Satellite Image Classification

Chinmay Brahmbhatt, Logan Sehr, Gurpreet Singh
Computer Vision I, Spring 2025

1 Abstract

Automated classification of small (80 x 80 pixels) satellite image patches as ship and non-ship is essential for scalable maritime surveillance in light of the impossibility of manual inspection at a very large scale. We build a balanced dataset of 6.7K patches of different satellite images, then use 512-dimensional embeddings from a frozen, pretrained ResNet-18 and train two light-weight and efficient classifiers (kNN with k=2 for binary classification and linear SVM) on these features. Similarly, in the interest of using contemporary architectures, we also leverage a contemporary Vision Transformer architecture to provide comparison to State of the Art (SOTA) architectures in Computer Vision. Our experiments demonstrate that these non-parametric classifiers on deep embeddings yield good ship versus non-ship patch separation without end-to-end backbone fine-tuning. The hybrid pipeline significantly reduces training complexity and inference cost, making near-real-time maritime surveillance feasible on resource-constrained hardware.

2 Introduction

2.1 Maritime Traffic and Port Congestion

Global maritime shipping carries over 80 percent of world trade by volume, and ports play critical nodes within the supply chain [1]. As vessels queue outside packed harbors each day of idleness can represent millions of dollars in extra fuel costs, demurrage charges, and time lost [2]. Such congestion not only slows down the flow of goods but also contributes to emissions and strains port infrastructure, with spillover impacts across the world economy.

2.2 Overall Objective: Automating Congestion Management

Our ultimate long-term goal is to create an automated system capable of tracking vessel locations, identifying incipient patterns that induce congestion, and warning port authorities in real time, ultimately diminishing waiting times, berth damage, and environmental impacts. This vision can be fulfilled by interweaving several computer vision tasks: detecting ships in satellite images, tracking their trajectory over time, and evaluating spatial patterns to forecast queue development.

2.2.1 Sub-Goal: Binary Ship Classification

As a precursor to congestion management, the first task is to solve the simpler but critical problem of binary classification: for a small image patch, determine whether it contains any part of a ship. Reliable ship vs non-ship detection is a stepping stone to more advanced objectives like object localization, counting, and trajectory analysis. If unoccupied ocean areas cannot consistently be excluded, downstream congestion-prediction and tracking models will exhibit large false-positive rates, rendering the entire pipeline questionable.

3 Related Work

3.1 Convolutional Neural Networks (CNN)

Convolutional Neural Network (CNN) architectures have been widely applied in other research. The ResNet-18 is popular among CNNs due to its strong ability to balance performance with complexity. In a paper by Qasem A. Al-Haija et al. [3], they examine how a pretrained ResNet-18 performs on classifying weather autonomously in IoT devices. In their approach, the output tensor of the ResNet-18 is fed through a classification module that utilizes the SoftMax activation function. The results of their data flow outperformed the accuracy of previous papers that tackled the problem with different architectures [3], showing strong performance from the use of the ResNet-18 in image classification.

In contrast, our work modifies the ResNet-18 architecture by removing the final block and instead feeds it into an additional classifier, with that being either a k-Nearest Neighbor (kNN) or Support Vector Machine (SVM), respectively. This final classifier will perform the binary classification on the reduced output from the CNN. This allows us to evaluate how well these less complex classifiers take on the extracted features from the pretrained ResNet-18.

3.2 Vision Transformers (ViT)

Vision Transformers (ViT) have also shown promise in image classification and have been developed to evaluate their performance on satellite imagery. Work has been done to develop fully-attentional models specific to this imagery by Michail Tarasiou et al. [4]. The group developed the Temporo-Spatial Vision Transformer (TSViT). The architecture processes the spatial and temporal information of images by extracting the patch tokens like a standard ViT, and then passing them through encoders that explicitly separate the tokens by time and class. This ability to dynamically capture satellite information has led to impressive performance in segmentation and classification tasks [4]. However, alongside the performance of this architecture, it has some noticeable constraints as it can struggle to handle large inputs and is hardware-intensive due to the operations performed in the temporal and spatial decoders [4]. Our particular work leverages the comparatively standard implementation of Vision Transformers from the original "An Image is Worth 16×16 Words" paper by Alex Dosovitskiy et al. [5].

3.3 Model Comparison (CNN vs ViT)

As we are implementing our own versions of the aforementioned architectures, it is good to calibrate expectations of performance to past research. In a paper comparing the performance of CNN vs ViT, Jose Mauricio et al. discuss the differences in robustness, performance, and evaluation [6]. These spaces where ViTs excel and CNNs are limited (and vice versa) will help visualize the pros and cons of these architectures in the scope of our work. It will also allow the ability to observe if the results are consistent with the related works and to find contradictions or new findings.

Many of the CNN strengths reinforce what was learned in lecture, with the efficiency and scalability being highlighted as key features, and being robust against augmented samples. Other strengths highlighted the many established architecture variants, mentioning ResNet as one with proven effectiveness in various image classification tasks [3, 6]. The limitations of CNNs focus on their inherent bias towards specific datasets by the architecture, and their struggles in capturing long-range context information. The performance can also suffer when the samples are high-resolution images [6]. This problem is avoided in our work, however, as our data is of low resolution (80 x 80 Pixels).

