Machine Learning Lab Report

Part 1 – Regression with Synthetic Data

In this section, we study how regression models — specifically multiple linear regression (MLR) and autoregressive exogenous input (ARX) models — function using pre-constructed synthetic data provided for the lab tasks. This setup highlights key principles of regression analysis as we examine the relationships between independent and dependent variables in both static and time series contexts. We also implement strategies to avoid common issues like outliers, and overfitting, especially given the limitations of small datasets. This work was conducted using `Python` 3.11 with libraries `scikit-learn` and `statsmodels`.

Submission 1 – Multiple Linear Regression with Outliers

We investigated the potential linear relationship between the concentration of toxic algae in a coastal region (y) and five environmental variables ($\mathbf{x} = [x_1, \ldots, x_5]^T$) using a MLR model. The acquired data is modeled as

$$\tilde{y}^{(n)} = \beta_0 + \sum_{i=1}^{5} \beta_i \tilde{x}_i^{(n)} + \eta^{(n)} + \xi^{(n)} \quad \text{and} \quad \tilde{x}_i^{(n)} = x_i^{(n)} + \eta_i^{(n)}, \tag{1.1}$$

where $n = 0, \ldots, N-1$ is the sample index, β_0 is the intercept, and $\beta_{1,\ldots,5}$ are the regression coefficients. The train dataset contains $N = 200$ samples, affected by zero-mean white Gaussian noise (η), with, approximately, an additional 25% of y values further corrupted by human error (ξ).

We started by developing a structured pipeline, with preprocessing and outlier removal, followed by hyperparameter tuning, cross-validation (CV), and testing, as shown in Fig. 1.1.

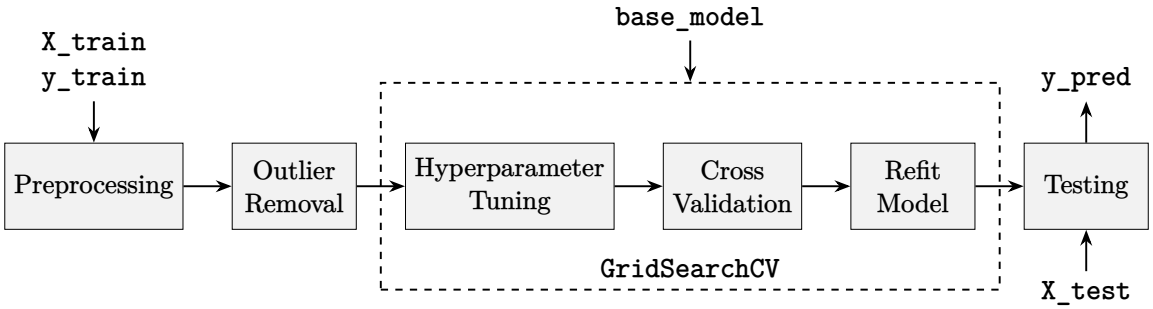


Fig. 1.1. Overview of the implemented structured pipeline for the MLR problem.

In the following subsections, we'll briefly describe the methods and configurations used at each stage.

A) Preprocessing and Outlier Removal: The input features are first *standardized*, transforming each feature to have zero mean and unit standard deviation. This ensures that all features contribute equally to the model, regardless of their original scale. Mathematically, we express it as

$$\tilde{\mathbf{x}}_{\text{train std}}^{(n)} = (\tilde{\mathbf{x}}_{\text{train}}^{(n)} - \boldsymbol{\mu}_{\text{train}}) \oslash \boldsymbol{\sigma}_{\text{train}} \quad \text{for} \quad n = 0, \ldots, N-1, \tag{1.2}$$

where $\tilde{\mathbf{x}}_{\text{train}}^{(n)}$ is the feature vector for the n-th sample, $\boldsymbol{\mu}_{\text{train}}$ is the vector of feature means, and $\boldsymbol{\sigma}_{\text{train}}$ is the vector of feature standard deviations. The operator $\oslash$ denotes element-wise division, ensuring that standardization is applied independently to each feature.

It should be noted that the same standardization process is applied to the test features using the means and standard deviations calculated from the training data, ensuring consistency. This is,

$$\tilde{\mathbf{x}}_{\text{test std}}^{(m)} = (\tilde{\mathbf{x}}_{\text{test}}^{(m)} - \boldsymbol{\mu}_{\text{train}}) \oslash \boldsymbol{\sigma}_{\text{train}} \quad \text{for} \quad m = 0, \ldots, M-1, \tag{1.3}$$

where $\tilde{\mathbf{x}}_{\text{test}}^{(m)}$ is the feature vector for the m-th test sample, and M is the number of test samples.

Next, we perform *outlier removal* on the target values, denoted as $\tilde{\mathbf{y}}_{\text{train}} = [\tilde{y}^{(0)}_{\text{train}}, \ldots, \tilde{y}^{(N-1)}_{\text{train}}]^T$. It is important to note that the target values do not need to be normalized. The used method follows an iterative approach based on the principles of residual analysis and outlier detection established by [1] and [2]. The algorithm can be summarized as follows:

1. Start by fitting an ordinary least squares (OLS) model to the data and compute the *externally studentized residuals* for each observation as $r_n = (\tilde{y}^{(n)}_{\text{train}} - \hat{y}^{(n)}_{\text{train}}) / \text{RMSE}_{(n)}\sqrt{1 - v_{nn}}$, where $\hat{y}^{(n)}_{\text{train}}$ is the predicted value for the n-th observation, $\text{RMSE}_{(n)}$ is the root mean squared error (RMSE) calculated excluding the n-th observation, and v_{nn} is the leverage of the n-th observation.
2. Observations with $|r_n| > t^{(193)}_{\alpha=0.005} = 2.839$[1] are flagged as outliers, which are then removed from the train dataset. The corresponding feature vector for that sample, $\tilde{\mathbf{x}}^{(n)}_{\text{train std}}$, is also discarded.
3. The model is refitted and the process is repeated iteratively until no $|r_n|$ values exceed the threshold.

