# 1. Introduction

The rapid growth of remote sensing technologies using satellite images has significantly contributed to the development of surveillance and security systems for water bodies. Maritime monitoring has become increasingly essential for government bodies to detect and prevent criminal activities occurring in international waters, such as unlawful fishing, hijacking of ships, encroachment of sea borders, illicit exchange of sea cargo, accidents, and military attacks. Deep learning techniques have been widely adopted in remote sensing applications for image classification and object detection.

Traditional manual approaches to ship classification are time-consuming, expensive, and error-prone, limiting their effectiveness in real-time applications. Consequently, the development of an automated ship classification system using deep learning techniques has the potential to revolutionize maritime monitoring and management. Automated ship classification from optical aerial images in the visible spectrum can be utilized for various applications, such as monitoring illegal fishing activities, tracking cargo ship movements, and detecting oil spills or other environmental hazards. It can also aid in search and rescue operations by identifying distressed vessels in open waters.

**2. Description** of your individual work.

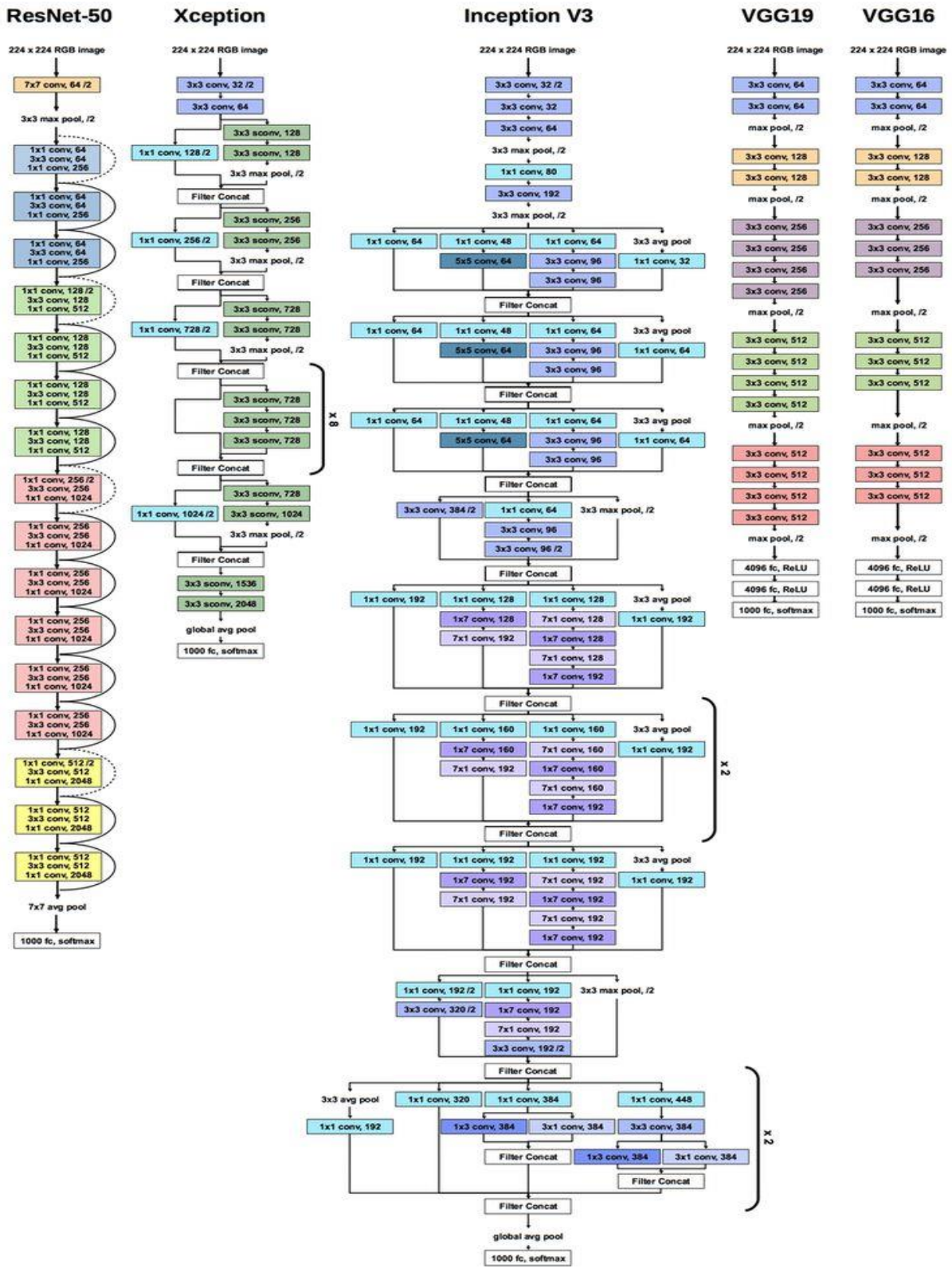


Fig1. Pre-trained models used in the project.

