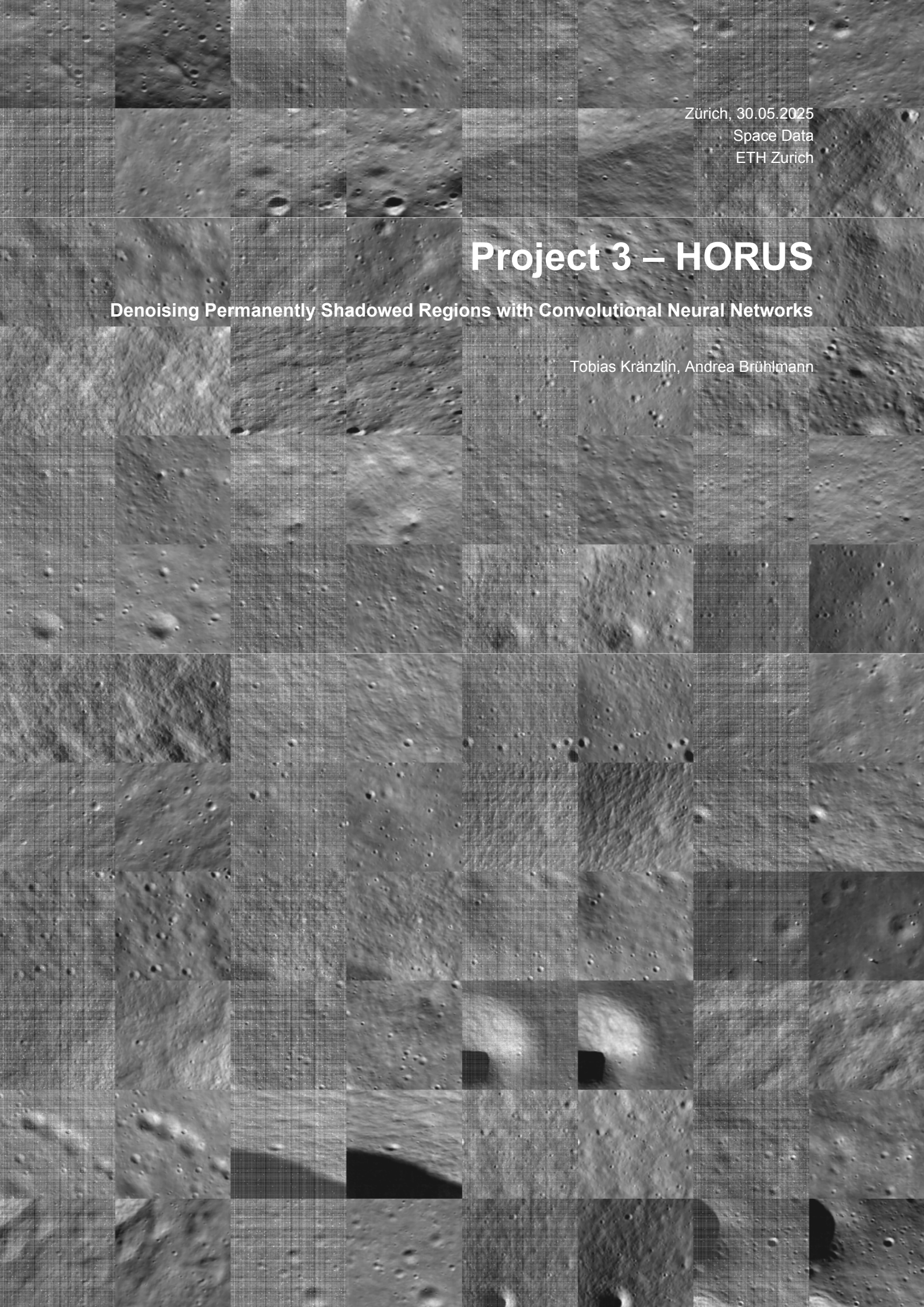


Zürich, 30.05.2025
Space Data
ETH Zurich

Project 3 – HORUS

Denoising Permanently Shadowed Regions with Convolutional Neural Networks

Tobias Kränzlin, Andrea Brühlmann

CONTENTS

List of Figures

1 Introduction **1**

2 Methods **1**
2.1 Previous Architectures 1
2.2 Final Model 1
2.2.1 Data Preprocessing 2
2.2.2 Data Augmentation 2
2.2.3 Early Stopping 2
2.2.4 Hyperparameter Grid Search 2
2.2.5 Final Training 3

