Machine Learning Final Report

Overview

This project for aims to apply Machine Learning to solve problems in two primary domains: regression and image analysis. Each domain consists of two distinct problems to be solved with Machine Learning techniques.

The goal of the project is twofold:

- To develop and evaluate regression models using synthetic data with outliers and noise, as well as implementing ARX (AutoRegressive with eXogenous input) models for dynamic systems.
- To build image classification and segmentation models for Mars crater detection using satellite imagery.

The project is done in Python, using its powerful libraries for machine learning applications, such as NumPy, Scikit-Learn, and other relevant tools.

The report will include a detailing of the methodologies used, model outputs, and statistical evaluations.

1 Part 1 - Regression with Synthetic Data

In this section, we address the first two regression problems outlined in the project description.

1.1 First Problem - Multiple Linear Regression with Outliers

1.1.1 Methods

Initially, we applied the Z-score method with a threshold of 3, which identified only 3 outliers. We then implemented the Interquartile Range (IQR) method, marking values below the 25th percentile and above the 75th percentile as outliers; this resulted in 45 outliers. Next, we used the **RANSAC** (RANdom SAmple Consensus) regressor from SciKit-learn (SKLearn) [1] to fit a model and isolate inliers and outliers. RANSAC removed 48 of the 200 data points (24%), which aligns closely with the estimated 25% data points containing human error, as specified in the project description. Figure 1 shows the identified outliers (in red) and inliers (in blue).

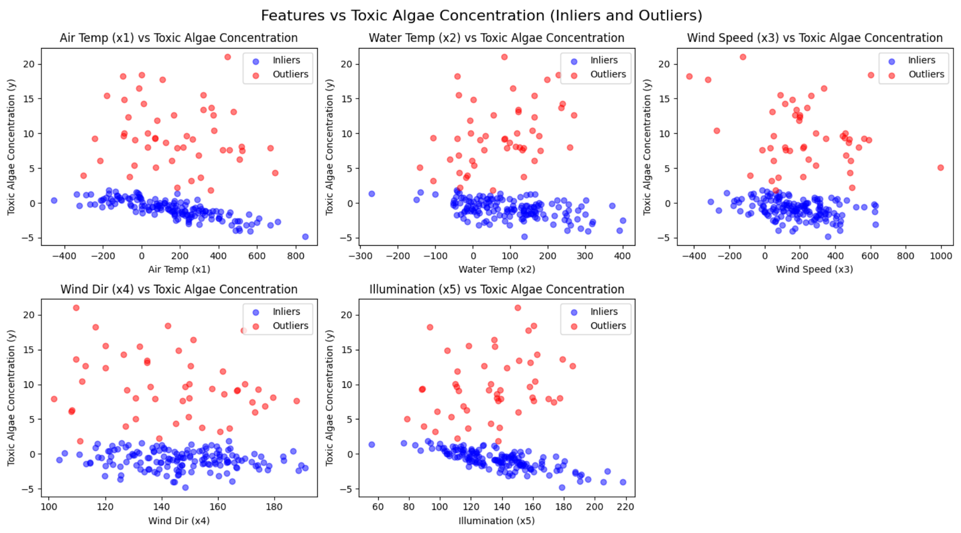

Figure 1: Plot of y as a function of x_i with inliers in blue and outliers in red.

After outlier removal, the cleaned dataset was split into a training set (80%) and a validation set (20%). We fitted a linear regression model on the training data, evaluating its performance on the validation set. To further optimize this model, we applied *regularization*.

A **Ridge regressor** was implemented using SKLearn, with 50 values for the regularization parameter α logarithmically spaced between 10^{-4} and 10^{4}. SKLearn's GridSearchCV method was used to identify the optimal α value for Ridge regression, providing additional robustness to the model against noise and overfitting [1].

Secondly a **Lasso regressor** with built in cross validation was fit to the cleaned training set, implemented using the method LassoCV from SKLearn [1].

The models were compared to each other using the Sum of Squared Errors SSE and Coefficient of Determination of the validation set, see Table 1; moreover the β_i were retrieved, see Table 2 which include also the alpha parameter.

