ENPM673 – Perception of Autonomous Robotics

Extra Assignment

Link to Github repo: https://github.com/nsaini2896/Classification-of-boats.git

Link to Google Drive: https://drive.google.com/drive/folders/1OV0CMCDdvrb6bJ43KeTatYNVIFQ3safk?usp=sharing

Introduction to Neural Network

Image recognition is the ability for computers to look at a photograph and understand what's in the photograph. A neural network is made up of separate nodes called neurons. These neurons are arranged into a series of groups called layers. Nodes in each layer are connected to the nodes in the following layer. Data flows from the input to the output along these connections. Each individual node is trained to perform a simple mathematical calculation and then feed its result to all the nodes it's connected to. The neural network takes in a set of input values in the input layer. Then those values pass through all the following layers. Each node tweaks the value it receives slightly and passes its result onto the next node.

Step 1: Coding a neural network with Keras

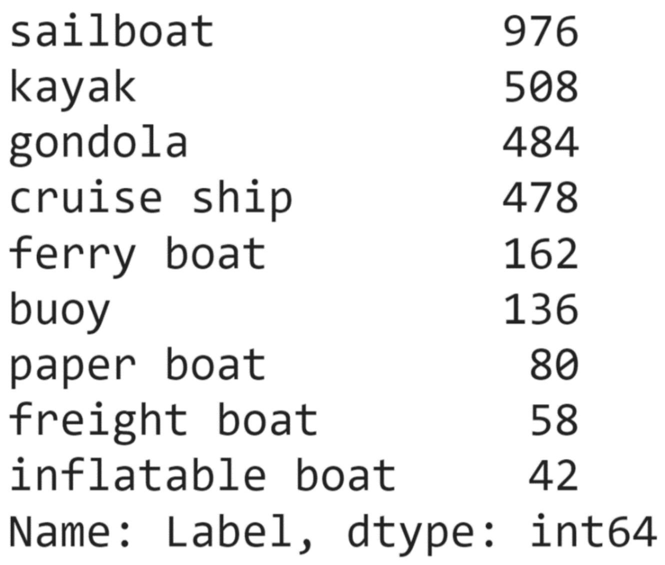
We use a software framework called Keras to code our neural networks. Keras is a high-level library for building neural networks in Python with only a few lines of code. First we create a new neural network model with Keras. It's called a sequential model because we're defining each layer in order sequentially, or one layer at a time. Next, we'll add a layer with three nodes. Next, we add the second layer with three nodes. And then finally, we add the final layer with one node to act as the output layer. The Keras also lets us customize how each layer works.

Step 2: Learning the use of proper Activation Functions

The most important things to configure are activation functions. Before values flow from the nodes in one layer to the next, they pass through an activation function. Activation functions decide which inputs from the previous layer are important enough to feed to the next layer. Keras lets us choose which activation function is used for each layer by passing in the name of the activation function that we want to use.

Step 3: Recognizing image contents with a neural network

Before we can use the neural network to make classifications we need to train it. We need to collect the dataset of training images. We need lots of images that belong to the class that we want to recognize. We also need lots of images that represent the other possible kinds of images. This way the neural network can learn to tell the two different types or classes of images apart. After we have training data we can train the neural network by showing it images and telling it what the correct answer will be for each one.



```
sailboat           976
kayak              508
gondola            484
cruise ship        478
ferry boat         162
buoy               136
paper boat          80
freight boat        58
inflatable boat     42
Name: Label, dtype: int64
```

