IMPERIAL

UDA Final Project: Convolutional Neural Network to Classify Presence of Ships within Satellite Images

MATH70103: Unstructured Data Analysis

Name: **Adam Muhtar***

College ID: **02485505**

Code repository: **https://github.com/adammuhtar/ship-detection.git** †

1 Problem Statement

Maritime activities are integral to global trade, resource management, and national security. As commercial shipping routes expand and offshore operations become increasingly complex, the timely and accurate detection of ships—whether they are in the middle of the ocean or hugging near the coastlines—is critical for ensuring safety, efficiency, and compliance of maritime activities in coastal and international waters. For example, collisions or dangerous shoals could threaten lives of seafarers, disrupt supply chains, and can potentially result in environmental crises such as oil spills. As another example, unregulated and unmonitored fishing exacerbates the ongoing problem of overfishing, depleting fish stocks and undermining food security worldwide.

One particular domain that has recently surfaced as a major hazard to law-abiding mariners and the global economy alike is the issue of maritime security. In recent years, the maritime trade routes have become increasingly vulnerable to a range of security threats such as piracy, armed robbery, illegal ship seizures, and terrorism (International Maritime Organization 2024). This has been particularly pronounced in recent years, with an notable increase of attacks on ships operating along key maritime choke-points such as the Red Sea—with many of these attacks conducted via explosive-laden remote-controlled boats (Reuters 2024a). Beyond overt threats, maritime security is also dogged by more clandestine efforts, such as the various ongoing investigations into vessels suspected of sabotaging undersea fibre-optic cables (Reuters 2024b). These attacks have posed unacceptable risks in terms of the immediate threat to human lives operating in these areas, the potential damage to the local ecology, and the wider repercussions to the global economy.

*I have worked on this project independently.

†Access will be granted upon request. Refer to the `UDA_FinalProject_Muhtar.ipynb` Jupyter notebook in the notebooks directory for the code used for this project.

Ships have been a primary medium in which criminals traffic humans, arms, narcotics, and conduct other illicit activities such as sanctions evasion, exacerbating international crime and undermining regional stability. The vessels in which these illegal activities are often referred to as part of the "dark" fleet, which often operate while keeping their Automatic Identification Systems (AIS) and transponders switched off for the explicit purpose of evading detection (Atlantic Council 2024). This leaves us with allowing only a handful of reasonable means to properly monitor these ships.

Satellite imagery has emerged as a vital tool in addressing the multifaceted challenges posed by maritime activities, particularly in enhancing maritime situational awareness and security. High-resolution satellite images allow for the detection, tracking, and analysis of vessels across vast stretches of the ocean, providing critical data even in regions where traditional monitoring systems, such as transponders or AIS, fall short. Additionally, coverage of satellite imagery is global, enabling continuous surveillance of both open oceans, congested choke-points, and coastlines alike. However, the proliferation of satellite imagery have also led the need to parse large volumes of image data to sift through and effectively flag the presence of a ship within an image. This necessitates a machine-learning (ML) based approach to effectively scale the task of identification of presence of ship within a satellite image. This projects aims to tackle this challenge, by training an ML model to detect whether a given satellite image contains a ship or not.

2 Dataset: MASATI-v2

The dataset utilised in this project is sourced from the Maritime Satellite Imagery v2 (MASATI-v2) dataset, made available from Gallego et al. (2018).[1] The dataset contains 7,389 optical aerial satellite images in red-green-blue (RGB) channels from the visible spectrum of different sizes, with the average resolution of 512×512 pixels per image, and stored in portable network graphic (PNG) format.[2]

What makes this dataset particularly useful for this project is the range of images across different conditions and locations. Some of the key strengths of this dataset:

- Contains satellite images compiled between March and September of 2016 from different regions in Europe, Africa, Asia, the Mediterranean sea, the Atlantic, and Pacific oceans.
- Contains satellite images across different weather and illumination conditions.
- Contains satellite images captured at varying altitudes and distances.

Altogether, these features of the dataset makes it a relatively comprehensive representation of maritime environments that is suitable for our task. The dataset itself is generally divided into two broad classes, images with or without ships, and is further divided into seven sub-categories to cover the range of maritime scenes, as shown in Table 1. A random selection of this dataset is displayed in Figure 1, to provide some context on the types of scenes captured in this dataset.

[1] The dataset attached with this project is a subset of the full dataset (i.e. the first 37 images of each sub-folder) for testing purposes only, to ensure compliance with the 100MB upload limit of the assignment. The size of the full dataset is roughly 2.5GB and is available for free for non-profit research or educational purposes; the MASATI-v2 dataset can be downloaded at https://www.iuii.ua.es/datasets/masati/

[2] The MASATI-v2 dataset also provides bounding boxes co-ordinates containing the location of ships within the images (if present). This is outside the scope of the project and is not used for this project.

Main class	Sub-class	Sample size	Description
Ship	Ship	1,027	Images of ship(s) in the open sea
	Detail	1,789	Close-up views of ships.
	Multi	304	Images featuring multiple ships together in the frame.
	Coast & ship	1,037	Images of ship(s) near coastlines.
Non-ship	Sea	1,022	Open sea without any ships.
	Coast	1,132	Coastlines without any ships.
	Land	1,078	Land-based images.