Model	SSE	R^2
Linear Regression	0.0336	0.9992
Ridge Regression	0.0295	0.9993
Lasso Regression	0.0225	0.9995

Table 1: Results for the different linear regression models

Model	α	β_0	β_1	β_2	β_3	β_4	β_5
Linear Regression	0	1.8649	-4.379e-03	-3.286e-03	-1.884e-03	-2.627e-05	-1.015e-02
Ridge Regression	75.4312	1.6042	-4.491e-03	-3.825e-03	-1.883e-03	-2.465e-05	-7.755e-03
Lasso Regression	0.2548	0.7543	-4.850e-03	-5.551e-03	-1.876e-03	0.0000	0.0000

Table 2: Comparison of Alpha (α), Intercept (β_0), and Coefficients (β_1 to β_5) for Linear, Ridge, and Lasso Regression Models

Since Table 1 shows that the Lasso model had a lower SSE, this model was chosen to predict the test data which was sent in for submission. A visualization of the predicted and true values of the validation set is shown below in Figure 2.

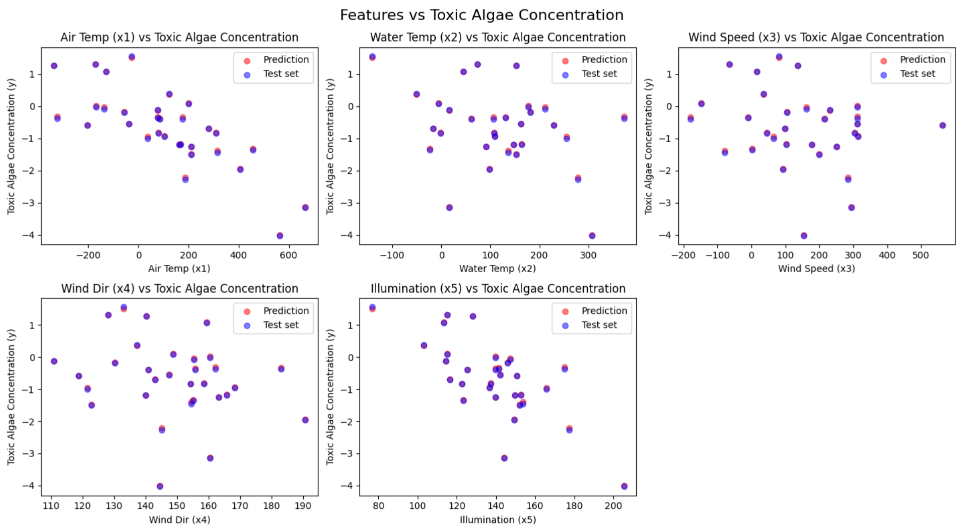

Figure 2: Plots of y as a function of x_i with real data in blue and predicted data in red.

1.1.2 Results and Discussion

Z-score ineffectiveness. The Z-score method identifies outliers by measuring how far data points deviate from the mean, assuming a normal (bell-curve) distribution. When data isn't normally distributed, meaning it has skewness, outliers, or other irregularities, the Z-score method fails to accurately detect outliers, as it misinterprets natural deviations as typical values.

Lasso evaluation. The coefficients (β) for each model show notable differences, particularly in the context of Lasso Regression, which results in some coefficients being exactly zero. This characteristic underscores Lasso's ability to perform feature selection by completely excluding less important predictors ($\beta_4 = 0$ and $\beta_5 = 0$). The regularization parameter α varies significantly among the models. The Linear Regression model has no regularization, while Ridge and Lasso introduce substantial values. The large value in Ridge indicates strong regularization, which helps reduce model complexity and prevent overfitting. On the other hand, Lasso's lower α suggests a more moderate approach, emphasizing variable selection while still penalizing complexity.

Submitted result evaluation. Upon submission, we received a Sum of Squared Errors (SSE) score of 0.1555 from the teaching team. This is consistent with our calculations on a subset of 31 data points, while the teaching team's calculation used the full set of 200 points. As expected, the SSE value from the teaching team is approximately seven times ours due to the difference in dataset size.

Miss outliers detection. The best-performing groups reported an SSE of around 0.03. According to feedback, achieving a score in this range requires identifying exactly 50 outliers, while our model found only 48. This discrepancy suggests that RANSAC might benefit from additional tuning to capture these two remaining outliers. Alternatively, RANSAC's inlier threshold could be adjusted or hard-coded to match the known 50 outliers, as suggested by its documentation. An additional technique to apply after RANSAC could be to calculate the squared error of each inlier, compute the average, and check if any points deviate significantly from this value.

