Ship Classification using CNN Model

Table of Contents

Introduction

OceanVision AI is a leading maritime analytics company, specializing in automated ship detection and classification using satellite and drone imagery. Their clients—maritime authorities, logistics firms, and defense agencies—rely on precise ship identification to monitor sea traffic, detect unauthorized activities, and optimize naval operations.

Currently, ship classification is performed manually by experts, making the process time-consuming, labor-intensive, and prone to errors. Misclassifications can lead to security risks, logistical inefficiencies, and financial losses. To overcome these challenges, OceanVision AI aims to develop an AI-driven ship classification model that can automatically categorize ships into five distinct types: Cargo Ships, Military Ships, Carriers, Cruise Ships, and Tankers.

This report presents a Deep Learning-based Image Classification Model designed to accurately identify and classify ships using convolutional neural networks (CNNs) and transfer learning techniques. The model is trained on labeled image datasets and optimized for high accuracy, robustness, and real-world deployment. By leveraging AI, OceanVision AI seeks to streamline maritime monitoring, enhance security measures, and improve operational efficiency across the global shipping industry.

Methodology

- Data Preprocessing:
    - Load and inspect 6252 labeled ship images across five categories.
    - Apply data augmentation (rotation, flipping, zooming) for generalization.
    - Normalize pixel values and resize images to 128×128(I used 128 as my computer could only support this much), images can also use 224*224.

    - Split data into 80% training (6,252 images) and 20% testing (2,680 images).
- Model Development:
    - Implement a custom CNN model with convolutional, pooling, and fully connected layers.
    - Use transfer learning (MobileNetV2, ResNet50) for improved feature extraction.
    - Apply softmax activation for multi-class classification.
- Training & Evaluation:
    - Compile using Adam optimizer and categorical_crossentropy loss.
    - Evaluate using accuracy, precision, recall, F1-score, and confusion matrix.
    - Monitor training to detect overfitting using accuracy/loss plots.
- Optimization:
    - Apply dropout, batch normalization, and early stopping to improve performance.
    - Tune hyperparameters like learning rate and batch size.
- Deployment:
    - The model will be deployed using streamlit.

Data Preprocessing

To prepare the dataset for model training, several preprocessing steps were applied to ensure data quality, consistency, and improved generalization.

Data Loading & Inspection

```
[ ]  import pandas as pd

     df = pd.read_csv(CSV_PATH)
     df.head()
```

	image	category
0	2823080.jpg	1
1	2870024.jpg	1
2	2662125.jpg	2
3	2900420.jpg	3
4	2804883.jpg	2

```
[ ]  df.count()#data count

     image       6252
     category    6252
     dtype: int64

[ ]  df.image.value_counts().unique() #Check for duplicates

     array([1])

[ ]  # percentage of class distribution in the dataset
     (df.value_counts('category')/6252)*100

     category
     1    33.909149
     5    19.465771
     2    18.666027
     3    14.651312
     4    13.307742
     Name: count, dtype: float64
```

- The dataset consists of 6,252 labeled ship images categorized into five classes.
- Image file names and their respective category labels were retrieved from train.csv.
- A class distribution analysis revealed slight class imbalance, with Category 1 being the most frequent (33.91%) and Category 4 the least (13.31%).

Image Preprocessing

```python
# Load and preprocess images
def load_images():
    for index, row in df.iterrows():
        img_name = row['image']
        label = row['category']-1 # Adjust label to start from 0

        img_path = os.path.join(IMAGE_PATH, img_name)

        if not os.path.exists(img_path):
            print(f"Warning: {img_path} not found, skipping...")
            continue

        # Read image using OpenCV
        img = cv2.imread(img_path)
        if img is None:
            print(f"Warning: Could not read {img_path}, skipping...")
            continue

        # Convert BGR to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Resize image
        img = cv2.resize(img, IMAGE_SIZE)

        # Check if the image is blank
        if np.all(img == img[0, 0, :]):  # Check if all pixels are the same
            print(f"Warning: {img_path} is blank, skipping...")
            continue

        # Normalize pixels to [0,1] range
        img = img / 255.0

        images.append(img)
        labels.append(label)
```

- Image Loading: Images were loaded using OpenCV, ensuring each image was correctly mapped to its label.
- Resizing: All images were resized to 128×128 pixels to standardize input size, reduce computational load, and retain essential features.
- Format Conversion: OpenCV loads images in BGR format, which was converted to RGB for model compatibility.
- Quality Filtering: Missing images and unreadable files were skipped.
- Completely blank images (all pixels identical) were removed to maintain dataset integrity.
- Normalization: Pixel values were scaled to the [0,1] range for improved model convergence.