The pre-trained models used in this project are based on the work done by [1]. I added the additional layers for the pre-trained models as follows: -

| # | Layer | Output Size |
|---|---|---|
| 1 | Global Average Pooling<br>Fully-connected<br>Activation ReLU<br>Dropout 0.2 | 1 × 2048 |
| 2 | Fully-connected<br>Activation ReLU<br>Dropout 0.2 | 1 × 1256 |
| 3 | Fully-connected<br>SoftMax | 1 × (#classes) |

Fig2. Additional layers used in the project.

3. Describe the portion of the work that you did on the project in detail.

Modelling – Transfer learning (Codes): -
For pre-trained models, I set the trainable parameter to False since I wanted to use the weights used by these models for my dataset. This means that your model will train and update weights only on the additional layers. Also, the input shape to be provided here is based on the image size we are using for our problem. I set the parameter include_top = False since we don't want to include the classification layers at the top [3]. When we create an object of these classes, we pass the parameter num_classes = 7 since we have 7 classes in the target variable.
The call method is to ensure that we are passing the inputs to our pre-trained model. If we don't provide the call method, the default call method gets called which does not work for our dataset, so we use our own call method to override the existing call method.

```
3 usages  Akhil Bharadwaj +1
class VGG16(tf.keras.Model):
    Akhil Bharadwaj +1
    def __init__(self, num_classes, learning_rate):
        super(VGG16, self).__init__()
        self.vgg16 = tf.keras.applications.VGG16(include_top=False, weights='imagenet', input_shape=(Image_Size, Image_Size, CHANNELS))
        for layer in self.vgg16.layers:
            layer.trainable = False
        self.flatten = tf.keras.layers.Flatten()
        self.dense1 = tf.keras.layers.Dense(2048, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm1 = tf.keras.layers.BatchNormalization()
        self.dropout1 = tf.keras.layers.Dropout(0.2)
        self.dense2 = tf.keras.layers.Dense(1256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm2 = tf.keras.layers.BatchNormalization()
        self.dropout2 = tf.keras.layers.Dropout(0.2)
        self.dense3 = tf.keras.layers.Dense(num_classes, activation='softmax', kernel_regularizer=tf.keras.regularizers.l2(0.01))

    Akhil Bharadwaj +1
    def call(self, inputs):
        x = self.vgg16(inputs)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.batchnorm1(x)
        x = self.dropout1(x)
        x = self.dense2(x)
        x = self.batchnorm2(x)
        x = self.dropout2(x)
        outputs = self.dense3(x)
        return outputs
```

Fig3. VGG16 code

```
3 usages  Akhil Bharadwaj +1
class VGG19(tf.keras.Model):
    Akhil Bharadwaj +1
    def __init__(self, num_classes, learning_rate):
        super(VGG19, self).__init__()
        self.vgg19 = tf.keras.applications.VGG19(include_top=False, weights='imagenet', input_shape=(Image_Size, Image_Size, CHANNELS))
        for layer in self.vgg19.layers:
            layer.trainable = False
        self.flatten = tf.keras.layers.Flatten()
        self.dense1 = tf.keras.layers.Dense(2048, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm1 = tf.keras.layers.BatchNormalization()
        self.dropout1 = tf.keras.layers.Dropout(0.2)
        self.dense2 = tf.keras.layers.Dense(1256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm2 = tf.keras.layers.BatchNormalization()
        self.dropout2 = tf.keras.layers.Dropout(0.2)
        self.dense3 = tf.keras.layers.Dense(num_classes, activation='softmax', kernel_regularizer=tf.keras.regularizers.l2(0.01))

    Akhil Bharadwaj +1
    def call(self, inputs):
        x = self.vgg19(inputs)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.batchnorm1(x)
        x = self.dropout1(x)
        x = self.dense2(x)
        x = self.batchnorm2(x)
        x = self.dropout2(x)
        outputs = self.dense3(x)
        return outputs
```