ViTs thrive in areas where CNNs are challenged, as they are able to effectively capture global contextual information through their self-attention mechanism [4, 6]. The scalability of ViTs is also highlighted, alongside flexibility in accommodating different input sizes, patch sizes, and depth. However, the lack of evaluation benchmarks for ViTs (likely contributed to by its youth) can make standardizing metrics more difficult [6]. Additionally, the computational costs of the architecture are considerably higher than CNNs, which could lead to issues in resource-strained environments due to hardware or memory limitations.

4 Data Preprocessing

4.1 Dataset Acquisition and Initial Setup

The dataset of choice was the "Ships in Satellite Imagery" dataset from Kaggle, which could be downloaded using the Kaggle API [7]. After authenticating, the dataset is downloaded and unzipped to a local directory. The dataset includes a JSON index file and a patches directory of satellite image patches, each 80×80 pixels in PNG format. To make the codebase environment-agnostic, important configuration parameters such as dataset path, output path, and batch size are stored in a .env file. These are dynamically imported using the dotenv package, so that reconfiguration with ease without modifying code is feasible. A utility function also makes sure required subdirectories in the root output path exist when required.

4.2 Label Extraction and Image Categorization

All image file names embed its class label in a fixed format: a binary label (0 for non-ship, 1 for ship), followed by a double underscore and a unique identifier. The images are loaded with this convention and categorized into two groups depending on whether they are from the ship or non-ship images. Full file paths to all images are collected and stored in two lists (ships and nonships), and these are utilized for all further processing procedures.

4.3 Image Loading and Normalization

To preprocess the data for training, each image is loaded with the PIL library and transformed into a PyTorch tensor. This is achieved through a custom load_image() function using a composed transform pipeline with torchvision.transforms.v2. The transform initially converts the image to tensor form with ToImage(), and then normalizes the pixel intensities from the range [0, 255] to [0.0, 1.0] with ToDtype(torch.float32, scale=True). This operation ensures all the images are numerically normalized and ready for consumption by models.

4.4 Data Splitting and Set Construction

Once the labeled image paths are organized, the dataset is divided into three distinct sets to enable proper model evaluation. A deterministic splitting approach is used to maintain class balance:

- The first 250 images from each class are used for the **test set**.
- The next 100 images from each class are allocated to the **validation set**.
- All remaining images are used for the **training set**.
  - 650 **Ships** images are augmented to produce $3K$ samples from the remaining ship images

– $2.65K$ **Non-Ships** images are augmented to produce $3K$ samples from the remaining non-ship images.

The explicit control over partitioning ensures that model evaluation is reliable and unbiased. The augmentations used are described in 4.5.

4.5 Data Augmentation and Class Balancing

In order to have the training set populated with a balanced and adequate number of samples from both the ship and non-ship classes, the other remaining ship and non-ship images are augmented to a point where each of the classes has exactly 3,000 images. This is done through the pad_images_with_augmentations() function, which uses transformation methods like random rotations (0–360 degrees), horizontal and vertical flipping, Gaussian blur, and color jittering. An augment function, augment_image(), iterates over these methods based on the current iteration number, evenly distributing augmentation types across the data set.

Not only does this method grow dataset size, but it also brings diversity to limit model overfitting. Re-balancing the dataset also provides an equal and valid representation of the problem while critically ensuring that a random guess remains at a 50/50 chance of being correct or incorrect. Hence, subsequent evaluation metrics don't present any innate skew from the dataset.

4.6 Dataset Wrapping and Label Assignment

After finding out all three sets, labels are explicitly assigned to each image tensor: 1 for the ships and 0 for non-ships. These are cached as (image_tensor, label) pairs and wrapped in a custom PyTorch dataset class ShipsDataset, which supplies the standard __len__() and __getitem__() methods. This class makes them fully PyTorch's data-loading pipeline compatible and provides optional transformation hooks for future expansion.

4.7 Serialization and DataLoader Initialization

To avoid each iteration of redoing the entire preprocessing pipeline, the preprocessed train, validation, and test sets are serialized and dumped to disk in .pt files. If these files already exist, they are loaded in directly by calling torch.load(). Finally, each dataset is passed into a DataLoader object with parameters driven by the batch size set in the environment variables. Unlike time-series data, image data benefits from providing a practical no-cost format for introducing variability in the data using random sampling. Hence, all three loaders are shuffled in order to provide variability in the data.

5 Methods and Methodology

5.1 CNN - ResNet 18 Approach

This method used a hybrid classical machine learning and deep learning technique for binary classification of ship vs. non-ship satellite images. The pipeline included a fixed feature extractor, two classifiers (Support Vector Machine and k-Nearest Neighbors), and a strict evaluation framework with standard metrics and confusion matrix visualizations.

5.1.1 Feature Extraction

The feature extraction module was derived from a pre-trained ResNet-18 model in torchvision.models. To obtain a compact feature representation, the final fully connected layer was removed, leaving a network that produces a 512-dimensional feature vector for every image. To

ensure consistent and reproducible results, all backbone parameters were frozen by setting the model to evaluation mode (.eval()), and gradient computation was turned off. The output of the network was a tensor of size $[B, 512, 1, 1]$, which was flattened into a 2D tensor $[B, 512]$ prior to inputting it into downstream classifiers. To clarify, the value B represent the batch size of the input. Batching the input enables to model to converge faster and learn more about the data concurrently.

5.1.2 Classifier 1 – k-Nearest Neighbors

The first classifier employed was a k-Nearest Neighbors (kNN) classifier. Euclidean distances were then computed for the feature vector of each test image against all vectors in the training set using `torch.cdist`. The two closest neighbors (k = 2) were selected, and a majority vote was taken to conclude the predicted class. This classifier was built completely inside PyTorch in order to exercise end-to-end control over data movement and tensor computations.