One-Hot Encoding & Data Splitting

```
# One-hot encode labels. # use mapping
y = to_categorical(y, num_classes=NUM_CLASSES)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

- Labels were one-hot encoded to convert categorical values into a binary matrix format.
- The dataset was split into 80% training (5,001 images) and 20% testing (1,251 images) using stratified sampling to maintain class balance.

Data Augmentation

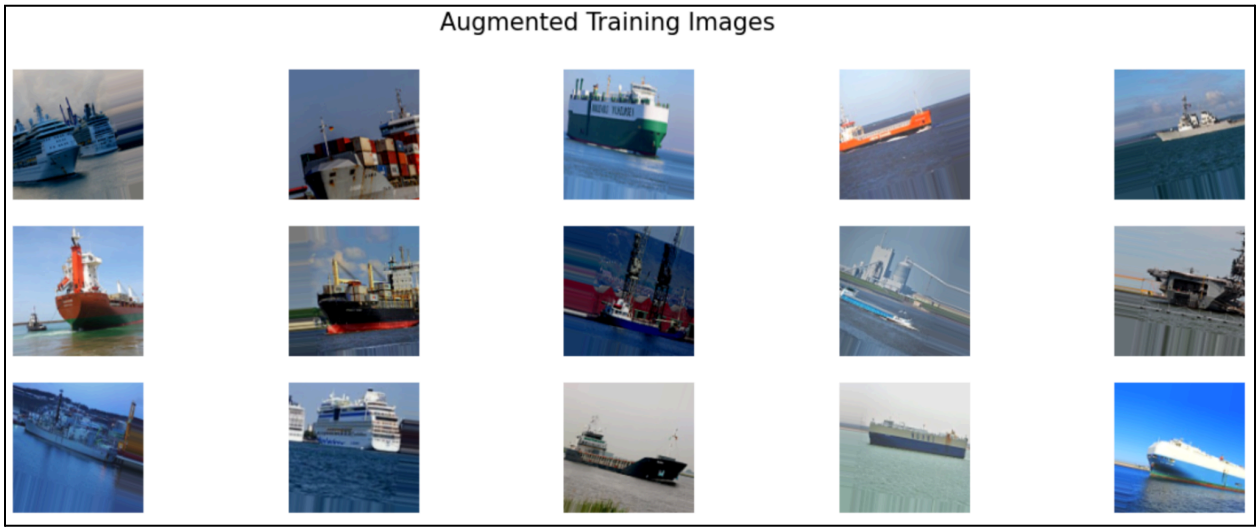
```
#image augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Define data augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20, # Randomly rotate images
    width_shift_range=0.2, # Randomly shift images horizontally
    height_shift_range=0.2, # Randomly shift images vertically
    shear_range=0.2,# Randomly shear images
    zoom_range=0.2,# Randomly zoom in/out
    channel_shift_range=0.2, # Randomly change brightness
    horizontal_flip=True,# Randomly flip images
    fill_mode='nearest'# Fill in new pixels
)
# Display augmented images
plt.figure(figsize=(15, 5))
for i in range(15):  # Show up to 15 images
    plt.subplot(3, 5, i + 1)
    augmented_image = datagen.random_transform(X_train[i])
    plt.imshow(augmented_image)
    plt.axis('off')
plt.suptitle("Augmented Training Images", fontsize=15)
plt.show()
```

To improve model generalization and prevent overfitting, data augmentation was applied using the ImageDataGenerator. The following transformations were used:

- Rotation – Simulates different viewing angles.
- Shifts) – Introduces minor displacements.
- Shearing & Zooming– Enhances perspective variation.
- Channel Shifting – Adjusts brightness for better contrast adaptability.
- Horizontal Flipping – Helps recognize mirrored patterns.
- Fill Mode: Fills missing pixels after transformations to maintain image integrity.

A visualization of augmented images confirmed the effectiveness of these transformations in creating a diverse dataset.


Sample Training Images


Augmented Training Images

Model Architecture

Model 1

