```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
mcagriaksoy_trees_in_satellite_imagery_path = kagglehub.dataset_download('mcagriaksoy/trees-in-satellite-imagery')

print('Data source import complete.')
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Sav
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
import os
import random
import matplotlib.pyplot as plt
from matplotlib.image import imread

# Define the paths to the folders
tree_folder = '/kaggle/input/trees-in-satellite-imagery/Trees in Satellite Imagery/Trees'
notree_folder = '/kaggle/input/trees-in-satellite-imagery/Trees in Satellite Imagery/NoTrees'

# Function to display random images from a folder
def display_random_images(folder, class_name, num_images=10):
    # Get list of all images in the folder
    images = os.listdir(folder)

    # Randomly select 'num_images' images
    random_images = random.sample(images, num_images)

    # Plot the images
    plt.figure(figsize=(15, 5))
    for i, image_name in enumerate(random_images):
        plt.subplot(2, 5, i+1)
        img = imread(os.path.join(folder, image_name))
        plt.imshow(img)
        plt.title(f'{class_name} {i+1}')
        plt.axis('off')
    plt.show()

# Display 10 random images from the "tree" folder
display_random_images(tree_folder, 'Tree')

# Display 10 random images from the "notree" folder
display_random_images(notree_folder, 'No Tree')
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

```
tree_folder = '/kaggle/input/trees-in-satellite-imagery/Trees in Satellite Imagery/Trees'
notree_folder = '/kaggle/input/trees-in-satellite-imagery/Trees in Satellite Imagery/NoTrees'
```

```
# Load image paths and labels
tree_images = [os.path.join(tree_folder, img) for img in os.listdir(tree_folder)]
notree_images = [os.path.join(notree_folder, img) for img in os.listdir(notree_folder)]
```

```
# Create labels (1 for tree, 0 for notree)
X = tree_images + notree_images
```

```
y = [1] * len(tree_images) + [0] * len(notree_images)
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# X_train and X_test are lists of file paths to the images.
# y_train and y_test are lists of labels corresponding to the images.
# E.G
# X_train = ["image1.jpg", "image2.jpg", "image3.jpg"]
# y_train = [1, 0, 1] (1 for "tree", 0 for "notree")
```

```
# Function to load and preprocess images
def load_and_preprocess_image(path, img_size=(64, 64)):
    # Read the image file
    img = tf.io.read_file(path)
    # Decode the JPEG image
    img = tf.image.decode_jpeg(img, channels=3)
    # Resize the image
    img = tf.image.resize(img, img_size)
    # Normalize pixel values to [0, 1]
    img = img / 255.0
    return img
```

```
# Prepare datasets
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_dataset = train_dataset.map(lambda x, y: (load_and_preprocess_image(x), y))
train_dataset = train_dataset.batch(32).prefetch(tf.data.AUTOTUNE)

test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_dataset = test_dataset.map(lambda x, y: (load_and_preprocess_image(x), y))
test_dataset = test_dataset.batch(32).prefetch(tf.data.AUTOTUNE)
```

```
# Verify the dataset
for images, labels in train_dataset.take(1):
    print("Image shape:", images.shape)  # Should print (32, 64, 64, 3)
    print("Labels:", labels.numpy())    # Should print an array of 0s and 1s
```

```
# define function to Visualize a few images
def plot_images(images, labels):
    plt.figure(figsize=(10, 10))
    for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])
        plt.title(f"Label: {labels[i]}")
        plt.axis('off')
    plt.show()
```

```
# Visualize images from the training dataset
print("Training Dataset Images:")
for images, labels in train_dataset.take(1):
    plot_images(images.numpy(), labels.numpy())
```

```
# Visualize images from the testing dataset
print("Testing Dataset Images:")
for images, labels in test_dataset.take(1):
    plot_images(images.numpy(), labels.numpy())
```

```
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),  # Regularization to prevent overfitting
    Dense(1, activation='sigmoid')  # Binary classification
])
```

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

```
# Model summary
model.summary()
```

```
# Train the model
history = model.fit(train_dataset,
                    epochs=50,
                    validation_data=test_dataset)

#Training accuracy is the percentage of correctly classified samples in the training dataset during training.
#Training Accuracy = (number of coreect predictions/Total number of samples) * 100

#Training loss Measures how well the model is minimizing the error on the training data.
#A decreasing training loss indicates that the model is learning.
#If the training loss is very low but the validation loss is high, it might indicate overfitting.