Table 1: Sample distribution of ship and non-ship satellite images from Gallego et al. (2018).

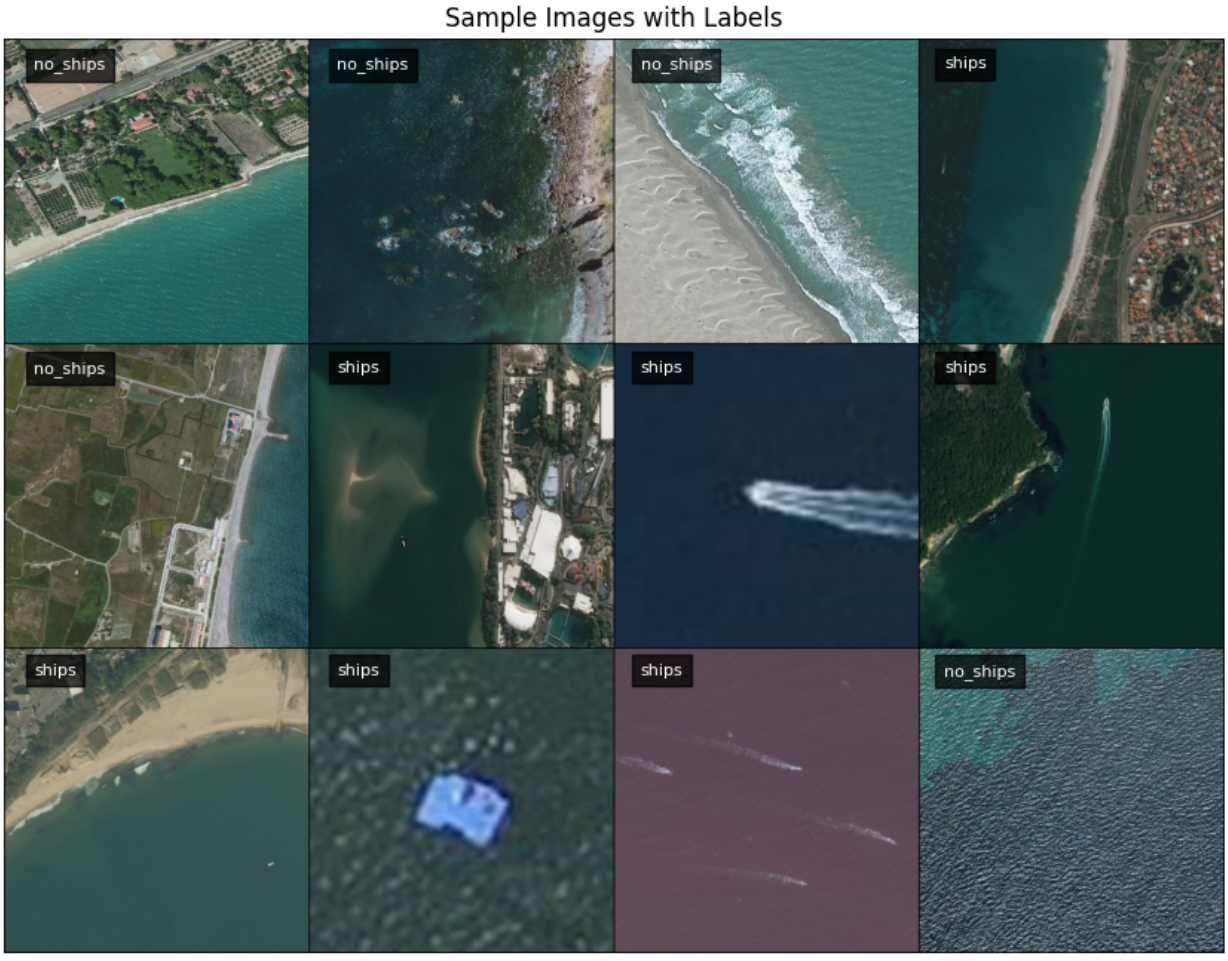

Figure 1: Selection of images provided by the MASATI-v2 dataeset (Gallego et al. 2018).

Transformations applied to MASATI-v2

We also apply various image transformations to improve robustness and generalisability of the model. The transformations applied are done via the `torchvision` library (PyTorch 2024c). Specifically, we apply the following transformations:

- `torchvision.transforms.Resize`: Standardises image input size to 512×512.
- `torchvision.transforms.RandomResizedCrop`: Partial occlusion or different scales.
- `torchvision.transforms.RandomHorizontalFlip`: Randomly flip the image horizontally.
- `torchvision.transforms.RandomVerticalFlip`: Randomly flip the image vertically.
- `torchvision.transforms.ColorJitter`: Randomly modifies the brightness and contrast of the images.
- `torchvision.transforms.ToTensor`: Converts image data to PyTorch-compatible tensors.
- `torchvision.transforms.Normalize`: Standardises pixel values around the mean and standard deviation values based on ImageNet dataset.

