

# Ship Classification using Deep Learning

## Introduction

The maritime industry plays a pivotal role in global trade, with ships serving as the lifeblood of international commerce. Accurate and efficient ship classification is essential for various applications including maritime security, environmental monitoring and trade regulation.

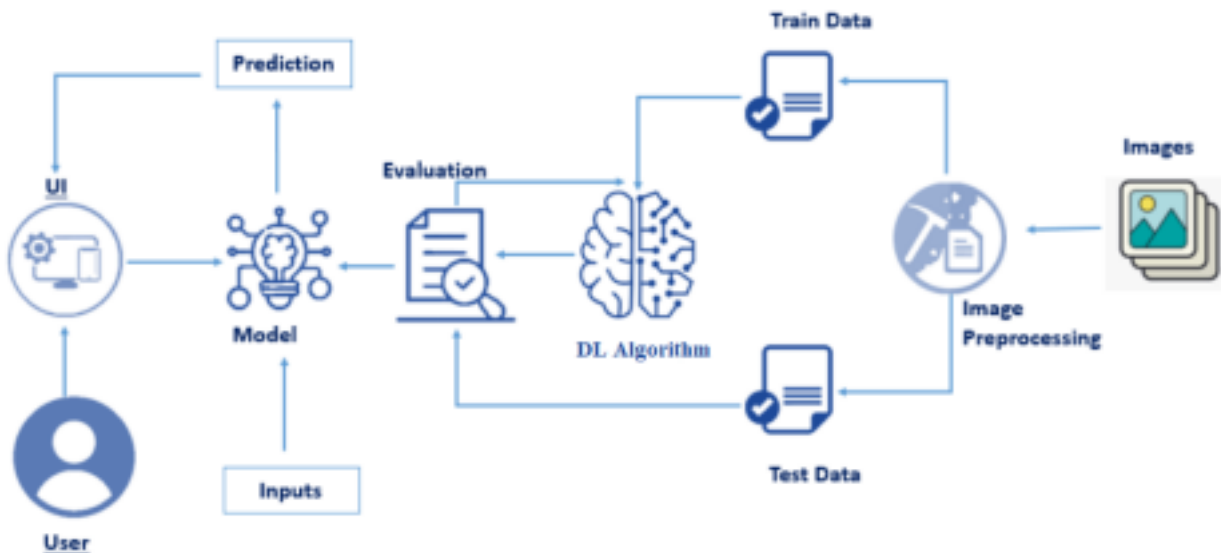
Traditional ship classification methods have often relied on manual inspection, which can be time-consuming and error-prone. With the advent of deep learning techniques, there is a tremendous opportunity to revolutionize ship classification by automating the process with high accuracy and speed.

Ship classification involves categorizing vessels into various classes such as military, cargo, cruise, tanker etc. bases on their size, type, function and other attributes. Deep learning models such as Convolution Neural Networks (CNNs) have demonstrated their ability to extract intricate features from images, enabling the development of accurate and efficient ship classification systems.

The images in the data belong to 5 categories of ships - Cargo, Carrier, Military, Cruise and Tankers.

The model used to train the model is VGG19, a computer vision model which is based on Convolutional Neural Networks (CNN). We will be using the pre-trained weights of the model and modify the top layer for performing our custom classification. The model is deployed using the Flask framework.

## Technical Architecture



## Prerequisites

To complete this project, you would need the following software's, concepts and packages

- Anaconda Navigator: It is a free and open-source distribution of the python and R programming languages for data science and machine learning related applications.

It can be installed on Windows, Linux and macOS. It also comes with nice tools like JupyterLab, Jupyter Notebook etc.

To build the Machine learning models you would need the following packages

- Numpy: It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations.
- Scikit-learn: It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, etc. and also supports Numpy.
- Flask: It is a web framework used to build Web Applications
- Keras: It is an open-source library that provides a Python interface for artificial neural networks. It also acts as an interface for the TensorFlow library.
- Tensorflow: TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML powered applications.
- Pandas: This is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.

If you are using anaconda navigator, follow below steps to download required packages:

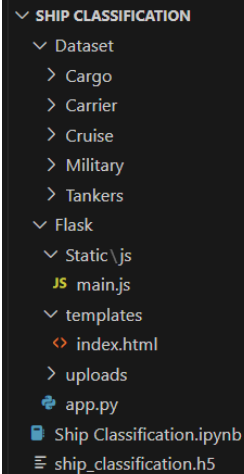
- Open anaconda prompt.
- Type "pip install numpy" and click enter.
- Type "pip install pandas" and click enter.
- Type "pip install tensorflow==2.8.2" and click enter.
- Type "pip install keras=2.8.0" and click enter.
- Type "pip install Flask" and click enter.
- Type "pip install requests" and click enter.