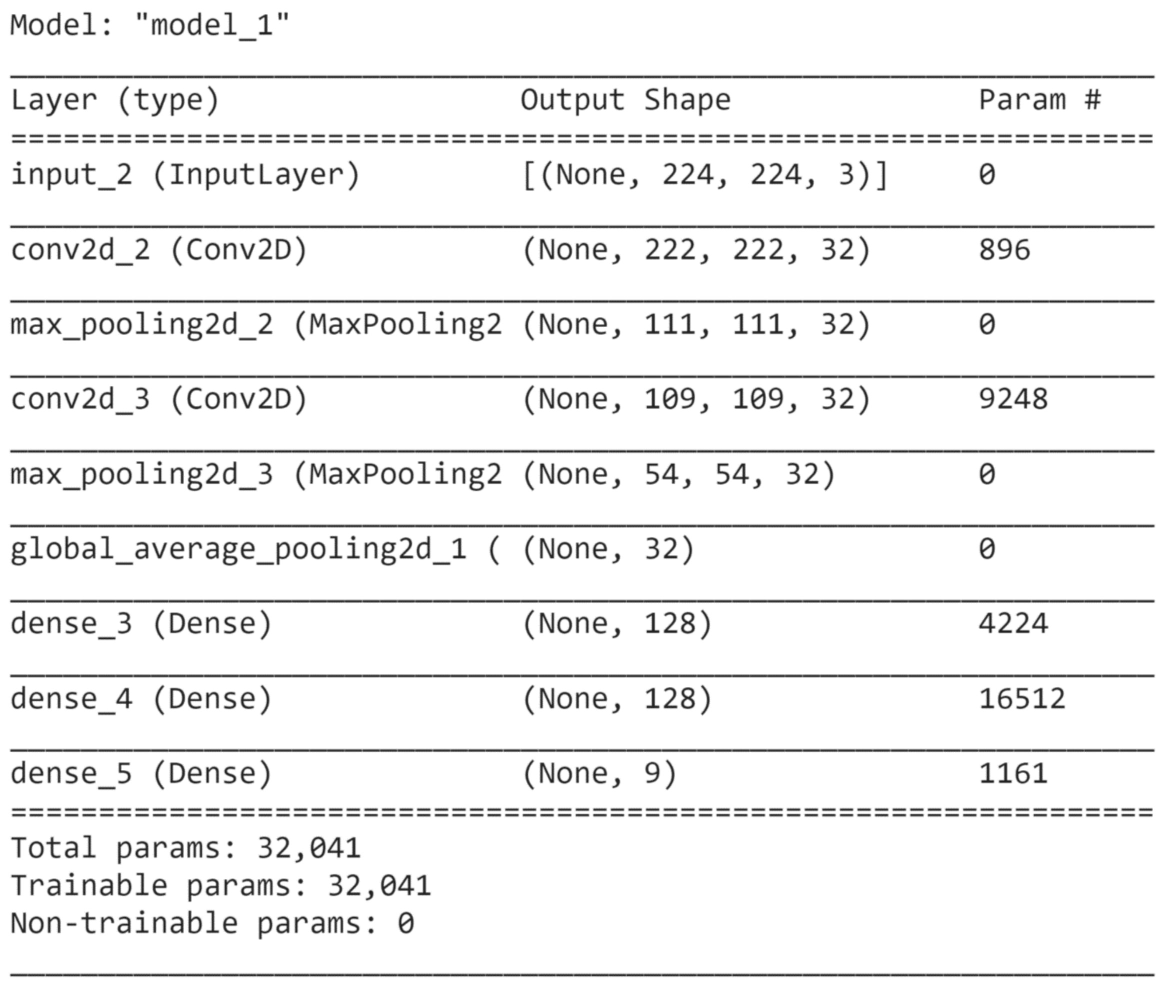
Figure 1 Number of boats corresponding to each label

Step 4: Adding convolution for translational invariance

If we only train the neural network with pictures of numbers that are perfectly centered, the neural network will get confused if it sees anything else. We need to improve our neural network so that it can recognize objects in any position. This is called translation invariance. The solution is to add a new type of layer to our neural network called the convolutional layer. Unlike a normal dense layer, where every node is connected to every other node, this layer breaks apart the image in a special way so that it can recognize the same object in different positions.

Step 5: Designing a neural network architecture for image recognition

The convolutional layers are looking for patterns in our image and recording whether or not they found those patterns in each part of our image, but we don't usually need to know exactly where in an image a pattern was found down to the specific pixel. It's good enough to know the rough location of where it was found. Take grid as the output of a convolutional filter that ran over a small part of our image. It's trying to detect a particular pattern and these numbers represent whether or not that pattern was found in the corresponding part of the image. A zero in the grid means that the pattern wasn't found at all and a one means the area was a strong match for the pattern. We could pass this information directly to the next layer in our neural network, but if we can reduce the amount of information that we pass to the next layer, it will make the neural network's job easier. The idea of max pooling is to down sample the data by only passing on the most important bits. And then finally, we'll create a new array that only saves the numbers that we selected. The idea is that we're still capturing roughly where each pattern was found in our image, but we're doing it with 1/4 as much data. We'll get nearly the same end result, but they'll be a lot less work for the computer to do in the following layer of the neural network. The convolutional layers add translational invariance, the max pooling layers down sample the data, and dropout forces the neural network to learn in a more robust way. And then finally, the dense layer maps the output of the previous layers to the output layer so we can predict which class the image belongs to.



```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 109, 109, 32)      9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               4224
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 9)                 1161
=================================================================
Total params: 32,041
Trainable params: 32,041
Non-trainable params: 0
_________________________________________________________________
```

Figure 2 Summary of the Neural Network Architecture

Why do we implement VGG16 architecture?