This iterative detection and removal process reduces the impact of extreme values as can be seen in Fig. 1.2, resulting in a reliable set of train data. In total, 52 outliers were removed from the original dataset, which aligns with the initial assumption that approx. 25% of the target values are corrupted.

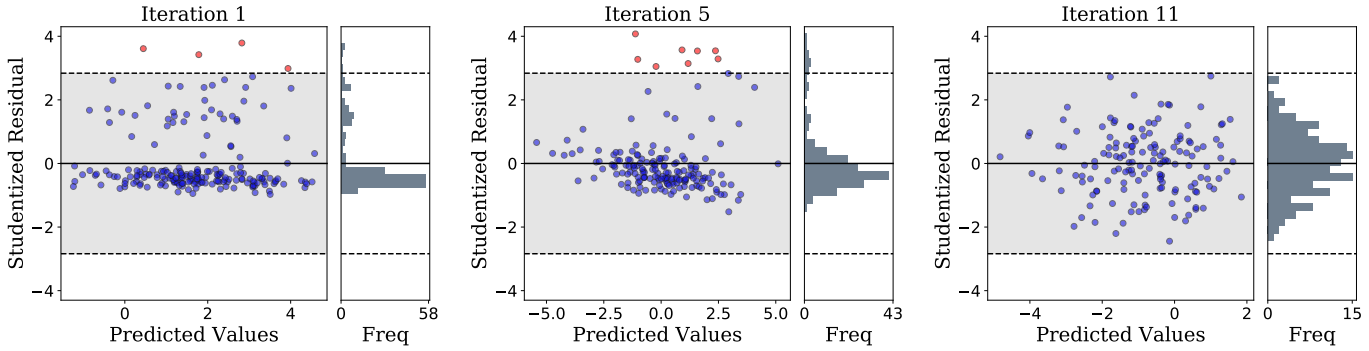


Fig. 1.2. Outlier removal process after the 1st, 5th, and final (11th) iterations. The left side plot shows the relationship between the predicted values and the studentized residuals, while the right side plot shows the distribution of the studentized residuals. Blue dots represent inliers, and red dots are considered outliers.

Additionally, for the last iteration, both the Breusch-Pagan test (p-value: 0.5828) and the White test (p-value: 0.3460) show no significant evidence of heteroscedasticity, suggesting that the residuals have stabilized with constant variance across the data. [3] This finding supports the effectiveness of the iterative outlier removal process in achieving a more reliable model fit. With homoscedastic residuals[2], the model's predictions will be more consistent across different data points, enhancing its reliability and interpretability for the next step of our pipeline.

B) Hyperparameter Tuning and Cross-Validation: At this stage, we made use of the `GridSearchCV` function imported from the `scikit-learn` package to systematically test a range of hyperparameters for each model, whose performance was evaluated using CV RMSE and respective standard deviation.

After outlier removal, the dataset was reduced to 148 points, with 75% of this data ($\approx$ 111 points) used for training and validation during CV.[3] We chose to perform the `GridSearchCV` with a 5-fold CV, fitting the model on 4 sets with $\lfloor 111 \cdot 4/5 \rfloor = 89$ examples and validating on a set with $\lfloor 111 \cdot 1/5 \rfloor = 22$. This ensures around $\lfloor 89/6 \rfloor$ points per estimator, providing a sufficient sample size for reliable parameter estimation while maintaining a robust validation process. [3, 4] After testing all hyperparameter combinations, the model is automatically refitted using the parameters that minimized the average RMSE score (with corresponding low standard deviation). This process is appreciable in Fig. 1.3.

[1] This is the critical value for a two-tailed t-distribution test with an overall significance level of $\alpha = 0.005$, corresponding to $\alpha/2 = 0.0025$ per tail, and 193 degrees of freedom (dof = observations − parameters − intercept − 1 = 193).
[2] We infer that the OLS is the best linear unbiased estimator, having no need for a generalized least squares method. [3]
[3] This split, done with `scikit-learn`'s `train_test_split`, ensures random partitioning and reproducibility of data.

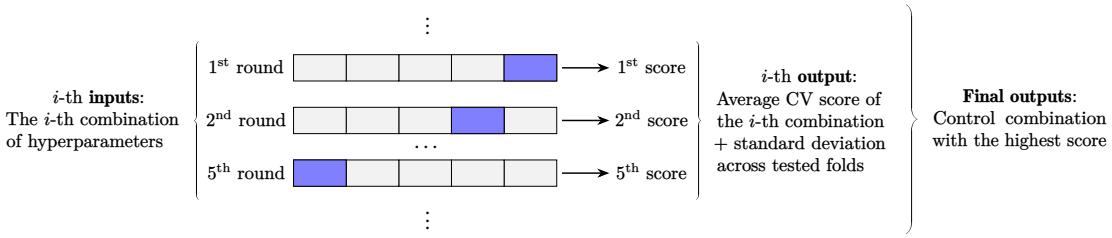


Fig. 1.3. Hyperparameter tuning using 5-fold CV (`GridSearchCV`).

