Hack4dev
hack4dev.org

Regional Hackathon Pipeline Submission

Team Number: 3
Regional Hackathon Location: Brazilian Center for Physics Research
Date of Submission:
Team Members: Gabriel Christino, Marcela Gouvêa and Matheus Poltronieri

1. Introduction

VERTECS (Visible Extragalactic background RadiaTion Exploration with CubeSat) is an astronomical nanosatellite designed to study the history of star formation by observing the extragalactic background light in the optical range. It utilizes a small telescope and precise attitude control to collect data. However, due to its compact size, the VERTECS nanosatellite has limited computing power, storage capacity, and communication bandwidth. These constraints result in slower data transmission, making it challenging to send large volumes of data efficiently and potentially impacting the mission's overall effectiveness.

To address this issue, machine learning techniques were developed to prioritize the most valuable data for transmission, optimizing the use of onboard resources. These models are deployed directly on the satellite to pre-select high-priority data, thereby improving the efficiency of data downlink.

The objective of the CubeSat challenge is to enhance the efficiency and accuracy of a machine learning model tasked with classifying the data captured by CubeSats. Specifically, the goal is to prioritize the transmission of the most scientifically valuable images back to Earth, within the constraints of limited onboard resources and slow communication speeds.

Participants were provided with notebooks presenting two initial approaches: a traditional machine learning method and a lightweight Convolutional Neural Network (CNN) model described in [1]. For this challenge, we chose to improve the provided CNN model for image classification, using it as the baseline for our enhancements, focusing on the balance between accuracy and computational performance. Our pipeline involved preprocessing the images and employing a Transfer Learning approach by integrating a pre-trained model into our CNN architecture.

2. Data overview and preprocessing

The training dataset consisted of 9,711 RGB images, while both validation and test sets included 3,237 images each. Each image was categorized into one of five classes: "Blurry," "Noisy," "Missing data," "Corrupt," or "Priority."

To prepare the data for the CNN, all images were resized to 224×224 pixels, reducing computational demands. Additionally, pixel values were normalized to the range [-1, 1], ensuring consistent input scaling.

3. Model Architecture and Training

3.1. Base model

We chose to employ a Transfer Learning approach by integrating a pre-trained model into our CNN architecture. Transfer learning is a machine learning technique where knowledge gained from training on one task or dataset is used to enhance model performance on a related task or a different dataset.

In our case, we utilized MobileNetV2 as the base model, a CNN architecture specifically designed to achieve high performance on mobile and resource-constrained devices. Its structure begins with an initial fully convolutional layer containing 32 filters, followed by 19 residual bottleneck layers that enhance efficiency and accuracy.

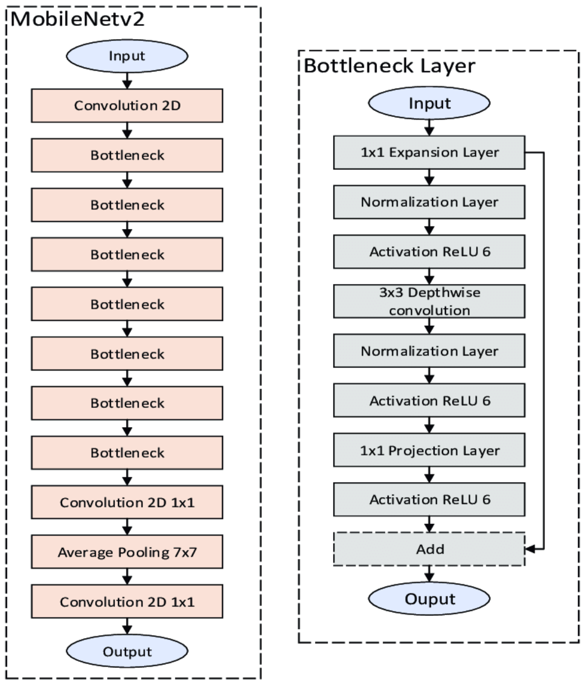


Figure 1. The architecture of MobileNetV2 DNN. [2]

MobileNetV2 starts with a standard 3x3 convolution layer, followed by several Inverted Residual Blocks. Each block consists of:

1. 1x1 convolution (Expansion layer): Increases the channel's depth.
2. 3x3 depthwise convolution: Applies lightweight filtering.
3. 1x1 convolution (Projection layer): Reduces channels back with linear activation.