```
model = keras.Sequential([
    keras.layers.Input(shape=(128, 128, 3)),

    # Block 1
    keras.layers.Conv2D(32, (3,3), activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D((2,2)),
    keras.layers.Dropout(0.2),

    # Block 2
    keras.layers.Conv2D(64, (3,3), activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D((2,2)),
    keras.layers.Dropout(0.3),

    # Block 3
    keras.layers.Conv2D(128, (3,3), activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D((2,2)),

    keras.layers.Flatten(),

    # Dense
    keras.layers.Dense(128, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.5),

    keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

The model has 3 convolutional blocks, each with increasing filters (32→64→128) to capture hierarchical features, for feature extraction and fully connected layers with 128 neurons for classification and the final output layer with 6 neurons.

- `keras.layers.Input(shape=(128, 128, 3)),`

  The input layer takes images of size 128*128 and three colour channels. This ensures that the model can processes colored images of size 128*128. Used 128 instead of 224 to

reduce computational complexity and saving my laptop from crashing. The 224*224 can increase accuracy but requires hardware that can support it.

- ```
  # Block 1
  keras.layers.Conv2D(32, (3,3), activation='relu'),
  keras.layers.BatchNormalization(),
  keras.layers.MaxPooling2D((2,2)),
  keras.layers.Dropout(0.2),
  ```

Conv2D(32, (3,3)): This layer applies 32 filters of size 3×3 to the input image. The filters are increased from 32 to 64 to 128, across the three blocks. Therefore Block 2 uses 64 filters allowing the model to learn more detailed features, Block 3 uses 128 filters which detects high level patterns.

ReLU Activation: The ReLU (Rectified Linear Unit) function introduces non-linearity, allowing the model to learn complex patterns. (max(0,x))

BatchNormalization(): This helps stabilize training by normalizing activations across mini-batches, reducing internal covariate shifts.

MaxPooling2D(2,2): This layer reduces the spatial dimensions by taking the maximum value in a 2×2 region. It retains the most important features while reducing computational cost.

Dropout(0.2): 20% of neurons are randomly dropped during training to prevent overfitting. Block 2 has a dropout rate of 0.3 and the final output layer has dropout rate of 0.5.

- ```
  keras.layers.Flatten(),

  # Dense
  keras.layers.Dense(128, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
  keras.layers.BatchNormalization(),
  keras.layers.Dropout(0.5),

  keras.layers.Dense(NUM_CLASSES, activation='softmax')
  ```

After extracting features, the network flattens the feature maps and feeds them into fully connected layers to classify the ships. *Keras.layers.Flatten()* is used here to convert the multi-dimensional feature maps into a 1D vector, making it compatible for classification.

Dense Layer with 128 neurons with ReLU activation allows the model to learn abstract relationships between ship features. L2 Regularization (0.001) helps prevent overfitting by penalizing large weights. Batch Normalization further stabilizes training. 50% Dropout prevents reliance on certain neurons and improves generalization.

The **final output layer** uses Softmax Activation to output probability scores for each of the five ship categories. The final output consists of five neurons, one for each category.

```
# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy',Precision(), Recall()])
```

- The model is then compiled using:
    - Adam Optimizer: Adaptive learning for faster convergence.
    - Categorical Crossentropy: Used for multi-class classification.
    - Metrics:
        - Accuracy: Measures correct predictions.
        - Precision: Focuses on how many predicted positives are correct.
        - Recall: Measures how many actual positives were detected.

```
# Train model with Early Stopping
history = model.fit( X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test),
                    callbacks=[early_stopping, reduce_lr],class_weight=class_weight_dic)
```

- Training with early stopping, class balancing, LR adjustment
    - Stops training if validation loss doesn't improve for 10 epochs. Prevents overfitting by avoiding unnecessary training.

- Reduces learning rate if the model stops improving. Prevents oscillation in training.
- Assigns higher weight to underrepresented classes to prevent bias. Ensures fair learning across all categories.

