Machine Learning Project - Report

António Maria de Almeida Pinto Alves Jotta (99893), António Vasco Morais de Carvalho (102643)
Group: 37, 2024/2025 – 1st Semester, P1
Emails: antonio.jotta@tecnico.ulisboa.pt, antonio.v.morais.c@tecnico.ulisboa.pt

1 Part 1 - Regression with Synthetic Data

1.1 First Problem - Multiple Linear Regression with Outliers

In this task, it is given a dataset with independent and dependent variables of $n = 200$ samples. The objective is to compute the predictions for a given test set and save the resulting predictions of the dependent variable that minimize the Sum of Squared Errors (SSE).

1.1.1 Outlier Removal

To address the presence of outliers and clean the dataset, the Random Sample Consensus algorithm [1] (RANSAC) was used to identify and remove outliers from the dataset. The algorithm is depicted below:

Algorithm 1 RANSAC Algorithm

1: **for** $m = 1$ to M **do**
2: Select N data items of the dataset as random.
3: Estimate parameter $\overrightarrow{x}$ (model parameters).
4: Find a subset that fits the model (inliers) with parameter vector $\overrightarrow{x}$ within a user given threshold (call this P).
5: **if** P is big enough **then**
6: **return** subset of inliers
7: **end if**
8: **end for**

The algorithm is implemented by the `RANSACRegressor` class provided by the `sklearn` Python library. The parameters M, N, and P are set to default values by the class.

After the algorithm is run on the original dataset, 48 out of the 200 original samples are considered outliers and removed (about 25%), resulting in a cleaned dataset of 152 samples considered inliers - Figure 1.

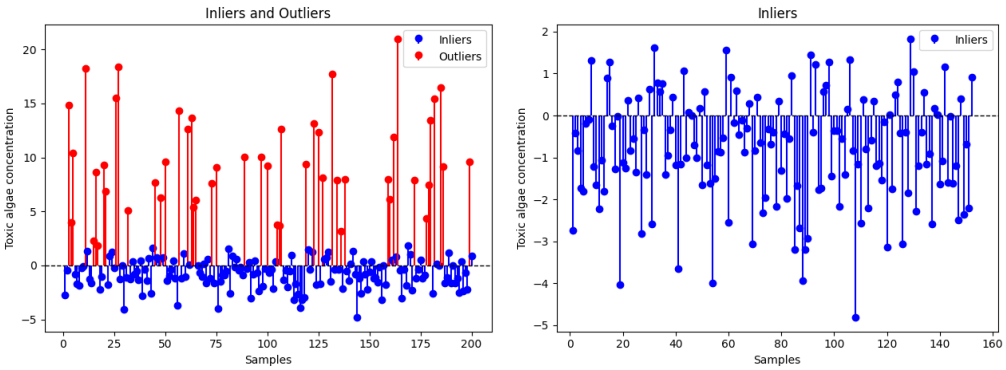

Figure 1: Outlier removal visualization.

1.1.2 K-Fold Cross-Validation

Due to the small size of the dataset, the K-Fold Cross-Validation technique was employed to make the models robust to overfitting. The dataset is split into K subsets, and on each iteration, one subset is chosen as the validation set while the remaining subsets are used for training.

The group decided to use $K = 8$. In each iteration, the dataset is split into (considering $n = 152$) $\frac{n(K-1)}{K} = 133$ samples for training and $\frac{n}{K} = 19$ samples for validation. Several values of K were tested, and the chosen value ensures that each subset has the same size while providing a reasonable number of samples for validation.

1.1.3 Ridge Regression

Ridge regression modifies the cost function of Linear regression by adding a penalty term proportional to the square of the regression coefficients. Consequently, it minimizes the following cost function:

$$J(\theta, \beta) = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \sum_{i=1}^{m} \beta_i^2. \tag{1}$$

Here, λ is a regularization parameter that controls the strength of the penalty, and β_i represents the model coefficients. As λ increases, the model coefficients shrink; conversely, a lower value of λ gives greater importance to minimizing the SSE.

