```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from keras.layers import Dense,Flatten
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from keras.applications.vgg16 import VGG16
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.model_selection import train_test_split
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Loading the test labels

- Reading test labels from the csv
- Conducting stratified train-validation split

```
input_data_folder = "drive/MyDrive/GovTech VA Assessment/Assignment 1 - Training Data/"
# input_data_folder = "./Assignment 1 - Training Data/"
```

```
labels_df = pd.read_csv(input_data_folder+'labels.csv')
labels_df.head()
```

	image	category
0	2788353.jpg	0
1	2782131.jpg	0
2	2884349.jpg	0
3	2900596.jpg	0
4	2841543.jpg	0

```
# Convert the category to str
labels_df.loc[:,"category"] = labels_df.loc[:,"category"].astype(str)
```

```
labels_df.shape[0]
```

```
900
```

```
# Checking the number of unique categories
n_classes = labels_df.category.nunique()
```

```
n_classes
```

```
5
```

```
# Checking the distribution of categories
labels_df.groupby("category").count()
```

	image
category	
0	100
1	200
2	200
3	200
4	200

```
# Stratified train-validation split split
# Especially with such a small dataset and numerious different categories, it is important
# 80-20 train-validation split is done to ensure that the model has sufficient data to tra
image_train, image_val, category_train, category_val = train_test_split(labels_df.image, l
```

```
# Separating the Training Data
train_df = labels_df.loc[labels_df.image.isin(image_train),:].copy()
train_df.groupby("category").count()
```

	image
category	
0	80
1	160
2	160
3	160
4	160

```
train_df.head()
```

	image	category

```
# Separating the Validation Data
val_df = labels_df.loc[labels_df.image.isin(image_val),:].copy()
val_df.groupby("category").count()
```

	image
category	
0	20
1	40
2	40
3	40
4	40

```
val_df.head()
```

	image	category
1	2782131.jpg	0
2	2884349.jpg	0
14	2798385.jpg	0
20	2837597.jpg	0
21	2829278.jpg	0

Data Augmentation Testing

Important to generate samples for manual inspection of the Augmented image to ensure that the model is learning pictures that are realistic and recognisable by humans too

```
datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    width_shift_range=0.2, height_shift_range=0.1,
    brightness_range=[0.5, 1.5],
    # rotation_range=30, fill_mode='nearest',
    # zoom_range=0.2
    )
```

```
# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow_from_dataframe(
        dataframe=train_df,
        directory=input_data_folder,
```

```
        x_col="image",
        y_col="category",
        batch_size=9,
        shuffle=True,
        class_mode="categorical",
    target_size=(224,224)):

    # create a grid of 3x3 images
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)
        plt.imshow(X_batch[i].reshape(224, 224,3))
    # show the plot
    plt.show()
    break
```

```
Found 720 validated image filenames belonging to 5 classes.
```

Data Augmentation Implementation

Train Set Image Augmentation

Important Benefits:

- **Increase the training data size**: Especially since the model only has 80-160 original images to learn from.
- **Reduce overfitting of model**: Allows the models to learn to classify the image even when the input image is noisy

```
input_image_generators = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    width_shift_range=0.2, height_shift_range=0.1,
    brightness_range=[0.5, 1.5],
    # rotation_range=30, fill_mode='nearest',
    # zoom_range=0.2
    )
```

```
train_data_gen = input_image_generators.flow_from_dataframe(
        dataframe=train_df,
        directory=input_data_folder,
        x_col="image",
        y_col="category",
        batch_size=20,
        seed=42,
        shuffle=True,
        class_mode="categorical",
    target_size=(224,224))
```

```
Found 720 validated image filenames belonging to 5 classes.
```

Validation Set Image Augmentation

No augmentation to preserve reality

```
input_image_generators = ImageDataGenerator(
    rescale=1./255
    )
```

```
validation_data_gen = input_image_generators.flow_from_dataframe(
        dataframe=val_df,
        directory=input_data_folder,
        x_col="image",
        y_col="category",
        batch_size=20,
        seed=42,
        # shuffle=True,
        class_mode="categorical",
    target_size=(224,224))
```

```
Found 180 validated image filenames belonging to 5 classes.
```

Model Building: Using the Pretrained VGG16 Model to extract Features

Due to the severe lack of training data, the model will have to be built upon the pretrained VGG16 model and not trained from scratch.

