classification deep learning task, the following steps are performed:

Classification Target Variable Feature Creation (create_features):

This function generates new ‘House Category’ features based on the existing columns to get the classification target variable. It reduces the categories of the BldgType and HouseStyle features by mapping various categories into broader groups. It also computes the age of the house using either the year built or the year of remodeling, whichever is later, and classifies it as 'New' or 'Old' based on whether this date is within 15 years of the current year.

The feature, 'House Category', concatenates the building type, house style group, and age category into a single string to capture the interaction between these features. The dataset uses 2016 as the 'current_year' to calculate the 'AgeCategory' because that is the date of the dataset was posted. This ensures that the age classification is relevant to the time of the data.

Conversion of Data types

Categorical variables that were initially present as integer data types ('MSSubClass', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities') are converted to the categorical data type, which is required for certain types of encoding later on. Several columns, including those used to create 'House Category', as well as 'Id', are dropped from df_train. This is because 'Id' is a unique identifier that does not provide predictive value, and the original features are now redundant due to the creation of 'House Category'.

Data Splitting:

The dataset is split into features (X) and targets (y), further splitting these into training, validation, and test sets to assess model performance and avoid overfitting. The preprocessing steps involve imputation for missing values and scaling for numerical features. For categorical features, missing values are filled with a constant string 'missing', and one-hot encoding is applied. A column transformer is used to apply these preprocessing steps to the appropriate columns. This helps in automating the workflow and keeping the preprocessing steps modular and organized. The full_pipeline encapsulates the entire preprocessing workflow, making it easy to apply the same transformations to new data.

Target Variable Handling:

Since there are 2 target variables- one continuous and the other categorical, they must be processed separately. The Sale prices are scaled using StandardScaler, which standardizes the target which is a common practice for regression tasks. For the classification target, 'House Category' is one-hot encoded, creating a binary column for each category. This is necessary for most classification algorithms. Finally, the regression and classification targets are combined horizontally into a single array for each of the train, validation, and test sets. This is required for a multi-output machine learning model.

Model Evaluation Process

The modelling process involves using a multi-task deep learning framework that is designed to handle both regression (predicting continuous values like sale prices) and classification (categorizing houses into different classes) tasks simultaneously. Three variations of the model were tested, and the evaluation was done on the performance of the three tested models on both MSE and classification criteria (accuracy, precision, recall, F1,ROC AUC).

Data Preparation for Pytorch Lightening

The data for the model has been converted into tensors, which are multi-dimensional arrays used by PyTorch. The code creates datasets and loaders for training, validation, and testing. DataLoaders allow for batch processing of the data, which is essential for training neural networks efficiently.

Model Architectures Tested

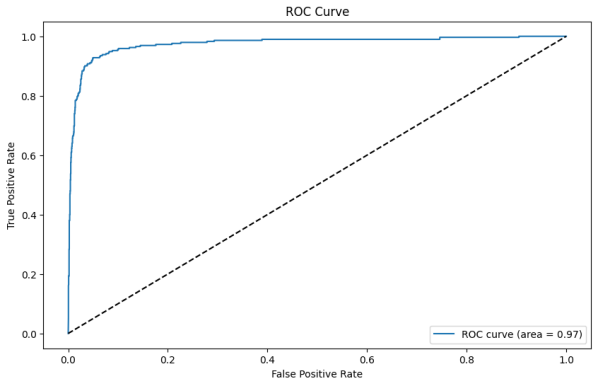
Model 1: Multi-Task Neural Network

This model is a multi-task network where both tasks share the same early layers and then branch off into their respective task-specific heads. The input passes through a fully connected layer (nn.Linear) with an output size of 128 nodes. The ReLU (Rectified Linear Unit) activation function is applied, introducing non-linearity and helping with the vanishing gradient problem. A dropout layer with a rate of 0.2 is included to regularize the model and reduce the chance of overfitting by randomly setting input units to 0 at each step during training time. Another fully connected layer reduces the dimension to 64 nodes, followed by another ReLU activation.

Regression Head: A single fully connected layer with one output node takes the output from the shared layers. This output node represents the regression prediction, i.e., the sale price. Since sale price prediction is a regression problem, no activation function is applied at this layer because we are predicting a continuous variable.

