End-to-end deep learning project for classifying ship types.

The objective of this project is to develop an end-to-end deep learning solution for classifying ship images into three categories: cargo, tanker, and passenger vessels. The proposed solution involves the use of convolutional neural networks (CNNs) to extract relevant features from the input ship images and classify them into one of the three categories.

Predicting ship types as soon as possible is crucial because it can have significant impacts on maritime operations and safety. Early detection allows maritime authorities to take prompt action to ensure safe navigation, manage port activities, and make informed decisions. Here are some reasons why predicting ship types early is essential:

1. Enhance maritime safety: Accurate ship classification helps ensure the safety of vessels at sea and in ports. By identifying the type of ship early, maritime authorities can implement appropriate safety measures and navigation protocols.

2. Optimize port operations: Different types of ships require different handling procedures at ports. Early classification of incoming vessels allows port operators to allocate resources efficiently and streamline cargo handling.

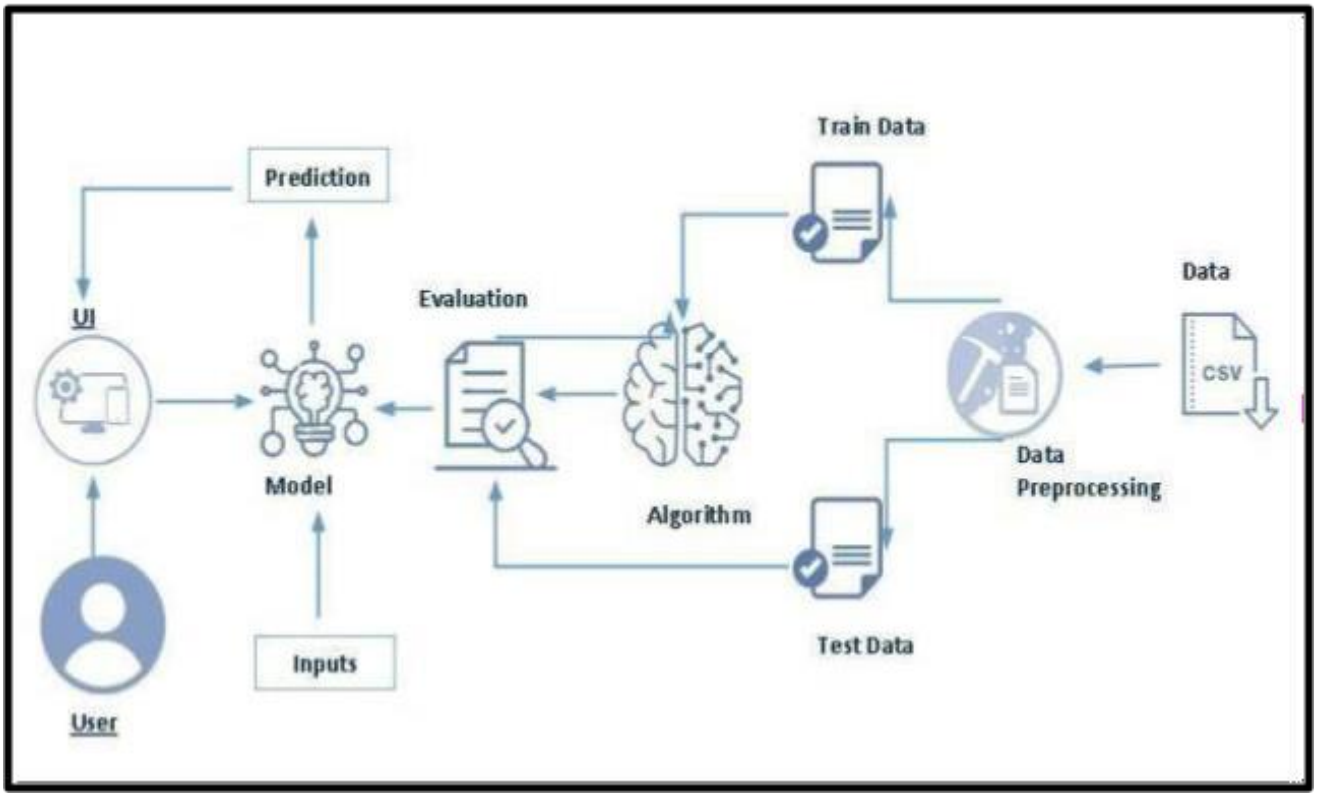
3. Environmental protection: Cargo and tanker ships may carry hazardous materials or pollutants. Early classification of these ships helps in implementing appropriate environmental safeguards and response plans to protect the marine ecosystem.