3 Results **3**
3.1 Configuration and Performance Comparison 3
3.2 Training and Validation Losses 3
3.3 Cross-Validation 4

4 Discussion **5**
4.1 Image Visualisations 5
4.2 Challenges 5

LIST OF FIGURES

0.1 Title page background image, generated using the final model . . . . . .
2.1 Overview of the Residual Block Architecture 1
2.2 Architecture Overview of the final Model 2
3.1 Comparison model with and without early stopping 4
3.2 Comparison model with the best and worst RSME performance 4
4.1 Comparison of noisy input, ground truth clean images, and model predictions for two random samples each from the training, validation, and test datasets 6

1 INTRODUCTION

Images captured by space missions of the Moon's permanently shadowed regions (PSRs) often exhibit significant noise due to extremely low or absent direct sunlight. Denoising such images helps improve their quality and usefulness for further analysis.

In this project, several CNN-based approaches for denoising low-resolution (64×64) grayscale images of the Moon are investigated. The goal is to identify a model that delivers strong denoising performance while also training efficiently on the available dataset. Different architectures and training strategies were evaluated, including a U-Net-inspired convolutional network leveraging residual blocks and an approach using image rotations to improve robustness and data augmentation.

Given the high computational requirements associated with training convolutional neural networks on large-scale image datasets, most experiments were executed on the Euler high-performance computing cluster at ETH Zürich to ensure efficient processing.

2 METHODS

2.1 Previous Architectures

The first implementation was a simple autoencoder, utilising three convolutional layers in the encoder and three convolutional layers in the decoder. With a Root Mean Square Error (RMSE) of 0.0505, this gave acceptable results for the normal images, but a larger RMSE of 0.0622 for the images with more noise. A second, more advanced model was developed based on the U-Net architecture. This version included improvements such as batch normalisation, the Adam optimiser and consisted of 15 layers. This model resulted in an RMSE of 0.0559 for the images with more noise.

2.2 Final Model

The final model uses the UNet-like architecture from the previous model and expands it using residual blocks. Residual blocks can help optimise deeper networks by essentially making the layers learn a residual function instead of a direct mapping, which improves gradient flow during training. This can stabilise learning and can be especially useful in denoising, where small pixel-wise corrections matter. The architecture of the residual block is shown in Figure 2.1. It consists of two convolution layers and two batch normalisation layers, with a ReLU activation function after the first batch normalisation.

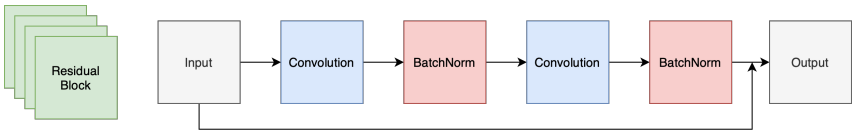


Figure 2.1. *Overview of the Residual Block Architecture*

The model itself consists of 2 encoding blocks, each followed by a pooling layer, a bottleneck, and two decoding blocks, each with an upsampling layer. The encoding blocks extract the features and compress the image. They each consist of a convolution layer, followed by a batch normalisation layer and an inline ReLU activation function, and finally a residual block as described above. Each encoding block is followed by a pooling layer to reduce the spatial size. The bottleneck acts as a transformer, again consisting of a convolution and a batch normalisation layer, an inline ReLU activation function and a subsequent residual block. The decoding blocks are both prepended by an upsampling layer

and a concatenation with the corresponding encoding block. This skip-connection allows the model to retain spatial info. Finally, the decoding blocks themself again consist of a convolution and a batch normalisation layer, followed by an inline ReLU activation function and a residual block. A final overview of the model can be seen in Figure 2.2.

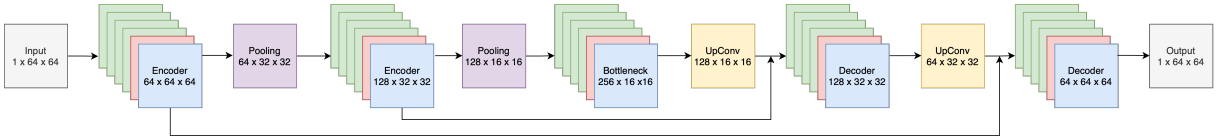


Figure 2.2. *Architecture Overview of the final Model*

2.2.1 Data Preprocessing

For all tests of the model, the images with more noise were used. The noisy and clear images were separated into a train, validation and test set. The function of the test set is to estimate the generalisation error by testing the model on images it has never seen before. The test set contained 10% of the images. The validation set, containing 5% of the images, is used to evaluate the model during training. The rest of the images were used for training the model. The images were normalised to $[0, 1]$ by a simple division by 255. To ensure reproducibility, the randomised splitting of the test, train and validation sets was based on a seed. This allowed local testing of the model on the CPU while ensuring the test set did not contain any images the model had seen before.