## Deep Learning Concepts

CNN: A convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.

Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

## Project Structure



The dataset folder contains the images of different categories of ships in their respective folders.

The flask folder has all necessary files to build the flask application.

The static folder contains the images and the style sheets needed to customize the web page.

The templates folder has the HTML pages.

App.py is the python script for server-side computing.

Ship Classification.ipynb is the notebook on which the code is executed.

## Project Objectives

By the end of this project, you'll understand:

- How to preprocess the images.
- Training VGG19 model with custom data.
- How pre-trained models will be useful in object classification.
- How to evaluate the model.
- Building a web application using the Flask framework.

## Project Flow

Preparing the data

- Downloading the dataset
- Categorize the images

Data pre-processing

- Import ImageDataGenerator Library and Configure it
- Apply ImageDataGenerator functionality to Train and Test set

Building the model

- Creating and compiling the pre-trained model
- Look at the model summary
- Train the model while monitoring validation loss
- Test the model with custom inputs

## Building Flask application

- Build a flask application
- Build the HTML page and execute

## Milestone 1: Preparing the data

The images need to be organized before proceeding with the project.

### Activity 1: Downloading the dataset

Create Train and Test folders with each folder having subfolders with ship images of different types. You can collect the data from the below link:

<https://www.kaggle.com/code/abdullahhaxsh/ship-classifier-using-cnn/data>

### Activity 2: Categorize train images

The original dataset has a single folder known as images. We will be using the train.csv file to fetch the image Id's of training images and sort them according to their category. The train.csv file looks like below:

```
import pandas as pd
✓ 3.7s

df = pd.read_csv(r"C:\Users\shobi\OneDrive\Desktop\Ship Classification\Dataset\train.csv")
df.head()
✓ 0.1s
```

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

The category column contains the encoded values of the category the ship in the corresponding image belongs to. Let us add another column that contains the decoded value of category:

```
ship = {1:"Cargo",2:"Military",3:"Carrier",4:"Cruise",5:"Tankers"}
df["ship"] = df["category"].map(ship).astype('category')
df.head()
✓ 0.0s
```

	image	category	ship
0	2823080.jpg	1	Cargo
1	2870024.jpg	1	Cargo
2	2662125.jpg	2	Military
3	2900420.jpg	3	Carrier
4	2804883.jpg	2	Military

Let us now organize our train images into folders representing their corresponding categories.

```

classes = list(ship.values())
import os
for i in classes:
    os.makedirs(os.path.join("C:/Users/shobi/OneDrive/Desktop/Ship Classification/Dataset/Train",i))
labels = df.sort_values('ship')

import shutil
for c in classes:
    for i in list(labels[labels['ship']==c]['image']):
        image = os.path.join("C:/Users/shobi/OneDrive/Desktop/Ship Classification/Dataset/Train/images",i)
        move = shutil.move(image,"C:/Users/shobi/OneDrive/Desktop/Ship Classification/Dataset/Train/"+c)

```

As seen in the above image, we have used a dictionary to map the values to the corresponding category that the ship belongs to. Let us now organize our train images into folders representing their corresponding categories.

#### ✓ SHIP CLASSIFICATION

- ✓ Dataset
  - > Cargo
  - > Carrier
  - > Cruise
  - > Military
  - > Tankers

## Milestone 2: Data Pre-Processing

The dataset images are to be preprocessed before giving it to the model.

### Activity 1: Import ImageDataGenerator Library and Configure it

ImageDataGenerator class is used to augment the images with different modifications like considering the rotation, flipping the image etc.

```

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_gen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

```

### Activity 2: Apply ImageDataGenerator functionality to Train and Validation set

Specify the path of both the folders in flow\_from\_directory method. We are importing the images in 300\*300 pixels.

```

train_batch = train_gen.flow_from_directory(
    directory="/content/Dataset",
    target_size=(300, 300),
    batch_size=64,
    subset='training',
    class_mode = 'categorical',
    seed=64
)

valid_batch = train_gen.flow_from_directory(
    directory="/content/Dataset",
    target_size=(300, 300),
    batch_size=64,
    subset='validation',
    class_mode = 'categorical',
    seed=64
)

```

Found 5003 images belonging to 5 classes.  
Found 1249 images belonging to 5 classes.