1.1.4 Lasso Regression

Now for Lasso regression, the penalty is proportional to the absolute values of the regression coefficients, minimizing the following cost function:

$$J(\theta, \beta) = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \sum_{i=1}^{m} |\beta_i|.$$

Because Lasso uses the ℓ_1-norm (instead of the ℓ_2-norm used in (1)), it imposes a greater penalty on the existence of low-value coefficients, shrinking some coefficients to exactly zero. This makes Lasso effective for relevant feature selection. During model testing, this occurred with the x_4 and x_5 features as $\beta_{4,5} = 0$.

1.1.5 Elastic Net Regression

Elastic Net combines both Lasso and Ridge regressions, introducing a balance between both techniques. The cost function is depicted below:

$$J(\theta, \beta) = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \left(\frac{1 - \ell_1\text{-ratio}}{2} \sum_{i=1}^{m} \beta_i^2 + \ell_1\text{-ratio} \sum_{i=1}^{m} |\beta_i| \right).$$

The parameter λ continues to control the penalty based on the coefficients. Elastic Net now includes the parameter ℓ_1-ratio, which determines the relative contribution of Lasso and Ridge

penalties in the overall cost function. An ℓ_1-ratio $= 1$ corresponds to Lasso regression, emphasizing the absolute penalty, while an ℓ_1-ratio $= 0$ corresponds to Ridge regression, focusing solely on the squared penalty. Values between 0 and 1 blend the two techniques.

1.1.6 Results and Discussion

Model	SSE	R^2	λ	ℓ_1-ratio
Ridge	1.13832	0.99392	10	-
Lasso	1.15279	0.99394	0.1	-
Elastic Net	2.95324	0.99394	10	1

Table 1: Model analysis (chosen model highlighted).

In Table 1, the performance of the tested models is presented. The hyperparameters λ and ℓ_1-ratio were tuned across a broad range of values (logarithmically from 0.00001 to 10) using K-Fold Cross-Validation, with the default scoring metric R^2. To select the best model, the SSE metric was used to ensure that the model was well-trained on the inlier data. Based on this analysis, the group selected the **Ridge** regression model as the final choice.

This model secured 49$^{\text{th}}$ place on the leaderboard, with a score very close to the groups ranked above. The small difference in scores may be attributed to the presence of two outliers that were not removed by the group, whereas some of the groups above removed exactly 50.

1.2 Second Problem - The ARX model

This section discusses the prediction of an output signal of a discrete input-output dynamical system and the tuning of parameters (n, m, d) to minimize the SSE, using Linear, Ridge, and Lasso regressions.

1.2.1 Optimal Parameters

The dataset provided was split into two subsets: training (80%) and validation (20%) — the validation set being used to evaluate the models in this task.

Given the constraints $n, m, d \in \{1, 2, \ldots, 10\}$, the group determined the SSE value for every combination of these parameters. The optimal combination of parameters found is illustrated in Figure 2, and is given by

$$n = 9,\ m = 9,\ d = 6.$$

During the tuning process, the regularization parameters for Ridge and Lasso regressions were also optimized, with values tested on a logarithmic scale from 0.00001 to 10. The optimal values are depicted in Table 2.

Furthermore, Figure 3 demonstrates how the SSE metric depends on each parameter individually. It is noticeable that increasing n and m generally decreases the SSE, whereas increasing d has the opposite effect. This suggests that looking further back in time improves predictions, up to a certain point.

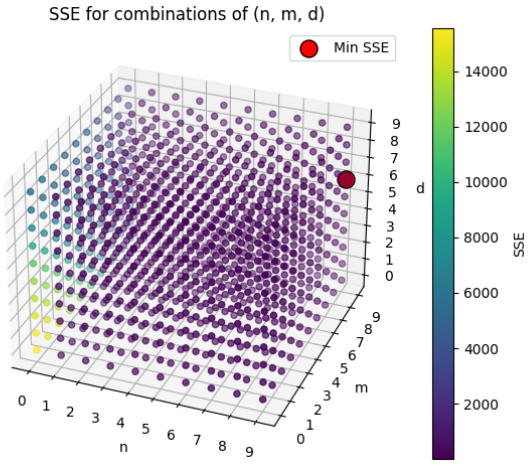

Figure 2: SSE over the model parameters (n, m, d).