1.2 Second Problem - The ARX Model

1.2.1 Methods

To address the second problem, code was developed to implement and test an **ARX (AutoRegressive with eXogenous inputs)** model for time series prediction. The ARX model structure can be represented as: $Y = X\theta$, the ARX model in matrix form where X (or ϕ) is the regressor matrix, θ the model parameters obtained after fitting the linear model, and Y the predicted values obtained using the `predict()` method.

The data was split into a training set comprising the first 80% of the series and a validation set of the remaining 20%. A brute-force search was used to identify the best combination of hyperparameters n, m, and d, testing all combinations within the range {1,...,9}. The ARX model was constructed using `LinearRegression` from SciKit-Learn [1], and while Ridge and Lasso regularization methods were also explored, no improvement in performance was observed with their use.

The fitting process was performed with custom `fit()` and `predict()` methods, specifically designed for the ARX model:

- `fit()` generates the regressor matrix X and target values Y, with Y containing the last $p = \max(n, d + m)$ values of y_{train} from the training set. The regressor matrix ϕ incorporates the AutoRegressive terms (past output values) and Exogenous terms (past input values). Then the linear regressor model is fit.

- `predict()` constructs the prediction matrix iteratively, beginning with the pre-built matrix phi_0 and updating each row with newly predicted values y_k, ensuring that the most recent predictions are utilized. In the equation below is shown the constructed X matrix and the vector of the Y predicted values (note that k in the first row corresponds to p).

$$\begin{bmatrix} y(k) \\ \cdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} y(k-1) & y(k-2) & \cdots & y(k-n) & u(k-d) & \cdots & u(k-d-m) \\ \cdots & & & & & & \\ y(N-2) & y(N-3) & \cdots & y(N-1-n) & u(N-1-d) & \cdots & u(N-1-d-m) \end{bmatrix} \cdot \Phi \quad (1)$$

The optimal hyperparameters (n, m, d) were chosen based on the configuration which had the lowest SSE on the validation set, while ensuring that the stability criterion is satisfied. The stability criterion ensures that a model's predictions remain bounded over time. For our discrete-time systems, this requires all roots (or poles) of the system's characteristic equation to be within the unit circle on the complex plane. If any root lies outside, the model will produce unstable, unbounded outputs. The combination with the lowest SSE for our code was $n = 9, m = 4, d = 1$, which obtained a SSE of 318.8. The results for the validation data can be seen in Figure 3, and the output data for the test data can be seen in Figure 4

1.2.2 Results and Discussion

Submitted result evaluation. In this problem we received a SSE of 284.8, which was a bad score compared to the best groups who received a SSE of around 4. The problem with our code is most likely how we implemented the predict function. The optimal values for the hyperparameters were around n = 9, m = 9 and d = 6, which is very different from what we concluded(n = 9, m = 4, d = 1). We tried appending the ϕ matrix with the true y values instead of our predicted ones and we received the values n = 8, m = 9, d = 6, which is a lot closer to the actual optimal values. Our SSE however was around 12,000 when we implemented this, which gives us the conclusion that something is wrong with our predict function.

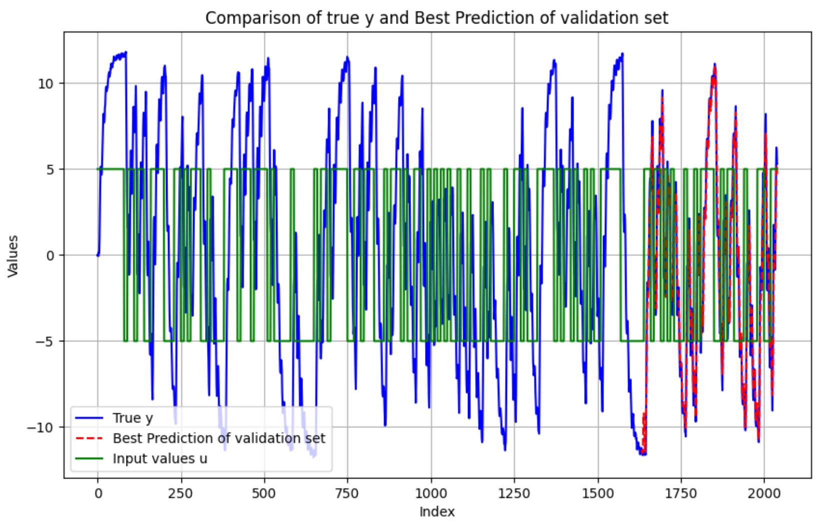

Figure 3: Plots of y as a function of time with real data in blue and predicted validation data in red.

