MULTI-LAYER ADVERSARIAL DETECTION FOR SHIP SEGMENTATION ON NVIDIA JETSON ORIN

Badr-Eddine Bouhlal*, Clemens-Alexander Brust*
* Institute of Data Science, German Aerospace Center (DLR), Jena, Germany

Abstract

Autonomous systems, including Unmanned Aircraft Systems (UAS), rely on deep learning models for critical tasks such as object detection and segmentation. However, they are targeted by different threats, among them adversarial attacks, where small perturbations in input images can deceive the model and lead it to mispredict. Adversarial images (AEs) remain a challenge, as they are often similar to the expected inputs, making them difficult to distinguish for human observers as well as some models. This study defines a method for detecting AEs in an autonomous system by leveraging multi-level system monitoring, including image-based, model-based, and hardware-based metrics. Our goal is to identify key indicators of adversarial image manipulation while minimizing experimental bias through controlled evaluation conditions. We generate multiple adversarial samples using both white-box and black-box attack strategies. Experiments are conducted using the Airbus Ship Detection dataset, which comprises high-resolution satellite images suitable for aviation-related applications such as aerial coastal monitoring and maritime surveillance. A ship segmentation model is deployed on an NVIDIA Jetson Orin AGX, and metrics are collected during inference under controlled conditions. Our contribution includes, first, identifying a relevant set of features that effectively distinguish AEs from genuine inputs. Second, we investigate if system profiling - by including model and hardware monitoring - can provide additional robust signals for adversarial detection beyond traditional image-based analysis. We propose a real-time adversarial detection pipeline based on supervised classification over Image, Model-profiling, and hardware metrics. The results reveal that the detection relies mainly on image features while profiling features provide complementary cues for some attacks and hardware (Tegrastats) metrics are noisy and largely uninformative.

Keywords
Adversarial Examples; Image Segmentation; NVIDIA Jetson Orin AGX; Tegrastats Utility

1. INTRODUCTION

Adversarial attacks represent a high safety challenge for deep neural networks, especially when deployed in real safety-critical and strategic use cases, such as in unmanned vehicles operating in edge computing environments with limited resources in terms of memory and computation power. Examples of such platforms include AI accelerators like the NVIDIA Jetson AGX. In these environments, the implementation of traditional defenses is more difficult due to resource restrictions. Different methods for defending against Adversarial Examples (AEs) exist, ranging from robustness-enhancing approaches (defense mechanisms) [1, 2], where techniques like adversarial training are employed, to detection mechanisms [3, 4], where detectors are implemented at the input of the inference pipeline to monitor incoming data and catch the unusual image input before they reach the inference step.

In this study, we examine a detection-based approach that deals with an un-targeted attack scenario, where the adversary's objective is not to force the model toward a specific mis-classification but rather to degrade prediction accuracy by inducing misleading outputs. We employ two categories of multiple adversarial attack methods in this work. Black-box methods [5] where the adversary does not have knowledge of the internal characteristics of the victim Deep Learning model but can access its inputs and outputs, and white-box methods [1] where the adversary has partial or complete knowledge of the victim deep learning model [6].

In this paper, we explore a metric-driven approach for detecting adversarial examples during inference. Our main contributions are as follows:

- We deployed an image segmentation model on the NVIDIA Jetson AGX platform and used it as the basis for evaluating adversarial robustness in a realistic edge computing environment.
- We analyzed different types of features separately and combined, including image-based indicators, hardware-level behaviors, and model profiling metrics, to understand their effectiveness in detecting adversarial manipulations.
- We conduct experiments under both white-box and black-box untargeted attacks to evaluate how differ-

doi: 10.25967/650199

ent adversarial strategies impact the detection potential of each feature type.

2. RELATED WORK

Several studies have explored the use of low-level system and hardware signals to identify adversarial input.

[7] proposes detecting AEs by monitoring micro-architectural activities during inference. It focuses on detecting altered neural activation patterns caused by adversarial perturbations that may lead to different behavior in cache misses or branch instructions that do not occur in normal inputs. For example, the study shows that cache-miss monitoring yielded up to 99.25% detection accuracy.

[8] examines the usability of the CPU hardware, more specifically the Performance Monitoring Unit (PMU) to detect AEs. For the study, they used the FGSM method with perturbation $\epsilon = 0.01$. The results show that no robust pattern was found that could distinguish between normal and AEs. Some counters varied by 1%, but these variations were inconsistent and statistically not relevant for the detection of AEs.

[9] used NVIDIA-smi which provides monitoring and management capabilities for each of NVIDIA's Tesla, Quadro, GRID and GeForce devices [10] to log GPU utilization, power, memory, and other NVIDIA-smi hardware metrics, the authors evaluated these metrics on a classification task using 14 different AEs. The results show that the detection accuracy of using GPU metrics might reach 80-90% for attacks like DeepFool and Simba on the MINT dataset. However, for other datasets like CIFAR and STL, the accuracy drops (65-70%).

Other works focus on detecting adversarial perturbations by analyzing image representations directly.

[11] used images-based extracted features. They employed two types of spectral representations of the Fourier transform as detection artifacts, detection based on the magnitude of Fourier coefficients, and detection based on the phase information of Fourier coefficients. The results showed that these Fourier-based features effectively work on gradient-based attacks such as Fast Gradient Sign Method (FSGM), Basic Iterative Method (BIM), and Projected Gradient Descent (PGD) with a detection score up to 90%, on the other hand, their method was less successful for optimization-based attacks like Deepfool and Carlini and Wagner (C&W).

[12] demonstrates that, the difference between compression and decompression of normal images and AEs reveals discrepancies because adversarial noise is mostly concentrated in high-frequency details, which JPEG compression tends to remove, while the main visual content of natural images remains largely unaffected. JPEG preprocessing reduces misclassification success (attack success rate) for FGSM and DeepFool on CIFAR-10 and GTSRB datasets.

Some approaches aim to detect AEs by profiling the internal behavior of the model parameters during inference, including layer-level statistics.

[13] uses an AI Performance Counter (APC) to monitor metrics such as layer sparsity and zero counts, dense layer activity (avg/min/max activation), FLOPs, MACs, entropy, throughput, etc., and analyze these APC metrics in real time to detect anomalies. The metrics collected during DeepFool adversarial attacks show high accuracy (up to 98%) in distinguishing adversarial inputs.