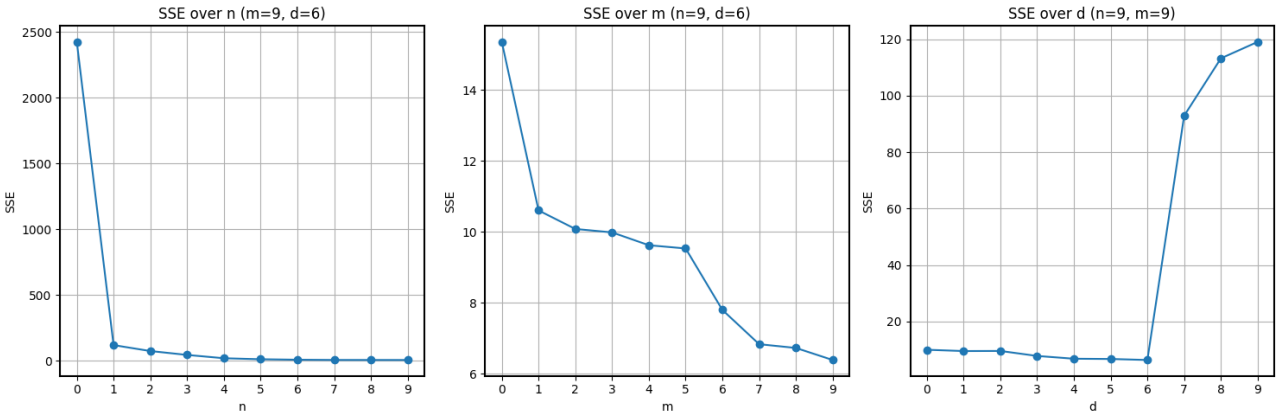

Figure 3: SSE over, individually, the model parameters (n, m, d).

1.2.2 Results and Discussion

Model	SSE	R^2	λ
Linear	6.3936	0.9995	-
Ridge	6.3936	0.9995	1×10^{-5}
Lasso	6.3744	0.9995	1×10^{-4}

Table 2: Model analysis (chosen model highlighted).

Once again, the model chosen for delivery is highlighted. The **Lasso** regression, with a regularization parameter of $\lambda = 1 \times 10^{-4}$ and the lowest SSE metric value obtained, placed the group in 48^{th} position on the leaderboard. One aspect the group could have explored further was testing higher values for the (n, m, d) parameters, as it was mentioned by the faculty that values greater than 10 could be considered. The group tested these values after submission and

indeed achieved better results (supporting the previous observation about looking further back in time), which could have resulted in a higher position on the leaderboard.

2 Part 2 - Image Analysis

2.1 First Problem - Image Classification

In this image classification task, the goal is to predict labels for a dataset of **Crater** and **No Crater** images, exploring techniques for balancing the dataset, using an additional provided unlabeled dataset, and applying models for image classification.

2.1.1 Dataset Imbalance

Given that the dataset was imbalanced, the group first identified the minority class. It was determined that the **No crater** class ($1006 \rightarrow 36.15\%$) was the minority compared to the **Crater** class ($1777 \rightarrow 63.85\%$). Addressing this imbalance is crucial, as training a model on imbalanced data could lead to bias towards the **Crater** class.

To balance the dataset, three oversampling techniques were explored: `SMOTE` [2]: generates synthetic data points by identifying k nearest neighbors of a minority class sample and placing new points between the selected sample and its neighbors; `RandomOverSampler` (ROS): randomly duplicates minority class samples until the class sizes are equal; `ImageDataGenerator`: augments the minority class by applying transformations to samples, such as rotation, pixel brightness adjustments, and horizontal/vertical flipping.

After addressing the imbalance, the dataset was split into three subsets: training (80%), validation (10%), and testing (10%). Care was taken to ensure that the training subset was balanced, while the validation and testing subsets remained imbalanced to allow for realistic performance evaluation. The metrics presented throughout this task were obtained from the test subset and the plots presented pertain to the best results of the model under discussion.