- Model Training:
  - Epochs: 50 (adjusted dynamically with early stopping).
  - Batch Size: 32
  - Validation Data: Helps monitor generalization.
  - Callbacks: Early stopping & learning rate adjustment.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
batch_normalization (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
dropout (Dropout)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_1 (Dropout)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
batch_normalization_3 (BatchNormalization)	(None, 128)	512
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 5)	645

Total params: 9,918,673 (37.84 MB)
Trainable params: 3,305,989 (12.61 MB)
Non-trainable params: 704 (2.75 KB)
Optimizer params: 6,611,980 (25.22 MB)

Model 2

```
# Load Pretrained Model (MobileNetV2)
base_model = keras.applications.MobileNetV2(
    input_shape=(128, 128, 3), include_top=False, weights="imagenet"
)
base_model.trainable = False  # Freeze base model layers

# Define the Model
model1 = keras.Sequential([
    base_model,
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(NUM_CLASSES, activation='softmax')
])

for layer in base_model.layers[-30:]:
    layer.trainable = True

# Compile Model
model1.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy', Precision(), Recall()])



# Train Model
history = model1.fit(
    X_train, y_train, batch_size=64, epochs=50, validation_data=(X_test, y_test), callbacks=[early_stopping, reduce_lr], class_weight=class_weight_dic
)
```

Transfer learning using MobileNetV2, a lightweight convolutional neural network (CNN) that is pretrained on the ImageNet dataset.

- Loading the Pretrained Model (MobileNetV2)
    - MobileNetV2 is loaded with pretrained weights from ImageNet.
    - The include_top=False option removes the original classification head, allowing us to add a custom classifier.
    - The model is frozen initially (base_model.trainable = False) to retain the learned feature extraction layers.
- Building the Custom Classifier
    - Global Average Pooling (GAP): Replaces fully connected layers, reducing parameters and overfitting.
    - Batch Normalization: Stabilizes activations, improving training efficiency.

    - Dense Layer (256 neurons, ReLU): Learns high-level patterns for ship classification.
    - Dropout (50%): Prevents overfitting by randomly deactivating neurons during training.
    - Output Layer: Uses softmax activation for multi-class classification, predicting probabilities for each ship category.
- Fine-tuning the Last 30 Layers
    - Initially, MobileNetV2 was frozen to use its pretrained knowledge.
    - Now, the last 30 layers are unfrozen, allowing fine-tuning for ship-specific features.
- Compiling the Model
    - Optimizer: adam (Adaptive Moment Estimation) dynamically adjusts learning rates.
    - Loss Function: categorical_crossentropy, ideal for multi-class classification.
    - Metrics: accuracy, precision, and recall to measure classification performance.
- Training the Model
    - Batch size: 64, meaning 64 images are processed per iteration.
    - Epochs: 50, but training stops early if no improvement is detected.
    - Validation Data: Helps monitor performance on unseen data.
    - Callbacks:
        - early_stopping: Stops training if validation loss stops improving.
        - reduce_lr: Lowers the learning rate if progress slows.

- Class Weights: Adjusts for class imbalance to ensure underrepresented classes are learned properly.

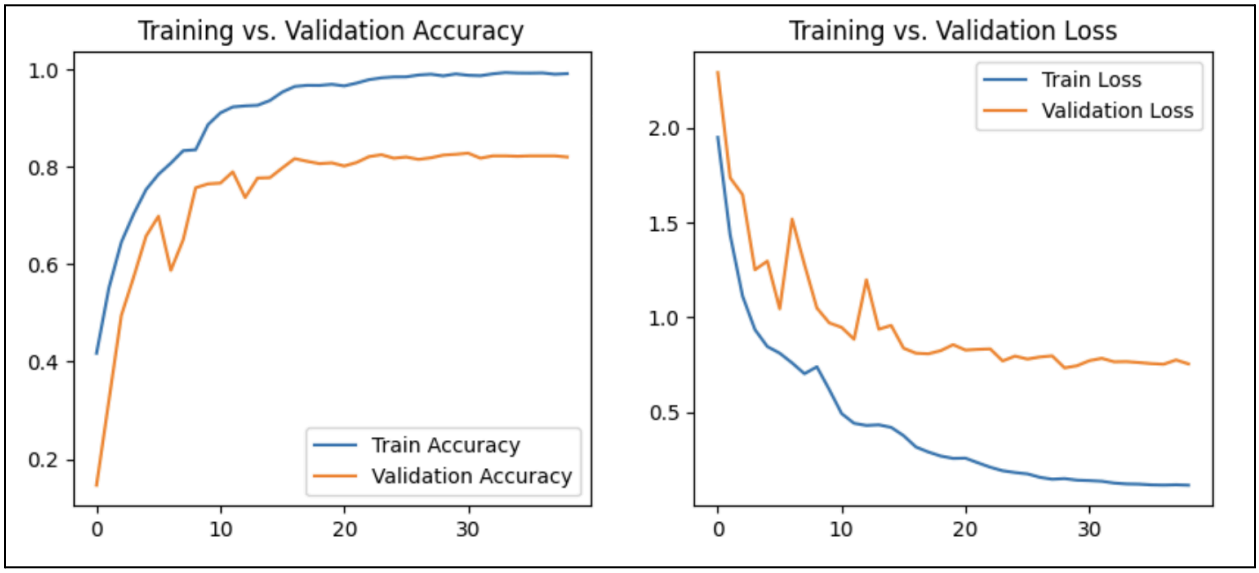
Model: "sequential_1"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_128 (Functional)	(None, 4, 4, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
batch_normalization_4 (BatchNormalization)	(None, 1280)	5,120
dense_2 (Dense)	(None, 256)	327,936
dropout_3 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 5)	1,285