Prior to running the model training algorithm, we split the MASATI-v2 dataset into training and validation sets into 80%-20% split.

3 Methodology: Convolutional Neural Network for Image Classification

The model architecture chosen for this task is the convolutional neural network (CNN). CNN is widely used for various image-related tasks due to its ability to automatically and adaptively learn spatial hierarchies of features from data (LeCun et al. 2015). By utilising convolutional layers, CNNs apply learnable filters across the input image to detect local patterns such as edges, corners, or textures. These filters are convolved with the input data, enabling the model to recognise spatial and structural features regardless of their exact location within the image. This process captures spatial dependencies and hierarchical relationships, with earlier layers focusing on low-level features and deeper layers identifying complex structures. The feature map learned by the convolutional layers are then passed on to pooling layers, which reduces the spatial dimensions of the feature maps and thereby decreasing the computational complexity and preventing over-fitting. In other words, pooling operations summarise regions of the feature maps by extracting dominant information, making the representations more invariant to small shifts or distortions in the input data. Finally the fully connected layers take in high-level features extracted by the convolutional and pooling layers and map them to the desired output space.

By combining convolutional layers, pooling layers, and fully connected layers, CNNs effectively capture and represent both low-level features (e.g., edges and textures) and high-level features (e.g., shapes and objects). This makes them particularly suitable for tasks such as ours, which is an image classification task. This indeed the model architecture that is explored in Gallego et al. (2018), comparing performance of custom pre-trained models with various state-of-the-art CNN models such as VGG16 model from Simonyan and Zisserman (2015).

Model Architectures and Training Parameters

After several rounds of experimentation with model architectures, we arrive at two model candidates based on the CNN architecture for the task of image classification. The two models, listed here as `ShipClassifier2ConvNet` and `ShipClassifier4ConvNet`, are both $\sim$134 million parameter models that contain two and four convolutional layers respectively:[3]

At the high-level, both `ShipClassifier2ConvNet` and `ShipClassifier4ConvNet` consists of two main parts:

- **Convolutional feature extraction layers:** These layers extract spatial features from the input image. They progressively reduce the spatial dimensions while increasing the depth of the feature maps.
- **Fully connected feed-forward neural network classification layers:** These layers process the features extracted by the convolutional layers to classify the input into one of two categories: "ship" or "no ship".

Both models expect input images to be represented as tensors of shape (3, 512, 512), where 3 corresponds to the RGB channels of the image, and 512×512 is the spatial resolution of the image. Both network passes on this input tensor through either two or four convolutional layers, each comprising of the following:

- **Convolution:** Applies a convolution operation to its input, extracts spatial patterns (e.g., edges, textures, shapes), and outputs feature maps.
- **Batch normalisation:** Normalises the feature maps to accelerate training and improve stability by mitigating issues such as vanishing or exploding gradients.
- **Rectified linear unit (ReLU) activation:** Introduces non-linearity, allowing the network to learn complex functions.
- **Max pooling:** Applied at the end of the convolutional layer, which reduces spatial dimensions by half and lowering computational requirements, but still retain critical features while discarding less important details.

After the convolutional layers, the output feature maps are flattened into a one-dimensional vector and passed through two fully connected layers for classification, with ReLU activation function and dropout probability of 0.2 applied to reduce over-fitting by randomly zeroing out some neuron activations during training. The motivation behind this architecture is that these layers progressively increase the depth of feature extraction, i.e. the use of increasing feature maps in convolutional layers allows the network to learn increasingly complex patterns, followed by a feed-forward neural network to classify these features into the target space.

The specifics of the differences between the two models comes down to the total number of convolutional layers each model has, and this is best described via the full dimension breakdowns of each layers of the models:

- `ShipClassifier2ConvNet`**:**
  - **Input dimensions:** Input images as $3 \times 512 \times 512$ tensor as described above.

[3] Full PyTorch code is described in `UDA_FinalProject_Muhtar.ipynb`

- **First convolutional layer:** Using the formula to calculate the output spatial dimensions of a convolutional layer described in PyTorch (2024a):

$$\text{Output Height/Width} = \left\lfloor \frac{\text{Input Size} + 2 \times \text{Padding} - \text{Dilation} \times (\text{Kernel Size} - 1) - 1}{\text{Stride}} + 1 \right\rfloor$$

With dilation set to its default value of 1, we obtain the following output size following the convolution:

$$\text{Output Height/Width} = \frac{512 + 2 \times 1 - 1 \times (3 - 1) - 1}{1} + 1 = 512$$

The max pooling layer halves the output height/width from 512 to 256. With a pre-defined 32 output channels, we have an output shape of (32, 256, 256)

- **Second convolutional layer:** Similar to above, we have the following output size following the convolution:

$$\text{Output Height/Width} = \frac{256 + 2 \times 1 - 1 \times (3 - 1) - 1}{1} + 1 = 256$$

The max pooling layer quarters the output height/width from 256 to 64.[4] With a pre-defined 64 output channels, we have an output shape of (64, 64, 64)

- **Flatten and first fully connected layer:** We flatten the 3D tensor from the fourth convolutional layer to a 1D tensor, i.e. 64 $\times$ 64 $\times$ 64 = 262,144. This is fed to a feed forward fully connected layer with 512 outputs.
- **Second fully connected layer:** The 512 outputs from the previous fully connected layer is fed into a second feed forward fully connected layer with 2 outputs (ship or no ship). The final output shape is 2.