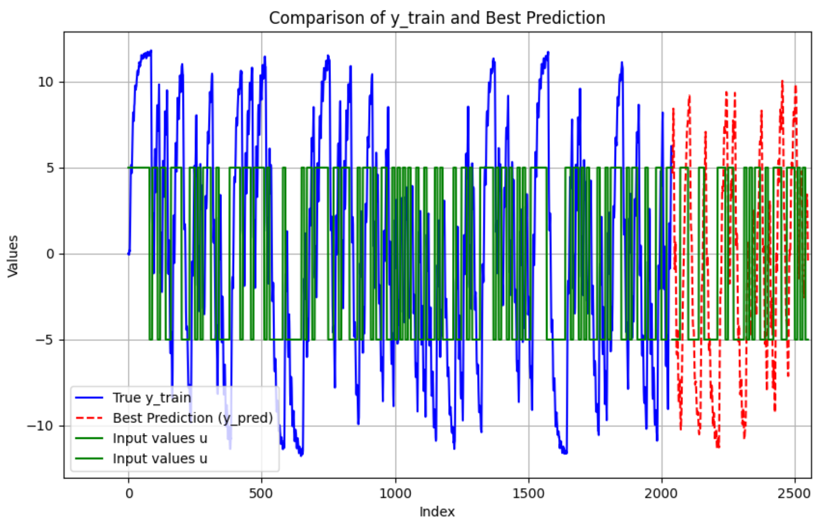

Figure 4: Plots of y as a function of time with real data in blue and predicted test data in red.

2 Part 2 - Image Analysis

In this section, we address the image binary classification and segmentation problems outlined in the project description.

2.1 First Problem - Image classification

2.1.1 Methods

Two different machine learning models were implemented and compared to achieve the task, an SVC (Support Vector Classification) from SKLearn [1], and a CNN (Convolutional Neural Net) using Keras [3].

For training, the data was split into a training set containing 80% of the training data, a test set containing 10% of the training data and a validation set containing the remaining 10%.

Two SVC models were trained with the training set, one using a linear kernel and one using a RBF (Radial Basis Functions) kernel. Both were balanced using class weights. They were evaluated using the test set.

To compare, a sequential CNN model was implemented using Keras. It starts with a Conv2D layer with 32 filters (3x3), using ReLU activation to detect local features, followed by MaxPooling (2x2) to reduce spatial dimensions. Another Conv2D layer with 64 filters (3x3) and MaxPooling (2x2) is added for deeper feature extraction. The output is flattened and passed through a Dense layer with 64 units using a ReLU activation function. The final Dense layer has 2 units with a softmax activation for binary classification, providing a probability of wether the pictures contain a crater or not. The model uses the Adam optimizer, binary cross-entropy loss, and incorporates early stopping which stops if the validation loss, computed with the validation set, has not decreased in the last 10 epochs to prevent overfitting. The model was evaluated using the test set. No balancing of the data was implemented. [3]

We also experimented with K-Nearest Neighbors (KNN), this classifier was initialized with 5 neighbors and trained on the flattened training dataset. Predictions were made on the test set, and the model's performance was evaluated using the F1 score and confusion matrix.

The results of the initial models are tabulated in table 3.

Model	F_1
SVC (Linear)	0.66
SVC (RBF)	0.81
CNN	0.90
KNN	0.0

Table 3: Results for the different binary classification models

Since the CNN obtained the best results further optimization of this was made to obtain the final model. First the number of filters in the convolution layers were doubled to see if the F_1 score improved, secondly an extra convolutional layer was added to the previous model after the first 2 with 128 filters (3x3). The results are tabulated in table 4.

Model	F_1
Base	0.903
Double filters	0.919
Extra layer	0.923

Table 4: Results for the CNN models with different architecture

Since the model with the extra convolution layer performed best, this was further optimized using data balancing techniques.