2.1.2 Extra Dataset

An additional unlabeled dataset was provided, and the faculty encouraged students to explore ways of using it. The group identified a potential use for this dataset and incorporated it into the training subset, while consistently maintaining balance between the classes. For the models tested, the group initially trained the models on the labeled dataset and then used them to make predictions on the extra dataset. To avoid introducing noisy data into the original training subset, the group selected samples based on the confidence level of the predicted labels. A threshold of 0.9 was chosen, meaning that a sample was included if the predicted probability for class 1 exceeded 0.9 (and vice-versa for class 0).

2.1.3 Support Vector Machine (SVM)

A classifier used was a SVM which classifies data by finding a hyperplane that best separates different classes, maximizing the margin between the nearest points of each class. To implement it, the class `SVC` from `sklearn` was used, which was paired with `GridSearchCV` for hyper-parameter tuning. Hyper-parameters tested include the kernel ('linear', 'rbf', 'sigmoid' and

poly'), γ, C (logarithmically from 0.00001 to 10) and degree (for 'poly' kernel - $\{1, 2, \ldots, 9\}$). With the scoring metric being F1-Score, the best parameters found were {'kernel': 'rbf', C: 10, γ: 'scale'} (with a balancing method already implemented).

2.1.4 Concurrent Neural Network (CNN)

The other classifier used was a CNN, which is known for its good performance with image data. The architecture used is as follows: [one convolutionnal layer with 32 filters and a 3×3 kernel size (ReLU as the activation function) and one max pooling layer with a pool size of 3×3 and stride of 3] $\times 2$; one convolutionnal layer with 32 filters and a 2×2 kernel size (ReLU as the activation function) and one max pooling layer with a pool size of 2×2 and stride of 2; one flatten layer; [one fully connected layer with η neurons (ReLU as the activation function) and one dropout layer with a dropout probability of 0.5] $\times 3$, for $\eta = 1024, 256, 56$; one output layer with 1 neuron (sigmoid as the activation function).

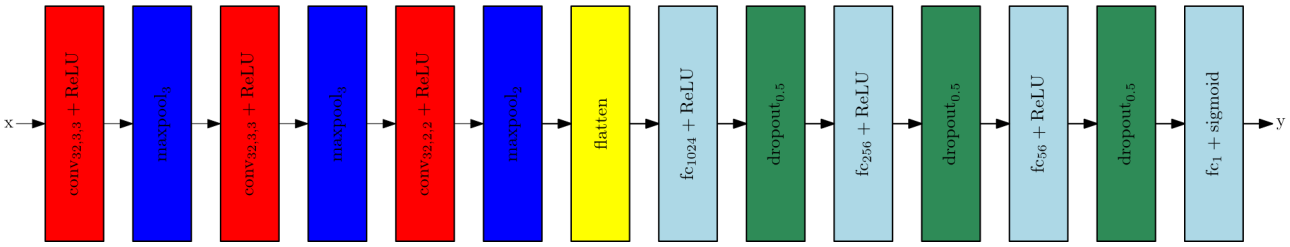

Figure 4: CNN architecture used.

The optimizer used was `Adam` with a fine-tuned learning rate of 0.0001, aiming to minimize the binary cross-entropy loss function. To prevent overfitting, a callback `EarlyStopping` was employed to monitor the validation loss. The callback was configured with a patience of 15 epochs, meaning that if the validation loss started increasing after a period of decrease, the callback would trigger, and the CNN's weights would be restored to those from the epoch with the lowest validation loss. The CNN was trained for a maximum of 100 epochs. In Figure 5, training stopped at the 80$^{\text{th}}$ epoch, preserving the best weights found at the 65$^{\text{th}}$ epoch, when the validation loss began to increase, triggering the callback with the specified patience.

