

# Final

## Chương trình làm gì :

Chương trình demo sẽ dùng mạng Deep Neutral Network CNN để xác định đầu ra của bức ảnh input được cho vào là ảnh có chứa tàu hay không chứa tàu. Hay gọi là bài toán Xác định tàu.

## Chương trình sẽ giải quyết được vấn đề gì :

**Trong bài toán Embedding trong Deep Neutral Network chương trình sẽ thể hiện được :**

- Cách hoạt động của Neutral network
- Lý do cần phải embedding trong Deep Neutral Network
- Cách tiếp cận Black box trong thực tế để kiểm tra các mô hình khác.

## Các chức năng của bài demo :

- Bài demo mô tả rõ ràng các dữ liệu được học (input)
- Cân bằng lại dữ liệu được học để dữ liệu có thiếu số sẽ cân bằng lại với dữ liệu đa số  $\Rightarrow$  Học máy trở lên chính xác hơn
- Chia tập dữ liệu được nhận thành 70% training, 20% validation, 10% testing
- Xây dựng model gồm 4 tầng 1 lớp input đầu vào ảnh 48x48 giống khi mình xử lý ảnh về 48x48 ở trên
- Huấn luyện mô hình
- Xây dựng biểu đồ kết quả Hàm Lost và Accuracy của tập train và test
- Kiểm tra lại mô hình.

## Trình tự các bước chương trình chạy ra sao :

### B1. Import libraries :

Import vô chương trình các thư viện cần thiết như :

- Numpy
- Imgaug

- Pandas
- Seaborn
- Tesorflow
- Sklearn...

## B2. Upload Dữ liệu :

Hình ảnh sẽ được upload thành 1 numpy arrays, với giá trị là [0,1] tương đương với các giá trị là ship hay là no ship.

```
datasets = ['input/satellite-imagery-of-ships']
class_names = ["no-ship", "ship"]
class_name_labels = {class_name:i for i,class_name in enumerate(class_names)}
num_classes = len(class_names)
class_name_labels
{'no-ship': 0, 'ship': 1}
```

Sau đó ảnh và nhãn sẽ được đưa vào một hàm có tên là load data (). Hàm đi qua từng ảnh nhận thông số label của ảnh, ảnh qua đó cũng được chuyển về định dạng (48x48) và chuyển về các màu RGB để dễ cho việc xử lý.

```
def load_data():
    images, labels = [], []

    for dataset in datasets:
        for folder in os.listdir(dataset):
            label = class_name_labels[folder]

            for file in tqdm(os.listdir(os.path.join(dataset, folder))):
                img_path = os.path.join(dataset, folder, file)

                img = cv2.imread(img_path)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img = cv2.resize(img, (48, 48))

                images.append(img)
                labels.append(label)
            pass

        images = np.array(images, dtype=np.float32) / 255.0
        labels = np.array(labels, dtype=np.float32)
        pass

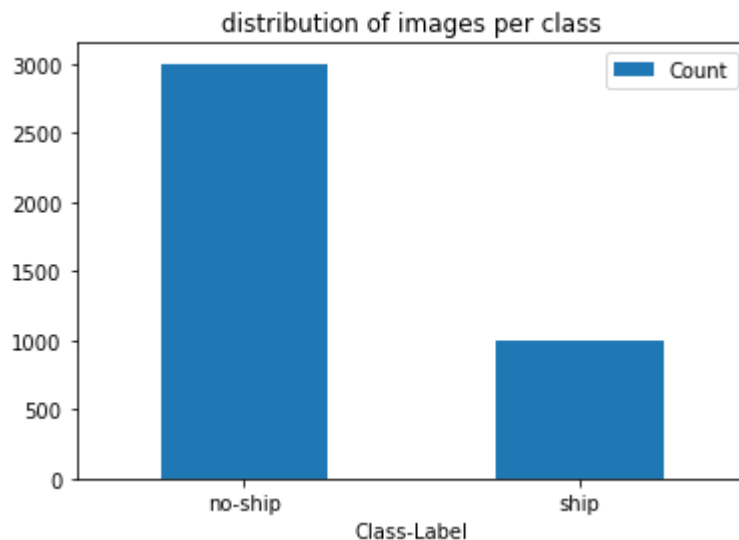
    return (images, labels)
pass
```

```
(images, labels) = load_data()
images.shape, labels.shape
```

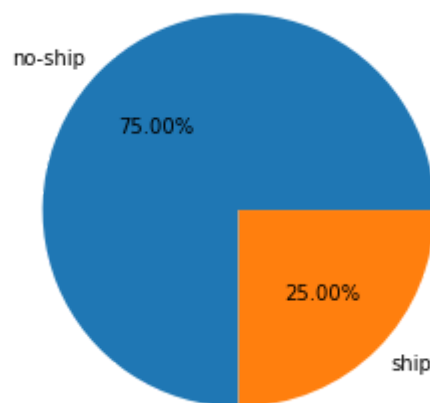
```
100%|████████████████████████████████████████████████████████████████████████████████| 3000/3000 [00:39<00:00, 75.53it/s]
100%|████████████████████████████████████████████████████████████████████████████████| 1000/1000 [00:12<00:00, 82.87it/s]
```

## B3. Phân tích các biểu đồ trực quan :

Ta dùng plot để vẽ lên thì thấy sự phân bố không đồng đều của các nhãn no ship và ship :



Vẽ biểu đồ pie (hình cầu ta được như sau :

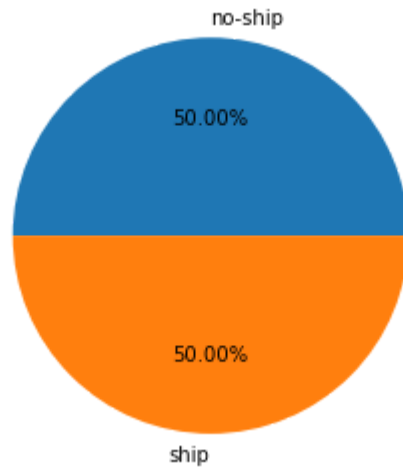


Vì dữ liệu đa số sẽ bị nghiêng theo nhãn là no-ship đến 75% và 25% là ship  $\Rightarrow$  nhãn no-ship sẽ có trọng số cao hơn khi được học và chương trình học máy sẽ dự đoán sai. Để cải thiện điều này ta có thể dùng một số thuật toán để cải thiện lớp nhãn thiểu số.

#### B4. Xử lý dữ liệu :

Như bài toán trên tỉ lệ là 1:3, tức là có 1 tấm có tàu sẽ có 3 tấm là không có. Ta có thể dùng thuật toán tăng cường ảnh để thêm vào 2 tấm có tàu như vậy dữ liệu bài toán sẽ trở nên cân bằng hơn.