The results of the reviewed AE detection methods revealed an inconsistency of the results, especially in regard to hardware metrics showing inconsistency. [7] presented excellent detection results using cache misses. On the other hand, [9] demonstrates that GPU-based metrics could work only depending on the use case, model, data, AEs, and the hardware studied, since both studies used different types of hardware and which are different from the hardware we are using in this study. In terms of image features, while promising results have been achieved, the effectiveness does not extend to all AEs. For example, the C&W still poses issues due to its complexity and the slight perturbation it causes, as demonstrated by [11]. In our study, we evaluated the detection of AEs using all three categories of characteristics. Hardware metrics (on Jetson Orin AGX via tegrastats), image-based features, and model-level (segmentation) profiling, using a specific hardware architecture designed specially to run Deep Learning models on unnamed machines.

3. METHODOLOGY

The methodology consists of two main stages: offline and online, as shown in Figure 1.

The `Offline stage` involves deploying a ship segmentation model on a Jetson Orin AGX platform and then generate AEs using high resolution satellite images, an example of these images is shown in Figure 2. We perform black-box and white-box attacks on the normal images and the segmentation model to generate AEs. We perform tuning of multiple attacks parameters to determine the appropriate degree of perturbation. The latter is determined on the basis of the level of misprediction of the model and human perception. The perturbed image (the resultant AEs) should not be altered to the point where humans can recognize the differences between normal images and AEs.

The model will then be deployed, and we collect metrics to determine AEs artifact signals. The metrics include image-based features, hardware execution artifacts, and model profiling metrics.

First, we assess each feature separately to understand how it performs in detecting AEs. After that, we combine them to evaluate whether the combination improves performance. Our goal is not only to define the set of best features, but also to understand how these different feature types contribute to detecting AEs inputs.

`Online stages` consist of deploying a classification model that uses the combination of different metrics to detect AE images.

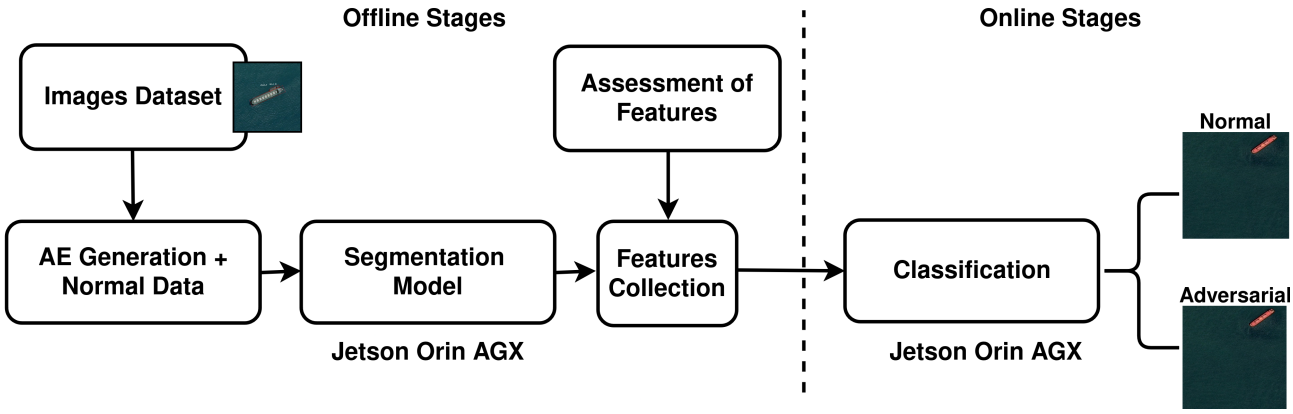


FIG 1. Workflow of the proposed methodology for adversarial example detection, from satellite images and AE generation to model deployment, metric collection, analysis, and final classification.

3.1. Ship Segmentation Model

For this work, we deployed a model of ship segmentation trained with the Airbus ship detection dataset [14], which contains satellite images for evaluating systems on the detection of ships in maritime environments. The model was implemented using Pytorch [1]. The model is relatively small, we did not need advanced optimization methods or tools such as TensorRT (an ecosystem of APIs for high-performance deep learning inference) [15], we instead used the Pytoch native optimizations and CUDA support to ensure efficient processing on the platform.
The model employs a DeepLabV3+ architecture [16]. The input images are RGB with three channels and the model outputs is a single-channel prediction map representing the probability of ship presence at each pixel.
Before the inference, input images are normalized using an encoder with specific mean and deviation values, this will ensure the consistency with the encoder pretraining. To handle class imbalance the Dice Loss with logits as a training objective is used, due to the fact that for the segmentation of the ship detection use case the background pixels vastly outnumber the foreground ones (ships).
During the inference, a fixed threshold of 0.5 to the Sigmoid outputs is applied to obtain binary segmentation masks. The model is evaluated using Intersection-over-Union (IoU) following the COCO method [17], computed both on a per-image basis and aggregated over the dataset. The model was trained offline using the Adam optimizer with a learning rate of 0.001. The model is trained with 220 epochs and achieves a validation IoU of 0.90

3.2. Adversarial Image Generation

To assess the robustness of the model of ship segmentation, we generate AEs using a combination of black-box and white-box attack methods as shown in Table 1. These attacks apply various perturbations to input images to simulate adversarial conditions. Under different attack scenarios, black-box attacks do not require any information about the target model, whereas white-box attacks rely on access to internal model information. These methods include both noise-based perturbations (black-box) and gradient-based perturbations (white-box).
We also tuned the perturbation magnitude (e.g. epsilon in FGSM or the amount of noise in the salt-and-pepper attack) with the model performance, assessing how different levels of perturbation impacted segmentation accuracy, the list of the methods and parameters is shown in Table 1 and an example of ship segmentation using a normal vs. an AE image is shown in Figure 3.

FIG 2. Sample satellite image from the airbus ship detection Dataset [14], showing a ship at sea.