This procedure provides a statistically robust estimate of model performance compared to a single training/test split. By averaging results across multiple folds, CV reduces the risk of selecting a model that performs well on a single test set by chance. This helps ensure that the chosen hyperparameters generalize well to unseen data. After identifying the optimal hyperparameters, the model is refitted on the entire dataset, allowing it to leverage all available data without overfitting to any particular subset. This final refitting step, combined with consistent CV results, validates that the model captures underlying patterns without memorizing the training data, thus minimizing the risk of overfitting. [3, 4]

C) Chosen Models and Results Analysis: To improve model generalization and further prevent overfitting, we explored several regularization techniques. L_1 regularization (`Lasso`) and L_2 regularization (`Ridge`) have the following objectives

$$\texttt{Lasso:}\ \min_{\beta,\beta_0} \sum_i V(\tilde{\mathbf{x}}_i, \tilde{y}_i) + \lambda \|\beta\|_1, \quad \texttt{Ridge:}\ \min_{\beta,\beta_0} \sum_i V(\tilde{\mathbf{x}}_i, \tilde{y}_i) + \lambda \|\beta\|_2^2, \tag{1.4}$$

where $V(\tilde{\mathbf{x}}_i, \tilde{y}_i)$ represents the error term for each sample $(\tilde{\mathbf{x}}_i, \tilde{y}_i)$ within the training data, with λ controlling the strength of regularization. `ElasticNet` combines these two approaches

$$\texttt{ElasticNet:}\ \min_{\beta,\beta_0} \sum_i V(\tilde{\mathbf{x}}_i, \tilde{y}_i) + \lambda \left(\rho \|\beta\|_1 + \frac{1-\rho}{2} \|\beta\|_2^2 \right). \tag{1.5}$$

Here, ρ is a mixing parameter that adjusts the balance between L_1 and L_2 regularization, allowing `ElasticNet` to combine both sparsity and coefficient shrinkage. [3, 4] We conducted extensive hyperparameter tuning and CV on the models in Tab. 1.1, which details CV metrics and optimized hyperparameter values. Tab. 1.2 presents metrics on the full dataset post-outlier removal, emulating a performance *upper bound* since the evaluation predicts all 200 points, including the removed outliers.

`Lasso`

Param	Value
CVRMSE	0.014430
STDDEV	0.007148
λ	0.0002

`Ridge`

Param	Value
CVRMSE	0.014435
STDDEV	0.007425
λ	0.0001

`ElasticNet`

Param	Value
CVRMSE	0.014309
STDDEV	0.007145
(λ, ρ)	$(0.0001, 0.9)$

Tab. 1.1. `GridSearchCV` results for `Lasso`, `Ridge` and `ElasticNet` (111 data points).

`OLS`

Metric	Value
MAE	0.011567
SSE	0.030001
R^2	0.999882

`Lasso`

Param	Value
MAE	0.011570
SSE	0.029805
R^2	0.999883

`Ridge`

Param	Value
MAE	0.011568
SSE	0.030008
R^2	0.999882

`ElasticNet`

Param	Value
MAE	0.011563
SSE	0.030017
R^2	0.999882

Tab. 1.2. Comparison of model *upper bound* metrics for `OLS`, `Lasso`, `Ridge` and `ElasticNet` (148 data points).

In addition to sum of squared errors (SSE), we incorporated R^2, which measures the proportion of variance in the dependent variable explained by the independent variables, and mean absolute error (MAE), which represents the average error magnitude between predicted and actual values. Based on these metrics, `Lasso` was chosen as the optimal model for further tuning using the full 148-point training dataset after outlier removal (another stage of `GridSearchCV`). This yielded the results:

$$\boxed{\texttt{CVRMSE} = 0.014681, \quad \texttt{STDDEV} = 0.005270, \quad \lambda = 0.0003.}$$

We also note that, interestingly, the best-performing hyperparameters presented above (also the ones in Tab. 1.1) suggested minimal regularization was necessary, likely due to the effective outlier handling.

For the final evaluation, we applied this model to `X_test`. Unfortunately, a coding mistake led us to standardize `X_test` using its own mean and standard deviation, rather than adhering to the intended normalization procedure in (1.3). This misstep caused `y_pred` to perform poorly, resulting in an SSE of 0.517380 and placing us in the 98$^{\text{th}}$ position. Without this issue, our experimental results indicated a performance that would have ranked us possibly within the top 10.

Submission 2 – Autoregressive Exogenous Input Model

We investigated system response estimation using an ARX model to capture the relationship between input and output signals of a discrete-time *dynamic system* (without feedback loops). The ARX model is formulated as

$$y(k) + a_1 y(k-1) + \ldots + a_n y(k-n) = b_0 u(k-d) + \ldots + b_m u(k-d-m) + e(k), \tag{2.1}$$

where $y(k)$ represents the *system output* at discrete-time step k, $u(k)$ is the *exogenous input*, n and m are the model order parameters, d is the input delay, $a_{1,\ldots,n}$ and $b_{0,\ldots,m}$ are the model coefficients, and $e(k)$ denotes the noise term, which is assumed to be a zero-mean white Gaussian process.

The problem can be reformulated as a MLR by writting (2.1) in regressor form

$$y(k) = \phi(k)^T \theta + e(k) \quad \text{for} \quad k = 0, \ldots, N-1, \tag{2.2}$$

where $\phi(k) = [y(k-1), \ldots, y(k-n), u(k-d), \ldots, u(k-d-m)]^T$ and $\theta = [-a_1, \ldots, -a_n, b_0, \ldots, b_m]^T$ for a given configuration of n, m and d. The data set has a total of $N = 2040$ points and we operate with $k \in [p = \max(n, m+d),\ N-1]$. To tune the ARX model parameters, we implemented a pipeline similar to that used in the previous regression task, as shown in Fig. 1.1. However, unlike the previous task, outlier removal was unnecessary given the clean nature of this dataset. Moreover, besides the model hyperparameters, this model also adds (n, m, d) to the mixture.