5.1.3 Classifier 2 – Linear Support Vector Machine (SVM)

In parallel, we also trained a linear Support Vector Machine (SVM) using scikit-learn. The 512-dimensional feature vectors extracted from the training set were presented as input to the SVM, along with their binary labels. The SVM learned a linear decision boundary that maximizes the margin between the two classes in this high-dimensional space. Prediction on the test set was performed using the same extracted features.

5.1.4 Evaluation Strategy

For testing, features were extracted from all images in both the `train_loader` and `test_loader`, generating tensors for training and for testing. SVM was trained on feature tensors transformed to NumPy, but k-NN classification was performed using in-memory PyTorch tensors and majority voting scheme. Both classifiers employed confusion matrices constructed with Seaborn. Green heatmap was used in the visualization of kNN results, while blue heatmap depicted the SVM.

5.2 Vision Transformer (ViT) Approach

This methodology leverages the more contemporary model architecture of a Vision transformer. The architecture is derived from the work primarily sourced from the transformer architecture within the NLP domain. Our work leverages the same model architecture defined within the "An Image is Worth 16x16 Words" by Alex Dosovitskiy et al [5]. Our particular approach leverages a variation of that particular implementation.

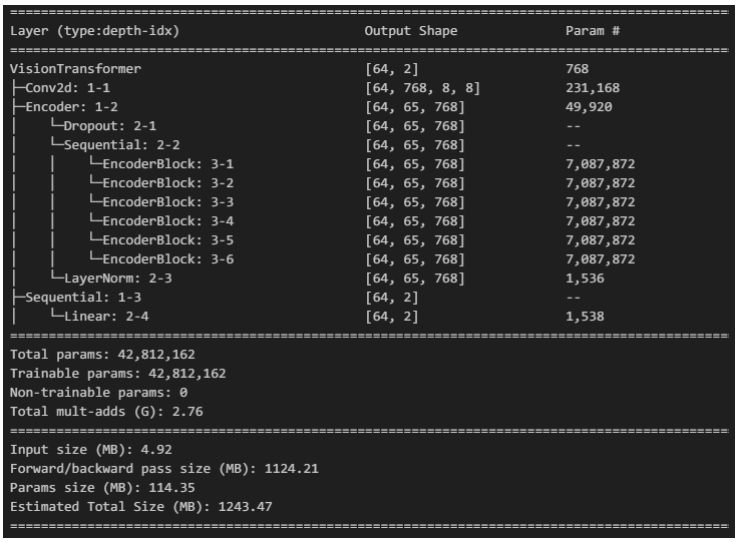
5.2.1 Model Specifications

Our particular model implementation leverages the core architecture of the vanilla Vision Transformer [5] with a different number of attention heads, patch size, number of layers, and different hyperparameters. We use a patch size of 10 to over the input image data of 80×80 to produce 8 unique patches to use within our model. More importantly, the model also uses two attention heads with the consideration of a binary classification. The relatively simplify of our task compared to other tasks validated the use of fewer attention heads on the idea of needing fewer features. Similarly, we also use an attention dropout and neuron dropout of 0.1 in the interest of avoiding overfitting and improving generalization. Our model leverages the same hidden internal dimension of 768 as well as the same 3072 multi-layer perceptron dimension as within the original paper for the ViT-Base model. Doing so provided us with a grounded implementation that is closer to contemporary while still being adequate for our task. In the interest of our

classification task being binary instead of mulit-class our final classification output from the final Linear layer was also reconfigured to output to two neural for a raw logit for each of the two classes.

The model uses Binary Cross entropy loss with logits as its loss function and the PyTorch implementation of the Adam optimizer as its optimizer during training.

