Final Project Group Report

Automated ship classification from satellite images using Deep Learning

Subject: Deep Learning

Professor: Amir Jafari

Team Members: Akhil Bharadwaj, Shumel Siraj Khan

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

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.

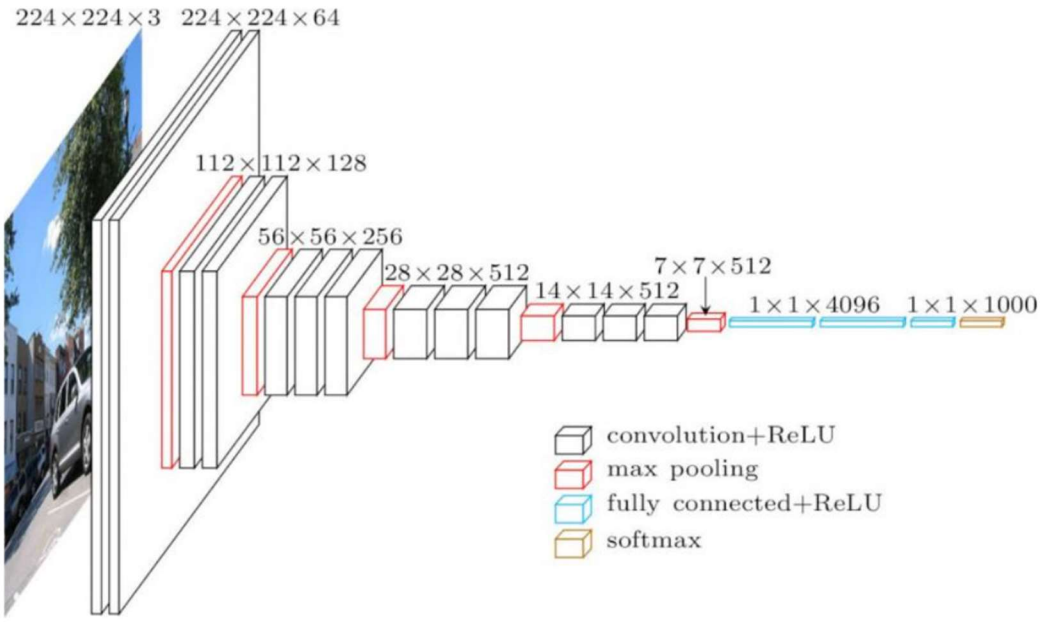

Fig2. VGG16 Architecture (Source: ResearchGate)

2. 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.

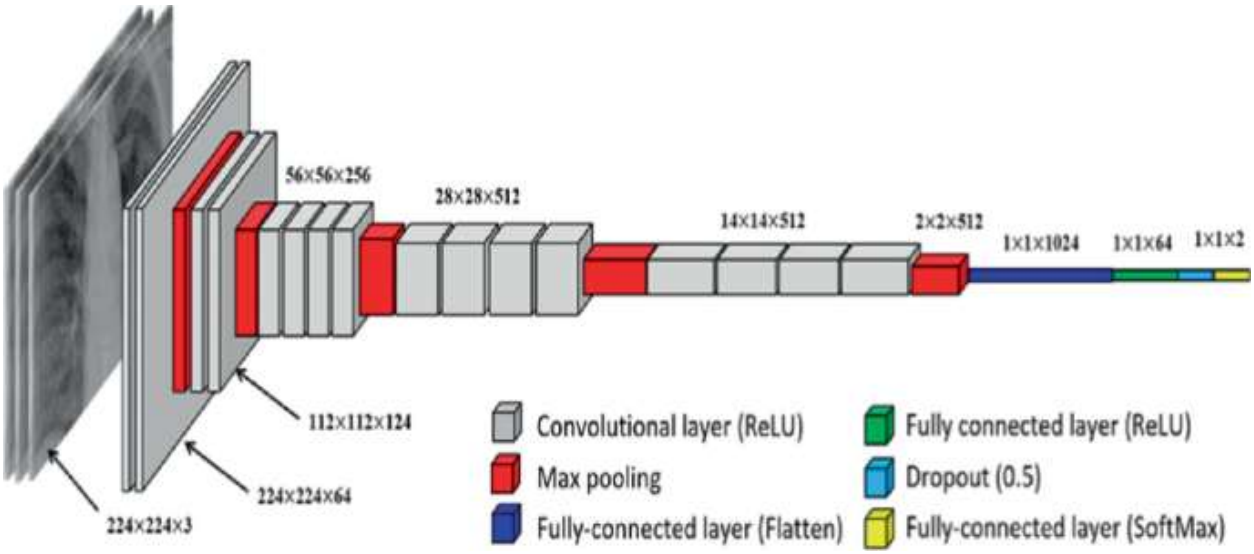

Fig3. VGG19 Architecture

3. 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.