### Milestone 3: Building the model

We will be creating the pre-trained VGG16 model and Inception V3 model along with ResNet50 model for predicting custom classes and choose the model with the highest accuracy.

#### Activity 1: Creating and compiling the model

##### VGG19

Firstly, import the necessary libraries and define a function for creation of the model. Next, call the pre-trained model with the parameter include\_top=False, as we need to use the model for predicting custom images.

```

from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.optimizers import Adam

```

```

base_model = VGG19(weights='imagenet',
                    include_top=False,
                    input_shape=(300, 300, 3))

```

## Inception V3

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Model
import tensorflow as tf

# Load the InceptionV3 model
inception_model = InceptionV3(input_shape=(248, 248, 3), include_top=False, weights='imagenet')
inception_model.trainable = False
```

## ResNet50

```
resnet_model = Sequential()
pretrained_model = ResNet50(include_top=False,
                             input_shape=(240, 240, 3),
                             pooling='avg',
                             weights='imagenet')
for layer in pretrained_model.layers:
    layer.trainable = False
```

Next, add dense layers to the top model. Finally, add an output layer with the number of neurons equal to out output classes.

## VGG19

```
for layer in base_model.layers:
    layer.trainable = False

x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
output = Dense(5, activation='softmax')(x)
```

## Inception V3

```
# Add custom layers on top of InceptionV3
x = tf.keras.layers.Flatten()(transfer_final_layer)
x = tf.keras.layers.Dense(5, activation='softmax')(x) # Assuming you have 5 classes

# Create the final transfer model
inception_model = Model(inception_model.input, x)
```

## ResNet50

```
resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(BatchNormalization())
resnet_model.add(Dense(128, activation='relu'))
resnet_model.add(BatchNormalization())
resnet_model.add(Dense(5, activation='softmax'))
```

Finally, compile the model.

### VGG19

```
model = Model(inputs=base_model.input, outputs=output)
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

### Inception V3

```
inception_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    metrics=['accuracy'],
    loss='categorical_crossentropy' # Use categorical for multi-class classification
)
```

### Resnet50

```
resnet_model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

### Activity 2: Look at the model summary

Call the summary() function to have a look at the model summary:

### VGG19

```
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 300, 300, 3)]	0
block1_conv1 (Conv2D)	(None, 300, 300, 64)	1792
block1_conv2 (Conv2D)	(None, 300, 300, 64)	36928
block1_pool (MaxPooling2D)	(None, 150, 150, 64)	0
block2_conv1 (Conv2D)	(None, 150, 150, 128)	73856
block2_conv2 (Conv2D)	(None, 150, 150, 128)	147584
block2_pool (MaxPooling2D)	(None, 75, 75, 128)	0
block3_conv1 (Conv2D)	(None, 75, 75, 256)	295168
block3_conv2 (Conv2D)	(None, 75, 75, 256)	590080
block3_conv3 (Conv2D)	(None, 75, 75, 256)	590080
block3_conv4 (Conv2D)	(None, 75, 75, 256)	590080
...		
Total params: 30682949 (117.05 MB)		
Trainable params: 10658565 (40.66 MB)		
Non-trainable params: 20024384 (76.39 MB)		



## Inception V3

```
inception_model.summary()
```

Model: "model\_5"

Layer (type)	Output Shape	Param #	Connected to
input_8 (InputLayer)	[(None, 248, 248, 3)]	0	[]
conv2d_470 (Conv2D)	(None, 123, 123, 32)	864	['input_8[0][0]']
batch_normalization_472 (Batch Normalization)	(None, 123, 123, 32)	96	['conv2d_470[0][0]']
activation_470 (Activation)	(None, 123, 123, 32)	0	['batch_normalization_472[0][0]']
conv2d_471 (Conv2D)	(None, 121, 121, 32)	9216	['activation_470[0][0]']
batch_normalization_473 (Batch Normalization)	(None, 121, 121, 32)	96	['conv2d_471[0][0]']
activation_471 (Activation)	(None, 121, 121, 32)	0	['batch_normalization_473[0][0]']
conv2d_472 (Conv2D)	(None, 121, 121, 64)	18432	['activation_471[0][0]']
batch_normalization_474 (Batch Normalization)	(None, 121, 121, 64)	192	['conv2d_472[0][0]']
...			
Total params: 22171429 (84.58 MB)			
Trainable params: 368645 (1.41 MB)			
Non-trainable params: 21802784 (83.17 MB)			