Figure 1 - ViT Architecture



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
VisionTransformer                        [64, 2]                   768
├─Conv2d: 1-1                            [64, 768, 8, 8]           231,168
├─Encoder: 1-2                           [64, 65, 768]             49,920
│    └─Dropout: 2-1                      [64, 65, 768]             --
│    └─Sequential: 2-2                   [64, 65, 768]             --
│    │    └─EncoderBlock: 3-1            [64, 65, 768]             7,087,872
│    │    └─EncoderBlock: 3-2            [64, 65, 768]             7,087,872
│    │    └─EncoderBlock: 3-3            [64, 65, 768]             7,087,872
│    │    └─EncoderBlock: 3-4            [64, 65, 768]             7,087,872
│    │    └─EncoderBlock: 3-5            [64, 65, 768]             7,087,872
│    │    └─EncoderBlock: 3-6            [64, 65, 768]             7,087,872
│    └─LayerNorm: 2-3                    [64, 65, 768]             1,536
├─Sequential: 1-3                        [64, 2]                   --
│    └─Linear: 2-4                       [64, 2]                   1,538
==========================================================================================
Total params: 42,812,162
Trainable params: 42,812,162
Non-trainable params: 0
Total mult-adds (G): 2.76
==========================================================================================
Input size (MB): 4.92
Forward/backward pass size (MB): 1124.21
Params size (MB): 114.35
Estimated Total Size (MB): 1243.47
==========================================================================================
```

5.2.2 Evaluation Strategy

Due to complex nature of model, there was an evident expectation of overfitting. The model was trained for 100 epochs with a learning rate of 0.0001. The dataset's batch size was configured to 64 to improve the convergence time of the model. Our model managed to train within 9.79 minutes on a single Nvidia RTX 6000 Ada Generation GPU. Similarly, the model is evaluated for it's accuracy and loss computations across all three datasets with validation metrics being used for saving the optimal model.

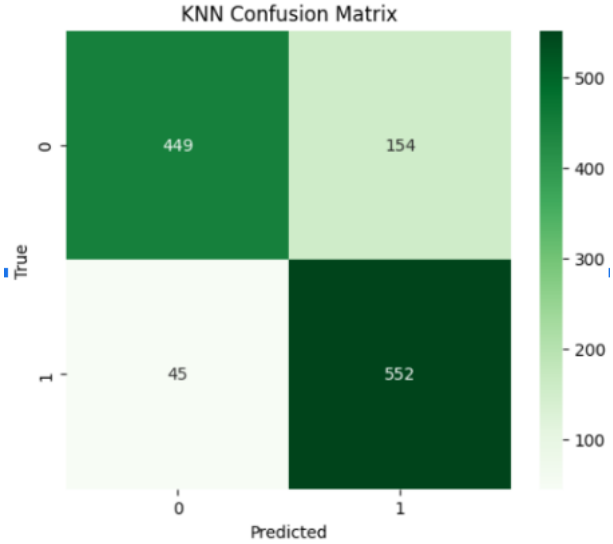
6 Interpretation of Output

6.1 CNN Results/Evaluations

The performance of our ResNet18 in this binary classification task was measured at multiple levels. A big focus point in our results was the performance of the different classifiers we attached to the end of our custom implementation, the kNN and SVM, respectively. Confusion Matrices were constructed for both classifiers to see how the model performed against the test set, along with additional graphs as we progressed through the implementation.

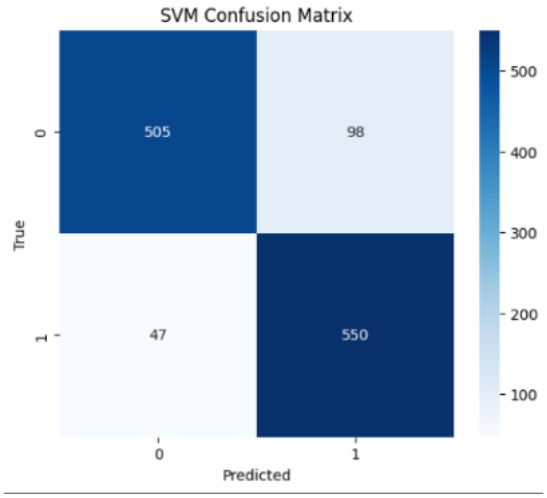
6.1.1 Preliminary Results

Figure 2 - ResNet18 + kNN Confusion Matrix



Our first iteration of the ResNet18 + kNN (with hyperparameters $k = 2$ and L2 Norm distance metric) performed with an average accuracy of 83%. However, you can observe from Figure **2** that the model was classifying non-ships as ships at a rate higher than intended.

Figure 3 - ResNet18 + SVM Confusion Matrix



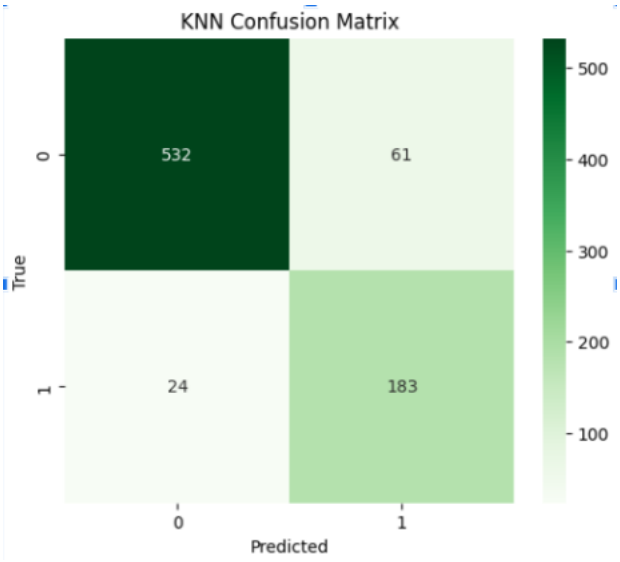
With ResNet18 + SVM, the model's average accuracy was a strong 88%. Although to a lesser degree, it is seen in Figure **3** that this combination also classifies too many false positives.

Our preliminary results were based on a preprocessing strategy that used an augment-before-split strategy. This method created scenarios where either the train, validation, or test sets could have disproportionately more augmented samples than the other sets, therefore manipulating the results. This prevented the model's ability to deal with augmentations consistently, often classifying ships as non-ships when the model was not accustomed to rotations or Gaussian blur.

6.1.2 Intermediary Results

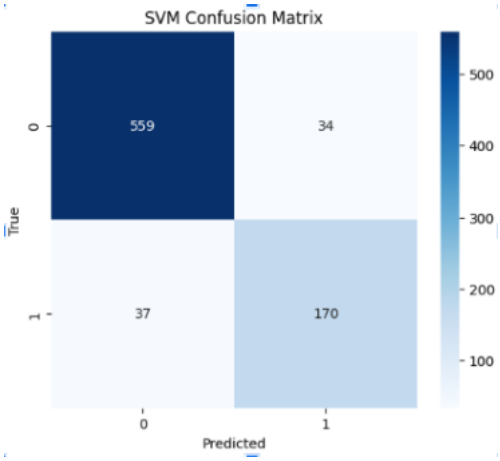
With bias and further issues that the augment-before-split strategy was producing, simple preprocessing changes were made. A change of strategy to split-before-augment was implemented, and we only augmented the training set.

Figure 4 - ResNet18 + kNN Confusion Matrix