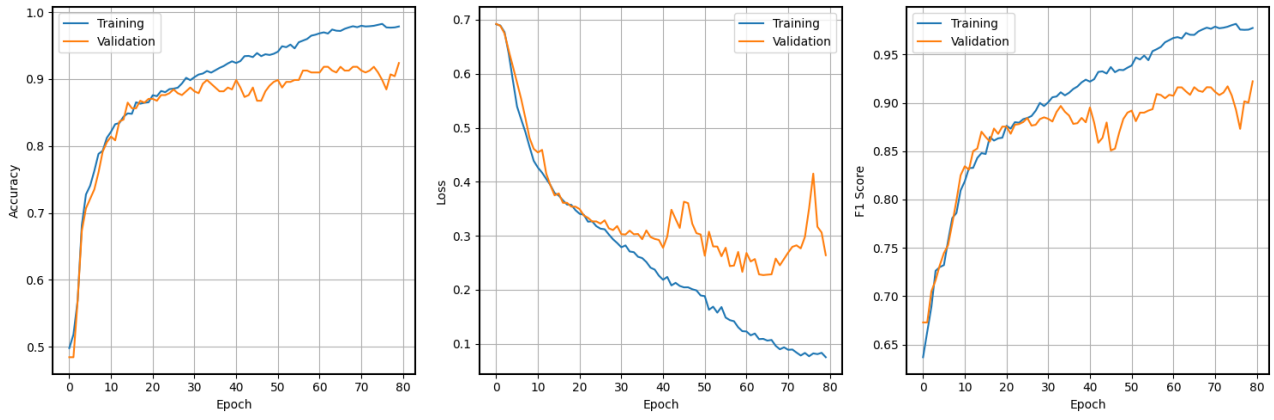

Figure 5: Train and validation accuracy, loss and F1-Score for CNN + ROS + Extra Dataset.

2.1.5 Results and Discussion

Method	Model	
	SVM	**CNN**
Imbalanced	0.6495	0.8280
`SMOTE`	0.8056	0.8988
`ImageDataGenerator`	0.7465	0.8464
`RandomOverSampler`	0.8678	0.9185
`RandomOverSampler` + **Extra Dataset**	0.8678	0.9213

Table 3: F1-Score for dataset manipulation methods and models (chosen model highlighted).

The extra dataset was only tested with ROS, as this technique achieved the best results for both models. The model with the highest F1-Score is highlighted in Table 3, **CNN + ROS + Extra Dataset**, making it the chosen model for delivery. However, this resulted in an unexpected position near the end of the leaderboard, despite the group testing different oversampling techniques, utilizing the extra dataset, fine-tuning all hyperparameters, and exploring various architectures for the CNN. Initially, it was hypothesized that ROS might have contributed to this outcome by introducing potential overlap between training and test samples, leading to misleading results. However, this was later ruled out, as ROS functiones similarly to a class weights method, which ranked among the leaderboard's top approaches. It was ultimately concluded that the chosen CNN architecture was the primary factor in the model's performance.

2.2 Second Problem - Image Segmentation

In this second binary classification task, each pixel in a 48×48 image must be classified. Two data formats are provided: **Format a**, where each pixel is represented by the 7×7 neighborhood of surrounding pixels as features; **Format b**, where complete input images along with their corresponding segmentation masks are provided. Different classification and balancing methods were applied for each data format.

2.2.1 Dataset Imbalance

Once again, the datasets provided are imbalanced, hence the necessity of using balancing methods. In both data formats, background pixels represent the majority class (**format a:** 649624 $\rightarrow$ 67.32%, **format b:** 896429 $\rightarrow$ 71.13%) compared to the crater pixel class (**format a:** 315284 $\rightarrow$ 32.68%, **format b:** 363859 $\rightarrow$ 28.87%). For **dataset a**, `SMOTE` was employed, while for **dataset b**, class weights were utilized. Additionally, for **format b**, overall data augmentation (ODA) was performed by executing rotations of 90°, 180°, and 270° on each image.

Following this, the dataset was split into three subsets: training (80%), validation (10%), and testing (10%). Care was taken to ensure that the training subset remained balanced. The balanced accuracy (BAcc) metric presented throughout this task was obtained from the test subset and the plots presented pertain to the best results of the model under discussion.

2.2.2 K-Nearest Neighbors (KNN)

The `KNeighborsClassifier` was tested, with **format a**, for a range of $k = \{1, 2, \ldots, 20\}$ neighbors. It classifies a new point based on the majority class of its K nearest neighbors in the feature space, utilizing the Euclidean distance. The classifier calculates the distances to all training points, selects the K closest neighbors, and assigns the majority class to the new point. The best parameter found was $k = 2$, and the BAcc over k is depicted in Figure 6.