3.2.1. Black-Box Attacks

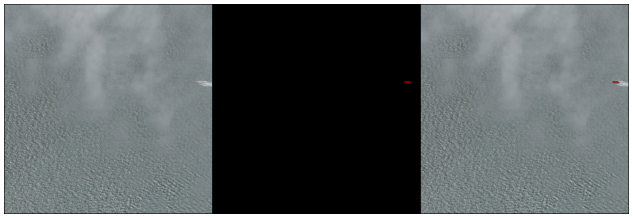
We used black-box attacks to simulate the real use case where the attacker does not have access or knowledge of the model internal parameters. Multiple methods were implemented:

- Random perturbation [18]: Adds Gaussian noise to the entire image, the noise grad can be adjusted by

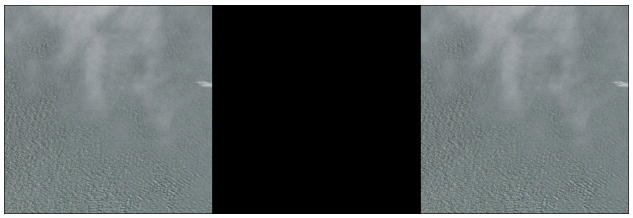
[1] PyTorch: https://pytorch.org

Method	Parameters	Type
FGSM [2]	$\epsilon = 0.02$	White-box
PGD [1]	$\epsilon = 0.1$, $\alpha = 0.05$, steps = 5	White-box
Random Perturbation [18]	(1) mean = 0, std = 0.05 (2) mean = 0, std = 0.1	Black-box
Gaussian Blur [18]	(1) kernel = 7×7, $\sigma = 5$ (2) kernel = 11×11, $\sigma = 10$	Black-box
Salt & Pepper [18]	(1) prob = 0.01, salt/pepper = 0.1 (2) prob = 0.09, salt/pepper = 0.2	Black-box
Negative Brightness [19]	(1) factor = 0.5 (2) factor = 1.0	Black-box

TAB 1. Adversarial and perturbation methods with corresponding parameters and type.



(a) Normal image: the ship is correctly segmented. Predicted ship pixels are shown in red while the absence of red segmentation indicates no detection



(b) Adversarial image generated using FGSM ($\epsilon = 0.02$): the ship is not segmented properly. Predicted ship pixels are shown in red while the absence of red segmentation indicates no detection

FIG 3. Comparison between normal and AEs for ship segmentation. The normal image is segmented accurately, whereas the adversarial image crafted using FGSM with $\epsilon = 0.02$ causes segmentation failure. Predicted ship pixels are shown in red while the absence of red segmentation indicates no detection.

a mean which enables to define the average noise to be added and standard deviation which controls the intensity of the noise, and these parameters enable us to simulate different degrees of distortion of the image.

- Negative [19]: Inverts the pixels of the images. We used this method to test the ability of the model to recognize ships under extreme color changes, the resulting image is the flipped image of the original (using a factor).
- Salt-and-Pepper Noise [18]: Adds randomly sets of percentage of pixels in the image to either maximum (white) or minimum (black) values. The proportion of noise is also controlled by a probability parameter. This attack enables us to assess the model under extreme pixel-level distortions.
- Gaussian blurring [18]: Apply a smoothing filter to the image, and this modification degrades the quality of high-frequency details, which can simulate, for example, scenarios where images are blurred due to environmental factors such as low resolution.

3.2.2. White-Box Attacks

The white-box attack uses the model's internal information to create perturbations designed specifically to exploit the weakness of the targeted model. we implemented the following gradient-based methods using the segmentation model:

- Fast Gradient Sign Method (FGSM): [2] This method perturbs the input image by applying a small step in the direction of the loss gradient with respect to the input.
- Projected Gradient Descent (PGD) [1]: which is an iterative attack that modify the image iteratively in multiple small perturbations steps while keeping the modifications small. Unlike FGSM which applies the modifications in one single step.

3.3. Inference Procedure

To evaluate the ship segmentation model and the NVIDIA Jetson response to normal and AEs, we processed the images corresponding to each method sequentially, but we observed the effect of GPU warm-up and memory saturation on the hardware metrics. In addition to this effect, we noticed that long sequential runs of the inferences caused cache growth in both the OS cache and the CUDA caching allocator. This led to memory pressure and, when the dataset was large, to engine stalls, so we implemented a round-robin processing method [20] as shown in Figure 4.

Each original image and its corresponding adversarial variants were processed iteratively, in order to mitigate this bias. At each iteration, the first unprocessed image from each adversarial method (including the normal image) is processed. We also performed per-inference cleanup, where we deleted intermediate tensors and arrays, synchronized the device, and cleared the CUDA allocators.

In this way, the effects of increasing GPU temperature, memory fragmentation will be distributed across all image types, since the experiment using a large dataset may last for multiple days and the caching effect will be avoided.

3.4. Segmentation Monitoring And Features Generation

3.4.1. Image Features

Our multi-layered adversarial detection method requires image features that show systematic differences

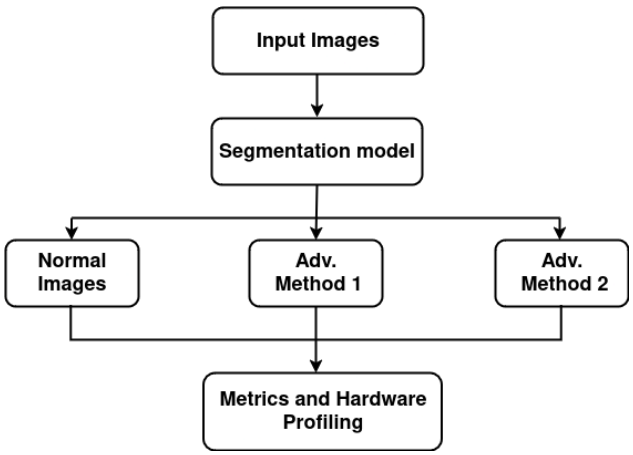


FIG 4. Round-Robin Image Inference

between normal and AEs. For this reason, we extract a set of features from each input image before inference. This ensures applicability to all types of images: normal, as well as black-box and white-box AEs. The features describe global intensity statistics, structural characteristics, frequency information, compression sensitivity, and local texture:

- `Metadata.` We extract basic image properties including width, height, area, file size, and number of channels, which describe the geometry and storage characteristics of the image.
- `Intensity statistics.` We compute the global mean and variance of pixel values, as well as per-channel mean and variance, summarizing brightness and contrast distributions.
- `Entropy.` Entropy is calculated from grayscale histograms to quantify the information content of pixel intensity distributions [21].
- `Structural features.` We extract the number of keypoints detected by the ORB (Oriented FAST and Rotated BRIEF) algorithm [22] and the density of edges obtained using the Canny detector [23], capturing local geometric structures.
- `Gradient statistics.` The mean, variance, and maximum gradient magnitudes are obtained from Sobel operators [24], summarizing local intensity transitions.
- `Frequency features.` We compute the mean, variance, and high-frequency energy of the Fourier magnitude spectrum [11] to describe spectral characteristics of the images.
- `Compression sensitivity.` JPEG recompression error [25] is measured as the mean squared difference between the original image and its JPEG-compressed version.
- `Texture descriptors.` Local Binary Patterns (LBP) are extracted with parameters $P = 8, R = 1$, and the normalized histogram is used to capture local texture patterns.

3.4.2. Hardware Features

To monitor the hardware metrics during the inference on the engine we used the Tegrastats utility, which is already integrated in the NVIDIA Jetson Orin. For each image input, we start the tegrastats in the background immediately before the inference and kill it immediately afterward, ensuring per-image measurement. Tegrastats enables us to capture device-level resource utilization, thermal behavior and power consumption. the device metrics are collected for a temporal resolution of 10 ms intervals and written into a log-file tagged with the method (normal or adversarial with the exact name the adversarial method) and the image ID to ensure traceability. Once the inference of the single image is complete, the system terminates the tegrastats process, waits a few seconds until the log file is finalized and non-empty, then parses its content to extract the metrics. After the parsing the log is deleted to minimize storage overhead because the limited memory capacity of the device does not allow long-term storage of intermediate logs. An example of a log entry generated by Tegrastats is shown in Figure 5.

Each line of the `Tegrastats` log file provides a snapshot of the system utilization. We extract four key groups of metrics from each sample:

(a) `Memory metrics`: Include `RAM_Used` and `RAM_Total` (in MB), the size of the largest free block (`RAM_LFB`), and swap-related statistics (`SWAP_Used`, `SWAP_Total`, `SWAP_Cached`).

(b) `CPU and GPU utilization`: Captured via the summed CPU core usage (`CPU`) and the Jetson GPU load percentage (`GR3D_FREQ`).

(c) `Thermal metrics`: Obtained from multiple onboard sensors, including `CPU_Temperature`, `GPU_Temperature`, `Tboard_Temperature`, `TJ_Temperature`, and SoC regions (`SOC0`, `SOC1`, `SOC2`).

(d) `Power consumption metrics`: Measure both instantaneous and average power draw for key voltage rails: `VDD_GPU_SOC`, `VDD_CPU_CV`, `VIN_SYS_5V0`, and `VDDQ_VDD2_1V8AO`.

Once the raw samples are extracted from the `tegrastats` log files, we compute both aggregate statistics and engineered hardware features for each image. The aggregate metrics consist of the mean and standard deviation of all numeric measurements, providing a compact summary of the device state during inference. To capture dynamic behavior, we derive additional engineered features such as the difference in GPU load (ΔGPU_Load), defined as the change in `GR3D_FREQ` between the start and end of inference, and the difference in GPU power (ΔGPU_Power), computed as the variation in instantaneous `VDD_GPU_SOC` power draw across the same window.

3.4.3. Model Profiling Features

In addition to image and hardware metrics, we also capture the model profiling internal behavior during inference of both normal and AEs. by analyzing these metrics, we can demonstrate that deviations in model behavior might serve as signal of adversarial pertur-

```
RAM  3242/15928MB (lfb 403x4MB)
SWAP 0/4095MB (cached 0MB)
CPU  [4%@1190, 6%@1190, 3%@1190, 2%@1190,
↪  1%@1190, 1%@1190]
GR3D_FREQ 62%
cpu@59C  tboard@35C  soc2@52C  tdiode@50C
↪  soc0@53C  gpu@55C  tj@57C  soc1@52C
VDD_GPU_SOC    1430mW/1200mW
VDD_CPU_CV      980mW/820mW
VIN_SYS_5V0    3200mW/2900mW
VDDQ_VDD2_1V8AO  90mW/80mW
```

FIG 5. Example output from NVIDIA Jetson's `tegrastats` utility showing real-time monitoring of memory usage, CPU/GPU load, temperatures, and power consumption during adversarial inference.

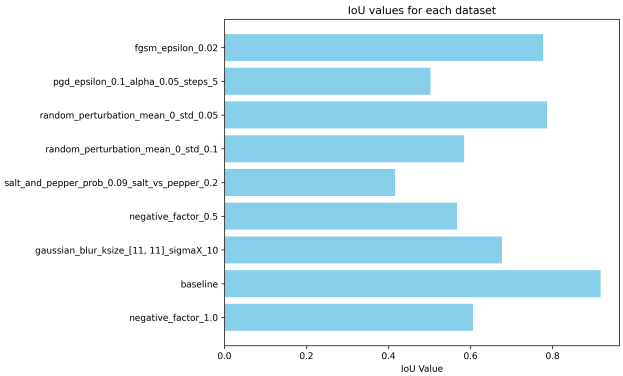


FIG 6. IoU of the ship segmentation model on datasets of 10,000 images each, including AEs generated with predefined perturbation parameters under white-box and black-box attacks. The baseline with normal images is shown for comparison to illustrate performance degradation under adversarial conditions.

bations. From each inference, the model's probability maps are used to derive descriptive statistics for every output channel:

- Mean probability: Average activation across the spatial map.
- Maximum probability: Peak activation within the map.
- Standard deviation of probability: Variability of activations.

In addition, we use PyTorch Profiler to obtain runtime profiling metrics:

- Operator-level CPU time (ms).
- Operator-level CUDA time (ms), when GPU is available.
- Total CPU time across all operations.
- Total CUDA time across all GPU kernels.