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

## Milestone 5: Training and testing the model

### Activity 1: Train the model while monitoring validation loss

We will be training the model for 12 epochs using the fit() function:

### VGG19

```
history = model.fit(train_batch,
                    steps_per_epoch=len(train_batch),
                    epochs=12,
                    validation_data=valid_batch,
                    validation_steps=len(valid_batch))
```

```
Epoch 1/12
79/79 [=====] - 186s 2s/step - loss: 1.2540 - accuracy: 0.5607 - val_loss: 0.6341 - val_accuracy: 0.7510
Epoch 2/12
79/79 [=====] - 163s 2s/step - loss: 0.5388 - accuracy: 0.7943 - val_loss: 0.5363 - val_accuracy: 0.7902
Epoch 3/12
79/79 [=====] - 152s 2s/step - loss: 0.4042 - accuracy: 0.8393 - val_loss: 0.4882 - val_accuracy: 0.8062
Epoch 4/12
79/79 [=====] - 151s 2s/step - loss: 0.4007 - accuracy: 0.8393 - val_loss: 0.5116 - val_accuracy: 0.8070
Epoch 5/12
79/79 [=====] - 152s 2s/step - loss: 0.3065 - accuracy: 0.8795 - val_loss: 0.6296 - val_accuracy: 0.7518
Epoch 6/12
79/79 [=====] - 151s 2s/step - loss: 0.2896 - accuracy: 0.8851 - val_loss: 0.5136 - val_accuracy: 0.8151
Epoch 7/12
79/79 [=====] - 151s 2s/step - loss: 0.2220 - accuracy: 0.9157 - val_loss: 0.5521 - val_accuracy: 0.7966
Epoch 8/12
79/79 [=====] - 151s 2s/step - loss: 0.2313 - accuracy: 0.9083 - val_loss: 0.5763 - val_accuracy: 0.7982
Epoch 9/12
79/79 [=====] - 150s 2s/step - loss: 0.2022 - accuracy: 0.9182 - val_loss: 0.6423 - val_accuracy: 0.8022
Epoch 10/12
79/79 [=====] - 148s 2s/step - loss: 0.1977 - accuracy: 0.9266 - val_loss: 0.5631 - val_accuracy: 0.8127
Epoch 11/12
79/79 [=====] - 150s 2s/step - loss: 0.1941 - accuracy: 0.9230 - val_loss: 0.6318 - val_accuracy: 0.7814
Epoch 12/12
79/79 [=====] - 151s 2s/step - loss: 0.1879 - accuracy: 0.9260 - val_loss: 0.4851 - val_accuracy: 0.8319
```

## Inception V3

```
# Train the model
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=4)
inception_model.fit(train_batch, validation_data=valid_batch, verbose=1, epochs=12, callbacks=[callback])

Epoch 1/12
79/79 [=====] - 129s 2s/step - loss: 0.6129 - accuracy: 0.7721 - val_loss: 0.4604 - val_accuracy: 0.8303
Epoch 2/12
79/79 [=====] - 93s 1s/step - loss: 0.3578 - accuracy: 0.8641 - val_loss: 0.4555 - val_accuracy: 0.8375
Epoch 3/12
79/79 [=====] - 94s 1s/step - loss: 0.2976 - accuracy: 0.8865 - val_loss: 0.4156 - val_accuracy: 0.8455
Epoch 4/12
79/79 [=====] - 91s 1s/step - loss: 0.2582 - accuracy: 0.9061 - val_loss: 0.4036 - val_accuracy: 0.8599
Epoch 5/12
79/79 [=====] - 91s 1s/step - loss: 0.2063 - accuracy: 0.9214 - val_loss: 0.3943 - val_accuracy: 0.8711
Epoch 6/12
79/79 [=====] - 89s 1s/step - loss: 0.2037 - accuracy: 0.9220 - val_loss: 0.3858 - val_accuracy: 0.8543
Epoch 7/12
79/79 [=====] - 88s 1s/step - loss: 0.1865 - accuracy: 0.9310 - val_loss: 0.3691 - val_accuracy: 0.8599
Epoch 8/12
79/79 [=====] - 88s 1s/step - loss: 0.1832 - accuracy: 0.9296 - val_loss: 0.3900 - val_accuracy: 0.8591
Epoch 9/12
79/79 [=====] - 90s 1s/step - loss: 0.1733 - accuracy: 0.9312 - val_loss: 0.3661 - val_accuracy: 0.8687
Epoch 10/12
79/79 [=====] - 88s 1s/step - loss: 0.1441 - accuracy: 0.9466 - val_loss: 0.3616 - val_accuracy: 0.8671
Epoch 11/12
79/79 [=====] - 91s 1s/step - loss: 0.1296 - accuracy: 0.9538 - val_loss: 0.3457 - val_accuracy: 0.8735
Epoch 12/12
79/79 [=====] - 90s 1s/step - loss: 0.1232 - accuracy: 0.9558 - val_loss: 0.3623 - val_accuracy: 0.8767

<keras.src.callbacks.History at 0x7a41cd0a03a0>
```