Fig4. VGG19 code

```
3 usages  Akhil Bharadwaj +1
class InceptionModel(tf.keras.Model):
    Akhil Bharadwaj +1
    def __init__(self, num_classes, learning_rate):
        super(InceptionModel, self).__init__()

        # Load the pre-trained InceptionV3 model
        self.inceptionv3 = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet',
                                                             input_shape=(Image_Size, Image_Size, CHANNELS))

        # Freeze the weights of the pre-trained layers
        for layer in self.inceptionv3.layers:
            layer.trainable = False

        # Add custom layers on top of the pre-trained model
        self.avgpool = tf.keras.layers.GlobalAveragePooling2D()
        self.dense1 = tf.keras.layers.Dense(2048, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm1 = tf.keras.layers.BatchNormalization()
        self.dropout1 = tf.keras.layers.Dropout(0.2)
        self.dense2 = tf.keras.layers.Dense(1256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm2 = tf.keras.layers.BatchNormalization()
        self.dropout2 = tf.keras.layers.Dropout(0.2)
        self.dense3 = tf.keras.layers.Dense(num_classes, activation='softmax', kernel_regularizer=tf.keras.regularizers.l2(0.01))

    Akhil Bharadwaj +1
    def call(self, inputs):
        x = self.inceptionv3(inputs)
        x = self.avgpool(x)
        x = self.dense1(x)
        x = self.batchnorm1(x)
        x = self.dropout1(x)
        x = self.dense2(x)
        x = self.batchnorm2(x)
        x = self.dropout2(x)
        outputs = self.dense3(x)
        return outputs
```

Fig5. Inception code

```
3 usages  Akhil Bharadwaj +1
class ResNet50(tf.keras.Model):
    Akhil Bharadwaj +1
    def __init__(self, num_classes, learning_rate):
        super(ResNet50, self).__init__()
        self.resnet50 = tf.keras.applications.ResNet50(include_top=False, weights='imagenet', input_shape=(Image_Size, Image_Size, CHANNELS))
        for layer in self.resnet50.layers:
            layer.trainable = False
        self.flatten = tf.keras.layers.Flatten()
        self.dense1 = tf.keras.layers.Dense(2048, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm1 = tf.keras.layers.BatchNormalization()
        self.dropout1 = tf.keras.layers.Dropout(0.2)
        self.dense2 = tf.keras.layers.Dense(1256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm2 = tf.keras.layers.BatchNormalization()
        self.dropout2 = tf.keras.layers.Dropout(0.2)
        self.dense3 = tf.keras.layers.Dense(num_classes, activation='softmax', kernel_regularizer=tf.keras.regularizers.l2(0.01))

    Akhil Bharadwaj +1
    def call(self, inputs):
        x = self.resnet50(inputs)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.batchnorm1(x)
        x = self.dropout1(x)
        x = self.dense2(x)
        x = self.batchnorm2(x)
        x = self.dropout2(x)
        outputs = self.dense3(x)
        return outputs
```

Fig6. Resnet code

```
3 usages  Akhil Bharadwaj +1
class Xception(tf.keras.Model):
    Akhil Bharadwaj +1
    def __init__(self, num_classes, learning_rate):
        super(Xception, self).__init__()
        self.xception = tf.keras.applications.Xception(include_top=False, weights='imagenet', input_shape=(Image_Size, Image_Size, CHANNELS))
        for layer in self.xception.layers:
            layer.trainable = False
        self.flatten = tf.keras.layers.Flatten()
        self.dense1 = tf.keras.layers.Dense(2048, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm1 = tf.keras.layers.BatchNormalization()
        self.dropout1 = tf.keras.layers.Dropout(0.2)
        self.dense2 = tf.keras.layers.Dense(1256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))
        self.batchnorm2 = tf.keras.layers.BatchNormalization()
        self.dropout2 = tf.keras.layers.Dropout(0.2)
        self.dense3 = tf.keras.layers.Dense(num_classes, activation='softmax', kernel_regularizer=tf.keras.regularizers.l2(0.01))

    Akhil Bharadwaj +1
    def call(self, inputs):
        x = self.xception(inputs)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.batchnorm1(x)
        x = self.dropout1(x)
        x = self.dense2(x)
        x = self.batchnorm2(x)
        x = self.dropout2(x)
        outputs = self.dense3(x)
        return outputs
```

Fig7. Xception code

I used transfer learning using subclassing in TensorFlow for model.py script. After this one can directly add these models to train.py using the line: -

```
from model import VGG16, VGG19, InceptionModel, ResNet50, Xception
```

