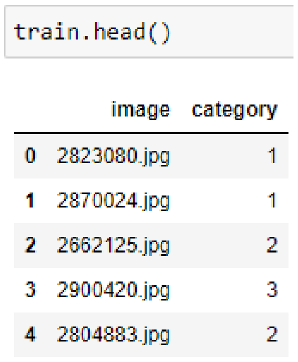
Approach file

• Data Description:

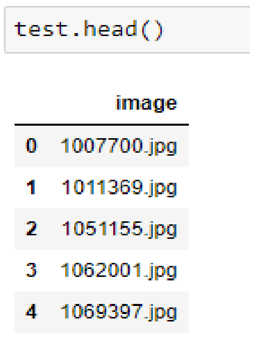
There are **6252 images in train and 2680 images in test** data. The categories of ships and their corresponding codes in the dataset are as follows -

```
{'Cargo': 1,
'Military': 2,
'Carrier': 3,
'Cruise': 4,
'Tankers': 5}
```



```
train.head()
```

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

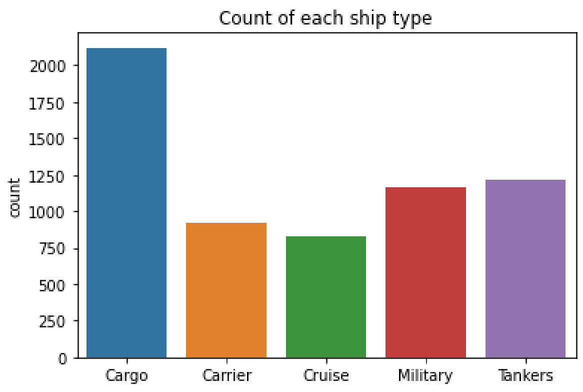


```
test.head()
```

	image
0	1007700.jpg
1	1011369.jpg
2	1051155.jpg
3	1062001.jpg
4	1069397.jpg

• EDA and Data preprocessing:

1. There are no null values in the training set.
2. The distribution of data in the various category of ships are as follows:



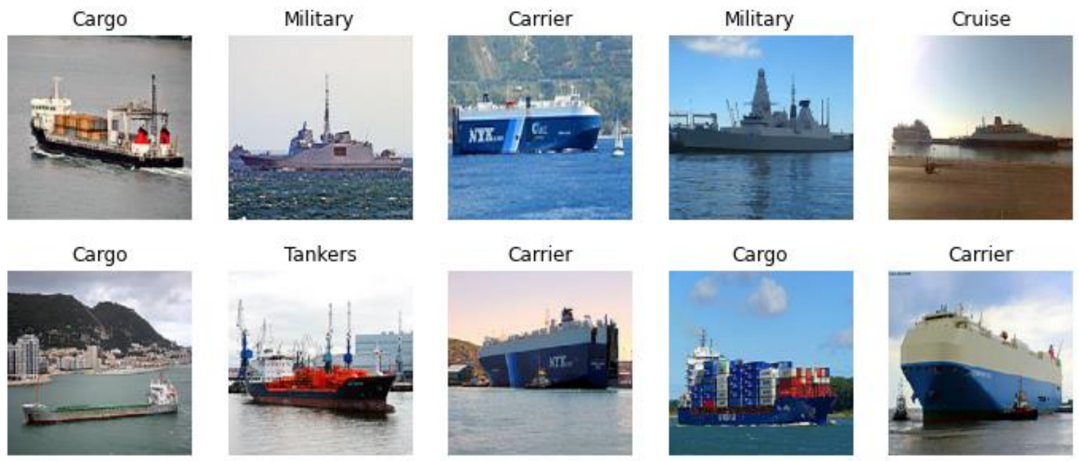
```
Cargo       0.339091
Tankers     0.194658
Military    0.186660
Carrier     0.146513
Cruise      0.133077
```

So, we can see Cargo type ships are in much more quantity in the train dataset. So, when splitting the data into training and validation set, we use stratify according to the categories. This parameter makes a split so that the proportion of values in both the sets will be the same as the proportion of values provided to parameter stratify (category column).

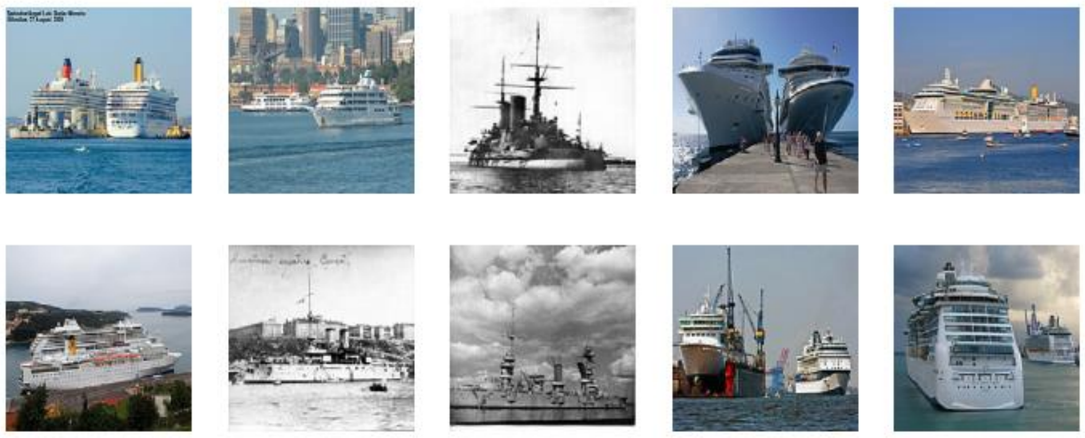
3. While loading the image, we reshape each image to a particular shape, because they have to be of same size while feeding to model. Here we choose the size (224,224,3). As , they are colored images, the number of channels are kept 3.