## ResNet50

```
history = resnet_model.fit(training_data, steps_per_epoch=len(training_data), epochs=20, validation_data=validation_data, validation_steps=len(validation_data))

Epoch 1/20
72/72 [=====] - 29s 335ms/step - loss: 0.6995 - accuracy: 0.7535 - val_loss: 0.5793 - val_accuracy: 0.7904
Epoch 2/20
72/72 [=====] - 25s 338ms/step - loss: 0.2549 - accuracy: 0.9144 - val_loss: 0.4395 - val_accuracy: 0.8488
Epoch 3/20
72/72 [=====] - 24s 324ms/step - loss: 0.1391 - accuracy: 0.9652 - val_loss: 0.3891 - val_accuracy: 0.8600
Epoch 4/20
72/72 [=====] - 25s 343ms/step - loss: 0.0797 - accuracy: 0.9846 - val_loss: 0.3590 - val_accuracy: 0.8688
Epoch 5/20
72/72 [=====] - 24s 330ms/step - loss: 0.0462 - accuracy: 0.9930 - val_loss: 0.3787 - val_accuracy: 0.8744
Epoch 6/20
72/72 [=====] - 24s 323ms/step - loss: 0.0331 - accuracy: 0.9982 - val_loss: 0.3854 - val_accuracy: 0.8656
Epoch 7/20
72/72 [=====] - 24s 327ms/step - loss: 0.0211 - accuracy: 0.9980 - val_loss: 0.3958 - val_accuracy: 0.8680
Epoch 8/20
72/72 [=====] - 25s 336ms/step - loss: 0.0152 - accuracy: 0.9994 - val_loss: 0.4264 - val_accuracy: 0.8744
Epoch 9/20
72/72 [=====] - 24s 321ms/step - loss: 0.0117 - accuracy: 0.9992 - val_loss: 0.4633 - val_accuracy: 0.8632
Epoch 10/20
72/72 [=====] - 24s 335ms/step - loss: 0.0095 - accuracy: 0.9996 - val_loss: 0.4434 - val_accuracy: 0.8720
Epoch 11/20
72/72 [=====] - 24s 326ms/step - loss: 0.0077 - accuracy: 0.9994 - val_loss: 0.4566 - val_accuracy: 0.8680
Epoch 12/20
72/72 [=====] - 25s 334ms/step - loss: 0.0119 - accuracy: 0.9982 - val_loss: 0.4826 - val_accuracy: 0.8544
Epoch 13/20
...
Epoch 19/20
72/72 [=====] - 24s 326ms/step - loss: 0.0051 - accuracy: 0.9998 - val_loss: 0.5352 - val_accuracy: 0.8680
Epoch 20/20
72/72 [=====] - 24s 335ms/step - loss: 0.0032 - accuracy: 1.0000 - val_loss: 0.5363 - val_accuracy: 0.8736
```

Save the model after training it.

```
model.save("vgg19.keras")
```

```
inception_model.save("inception_model.keras")
```

```
resnet_model.save("resnet_model.keras")
```

Since the Inception model exhibits the highest level of accuracy, we have decided to select this model for our project.