4. Trade and commerce: Identifying the type of ship can have economic implications. It helps in monitoring trade activities and ensuring that import and export operations are carried out efficiently.

By accurately classifying ships early, this project aims to improve maritime safety, environmental protection, and the efficiency of port and trade operations.

Let us look at the Technical Architecture of the project.

Technical Architecture:



Project Flow:

- Preparing the data
  - o Downloading the dataset
  - o Categorize the images

- Data pre-processing
  - o Import ImageDataGenerator Library and Configure it
  - o Apply ImageDataGenerator functionality to Train and Test set

- Building the model
  - o Create function for creation and compilation of model
  - o Create the model using our function
  - o Look at the model summary
  - o Create a checkpoint for the model

- Training and testing the model
  - o Train the model while monitoring validation loss
  - o Test the model with custom inputs

- Building Flask application
  - o Build a flask application
  - o Build the HTML page and execute

Project Objectives:

By the end of this project you'll understand:

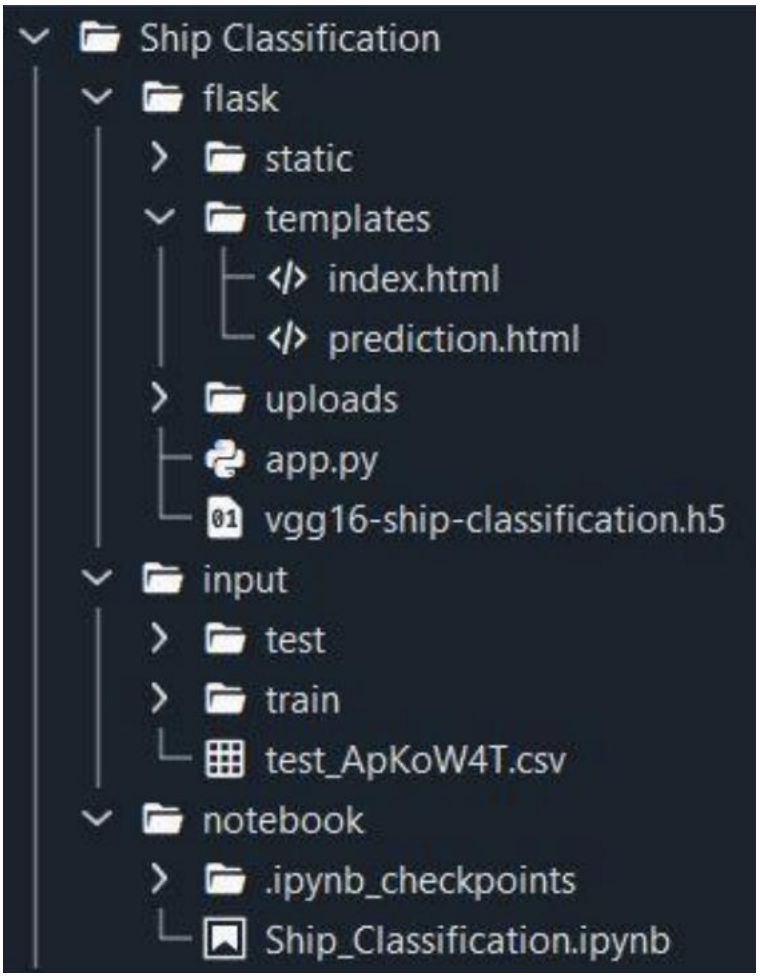
- Preprocess the images.
- Training VGG16 model with custom data.
- How pre-trained models will be useful in object classification.
- How to evaluate the model.
- Building a web application using the Flask framework.