```
# Loading the pretrained VGG16 model without the fully connected layers
vgg_model = VGG16(include_top=False, weights='imagenet', input_shape=(224,224,3))
print(vgg_model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
58892288/58889256 [==============================] - 0s 0us/step
```

```
58900480/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
 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         
                                                                 
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
None
```

```
# Fixing the weights of the convolutional layers
# Due to the lack of sizeable training data to train the weights of the layers, the convol
for layer in  vgg_model.layers:
    layer.trainable = False
```

```
# Create a new 'top' fully-connected layers
# Trial and tested. Due to the lack of sizeable training data to train the weights of the
```

```
top_model = vgg_model.output
top_model = Flatten(name="flatten")(top_model)
output_layer = Dense(n_classes, activation='softmax')(top_model)

# Group the convolutional base and new fully-connected layers into a Model object.
final_model = Model(inputs=vgg_model.input, outputs=output_layer)
```

```
# Compile the model
final_model.compile(
    optimizer= Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

```
final_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, 5)                125445

=================================================================
Total params: 14,840,133
Trainable params: 125,445
Non-trainable params: 14,714,688
_________________________________________________________________
```

Model Training

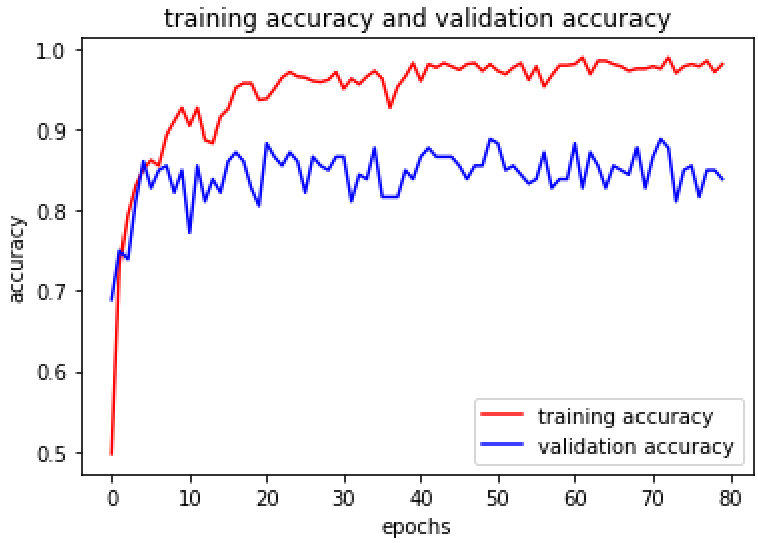
```
n_steps = train_data_gen.samples // 20
n_val_steps = validation_data_gen.samples // 20
n_epochs = 100
```