- `ShipClassifier4ConvNet`:

  - **Input dimensions:** Input images as 3 $\times$ 512 $\times$ 512 tensor as described above.
  - **First convolutional layer:** Using the formula to calculate the output spatial dimensions of a convolutional layer is given by:

$$\text{Output Height/Width} = \frac{\text{Input Size} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

We obtain the following output size following the convolution:

$$\text{Output Height/Width} = \frac{512 + 2 \times 1 - 1 \times (3 - 1) - 1}{1} + 1 = 512$$

The max pooling layer halves the output height/width from 512 to 256. With a pre-defined 32 output channels, we have an output shape of (32, 256, 256)

  - **Second convolutional layer:** Similar to above, we have the following output size following the convolution:

$$\text{Output Height/Width} = \frac{256 + 2 \times 1 - 1 \times (3 - 1) - 1}{1} + 1 = 256$$

The max pooling layer halves the output height/width from 256 to 128. With a pre-defined 64 output channels, we have an output shape of (64, 128, 128)

[4] The quartering of the output is to ensure the model with 2 convolutional layers parameter size is comparable to the one with 4 convolutional layers.

- **Third convolutional layer:** Similar to above, we have the following output size following the convolution:

$$\text{Output Height/Width} = \frac{128 + 2 \times 1 - 1 \times (3 - 1) - 1}{1} + 1 = 128$$

  The max pooling layer halves the output height/width from 128 to 64. With a pre-defined 128 output channels, we have an output shape of (128, 64, 64)

- **Fourth convolutional layer:** Similar to above, we have the following output size following the convolution:

$$\text{Output Height/Width} = \frac{64 + 2 \times 1 - 1 \times (3 - 1) - 1}{1} + 1 = 64$$

  The max pooling layer halves the output height/width from 64 to 32. With a pre-defined 256 output channels, we have an output shape of (256, 32, 32)

- **Flatten and first fully connected layer:** We flatten the 3D tensor from the fourth convolutional layer to a 1D tensor, i.e. $256 \times 32 \times 32 = 262{,}144$. This is fed to a feed forward fully connected layer with 512 outputs.

- **Second fully connected layer:** The 512 outputs from the previous fully connected layer is fed into a second feed forward fully connected layer with 2 outputs (ship or no ship). The final output shape is 2.

During the training process itself, we utilise the following loss function, optimiser, and learning rate scheduler respectively:

- **Loss function—Cross entropy loss:** Takes raw logits of the model as inputs, applies softmax operation internally to convert logits into probabilities, computes the negative log likelihood of the true class's probability, and averages the loss over all samples in the batch. Label smoothing is applied, which means 10% of the confidence is redistributed among the incorrect classes. For our binary classification problem, this means the target probabilities become 0.9 for the true class and 0.1 for the other class—this is done with the intention of preventing over-fitting and encourages better generalisation by slightly penalising overconfidence.

- **Optimiser—Adaptive Moment Estimation (Adam):** Adam optimiser is applied with a learning rate 0.0001, which determines the step size for updating weights during backpropagation. Weight decay of 0.00001 is applied as a regularisation term to the loss function, penalising large weights to prevent over-fitting.

- **Learning rate scheduler:** Gradually decreases the learning rate during training, which helps the model converge more effectively. In our case, for every 10 epochs, the learning rate is multiplied by a factor of 0.1. This allows a variable learning rate where at the start, higher learning rate allows the model to make larger updates and learn quickly, while at the later stages, reduced learning rate allows the model to fine-tune and converge on the optimal solution without overshooting.

Hardware specifications

We run the training process using an Apple M1 Pro chip (10-core CPU, with 8 performance cores and 2 efficiency cores) with 16GB unified memory. We run the training operations using the integrated

GPUs that are part of Apple's M1 Pro System-on-Chip (SoC) architecture, by setting `torch.device("mps")` and sending the model to that device (PyTorch 2024b). This allows our training script to leverage the GPU for training or inference, allowing us to utilise the hardware-accelerated compute capabilities of Apple devices efficiently.

We train our model over 30 epochs, with each epoch taking roughly 4-5 minutes to train using this hardware. For a single complete training run, we would expect the complete training run to take between 2-3 hours, depending on the compute load and resources available at hand. Training both models, we would expect the full training run to take around 5-6 hours to complete.