Total params: 6,308,689 (24.07 MB)
Trainable params: 1,858,181 (7.09 MB)
Non-trainable params: 734,144 (2.80 MB)
Optimizer params: 3,716,364 (14.18 MB)

Evaluation

Model 1

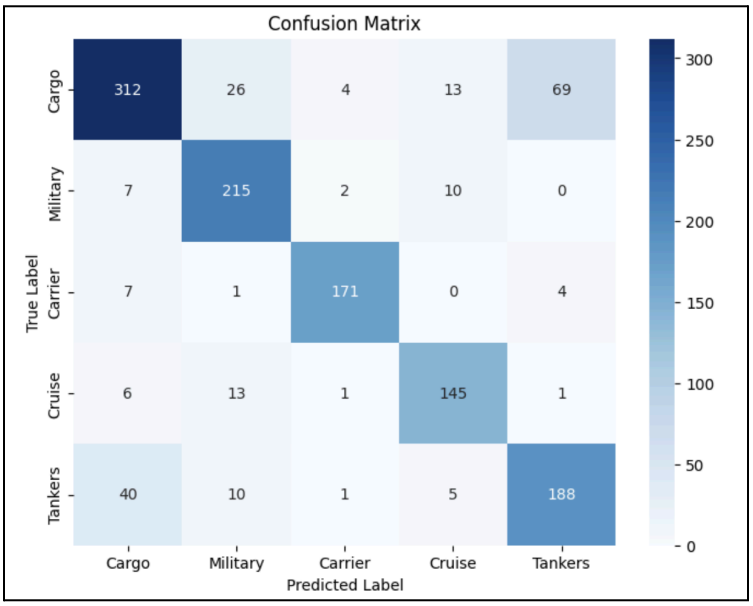


- Training vs Validation Accuracy

  The training accuracy increases rapidly in the initial epochs. It plateaus around 1.0 (or very close) after about 20 epochs. This indicates that the model is learning the training data very well. The validation accuracy also increases initially, but at a slower rate than the training accuracy. It plateaus around 0.8 (or slightly higher) after about 10 epochs. There's a significant gap between the training and validation accuracy, particularly after the training accuracy reaches near 1.0. This is a strong indicator of overfitting.

- Training vs Validation Loss

  The training loss decreases rapidly in the initial epochs. It continues to decrease and approaches near 0 after about 20 epochs. This confirms that the model is fitting the training data very closely. The validation loss decreases initially, but then plateaus and even starts to increase slightly after about 10 epochs. The validation loss starts to diverge from the training loss, which is another clear sign of overfitting.



Class-Specific Performance:

- **Cargo:** 312 correctly classified. Significant confusion with Tankers (69 misclassified as Tankers). Some confusion with Military (26), Carrier (4), and Cruise (13).
- **Military:** 215 correctly classified. Some confusion with Cargo (7), Carrier (2), and Cruise (10).
- **Carrier:** 171 correctly classified. Some confusion with Cargo (7), Military (1), and Tankers (4).
- **Cruise:** 145 correctly classified. Some confusion with Military (13), Cargo (6), Carrier (1), and Tankers (1).
- **Tankers:** 188 correctly classified. Significant confusion with Cargo (40 misclassified as Cargo). Some confusion with Military (10), Carrier (1), and Cruise (5).

The most significant source of confusion is between Cargo and Tankers. This suggests that these two classes might have similar visual features, making it difficult for the model to distinguish between them. The model performs reasonably well, as indicated by the relatively strong diagonal values. However, the off-diagonal values highlight specific areas where the model struggles.