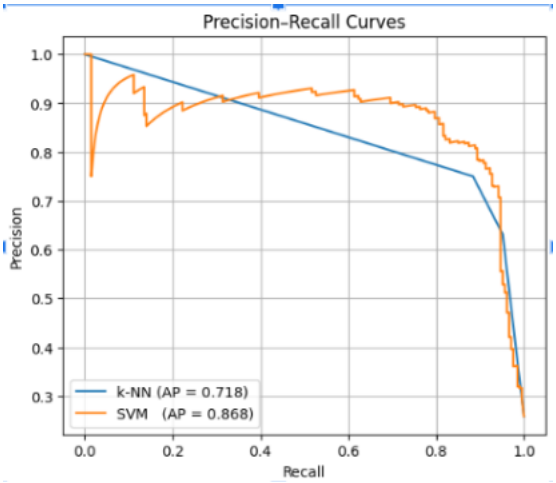
The efficiency of our ReNet18 + kNN saw considerable improvement, with an improved average accuracy of 89%. It also cut the amount of non-ship false positives in half consistently, which reinforced that the preprocessing changes were needed to remove bias from the data.

Figure 5 - ResNet18 + SVM Confusion Matrix



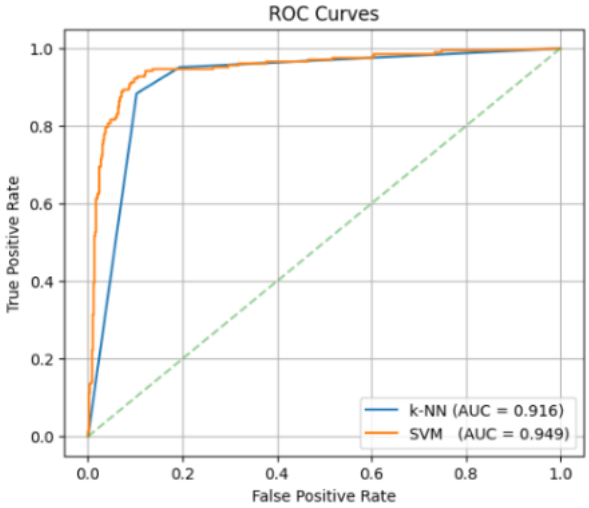
The SVM classifier saw a slight improvement, with an improved average accuracy of 91%. It saw a similar halving the kNN did for non-ship false positives. However, its rate of overlooking ships and classifying them as non-ships remained relatively unaffected by these changes.

Figure 6 - Precision-Recall Curves



At this intermediary stage, we introduced two new graphs. The precision and recall scores at this stage were not ideal. The biggest concern with kNN was the lower scores compared to SVM, with an average precision of 71%. SVM struggled with volatility, as Figure **6** shows how the correlation between the scores did not follow any clear pattern beyond the best cases.

Figure 7 - ROC Curves

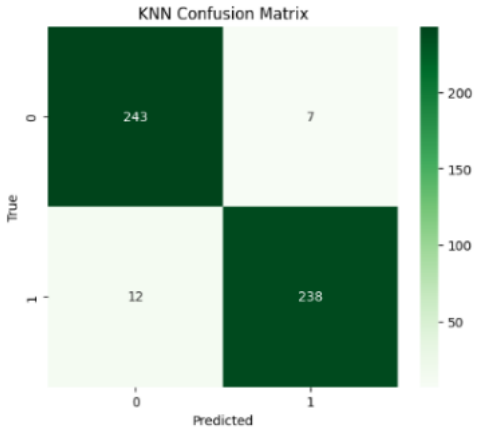


The ROC curves were stronger compared to the precision-recall curves, with both the kNN and SVM ResNet18 models limiting the false positives much better than in the preliminary results. The noticeable strength of these models can be observed with the AUC percentages, with both averaging an area over 90% across many runs.

While the model (and its chosen final classifier) were operating better with these changes, we felt there was still room for improvement.

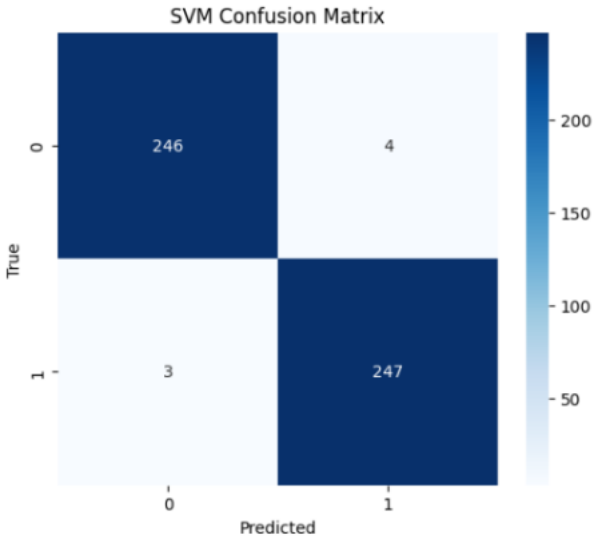
6.1.3 Final Results

Figure 8 - ResNet18 + kNN Confusion Martrix



On multiple iterations, the ResNet18 + kNN (with hyperparameters k = 2 and L2 Norm distance metric) performed with an average accuracy of 96%. You can observe from the provided confusion matrix (Figure **8**) that fewer than 30 of the 500 test samples were classified incorrectly based on their true labels.

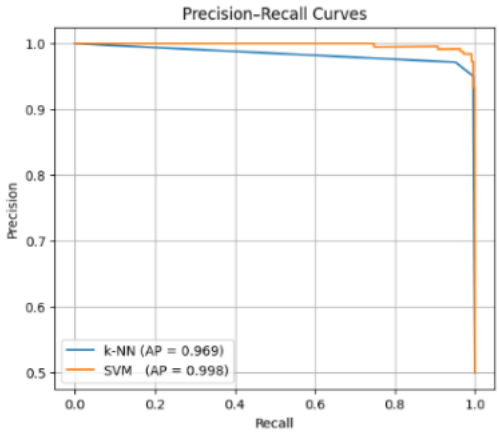
Figure 9 - ResNet18 + SVM Confusion Matrix



The ResNet18 + SVM also performed impressively, boasting an average accuracy of 98%. Similar to kNN, you can observe a minimal number of samples being classified incorrectly.

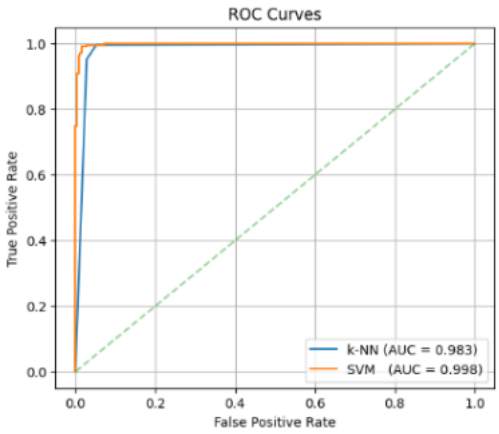
In this final iteration, the CNN model approach excels at the task of classifying ships vs non-ships. The performance of our model further supports analysis of the CNN architecture for image classification tasks [6].

Figure 10 - Precision-Recall Curves



The precision and recall scores for both kNN and SVM were very strong, likely attributed to the preprocessing to balance the dataset and the robustness of the model. The average precision of both final classifiers on our CNN exceeded 95% consistently.

Figure 11 - ROC Curves



The ROC curves for our ResNet18 with both ending classifiers showcased the minimal number of false positives, reinforced by our AUC percentages. Which were both consistently exceeding 97%.

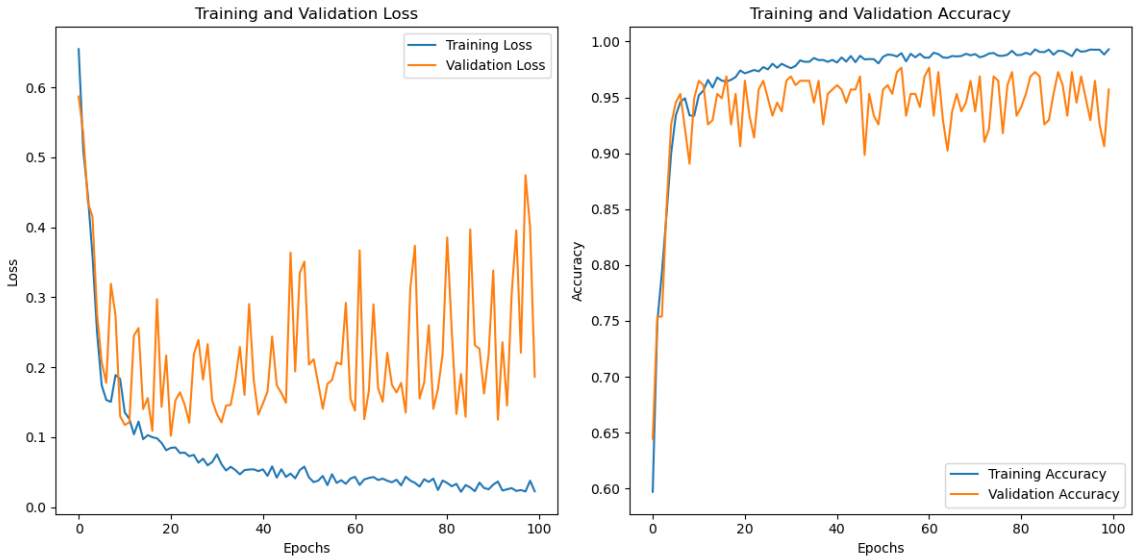
6.1.4 CNN Synopsis

The progression of our CNN architecture was one of positive growth, with each change made along the way (either to the model or preprocessing) only making the model more robust and increasing performance metrics. The performance of the model is promising in what it could contribute to future work and real-world implementation.

6.2 Vision Transformer Results/Evaluations

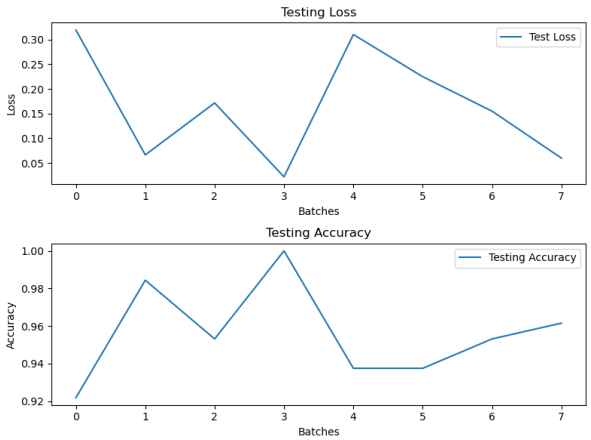
The Vision Transformer showed effective results with an average accuracy of 95% across the testing set. Similarly, the model's comparatively higher complexity enabled it to converge faster on the local minimum.

Figure 12 - Training & Validation: Loss & Accuracy



The figure above visualizes the overall performance of the model during training. More specifically, it's evident that model begins to overfit after around 20 epochs. Specifically, because of the discrepancy between the model's training and validation loss and the lack of convergence on the validation loss. Similarly, this is also evident within the validation accuracy where the training accuracy consistently converges and the validation accuracy proceeds to vary quite a bit.