4. The images are changed into 2D array, with the pixel values normalized. Because the computation of high numeric values is more complex. The pixel values are made to have value between 0-1. The pixel values of RGB image lies from 0 to 255. So, we divide each pixel value with 255.

5. Some images from the train and test set are visualized.



Training set



Test Set

So, in the test set, there are greyscale images with colour images. Also, some images contain a large portion of the background. So, to address these issues, we will need to perform data augmentation to create a more robust training set. Data augmentation create modified versions of training set images, by cropping a part of the picture, changing the intensity of RGB channels, flipping them etc. This helps creating a more generalized training set, so that accuracy on the unseen test data increases. This will also prevent overfitting.

6. The category column contains values from 1 to 5. As these are labels, we hot encode the column into five different columns. As, we will use the data for train test split, we convert it into numpy array.

• Splitting the data into train and validation set:

The data is split in 80% training and 20% validation set. We use stratify as mentioned earlier.

```
X_train, X_val, y_train, y_val = train_test_split(train_imgs, y, stratify=y, test_size=0.20, random_state=1)
```

- ## Data Augmentation:

The training data is augmented and a generator is made for feeding into model.

```
BS = 8
gen = ImageDataGenerator(rotation_range=45,
                         horizontal_flip=True,
                         width_shift_range=0.5,
                         height_shift_range=0.5)

train_generator = gen.flow(X_train, y_train,batch_size=BS)
```

- ## Transfer learning:

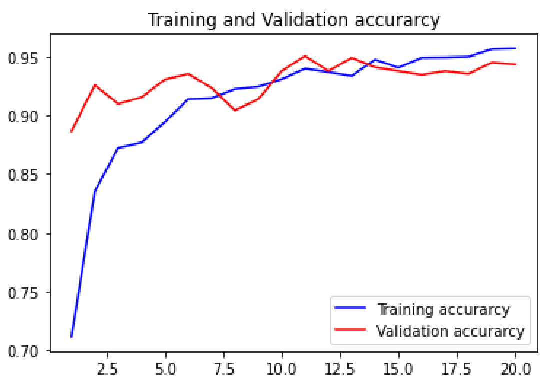
CNN model contains **convolutional** layers followed by **fully connected** layers. Convolution layers extract features from the image and fully connected layers classify the image using extracted features. When we train a **CNN** on image data, It is seen that top layers of the network learn to extract **general** features from images such as edges, distribution of colours, etc. As we keep going deep in the network, the layers tend to extract more **specific** features. Now we can use these pretrained models which already know how to extract features and avoid the training from scratch.
So, similarly here a pretrained model is used. The Xception is a convolutional neural network with 71 layers. This is already pretrained on millions of images. So, we keep the pretrained weights in the model. We use a sequential model with base as Xception. Then we attach our own classifier, so we disable the default classifier in the Xception by setting include_top=False. After the convolutional layer, a average pooling layer is used. The layer also turns the output 2D, as the Dense layer only accepts 2D. Then we use softmax at Dense layer, since it is a non-binary classification. The model architecture looks like,

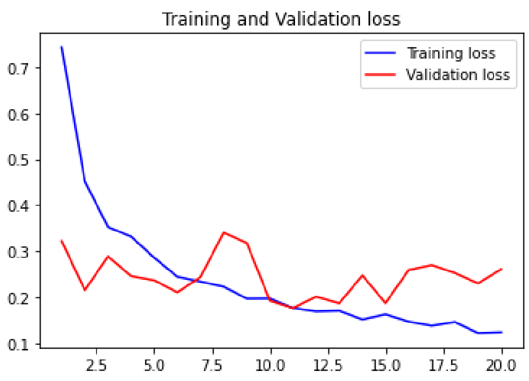
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
xception (Functional)        (None, None, None, 2048)  20861480  
_________________________________________________________________
global_average_pooling2d_2 ( (None, 2048)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 10245     
=================================================================
Total params: 20,871,725
Trainable params: 20,817,197
Non-trainable params: 54,528
_________________________________________________________________
```

Since the model is pretrained, we don't need to train for large epochs. So, here 20 epochs are run with initial learning rate as 0.0001, which keeps decreasing at each epoch.

- ## Model Evaluation:

The training and validation loss and accuracy are as follows:





- Training loss becomes almost flat at the end of the epochs. So, the model is not significantly underfitting.
- There is no significant gap between training and validation accuracy. But, validation loss increases after a certain point. The training could have been stopped at that point. Because, at the end of 20 epochs, the model overfits.
- Training loss curve is not noisy. So, the learning rate is suitable.
- The training F1 score is 0.9942 and validation F1 score is 0.9436. Training F1 score is significantly higher than validation. This also indicates over-fitting.

Conclusion:

- Early stopping could have been used to prevent the overfitting. Or we could use the callbacks to save the best model.
- As the model is pretrained, we could have frozen some layers , because it is taking a really long time to train the whole model .
- 2 or 3 other pre-trained models could have been used, and the prediction could have been done on voting.