```
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os
import numpy as np
import matplotlib.pyplot as plt



# Colab library to upload files to notebook
from google.colab import files

# Install Kaggle library
!pip install -q kaggle

import os
os.environ['KAGGLE_USERNAME'] = "montysteele" # username from the json file
os.environ['KAGGLE_KEY'] = "a3b1a776c48c701e0adcfc08a8d23c22" # key from the json file
!kaggle datasets download -d rhammell/ships-in-satellite-imagery # api copied from kag
```

```
Downloading ships-in-satellite-imagery.zip to /content
 98% 181M/185M [00:04<00:00, 39.3MB/s]
100% 185M/185M [00:04<00:00, 46.0MB/s]
```

```
from zipfile import ZipFile
# Create a ZipFile Object and load sample.zip in it
with ZipFile('ships-in-satellite-imagery.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()


!mkdir ship
!mkdir noship

! cp shipsnet/shipsnet/0* noship
! cp shipsnet/shipsnet/1* ship

!mkdir data
!mv  ship data
!mv  noship data


#!ls shipsnet/shipsnet #&& ls -1 | wc -l # 4000 files as expected
!ls data
```

```

batch_size = 128
epochs = 15
```

```
epochs = 15
IMG_HEIGHT = 160
IMG_WIDTH = 160

split_size = 0.2

train_data_dir = '/content/data'

total_images = 4000
total_train = total_images*split_size
total_val = total_images*(1 - split_size)

print(total_train)
print(total_val)
```

```
train_datagen = ImageDataGenerator(rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=45,
    horizontal_flip=True,
    validation_split=split_size) # set validation split

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True,
    subset='training') # set as training data

validation_generator = train_datagen.flow_from_directory(
    train_data_dir, # same directory as training data
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False,
    subset='validation') # set as validation data
```

```
# This function will plot images in the form of a grid with 1 row and 5 columns where
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
```

```
        plt.tight_layout()
        plt.show()

    sample_training_images, _ = next(train_generator)

    plotImages(sample_training_images[:5])
```

```
    '''
    base_model = tf.keras.applications.vgg19.VGG19(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3),
                                                   include_top=False, weights='imagenet')
    # ~85% after 15 epochs

    '''

    '''
    base_model = tf.keras.applications.inception_resnet_v2.InceptionResNetV2(include_top=F
                                                          weights='imagenet', input_sha
    # ~96% after 15 epochs

    '''


    base_model = tf.keras.applications.MobileNetV2(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3),
                                                   include_top=False,
                                                   weights='imagenet')

    # ~98.6% after 15 epochs
    # ~99% after 50 epochs


    '''
    base_model = tf.keras.applications.xception.Xception(include_top=False, weights='image
                                                         input_shape=(IMG_HEIGHT, IMG_WIDT

    # ~96.9%
    '''
```

```
base_model.trainable = False

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
mid_layer = tf.keras.layers.Dense(4096)
prediction_layer = tf.keras.layers.Dense(1)


model = tf.keras.Sequential([
  base_model,
  global_average_layer,
  mid_layer,
  prediction_layer
])

base_learning_rate = 5e-6
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.summary()
```

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=total_train // batch_size,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=total_val // batch_size
)
```

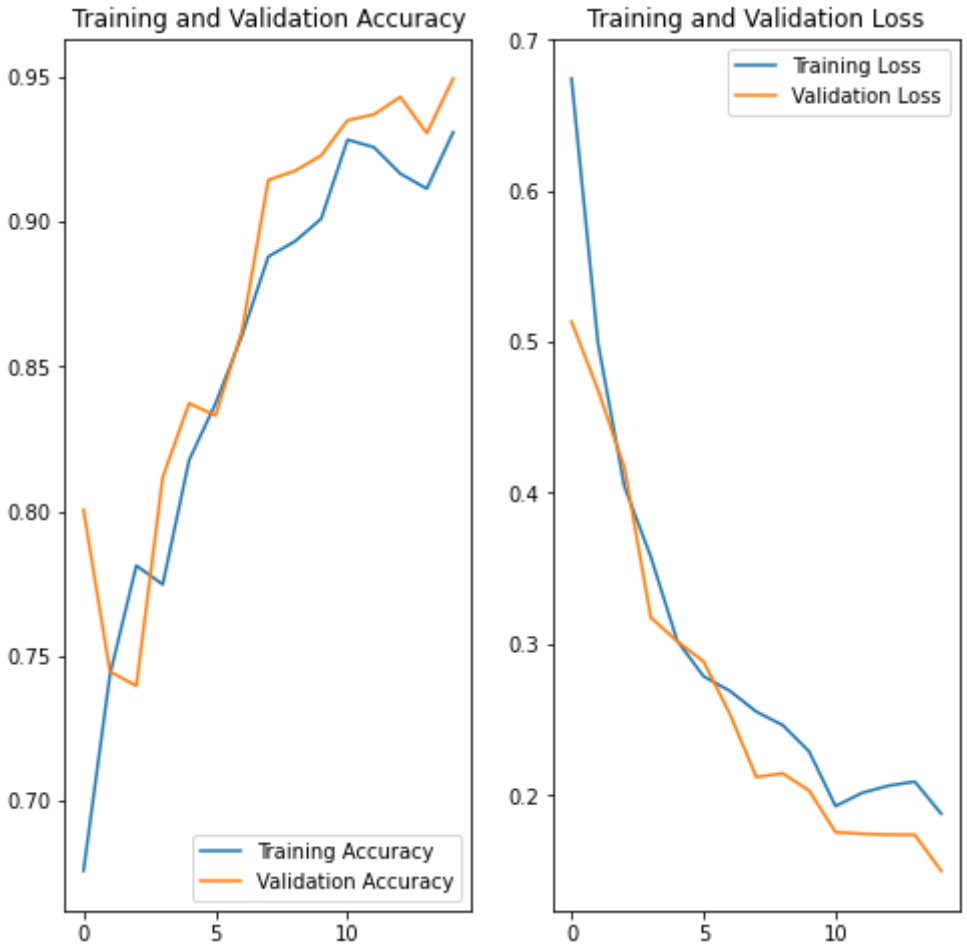
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range = range(epochs)
#epochs_range = range(50)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```


Training and Validation Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0
5
10
Training Accuracy
Validation Accuracy
Training and Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
Training Loss
Validation Loss