Dữ liệu khi được xử lý tăng cường ảnh :



Tiếp đến ta dùng one hot encoding variables để mã hóa các nhãn qua đó loại bỏ được các bias và cân bằng

### **B5: Split train/validation/test**

Trước tiên ta có thể sử dụng random để làm ngẫu nhiên dữ liệu chia.

Ta chia dữ liệu như sau :

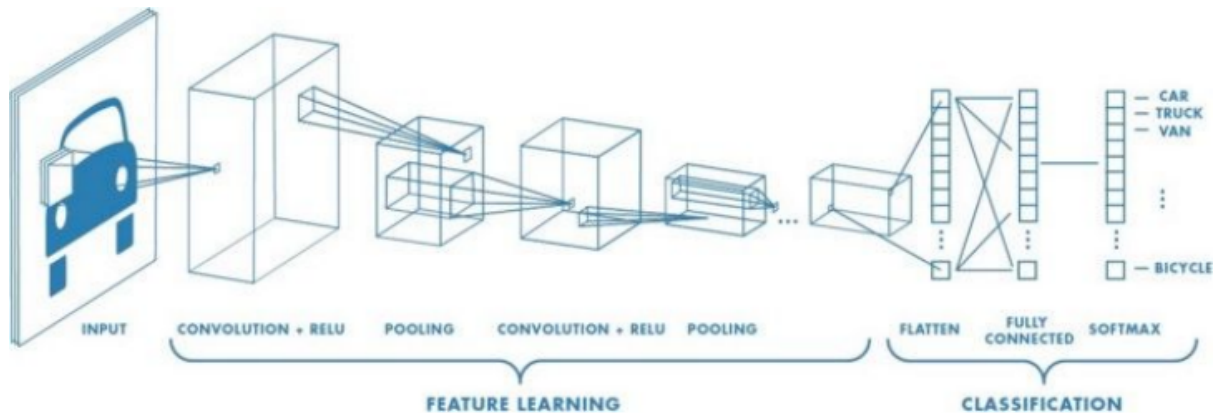
- 70% - Training
- 20% - Validation
- 10% - Testing

### **B6: Xây dựng mô hình CNN**

Mô hình CNN (mạng tích chập) là một thuật toán phổ biến nhằm phân loại ảnh trong Deep Neural Network

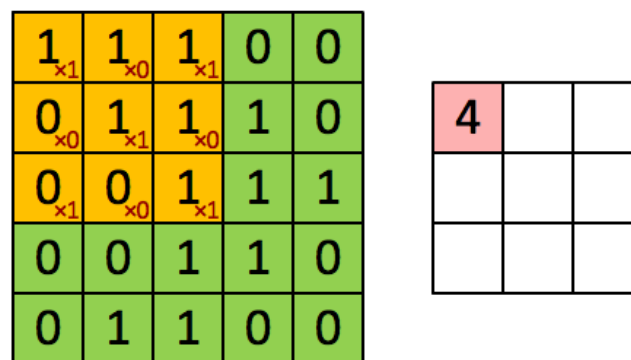
Trong mạng neural, mô hình mạng tích chập (CNN) là một trong những mô hình để nhận dạng và phân loại hình ảnh

Về kỹ thuật, mô hình CNN training và kiểm tra, mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập với các bộ lọc (Kernels), tổng hợp lại các lớp đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng có giá trị xác suất 0 và 1.



Ở bài toán, Xác định tàu như trên, Các input đầu vào sẽ đi qua các layer khác để lấy được các thuộc tính quan trọng nhất. Ta có đi qua các Convolution layer nhân từng phần bên trong ma trận 3x3 (như hình ma trận bên trái)  $\Rightarrow$  để chiết xuất thuộc tính qua đó dùng các thuộc tính này để học dự đoán các dữ liệu mới có thuộc tính giống vậy.

Như trên chúng ta có thể dùng các lớp convulational để lấy các đường nét trắng đen của bài toán ship detection và bỏ qua các yếu tố có ảnh hưởng.



Image

Convolved  
Feature

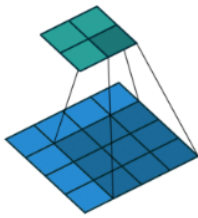
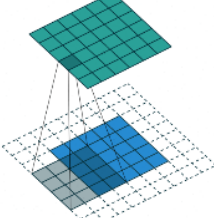
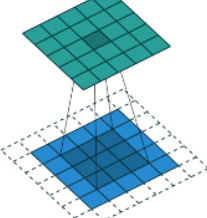
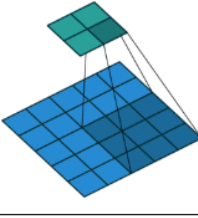
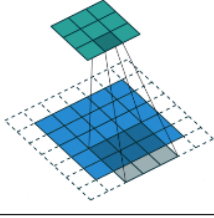
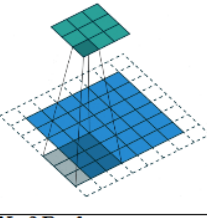
Xây dựng mô hình CNN cơ bản trong bài toán gồm có 1 lớp đầu ra input, 3 lớp trung gian (hidden layer để xử lý tính toán)

- Lớp đầu vào (Block 1) :

Lớp đầu vào sẽ tiếp nhận ảnh 48x48 mà đã được xử lý dữ liệu ở trên.

```
X_input = Input(input_shape)
X = ZeroPadding2D((5,5))(X_input)
```

ZeroPadding nhằm lấy các điểm ảnh vừa với ranh giới của hình đã có.

		
$P = 0, S = 1$	$P \text{ bất kỳ}, S = 1$	$P = (f - 1)/2, S = 1$
		
$P = 0, S = 2$	$P = (f - 1)/2, S = 2$	$\frac{N+2P-f}{S} \notin \mathbb{N}$

Hình 4: Minh họa tích chập hai chiều đơn kênh với các cặp  $(P, S)$ .

- Lớp thứ 2 (Block 2) :

```
# stage 2
X = Conv2D(16, (3,3), strides=(2,2), name='conv1', padding="same") (X)
X = BatchNormalization(name='bn_conv1') (X)

X = conv_block(X, 3, 32, 2, block='A', s=1)
X = MaxPooling2D((2,2)) (X)
X = Dropout(0.25) (X)
```

Block 2 gồm 16 lớp filters và kernel size (3,3) là số tầng của 1 lớp filter layer, stride là số trượt của filter các thông số do developer quyết định. Sau khi đi qua **Batch Normalization**. Thì Batch Normalization là một phương pháp hiệu quả khi training một mô hình mạng nơ ron. Mục tiêu của phương pháp này chính là việc muốn chuẩn hóa các feature đầu ra.