### Activity 2: Test the model with custom inputs

First, specify the path of the image to be tested. Then, preprocess the image and perform predictions.

```
from tensorflow.keras.preprocessing import image
import numpy as np

class_labels = ['Cargo', 'Carrier', 'Cruise', 'Military', 'Tankers'] # Update with your actual class labels

# Load and preprocess the image
img = image.load_img('/content/Dataset/Cruise/2839958.jpg', target_size=(248, 248))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0) # Add a batch dimension

# Make predictions
predictions = inception_model.predict(img_array)
predicted_class_index = np.argmax(predictions)
# Get the predicted class label
predicted_class = class_labels[predicted_class_index]
print("Predicted class:", predicted_class)

# Load and preprocess the image
img = image.load_img('/content/Dataset/Military/1048999.jpg', target_size=(248, 248))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0) # Add a batch dimension

# Make predictions
predictions = inception_model.predict(img_array)
# Get the class with the highest probability
predicted_class_index = np.argmax(predictions)
# Get the predicted class label
predicted_class = class_labels[predicted_class_index]
print("Predicted class:", predicted_class)

1/1 [=====] - 0s 104ms/step
Predicted class: Cruise
1/1 [=====] - 0s 69ms/step
Predicted class: Military
```

As we can see the custom images have been predicted successfully.

## Milestone 5: Building Flask application

After the model is built, we will be integrating it to a web application so that normal users can also use it. The new users need to initially register in the portal. After registration users can login to browse the images to detect the category of ships.

### Activity 1: Build a python application

Step 1: Load the required packages

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask, render_template, request
import os
import numpy as np
```

Step 2: Initialize the flask app, load the model and configure the html pages

Instance of Flask is created and the model is loaded using load\_model from keras.

```
app = Flask(__name__)
model = load_model(r"ship_classification.h5", compile = False)
@app.route('/')
def index():
    return render_template("index.html")
@app.route('/predict', methods = ['GET', 'POST'])
```

Step 3: Pre-process the frame and run

```
def upload():
    if request.method == 'POST':
        f = request.files['image']
        basepath = os.path.dirname(__file__)
        filepath = os.path.join(basepath, 'uploads', f.filename)
        f.save(filepath)
        img = image.load_img(filepath, target_size = (248, 248))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis = 0)
        pred = np.argmax(model.predict(x), axis = 1)
        index = ['Cargo', 'Carrier', 'Cruise', 'Military', 'Tankers']
        text = "The classified Ship is : " + index[pred[0]]
    return text
if __name__ == '__main__':
    app.run(debug=True)
```

Run the flask application using the run method. By default, the flask runs on port 5000. If the port is to be changed, an argument can be passed and the port can be modified.

## Activity 2: Build the HTML page and execute

Build the UI where a home page will have details about the application and prediction page where a user is allowed to browse an image and get the predictions of images.

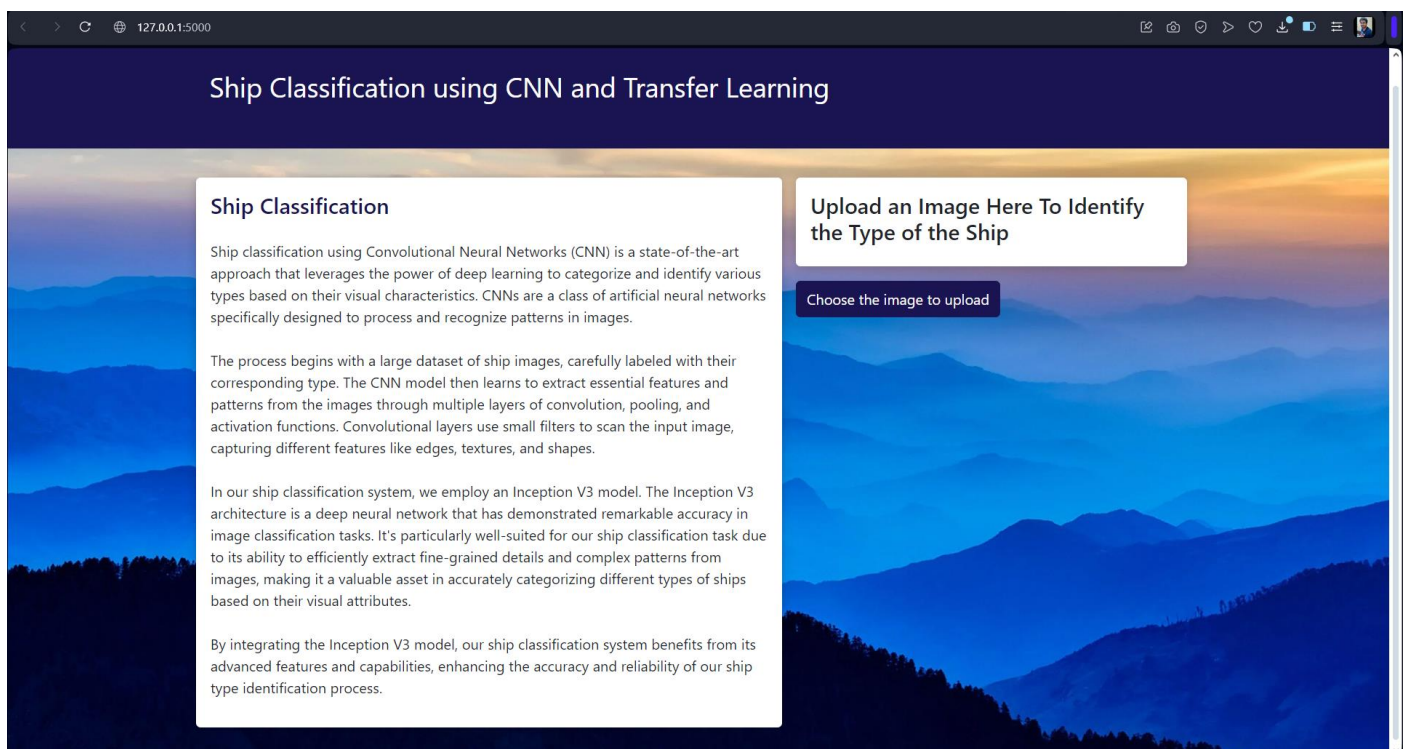
### Step 1: Run the application

In the anaconda prompt, navigate to the folder in which the flask app is present. When the python file is executed, the localhost is activated on port 5000 and can be accessed through it.