Firstly, the extra training dataset without labels was predicted by the model. All pictures where the model was more than 99.5% sure no crater was present were added to the training set and a new model with the same architecture was trained using the extended training data. 360 out of the total 904 pictures in the extra dataset were added. The new model achieved a F_1 score of 0.908, slightly decreasing the score. A subset of the added pictures is shown in figure 5

Three additional balancing techniques were implemented, SMOTE from Imbalanced Learn, RandomOverSampler from Imbalanced Learn [2], and class weights calculated using compute_class_weight from SKLearn [1]. The results are tabulated in table 5

Balancing technique	F_1
SMOTE	0.867
Random Over Sampler	0.826
Class weights	0.912

Table 5: Results for the CNN model with different balancing techniques

Since no improvements were found over the initial model the test data to be submitted for evaluation was trained using the CNN with an extra convolution layer without any balancing. A subset of the pictures with predicted labels is shown in figure 6

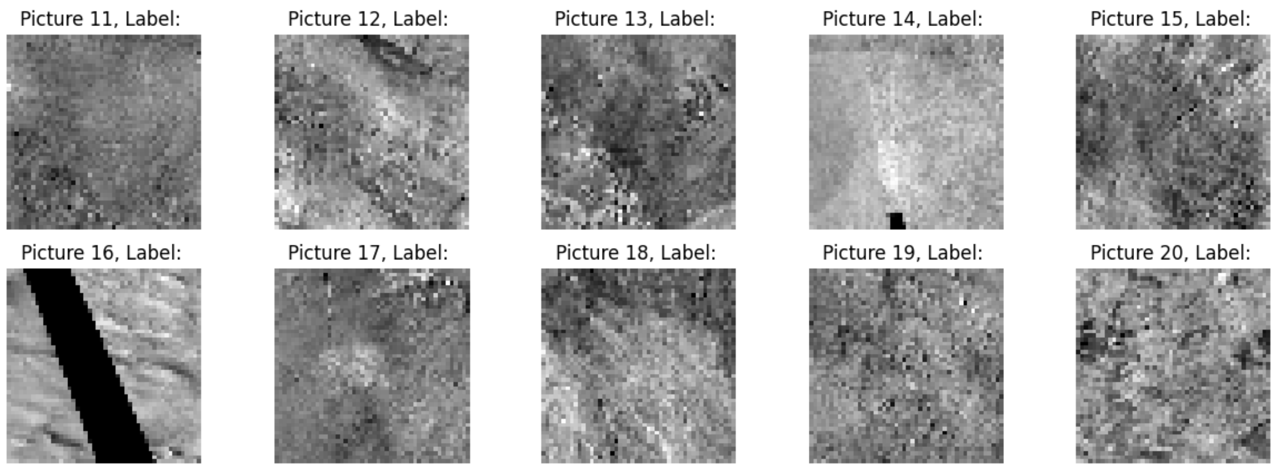

Figure 5: Subset of pictures predicted to have no craters by model

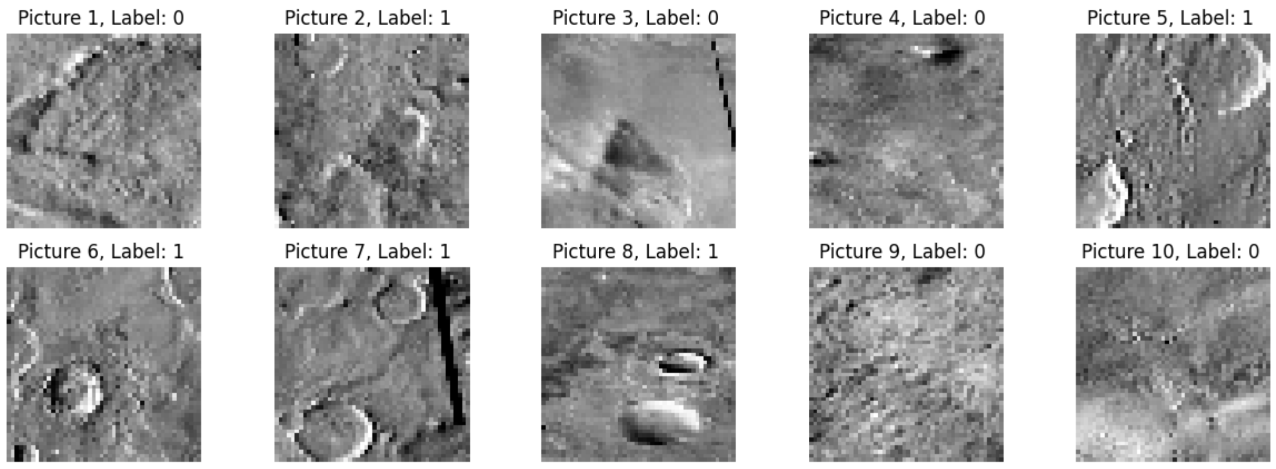

Figure 6: Subset of pictures predicted by final model with corresponding labels.

2.1.2 Results and Discussion

SVM Model Evaluation and Kernel Choice. Two Support Vector Machine (SVM) models were trained: one with a linear kernel and another with a Radial Basis Function (RBF) kernel. The linear SVM achieved an F1 score of 0.661, with a confusion matrix. The RBF kernel SVM, however, outperformed the linear SVM, with an F1 score of 0.814 (see Table 6).

Model	**Linear Kernel SVM**		**RBF Kernel SVM**	
	Predicted Crater	Predicted No Crater	Predicted Crater	Predicted No Crater
Actual Crater	43	42	52	33
Actual No Crater	77	116	38	155

Table 6: Confusion Matrices for Linear and RBF Kernel SVM Models

In SVMs, a kernel is a function that transforms data into a higher-dimensional space to make it more separable. The RBF (Radial Basis Function) kernel maps data in a nonlinear way, allowing SVM to handle complex patterns and relationships in data, unlike the linear kernel, which separates only linearly. Thus RBF can be more suitable for images that contains non-linear patterns.