- Lớp thứ 3 và 4 (Block 3 và Block 4) :

```
# Stage 3
X = conv_block(X, 5, 32, 3, block='A', s=2)
X = MaxPooling2D((2,2)) (X)
X = Dropout(0.25) (X)

# Stage 4
X = conv_block(X, 3, 64, 4, block='A', s=1)
X = MaxPooling2D((2,2)) (X)
X = Dropout(0.25) (X)
```

Tiếp tục đi qua lớp thứ 3 và lớp thứ 4, Lớp thứ 3 có 32 lớp filters, kernels size (5,5) stride (2,2), Lớp thứ 4 có 64 filters, kernel size (3,3) stride (2,2). MaxPooling2D và

Dropout là các hàm giảm các lượng neutron và giảm đi vấn đề bị overfitting.

- Output layer

Kết quả sẽ được chuyển qua 1 column sử dụng Flatten Layer và được phân loại sử dụng Dense layer.

## B7: Compiling Model và Tranning Model.

Mô hình sẽ được compiling sử dụng cá hàm như Adam optimizer và Binary Crossentropy.

### Compiling the Model

The model is compiled using the *Adam* optimizer with learning rate set at 1e-3. *Binary Crossentropy* is used as a loss function as there are only two classes.

```
opt = Adam(lr=1e-3)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
logs = TensorBoard("logs")
```

### Training model

The model is trained for 50 epochs with a batch size of 16. The best model weights are stored in the file *model\_weight.h5* with TensorBoard logs being stored in the *logs* directory.

```
epochs = 50
batch_size = 16

history = model.fit(train_images, train_labels,
                    steps_per_epoch=len(train_images)//batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(val_images, val_labels),
                    validation_steps=len(val_images)//batch_size,
                    callbacks=[checkpoint, logs]
                    # class_weight=classWeight # Uncomment if AUGMENTATION is set to FALSE
                    )
```

Sau khi được compiling mô hình sẽ được training với 50 epoch và 16 batch size.

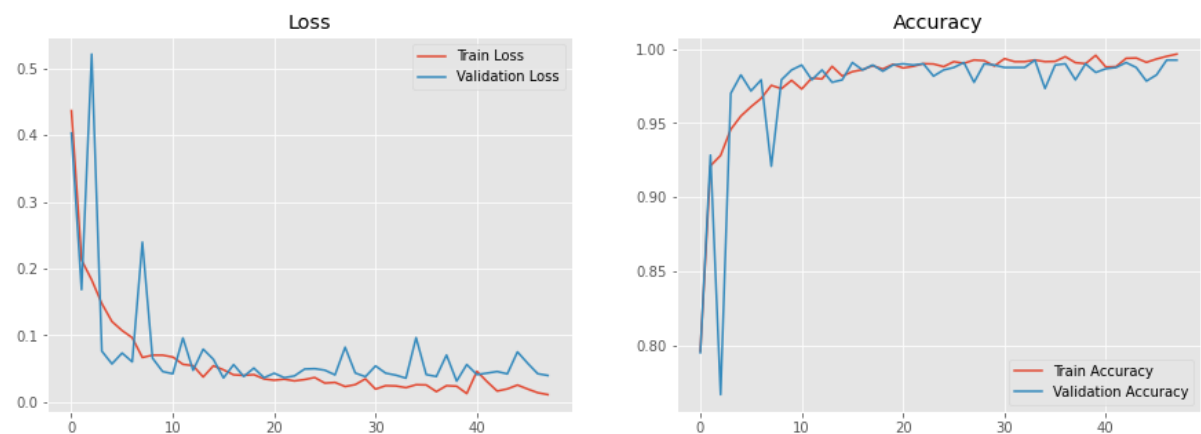
Một Epoch được tính là khi chúng ta đưa tất cả dữ liệu vào mạng neural network 1 lần.

Khi dữ liệu quá lớn, chúng ta không thể đưa hết mỗi lần tất cả tập dữ liệu vào để huấn luyện được. Buộc lòng chúng ta phải chia nhỏ tập dữ liệu ra thành các batch (size nhỏ hơn)

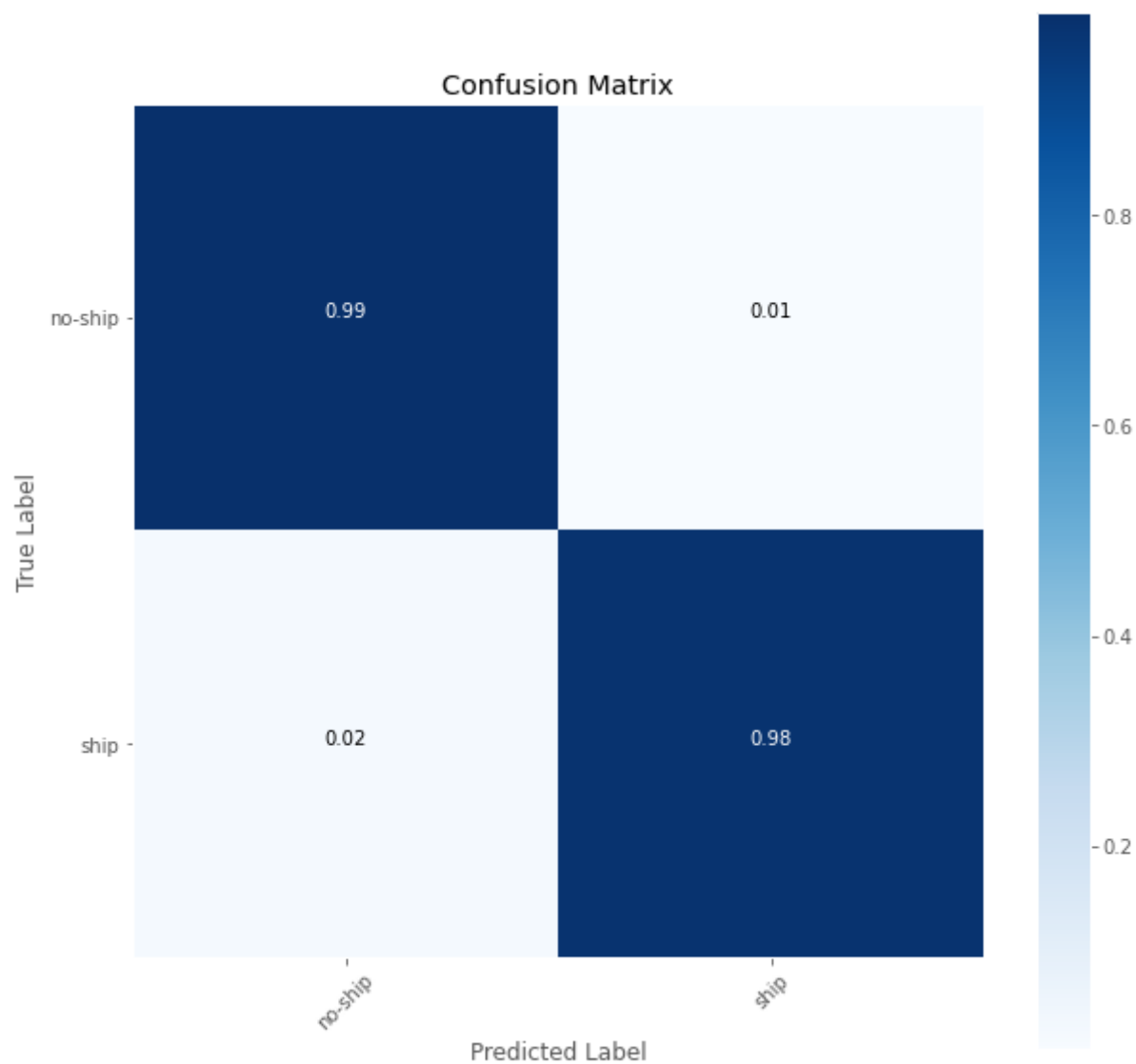
Mô hình sẽ được training đến khi đạt mức độ tốt nhất và sẽ kết thúc khi ở các epoch về sau không có sự thay đổi hiệu năng gì.