2.2.2 Data Augmentation

To achieve better generalisation of the model, data augmentation was implemented. Here, each image was rotated by $90°$, $180°$ and $270°$, making the resulting dataset 4 times as large as the original. This can help prevent overfitting and allows the network to artificially have more data to learn from. To ensure comparison with other models, the test did not consist of augmented data.

2.2.3 Early Stopping

To avoid overfitting to the training and validation sets and to shorten model training time, an early stopping algorithm was implemented. This algorithm simply checked if the current validation loss is better than the best validation loss seen so far, and if not, stopped the training after a patience limit was exceeded. Then, to get the best performance, the model is saved at the state where the validation loss was the lowest.

2.2.4 Hyperparameter Grid Search

To optimise the model, a grid search over the parameters in Table 2.1 was done. To enhance efficiency, each combination of the parameters was run in parallel on the Euler cluster. Results showed that a combination of a learning rate of 1e-4, batch size of 16 and a patience of 5 performed the best.

Parameter	Learning Rate	Batch Size	Patience
Values	1e-3, 5e-4, 3e-4, 1e-4, 5e-5	16, 32, 64	5, 10

Table 2.1. *Hyperparemeter Overview*

2.2.5 Final Training

After the best configuration of the model was found, a final training of the model was done. Here, to ensure the best generalisation error, the model was trained on the full combined test and validation set and then validated once on the hidden test set. The values used for the final training can be seen in Table 2.2

Parameter	Learning Rate	Batch Size	Patience	Epochs
Value	1e-4	16	5	18

Table 2.2. *Overview of the final parameters*

3 RESULTS

3.1 Configuration and Performance Comparison

To compare the different models and configurations, the RMSE on the test set was calculated for all experiments to ensure a consistent basis for evaluation. The final model with data augmentation was compared to the same model without data augmentation. Further, the model was also compared to the previous UNet-like architecture. The RMSE on the test set is reported in Table 3.1.

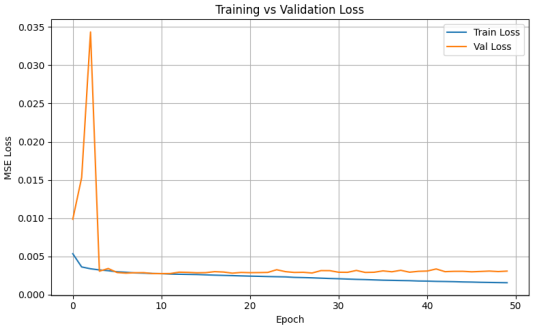
Model	No Augmentation	Data Augmentation	UNet
RMSE	0.0553	0.0507	0.0559

Table 3.1. *Comparison of different model parameters and architectures*

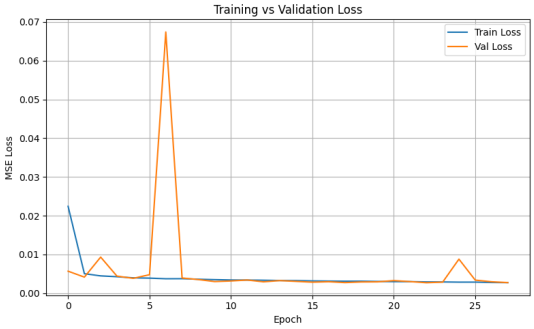
The results show that the UNet-like architecture performs almost as good as the final model using residual blocks. This is probably because both of the models train on a comparably small data set. Still, in tests with less noise, the model with residual blocks showed a consistently better error. For the final model using data augmentation, the RMSE is significantly better, showing that the model benefits from using more data.

3.2 Training and Validation Losses

To further evaluate model behaviour, training and validation loss curves were plotted for different configurations. First, the final model was evaluated with and without early stopping, as shown in Figure 3.1.