Project Structure:

The input folder contains two folders for train and test images, each of them having

images of different categories of ships, arranged folder-wise.

- Flask folder has all the files necessary to build the flask application.
  - ➔ static folder has the images, style sheets and scripts that are needed in building the web page.
  - ➔ templates folder has the HTML pages.
  - ➔ uploads folder has the uploads made by the user.
  - ➔ app.py is the python script for server side computing.
  - ➔ .h5 file is the model file which is saved after model building.
- Ship_Classification.ipynb is the notebook on which code is executed.



Milestone 1: Preparing the data

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 IDs of training images and sort them according to their category. The train.csv file looks like below:

```
train_files.head()
```

	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'}
```

```
train_files['ship'] = train_files['category'].map(ship).astype('category')
```

```
train_files.head()
```

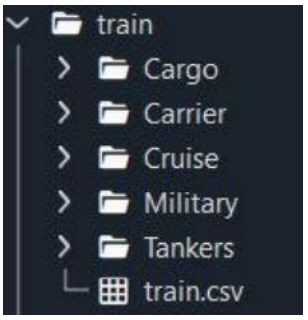
	image	category	ship
0	2823080.jpg	1	Cargo
1	2870024.jpg	1	Cargo
2	2662125.jpg	2	Military
3	2900420.jpg	3	Carrier
4	2804883.jpg	2	Military

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 organise our train images into folders representing their corresponding categories.

```
labels = train_files.sort_values('ship')
class_names = list(labels.ship.unique())
for i in class_names:
    os.makedirs(os.path.join('/content/drive/MyDrive/SmartBridge/Ship Classification/input/train',i))
```

```
import shutil
for c in class_names: # Category Name
  for i in list(labels[labels['ship']==c]['image']): # Image Id
    get_image = os.path.join('/content/drive/MyDrive/SmartBridge/Ship Classification/input/images/', i) # Path to Images
    move_image_to_cat = shutil.move(get_image, '/content/drive/MyDrive/SmartBridge/Ship Classification/input/train/'+c)
```

The above code will create sub-directories within the train folder and move the images into them. The final folder structure of train will look like:



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. A function known as the preprocess_input from VGG16 library will perform the necessary preprocessing operations on the images in order to make them suitable for getting trained.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input

train_datagen = ImageDataGenerator(rotation_range=45,
                                   horizontal_flip=True,
                                   width_shift_range=0.5,
                                   height_shift_range=0.5,
                                   validation_split=0.2,
                                   preprocessing_function=preprocess_input
                                   )

test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
```

Activity 2: Apply ImageDataGenerator functionality to Train and Test set

Specify the path of both the folders in flow_from_directory method. We are importing the images in 224*224 pixel

```
train_set = train_datagen.flow_from_directory( '/content/drive/MyDrive/SmartBridge/Ship Classification/input/train/',
                                batch_size=16, subset='training',
                                target_size=(224,224))

validation_set = train_datagen.flow_from_directory('/content/drive/MyDrive/SmartBridge/Ship Classification/input/train/',
                                                   batch_size=16, subset='validation',
                                                   target_size=(224,224)
                                                  )


test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/SmartBridge/Ship Classification/input/test',batch_size=16,
                                            target_size=(224,224))
```

```
Found 5003 images belonging to 5 classes.
Found 1249 images belonging to 5 classes.
Found 30 images belonging to 5 classes.
```

Milestone 3: Building the model

We will be creating a function that uses the pre-trained VGG16 model for predicting custom classes.

