AN2DL - Second Homework Report
LOS PINGUINOS

Zeno Peracchione, Daniele Vozza, Francesco Tomasi, Lorenzo Galatea

zenopera, danielevozza, francescotomasi, lorenzogalatea

250643, 252574, 252533, 271594

December 14, 2024

1 Introduction

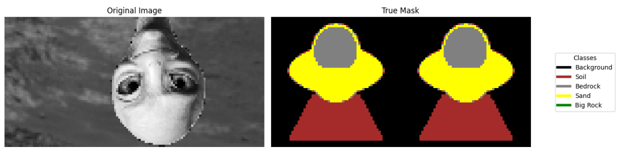
This report addresses a *semantic segmentation* task involving 64x128 grayscale images of Mars terrain. Each pixel in these images belongs to one of five terrain classes:

- **0**: Background
- **1**: Soil
- **2**: Bedrock
- **3**: Sand
- **4**: Big Rock

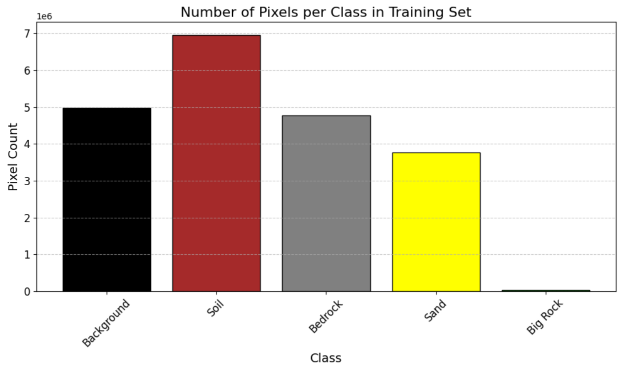
The goal is to accurately assign the correct class label to each pixel, enabling detailed terrain classification.

2 Problem Analysis

In this study, we divided the dataset into three subsets: **training**, **validation**, and **test** sets, ensuring a proper split to evaluate the model's performance effectively. Upon analyzing the data, we observed the presence of *Aliens outliers* that might affect the overall model performance. One such outlier is shown in the following image:



Furthermore, we noticed that the distribution of class labels is significantly imbalanced. In particular, the class 4 *"Big Rock"* is underrepresented, with a notably low number of labeled pixels. To visualize this imbalance, we provide the histogram below:



The imbalance in class distribution presents a challenge for model training and may require special techniques, such as class weighting or data augmentation, to mitigate its impact on the model's performance.

3 First Model

In the first model, we remove outliers in the dataset that could potentially affect the training process. To address the semantic segmentation task, we wrote the code based on the concepts covered during lectures and the exercises, particularly the *"Semantic Segmentation with U-Net.ipynb"* notebook from the laboratory sessions.

We implemented the `get_unet_model` function with the following structure:

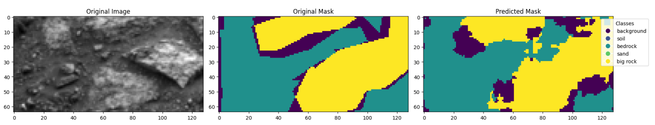
- **2 downsampling paths**: to extract features at different scales.
- **1 basic bottleneck**: to capture compressed feature representations.
- **2 upsampling paths**: to reconstruct the original resolution of the input image.

The final `output_layer` was built with a `softmax` activation function to produce class probabilities for each pixel.

For training, we compiled the model using the `Adam` optimizer, `sparse_categorical_crossentropy` loss, and the `mean_IoU` as a metric. The training process included the following callbacks:

- **EarlyStopping**: to monitor the validation loss and stop training if it did not improve for 20 consecutive epochs, restoring the best weights.
- **ReduceLROnPlateau**: to reduce the learning rate by a factor of 0.1 if the validation loss plateaued for 15 epochs.
- **Visualization Callback** (`viz_callback`): to monitor qualitative improvements during training.

The model achieved an accuracy of **0.446** on the Kaggle leaderboard.



The previous image is an example taken from the training set during the model's training process. Although the predicted segmentation for this sample looks reasonable, the model's low performance on the test set indicates overfitting: the model adapts too closely to the training data but fails to generalize well to unseen examples.