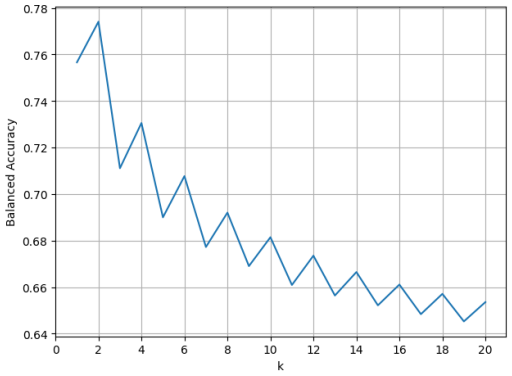

Figure 6: BAcc over k neighbor values.

2.2.3 Multilayer Perceptron (MLP)

In this task we decided to use a MLP, which is a type of fully connected neural network that is commonly used for classification tasks, and the main reason to use it is that with **format a** the images are much smaller $48 \times 48 \rightarrow 7 \times 7$. The architecture used is as follows: [one fully connected layer with η neurons (ReLU as the activation function)] $\times 2$, for $\eta = 128, 64$; one output layer with 1 neuron (sigmoid as the activation function).

As in 2.1.4, the optimizer `Adam` was used with a learning rate of 0.001 (same loss function), and the callback `EarlyStopping` was defined with a patience of 5 epochs. Due to the notorious computational cost of the **format a** dataset using a CNN, training was performed up to a maximum of 50 epochs. In Figure 7 the callback did not trigger, stopping training at the 50^{th} epoch. Note that the plot below refers to the accuracy metric and not the BAcc. The BAcc was calculated only on the test subset to elaborate Table 4.

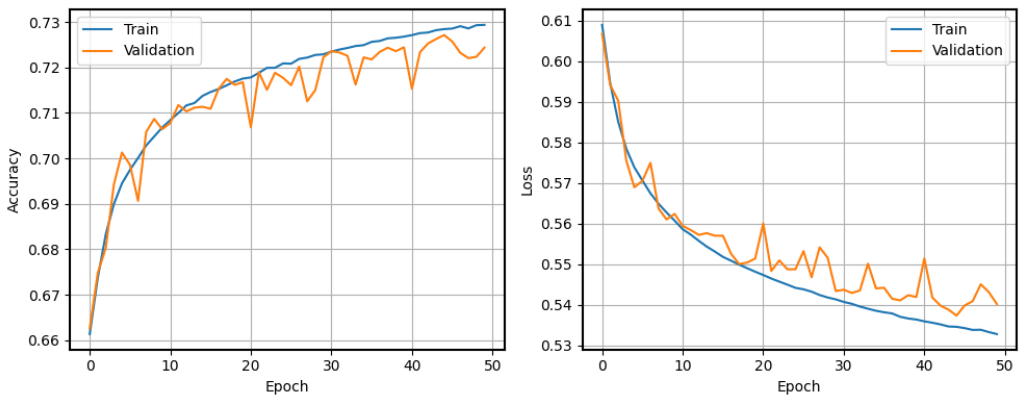

Figure 7: Train and validation accuracy and loss for MLP + `SMOTE`.

2.2.4 Random Forest (RF)

The `RandomForestClassifier` was employed to implement the RF method, which is an ensemble learning technique for classification. This method constructs multiple decision trees during training and predicts the class that receives the majority vote among the trees. This aggregation reduces the risk of overfitting and enhances accuracy.

The model was tested on **format a** with several numbers of estimators, i.e. the number of trees to be used in the forest, $\tau = \{25, 50, 75, \ldots, 300\}$. The best number of estimators identified was $\tau = 100$, using `SMOTE`; however, this model was the least effective among those tested.

2.2.5 U-Net

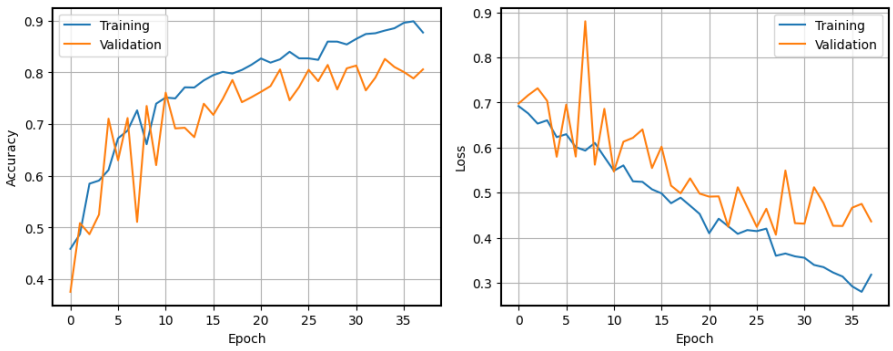

Figure 8: Train and validation accuracy and loss for U-Net + Class Weights + ODA.

