```
!pip install -q kaggle
```

```
from google.colab import files
files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle (2).json to kaggle (2).json
{'kaggle (2).json': b'{"username":"crazyburg","key":"a2b356a8d172b567d86528de0d43e8a6"}'}
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

```
!kaggle datasets download -d rhammell/ships-in-satellite-imagery
```

```
Dataset URL: https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery
License(s): CC-BY-SA-4.0
Downloading ships-in-satellite-imagery.zip to /content
 96% 178M/185M [00:01<00:00, 125MB/s]
100% 185M/185M [00:01<00:00, 130MB/s]
```

```
!unzip ships-in-satellite-imagery.zip
```

```
  inflating: shipsnet/shipsnet/0__20161218_180844_0e26__-122.43570635251734_37.81780153646771.png
```

```
  inflating: shipsnet/shipsnet/0__20161218_180844_0e26__-122.43818561434492_37.88189597797097.png
  inflating: shipsnet/shipsnet/0__20161218_180844_0e26__-122.44292051246497_37.86017914204665.png
  inflating: shipsnet/shipsnet/0__20161218_180844_0e26__-122.4796958662502_37.829549813389136.png
  inflating: shipsnet/shipsnet/0__20161218_180844_0e26__-122.47984174489888_37.83790556670464.png
  inflating: shipsnet/shipsnet/0__20161218_180844_0e26__-122.48871932039954_37.84589395485927.png
```

```
!ls
```

```
'kaggle (2).json'   scenes                          shipsnet
 sample_data        ships-in-satellite-imagery.zip   shipsnet.json
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
```

```
!ls /content
```

```
'kaggle (2).json'   scenes                          shipsnet
 sample_data        ships-in-satellite-imagery.zip   shipsnet.json
```

```
!unzip /content/ships-in-satellite-imagery.zip -d /content/ships-in-satellite-imagery
```

```
inflating: /content/ships-in-satellite-imagery/shipsnet/shipsnet/0__20161102_180658_0e26__-122.23122736968294_37.75239959943399.png
inflating: /content/ships-in-satellite-imagery/shipsnet/shipsnet/0__20161102_180658_0e26__-122.28126318568829_37.769020038003475.pn
inflating: /content/ships-in-satellite-imagery/shipsnet/shipsnet/0__20161102_180658_0e26__-122.31527680074582_37.79277375149748.png
inflating: /content/ships-in-satellite-imagery/shipsnet/shipsnet/0__20161102_180658_0e26__-122.32843644745485_37.73923054907409.png
inflating: /content/ships-in-satellite-imagery/shipsnet/shipsnet/0__20161102_180658_0e26__-122.34040514248915_37.748761931300486.pn
```

```
!ls /content/ships-in-satellite-imagery
```

```
scenes  shipsnet  shipsnet.json
```

```
dataset_path = "/content/ships-in-satellite-imagery/shipsnet"
```

```
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Path to the dataset (update this if necessary)
dataset_path = "/content/ships-in-satellite-imagery"

# Verify the dataset path
if not os.path.exists(dataset_path):
    # Debug: Print available directories
    print("Available directories in /content:")
    print(os.listdir("/content"))
    raise FileNotFoundError(f"The dataset path '{dataset_path}' does not exist. Please check the path.")

# Debug: Print contents of the dataset directory
print("Contents of the dataset directory:")
print(os.listdir(dataset_path))

# Load images and labels
images = []
labels = []

# Recursively search for .png files
for root, dirs, files in os.walk(dataset_path):
    for filename in files:
        if filename.endswith(".png"):  # Ensure only PNG files are processed
            # Load image
            img_path = os.path.join(root, filename)
            img = load_img(img_path, target_size=(80, 80))  # Resize images to 80x80
            img = img_to_array(img) / 255.0  # Normalize pixel values to [0, 1]
            images.append(img)

            # Extract label from filename (e.g., "ship_1.png" -> "ship")
            label = filename.split("_")[0]  # Adjust based on filename structure
            labels.append(label)

# Check if any images were loaded
if len(images) == 0:
    raise ValueError("No images were loaded. Check the dataset path and file structure.")

# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Encode labels to integers
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels_encoded, test_size=0.2, random_state=42)

print(f"Loaded {len(images)} images and {len(labels)} labels.")
print(f"Unique labels: {np.unique(labels)}")
print(f"Training data shape: {X_train.shape}, Testing data shape: {X_test.shape}")
```

```
Contents of the dataset directory:
['shipsnet.json', 'shipsnet', 'scenes']
Loaded 4008 images and 4008 labels.
Unique labels: ['0' '1' 'lb' 'sfbay']
Training data shape: (3206, 80, 80, 3), Testing data shape: (802, 80, 80, 3)
```

```
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split


dataset_path = "/content/ships-in-satellite-imagery"
if not os.path.exists(dataset_path):
    raise FileNotFoundError(f"The dataset path '{dataset_path}' does not exist. Please check the path.")


images = []
labels = []

for filename in os.listdir(dataset_path):
    if filename.endswith(".png"):  # Ensure only PNG files are processed
        # Load image
        img_path = os.path.join(dataset_path, filename)
        img = load_img(img_path, target_size=(80, 80))  # Resize images to 80x80
        img = img_to_array(img) / 255.0  # Normalize pixel values to [0, 1]
        images.append(img)

        # Extract label from filename (e.g., "ship_1.png" -> "ship")
        label = filename.split("_")[0]  # Adjust based on filename structure
        labels.append(label)


images = np.array(images)
labels = np.array(labels)


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

# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(80, 80, 3)),  # First convolutional layer
    MaxPooling2D((2, 2)),  # First max-pooling layer
    Conv2D(64, (3, 3), activation='relu'),  # Second convolutional layer
    MaxPooling2D((2, 2)),  # Second max-pooling layer
    Conv2D(128, (3, 3), activation='relu'),  # Third convolutional layer
    MaxPooling2D((2, 2)),  # Third max-pooling layer
    Flatten(),  # Flatten the output for dense layers
    Dense(128, activation='relu'),  # Fully connected layer
    Dropout(0.5),  # Dropout to prevent overfitting
    Dense(len(label_encoder.classes_), activation='softmax')  # Output layer
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print model summary
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 78, 78, 32)	896
max_pooling2d (MaxPooling2D)	(None, 39, 39, 32)	0
conv2d_1 (Conv2D)	(None, 37, 37, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 18, 18, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1,048,704
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516

```
Total params: 1,142,468 (4.36 MB)
Trainable params: 1,142,468 (4.36 MB)
```