After that, I wrote the code for calling these models from model.py into train.py using the following lines: -

```
    if args.model == 'VGG16':
        vgg16 = VGG16(num_classes, learning_rate)
        train_model(vgg16, X_train, Y_train, X_val, Y_val, n_epochs, batch_size)
        evaluate(vgg16, X_train, Y_train, X_val, Y_val, X_test, Y_test)
    if args.model == 'VGG19':
        vgg19 = VGG19(num_classes, learning_rate)
        train_model(vgg19, X_train, Y_train, X_val, Y_val, n_epochs, batch_size)
        evaluate(vgg19, X_train, Y_train, X_val, Y_val, X_test, Y_test)
    if args.model == 'Inception':
        inception = InceptionModel(num_classes, learning_rate)
        train_model(inception, X_train, Y_train, X_val, Y_val, n_epochs, batch_size)
        evaluate(inception, X_train, Y_train, X_val, Y_val, X_test, Y_test)
    if args.model == 'Resnet':
        resnet = ResNet50(num_classes, learning_rate)
        train_model(resnet, X_train, Y_train, X_val, Y_val, n_epochs, batch_size)
        evaluate(resnet, X_train, Y_train, X_val, Y_val, X_test, Y_test)
    if args.model == 'Xception':
        xception = Xception(num_classes, learning_rate)
        train_model(xception, X_train, Y_train, X_val, Y_val, n_epochs, batch_size)
        evaluate(xception, X_train, Y_train, X_val, Y_val, X_test, Y_test)
    if args.model == 'CNNmodel':
        model = model_definition(num_classes, learning_rate)
        train_model(model, X_train, Y_train, X_val, Y_val, n_epochs, batch_size)
        evaluate(model, X_train, Y_train, X_val, Y_val, X_test, Y_test)
```

Fig8. Main code for command line arguments

## 4. Results.

| Model | Dataset | Accuracy | F1 Score | Recall | Precision |
|---|---|---|---|---|---|
| CNN model | Validation | 0.61 | 0.604 | 0.619 | 0.656 |
| | Train | 0.68 | 0.662 | 0.682 | 0.719 |
| | Test | 0.62 | 0.614 | 0.628 | 0.659 |
| VGG16 | **Validation** | **0.77** | **0.767** | **0.776** | **0.764** |
| | **Train** | **0.80** | **0.851** | **0.812** | **0.811** |
| | **Test** | **0.78** | **0.775** | **0.784** | **0.772** |
| VGG19 | Validation | 0.77 | 0.765 | 0.776 | 0.758 |
| | Train | 0.80 | 0.797 | 0.805 | 0.805 |
| | Test | 0.76 | 0.765 | 0.776 | 0.767 |
| Inception | Validation | 0.73 | 0.724 | 0.794 | 0.795 |
| | Train | 0.79 | 0.793 | 0.794 | 0.795 |
| | Test | 0.70 | 0.698 | 0.704 | 0.703 |
| ResNet50 | Validation | 0.47 | 0.457 | 0.478 | 0.487 |
| | Train | 0.55 | 0.526 | 0.550 | 0.529 |
| | Test | 0.45 | 0.425 | 0.454 | 0.452 |
| Xception | Validation | 0.74 | 0.730 | 0.740 | 0.727 |
| | Train | 0.78 | 0.772 | 0.782 | 0.790 |
| | Test | 0.72 | 0.707 | 0.720 | 0.726 |

Table1 Results for train, validation and test sets

When I saw the results for all the models, I wasn't quite satisfied with the results because I was hoping for better values for the F1 score. From the table above, I felt that the models used were maybe too complex for our dataset and probably simpler models could have worked better because the dataset that we are using is comparatively smaller and less complex than ImageNet. The reason why I feel that VGG16 gave the best results among all the models used above is using l2 regularization worked out best for VGG since it has a higher number of parameters among the models used and this could have probably helped in reducing overfitting since l2 regularization penalizes additional parameters that we are using in our models.

## 5. Summary and conclusions.

From our experiments, VGG16 turned out to be the best model among the other models. For, further improvements, we could try using higher quality satellite images like SAR (Synthetic Aperture Radar)

images if available publicly should give much better results since it is not possible to use visible spectrum images for night times [1]. If we look at the sample images for ship class alone, they are not very clearly found from optical sensor images so one could also possibly try running an object detection pipeline to locate ships from images more precisely.

**6.** Calculate the percentage of the code that you found or copied from the internet.

In train.py, below 3 lines taken from [2], modified the second line and added 24 lines as mentioned in Fig8, so
% of code copied = (3-1)/(3 + 24) * 100 = 7.4%

```
parser = argparse.ArgumentParser()
parser.add_argument('--model', help='VGG16, VGG19, Inception, Resnet, Xception, CNNmodel')
args = parser.parse_args()
```

In train.py, below 4 lines taken from Exam1, modified first and fourth line.
% of code copied = (4-2)/(4) * 50 = 50%

```
# image dataset path
code_dir = os.getcwd()
os.chdir("..") # Change to the parent directory
project_dir = os.getcwd()
image_dataset_path = project_dir + os.path.sep + 'Data'
```

## 7. References


[1] Feng, Y., Diao, W., Sun, X., Yan, M., & Gao, X. (2019, August 14). Towards automated ship detection and category recognition from high-resolution aerial images. MDPI. Retrieved April 20, 2023, from https://www.mdpi.com/2072-4292/11/16/1901

[2] https://docs.python.org/3/library/argparse.html

[3] https://www.tensorflow.org/tutorials/images/transfer_learning