To proceed, we first conduct an exploratory analysis of the data, followed by a tweaked cross-validation strategy compatible with the temporal dependencies of time series data. This pipeline is then adapted to perform `GridSearchCV` across various configurations of (n, m, d), as to refine the model.

A) Data Analysis: For the initial analysis, we examined the autocorrelation (ACF) and partial autocorrelation (PACF) functions of the output data, shown in Fig. 2.1. The ACF and PACF indicate strong dependencies up to lag 9, with the PACF showing a sharp drop in correlation beyond this point. This suggests that setting $n = 9$ is sufficient, as additional lags would add minimal explanatory power. Further tuning of parameters, particularly m and d, will be explored in subsequent experiments.

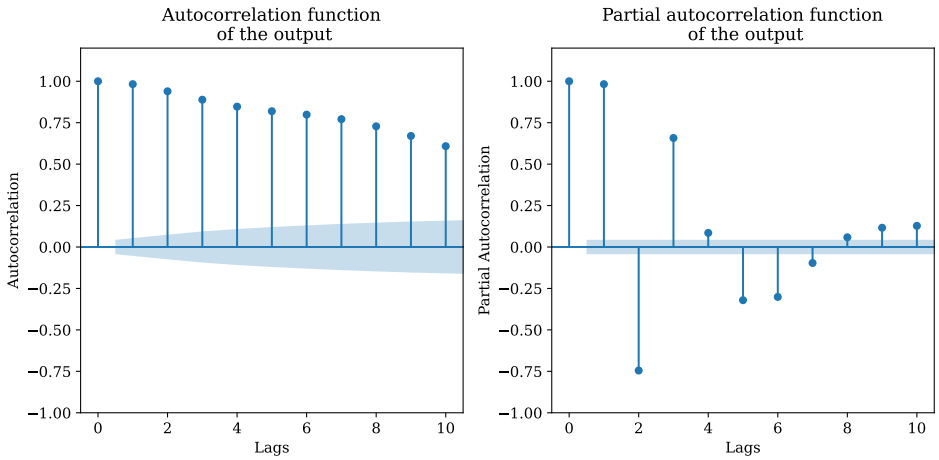


Fig. 2.1. Correlation plots for ARX model parameter estimation.

B) Hyperparameter Tuning and Cross-Validation: In our hyperparameter tuning process, we used CV within a loop that varied the model order. The structure of this loop was as follows: for each n, m and d setting, `GridSearchCV` was applied to optimize model parameters, and the results were subsequently checked for stability.

Based on the previous analysis, we fixed $n = 9$. We then explored values of m to identify the optimal exogenous input order. After extensive testing, we determined that $m = 12$ yielded the best model according to the Akaike information criterion (AIC) [5], which is particularly useful here, as it penalizes higher-order models and helps prevent overfitting by balancing model complexity with goodness of fit. Additionally, in all runs, fixing $n = 9$ consistently led to $d = 6$ as the optimal delay parameter.

The final model, therefore, was selected with parameters $n = 9$, $m = 12$[4], and $d = 6$, achieving the best CV metrics[5]. In addition, stability was checked based on the rule explained below:

We analyze the stability of the ARX model by considering its characteristic polynomial $\mathcal{P}(z)$. Nullifying the input, the autoregressive part of the model can be written as

$$y(k) + \hat{a}_1 y(k-1) + \ldots + \hat{a}_n y(k-n) = 0 \quad \xrightarrow{\text{Z.T.}} \quad 1 + \hat{a}_1 z^{-1} + \ldots + \hat{a}_n z^{-n} = 0, \tag{2.3}$$

where the coefficients $\hat{a}_1, \ldots, \hat{a}_n$ are the (negated) estimated parameter values. This expression is obtained by applying the Z-transform to the ARX equation (with no input), which transforms the difference equation into its z-domain polynomial. Multiplying both sides by z^n, we derive

$$\mathcal{P}(z) = z^n + \hat{a}_1 z^{n-1} + \hat{a}_2 z^{n-2} + \ldots + \hat{a}_n = 0. \tag{2.4}$$

The stability of the system is determined by the roots of the polynomial $\mathcal{P}(z)$, this is, if all roots lie strictly within the unit circle in the complex plane (i.e., $|z_{\text{root}}| < 1$ for all roots), then the system is considered *asymptotically stable* — the output will remain bounded. Otherwise the system is either *marginally stable* or *unstable*. If a model is deemed unstable after the hyperparameter tuning and CV, it is automatically discarded, leaving out only the best stable model solution. We've also supported these results with the unit-step response and root locus of the model in `MATLAB®`.

It is imperative to add that, for cross-validation, we used a blocking time series split, a method that divides the dataset into sequential blocks to maintain temporal order. This approach keeps training and validation sets distinct and ordered chronologically, preventing data leakage and respecting the time-dependent structure, which is essential for ARX models [6].