4. EXPERIMENTAL SETUP

4.1. Hardware and Experiment Tools

All experiments were performed on the NVIDIA Jetson AGX Orin Developer Kit, a high-performance edge computing platform designed for real-time AI workloads. The board features a 12-core ARM Cortex-A78AE CPU, an NVIDIA Ampere GPU with 2048 CUDA cores and 64 Tensor Cores, and 64GB of LPDDR5 memory. Storage is provided via 64 GB eMMC 5.1, which can be expanded with NVMe drives. The platform delivers up to 275 TOPS of AI performance in INT8 precision and supports configurable power modes at 15W, 30W, and 50W, allowing flexible trade-offs between energy consumption and performance [26]. The system ran Ubuntu 22.04 with NVIDIA JetPack 6, and a Docker container was used to provide the additional libraries required for the experiments[2].
An external microSD card was added to store initial datasets, results, and generated adversarial examples, as the onboard storage was insufficient for large image data. In our setup, we applied FP16 optimization, but observed a performance loss with the converted model. Therefore, we used the native PyTorch version of the model to maintain segmentation performance.

[2] https://github.com/dusty-nv/jetson-containers?tab=readme-ov-file

4.2. Evaluation Metrics

We evaluate the classification detection performance using the standard confusion-matrix terms: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). From these, we report F1, accuracy, precision, recall, and ROC-AUC [27, 28].

5. RESULTS AND ANALYSIS

5.1. Impact of Adversarial Attacks on Model Performance

In order to analyse the metrics that lead to the detection of AEs, we first assess the robustness of the segmentation model against these methods as shown in Figure 6. We created 10,000 images for each adversarial method and evaluated the performance of the model using them against the normal, uncrafted images. Each attack was carefully tuned to maximize the model misprediction while maintaining high visual fidelity, ensuring that the AEs remain difficult to distinguish from clean ones for human observers.
We measured the model's performance using IoU across all methods. We noticed that clean images have consistently high IoU values, which confirm that the model performs very well under normal conditions. On the other hand, the IoU drops with a severity depending on the type of attack for the AEs.

5.2. Results Analysis by Feature Type

In this section, we analyze the classification performance using each feature type. The goal is to assess,

how well the model distinguishes between clean (baseline) and adversarially perturbed images using different types of features: image-based, hardware-based, and model profiling-based.
For each classification task we have a balanced 20,000 images, 10,000 perturbed images we generated for each method, and 10,000 clean baseline images.
The dataset is split into 70% for training and 30% for testing. We perform this analysis independently for each perturbation method to understand how each feature type contributes to detecting different types of adversarial attacks and distortions.

5.2.1. Image-based Features

For the image-based features we noticed that the classification model achieves perfect separation on this dataset across all adversarial methods. The ROC curves consistently yield an AUC of 1.0, and this includes both white-box and black-box methods as shown in Figure 8.
To assess the discriminative influence of the different features of the image, we performed two distinct analyzes, their relative importance and their ability to differentiate the adversary method from the baseline images, as shown in Figure 7 and in Figure 10.
We noticed that across all methods the image-based features emerged as the most informative for the detection of the adversarial perturbation.
FGSM, PGD, Gaussian blur, random perturbations, and salt-and-pepper noise produce shifts in feature space that are clearly captured by image-based metrics. Features such as Local Binary Patterns (LBP) and frequency-domain statistics are particularly sensitive to changes in texture, smoothness, and high-frequency content.
However, methods with low-pixel-level perturbation, such as low-probability salt-and-pepper noise or "negative factor" transformations, often leave image features relatively intact, and detection relies more on global intensity/channel statistics as shown in Table 2.

Feature	Perturbation Methods
freq_mean	Gaussian Blur, PGD, FGSM, Salt-and-Pepper
freq_std	PGD, FGSM, , Salt-and-Pepper
lbp_0	Blur, PGD, FGSM, Salt-and-Pepper
high_freq_energy	Gaussian Blur
jpeg_error	FGSM
channel_means_1	Negative
channel_means_2	Negative
channel_means_3	Negative

TAB 2. Feature–perturbation mapping: for each feature, the perturbation methods under which it is most informative.

In summary, the feature-importance analysis across perturbations reveals two major patterns:

- First Pattern: For the Perturbation methods like Gaussian blur (= 5, 10), PGD (= 0.1, = 0.05), FGSM (= 0.02), random noise (std = 0.05, 0.1) and salt-and-pepper noise (prob = 0.01, 0.09). We can find that the most discriminative features are :
  `Frequency features`: `freq_mean`, `freq_std`, and `high_freq_energy`, derived from the Fourier magnitude spectrum of the image. These features indicate changes in the spectral structure of the image.
  `JPEG re-compression features`: capture compression artifacts that differ between adversarial (AE) and non-perturbed images. This feature is especially informative for the FGSM attack.
  These results show that pixel-level changes affect spectral characteristics and local texture; accordingly, `Frequency features` and `Texture descriptors` dominate importance, with `Compression sensitivity` particularly relevant for FGSM.
- Second Pattern: In case of the Negative attack with two different parameters (factor = 0.5, 1.0), the intensity statistics were deterministic, per-channel mean intensities (channel_means_1, channel_means_2, channel_means_3) enable us to detect these kind of attacks . Thus Intensity statistics dominate, while texture/frequency features are less affected.

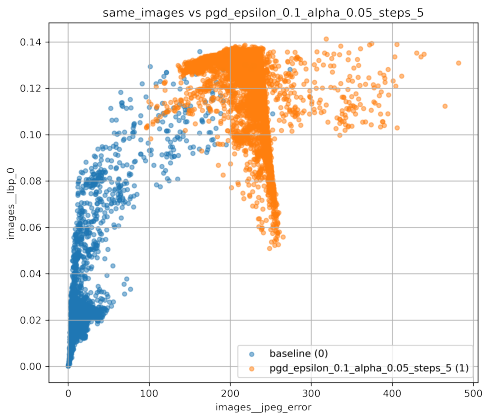


FIG 7. Scatter plot of the two most discriminative features for PGD attacks. Baseline images are shown in blue, AEs in orange. The clusters demonstrate near-perfect separation, highlighting the effectiveness of image-derived features in capturing adversarial perturbations.