4 Results and Discussion

After running both models over 30 epochs, we obtain the results of the training loss, validation loss, and validation accuracy shown in Table 2. While both models exhibit signatures of improving performance over the validation set—indicating some degree of learning by the models—we can observe that `ShipClassifier4ConvNet` model fares slightly better than `ShipClassifier2ConvNet` model during the course of their training runs. For `ShipClassifier4ConvNet`, we note that epoch 25 is the best performing, with a validation loss of 0.4339 and a validation set accuracy of 0.8356, whereas for `ShipClassifier2ConvNet`, the best model would either be the epoch 26 model, with validation loss of 0.4628, or epoch 20 model, with validation accuracy of 0.8221, depending on the metric of interest. We visualise the results of the training runs for both models in Figures 2 and 3.

`ShipClassifier4ConvNet`'s relatively better performance was expected, highlighting the capabilities of more convolutional layers progressively learning increasingly complex patterns to discern the presence of ships within satellite imagery, compared to `ShipClassifier2ConvNet` which has 2 convolutional layers less. Additionally, from Figure 2, we note that the model performance for `ShipClassifier2ConvNet` has much more varied accuracy scores at the start and seems to have plateaued near the end of the 30 epochs. On the other hand, from Figure 3, we observe that the model with 4 convolutional layers has a relatively more consistent trend of improvement at the start and plateaus at a higher accuracy level compared to `ShipClassifier2ConvNet`.

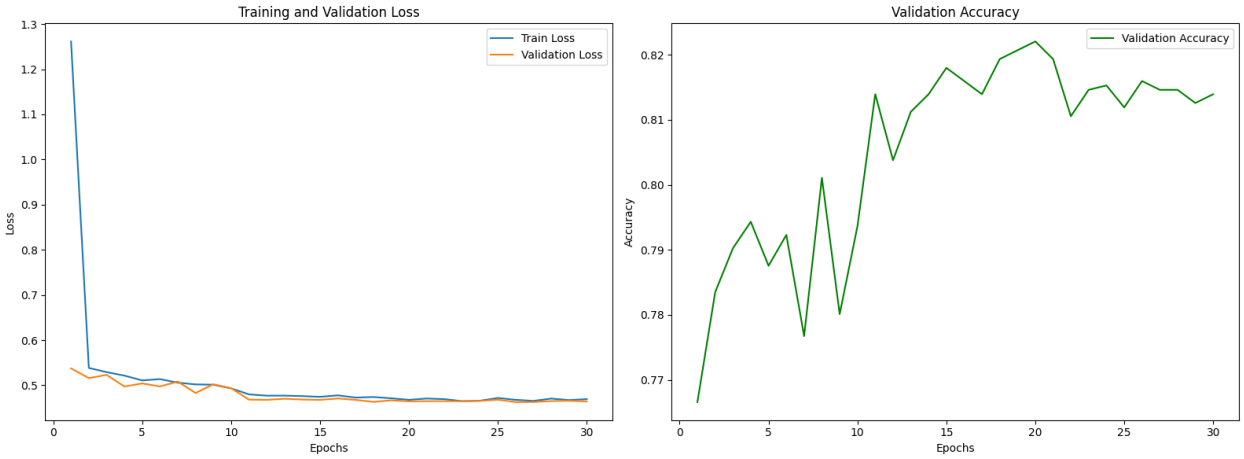

Figure 2: Training and validation loss (LHS) and validation accuracy (RHS) over training epochs for `ShipClassifier2ConvNet`.

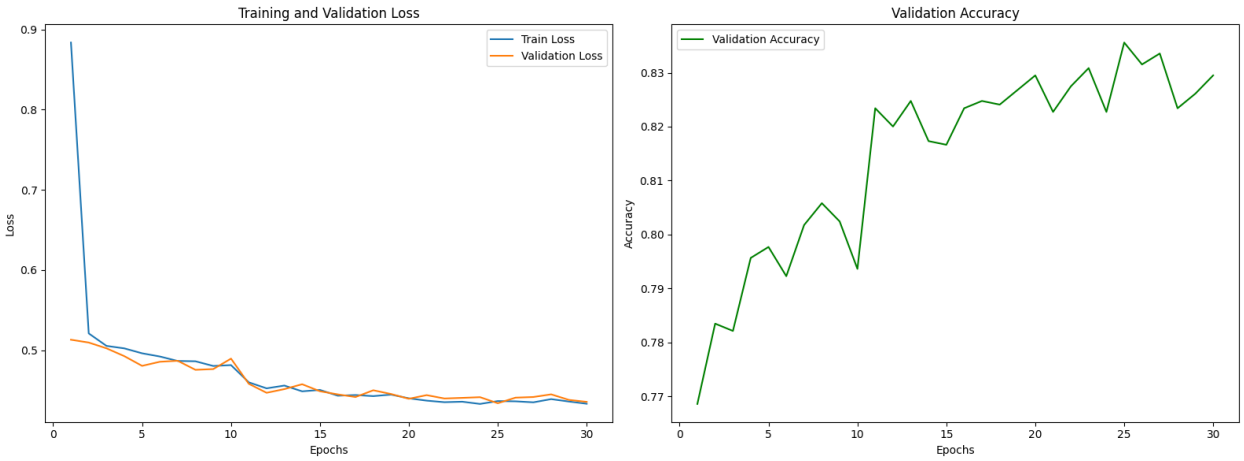

Figure 3: Training and validation loss (LHS) and validation accuracy (RHS) over training epochs for `ShipClassifier4ConvNet`.