Activity 1: Create function for creation and compilation of model

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 keras.layers import Dense,Flatten, Dropout
from keras.models import Model

def create_model(input_shape, n_classes, optimizer='rmsprop'):
    conv_base = VGG16(include_top=False,
                      weights='imagenet',
                      input_shape=input_shape)

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

Next, add dense and dropout layers to the top model. Finally, add an output layer with the number of neurons equal to the number of classes we are predicting.

```
top_model = conv_base.output
top_model = Flatten(name="flatten")(top_model)
top_model = Dense(4096, activation='relu')(top_model)
top_model = Dense(1072, activation='relu')(top_model)
top_model = Dropout(0.2)(top_model)
output_layer = Dense(n_classes, activation='softmax')(top_model)

model = Model(inputs=conv_base.input, outputs=output_layer)
```

Finally, compile the model and return it.

```
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

return model
```

Activity 2: Create the model using our function

Use the function we have just defined to create the model for ship classification.

```
from tensorflow.keras.optimizers import Adam

input_shape = (224, 224, 3)
optim = Adam(learning_rate=0.001)
n_classes=5

vgg_model = create_model(input_shape, n_classes, optim)
```

Activity 3: : Look at the model summary

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

```
vgg_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
 flatten (Flatten)           (None, 25088)             0
 dense (Dense)               (None, 4096)              102764544
 dense_1 (Dense)             (None, 1072)              4391984
 dropout (Dropout)           (None, 1072)              0
 dense_2 (Dense)             (None, 5)                 5365
=================================================================
Total params: 121,876,581
Trainable params: 107,161,893
Non-trainable params: 14,714,688
_________________________________________________________________
```

Activity 4: Create a checkpoint for the model

We need to create a checkpoint to track the validation loss of the model in order to save the best weights.

```
from keras.callbacks import ModelCheckpoint
cp = ModelCheckpoint('best.hdf5', monitor='val_loss',verbose=1, save_best_only=True)
```

Milestone 4: Training and testing the model

Activity 1: Train the model while monitoring validation loss

We will be training the model for 25 epochs using the fit_generator() function:

```
epochs = 25
history = vgg_model.fit_generator(generator=train_set,
                                  steps_per_epoch=train_set.n//train_set.batch_size,
                                  validation_steps = validation_set.n//validation_set.batch_size,
                                  validation_data=validation_set,
                                  callbacks=[cp],
                                  epochs=epochs)
```

Save the model after training.

```
vgg_model.save('vgg16-ship-classification.h5')
```

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

img= image.load_img('/content/drive/MyDrive/SmartBridge/Ship Classification/input/test/Cargo/cargo1.jpg',target_size=(224,224))

img = image.img_to_array(img)
img = img.reshape((1, img.shape[0], img.shape[1], img.shape[2]))
img = preprocess_input(img)

pred = model.predict(img)

pred=pred.flatten()
pred = list(pred)
m = max(pred)

val_dict = {0:'Cargo', 1:'Carrier', 2:'Cruise', 3:'Military', 4:'Tankers'}

result=val_dict[pred.index(m)]
print(result)
```

```
Cargo
```

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

```
import numpy as np
import os
from flask import Flask, app,request,render_template
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
```

Step 2: Initialize the flask app and load the model

Instance of Flask is created and the model is loaded using load_model from keras.

```
app=Flask(__name__)

model=load_model(r"vgg16-ship-classification.h5")
```

Step 3: Configure html pages

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/prediction.html')
def prediction():
    return render_template('prediction.html')

@app.route('/index.html')
def home():
    return render_template("index.html")
```

Step 4: Pre-process the frame and run

Pre-process the captured frame and give it to the model for prediction. Based on the prediction the output text is generated and sent to the HTML to display.

```
@app.route('/result',methods=["GET","POST"])
def res():
    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=(224,224))

        img = image.img_to_array(img)
        img = img.reshape((1, img.shape[0], img.shape[1], img.shape[2]))
        img = preprocess_input(img)
        # reshape data for the model

        pred = model.predict(img)
        pred=pred.flatten()
        pred = list(pred)
        m = max(pred)

        val_dict = {0:'Cargo', 1:'Carrier', 2:'Cruise', 3:'Military', 4:'Tankers'}
        #print(val_dict[pred.index(m)])


        result=val_dict[pred.index(m)]
        #print(result)
        return render_template('prediction.html',prediction=result)
```

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.

```
if __name__ == "__main__":
    app.run()
```

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.