5.2.2. Hardware-based Features

The performance of hardware metrics alone does not exceed near 0.5 AUC, for example, in the FGSM and PGD cases as shown in Figure 8, and this trend persists across all types of AEs, including white-box and black-box, which indicates random guessing as shown in Table 3. These results show that the collected hardware metrics (CPU/GPU consumption, power draw (W), temperature, etc.) alone do not capture artifacts that correlate with the existence of perturbed inputs.
We can conclude that system resource usage gathered using the Tegrastats tools does not capture significant hardware indicators, since metrics like CPU%, GPU%, and memory remain nearly similar between normal

Method	Source	tp	fp	tn	fn	Acc.	Prec.	Rec.	F1	ROC-AUC
FGSM ($\epsilon = 0.02$)	hardware	1430	1458	1542	1570	0.495	0.495	0.477	0.486	0.493
FGSM ($\epsilon = 0.02$)	images	2994	14	2986	6	0.997	0.995	0.998	0.997	1.000
FGSM ($\epsilon = 0.02$)	profiling	1733	724	2276	1267	0.668	0.705	0.578	0.635	0.710
Gaussian blur (11×11, $\sigma = 10$)	hardware	1436	1530	1470	1564	0.484	0.484	0.479	0.481	0.484
Gaussian blur (11×11, $\sigma = 10$)	images	3000	3	2997	0	1.000	0.999	1.000	1.000	1.000
Gaussian blur (11×11, $\sigma = 10$)	profiling	1788	1468	1532	1212	0.553	0.549	0.596	0.572	0.568
Gaussian blur (7×7, $\sigma = 5$)	hardware	1494	1448	1552	1506	0.508	0.508	0.498	0.503	0.500
Gaussian blur (7×7, $\sigma = 5$)	images	2996	3	2997	4	0.999	0.999	0.999	0.999	1.000
Gaussian blur (7×7, $\sigma = 5$)	profiling	1606	1521	1479	1394	0.514	0.514	0.535	0.524	0.530
Negative (factor=0.5)	hardware	1420	1481	1519	1580	0.490	0.489	0.473	0.481	0.487
Negative (factor=0.5)	images	2997	13	2987	3	0.997	0.996	0.999	0.997	1.000
Negative (factor=0.5)	profiling	1500	1424	1576	1500	0.513	0.513	0.500	0.506	0.521
Negative (factor=1.0)	hardware	1507	1520	1480	1493	0.498	0.498	0.502	0.500	0.491
Negative (factor=1.0)	images	2973	19	2981	27	0.992	0.994	0.991	0.992	1.000
Negative (factor=1.0)	profiling	1464	1353	1647	1536	0.518	0.520	0.488	0.503	0.528
PGD ($\epsilon = 0.1$, $\alpha = 0.05$, steps=5)	hardware	1422	1447	1553	1578	0.496	0.496	0.474	0.485	0.489
PGD ($\epsilon = 0.1$, $\alpha = 0.05$, steps=5)	images	3000	7	2993	0	0.999	0.998	1.000	0.999	1.000
PGD ($\epsilon = 0.1$, $\alpha = 0.05$, steps=5)	profiling	2817	750	2250	183	0.844	0.790	0.939	0.858	0.889
Random perturbation ($\mu = 0$, $\sigma = 0.05$)	hardware	1461	1460	1540	1539	0.500	0.500	0.487	0.493	0.507
Random perturbation ($\mu = 0$, $\sigma = 0.05$)	images	2999	6	2994	1	0.999	0.998	1.000	0.999	1.000
Random perturbation ($\mu = 0$, $\sigma = 0.05$)	profiling	1541	1408	1592	1459	0.522	0.523	0.514	0.518	0.530
Random perturbation ($\mu = 0$, $\sigma = 0.1$)	hardware	1447	1458	1542	1553	0.498	0.498	0.482	0.490	0.492
Random perturbation ($\mu = 0$, $\sigma = 0.1$)	images	3000	1	2999	0	1.000	1.000	1.000	1.000	1.000
Random perturbation ($\mu = 0$, $\sigma = 0.1$)	profiling	1670	1038	1962	1330	0.605	0.617	0.557	0.585	0.654
Salt–pepper ($p = 0.01$, $s{:}p = 0.1$)	hardware	1434	1468	1532	1566	0.494	0.494	0.478	0.486	0.490
Salt–pepper ($p = 0.01$, $s{:}p = 0.1$)	images	2994	9	2991	6	0.998	0.997	0.998	0.998	1.000
Salt–pepper ($p = 0.01$, $s{:}p = 0.1$)	profiling	1404	1080	1920	1596	0.554	0.565	0.468	0.512	0.587
Salt–pepper ($p = 0.09$, $s{:}p = 0.2$)	hardware	1433	1438	1562	1567	0.499	0.499	0.478	0.488	0.501
Salt–pepper ($p = 0.09$, $s{:}p = 0.2$)	images	3000	0	3000	0	1.000	1.000	1.000	1.000	1.000
Salt–pepper ($p = 0.09$, $s{:}p = 0.2$)	profiling	2245	263	2737	755	0.830	0.895	0.748	0.815	0.891

TAB 3. Classification metrics for baseline–vs–attack detection across three feature sources (images, hardware, profiling). For each method we use a balanced set of 10000 adversarial and 10000 normal images. The Classifier is trained with a 70/30 split (14000 train / 6000 test), and metrics are reported on the test set. The green highlighted rows show the best indicator features type per method.

and adversarial examples. We hypothesize that this is primarily due to the following reasons:
(i) The measurement level: the Tegrastats tool generates hardware statistics for the entire Jetson Orin system, which runs a full Linux environment, and not at a process level for only running the segmentation inference. This means that all processes running on the Linux OS are included; for example, kernel tasks and system daemons in the background also contribute to the measurement. As a result, the aggregated hardware metrics incorporate multiple sources of activity rather than being isolated to the ship segmentation model inference.
(ii) Weak AE signal: AE perturbations include only minimal and subtle changes to the images and, as a consequence, to the computation graph of the model. These perturbations may cause very marginal variations at the hardware level (CPU/GPU/memory/power), which can be easily masked by natural fluctuations caused by the background processes of the OS. In other words, the AE signal at the hardware level is weaker than the noise of the system-wide hardware utilization.