1. For accuracy of the training and classification models, the depth of the model is necessary. The more layers used to train the model, the better is the accuracy of the model. The data on internet says that VGG 16 makes the improvement over older models such as AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another as shown in Figure 3.

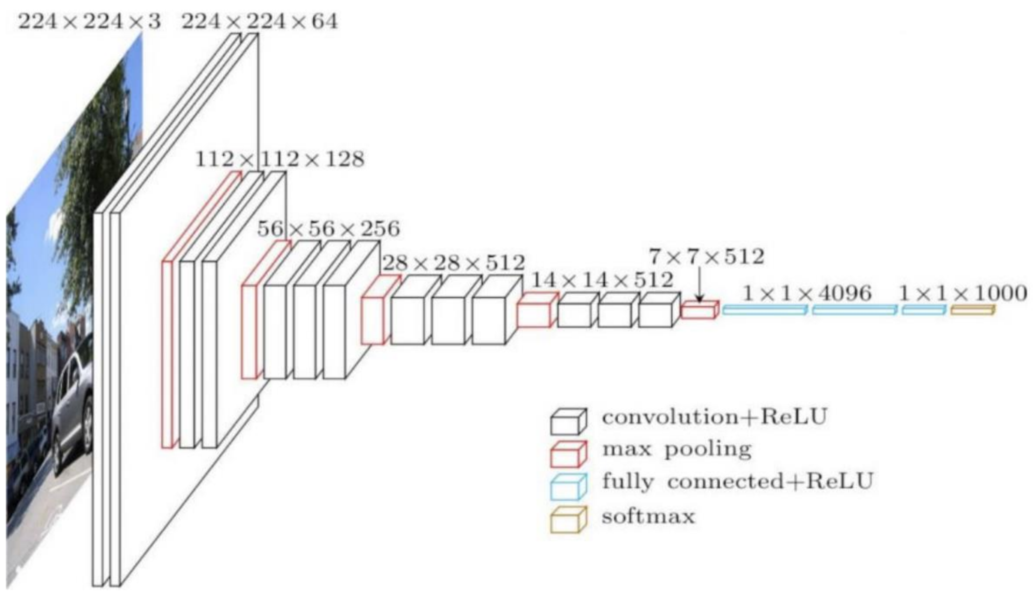


Figure 3 VGG 16 Architecture and layers representation

2. The VGG 16 model has been trained with millions of image datasets for weeks to provide a better performance. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

Results:

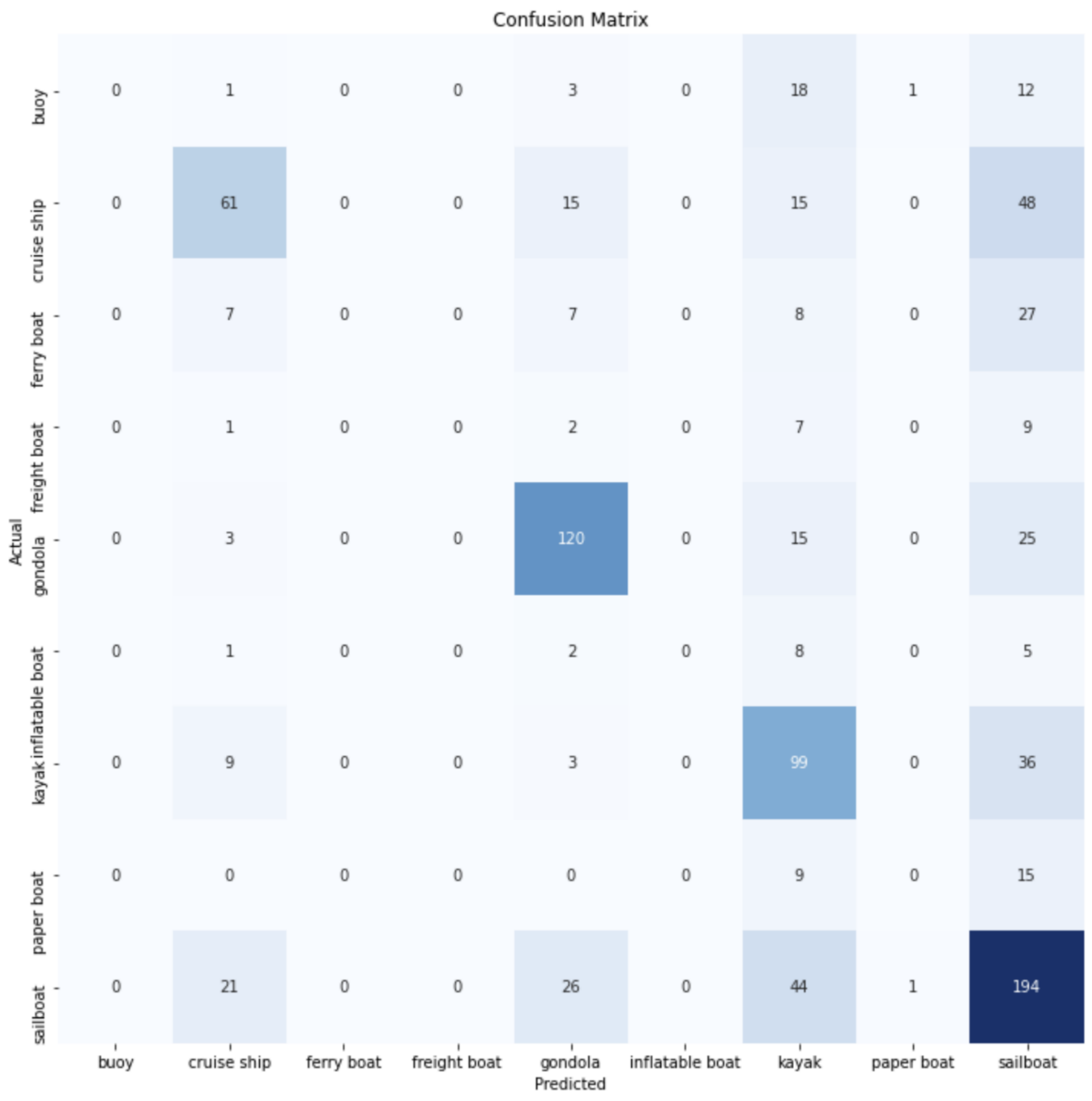
Test Loss: 1.42438

Test Accuracy: 53.99%

```
Classification Report:
----------------------
                 precision    recall  f1-score   support

           buoy       0.00      0.00      0.00        35
    cruise ship       0.59      0.44      0.50       139
     ferry boat       0.00      0.00      0.00        49
   freight boat       0.00      0.00      0.00        19
        gondola       0.67      0.74      0.70       163
inflatable boat       0.00      0.00      0.00        16
          kayak       0.44      0.67      0.54       147
     paper boat       0.00      0.00      0.00        24
       sailboat       0.52      0.68      0.59       286

       accuracy                           0.54       878
      macro avg       0.25      0.28      0.26       878
   weighted avg       0.46      0.54      0.49       878
```



Actual \ Predicted	buoy	cruise ship	ferry boat	freight boat	gondola	inflatable boat	kayak	paper boat	sailboat
buoy	0	1	0	0	3	0	18	1	12
cruise ship	0	61	0	0	15	0	15	0	48
ferry boat	0	7	0	0	7	0	8	0	27
freight boat	0	1	0	0	2	0	7	0	9
gondola	0	3	0	0	120	0	15	0	25
inflatable boat	0	1	0	0	2	0	8	0	5
kayak	0	9	0	0	3	0	99	0	36
paper boat	0	0	0	0	0	0	9	0	15
sailboat	0	21	0	0	26	0	44	1	194

Figure 4 The confusion matrix representing the actual and predicted images.