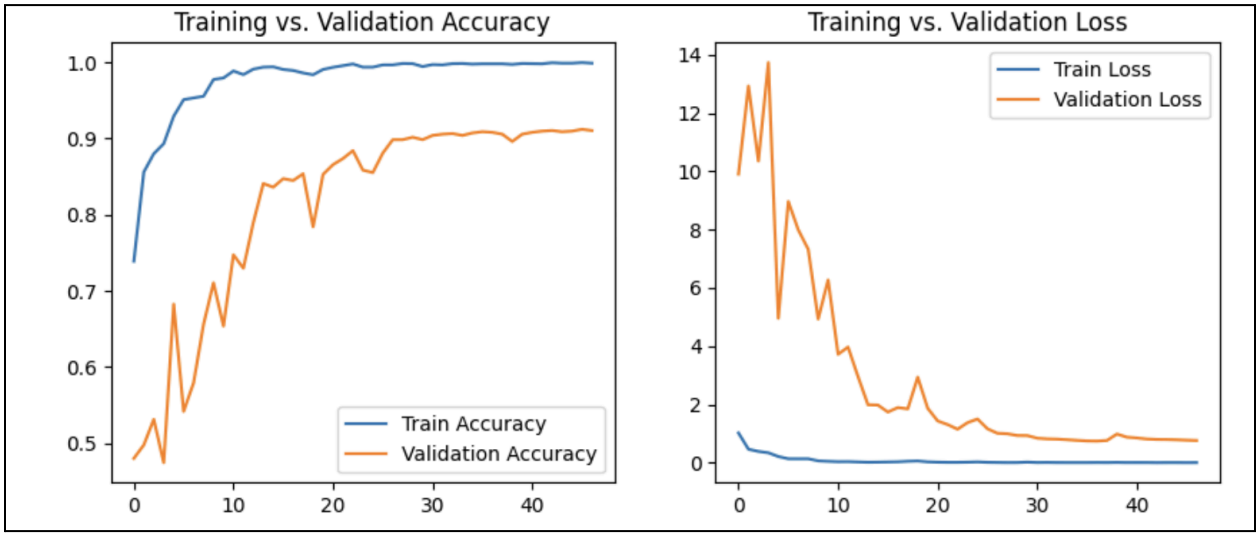
```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.74      0.78       424
           1       0.81      0.92      0.86       234
           2       0.96      0.93      0.94       183
           3       0.84      0.87      0.86       166
           4       0.72      0.77      0.74       244

    accuracy                           0.82      1251
   macro avg       0.83      0.85      0.84      1251
weighted avg       0.83      0.82      0.82      1251
```

The CNN model demonstrates reasonable overall performance with an accuracy of 82%, but exhibits significant class-specific variations. Notably, the model excels in classifying Carriers with high precision and recall, while struggling to distinguish between Cargo and Tankers, leading to lower precision for the latter. This confusion, coupled with potential class imbalance as indicated by varying support values, suggests the need for targeted improvements.

Model 2



- Training vs Validation Accuracy

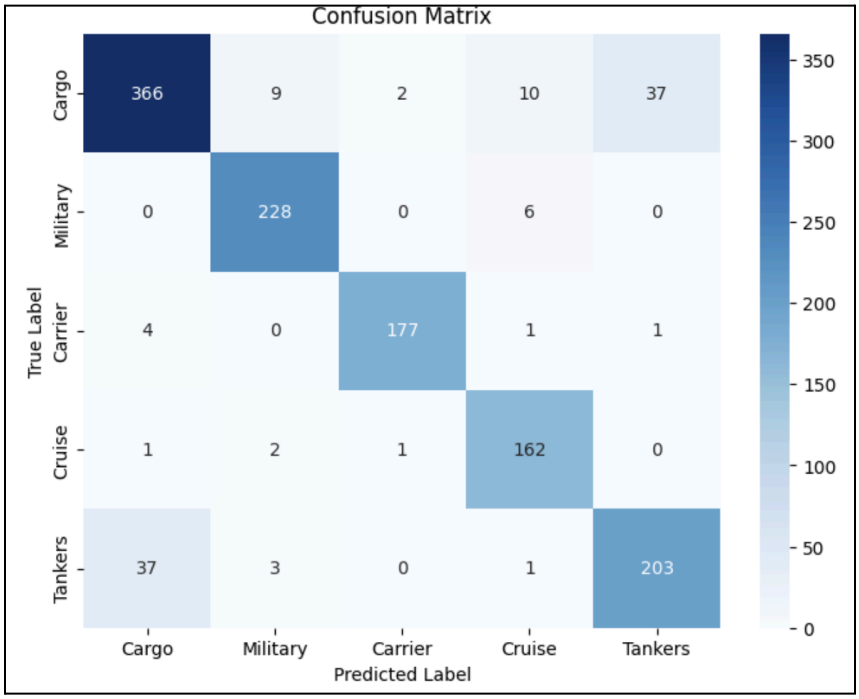
  The training accuracy increases rapidly in the initial epochs, similar to the first model.