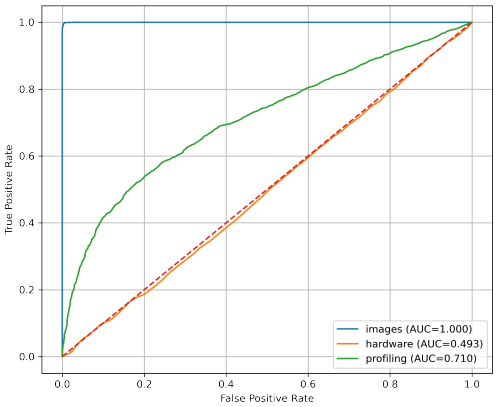
5.2.3. Segmentation-based Features

Segmentation model features show moderate to strong discriminative performance depending on the adversarial method and the parameter.

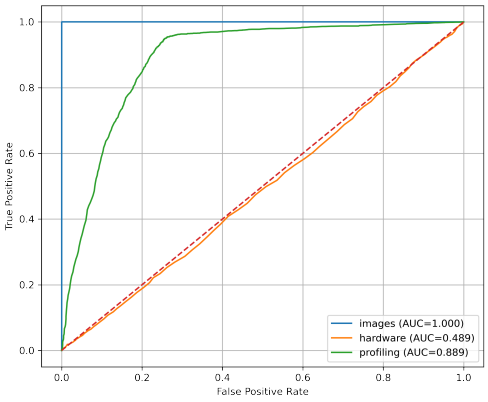
We noticed as an example for methods like PGD, FGSM the model profiling-based classifier achieves AUCs of 0.889 and 0.710 as shown in Figure 8. For the Salt-and-Pepper method with parameters (probability = 0.09, salt vs. pepper ratio = 0.2), the classifier achieves an AUC of 0.891, but for the same method with parameters (probability = 0.01, salt vs. pepper ratio = 0.1), it achieves only 0.587 AUC as shown in Figure 9. This indicates that input perturbations affect the model runtime behavior or execution patterns.
The discriminative profiling features allowing the best results of these methods (PGD, FGSM, and Salt-and-Pepper method with parameters (probability = 0.09, salt vs. pepper ratio = 0.2)) correspond to the statistics derived from the segmentation model probability map as shown in Figure 10. The features extracted using the PyTorch profiling, which correspond to the layer operation CPU/GPU consumption and execution time, were less informative.

6. CONCLUSION AND FUTURE WORK

Our study demonstrates that AEs pose a real risk to segmentation models, even when these models achieve strong baseline performance on clean satellite ship images. Our experiments confirmed that small perturbations crafted with black-box and white-box attacks degrade the segmentation performance, with a marked

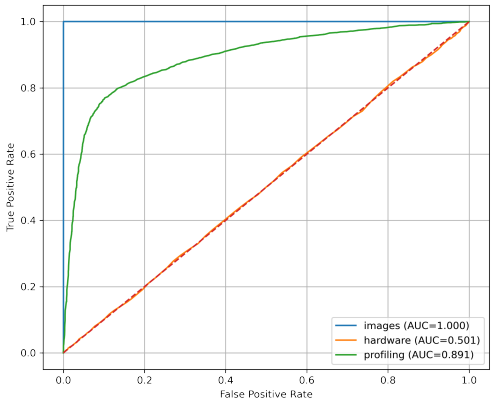


(a) ROC curve for detecting FGSM adversarial examples (= 0.02) using image-based, model profiling-based, and hardware-based features.

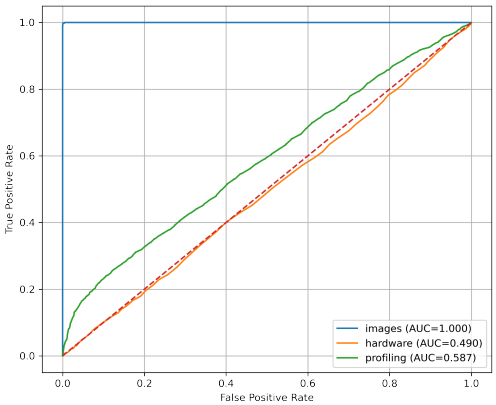


(b) ROC curve for PGD adversarial examples (= 0.1, = 0.05, steps = 5) using image-based, , model profiling-based, and hardware-based features.

FIG 8. ROC curves comparing the detection performance of image-based, hardware-based, and profiling features on adversarial examples generated by (a) FGSM (= 0.02) and (b) PGD (= 0.1, = 0.05, steps = 5). The curves demonstrate the effectiveness of the three feature types in separating normal and AEs.



(a) ROC curve for Salt-and-Pepper noise (probability = 0.09, salt vs. pepper ratio = 0.2) with AUC = 0.891.



(b) ROC curve for Salt-and-Pepper noise (probability = 0.01, salt vs. pepper ratio = 0.1) with AUC = 0.587.

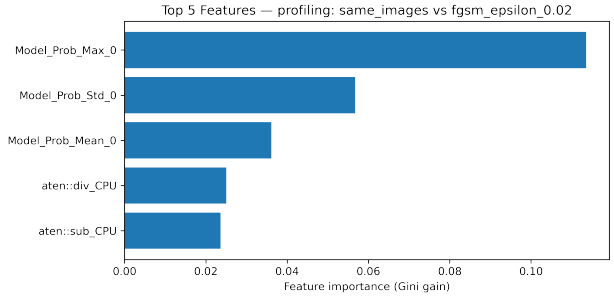
FIG 9. ROC curves showing the model profiling-based classifier's ability to distinguish clean images from adversarial examples generated using Salt-and-Pepper noise with two different parameter settings. Subfigure (a) corresponds to perturbations with probability = 0.09 and salt vs. pepper ratio = 0.2, yielding a high AUC of 0.891. Subfigure (b) shows performance for a lower noise level (probability = 0.01, ratio = 0.1), resulting in a reduced AUC of 0.587. The comparison highlights the impact of perturbation strength and distribution on the runtime behavior captured by profiling features.

IoU reduction depending on the attack and the perturbation degree.