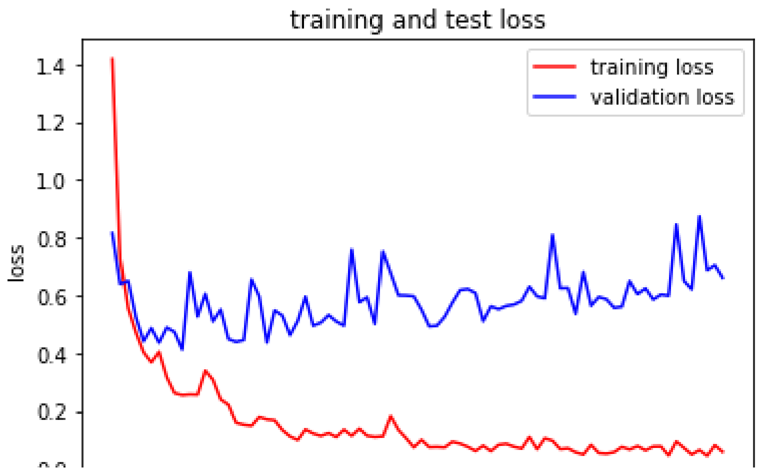
```
checkpoint = ModelCheckpoint("vgg16_3.h5", monitor='val_accuracy', verbose=1, save_best_on
early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=30, verbose=1, mode='a

history = final_model.fit(train_data_gen,
                          batch_size=20,
                          epochs=100,
                          validation_data=validation_data_gen,
                          steps_per_epoch=n_steps,
                          validation_steps=n_val_steps,
                          callbacks=[checkpoint, early],
                          verbose=1)
final_model.save_weights("vgg16_3.h5")
```

```
Epoch 1/100
36/36 [==============================] - 159s 4s/step - loss: 1.4181 - accuracy: 0
Epoch 2/100
36/36 [==============================] - 13s 347ms/step - loss: 0.7268 - accuracy:
Epoch 3/100
27/36 [=====================>........] - ETA: 2s - loss: 0.5760 - accuracy: 0.7815W
36/36 [==============================] - 13s 349ms/step - loss: 0.5541 - accuracy:
Epoch 4/100
36/36 [==============================] - 13s 347ms/step - loss: 0.4703 - accuracy:
Epoch 5/100
36/36 [==============================] - 13s 348ms/step - loss: 0.4014 - accuracy:
Epoch 6/100
19/36 [==============>...............] - ETA: 5s - loss: 0.3311 - accuracy: 0.8816W
36/36 [==============================] - 13s 350ms/step - loss: 0.3673 - accuracy:
Epoch 7/100
36/36 [==============================] - 13s 352ms/step - loss: 0.4034 - accuracy:
Epoch 8/100
36/36 [==============================] - 13s 353ms/step - loss: 0.3154 - accuracy:
Epoch 9/100
11/36 [========>.....................] - ETA: 8s - loss: 0.2979 - accuracy: 0.8909W
36/36 [==============================] - 13s 358ms/step - loss: 0.2607 - accuracy:
Epoch 10/100
36/36 [==============================] - 13s 350ms/step - loss: 0.2535 - accuracy:
Epoch 11/100
```

```
36/36 [==============================] - 13s 348ms/step - loss: 0.2559 - accuracy:
Epoch 12/100
 3/36 [=>............................] - ETA: 10s - loss: 0.3238 - accuracy: 0.900
36/36 [==============================] - 13s 351ms/step - loss: 0.2549 - accuracy:
Epoch 13/100
36/36 [==============================] - 13s 351ms/step - loss: 0.3384 - accuracy:
Epoch 14/100
31/36 [========================>.....] - ETA: 1s - loss: 0.3245 - accuracy: 0.8774
36/36 [==============================] - 13s 349ms/step - loss: 0.3063 - accuracy:
Epoch 15/100
36/36 [==============================] - 13s 349ms/step - loss: 0.2386 - accuracy:
Epoch 16/100
36/36 [==============================] - 12s 345ms/step - loss: 0.2200 - accuracy:
Epoch 17/100
23/36 [==================>...........] - ETA: 4s - loss: 0.1378 - accuracy: 0.9587
36/36 [==============================] - 12s 345ms/step - loss: 0.1593 - accuracy:
Epoch 18/100
36/36 [==============================] - 13s 346ms/step - loss: 0.1523 - accuracy:
Epoch 19/100
36/36 [==============================] - 12s 345ms/step - loss: 0.1487 - accuracy:
Epoch 20/100
15/36 [===========>..................] - ETA: 6s - loss: 0.2284 - accuracy: 0.9133
36/36 [==============================] - 13s 347ms/step - loss: 0.1790 - accuracy:
Epoch 21/100
36/36 [==============================] - 13s 346ms/step - loss: 0.1710 - accuracy:
Epoch 22/100
36/36 [==============================] - 13s 346ms/step - loss: 0.1677 - accuracy:
Epoch 23/100
 7/36 [====>.........................] - ETA: 9s - loss: 0.1228 - accuracy: 0.9714
36/36 [==============================] - 13s 346ms/step - loss: 0.1340 - accuracy:
Epoch 24/100
36/36 [==============================] - 12s 344ms/step - loss: 0.1117 - accuracy:
Epoch 25/100
```

```
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
plt.plot(training_accuracy, 'r', label = 'training accuracy')
plt.plot(validation_accuracy, 'b', label = 'validation accuracy')
plt.title('training accuracy and validation accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

training_accuracy = history.history['loss']
validation_accuracy = history.history['val_loss']
plt.plot(training_accuracy, 'r', label = 'training loss')
plt.plot(validation_accuracy, 'b', label = 'validation loss')
plt.title('training and test loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```





Load and test Trained Model

Loading Trained Model