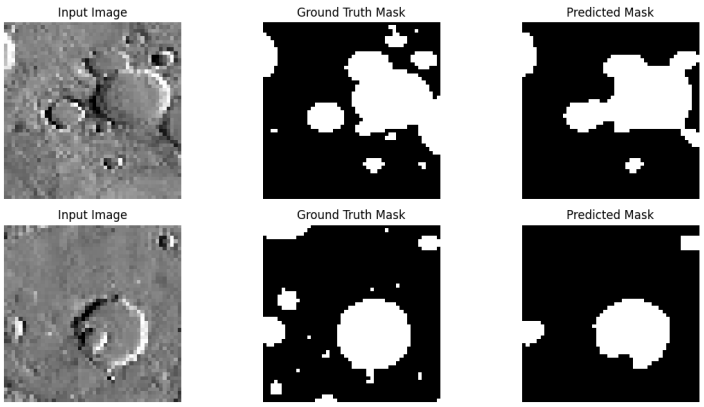

Figure 9: Examples of input images and ground truth/predicted masks comparison.

The U-Net architecture is a CNN designed for image segmentation. The architecture used (derived from the original implementation [3]) is as follows: *Contracting Path* - [Two convolutional layers with η filters and a 3×3 kernel size (ReLU as the activation function) and one max pooling layer with a pool size of 2×2] $\times 4$, for $\eta = 32, 64, 128, 256$; *Expanding Path* - [One upsampling layer followed by one convolutional layer with η filters and a 3×3 kernel size (ReLU as the activation function) and one concatenation with skip connection from the contracting path, followed by two convolutional layers with η filters and a 3×3 kernel size (ReLU as the

activation function)] $\times 3$, for $\eta = 128, 64, 32$; *Output Layer* - One convolutional layer with 1 filter and a 1×1 kernel size (sigmoid as the activation function).

The classifier was used with the **format b** data, to which ODA was applied, increasing the number of images available ($547 \rightarrow 2188$). Additionally, the class weights method was applied to balance the dataset. Again, the `Adam` optimizer was employed with a learning rate of 0.001, using the same loss function, and the `EarlyStopping` callback was set with a patience of 10 epochs. Training was conducted for a maximum of 100 epochs. As shown in Figure 8, the model stopped training at the 38^{th} epoch, restoring the best weights found at the 28^{th} epoch. Examples of predictions from this model are displayed in Figure 9.

2.2.6 Results and Discussion

Method	Model			
	Data Format a			Data Format b
	KNN	MLP	RF	U-Net
Imbalanced	0.6529	0.6681	0.6541	0.7883
`SMOTE`	0.7741	0.7262	0.6912	-
Class Weights + ODA	-	-	-	0.8105

Table 4: BAcc score for dataset manipulation methods and models, for both data formats (chosen model highlighted).

The highest BAcc was achieved using the **format b** data with **U-Net + Class Weights + ODA**, making this model the final selection for delivery. This model got the group 28^{th} place in the leaderboard. Further improvements may be possible by experimenting with different architectural configurations of the U-Net, given its substantial flexibility and complexity. Additionally, further tuning and testing could have also enhanced results.

3 Conclusion

All tasks were completed successfully, and the performance of the various models in the leaderboards generally met expectations, with the exception of the model discussed in 1.1.6. This project emphasized the importance of thorough dataset inspection, informed selection of models, appropriate data processing techniques and effective model tuning.

References

[1] Martin A. Fischler and et al. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 1981.

[2] Kevin W. Bowyer and et al. SMOTE: synthetic minority over-sampling technique. *CoRR*, 2011.

[3] Olaf Ronneberger and et al. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, 2015.