At the end, there's a global average pooling and a fully connected layer for classification. The base model was pre-trained on the ImageNet dataset, a large-scale collection of labeled images sourced from the internet. The key parameters selected were:

- `weights='imagenet':` Uses pre-trained weights from ImageNet
- `include_top=False:` Excludes the final classification layers, enabling customization for our specific task.
- `alpha=0.35:` Reduces the model's width, decreasing the number of parameters.
- `input_shape=(224, 224, 3):` Sets input image size to 224x224 pixels with 3 color channels (RGB).
- `base_model.trainable = False:` Freezes the base model's weights to retain pre-trained features

3. Model training and validation

For our CNN model, we used MobileNetV2 as the convolutional base and a top architecture consisting of Global Average Pooling, a dense layer with five neurons, and a Softmax output for classification.

To balance performance and computational cost, we tested multiple hyperparameter combinations and image resolutions (128×128, 200×200, and 256×256 pixels). We ultimately chose 224×224 pixels, as it aligns with the optimal input size for MobileNetV2 pretrained on ImageNet.

We also experimented with different values of the alpha parameter, which controls model complexity. Initially, we used the full model (alpha = 1), but due to high resource demands, we gradually reduced it to 0.75, 0.5, and finally 0.35. Notably, the model's performance remained stable at alpha = 0.35, representing a 35% width of the full MobileNetV2 model. We opted for a batch size of 32, as it is the TensorFlow default and provides a good balance between processing time and memory usage.

To expedite parameter tuning, we used only 20% of the training and validation set while maintaining class proportions. For all configurations, we analyzed loss and accuracy curves over epochs using the full dataset (9,711 training and 3,237 validation images) to assess underfitting or overfitting. Image resolution had the greatest impact: at 128×128 pixels, the model exhibited underfitting. Other parameter adjustments mainly affected convergence speed, but the model remained robust. Below, Figure 2 shows the loss and accuracy curves for our final model: 224×224 images, alpha = 0.35, trained on the full dataset.

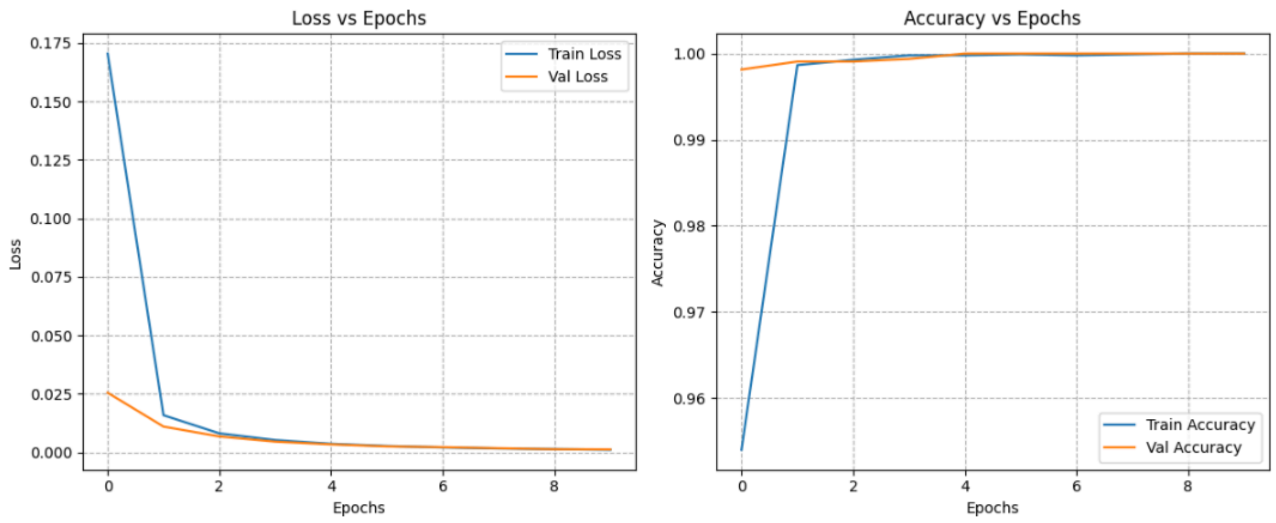


Figure 2: Loss and accuracy curves versus epochs for the training and validation set