Despite these improvements, further optimization could be: hyperparameter tuning, image preprocessing (eg. edge detections) or investigating in other kernels.

K-Nearest Neighbors (KNN) Performance. KNN relies on Euclidean distance calculations to determine similarity. Images as data points have different pixel features, therefore KNN resulted useless and the confusion matrix revealed lots misclassifications.

Convolutional Neural Network (CNN) Evaluation. As discussed, our attempt to handle dataset imbalance did not yield performance improvements; however, identifying a suitable technique for this challenge, such data augmentation, would likely enhance results. In hindsight, the model variant with doubled nodes but no additional layer may have been more efficient. With comparable accuracy to the final architecture, this simpler model could reduce overfitting risk while maintaining performance.

In conclusion, while the SVM with an RBF kernel achieved solid results, CNNs remain promising for future work. Their capacity to capture spatial hierarchies in pixel data could lead to substantial gains in crater detection accuracy.

2.2 Second Problem - Image Segmentation

2.2.1 Methods

The final task in this project addresses a segmentation problem, which can be managed through a pixel classification problem on a supervised learning dataset (format-a data) or a structured prediction task aimed at image mask creation (format-b data).

Regarding the pixel classification, we implemented a **Random Forest classifier**[1] with class weighting to address the inherent class imbalance in the data. To improve the model performance, we iteratively **tuned the hyperparameters**, selecting values that could improve accuracy while ensuring the model could compile within a reasonable time and memory constraint of 13 GB of RAM. The key hyperparameters tuned were:

- **n_estimators:** The number of trees in the forest, which impacts both model performance and computational load.
- **max_depth:** The maximum depth of each tree, allowing the model to capture more intricate data patterns while avoiding overfitting.
- **min_samples_split and min_samples_leaf:** Parameters that control tree branching and leaf creation, tuned to improve model generalization.