We can also observe the "sharpness" of each model's ship detection capabilities by testing the model on a previously unseen image and visualising the feature maps "learned" of their respective final convolutional layers. We test the model's prediction capabilities on the image in Figure 4 from Hammell (2018), and obtain the following confidence scores:

- `ShipClassifier2ConvNet`: Prediction: 'Ships'. Confidence score: 70.98%
- `ShipClassifier4ConvNet`: Prediction: 'Ships'. Confidence score: 80.29%

Visualising the feature maps "learned" by each model's final convolutional layers in Figures 5 and 6, we can observe that the `ShipClassifier4ConvNet` model arrives at a much sharper detection of ship presence, where almost all filters at this stage are dedicated to detecting ships and complex features in the given image, seen from the brighter spots in the region where ships are present. The `ShipClassifier2ConvNet` model on the other hand, contains filters that display varying degrees of detection of ships, with a number of them distinguishing more basic features, e.g. land vs sea.

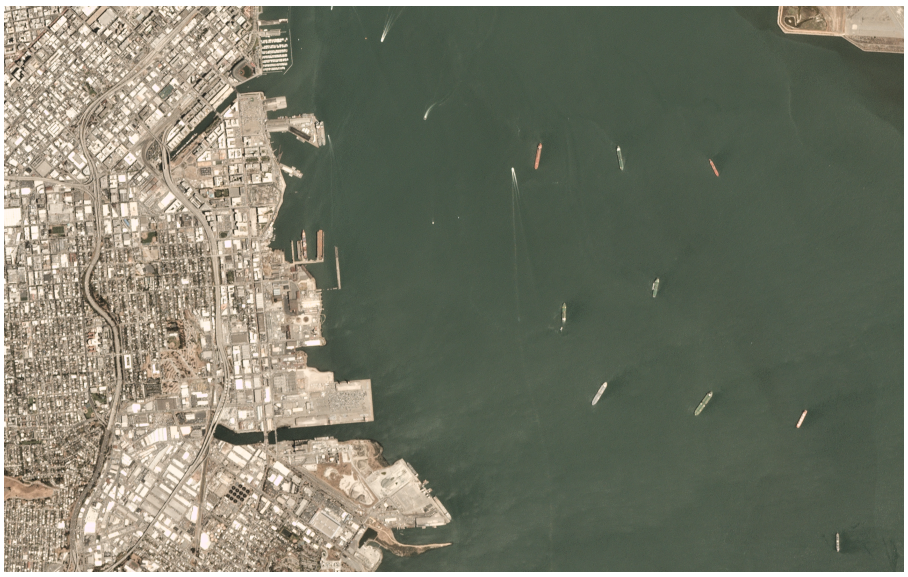

Figure 4: Test satellite image of the San Francisco Bay from Hammell (2018).