In Figure 2, both training and validation curves converge to zero loss and an accuracy of one, indicating that the model successfully passed validation. There are no signs of underfitting (poor overall performance) or overfitting (high training accuracy but poor validation accuracy). To further verify this, Figure 3 presents the confusion matrix for the validation set:

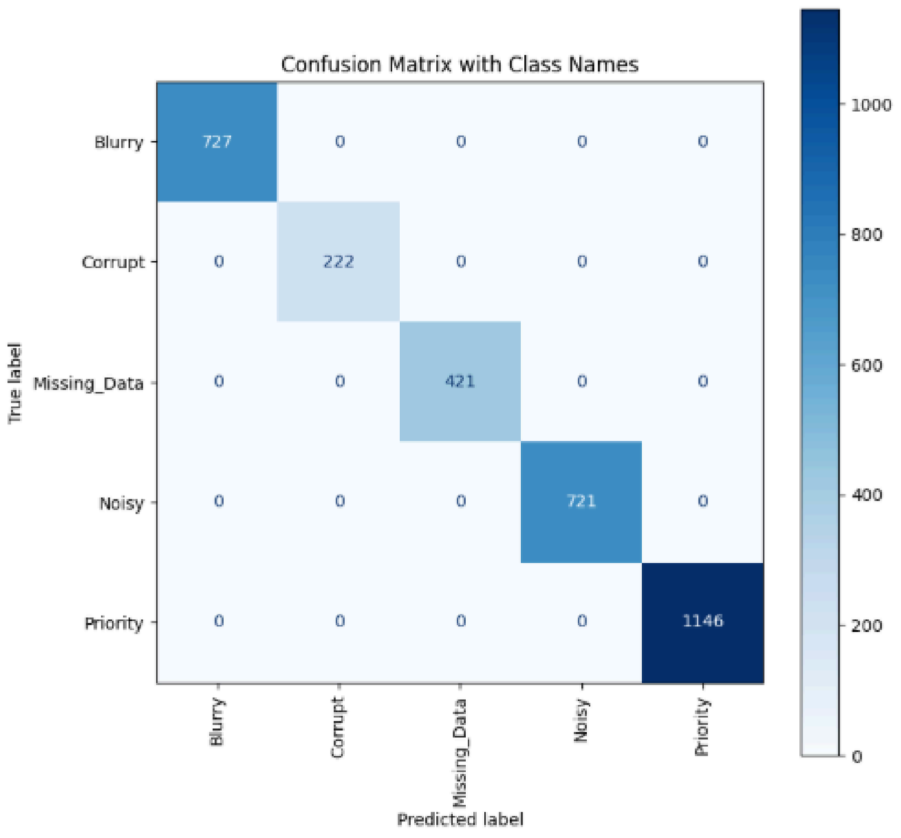
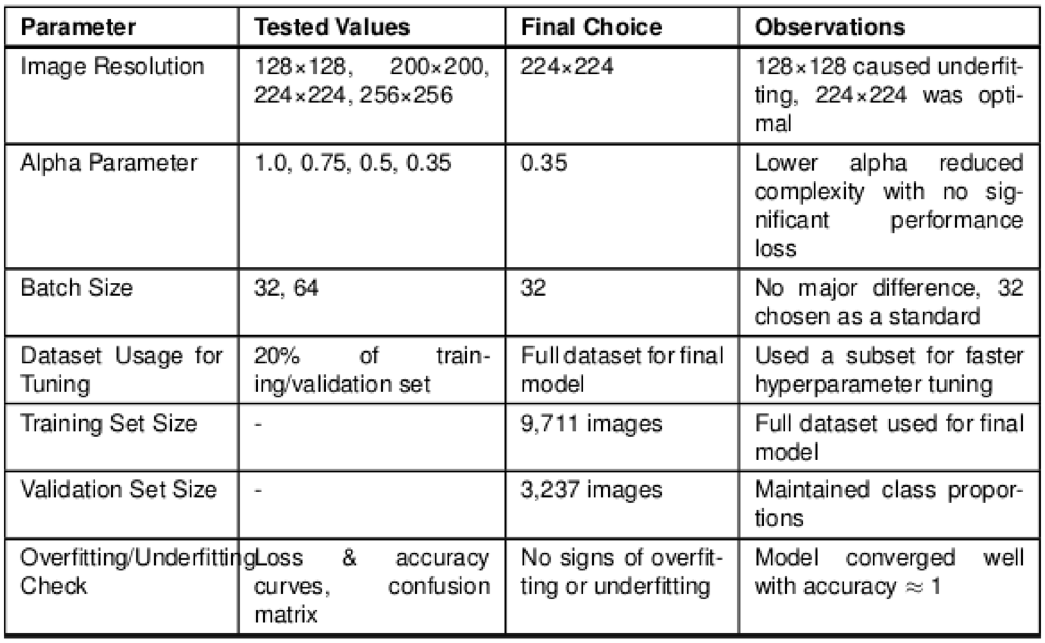