For the structured prediction task using **format-b data** aimed at image mask creation, we developed a **CNN U-Net**[3] model. This architecture is designed for segmentation tasks and is suited for the pixel-level classification required for image mask creation. Its structure is characterized by an encoder-decoder architecture that effectively captures both spatial and contextual information. The encoder path down-samples the input image through successive convolutional and max-pooling layers, progressively reducing spatial dimensions while increasing feature depth. This project's encoder begins with a Conv2D layer with 64 filters (3x3) and ReLU activation, detecting local features, followed by a MaxPooling2D layer (2x2) to halve spatial dimensions. Additional Conv2D layers with 128 and 256 filters (3x3) follow, each with max-pooling steps, allowing deeper feature extraction as spatial resolution decreases. The decoder path then up-samples the feature maps through transposed convolutions, gradually restoring spatial dimensions to produce a segmentation mask. This process starts with an UpSampling2D layer, paired with Conv2D layers with 128 and 64 filters (3x3), allowing the network to gradually recover spatial resolution. The final output layer uses a Conv2D layer with a single filter (1x1) and a sigmoid activation function, where a binary mask where each pixel's value represents the probability of class membership. This U-Net structure enables pixel-level segmentation, making it highly effective for applications like ours. A deeper model was tested but did not outperform the current architecture.

For training, the data was split into a training set containing 80% of the training data, a test set containing 10% of the training data and a validation set containing the remaining 10%. Early stopping was used to end the training after the validation loss stops improving for 7 epochs, saving the model with the lowest validation loss for the evaluation. To address the class imbalance between black and white pixels in the masks, we initially trained the model with weighted class adjustments, achieving a balanced accuracy of 0.7679. To further manage the imbalance problem, we applied data augmentation. Specifically, we selected images in which the ground truth mask contained at least 50% white pixels, as calculated by the binary mask mean ratio. Out of the 547 images in the training dataset, 54 met this criterion, containing more white pixels than black pixels. These images were augmented through rotations and flips along both axes, resulting in 216 additional samples to supplement the training set. This process increased the proportion of images with a higher white pixel ratio from 10% to 30% of the total training data.

2.2.2 Results

Random Forest Performance. Our baseline RF model, with minimal tuning and balanced class weights, achieved a balanced accuracy of 0.6579. After tuning, balanced accuracy values for each configuration are summarized in

Table 7.

Configuration	n_estimators	max_depth	min_samples_split	min_samples_leaf	Balanced Accuracy
Initial	20	None	2	1	0.6579
Tuning 1	100	10	5	2	0.6797
Tuning 2	100	20	5	2	0.6899

Table 7: Random Forest Hyperparameter Tuning and Balanced Accuracy Results

Hyperparameter tuning improved the model accuracy, with the highest balanced accuracy of 0.6899 achieved using 100 estimators and a maximum depth of 20. This suggests that increasing the number and depth of trees enhanced the model's segmentation accuracy. Memory and time constraints helped determine the practical limits for these parameters.

U-Net Performance. In the structured prediction task, our CNN U-Net model achieved an initial balanced accuracy of 0.7679 with class weighting alone. After augmenting the dataset to address pixel imbalance, balanced accuracy improved to 0.7950. The data augmentation effectively increased the representation of images with a higher ratio of white pixels, enhancing the model's ability to accurately segment features of interest in the test dataset. This final model, trained on the augmented dataset, yielded the highest balanced accuracy and was thus selected for generating the Ytest predictions for the Xtest_b dataset. The other deeper model with more convolutional layers and pooling layers did not perform as well as the simpler one

Figure 8 shows the comparison between the image mask created by the U-Net CNN with almost 80% fo balanced accuracy and, the ground truth mask related the gray scale picture with craters.

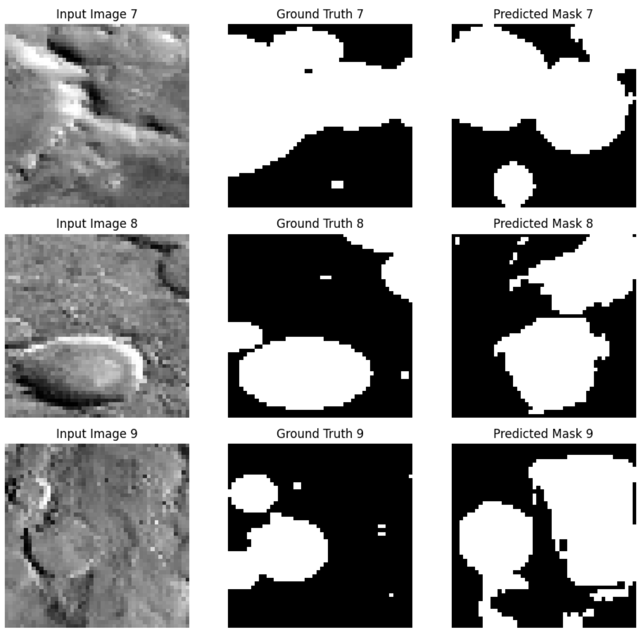

Figure 7: Crater image (left), ground truth masks (center), CNN predictions (right)

Figure 8 shows three samples of the prediction made by out model on the given set Xtest_b.