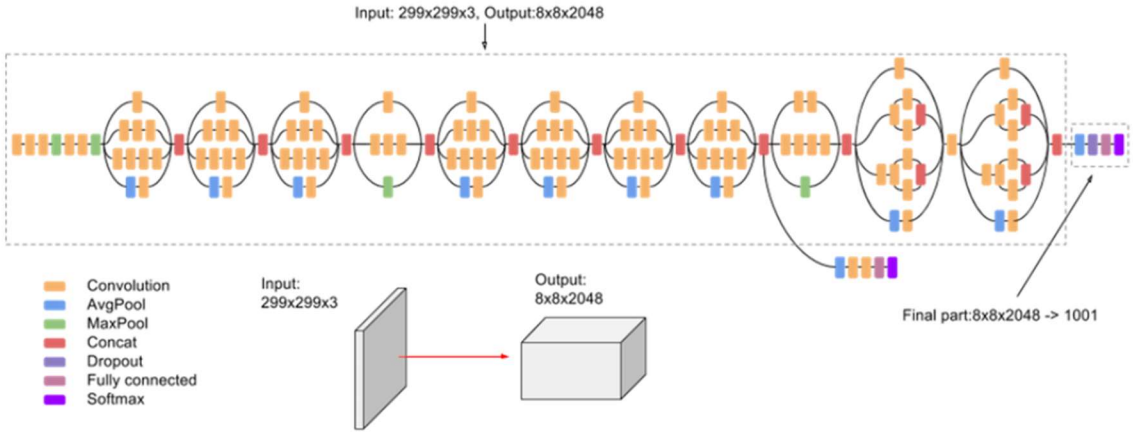

Fig4. Inception Architecture (Source: Link)

4. 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.