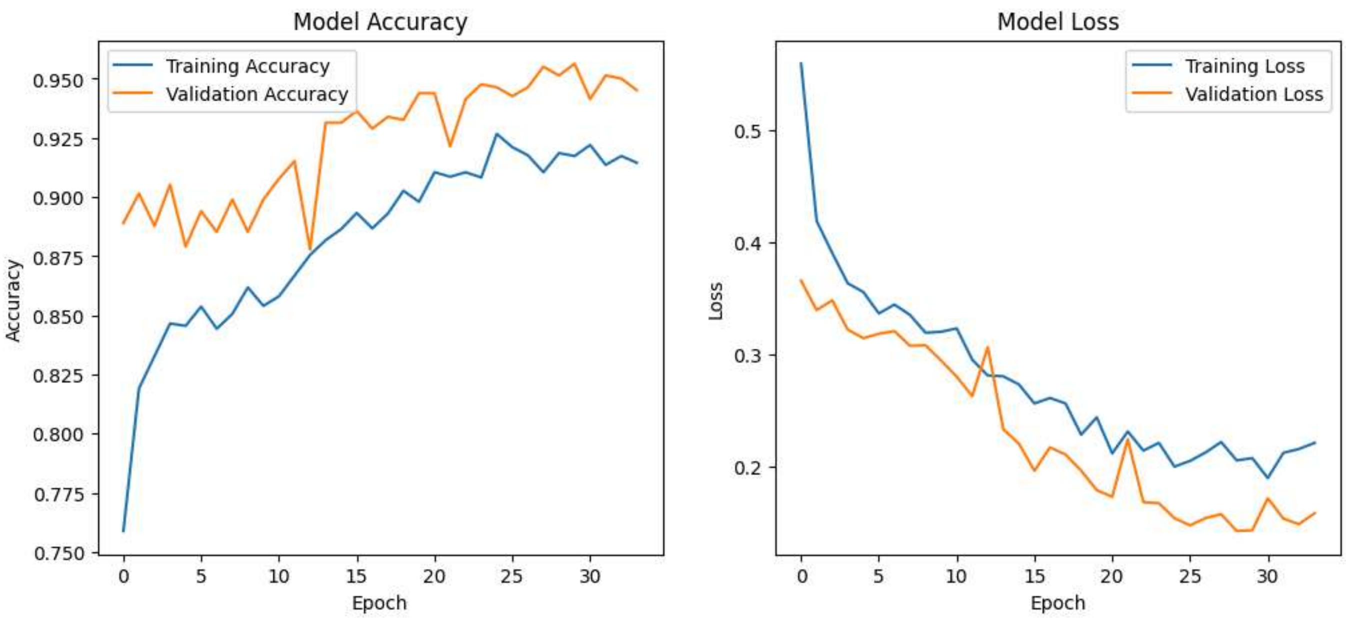
```
from tensorflow.keras.callbacks import EarlyStopping

# Data Augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Fit the data generator to the training data
datagen.fit(X_train)

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=50,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping]
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cla
  self._warn_if_super_not_called()
Epoch 1/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 63s 596ms/step - accuracy: 0.7100 - loss: 0.6549 - val_accuracy: 0.8890 - val_loss: 0.3661
Epoch 2/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 57s 556ms/step - accuracy: 0.8127 - loss: 0.4288 - val_accuracy: 0.9015 - val_loss: 0.3398
Epoch 3/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 60s 591ms/step - accuracy: 0.8301 - loss: 0.4136 - val_accuracy: 0.8878 - val_loss: 0.3486
Epoch 4/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 60s 599ms/step - accuracy: 0.8408 - loss: 0.3879 - val_accuracy: 0.9052 - val_loss: 0.3225
Epoch 5/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 75s 527ms/step - accuracy: 0.8452 - loss: 0.3505 - val_accuracy: 0.8791 - val_loss: 0.3148
Epoch 6/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 59s 588ms/step - accuracy: 0.8443 - loss: 0.3523 - val_accuracy: 0.8940 - val_loss: 0.3188
Epoch 7/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 61s 605ms/step - accuracy: 0.8412 - loss: 0.3459 - val_accuracy: 0.8853 - val_loss: 0.3211
Epoch 8/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 55s 543ms/step - accuracy: 0.8463 - loss: 0.3343 - val_accuracy: 0.8990 - val_loss: 0.3080
Epoch 9/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 81s 538ms/step - accuracy: 0.8575 - loss: 0.3300 - val_accuracy: 0.8853 - val_loss: 0.3085
Epoch 10/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 58s 578ms/step - accuracy: 0.8574 - loss: 0.3141 - val_accuracy: 0.8990 - val_loss: 0.2949
```

```
Epoch 11/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 53s 524ms/step - accuracy: 0.8610 - loss: 0.3108 - val_accuracy: 0.9077 - val_loss: 0.2806
Epoch 12/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 59s 585ms/step - accuracy: 0.8614 - loss: 0.2963 - val_accuracy: 0.9152 - val_loss: 0.2631
Epoch 13/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 53s 529ms/step - accuracy: 0.8816 - loss: 0.2730 - val_accuracy: 0.8778 - val_loss: 0.3068
Epoch 14/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 62s 612ms/step - accuracy: 0.8861 - loss: 0.2747 - val_accuracy: 0.9314 - val_loss: 0.2335
Epoch 15/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 57s 566ms/step - accuracy: 0.8913 - loss: 0.2592 - val_accuracy: 0.9314 - val_loss: 0.2207
Epoch 16/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 57s 564ms/step - accuracy: 0.8921 - loss: 0.2490 - val_accuracy: 0.9364 - val_loss: 0.1966
Epoch 17/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 62s 619ms/step - accuracy: 0.8986 - loss: 0.2350 - val_accuracy: 0.9289 - val_loss: 0.2175
Epoch 18/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 78s 577ms/step - accuracy: 0.8833 - loss: 0.2706 - val_accuracy: 0.9339 - val_loss: 0.2110
Epoch 19/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 55s 549ms/step - accuracy: 0.8962 - loss: 0.2408 - val_accuracy: 0.9327 - val_loss: 0.1968