```
PS C:\Users\shobi\OneDrive\Desktop\Ship Classification> & 'C:\Users\shobi\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\shobi\.vscode\extensions\ms-python.python-2023.20.0\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '50892' '--' 'c:\Users\shobi\OneDrive\Desktop\Ship Classification\Flask\app.py'
2023-11-04 23:25:06.494958: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
2023-11-04 23:25:13.699354: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
* Debugger is active!
* Debugger PIN: 894-934-489
```

### Step 2: Open the browser and navigate to localhost:5000 to check your application

The home page looks like this.



Click on choose the image to upload and select an image to be classified.

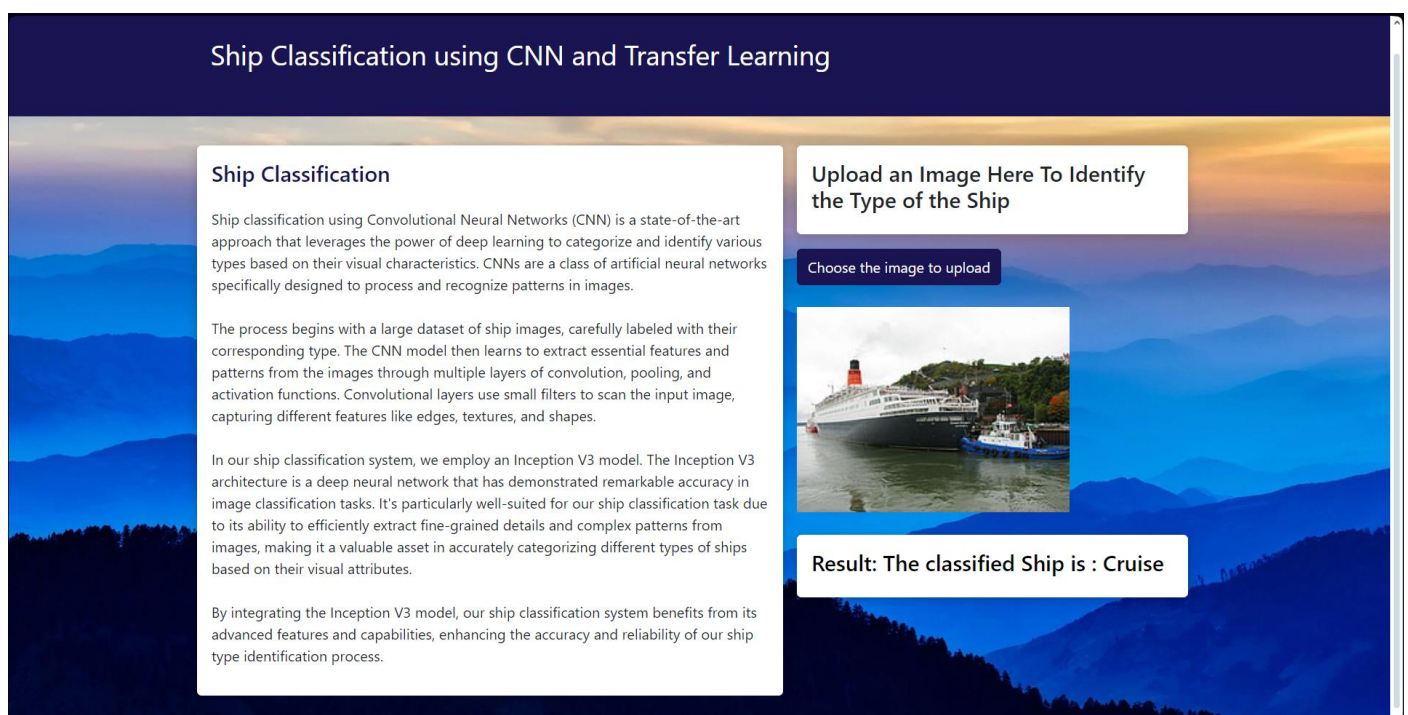


## Input-1 (Cruise)



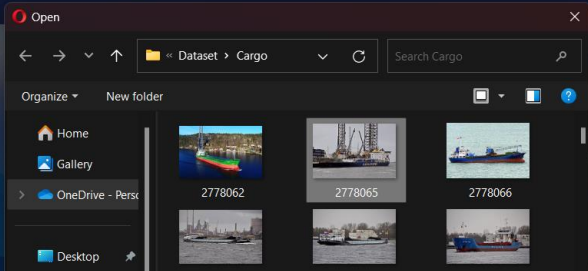
Now click on the predict button to predict the type of the ship using our Inception V3.

## Output



## Input-2 (Cargo)

### Ship Classification using CNN and Transfer Learning



Open

Dataset > Cargo

Search Cargo

Organize New folder

Home

Gallery

OneDrive - Pers

Desktop

2778062

2778065

2778066

Upload an Image Here To Identify the Type of the Ship

Choose the image to upload

Ship Classification

Ship classification using Convolutional Neural Networks (CNN) is a state-of-the-art approach that leverages the power of deep learning to categorize and identify various types based on their visual characteristics. CNNs are a class of artificial neural networks specifically designed to process and recognize patterns in images.

The process begins with a large dataset of ship images, carefully labeled with their corresponding type. The CNN model then learns to extract essential features and patterns from the images through multiple layers of convolution, pooling, and activation functions. Convolutional layers use small filters to scan the input image, capturing different features like edges, textures, and shapes.

In our ship classification system, we employ an Inception V3 model. The Inception V3 architecture is a deep neural network that has demonstrated remarkable accuracy in image classification tasks. It's particularly well-suited for our ship classification task due to its ability to efficiently extract fine-grained details and complex patterns from images, making it a valuable asset in accurately categorizing different types of ships based on their visual attributes.