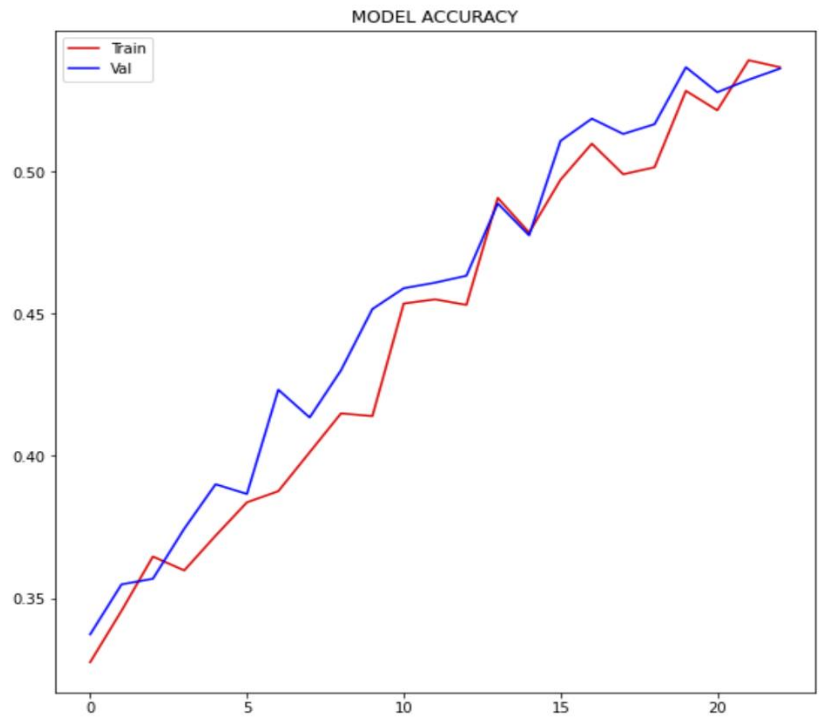


Figure 5 The graph representing accuracy of trained model

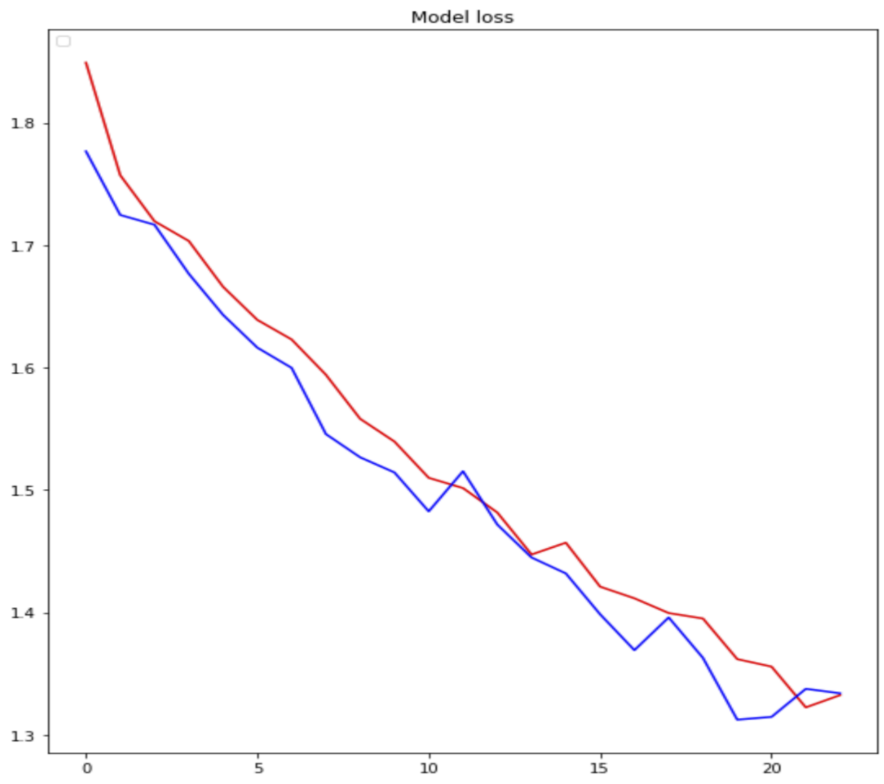
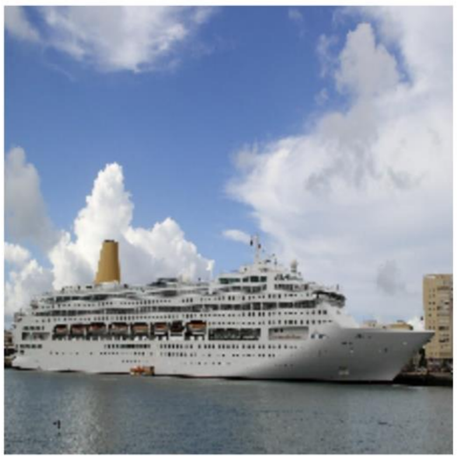