Figure 13 - Testing: Loss & Validation



The model sees a general convergence across the training set as well. Given the imbalanced nature of the dataset and the intent to perform the least number of augmentations or modifications to the test set, the number of batches of overall size remained small. Hence, the number of batches to compute testing results over was relatively small. Our primary reason for this difference in set size was based on the intent of allocating as much training data as possible to enable the most learning within the models. The testing data also provide insight into the model overfitting onto the data. It's shows the variability of the loss and accuracy of the model and the lack of evident convergence, even on the small number of batches.

6.2.1 Vision Transformer Synopsis

The ViT provides a contemporary approach to the binary classification vision task. Although the model performed well, it also showed direct indicators of overfitting onto the data. The complex nature of the architecture provides valid reasoning for said overfitting and shows potential for the task being solvable with less complex approaches.

7 Further Discussion

In the related works section, we highlight a paper that directly compares CNNs and ViTs as a collective [6]; this helped us set up a baseline of expectations for our custom CNN and ViT models. We will use this section to further discuss how our findings either deferred or stayed consistent with the provided context.

Both approaches leverage contemporary works within the Machine Learning domain to achieve the same task. Although both also more specifically use Deep Learning using ResNets and convolutional layers, the approaches differ in the use of different classification techniques within the final stage. Such differences highlight both the computational and generalizable capabilities of each architecture. Considering such differences, it's important to consider other variations of SOTA architectures within ViTs as well as other Machine Learning approaches. Through our work, it was critical to consider the optimal architecture based on the domain of the dataset and the application. Leveraging the context of both the application and data domain proved that a simpler architecture of a ResNet feature extractor with either a kNN classifier or an SVM classifier provided highly comparable performance to that of the ViT. All while being much smaller in size and computational load.

As is common for ViT, convergence tends to take a longer time, whereas generalization tends to surpass over longer intervals of training. Although the ViT did not exhibit such direct comparisons, it is still valuable to perceive the generalization capabilities of the ViT to be further than those of the kNN or SVM models. Primarily sourced from the increased number of layers and the understood notion that scale correlates with model generalization and performance in Machine Learning.

As is specific to kNNs and SVMs, problems that are linear tend to be easily solved by said models [8]. That is to say, data that can be mapped to a linear or nearly linear space is easily classified by kNNs and SVMs. Hence, kNNs and SVM provided knowledge into the problem that the task was highly linear in nature. The ability for both approaches to converge on the optimal representation of the problem shows that the data itself presented a fairly non-complex and easily understood answer to the binary classification problem. Similarly, this also provided insight as to why the ViT didn't provide much additional performance for the task. It could be speculated that the ViT was able to learn additional non-useful features for the task on the basis that it leverages many more parameters, unique architectures (multi-head attention), or simply non-linear activation functions (ReLU) to provide non-linearity through the entire model. It's critical to note that both approaches leverage ResNet to map the data to different representations. Hence, it's also valid to consider that ResNet enabled effective non-linear mapping of the data features into a linear space. In either way, the lightweight implementation of ResNet + kNN or SVM provides a computational load advantage over a ViT.