By integrating the Inception V3 model, our ship classification system benefits from its advanced features and capabilities, enhancing the accuracy and reliability of our ship type identification process.

Upload an Image Here To Identify the Type of the Ship

Choose the image to upload

Predict!

Now click the predict button to predict the type of the ship using our Inception V3 model.

## Output-2

### Ship Classification using CNN and Transfer Learning

Ship Classification

Ship classification using Convolutional Neural Networks (CNN) is a state-of-the-art approach that leverages the power of deep learning to categorize and identify various types based on their visual characteristics. CNNs are a class of artificial neural networks specifically designed to process and recognize patterns in images.

The process begins with a large dataset of ship images, carefully labeled with their corresponding type. The CNN model then learns to extract essential features and patterns from the images through multiple layers of convolution, pooling, and activation functions. Convolutional layers use small filters to scan the input image, capturing different features like edges, textures, and shapes.

In our ship classification system, we employ an Inception V3 model. The Inception V3 architecture is a deep neural network that has demonstrated remarkable accuracy in image classification tasks. It's particularly well-suited for our ship classification task due to its ability to efficiently extract fine-grained details and complex patterns from images, making it a valuable asset in accurately categorizing different types of ships based on their visual attributes.

By integrating the Inception V3 model, our ship classification system benefits from its advanced features and capabilities, enhancing the accuracy and reliability of our ship type identification process.

Upload an Image Here To Identify the Type of the Ship

Choose the image to upload

Result: The classified Ship is : Cargo