Submitted result evaluation. The professor's evaluation of our results yielded a balanced accuracy of 0.7981, closely aligning with our initial expectations, which we had estimated at approximately 0.795. Comparatively, the top-performing group achieved a balanced accuracy of 0.8434, a margin that suggests a more refined tuning of our model could potentially close the gap. The results indicate that our approach was well-calibrated but could benefit from further optimization to reach peak accuracy.

3 Conclusion

In conclusion, this project successfully applied machine learning techniques to solve challenging problems in regression and image analysis. In the regression tasks, methods for outlier detection and regularization, including Ridge

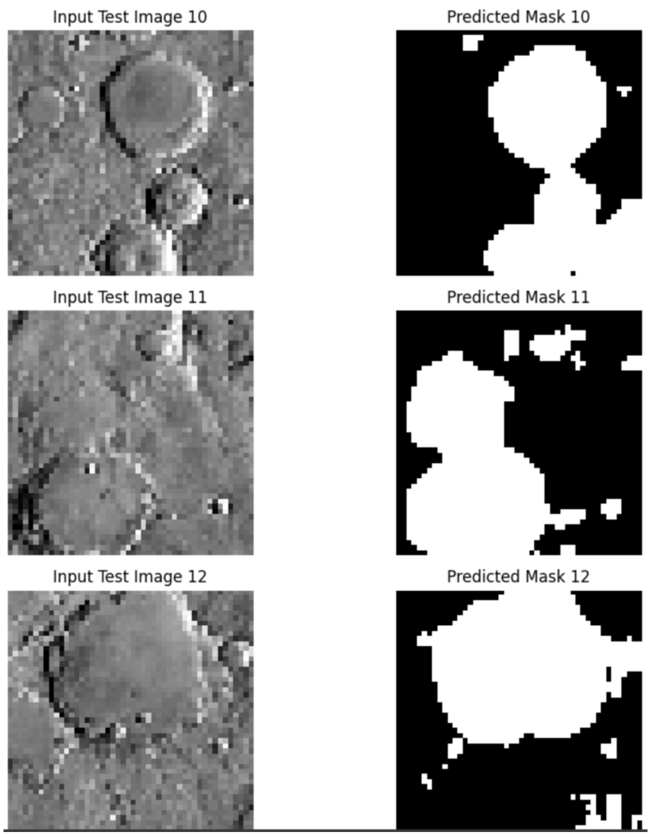

Figure 8: Image to mask (left), CNN mask (right)

and Lasso, were implemented to improve model robustness against noise and errors in synthetic data. Though the ARX model for time series prediction performed sub-optimally, insights into potential areas for refinement, especially within prediction function were identified, which could guide future improvements.

The image analysis tasks demonstrated the efficacy of machine learning for visual data. In the crater classification problem, the CNN model outperformed SVMs and KNNs, emphasizing the advantage of deep learning for complex image patterns. Further optimization efforts, including balancing techniques, underscored the importance of data preparation and model architecture adjustments in achieving higher accuracy. For the segmentation task, the U-Net CNN model achieved notable results in crater mask creation, with data augmentation proving essential for mitigating class imbalance.

Overall, this project highlighted the importance of methodical model evaluation, tuning, and refinement in handling both regression and image-based machine learning problems. Although some challenges persisted, especially with data imbalance and ARX prediction accuracy, the project's insights form a solid foundation for further exploration in both structured prediction and complex image classification tasks.

References

[1] scikit-learn developers. *API Reference*. 2024. URL: `https://scikit-learn.org/stable/api/index.html` (visited on 10/25/2024).

[2] The imbalanced-learn developers. *API Reference*. 2024. URL: `https://imbalanced-learn.org/stable/references/index.html` (visited on 10/24/2024).

[3] Keras. *Keras 3 API documentation*. 2024. URL: `https://keras.io/api/` (visited on 10/25/2024).