(a) *Without early stopping with RSME 0.0534*

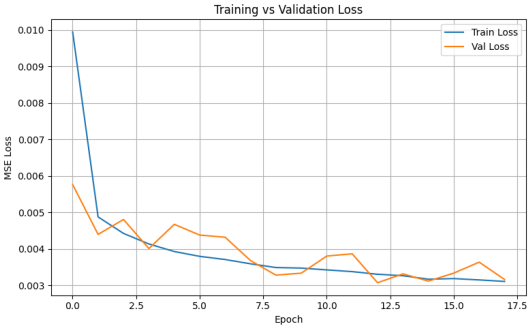


(b) *Early stopping with RMSE 0.0529*

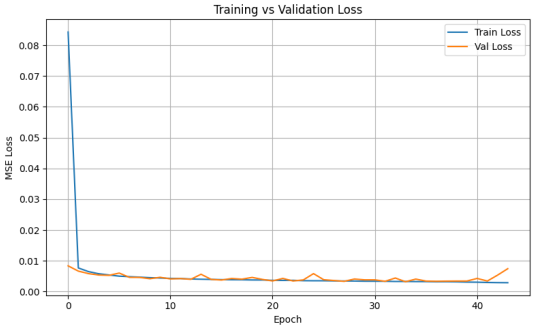
Figure 3.1. *Comparison model with and without early stopping*

In Figure 3.1a, the two loss curves initially follow a similar trajectory, but begin to diverge after epoch 10. The weights are saved at a point where the loss curves have already diverged significantly, leading to a higher RMSE. In contrast, with early stopping, the weights are saved at the point of optimal validation performance, resulting in better generalisation and a lower RMSE.

Another approach involved comparing two plots from models with different hyperparameter settings, such as learning rate, batch size, and patience. The plots in Figure 3.2 show the loss curves of the best-performing model and the worst-performing one.



(a) *Best performing model with batch size 16, learning rate 1e-4 and patience 5, resulting in an RSME of 0.0541*



(b) *Worst performing model with batch size 64, learning rate 5e-5 and patience 10, resulting in an RSME of 0.0852*

Figure 3.2. *Comparison model with the best and worst RSME performance*

In Figure 3.2b it is clearly visible that both the training and validation losses drop very quickly without showing a typical learning curve with a gradual decrease like in Figure 3.2a. This was probably due to the lower learning rate, which led to slow and unstable learning. Also, a larger batch size tends to produce smoother but less informative gradients, potentially limiting the model's ability to capture fine details needed for denoising. As a result, the model achieved a much higher RMSE compared to the best-performing model.

3.3 Cross-Validation

To evaluate the robustness and generalisation performance of the final model, a 5-fold cross-validation was conducted on the final model. For each fold, the training and validation MSE was reported and the RMSE was calculated on the test set. In Table 3.2, the model's performance across different data splits is summarised. Since the average train and validation MSE are almost identical, there is no sign

of overfitting to training data. With an average RMSE of 0.0520 and a standard deviation of 0.0004, the model shows consistency across the folds and a good generalisation error.

Fold	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	**Average**
Train MSE	0.0027	0.0028	0.0030	0.0027	0.0026	**0.0028**
Validation MSE	0.0027	0.0028	0.0029	0.0028	0.0027	**0.0028**
Test RMSE	0.0527	0.0514	0.0519	0.0520	0.0519	**0.0520**

Table 3.2. *Train, Validation and Test error for 5-fold cross-validation on the final model*

4 DISCUSSION

4.1 Image Visualisations

Figure 4.1 presents predicted images from the training set, the validation set, and the hidden test set, along with the corresponding noisy and clean images.

Overall, the predicted images are very accurate, showing only minimal differences compared to the clean images. A minor difference can be observed in the first image from the training set in the top left, where some very faint craters are not predicted. Also, flat surface regions sometimes get blurred a little bit, which makes them appear less sharp.

The minor missed craters in the first training image likely result from the model prioritising generalisation over memorising noise, thus preventing overfitting. The slight blurring of flat regions may be caused by the model's smoothing effect, which helps reduce noise but can soften fine details.

4.2 Challenges

One challenge was to avoid overfitting; to mitigate this, dropout layers were integrated into the network architecture, and validation loss was closely monitored during training to ensure that the model maintained good generalisation performance. We also experimented with different numbers of epochs and batch sizes to evaluate their impact on model performance.