8 Implementation and Future Work

8.1 Research Expansion

Based on the construction of our own CNN and ViT architectures and their positive results, the complexity of the task could be elevated by utilizing our models. Using our robustly constructed

models, we have set a strong baseline, having an accurate classification task for ships and non-ships. When satellite data is received, it will often be in images that cover a larger scope rather than a small area containing a single ship or non-ship entity. Future work to cover these large-scale images would be to create a robust object detection system. The classification aspect of this task would entail using our architecture, which has already proven to perform this task to a high degree of success.

Our architecture could be used in combination with well-known models for localizing and labeling objects in images. YOLOv8 could be paired with our CNN in a custom two-stage classification-to-detection system to create a model that can take an entire image and correctly classify and label all ships in satellite imagery, and remove uncertainty about ship-like terrain, producing false positives upon detection. This would upscale our research conducted on a singular level. We propose YOLOv8, as it is a very popular model in the realm of computer vision, used in many image and video-related tasks [9]. The biggest selling point of the architecture is its impressive speed and accuracy in real-time applications, balancing robustness and precision [9]. In our work, we also used many preprocessing techniques to denoise and improve the significance of the data. In addition to this, post-processing techniques could be used to handle the weather conditions of satellite images, like non-maximum suppression (NMS).

8.2 Real-World Implementation

The binary classification of satellite imagery is a powerful tool that helps automate and solve problems related to ports. Our focus on port congestion, as discussed in the Introduction (2), is a problem we want to solve. To solve this problem, our techniques can be deployed through or alongside a medium that allows us to apply this classification in real time.

Starlink, an international telecommunications company owned by SpaceX, has its focus for their satellites on providing broadband internet for straining tasks like streaming, gaming, etc. [10]. Much of the research and development regarding Starlink has revolved around passive radar [11]; however, should Starlink pursue visual tasks (beyond radio imagery), our image classification architecture could be used to kickstart any future research by the company.

A company where our work could be better applied would be Planet Labs and their SkySats. The company specializes in producing satellite-based data (imagery) and analytics [12]. Our data set is sourced from open source imagery from the company [7]; The proven performance of our models using a sample of the company's data would make the transition to large-scale collaboration or future work utilizing their data seamless.

Additionally, many projects with SkySats have been conducted, with an interesting one of note using satellite imagery provided to create mappings of Earth [13]. In partnership, our architecture would provide a service to private and public maritime companies, offering a different and that we propose as an improved solution to some of the current measures being taken.

9 Conclusion

Our research finds that the binary classification task on Satellite images is effectively solved using a ResNet + kNN or SVM implementation and a contemporary ViT implementation. However, our work further finds that both models provide comparable performance for this task. This is a major point since the two different approaches differ in computational overhead and complexity. More specifically, the ResNet + kNN or SVM implementation provides much lower computational overhead while performing at a comparable 95% and 98% accuracy, respectively, with effective representation of both classes. Comparably, the ViT transformer provided similar performance of 95% while having a larger overhead and a backpropagation size of about 1.1 GB. Hence, the work shows that simpler architectures of pre-trained ResNet with linear classification