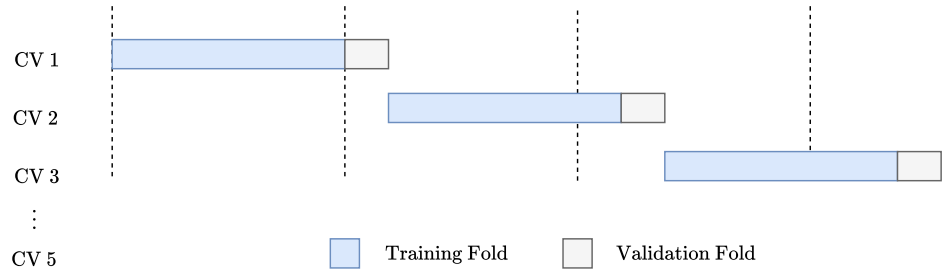


Fig. 2.2. Illustration of a blocked time series split. Each dashed line represents a data time index. We opted for a 5-fold split.

C) Chosen Models and Results Analysis: We chose to test the same models as in the previous submission, however, we omit OLS without regularization, since these high order time series are very prone to overfitting. The results can be seen in Tab. 2.1. For the final evaluation, we applied the selected ARX model based on `Ridge` to the latest 400 values of `U_test`.

[4] It should be added that, although the guide suggested testing n, m, and m only up to 10, it was clarified that we could explore higher values if it improved model performance.

[5] As outlined in the previous submission, we used SSE and R^2 for our *upper bound* metrics.

Our efforts yielded an SSE of 4.107975 that secured us the 2nd position on the leaderboard. This outcome validated the robustness of our tuning and preprocessing choices, confirming that our approach effectively captured the essential dynamics of the system.

`Lasso`

Param	Value
CVRMSE	0.114690
STDDEV	0.042286
CV R^2	0.999630
STDDEV	0.000134
λ	0.0002

`Ridge`

Param	Value
CVRMSE	0.113645
STDDEV	0.042492
CV R^2	0.999637
STDDEV	0.000132
λ	0.0010

`ElasticNet`

Param	Value
CVRMSE	0.114242
STDDEV	0.041747
CV R^2	0.999634
STDDEV	0.000128
(λ, ρ)	$(0.0002, 0.1)$

Tab. 2.1. CV results for `Lasso`, `Ridge` and `ElasticNet`, and respective best parameters.

Part 2 – Image Analysis

In this section, we address two primary tasks involving low-resolution grayscale Martian satellite images: crater classification and segmentation. For crater detection, we implement machine learning models to classify the presence of craters with a focus on accuracy and reliability. In the segmentation task, we explore both patch-based and binary mask approaches to delineate crater regions, evaluating the strengths and limitations of each method in capturing crater boundaries. This work was conducted using `Python` 3.11 with `torch`, `torchvision`, `torchmetrics`, `pytorch-lightning`, and `optuna`.

Submission 3 – Image Classification

This task aims to classify craters in low-resolution grayscale images of Mars by distinguishing non-crater images (label 0) from crater images (label 1). Craters often exhibit subtle, indistinct features, so the goal here is to build a model that can reliably detect these patterns. Model performance is assessed with the macro F1 score to ensure balanced sensitivity across both classes, emphasizing precision and recall equally for craters and non-craters.

We explore two distinct classification approaches for this problem: a support vector classifier (SVC) and a convolutional neural network (CNN). The pipeline for both methods begins by addressing data imbalance and incorporates geometric transformations for data augmentation to improve feature generalization. Hyperparameter tuning is subsequently applied to identify optimal configurations for each approach, with final model selection and analysis based on the best-performing configurations.

A) Data Preprocessing: In this classification task, we observed a notable class imbalance in the original dataset, with a distribution of 1006 non-crater (label 0) and 1777 for crater (label 1) classes. This imbalance typically causes models to favor the majority class, leading to biased predictions.

We began by reserving 20% of the dataset as a validation set using a stratified `train_test_split`, which preserves the original class imbalance in the subset, i.e., (196, 361). This ensures that the validation set is a reliable basis for evaluating the model's performance. After that, we tried undersampling the majority class of the remaining 80%, which we'll denote as train set henceforth, but found that this approach degraded model performance due to a loss of data.

To address this, we applied synthetic minority oversampling technique (SMOTE) [7] to balance the class distribution from (810, 1416) to (1416, 1416). SMOTE generates new crater samples by creating synthetic images that blend existing samples in the minority class. For each new sample, SMOTE selects a minority sample and one of its nearest neighbors, then interpolates between their values. The resulting synthetic sample is rounded and clipped to ensure it stays within the valid grayscale range of [0, 255]. This balanced dataset supports a more robust training process, enhancing the model's ability to generalize effectively across both classes. [7]

Following SMOTE, we applied data augmentation on the balanced training set to enhance model generalization. Specifically, we implemented random horizontal and vertical flips, rotations of $\pm 90^\circ$,

all with a probability of $p = 0.5$. These transformations increase diversity within the training data, allowing the model to better capture crater features and reducing overfitting on specific orientations and patterns (and theoretically increase generalization).

B) SVC Model Approach: SVC works by finding an optimal hyperplane that maximizes the margin between classes, helping it generalize well even with complex data structures. The pipeline we implemented includes, in addition to the data processing explained above, hyperparameter tuning, and model evaluation, similarly to what is done for the first submissions.

We used the SVC model provided by `scikit-Learn`, with a grid search to find the best values for C and γ, two key hyperparameters. C controls regularization and gamma determines the influence of individual data points on the decision boundary; high values create more complex, localized boundaries, whereas low values yield smoother, more generalized ones. `GridSearchCV` was used to select the optimal parameters based on the F1 score. This yielded:

$$\texttt{CV F1 score} = 0.765137, \quad \texttt{STDDEV} = 0.003034, \quad \gamma = 0.1, \quad C = 0.1.$$

The low values for both C and γ indicate that the model benefits from a simpler decision boundary with lower regularization. However, as we will see below, the performance of the CNN was significantly better, suggesting that the CNN is better suited to capture the patterns in this dataset.