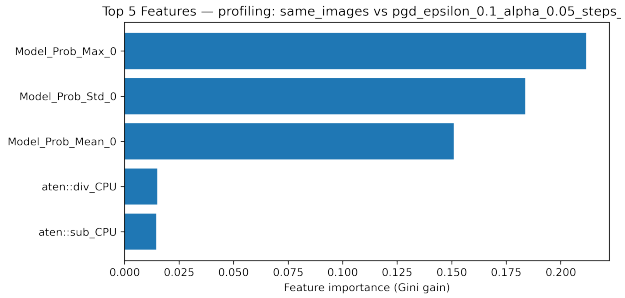
Concerning AEs detection, our results highlight the following findings: (i) The image-based features are the most reliable indicators of AEs. (ii) model profiling-based features offer complementary signals for some attacks. (iii) Using the Tegrastats tool for hardware metrics provides very little discriminative information about AEs.

The Tegrastats output might be noisy and variable because of background processes, since it is especially designed to capture running metrics of the whole Jetson Orin AGX, not specifically a single running process. This might result in the fact that the noise masks any weak signals from the adversarial perturbations.

Based on this hypothesis, future work could leverage more fine-grained tools that isolate hardware metrics only for the inference process and reduce the impact of noise caused by other resources.



(a) Feature importance for FGSM attack with $\epsilon = 0.02$.



(b) Feature importance for PGD attack with ($\epsilon = 0.1$, $\alpha = 0.05$, steps=5)

FIG 10. Top 5 most important profiling features for distinguishing clean versus adversarial examples.

ACKNOWLEDGMENTS

We thank Protim Bhattacharjee for providing the segmentation model used in this study.

References

[1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019. https://arxiv.org/abs/1706.06083.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. https://arxiv.org/abs/1412.6572.

[3] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In Proceedings 2018 Network and Distributed System Security Symposium, NDSS 2018. Internet Society, 2018. DOI: 10.14722/ndss.2018.23198.

[4] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations, 2017. https://arxiv.org/abs/1702.04267.

[5] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning, 2017. https://arxiv.org/abs/1602.02697.

[6] Ahmed Aldahdooh, Wassim Hamidouche, Sid Ahmed Fezza, and Olivier Déforges. Adversarial example detection for dnn models: a review and experimental comparison. Artificial Intelligence Review, 55(6):4403–4462, Jan. 2022. ISSN: 1573-7462. DOI: 10.1007/s10462-021-10125-w.

[7] Manaar Alam and Michail Maniatakos. Advhunter: Detecting adversarial perturbations in black-box neural networks through hardware performance counters. In Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN: 9798400706011. DOI: 10.1145/3649329.3655682.

[8] Preet Derasari, Siva Koppineedi, and Guru Venkataramani. Can hardware performance counters detect adversarial inputs? In 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), pages 945–948, 2020. DOI: 10.1109/MWSCAS48704.2020.9184684.

[9] Tommaso Zoppi and Andrea Ceccarelli. Detect adversarial attacks against deep neural networks with gpu monitoring. IEEE Access, 9:150579–150591, 2021. DOI: 10.1109/ACCESS.2021.3125920.

[10] NVIDIA Corporation. nvidia-smi - NVIDIA System Management Interface program. NVIDIA Corporation, Sept. 2025. https://docs.nvidia.com/deploy/nvidia-smi/index.html.

[11] Paula Harder, Franz-Josef Pfreundt, Margret Keuper, and Janis Keuper. Spectraldefense: Detecting adversarial attacks on cnns in the fourier domain, 2021. https://arxiv.org/abs/2103.03000.

[12] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Li Chen, Michael E. Kounavis, and Duen Horng Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression, 2017. https://arxiv.org/abs/1705.02900.

[13] Habibur Rahaman, Atri Chatterjee, and Swarup Bhunia. Runtime detection of adversarial attacks in ai accelerators using performance counters, 2025. https://arxiv.org/abs/2503.07568.

[14] inversion, Jeff Faudi, and Martin. Airbus ship detection challenge. https://kaggle.com/competitions/airbus-ship-detection, 2018. Kaggle.

[15] NVIDIA. Tensorrt. https://developer.nvidia.com/tensorrt-getting-started.

[16] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam.

Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018. https://arxiv.org/abs/1802.02611.

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. https://arxiv.org/abs/1405.0312.

[18] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations, 2019. https://arxiv.org/abs/1903.12261.

[19] Hossein Hosseini and Radha Poovendran. Deep neural networks do not recognize negative images, 2017. https://api.semanticscholar.org/CorpusID:17746324.

[20] Shengzhong Liu, Shuochao Yao, Xinzhe Fu, Rohan Tabish, Simon Yu, Ayoosh Bansal, Heechul Yun, Lui Sha, and Tarek Abdelzaher. Taming algorithmic priority inversion in mission-critical perception pipelines. Commun. ACM, 67(2):110–117, Jan. 2024. ISSN: 0001-0782. DOI: 10.1145/3610801.

[21] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. IEEE Transactions on Dependable and Secure Computing, 18(1):72–85, Jan. 2021. ISSN: 2160-9209. DOI: 10.1109/tdsc.2018.2874243.

[22] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In 2011 International Conference on Computer Vision, pages 2564–2571, 2011. DOI: 10.1109/ICCV.2011.6126544.

[23] John Canny. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6):679–698, 1986. DOI: 10.1109/TPAMI.1986.4767851.

[24] Irwin Sobel and G. M. Feldman. An isotropic 3×3 image gradient operator, 1990. https://api.semanticscholar.org/CorpusID:59909525.

[25] Zihao Liu, Qi Liu, Tao Liu, Yanzhi Wang, and Wujie Wen. Feature distillation: Dnn-oriented JPEG compression against adversarial examples, 2018. http://arxiv.org/abs/1803.05787.

[26] NVIDIA. Jetson orin: Embedded systems for autonomous machines. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/. Accessed: 2025-08-26.

[27] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. Information Processing Management, 45(4):427–437, 2009. ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2009.03.002.

[28] Tom Fawcett. An introduction to roc analysis. Pattern Recognition Letters, 27(8):861–874, 2006. ISSN: 0167-8655. ROC Analysis in Pattern Recognition. DOI: https://doi.org/10.1016/j.patrec.2005.10.010.