heads like kNNs or SVM provide comparable and faster performance on the task of binary classification on satellite imagery for ships.

Similarly, this presents the benefit of such models being leveraged on potential edge systems with low tolerance for excess computational power such as satellites. Moreover, our work also provides insight into the task of binary classification for our dataset's domain as it represents easily classifiable data through the use of feature extraction. The use of ViT solidifies its versatility in this task, aside from its complexity, while also providing speculation on its capabilities for more complex downstream tasks. Such implementations hold potential for tasks such as the aforementioned two-stage model (8), which can combine into a singular more complex model. Our work has the ability to be used in collaboration with Planet Labs and their SkySats (8.2) to commercialize the classification tasks our models have currently achieved with satellite imagery provided by the company. As well as pave the way for its elevation into detection tasks (8.1). This scalability showcases the versatility of our architectures as well as practical real-world value.

9.0.1 GitHub Repository

Here is a reference to our GitHub repository containing our source code for our work: CV Final Project Repository

10 References

[1] United Nations Conference on Trade and Development, "Review of Maritime Transport 2023," UNCTAD, Tech. Rep., (2023). [Online]. Available: https://unctad.org/publication/review-maritime-transport-2023

[2] Bureau of Transportation Statistics, "Freight Indicators," U.S. Department of Transportation, Tech. Rep., (2025). [Online]. Available: https://www.bts.gov/freight-indicators

[3] Q. A. Al-Haija, M. A. Smadi, and S. Zein-Sabatto, "Multi-Class Weather Classification Using ResNet-18 CNN for Autonomous IoT and CPS Applications," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, (2020), pp. 1586–1591.

[4] M. Tarasiou, E. Chavez, and S. Zafeiriou, "ViTs for SITS: Vision Transformers for Satellite Image Time Series," (2023). [Online]. Available: https://arxiv.org/abs/2301.04944

[5] A. K. e. a. Alex Dosovitskiy, Lucas Beyer, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," 2021. [Online]. Available: https://arxiv.org/pdf/2010.11929

[6] J. Maurício, I. Domingues, and J. Bernardino, "Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review," *Applied Sciences*, vol. 13, no. 9, (2023). [Online]. Available: https://www.mdpi.com/2076-3417/13/9/5521

[7] R. Rhammell, "Ships in Satellite Imagery," Kaggle Dataset, (2025). [Online]. Available: https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery

[8] A. Y. Shdefat, N. Mostafa, Z. Al-Arnaout *et al.*, "Optimizing HAR Systems: Comparative Analysis of Enhanced SVM and k-NN Classifiers," *International Journal of Computational Intelligence Systems*, vol. 17, p. 150, (2024). [Online]. Available: https://doi.org/10.1007/s44196-024-00554-0

[9] R. Varghese and S. M., "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, Chennai, India, (2024), pp. 1–6.

[10] Starlink, "Technology," https://www.starlink.com/us/technology, (2025).

[11] P. G. del Hoyo and P. Samczynski, "Starlink-Based Passive Radar for Earth's Surface Imaging: First Experimental Results," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 17, pp. 13 949–13 965, (2024).

[12] P. Labs, "Planet Labs: Satellite Imagery & Earth Data Analytics," (2025). [Online]. Available: https://www.planet.com/

[13] K. Jacobsen, "Mapping with SkySat Images," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLIII-B1, p. 99, (2022), commission I, WG I/4, XXIV ISPRS Congress (2022 edition), 6–11 June 2022, Nice, France. [Online]. Available: https://isprs-archives.copernicus.org/articles/XLIII-B1-2022/99/2022/isprs-archives-XLIII-B1-2022-99-2022.pdf