C) CNN Model Approach: We employed a CNN architecture as shown in Fig. 3.1. The network consists of three main convolutional blocks, each designed to capture increasingly complex crater features through deeper layers. The first block begins with a 3x3 convolution layer using 32 filters, followed by two more 3x3 convolutions with 64 filters, batch normalization, and max-pooling to reduce spatial dimensions. The second block includes three more 3x3 convolutions, increasing from 64 to 256 filters, followed by batch normalization and max-pooling, which helps capture spatial relationships critical to crater detection. The third block, with a single 3x3 convolution layer of 512 filters, uses batch normalization and max-pooling to capture higher-level features.

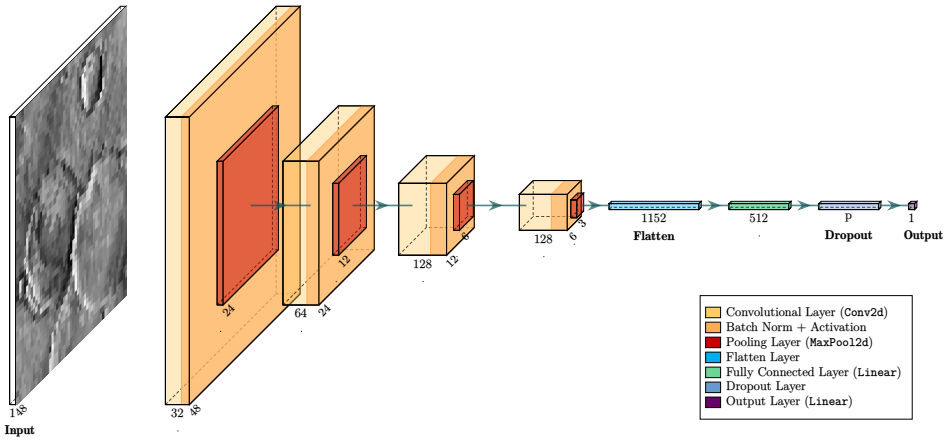


Fig. 3.1. CNN architecture used for the binary classification task.

A global average pooling layer reduces the spatial dimensions to a single 512-feature channel, followed by a fully connected layer with 256 units, where dropout (optimized via `Optuna`) mitigates overfitting. The output then passes to a single unit for binary classification.

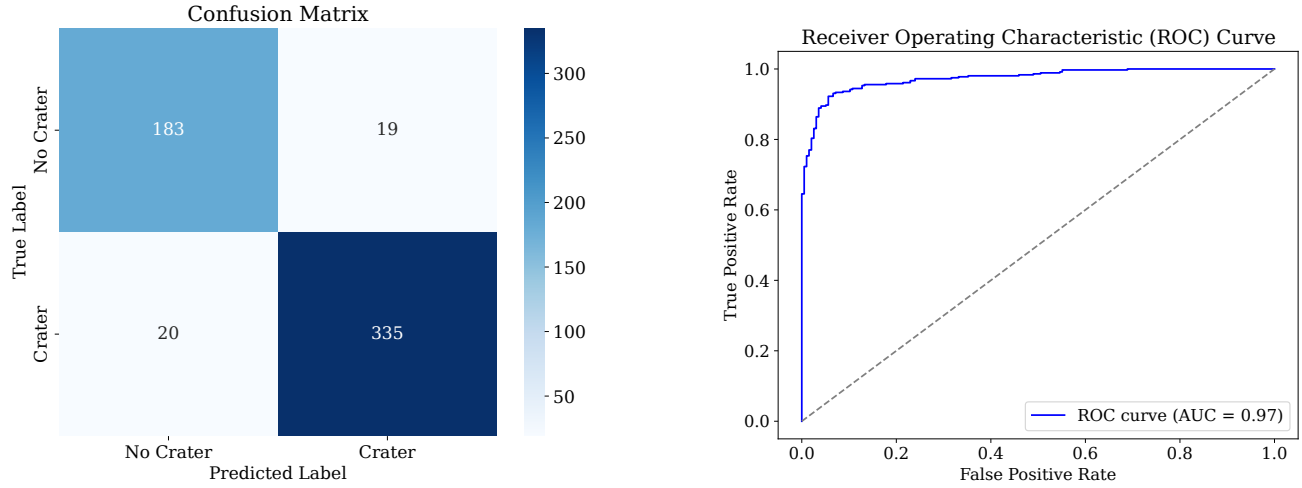
To avoid overfitting, we implemented several regularization strategies: weight decay (L_2 regularization) to constrain weight magnitudes, dropout in the fully connected layer with the dropout rate fine-tuned by `Optuna`, early stopping with a patience of 10 epochs, and a learning rate reduction on plateau that halves the rate when the validation F1 score plateaus.

Using `Optuna`, we systematically optimized the learning rate, dropout, and decision threshold to maximize the macro F1 score to ensure enhanced generalization and reliable classification.

D) Results Analysis and Discussion: After properly tuning the models we performed an analysis of their results on the remaining set put aside for validation purposes. Notice that this set was left unbalanced as to mimic realistic results.

Due to space constraints, we only present the visualizations for the results of our best-performing model, the CNN, which achieved an F1 score of 0.924 on the validation set. In comparison, the SVC model yielded a lower F1 score of 0.765 (close to the average CV one), highlighting our CNN's superior ability to capture the complex patterns and subtle features that characterize craters.