Classification Head: Similarly, it branches off from the shared layers and contains a fully connected layer with a number of output nodes equal to the number of classes in the classification task. The output from this head is a set of logits, one for each class.

Model-1 Results on Test Data:



Evaluation Results:

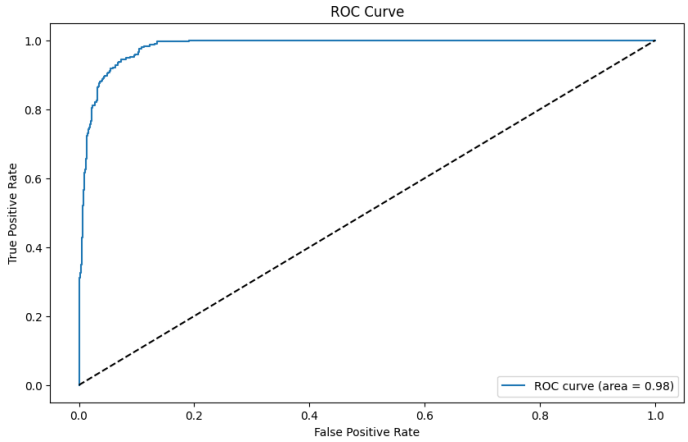
MSE: 0.252, Accuracy: 0.81, Precision: 0.81, Recall: 0.81, F1 Score: 0.80, ROC AUC: 0.97

Model 2: Alternative Activation Functions

The second model architecture explores the use of different activation functions and a lower dropout rate, which impacts the learning dynamics and performance of the model. The input first goes through a fully connected layer with 128 nodes, but in this architecture an ELU (Exponential Linear Unit) activation function is used. ELU is known to help reduce the vanishing gradient problem and often leads to better learning characteristics than ReLU. A dropout layer with a lower rate of 0.1 is used, indicating less regularization compared to Model 1. The next fully connected layer with 64 nodes uses a Tanh (hyperbolic tangent) activation. Tanh is a scaled sigmoid activation that maps values between -1 and 1. It's less commonly used in deep networks for hidden layers but can still be effective, especially when the data is normalized.

Regression and Classification Heads: This is the same as the Model 1 architecture in terms of structure, with the regression head outputting a continuous value and the classification head producing logits.

Model-2 Evaluation Results on Test data



Evaluation Results:

MSE: 0.263, Accuracy: 0.80, Precision: 0.80, Recall: 0.80, F1 Score: 0.79, ROC AUC: 0.98

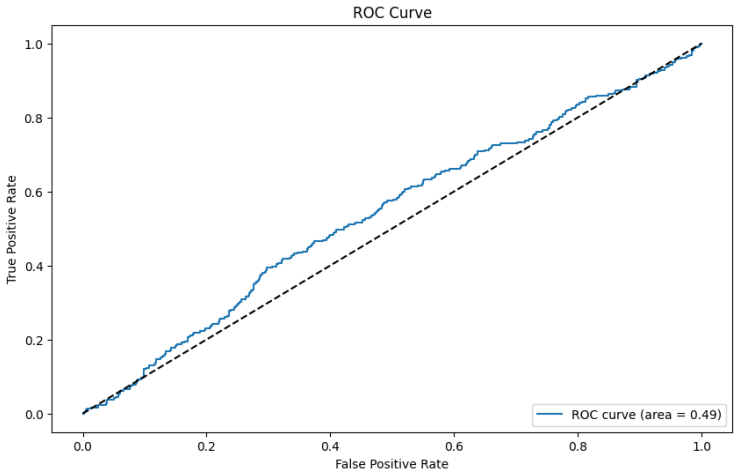
Model 3: Expanded Architecture with Batch Normalization

Model 3 is more complex, integrating additional layers and batch normalization, which could help in faster and more stable training. The initial fully connected layer is larger, with 256 nodes, followed by a ReLU activation function. Batch normalization is applied after the ReLU activation. Batch normalization standardizes the inputs to a layer for each mini-batch, stabilizing the learning process and reducing the number of epochs required to train deep networks. A dropout layer follows with a rate of 0.2. This process is repeated with another series of a fully connected layer, ReLU activation, batch normalization, and dropout, with a reduced number of 128 nodes in the second fully connected layer. The output from these shared layers is then passed through one more fully connected layer with 64 nodes and a ReLU activation to prepare for the task-specific heads.