Figure 3: Confusion matrix for the validation set

Figure 3 confirms the model's perfect classification, aligning with the accuracy convergence to one. The confusion matrix shows that every validation sample was correctly classified, with zero false positives and false negatives. This results in 100% accuracy, precision, recall, and F1-score across all five classes.

Despite variations in class representation—ranging from 1,146 samples in the "Priority" class to 222 in "Corrupt"—the model demonstrated consistent performance, highlighting its robustness and generalization capability.

Below is a summary of the development process:

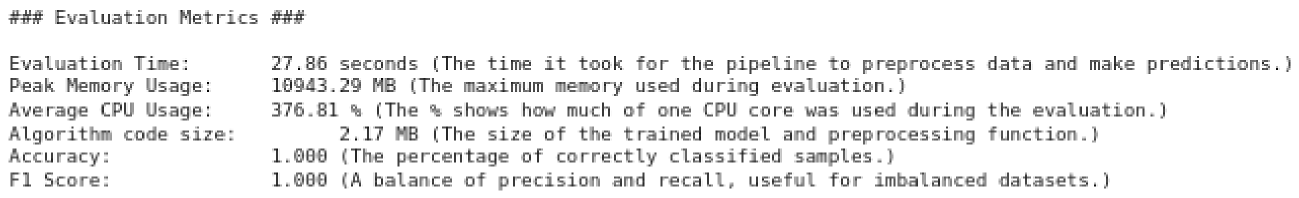


Parameter	Tested Values	Final Choice	Observations
Image Resolution	128×128, 200×200, 224×224, 256×256	224×224	128×128 caused underfitting, 224×224 was optimal
Alpha Parameter	1.0, 0.75, 0.5, 0.35	0.35	Lower alpha reduced complexity with no significant performance loss
Batch Size	32, 64	32	No major difference, 32 chosen as a standard
Dataset Usage for Tuning	20% of training/validation set	Full dataset for final model	Used a subset for faster hyperparameter tuning
Training Set Size	-	9,711 images	Full dataset used for final model
Validation Set Size	-	3,237 images	Maintained class proportions
Overfitting/Underfitting Check	Loss & accuracy curves, confusion matrix	No signs of overfitting or underfitting	Model converged well with accuracy ≈ 1

Table 1: Summary of the tested values of parameters and the final choice we made for our model, and its effects on the model performance.

4. Results and Discussion

After validation, we tested the model's generalizability using a test set of 3,237 images. This step assesses the model's real-world performance before deployment. The key metrics and resource usage details are summarized in the image below:

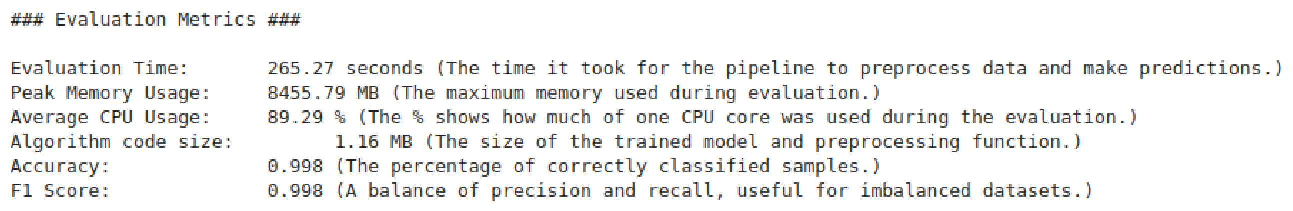


```
### Evaluation Metrics ###

Evaluation Time:        27.86 seconds (The time it took for the pipeline to preprocess data and make predictions.)
Peak Memory Usage:      10943.29 MB (The maximum memory used during evaluation.)
Average CPU Usage:      376.81 % (The % shows how much of one CPU core was used during the evaluation.)
Algorithm code size:          2.17 MB (The size of the trained model and preprocessing function.)
Accuracy:               1.000 (The percentage of correctly classified samples.)
F1 Score:               1.000 (A balance of precision and recall, useful for imbalanced datasets.)
```

Figure 4: Evaluation metrics of our model during the test.

For comparison, we present below the evaluation metrics of the CNN model from [1], tested on the same dataset and using the same resources:



```
### Evaluation Metrics ###

Evaluation Time:       265.27 seconds (The time it took for the pipeline to preprocess data and make predictions.)
Peak Memory Usage:     8455.79 MB (The maximum memory used during evaluation.)
Average CPU Usage:     89.29 % (The % shows how much of one CPU core was used during the evaluation.)
Algorithm code size:         1.16 MB (The size of the trained model and preprocessing function.)
Accuracy:              0.998 (The percentage of correctly classified samples.)
F1 Score:              0.998 (A balance of precision and recall, useful for imbalanced datasets.)
```

Figure 5: Evaluation metric of the CNN model presented in the article in [1]

The initial model took 265.27 seconds for evaluation, while our model completed preprocessing and classification of the last image batch in 27.86 seconds, achieving a 9.5x speedup.

Regarding resource usage, our model increased RAM usage from 8.5GB to 10.9GB, a 28% rise. While the original model from [1] utilized 89.29% of a single CPU during preprocessing and classification, our model employed 3 full CPUs and 76% of a fourth CPU.

Although both models demonstrated excellent accuracy and F1 scores, our model achieved a perfect accuracy of 1.000, compared to 0.998 in the original model. The confusion matrix for our model's test set is shown in Figure 6 below, highlighting this improvement:

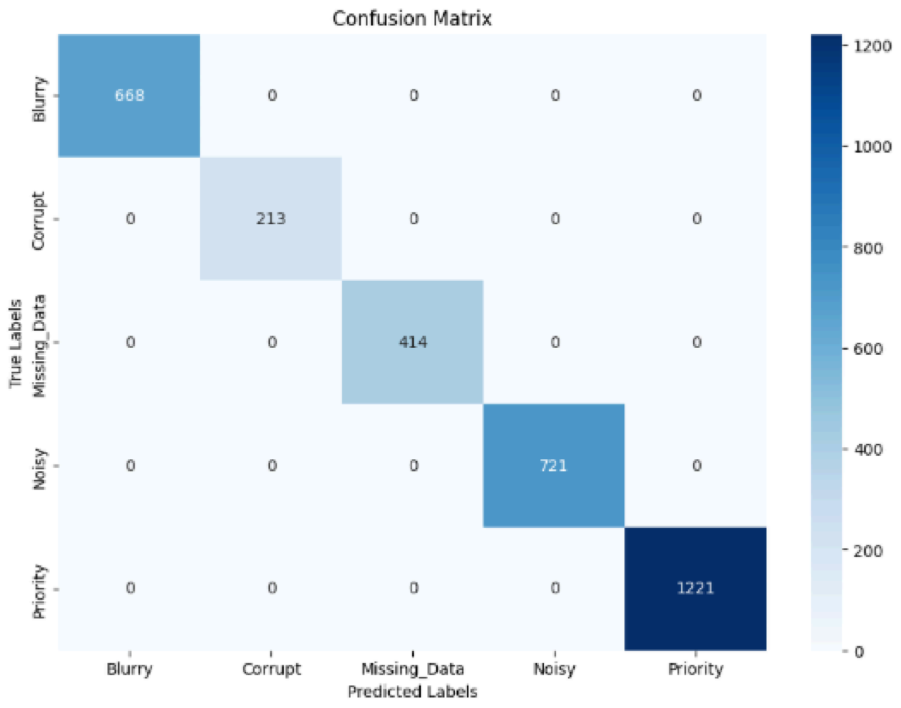


Figure 6: Confusion matrix during the test of the model developed by our team.