## B8. Đánh giá mô hình

Đánh giá qua train/lost

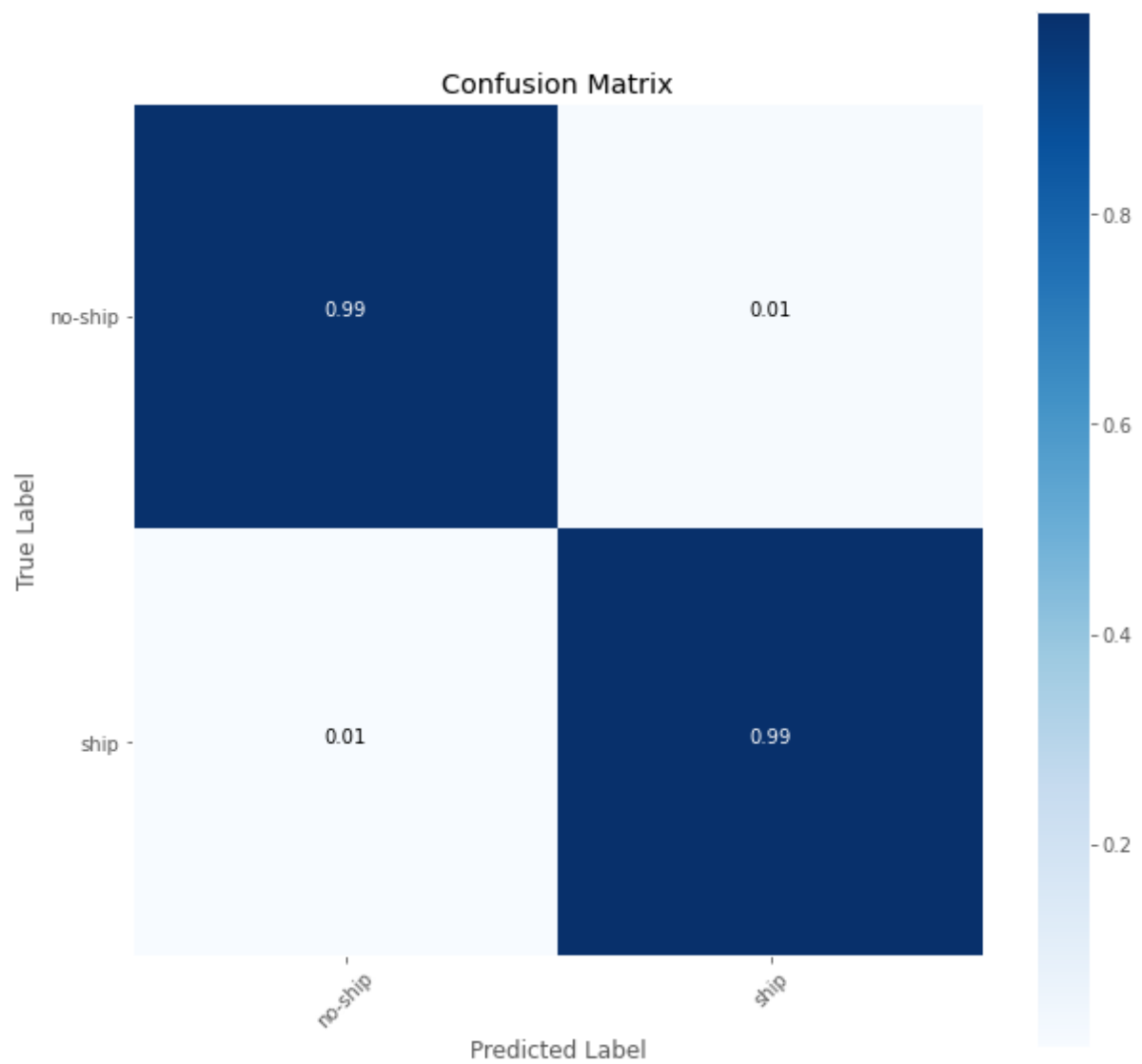


Ở validation test ta có confusion matrix sau :



Dự đoán đúng các nhãn no-ship 0.98%. và ship 0.99%.

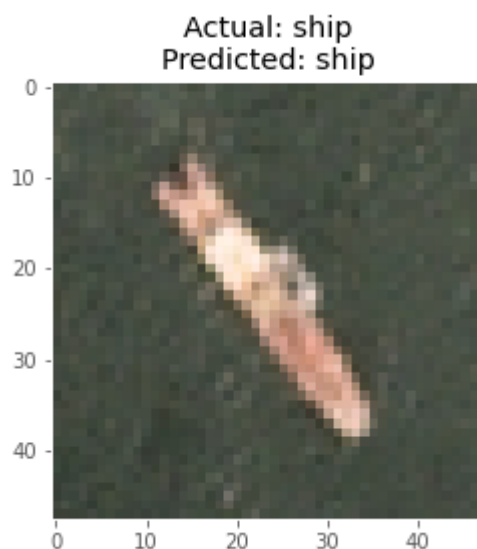
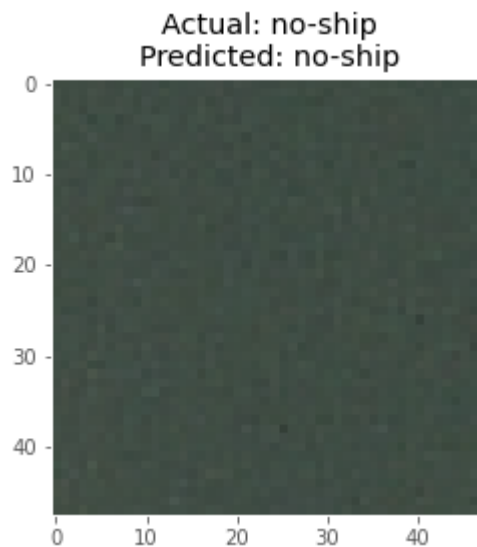




Dự đoán đúng các nhãn no-ship 0.99%. và ship 0.99%.