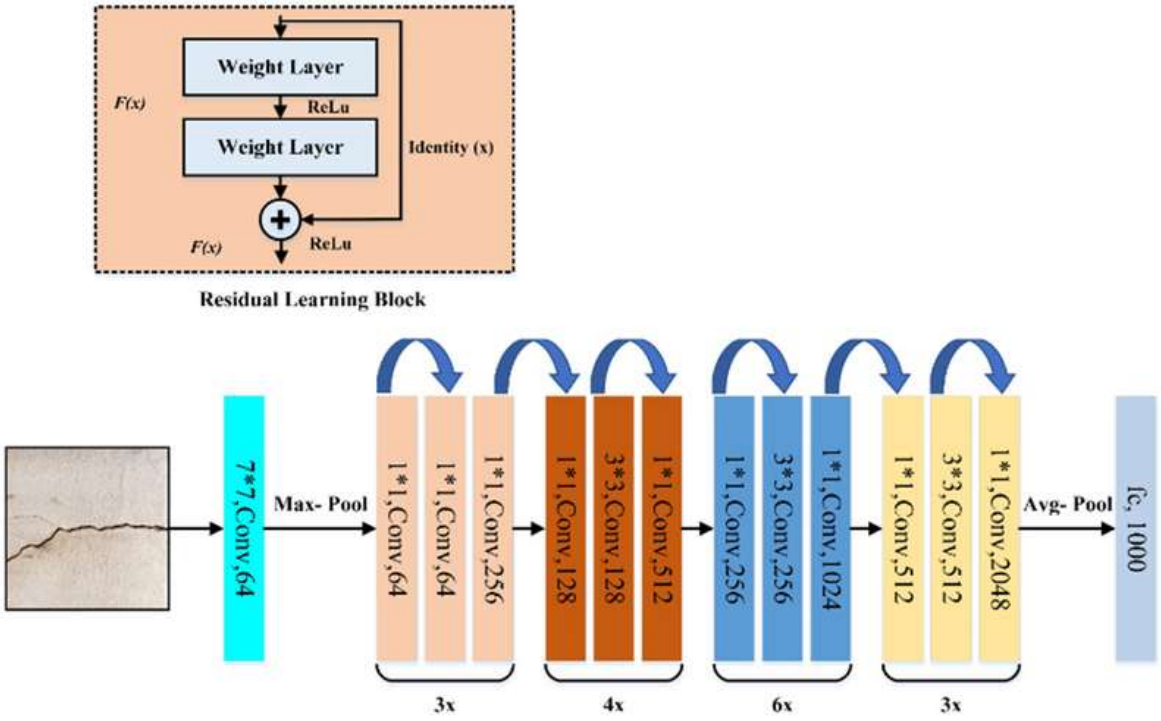

Fig5. ResNet50 Architecture (Source: Link)

5. 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.

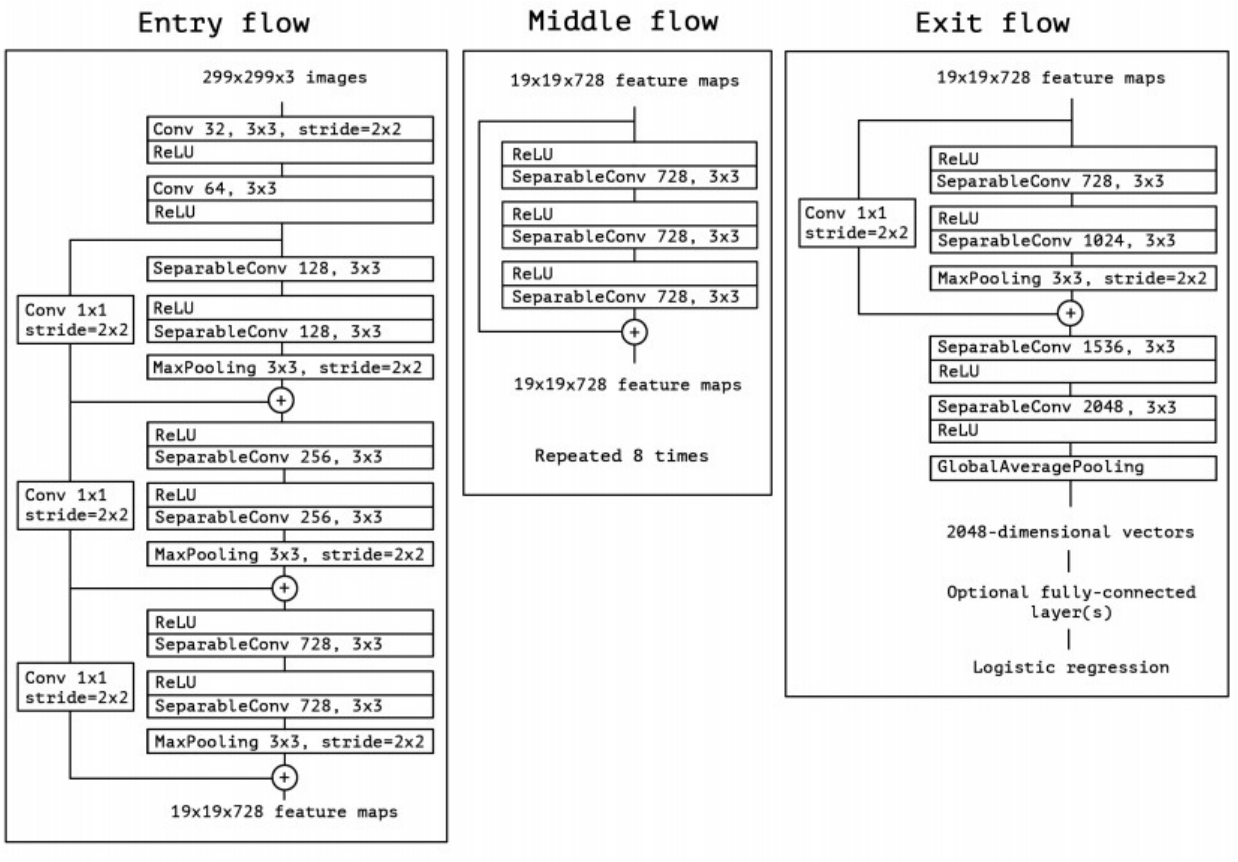

Fig6. Xception Architecture (Source: Link)