  It quickly plateaus around 1.0 (or very close) after about 15 epochs. This indicates that the model is learning the training data very effectively.The validation accuracy also increases, but at a slower rate than the training accuracy. It plateaus around 0.9 (or slightly higher) after about 20 epochs. There is a gap between the training and validation accuracy, but it is smaller than the first model. This still indicates some degree of overfitting, but is improved. The validation accuracy is significantly higher than the first model.

- Training vs Validation Loss

  The training loss decreases rapidly in the initial epochs. It continues to decrease and approaches 0, similar to the first model. The validation loss decreases initially and then plateaus. The validation loss is much closer to the training loss in this model, which is a good sign. This indicates that the model is generalizing better than the first one. Also the Validation loss is much lower than the first model.

The graphs indicate that using a pre-trained model has significantly improved the model's performance by reducing overfitting and enhancing generalization. The model converges faster and achieves higher validation accuracy compared to the first model.



Class-Specific Performance:

- Cargo: 366 correctly classified. Some confusion with Tankers (37 misclassified as Tankers). Very little confusion with other classes.
- Military: 228 correctly classified. Very little confusion with other classes.

- Carrier: 177 correctly classified. Very little confusion with other classes.
- Cruise: 162 correctly classified. Very little confusion with other classes.
- Tankers: 203 correctly classified. Some confusion with Cargo (37 misclassified as Cargo). Very little confusion with other classes.

Compared to the first model, this model shows a significant improvement in overall accuracy and class-specific performance. The off-diagonal values are generally lower, indicating less confusion between classes. Similar to the first model, there is still some confusion between Cargo and Tankers. However, the confusion is less than the first model. The model performs very well in classifying Military, Carrier, and Cruise ships.

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 6s 104ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.86      0.88       424
           1       0.94      0.97      0.96       234
           2       0.98      0.97      0.98       183
           3       0.90      0.98      0.94       166
           4       0.84      0.83      0.84       244

    accuracy                           0.91      1251
   macro avg       0.91      0.92      0.92      1251
weighted avg       0.91      0.91      0.91      1251
```

The second CNN model demonstrates an improvement over the initial model, achieving an accuracy of 91% compared to 82%. This enhancement is evident across all evaluation metrics, with higher precision, recall, and F1-scores for each ship class. While the model excels in classifying Military, Carrier, and Cruise ships, some residual confusion persists between Cargo and Tankers, though significantly reduced. The pre-trained model effectively mitigates overfitting and enhances generalization, leading to better performance and faster convergence.

Discussion & Business Recommendation

OceanVision AI's development of a high-precision, AI-driven ship classification model directly addresses the critical need for automated and reliable maritime surveillance. By replacing the current manual processes with an efficient CNN-based solution, the company significantly impacts its core client base: maritime authorities, logistics firms, and defense agencies. The model's demonstrated 91% accuracy in classifying ships into five key categories—Cargo, Military, Carriers, Cruise, and Tankers—means operational improvements. For maritime authorities, this means enhanced real-time monitoring capabilities, enabling quicker responses to unauthorized activities and improved port security. Logistics firms benefit from optimized vessel tracking and cargo management, leading to reduced operational costs and increased efficiency. Defense agencies gain a strategic advantage through accurate identification of naval assets, supporting optimized naval operations and improved situational awareness. The model's ability to minimize misclassifications, a critical pain point in manual processes, directly reduces security risks and financial losses. This technology allows for scalability and 24/7 monitoring, a feat not possible with human labor, making it a critical tool for the future of maritime operations. Some strong business use cases include Automated port traffic management, Illegal fishing detection/prevention and oil spill monitoring and response.