## B9. Checking model :

Ta sẽ checking lại hình dự đoán của mô hình.



## Áp dụng Embedding Watermark trong Deep Neutral Network :

### Mô hình trên cho ta thấy được :

Để tạo ra một mô hình chỉ là dạng đơn giản trong deep Neutral Network như trên :

- Tốn rất nhiều thời gian, công sức học hỏi của người lập trình.
- Tốn rất nhiều tài nguyên để tạo ra một mô hình hữu ích như là dữ liệu đầu vào để test, CPU máy tính để chạy thử.
- Tốn thời gian chạy mô hình, ...

Vậy nên để bảo vệ được mô hình thì trước hết chúng ta cần nhúng các watermark vào trong mạng deep neutral network. Thì có 2 phương pháp kiểm tra là white box

và black box, vì black box khá là đơn giản, dễ kiểm và có tích chuyên môn ít hơn white box nên nhóm sẽ quyết định sử dụng phương pháp black box để demo.

## Mô hình 1 (Mô hình gốc) :

Mô hình 1 là mô hình gốc có các tầng như đã nói ở trên.

```
X_input = Input(input_shape)

X = ZeroPadding2D((5,5))(X_input)

# stage 2
X = Conv2D(16, (3,3), strides=(2,2), name='conv1', padding="same")(X)
X = BatchNormalization(name='bn_conv1')(X)

X = conv_block(X, 3, 32, 2, block='A', s=1)
X = MaxPooling2D((2,2))(X)
X = Dropout(0.25)(X)

# Stage 3
X = conv_block(X, 5, 32, 3, block='A', s=2)
X = MaxPooling2D((2,2))(X)
X = Dropout(0.25)(X)

# Stage 4
X = conv_block(X, 3, 64, 4, block='A', s=1)
X = MaxPooling2D((2,2))(X)
X = Dropout(0.25)(X)

# Output Layer
X = Flatten()(X)
X = Dense(64)(X)
X = Dropout(0.5)(X)

X = Dense(128)(X)
X = Activation("relu")(X)

X = Dense(classes, activation="softmax", name="fc"+str(classes))(X)

model = Model(inputs=X_input, outputs=X, name='Feature_Extraction_and_FC')

return model
```

## Mô hình 2 (Mô hình copy) :

Mô hình 2 là mô hình copy, Nhìn thì thấy sẽ có sự giống nhau gần như tương đương tuy nhiên không thể đánh giá mô hình là copy được nếu chưa có bằng chứng rõ ràng.

```

X_input = Input(input_shape)

X = ZeroPadding2D((5,5))(X_input)

# stage 2
X = Conv2D(16, (3,3), strides=(2,2), name='conv1', padding="same")(X)
X = BatchNormalization(name='bn_conv1')(X)

X = conv_block(X, 3, 32, 2, block='A', s=1)
X = MaxPooling2D((2,2))(X)
X = Dropout(0.25)(X)

# Stage 3
X = conv_block(X, 5, 64, 3, block='A', s=2)
X = MaxPooling2D((2,2))(X)
X = Dropout(0.25)(X)

# Stage 4
X = conv_block(X, 3, 64, 4, block='A', s=1)
X = MaxPooling2D((2,2))(X)
X = Dropout(0.25)(X)

# Output Layer
X = Flatten()(X)
X = Dense(64)(X)
X = Dropout(0.5)(X)

X = Dense(128)(X)
X = Activation("sigmoid")(X)

X = Dense(classes, activation="softmax", name="fc"+str(classes))(X)

model = Model(inputs=X_input, outputs=X, name='Feature_Extraction_and_FC')
return model

```

Ta bắt đầu kiểm tra, Ở tầng input khá là giống nhau. Khi ta kiểm tra các tầng còn lại ta có thể thấy 1 chi tiết nhỏ ở mô hình 2 ở tầng 3 số filter khác với Mô hình 1 khi có 64 lớp filters khác với 32 lớp filters ở tầng 1. Ngoài ra ở tầng activation thì mô hình 2 sử dụng activation là sigmoid khác với relu ở mô hình 1.

```

def conv_block(X, k, filters, stage, block, s=1):

    conv_base_name = 'conv_' + str(stage)+block+'_branch'
    bn_base_name = 'bn_'+str(stage)+block+"_branch"

    F1 = filters

    X = Conv2D(filters=F1, kernel_size=(k,k), strides=(s,s),
               padding='same', name=conv_base_name+'2a')(X)
    X = BatchNormalization(name=bn_base_name+'2a')(X)
    X = Activation('sigmoid')(X)

    return X
pass

```

Ở hàm để xây block cũng có thể thấy relu đã bị thay thế. Thế nên nếu ta so sánh 2 kết quả của 2 mô hình ngay lúc này sẽ xảy output bị sai số. Người sử dụng đã tấn công như là **Fine-tuning** nhằm thay đổi một số tham số tuy nhiên vẫn giữ được output đầu ra ⇒ Không thể xác định được mô hình có copy của mô hình 1 không do vậy chúng ta cần nên chuyển các tham số gần giống mô hình 1 để so kết quả.

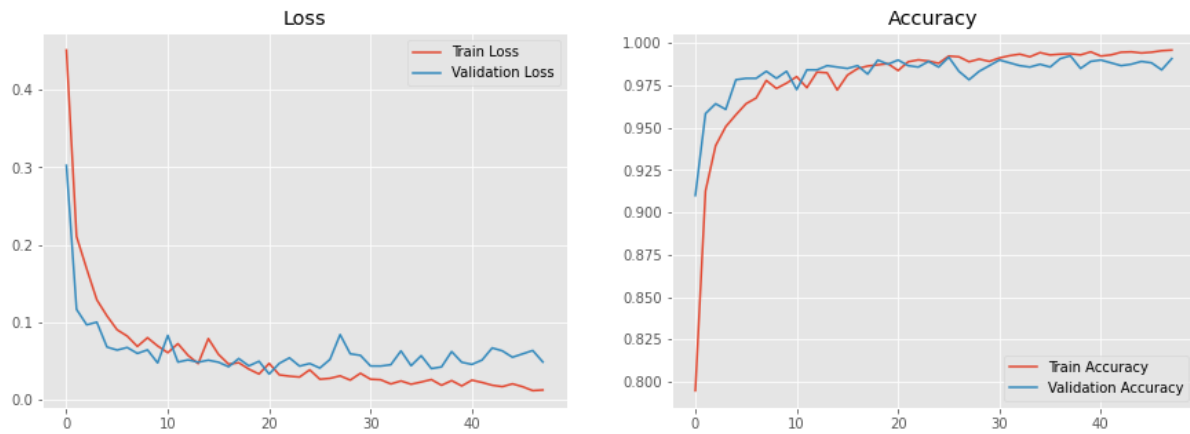
## So sánh kết quả :

Sau khi thay đổi các tham số về mô hình ta có thể so sánh :