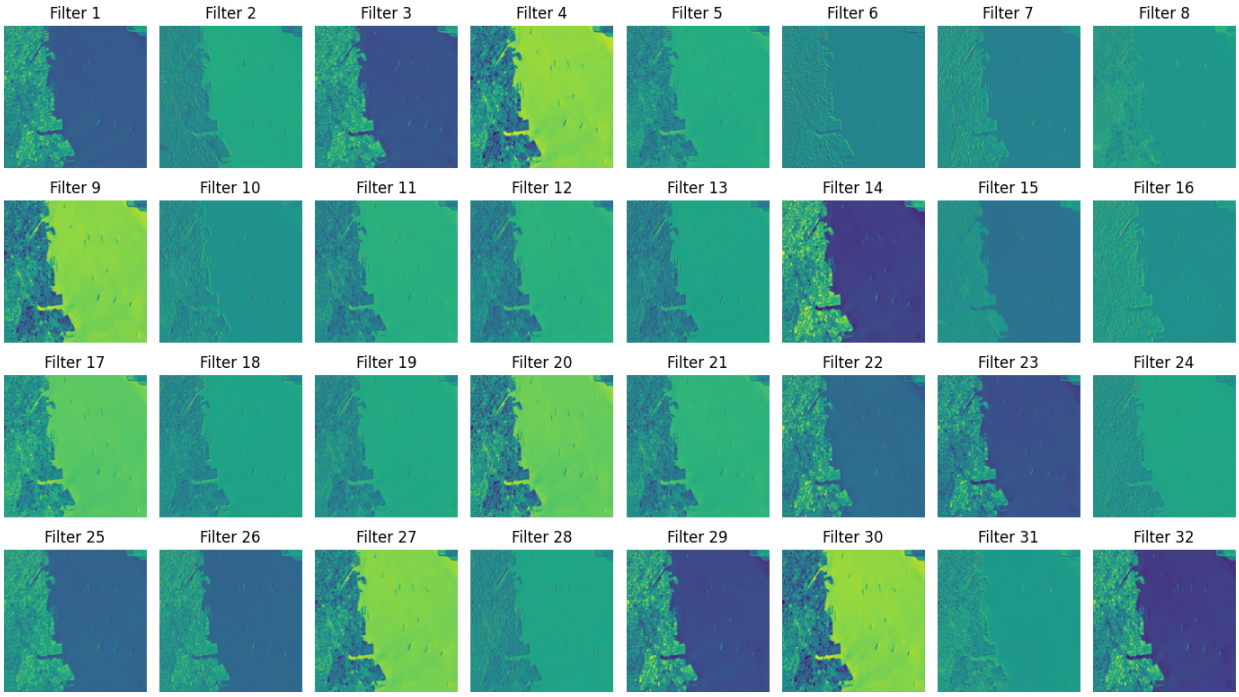

Figure 5: Feature maps of the final (2nd) convolutional layer of `ShipClassifier2ConvNet`.

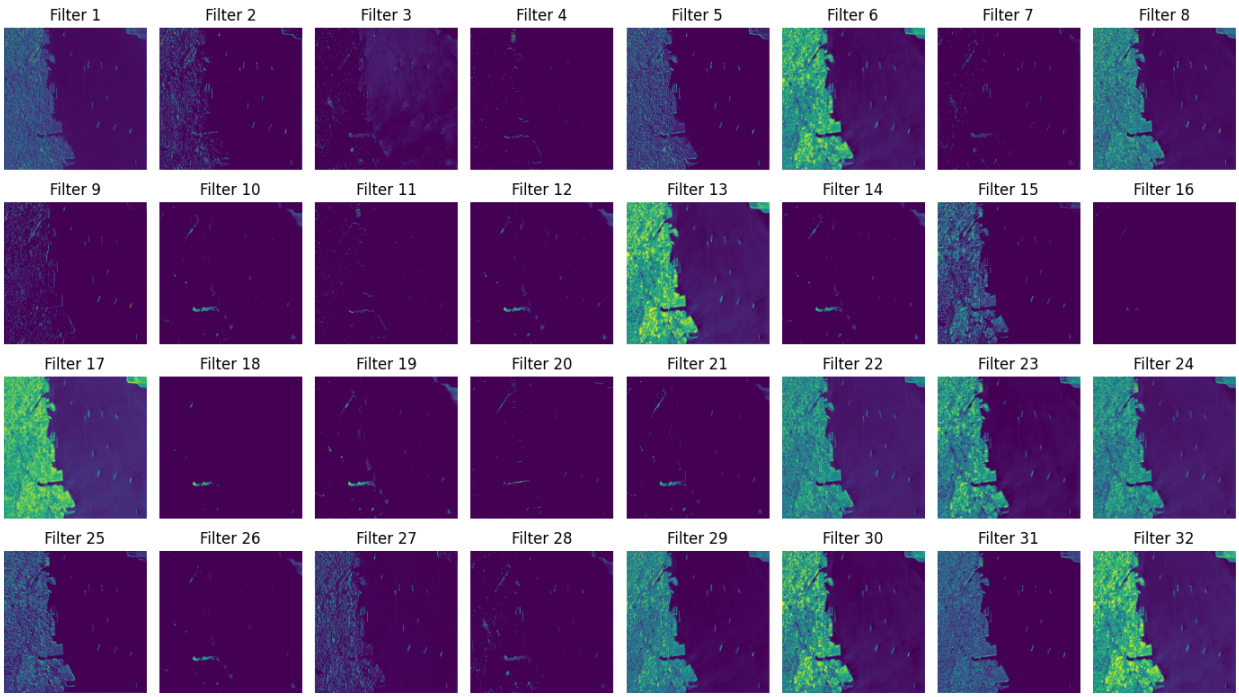

Figure 6: Feature maps of the final (4th) convolutional layer of `ShipClassifier4ConvNet`.