Regression and Classification Heads:

These layers are also similar to the previous models. The regression head is a fully connected layer with one output node, and the classification head outputs logits after a fully connected layer.

Model 3 Evaluation Results on Test Data



MSE: 1.143, Accuracy: 0.13, Precision: 0.25, Recall: 0.13, F1 Score: 0.12, ROC AUC: 0.49

Optimizers Used

For all the above models, the training, validation, and testing steps involve computing the loss for both regression (using MSE loss) and classification (using cross-entropy loss) and then backpropagating the combined loss to update the network's weights. Adam optimizer is used in Models 1 and 2, known for combining the benefits of two other extensions of stochastic gradient descent: AdaGrad and RMSProp. Meanwhile, Model 3 employs an SGD optimizer with momentum, which helps accelerate gradients vectors in the right directions, thus leading to faster converging.

Early Stopping Callback

An early stopping callback is set up to monitor validation loss, preventing overfitting by stopping training if the validation loss doesn't improve after a certain number of epochs (patience=3).

Analysis of the Model Results

For the Model 1 Results the MSE is relatively low, suggesting good performance on the regression task. High accuracy, precision, recall, and F1 scores indicate strong classification performance, and the ROC AUC is close to 1, which suggests excellent discriminative ability.

Model 2's MSE and classification metrics are slightly worse than Model 1, but still competitive. The ROC AUC is very high, similar to Model 1, which is an good indicator for the classification performance.

The MSE of Model 3 is significantly higher, suggesting poor regression performance. The classification metrics are also substantially lower than the other two models, indicating

that Model 3 has performed poorly on the classification task. A ROC AUC of around 0.49 is no better than random guessing, which is a poor result.

Overall, Model 1 seems to perform best, with the lowest MSE and highest classification metrics. Model 3, despite its complexity, performs poorly. This could be due to overfitting, inadequate training, or a poor choice of hyperparameters and architecture for the given task.

Hyperparameter Tuning Using Optuna

Based on the initial performance of Model 1's architecture, further tuning is performed on it using Optuna. It automates the optimization process by systematically searching through various hyperparameter configurations and evaluating them.

Parameters selected for Tuning:

lr: The learning rate for the Adam optimizer controls how much the model's weights should be adjusted with respect to the loss gradient. The search is conducted on a logarithmic scale between 1e-5 and 1e-1 to find the optimal learning rate.

dropout_rate: The dropout rate helps prevent overfitting by randomly setting a fraction of the input units to zero during training. The search range is between 0.1 and 0.8, which allows for variability in regularization strength.

hidden_dim1 and hidden_dim2: These represent the number of nodes in the first and second hidden layers, respectively. The options provided are [64, 128, 256, 512], covering a broad spectrum from relatively small to quite large layers.

Tuning Model Definition:

The MultiTaskModel class is defined with the flexibility to vary its architecture based on the hyperparameters provided. The PyTorchLightningPruningCallback is used along with early stopping. This callback allows Optuna to stop a trial early if it's clear that the trial will not result in a best-case scenario, thereby saving computational resources. The study aims to minimize the validation loss (val_loss), which is a direct indicator of model performance on unseen data.The study conducts multiple trials (specified by n_trials), or until a specified timeout (timeout) is reached. Each trial involves training a model with a unique set of hyperparameters proposed by Optuna.