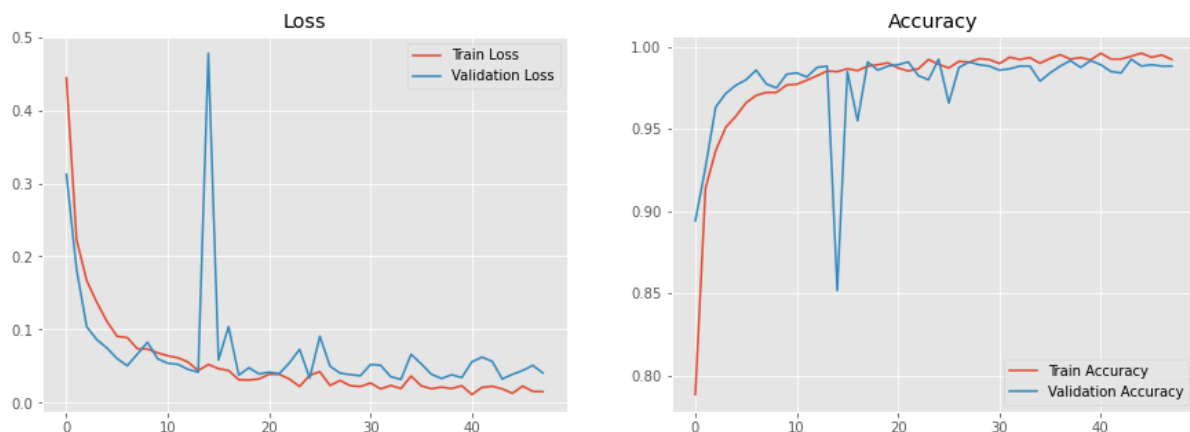
Ở số lần epoch chạy cả 2 mô hình đều dừng ở epoch 48/50.

## Loss và Accuracy, ta được :

Mô hình 1 :

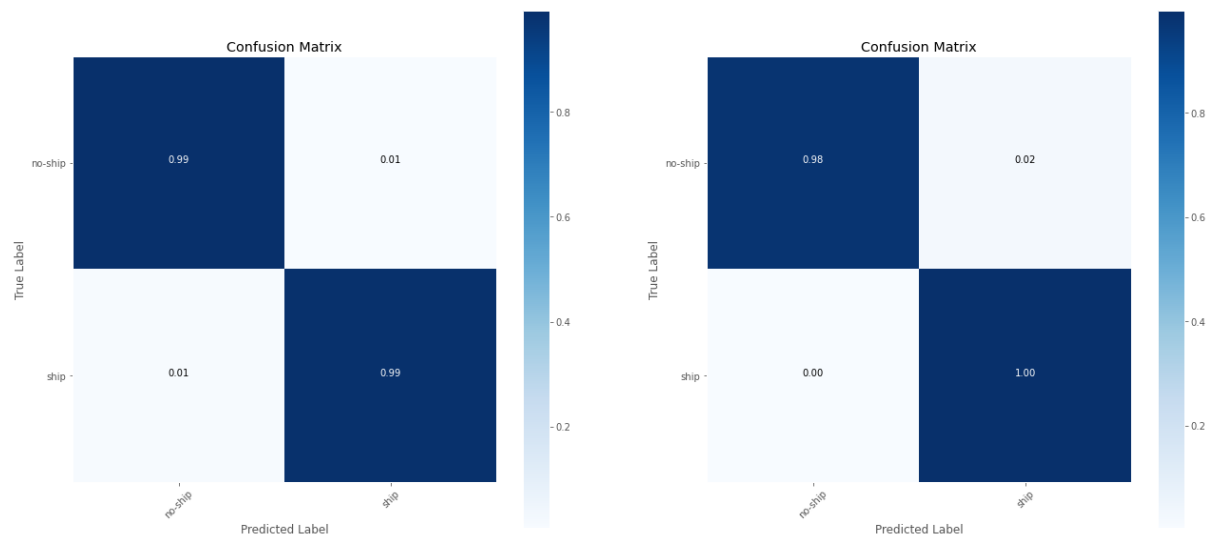


Mô hình 2 :



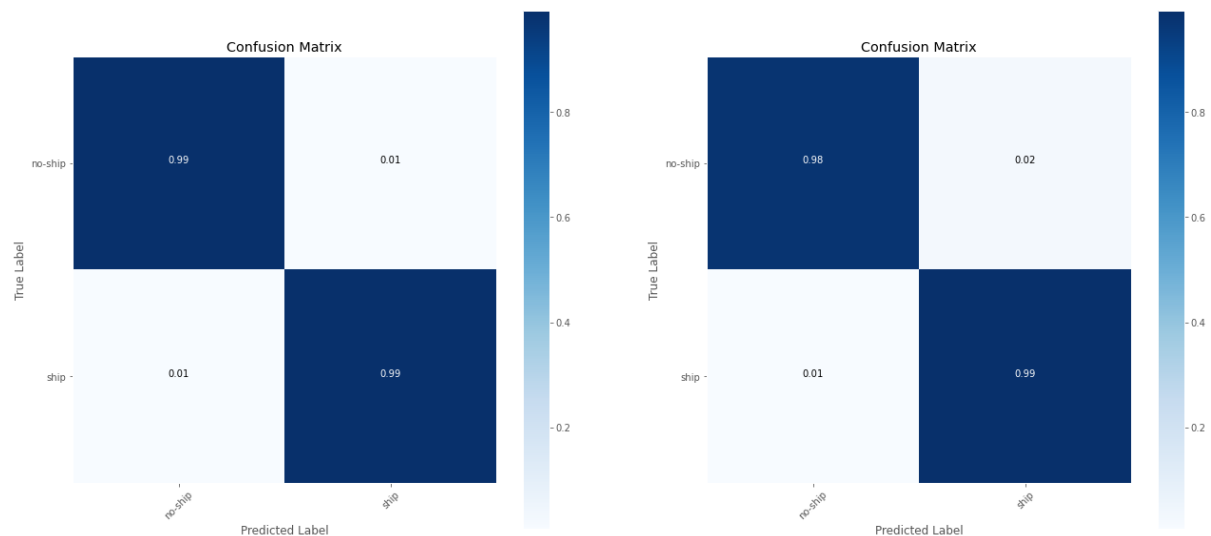
Ở giá trị loss, validation loss khá là khác biệt so với cả hai mô hình, tuy nhiên giá trị train loss của cả 2 thì khá là tương đương nhau. Giá trị của accuracy cũng như vậy cho thấy trong quá trình chạy train test 2 chương trình khá là tương đồng tuy nhiên chưa thể chắc chắn được.