#Validation Accuracy Measures how well the model generalizes to unseen data (the validation dataset).
#Validation Accuracy = (number of coreect predictions/Total number of samples) * 100
#High validation accuracy indicates that the model is performing well on data it has never seen before.
#If validation accuracy is much lower than training accuracy, it might indicate overfitting.

#Validation Loss Measures how well the model is minimizing the error on unseen data.
#A decreasing validation loss indicates that the model is generalizing well.
#If the validation loss starts increasing while the training loss continues to decrease, it indicates overfitting.

# Good Performance:
# Training accuracy and validation accuracy are both high and close to each other.
# Training loss and validation loss are both low and decreasing.

# Overfitting:
# Training accuracy is very high, but validation accuracy is significantly lower.
# Training loss is very low, but validation loss is high or increasing.

# Underfitting:
# Both training accuracy and validation accuracy are low.
# Both training loss and validation loss are high.

# Ideal Scenario:
# Training accuracy and validation accuracy are both high and close to each other.
# Training loss and validation loss are both low and stable.
```

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Start coding or generate with AI.
```

Dataset Preparation:

Mix all images (5,200 "tree" + 5,200 "notree" = 10,400 images).

Split into:

Training Set (80% = 8,320 images).

Testing Set (20% = 2,080 images).

Training:

Apply data augmentation to the training set.

Train the model on the training set.

Use the validation set (if available) to monitor performance and tune hyperparameters.

Deployment:

If the model performs well on the testing set, it can be deployed to make predictions on new, unseen data.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define data augmentation for the training set
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,  # Normalize pixel values to [0, 1]
    rotation_range=40,   # Randomly rotate images by up to 40 degrees
    width_shift_range=0.2,  # Randomly shift images horizontally by up to 20%
    height_shift_range=0.2,  # Randomly shift images vertically by up to 20%
    shear_range=0.2,  # Apply shear transformations
    zoom_range=0.2,  # Randomly zoom in/out by up to 20%
    horizontal_flip=True,  # Randomly flip images horizontally
    fill_mode='nearest'  # Fill in missing pixels after transformations
)

# Load and augment the training data
train_generator = train_datagen.flow_from_directory(
    '/kaggle/input/trees-in-satellite-imagery/Trees in Satellite Imagery',  # Directory containing "Trees" and "NoTrees" folders
    target_size=(64, 64),  # Resize images to 64x64
    batch_size=32,  # Number of images per batch
    class_mode='binary'  # Binary classification
)

# Prepare the testing dataset (no augmentation)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_generator = test_datagen.flow_from_directory(
    '/kaggle/input/trees-in-satellite-imagery/Trees in Satellite Imagery',  # Directory containing "Trees" and "NoTrees" folders
    target_size=(64, 64),  # Resize images to 64x64
    batch_size=32,  # Number of images per batch
    class_mode='binary'  # Binary classification
)
```

```
import matplotlib.pyplot as plt

# Function to display sample images from a generator
def display_sample_images(generator, title, num_images=10):
    # Get a batch of images and labels from the generator
    images, labels = next(generator)

    # Plot the images
    plt.figure(figsize=(15, 5))
    plt.suptitle(title, fontsize=16)
    for i in range(num_images):
        plt.subplot(2, 5, i + 1)
        plt.imshow(images[i])
        plt.title(f"Label: {int(labels[i])}")
        plt.axis('off')
    plt.show()
```

```
# Display 10 augmented images from the training dataset
print("Augmented Training Images:")
display_sample_images(train_generator, "Augmented Training Images")
```

```
# Display 10 non-augmented images from the testing dataset
print("Non-Augmented Testing Images:")
display_sample_images(test_generator, "Non-Augmented Testing Images")
```

```
# Train the model with augmented data
history = model.fit(train_generator,
                    epochs=50,
                    validation_data=test_generator)
```

```
#Confusion Matrix

from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Evaluate the model on the test set
y_true = []
y_pred = []

for images, labels in test_dataset:
    predictions = model.predict(images)
    predicted_labels = (predictions > 0.5).astype(int).flatten()
    y_pred.extend(predicted_labels)
    y_true.extend(labels.numpy())