4 Second Model

For the second model, we increased the number of downsampling and upsampling layers to 3, allowing the network to capture and process features at multiple scales more effectively.

4.1 Data Augmentation

To improve the model's generalization ability, we experimented with various data augmentation techniques, including:

- `random_flip`
- `random_contrast`
- `random_brightness`
- `random_pixel_shift`
- `random_noise`

However, we observed that most of these augmentations introduced distortions that negatively affected the model's performance. The only augmentation that consistently improved accuracy was `random_flip`, which provides sufficient variation without compromising the quality of the input data.

4.2 Bottleneck

After several attempts, we achieved a significant improvement by implementing our custom `bottleneck_advanced` function. Compared to the previous bottleneck function, `unet_block`, which consisted of stacked convolutions, batch normalization, and activation layers, the new bottleneck introduced the following enhancements:

- **Dilated Convolutions:** A sequence of convolutional layers with increasing dilation rates (`[1, 2, 4]`), enabling the network to capture features at different receptive fields.
- **Squeeze-and-Excitation (SE) Block:** this architecture recalibrates the feature map channels generated by intermediate layers. It

applies global average pooling to each channel of the feature maps, effectively summarizing the spatial information. This is followed by two dense layers that adaptively assign weights to each channel, highlighting the most relevant patterns for segmentation and suppressing less important ones.

The updated `bottleneck_advanced` function efficiently combines these elements, resulting in a better representation of multi-scale features and a noticeable improvement in model performance.

With these modifications, the second model achieved **0.496** of accuracy.

5 Third Model

In the third model, we introduced a custom loss function, `weighted_loss_with_zero_background`, to address the class imbalance issue and improve the model's ability to differentiate between terrain classes, particularly class 4 (*Big Rock*).

5.1 Weighted Loss Function

The weighted loss function was designed to assign specific importance to each class based on its frequency in the dataset. After several experiments, we achieved the best results using the class weights [**0, 1, 1, 1, 2**], where:

- Class **0** (Background): weight set to **0**, effectively excluding background pixels from contributing to the loss.
- Classes **1, 2, 3** (Soil, Bedrock, Sand): weights set to **1**, maintaining equal importance for these classes.
- Class **4** (Big Rock): weight set to **2**, emphasizing the underrepresented and significant class.

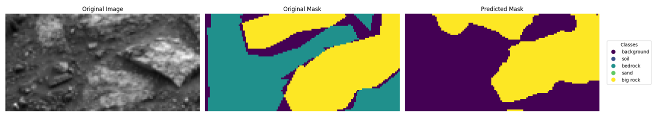
By excluding the background from the loss computation, the model focused on learning meaningful patterns in the data. Additionally, the increased weight for class 4 allowed the model to better capture features associated with *Big Rock*, addressing the class imbalance challenge.

The third model achieved an accuracy of **0.615** on the Kaggle leaderboard, marking a 12% improvement compared to the second model.

6 Final Model

For the final model, we focused on improving the network's ability to capture and process features at different scales. We introduced dropout layers with a rate of **0.2** and applied **L2 regularization** to all three upsampling and downsampling paths. These changes helped the model generalize better and reduce overfitting, leading to improved performance.

With these adjustments, the final model achieved an accuracy of **0.687** on the Kaggle leaderboard, marking a significant improvement over the previous models.



Last figure is the prediction from the best-performing model. Although the segmentation is still not perfectly accurate, the model successfully captures the general pattern and distinguishes the main classes.

7 Contribution

Our work was a collaborative effort: **Lorenzo** focused on exploring the dataset and removing outliers, **Francesco** developed the model architectures and identified an effective bottleneck structure, **Zeno** introduced and refined the weighted loss function, and **Daniele** tuned the hyperparameters to achieve the best leaderboard performance.

8 References

- Professor of Artificial Neural Networks, *Lectures and experiments on Artificial Neural Networks*, 2024.
- OpenAI, *ChatGPT: AI assistant for research and implementation*, 2024. Accessed via personal interactions to refine and develop models. `https://www.chatgpt.com`
- TensorFlow Developers, TensorFlow Keras API Documentation, 2024. Available at: `https://www.tensorflow.org/api_docs/python/tf/keras`.