```
# CNN Template
def vgg16_model_1():
    vgg_model = VGG16(include_top=False, weights='imagenet', input_shape=(224,224,3))
    for layer in  vgg_model.layers:
        layer.trainable = False

    # Create a new 'top' of the model (i.e. fully-connected layers).
    top_model = vgg_model.output
    top_model = Flatten(name="flatten")(top_model)
    output_layer = Dense(n_classes, activation='softmax')(top_model)

    # Group the convolutional base and new fully-connected layers into a Model object.
    final_model = Model(inputs=vgg_model.input, outputs=output_layer)

    # final_model.compile(optimizer= Adam(learning_rate=0.001), loss='categorical_crossent
    return final_model
```

```
# Initialising the model and loading the trained weights
model = vgg16_model_1()
model.load_weights('vgg16_3.h5')
```

Testing Single Image

```
load_img(input_data_folder+"299281.jpg")
```

```
image_array = img_to_array(load_img(input_data_folder+"299281.jpg",target_size=(224, 224))

# VGG16 expect a batch of images as input
img_batch = np.expand_dims(image_array, axis=0)
pred_results = model.predict(img_batch)
```

```
# Category Prediction
print(f"Category of ship: {np.argmax(pred_results,axis=1)[0]}")
```

```
Category of ship: 0
```

Testing multiple images

```
labels_df = pd.read_csv(input_data_folder+'labels.csv')
labels_df.loc[:,"category"] = labels_df.loc[:,"category"].astype(str)
labels_df.head()
```

	image	category
0	2788353.jpg	0
1	2782131.jpg	0
2	2884349.jpg	0
3	2900596.jpg	0
4	2841543.jpg	0

```
# Stratified train-validation split split
image_train, image_val, category_train, category_val = train_test_split(labels_df.image, l
```

```
val_df = labels_df.loc[labels_df.image.isin(image_val),:].copy()
val_df.groupby("category").count()
```

	image
category	
0	20
1	40
2	40
3	40
4	40

```
input_image_generators = ImageDataGenerator(
    rescale=1./255
    )
```

```
validation_data_gen = input_image_generators.flow_from_dataframe(
        dataframe=val_df,
        directory=input_data_folder,
        x_col="image",
        y_col="category",
        batch_size=20,
        seed=42,
        shuffle=False,
        class_mode="categorical",
    target_size=(224,224))
```

```
Found 180 validated image filenames belonging to 5 classes.
```

```
nb_samples = validation_data_gen.n
prediction = model.predict(validation_data_gen,steps = nb_samples)
```

```
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that
```

```
np.argmax(prediction, axis=1)
```

```
array([0, 0, 4, 0, 0, 0, 4, 1, 1, 0, 0, 0, 0, 4, 0, 0, 0, 0, 4, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
       1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 1, 3, 3, 3, 4, 4, 1, 4, 4, 4, 0, 4, 2, 4, 4, 4, 4, 0,
       1, 4, 4, 4, 4, 4, 4, 1, 1, 4, 4, 4, 0, 0, 4, 0, 1, 4, 4, 4, 4, 0,
       1, 4, 4, 0])
```

```
val_df.loc[:,"prediction"] = np.argmax(prediction, axis=1)

print("Accuracy: ",round((val_df.category == val_df.prediction.astype(str)).sum() / 180 *
```

```
Accuracy:  83.89 %
```