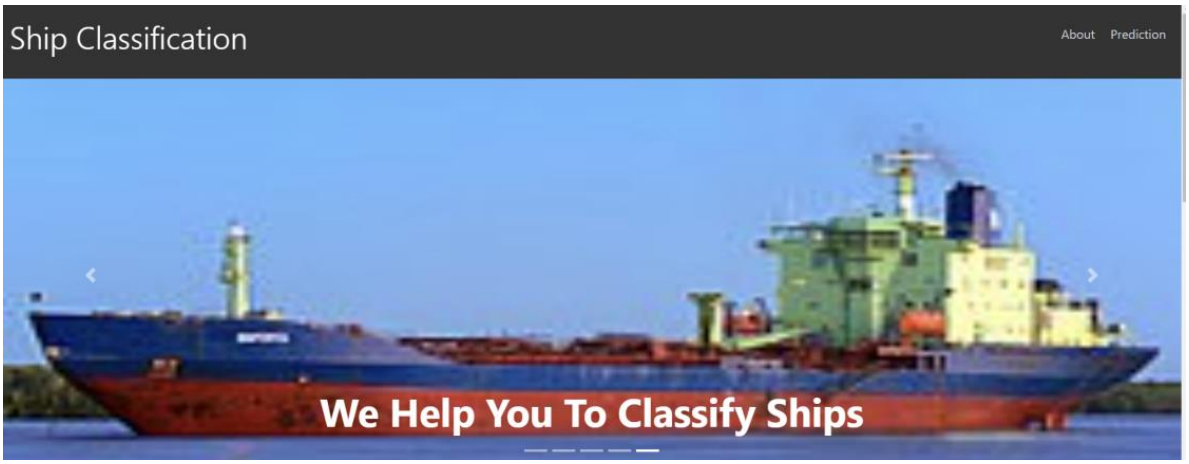
Step 1: Run the application

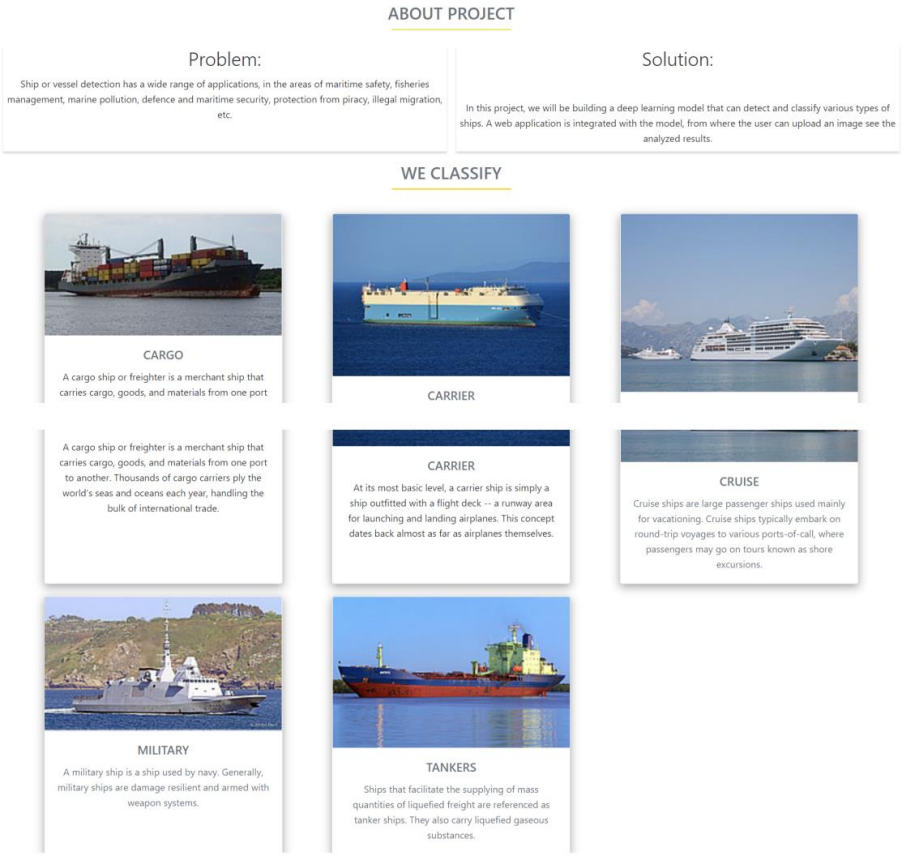
In the anaconda prompt, navigate to the folder in which the flask app is present. When the python file is executed the localhost is activated on port 5000 and can be accessed through it.

```
Serving Flask app 'app' (lazy loading)
Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
Debug mode: off
Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

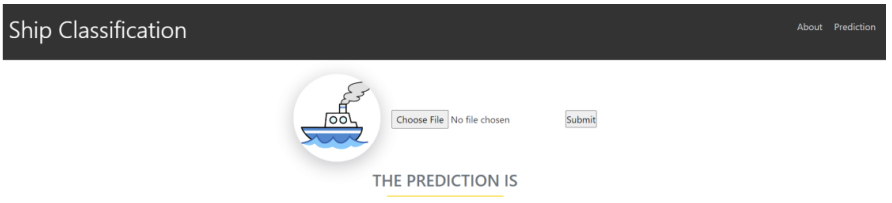
Step 2: Open the browser and navigate to localhost:5000 to check your application

The home page looks like this. You can click on login or register.

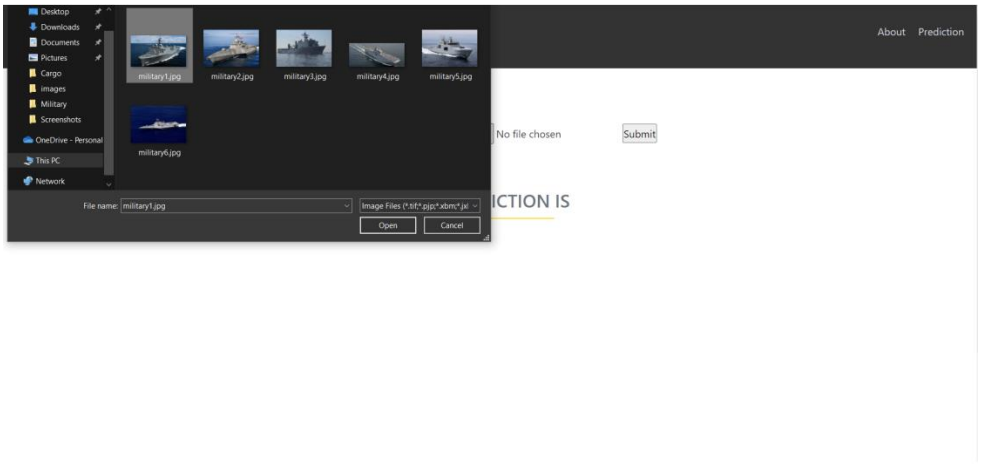




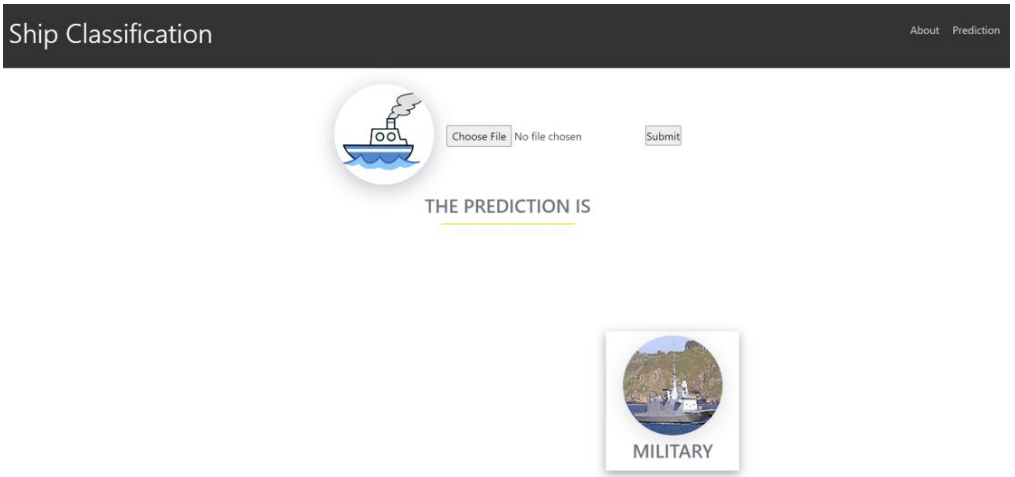