Best Hyperparameters and Model Selection:

Optuna provides the best performing hyperparameters, in this case:

lr: 0.0004680576040328229

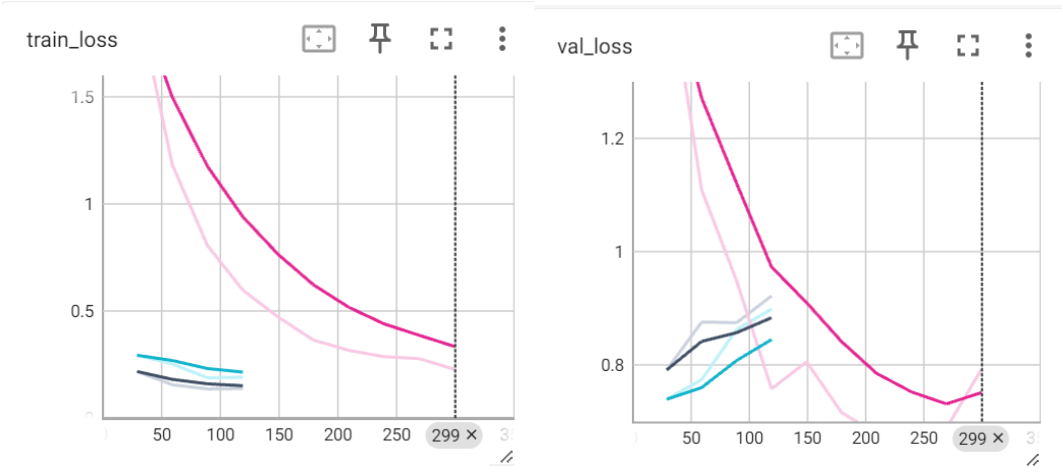
dropout_rate: 0.35443705476053255

hidden_dim1: 512

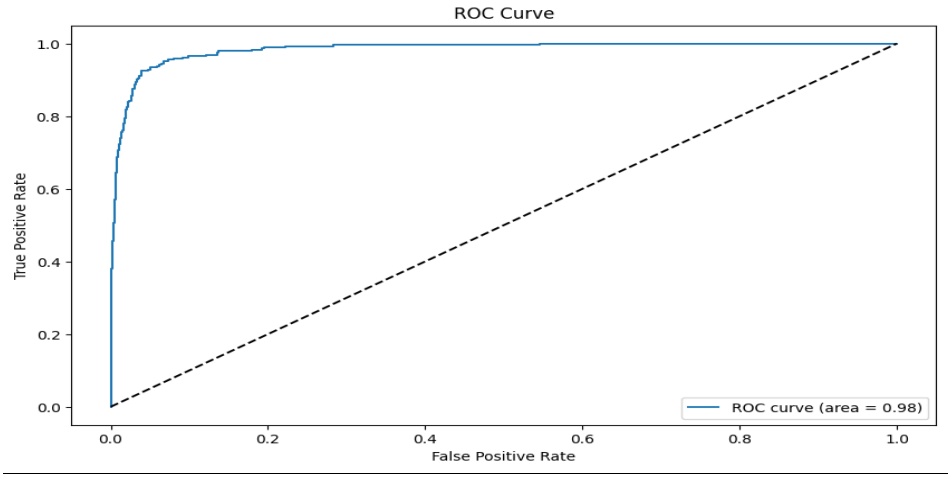
hidden_dim2: 512

Final Model Implementation:

Learning Curve:



All versions show a decreasing trend in training loss, which indicates learning is taking place. The pink/magenta line (version_2) starts off with the worst performance but then appears to learn quickly. All versions show that the validation loss is decreasing, which is a good sign that each version of the model is learning and improving its performance on the validation set. The fact that the validation loss is decreasing and not increasing (which would indicate overfitting) suggests that the model is generalizing well.



Evaluation Results:

MSE: 0.2494, Accuracy: 0.83 , Precision: 0.82 , Recall: 0.83 , F1 Score: 0.82, ROC AUC: 0.98

The final multi-task model incorporates the optimal hyperparameters discovered through the hyperparameter tuning process with Optuna. After training, the model is evaluated on the test dataset using a function evaluate_model, which computes various metrics. The results indicate that the model achieves a relatively low MSE of 0.2494, suggesting decent performance on the regression task. Classification metrics such as Accuracy, Precision, Recall, and F1 Score are around 0.83 and 0.82 respectively, showing strong performance on the classification task. The ROC AUC of approximately 0.98 which further confirms the model's robustness in classification, indicating a high true positive rate relative to the false positive rate. The results indicate that the model performs well on both regression and classification tasks with improved results from the initial Model 1 architecture, balancing the predictive accuracy with the ability to generalize. The trained model's parameters are saved in a .pth file for future inference or deployment.