**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.

The problem selected for this project is the classification of maritime scenes using optical aerial images from the visible spectrum. The goal is to detect and identify various objects in the images, such as ships, land, coast, and sea, to facilitate maritime monitoring and surveillance. This problem was chosen due to its significance in detecting and preventing criminal activities, accidents, and military attacks in international waters. Aiming to develop an automated ship classification system using deep learning techniques has the potential to greatly enhance our ability to monitor and manage maritime activities, positioning it as a major area of research in the fields of computer vision and remote sensing.

**Dataset description**

**Link to Download the dataset:** [**Click here**](https://www.kaggle.com/datasets/gasgallo/masati-shipdetection?resource=download)

**Composition:** The MASATI dataset is used for this project, consisting of 6,212 satellite images, which are categorized into seven classes: land, coast, sea, ship, multi, coast-ship, and detail.

**Image Details**: The images are captured in dynamic marine environments under varying weather and illumination conditions, acquired from Bing Maps in RGB format. The size of the images depends on the region of interest, with an average spatial resolution of around 512 x 512 pixels. The images are stored as PNG files, where pixel values represent RGB colors. The distance between targets and the acquisition satellite varies to obtain captures at different altitudes.

**Dataset Organization:** To label the category of each image, the dataset is organized into folders, where each folder represents a category. This organization facilitates the efficient handling of images for training and validation purposes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Main class** | **Sub-class** | **Samples** | **Description** |
| **Ship** | Ship | 1015 | Sea with a ship (no coast). |
|  | Detail | 1789 | Ship details. |
|  | Multi | 188 | Multiple ships |
|  | Coast & ship | 121 | Coast with ships. |
| **Non-ship** | Sea | 1010 | Sea (no ships). |
|  | Coast | 1054 | Coast (no ships). |
|  | Land | 1035 | Land (no sea). |

**Adequacy for Training:** The dataset is large enough to train a deep network, providing sufficient variation and complexity to build a robust model for maritime scene classification.

**Description of the deep learning network and training algorithm.**

1. **VGG16**

This model differed from previous high-performing models in several ways. First, it used a tiny 3×3 receptive field with a 1-pixel stride—for comparison, AlexNet used an 11×11 receptive field with a 4-pixel stride. The 3×3 filters combine to provide the function of a larger receptive field.

The benefit of using multiple smaller layers rather than a single large layer is that more non-linear activation layers accompany the convolution layers, improving the decision functions and allowing the network to converge quickly.

Second, VGG uses a smaller convolutional filter, which reduces the network’s tendency to over-fit during training exercises. A 3×3 filter is the optimal size because a smaller size cannot capture left-right and up-down information. Thus, VGG is the smallest possible model to understand an image’s spatial features. Consistent 3×3 convolutions make the network easy to manage.

Diagram

Description automatically generatedFig2. VGG16 Architecture (Source: [ResearchGate](https://www.researchgate.net/profile/Timea-Bezdan/publication/333242381/figure/fig2/AS:760979981860866@1558443174380/VGGNet-architecture-19.ppm))

1. **VGG19**

VGG-19 is composed of 16 convolutional and three fully connected layers. Both topologies use Dropout and Max-pooling techniques and ReLU activation functions.

Chart, waterfall chart

Description automatically generated  
Fig3. VGG19 Architecture

1. **Inception**

GoogLeNet, also known as Inception, is composed of 22 layers. Its main key point is that they designed a module called inception to handle the problem of computational efficiency. The inception module stacks a lot of those modules on top of each other to compute the network more efficiently. They’re applying several different kinds of filter operations in parallel on top of the same input coming into this same layer. Later they concatenate all these filter outputs together depth-wise, and so then this creates one tenser output at the end that is going to pass on to the next layer. Another point about the module is the bottleneck layer which is one-by-one kernel-sized filters. They reduce computational complexity by reducing the number of filters. Finally, there are no fully connected layers in the output of the network after the convolutional filters. Instead, they use a global average pooling layer which allows the usage of fewer parameters.

Diagram

Description automatically generated  
Fig4. Inception Architecture (Source: [Link](https://cloud.google.com/tpu/docs/inception-v3-advanced))

1. **ResNet 50**

The original ResNet architecture was ResNet-34, which comprised 34 weighted layers. It provided a novel way to add more convolutional layers to a CNN, without running into the vanishing gradient problem, using the concept of shortcut connections. A shortcut connection “skips over” some layers, converting a regular network to a residual network.

The regular network was based on the VGG neural networks (VGG-16 and VGG-19)—each convolutional network had a 3×3 filter. However, a ResNet has fewer filters and is less complex than a VGGNet. A 34-layer ResNet can achieve a performance of 3.6 billion FLOPs, and a smaller 18-layer ResNet can achieve 1.8 billion FLOPs, which is significantly faster than a VGG-19 Network with 19.6 billion FLOPs.