6. **CNN model**

To begin the project, we designed and implemented our own CNN model with the following architecture:

- A **Conv2D** layer with 64 filters of size (3,3) and ReLU activation function.
- A **BatchNormalization** layer is added to normalize the outputs of the first convolutional layer.
- A **MaxPooling2D** layer with pool size (2,2) is added to perform max pooling operation and reduce the spatial dimensions of the output.

- Another **Conv2D** layer with 32 filters of size (3,3) and ReLU activation function is added.
- Another **BatchNormalization** layer is added to normalize the outputs of the second convolutional layer.
- Another **MaxPooling2D** layer with pool size (2,2) is added to further reduce the spatial dimensions of the output.
- The output of the previous layer is flattened using a **Flatten** layer. This layer converts the 2D output of the convolutional layers into a 1D array.
- A fully connected **Dense** layer with 128 units and ReLU activation function is added. L2 regularization with a coefficient of 0.01 is applied using **kernel_regularizer** argument.
- A **Dropout** layer with a rate of 0.5 is added to reduce overfitting.
- Another fully connected **Dense** layer with 64 units and ReLU activation function is added. Another **Dropout** layer with a rate of 0.5 is added.
- The output layer is a fully connected **Dense** layer with **number of classes** units and a **softmax** activation function, which is used to classify the input images into **number of classes** different categories.

Experimental setup

Pre-processing the data:

The data is pre-processed by reading the images and applying image augmentation using **ImageDataGenerator** such as zoom, vertical and horizontal flip, rotation of 45 degree and resizing images to **100*100** size. The dataset is then split into training (80%), validation (10%), and testing (10%) sets using the **train_test_split** function. The resulting images and labels are then converted to NumPy arrays and normalized. Finally, the preprocessed data is saved to disk

using the **pickle** module, allowing for easy reuse of the dataset without having to preprocess it again.

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

We conducted a series of experiments to compare the performance of different pre-trained deep learning models on a given dataset. The models used in our experiments include Baseline, VGG16, VGG19, Inception, ResNet50, and Xception. The performance of each model was evaluated on validation, train, and test datasets, and the metrics used for the evaluation are accuracy, F1 score, recall, and precision.

The following table summarizes the results obtained from our experiments:

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

VGG16 and VGG19 models show the best performance across all datasets, with **VGG16** slightly outperforming VGG19. Both models achieve an F1 score of around 0.77 on the test dataset. Inception and Xception models also show satisfactory performance, with F1 scores around 0.70 for the test dataset. However, the CNN model does not perform as well, with an F1 score of only 0.61 on the test dataset. Finally, the ResNet model has the lowest performance, with F! score around 0.425 on test dataset.

Summary and conclusions

In the study of automated ship classification from satellite images, the VGG16 model demonstrated superior performance compared to other pre-trained deep learning models. Specifically, VGG16 achieved an F1 score of approximately 0.77 on the test dataset, indicating a high level of accuracy and precision in ship classification. This result suggests that the VGG16 model is highly effective for ship classification tasks using satellite images and can be a valuable tool for various maritime monitoring applications.

Future research could focus on the following aspects to enhance the performance of ship classification models:

- **Model optimization:** Exploring different model architectures, hyperparameters, and training strategies could potentially lead to even better performance.
- **Transfer learning:** Investigating the effectiveness of other pre-trained models or fine-tuning the VGG16 model on specific ship classes might yield improved results for specialized maritime monitoring tasks.
- **Feature extraction and fusion**: Combining features from multiple pre-trained models or leveraging additional information from satellite images, such as spectral or temporal features, could enhance the model's performance in ship classification tasks.
- **Utilizing GANs:** Employing Generative Adversarial Networks (GANs) to generate images for balancing the minority class in the dataset.

By addressing these areas, future research can continue to refine and improve upon the performance of automated ship classification models for satellite imagery, contributing to more effective maritime monitoring and management.

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