The CNN's high F1 score reflects balanced performance across both crater and non-crater classes, indicating effective learning of relevant features. Additionally, the ROC curve with an AUC of 0.97 (Fig. 3.2b) confirms the CNN's strong discriminative power. This suggests that the model generalizes well and is robust to varying crater presentations. The effectiveness of our analysis and chosen approach was further validated by our model's performance on the final leaderboard, where the CNN model achieved 4$^{\text{th}}$ place with an F1 score of around 0.916 on the test set.



(a) Confusion matrix.

(b) ROC curve, showing the AUC.

Fig. 3.2. Performance evaluation of the CNN model on the validation set.

Submission 4 – Image Segmentation

For this task, we implemented two distinct segmentation models to identify crater regions within low-resolution grayscale images of Mars. The crater segmentation problem requires accurately classifying each pixel as either part of a crater or background , with performance evaluated using the Balanced Accuracy metric.

We start by applying data preprocessing, addressing pixel class imbalance using stratified sampling techniques. For segmentation, we explored both an multiple linear regression (MLP) model (MLP-Fusion) and a U-Net architecture, each tailored to capture spatial relationships within the images.

A) Data Preprocessing: In our preprocessing pipeline, we implemented data augmentation and class balancing techniques to prepare the dataset for crater segmentation. Similar to the approach in the previous classification task, we applied data augmentation with geometric transformations, including random flips and rotations, to improve generalization and increase data variability. However, unlike the classification task, where we used SMOTE to handle class imbalance, here we managed the imbalance by computing class weights directly from the dataset.

These class weights were then passed to the loss function, assigning greater importance to underrepresented crater pixels during training. This approach effectively directed the model's learning toward accurate crater detection, supporting balanced performance across both crater and background regions and ensuring that the segmentation model could generalize well to new images.

B) MLP-Fusion Model Approach: We first employed a Fused MLP Mixer architecture with two parallel branches: a local branch for capturing patch-level details and a global branch for capturing the overall image context. The local branch processes 7x7 patches through a patch embedding layer, followed by token mixing to enable interaction across patches, and channel mixing. The global branch processes the entire 48x48 image, starting with a global embedding layer, followed by a global MLP. Finally, the outputs from both branches are concatenated into a combined representation that includes both local and global information, which is then passed through a fusion layer, which compresses it back to a unified hidden dimension.

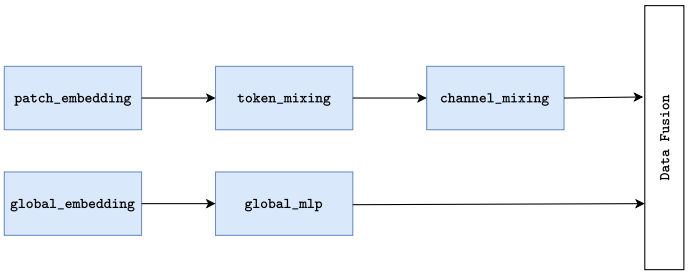


Fig. 4.1. Diagram of the Fused MLP Mixer model, showing parallel processing of local patches and global image context, followed by data fusion.

We implemented several regularization strategies in the MLP model akin to what as been described in the previous section.[6] The model achieved a balanced accuracy (BAcc) of 0.6674, reflecting moderate performance in distinguishing between classes. This result falls short in fully capturing the task at hand. As will be shown, the following model achieves significantly better performance.

C) U-Net Model Approach: identification. [8]

The U-Net architecture, shown in Fig. 4.2, is a CNN specifically designed for image segmentation, making it highly effective for pixel-wise classification tasks, such as crater identification, where both fine-grained details and contextual information are crucial [8].

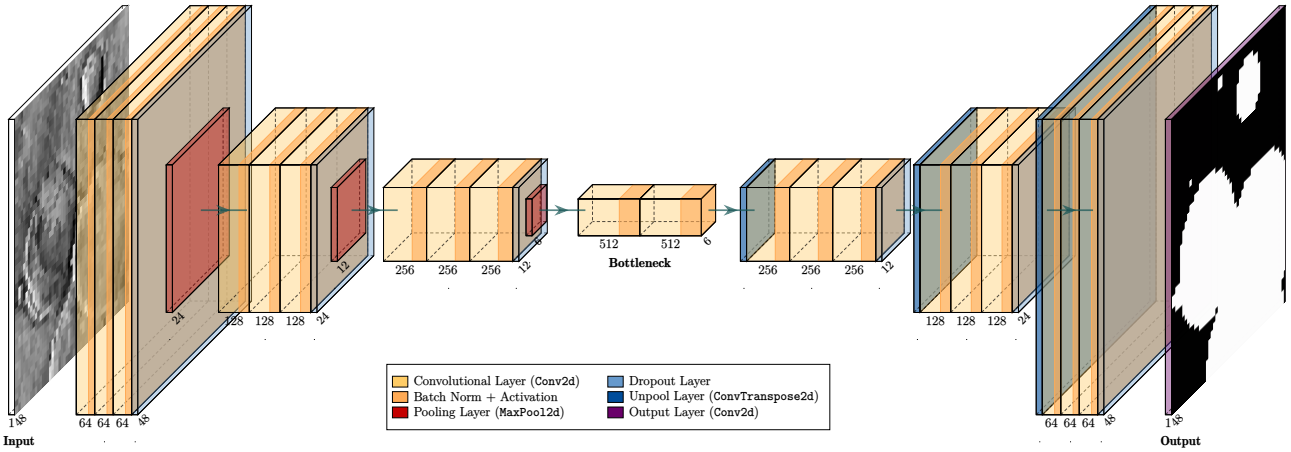


Fig. 4.2. U-Net architecture for segmentation. The network consists of an encoder (left side) and a decoder (right side) connected through a bottleneck.