## Confusion Matrix for Validation dataset :



Ở Confusion Matrix for Validation, thì các giá trị có sự chênh lệch tương đối nhỏ. Dự đoán no-ship và ship có sự chênh lệch là 0.01% cho thấy độ tương đồng của mô hình

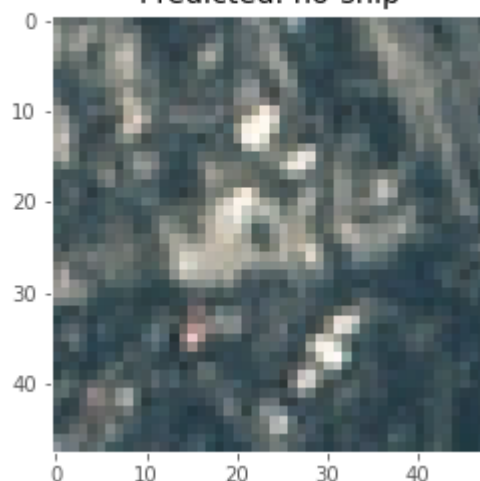
### Confusion Matrix for Testing Dataset :



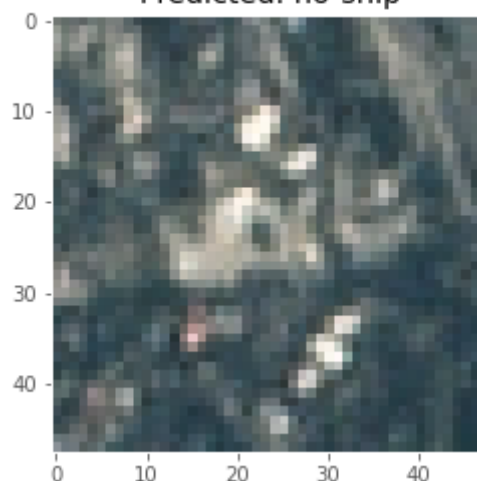
Đúng như biểu đồ loss, accuracy quá trình training và test kết quả khá là tương đồng nên kết quả khá là giống nhau và chỉ khác biệt khi dự đoán no-ship và không có khác biệt nhiều.