Diagram

Description automatically generated  
Fig5. ResNet50 Architecture (Source: [Link](https://www.researchgate.net/figure/The-architecture-of-ResNet-50-model_fig4_349717475))

1. **Xception**

These are convolutional neural network architecture based entirely on depth wise separable convolution layers. Fundamental hypothesis is mapping of cross-channels correlations and spatial correlations can be entirely decoupled. They are composed of 36 convolutional layers forming the feature extraction base of the network and structured into 14 modules, all of which have linear residual connections around them, except for the first and last modules.

Graphical user interface, application, Word

Description automatically generated  
Fig6. Xception Architecture (Source: [Link](https://stephan-osterburg.gitbook.io/coding/coding/ml-dl/tensorfow/ch3-xception/xception-architectural-design))

**Experimental setup**

**Pre-processing the data:**

The data is pre-processed by reading the images, resizing them, and applying image augmentation using ImageDataGenerator. The dataset is then split into training (80%), validation (10%), and testing (10%) sets using the train\_test\_split function.

**Model Definition:**

The CNN model is defined using TensorFlow Keras. Depending on the argument passed to the script, different pre-defined architectures such as VGG16, VGG19, Inception, ResNet50, and Xception can be used, or a custom CNN-KNN model can be built.

**Training the model:**

* The Adam optimizer is employed, utilizing sparse categorical cross-entropy loss and the accuracy metric to gauge performance.
* To address imbalanced datasets, class weights are calculated and incorporated into the training process.
* Early stopping is implemented as a safeguard against overfitting.
* Mini batches are utilized for training, with a batch size of 32 and a learning rate set at 0.001.

**Evaluating the model:**

* The trained model is evaluated using the train, validation, and test sets to measure its performance in terms of loss and accuracy.
* The features extracted from the CNN model are used to train a K-Nearest Neighbors (KNN) classifier.
* To optimize the KNN model's hyperparameters, randomized search cross-validation is employed.
* The KNN model is trained and appraised utilizing metrics such as f1 score, recall, and precision.
* To determine the size of the mini-batches, the batch size variable is set to 32, This size provides a good balance between computational efficiency and model convergence.
* The learning rate is set to 0.001, a common default value for the Adam optimizer.
* To detect and prevent overfitting, the code employs early stopping based on the validation accuracy. The training process will stop if the validation accuracy does not improve for ten consecutive epochs. This helps in avoiding training the model for too many epochs, which could lead to overfitting.
* Dropout layers and L2 regularization are used in the model to prevent overfitting by adding a penalty to the loss function and reducing the model's complexity.

**Results**

The table below summarizes the results of the experiments conducted using different models. The results include F1 score, recall, and precision for validation, train, and test datasets. The baseline model is compared with VGG16, VGG19, Inception, ResNet50, and Xception models.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Dataset** | **F1 Score** | **Recall** | **Precision** |
| Baseline | Validation | 0.231 | 0.278 | 0.603 |
|  | Train | 0.249 | 0.299 | 0.626 |
|  | Test | 0.242 | 0.297 | 0.634 |
| VGG16 | Validation | 0.784 | 0.793 | 0.793 |
|  | Train | 1.0 | 1.0 | 1.0 |
|  | Test | 0.783 | 0.792 | 0.781 |
| VGG19 | Validation | 0.770 | 0.784 | 0.768 |
|  | Train | 1.0 | 1.0 | 1.0 |
|  | Test | 0.758 | 0.773 | 0.776 |
| Inception | Validation | 0.700 | 0.709 | 0.695 |
|  | Train | 0.807 | 0.812 | 0.808 |
|  | Test | 0.733 | 0.733 | 0.714 |
| ResNet50 | Validation | 0.578 | 0.856 | 0.600 |
|  | Train | 0.625 | 0.635 | 0.644 |
|  | Test | 0.554 | 0.567 | 0.570 |
| Xception | Validation | 0.727 | 0.739 | 0.741 |
|  | Train | 1.0 | 1.0 | 1.0 |
|  | Test | 0.727 | 0.739 | 0.727 |

**VGG16 and VGG19** models show the best performance across all datasets, with VGG16 slightly outperforming VGG19. Both models achieve an F1 score of around 0.78 on the test dataset. Inception and Xception models also show satisfactory performance, with F1 scores around 0.73 for the test dataset. However, the ResNet50 model does not perform as well, with an F1 score of only 0.554 on the test dataset. The baseline model has the lowest performance, with F1 scores around 0.24 for all datasets.

**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] J. Alghazo, A. Bashar, G. Latif and M. Zikria, "Maritime Ship Detection using Convolutional Neural Networks from Satellite Images," 2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT), Bhopal, India, 2021, pp. 432-437, doi: 10.1109/CSNT51715.2021.9509628.

[3] *Deep residual learning for image recognition - arxiv*. (n.d.). Retrieved April 25, 2023, from https://arxiv.org/pdf/1512.03385.pdf