# Generate confusion matrix
cm = confusion_matrix(y_true, y_pred)
```

```
# Plot the confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Tree', 'Tree'], yticklabels=['No Tree', 'Tree'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
#Model Performance Metrics (Accuracy, Precision, Recall, F1 Score)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Compute evaluation metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

# Print evaluation metrics
print(f"Model Accuracy  : {accuracy:.4f}")
print(f"Model Precision : {precision:.4f}")
print(f"Model Recall    : {recall:.4f}")
print(f"Model F1 Score  : {f1:.4f}")
```

```
#ROC Curve

from sklearn.metrics import roc_curve, auc

# Generate ROC Curve
y_probs = []  # Probabilities from the model
for images, _ in test_dataset:
    probs = model.predict(images).flatten()
    y_probs.extend(probs)

fpr, tpr, _ = roc_curve(y_true, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='darkorange', label=f'ROC curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

```
# Plot training and validation accuracy - Model Accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

```
# Plot training and validation loss - Model Loss
# Plot how training and validation loss changed over the epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid(True)
plt.show()
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Define data augmentation for the training set
train_datagen = ImageDataGenerator(
```

```
    rescale=1.0/255.0,  # Normalize pixel values to [0, 1]
    rotation_range=40,   # Randomly rotate images by up to 40 degrees
    width_shift_range=0.2,  # Randomly shift images horizontally by up to 20%
    height_shift_range=0.2,  # Randomly shift images vertically by up to 20%
    shear_range=0.2,  # Apply shear transformations
    zoom_range=0.2,  # Randomly zoom in/out by up to 20%
    horizontal_flip=True,  # Randomly flip images horizontally
    fill_mode='nearest'  # Fill in missing pixels after transformations
)

# Load and augment the training data
train_generator = train_datagen.flow_from_directory(
    '/kaggle/input/trees-in-satellite-imagery/Trees in Satellite Imagery',  # Directory containing "Trees" and "NoTrees" folders
    target_size=(64, 64),  # Resize images to 64x64
    batch_size=32,  # Number of images per batch
    class_mode='binary'  # Binary classification
)

# Function to display augmented vs non-augmented images
def display_augmented_vs_original(train_generator, num_images=10):
    # Fetch a batch of augmented images and labels
    augmented_images, labels = next(train_generator)

    # Fetch the same batch of non-augmented images
    non_augmented_images = train_generator[0][0]  # First batch, images only

    # Plot the images
    plt.figure(figsize=(20, 8))
    plt.suptitle("Augmented vs Non-Augmented Images", fontsize=16)

    for i in range(num_images):
        # Display non-augmented image
        plt.subplot(2, num_images, i + 1)
        plt.imshow(non_augmented_images[i])
        plt.title(f"Original\nLabel: {int(labels[i])}")
        plt.axis('off')

        # Display augmented image
        plt.subplot(2, num_images, num_images + i + 1)
        plt.imshow(augmented_images[i])
        plt.title(f"Augmented\nLabel: {int(labels[i])}")
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Display 10 augmented vs non-augmented images
display_augmented_vs_original(train_generator)
```

```
# Save the model as an HDF5 file
model.save('deforestation_model.h5')
```

```
import joblib

# Save the model
joblib.dump(model, 'deforestation_model.joblib')
```

```
import random
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
```

```
# Function to load and preprocess a single image
def load_and_preprocess_single_image(path, img_size=(64, 64)):
    img = tf.io.read_file(path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, img_size)
    img = img / 255.0
    return img
```

```
# Get a list of all test images
test_images = X_test
```

```
# Randomly select 10 images from the test dataset
random_test_images = random.sample(test_images, 10)
```

```

```

```
# Display the selected images and make predictions
plt.figure(figsize=(20, 10))
for i, img_path in enumerate(random_test_images):
    # Load and preprocess the image
    img = load_and_preprocess_single_image(img_path)

    # Display the image
    plt.subplot(2, 5, i + 1)
    plt.imshow(img)

    # Make a prediction
    img_array = tf.expand_dims(img, 0)  # Add batch dimension
    prediction = model.predict(img_array)
    predicted_label = "Forest" if prediction > 0.5 else "Not Forest"

    # Get the true label
    true_label = "Forest" if y_test[X_test.index(img_path)] == 1 else "Not Forest"

    # Set the title with prediction and true label
    plt.title(f"Pred: {predicted_label}\nTrue: {true_label}")
    plt.axis('off')
```

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