Epoch	Training Loss (2ConvNet)	Validation Loss (2ConvNet)	Validation Accuracy (2ConvNet)	Training Loss (4ConvNet)	Validation Loss (4ConvNet)	Validation Accuracy (4ConvNet)
1	1.2614	0.5374	0.7666	0.8835	0.5131	0.7686
2	0.5384	0.5158	0.7835	0.5210	0.5096	0.7835
3	0.5291	0.5231	0.7903	0.5054	0.5024	0.7821
4	0.5214	0.4973	0.7943	0.5023	0.4926	0.7957
5	0.5108	0.5041	0.7876	0.4962	0.4805	0.7977
6	0.5137	0.4973	0.7923	0.4922	0.4856	0.7923
7	0.5058	0.5082	0.7767	0.4866	0.4869	0.8018
8	0.5019	0.4829	0.8011	0.4862	0.4756	0.8058
9	0.5013	0.5021	0.7801	0.4804	0.4765	0.8024
10	0.4931	0.4936	0.7936	0.4815	0.4895	0.7936
11	0.4799	0.4682	0.8139	0.4600	0.4581	0.8234
12	0.4770	0.4677	0.8038	0.4525	0.4469	0.8200
13	0.4771	0.4702	0.8112	0.4560	0.4515	0.8248
14	0.4761	0.4684	0.8139	0.4488	0.4576	0.8173
15	0.4745	0.4677	0.8180	0.4505	0.4489	0.8166
16	0.4776	0.4708	0.8160	0.4433	0.4450	0.8234
17	0.4728	0.4677	0.8139	0.4441	0.4417	0.8248
18	0.4740	0.4632	0.8194	0.4428	0.4501	0.8241
19	0.4712	0.4667	0.8207	0.4447	0.4454	0.8268
20	0.4678	0.4644	**0.8221**	0.4401	0.4395	0.8295
21	0.4707	0.4647	0.8194	0.4372	0.4440	0.8227
22	0.4693	0.4649	0.8106	0.4352	0.4398	0.8275
23	**0.4648**	0.4646	0.8146	0.4358	0.4406	0.8309
24	0.4658	0.4655	0.8153	**0.4330**	0.4415	0.8227
25	0.4719	0.4681	0.8119	0.4366	**0.4339**	**0.8356**
26	0.4678	**0.4628**	0.8160	0.4363	0.4409	0.8315
27	0.4654	0.4630	0.8146	0.4351	0.4417	0.8336
28	0.4706	0.4650	0.8146	0.4391	0.4450	0.8234
29	0.4672	0.4657	0.8126	0.4360	0.4381	0.8261
30	0.4693	0.4643	0.8139	0.4333	0.4357	0.8295

Table 2: `ShipClassifier2ConvNet` and `ShipClassifier4ConvNet` training metrics over 30 epochs; bold numbers indicate minimum values for training/validation loss or maximum value for validation accuracy.

Future research considerations

It is worth noting that while model performance for 4 convolutional layers model is better than the 2 convolutional layer model, the magnitude of the differnce in model performance is relatively small. We could reasonably hypothesise that given enough training epochs, both models could potentially converge to the same performance levels; indeed the baseline models trained in Gallego et al. (2018) was trained for 300 epochs. For future research considerations, we should consider the following:

- Extending the number of epochs of the training run to larger number to observe if there is indeed convergence.
- Consider alternative architectures, like that of VGG16 from Simonyan and Zisserman (2015). We considered implementing this model architecture for this project; however, the model size and training time was too large for the computational resources used for this project.
- Hyperparameter optimisation to obtain the best sets of training parameters to maximise model performance. This could be done using frameworks such as Optuna to find optimal mix of training parameters to obtain the best performing model. We also considered implementing hyperparameter tuning as part of the training process; however, the computational resources to run this was insufficient.
- Explore incorporating transfer learning by utilising pre-trained models such as ResNet (He et al. 2016)—which have demonstrated superior performance in image classification tasks—and fine-tune them on the MASATI-v2 dataset to potentially improve model performance while reducing training time.
- Expand the training and validation datasets to include new sources of satellite imagery, with a larger geographical footprint and covers a wider range of illumination, weather conditions, resolution, angles, elevation, etc.

References

Atlantic Council (Jan. 2024). "Russia's growing dark fleet: Risks for the global maritime order". In: Atlantic Council Issue Brief. Accessed: 2024-12-30. URL: https://www.atlanticcouncil.org/in-depth-research-reports/issue-brief/russias-growing-dark-fleet-risks-for-the-global-maritime-order/.

Gallego, Antonio-Javier, Antonio Pertusa, and Pablo Gil (2018). "Automatic Ship Classification from Optical Aerial Images with Convolutional Neural Networks". In: Remote Sensing 10.4. ISSN: 2072-4292. DOI: 10.3390/rs10040511.

Hammell, Ryan (2018). Ships in Satellite Imagery. https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery. Accessed: 2024-12-30.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

International Maritime Organization (May 2024). Maritime Safety Committee 108th session. https://www.imo.org/en/MediaCentre/MeetingSummaries/Pages/MSC-108th-session.aspx. Accessed: 2024-12-30.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: Nature 521.7553, pp. 436–444. DOI: 10.1038/nature14539.

PyTorch (2024a). Conv2d. Accessed: 2024-12-30. URL: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.

— (2024b). MPS backend. Accessed: 2024-12-30. URL: https://pytorch.org/docs/stable/notes/mps.html.

— (2024c). TorchVision: PyTorch Computer Vision Library. Accessed: 2024-12-30. URL: https://pytorch.org/vision/stable/index.html.

Reuters (July 2024a). "Houthi explosive drone boat attacks escalate Red Sea danger". In: Reuters. Accessed: 2024-12-30. URL: https://www.reuters.com/world/middle-east/houthi-explosive-drone-boat-attacks-escalate-red-sea-danger-2024-07-03/.

— (Nov. 2024b). "Sweden urges Chinese ship to return for undersea cable investigation". In: Reuters. Accessed: 2024-12-30. URL: https://www.reuters.com/world/sweden-asks-chinese-ship-yi-peng-3-move-swedish-waters-pm-says-2024-11-26/.

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: arXiv preprint arXiv:1409.1556. URL: https://arxiv.org/abs/1409.1556.