Epoch 20/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 57s 557ms/step - accuracy: 0.8983 - loss: 0.2476 - val_accuracy: 0.9439 - val_loss: 0.1793
Epoch 21/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 57s 563ms/step - accuracy: 0.9123 - loss: 0.2022 - val_accuracy: 0.9439 - val_loss: 0.1733
Epoch 22/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 56s 551ms/step - accuracy: 0.9016 - loss: 0.2376 - val_accuracy: 0.9214 - val_loss: 0.2247
Epoch 23/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 56s 558ms/step - accuracy: 0.9090 - loss: 0.2281 - val_accuracy: 0.9414 - val_loss: 0.1686
Epoch 24/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 56s 560ms/step - accuracy: 0.9040 - loss: 0.2263 - val_accuracy: 0.9476 - val_loss: 0.1676
Epoch 25/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 57s 566ms/step - accuracy: 0.9264 - loss: 0.2086 - val_accuracy: 0.9464 - val_loss: 0.1543
Epoch 26/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 55s 545ms/step - accuracy: 0.9128 - loss: 0.2122 - val_accuracy: 0.9426 - val_loss: 0.1479
Epoch 27/50
101/101 ━━━━━━━━━━━━━━━━━━━━ 56s 556ms/step - accuracy: 0.9081 - loss: 0.2285 - val_accuracy: 0.9464 - val_loss: 0.1544
```

```python
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

# Plot training and validation accuracy and loss
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 89ms/step - accuracy: 0.9565 - loss: 0.1425
Test Accuracy: 95.14%
```




```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import GlobalAveragePooling2D

# Load pre-trained VGG16 model (without the top layers)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(80, 80, 3))

# Freeze the base model layers
base_model.trainable = False

# Add custom layers on top of the base model
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(label_encoder.classes_), activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=20,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping]
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 397s 4s/step - accuracy: 0.7243 - loss: 0.6724 - val_accuracy: 0.9152 - val_loss: 0.2355
Epoch 2/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 393s 4s/step - accuracy: 0.8917 - loss: 0.2642 - val_accuracy: 0.9352 - val_loss: 0.1930
Epoch 3/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 391s 4s/step - accuracy: 0.9156 - loss: 0.2034 - val_accuracy: 0.9426 - val_loss: 0.1740
Epoch 4/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 394s 4s/step - accuracy: 0.9190 - loss: 0.2010 - val_accuracy: 0.9352 - val_loss: 0.1874
Epoch 5/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 394s 4s/step - accuracy: 0.9256 - loss: 0.1849 - val_accuracy: 0.9401 - val_loss: 0.1757
Epoch 6/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 392s 4s/step - accuracy: 0.9407 - loss: 0.1592 - val_accuracy: 0.9514 - val_loss: 0.1464
Epoch 7/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 386s 4s/step - accuracy: 0.9389 - loss: 0.1502 - val_accuracy: 0.9564 - val_loss: 0.1387
Epoch 8/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 395s 4s/step - accuracy: 0.9320 - loss: 0.1689 - val_accuracy: 0.9489 - val_loss: 0.1484
Epoch 9/20
101/101 ━━━━━━━━━━━━━━━━━━━━ 386s 4s/step - accuracy: 0.9414 - loss: 0.1560 - val_accuracy: 0.9551 - val_loss: 0.1465
Epoch 10/20
```

```
101/101 ━━━━━━━━━━━━━━━━━━━━ 394s 4s/step - accuracy: 0.9355 - loss: 0.1532 - val_accuracy: 0.9626 - val_loss: 0.1168
```