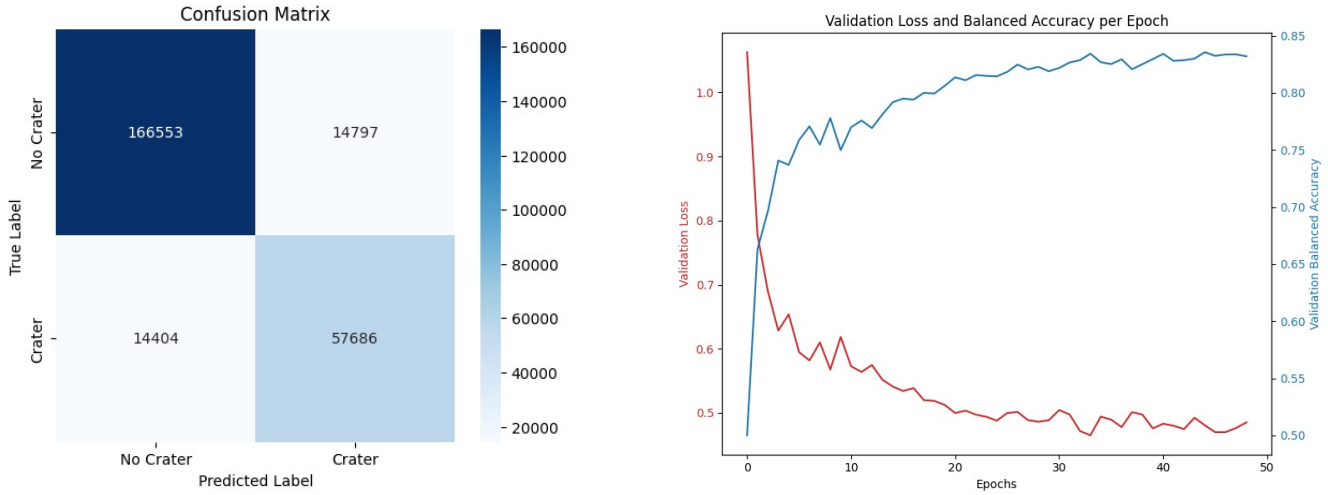
U-Net's design consists of a symmetric encoder-decoder structure that facilitates the capture of multi-scale features and sharp boundary delineation. The encoder path (left side) progressively reduces the spatial resolution of the input image while capturing increasingly complex features. In our implementation, each encoder block contains three 3x3 convolutional layers with ReLU activation and batch normalization, followed by dropout for regularization and a max-pooling layer for downsampling. This approach allows the model to capture global context efficiently by aggregating spatial information at varying scales. At the bottleneck layer, which represents the most abstract feature set, three

[6] Early stopping and learning rate reduction.

additional convolutional layers encode the highest-level features with minimal spatial information, ensuring that the model can later reconstruct relevant details accurately. The decoder path (right side) mirrors the encoder structure but performs upsampling through transposed convolutions to restore spatial resolution. To retain essential fine details lost during downsampling, U-Net employs skip connections between encoder and decoder layers at each depth level. These connections concatenate feature maps from the encoder to corresponding decoder layers, integrating high-resolution spatial details with the decoder's upsampled features, a crucial factor for precisely localizing crater boundaries. The network concludes with a 1x1 convolution and a sigmoid activation to produce a probability map, which indicates the likelihood of each pixel belonging to a crater.

For optimization, we used `Optuna` to tune key hyperparameters, including learning rate, dropout rate, and threshold for segmentation accuracy. Regularization strategies like dropout and weight decay, coupled with a learning rate reduction scheduler, were applied to mitigate overfitting (similarly to the previous image task), ensuring robust performance even with class imbalance.

D) Results Analysis and Discussion: The U-Net model demonstrated robust performance on the validation set, achieving a balanced accuracy of 0.844 without any clear signs of overfitting, as evidenced by the stable trend in the validation loss and balanced accuracy plots in Fig. 4.3b. The confusion matrix in Fig. 4.3a shows strong classification across both crater and non-crater classes, with well-balanced predictions and relatively low false positives and false negatives. Throughout training, the validation loss consistently decreased, stabilizing after about 15 epochs, while the balanced accuracy steadily increased and plateaued, indicating effective learning and generalization. At final evaluation on the test data, this approach yielded 2$^{\text{nd}}$ place on the leaderboard with a balanced accuracy score of 0.835, confirming the model's reliability in segmenting crater regions.



(a) Confusion matrix. **(b)** Validation loss and balanced accuracy.

Fig. 4.3. Performance evaluation of the U-Net model on the validation set.

References

[1] F. Anscombe and J. Tukey, "The Examination and Analysis of Residuals," *Technometrics*, 1963.
[2] R. Cook and S. Weisberg, *Residuals and Influence in Regression.* Chapman and Hall, 1982.
[3] J. Fox, *Applied Regression Analysis and Generalized Linear Models.* Sage, 2015.
[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* Springer, 2009.
[5] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. Autom. Control*, 1974.
[6] T. Snijders, "On Cross-Validation for Predictor Evaluation in Time Series." Springer, 1988.
[7] N. Chawla *et al.*, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Int. Res*, 2002.
[8] O. Ronneberger *et al.*, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 2015.