After clicking on the predict button you will be redirected to the prediction page where you can browse the images.



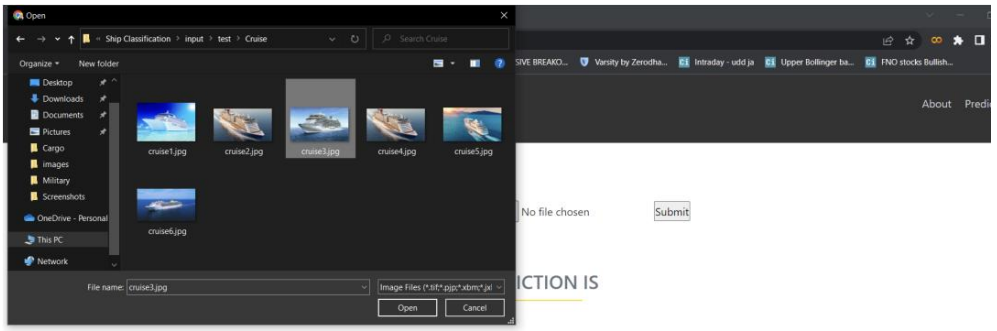
Input 1



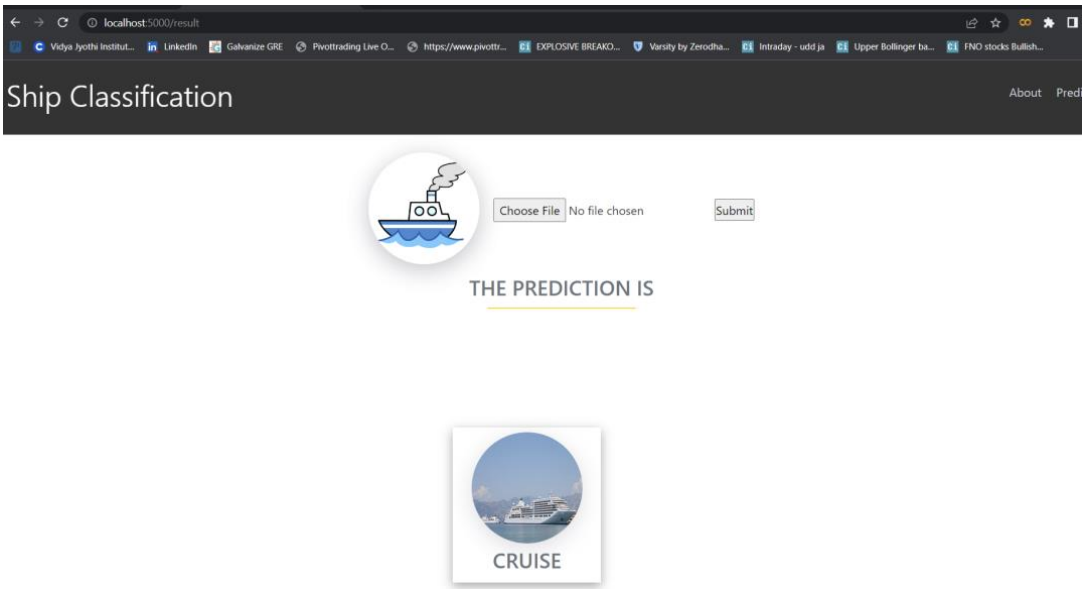
Output 1



Input 2



Output 2


localhost:5000/result
Ship Classification
About
Choose File
No file chosen
Submit
THE PREDICTION IS
CRUISE