Another challenge was maintaining a clear and organised directory structure on the Euler cluster to manage all code scripts, SLURM outputs, generated files, and images efficiently. To ensure this, the output name was initialised once at the beginning of the code, and all subsequent outputs were saved consistently using this name. Further, the use of flags in the code allowed easier handling of the code for different usages.

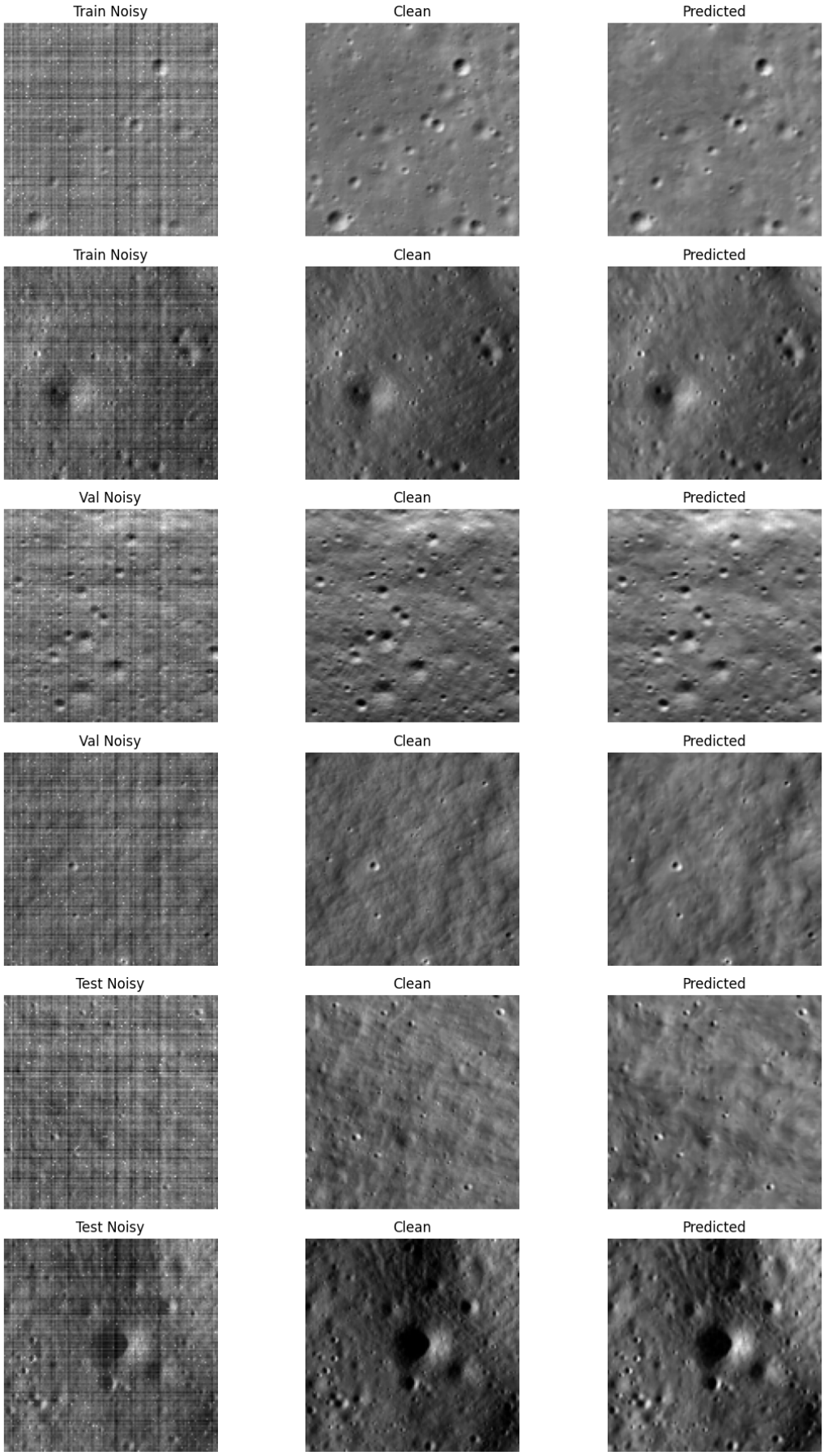


Figure 4.1. *Comparison of noisy input, ground truth clean images, and model predictions for two random samples each from the training, validation, and test datasets*