Figure 6 Graph representing Loss of the trained model

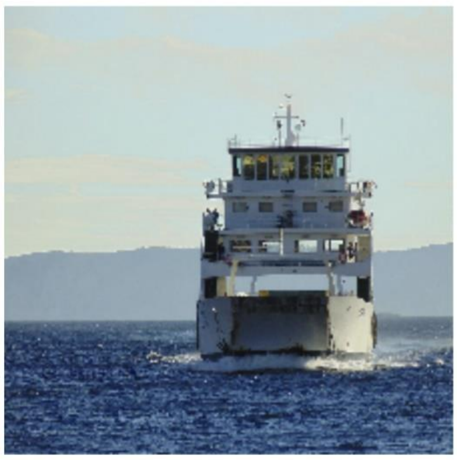
Sample Prediction Results:

1. Cruise Ship



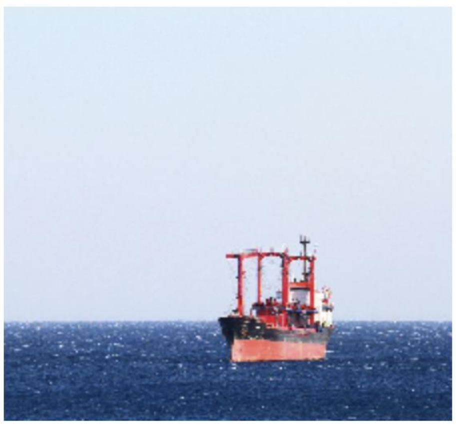
Predicted as: Sailboat

2. Ferry Boat



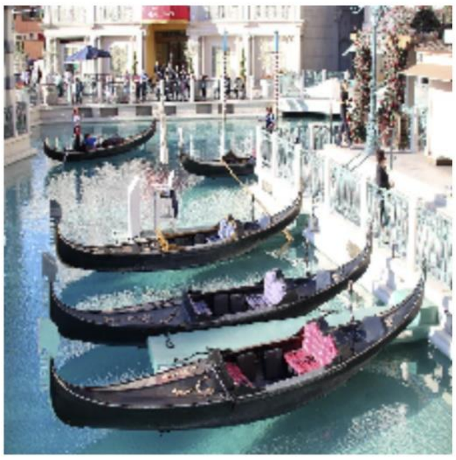
Predicted as: Sailboat

3. Freight Boat



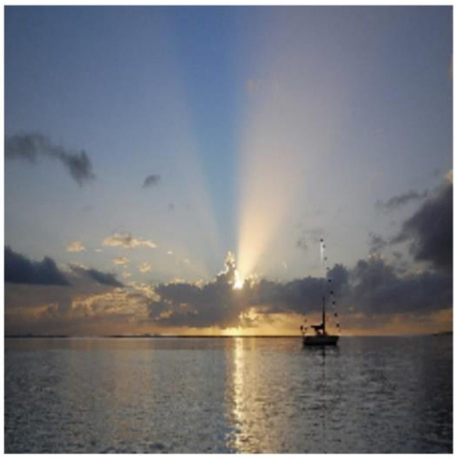
Predicted as: Sailboat

4. Gondola



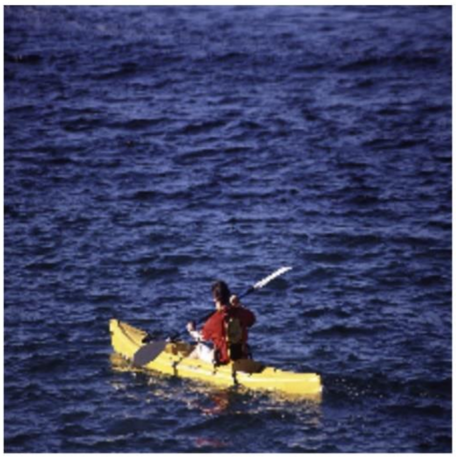
Predicted as: Gondola

5. Sailboat



Predicted as: Kayak

6. Kayak



Predicted as: Cruise Ship

Problems Encountered during the testing:

1. When I was trying to implement the trained model using the .json file directly for predicting the single image inputs, I faced the problem where the trained model was not being implemented on the individual images. So to rectify I cooperated the prediction code at the end of the training code and gave individual inputs.
2. I could not achieve a good accuracy even with high number of epochs and low number of epochs. The reason to this is when we use too many epochs we can face overfitting of the training dataset and too few lead to an underfitted model. So, we use method of early stopping using which we specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a held out validation dataset. In my code the model stopped training at 23 epochs whereas I gave an epoch value of 100.