In this instance, the model correctly classified all classes. However, the result may vary slightly due to randomness during training. In other instances, for example, the model made one mistake. For comparison, Figure 7 below shows the decision matrix of the original CNN model during testing:

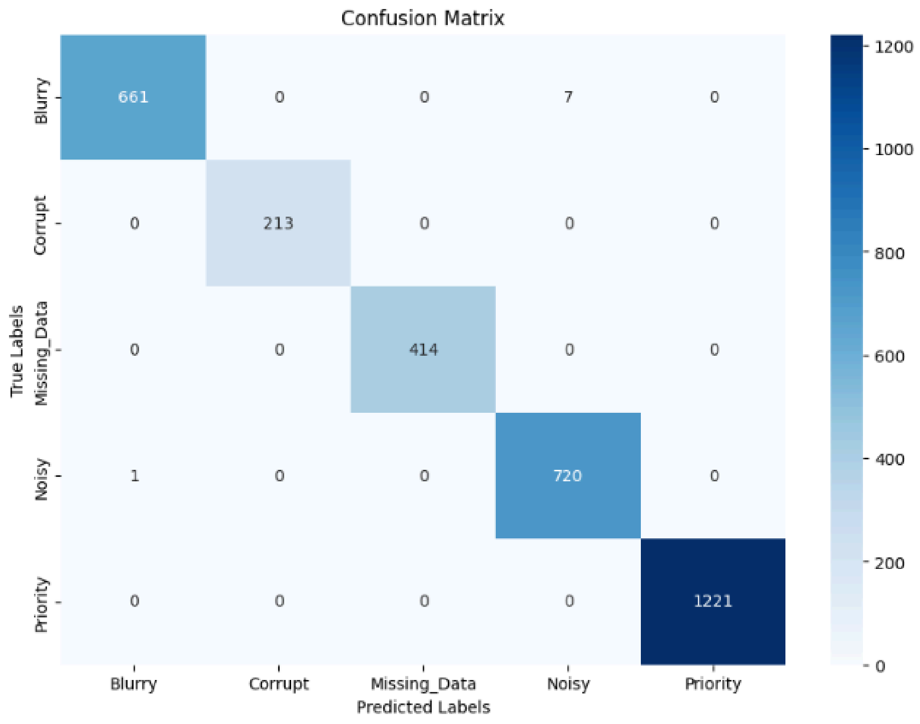


Figure 7: confusion matrix during test of the CNN model from [1]

As shown in Figure 7, the confusion matrix of the original CNN model misclassified 8 instances. The matrix reveals that the original model struggled to differentiate between blurry and noisy images, often confusing them. In contrast, our model was able to accurately distinguish between these images.

4. Conclusions

The model developed by our team improved evaluation time, preprocessing and classifying images nearly 10 times faster. However, resource usage did not increase at the same rate as performance. Memory usage rose by 28%, and CPU usage increased from nearly 1 CPU to almost 4 CPUs. Additionally, our model achieved better accuracy by successfully distinguishing between blurry and noisy images that the previous model misclassified.

Hack4dev

hack4dev.org

References

[1] Keenan A. A. Chatar, Ezra Fielding, Kei Sano, and Kentaro Kitamura "Data downlink prioritization using image classification on-board a 6U CubeSat", Proc. SPIE 12729, Sensors, Systems, and Next-Generation Satellites XXVII, 127290K (19 October 2023); https://doi.org/10.1117/12.2684047.

[2] Design Space Exploration of a Sparse MobileNetV2 Using High-Level Synthesis and Sparse Matrix Techniques on FPGAs - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-architecture-of-MobileNetV2-DNN_fig1_361260658 [accessed 23 Mar 2025]