### Checking model :

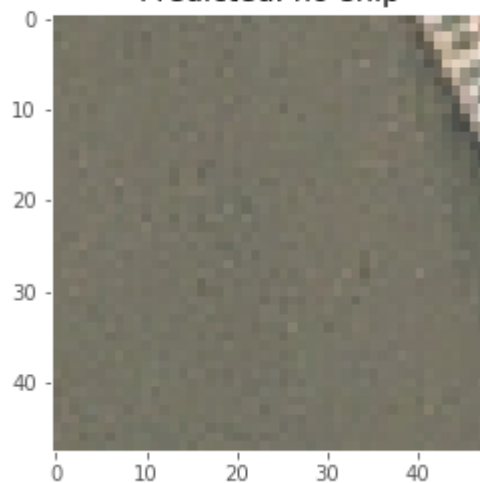
Actual: no-ship  
Predicted: no-ship



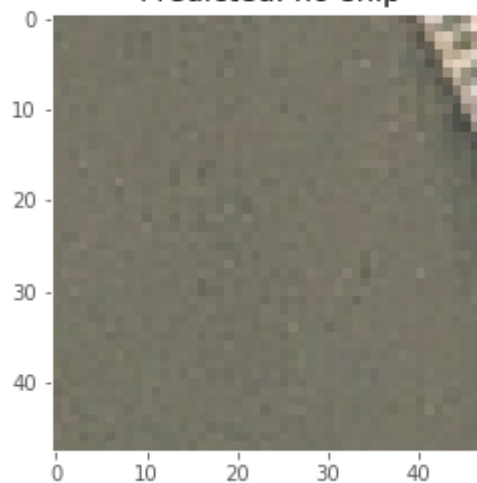
Actual: no-ship  
Predicted: no-ship



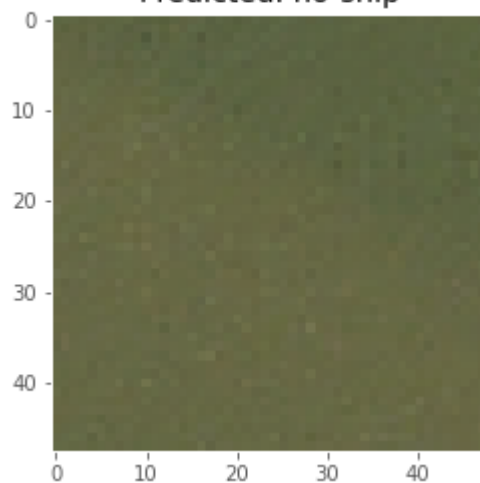
Actual: no-ship  
Predicted: no-ship



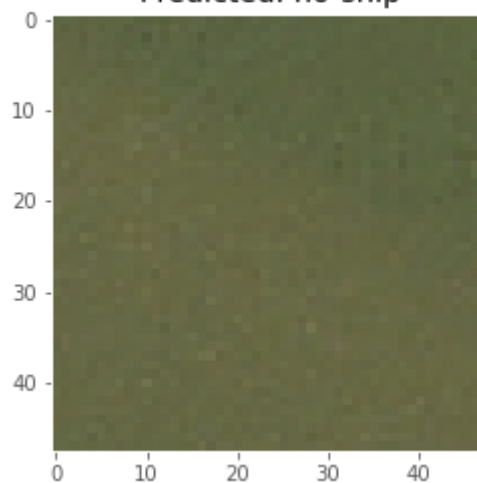
Actual: no-ship  
Predicted: no-ship



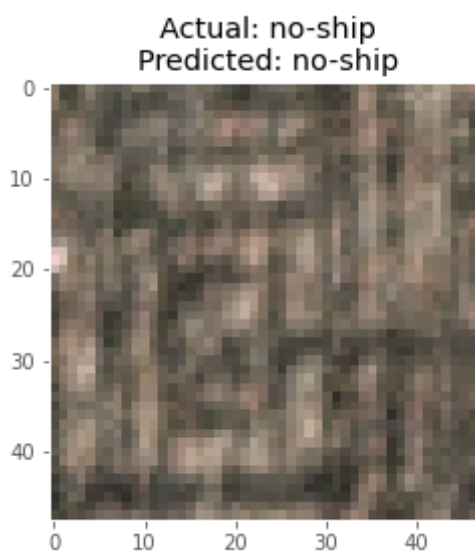
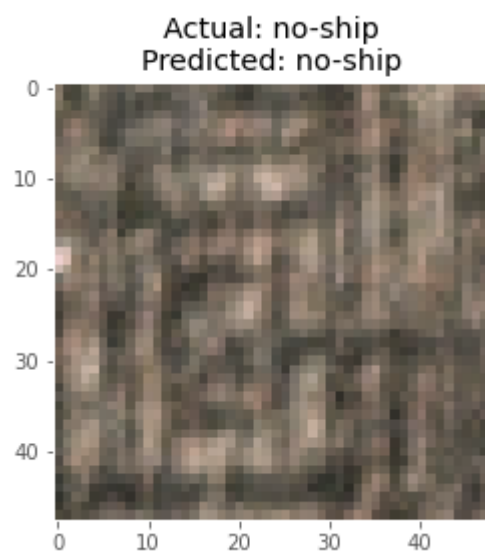
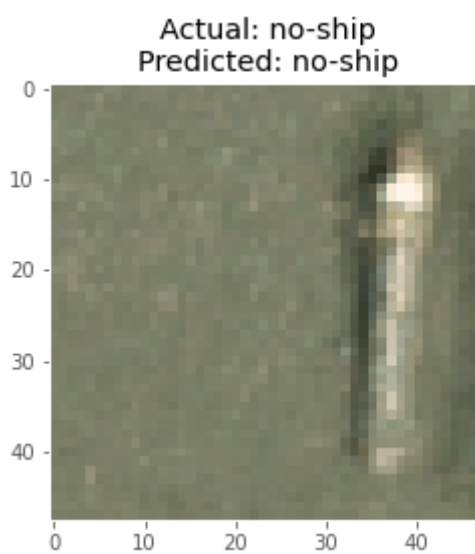
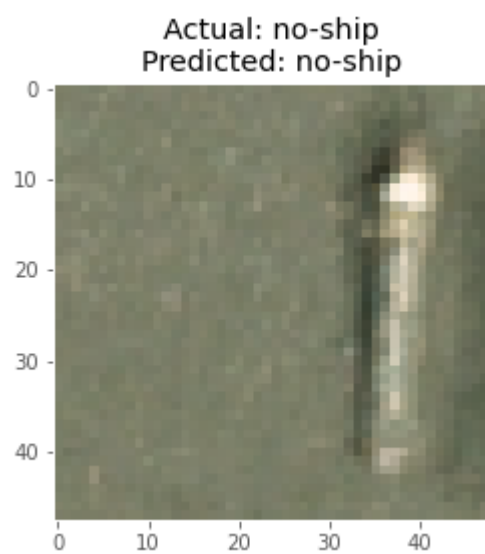
Actual: no-ship  
Predicted: no-ship



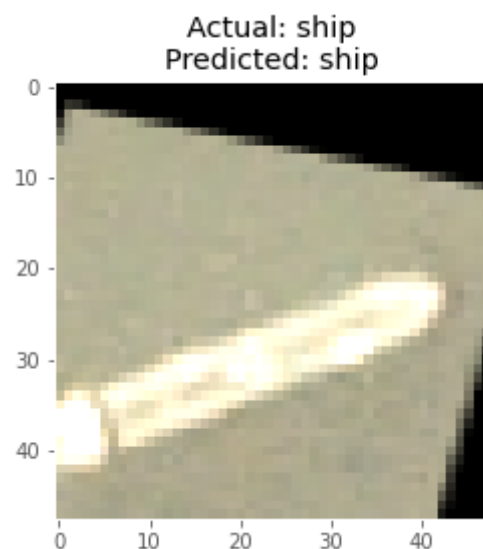
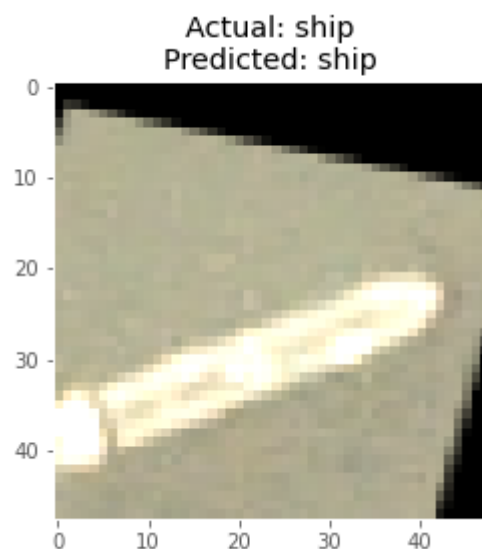
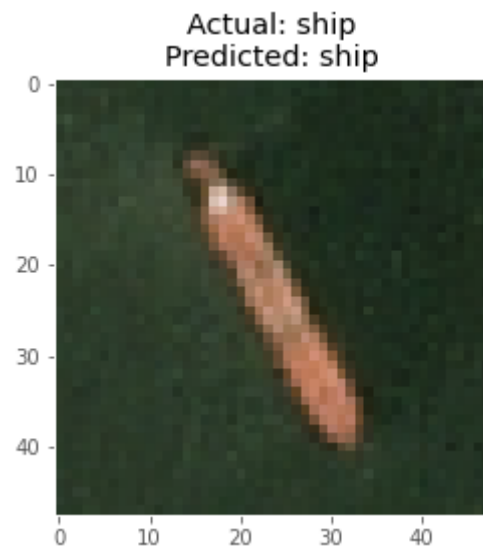
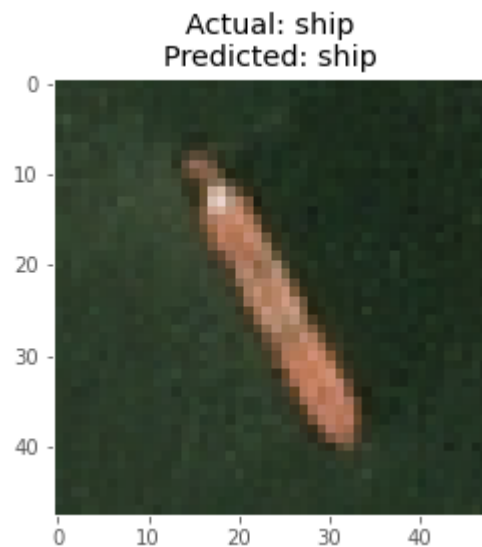
Actual: no-ship  
Predicted: no-ship



Đặc biệt là ở một số hình, 2 mô hình lại sai chung 1 hình ảnh như hình sau :







Đây là một số hình ảnh output, nhìn chung 2 mô hình đưa ra kết quả khá là tương đồng với nhau.

## Nhận xét chung :

Nhìn chung phương pháp black box đã làm tốt nhiệm vụ khi cho thấy điểm tương đồng. Rõ ràng black box nhìn hơn white box ở khoản kiểm tra. Tuy nhiên về mặt chất lượng thì black box chưa làm tốt được vì kết quả kiểm tra sẽ có sai số khác nhau ở output ở mỗi lần chạy nên. Nên kết quả chỉ cho biết sự tương đồng hay không, do đó không thể nên kết luận sớm.

Do vậy các đội nhóm developer, developer cần nên tìm hiểu các phương pháp như White Box. Chỉnh sửa các thông số kỹ thuật của mô hình qua đó đặt được nhãn và phát hiện nhãn qua mô hình sẽ chính xác hơn.

⇒ Chúng ta có thể sử dụng phương pháp Black-box để điều tra và White-box để truy tố pháp luật hình sự.