```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from google.colab import drive
drive.mount('/content/drive')
# Define paths
mountain_dir = '/content/drive/MyDrive/mountains'
beach_dir = '/content/drive/MyDrive/beaches'
# Load and preprocess the images
import os
from tensorflow import keras
def load_images(image_dir):
    images = []
    labels = []
    for img_name in os.listdir(image_dir):
        img_path = os.path.join(image_dir, img_name)
        img = keras.preprocessing.image.load_img(img_path, target_size=(128, 128))
        img_array = keras.preprocessing.image.img_to_array(img)
        images.append(img_array)
        # Assuming mountain is 0 and beach is 1
        labels.append(0 if 'mountain' in img_name else 1)
    return np.array(images), np.array(labels)
# Load images
mountain_images, mountain_labels = load_images(mountain_dir)
beach_images, beach_labels = load_images(beach_dir)
# Combine and split the dataset
X = np.concatenate((mountain_images, beach_images), axis=0)
y = np.concatenate((mountain_labels, beach_labels), axis=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Normalize the images
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
# 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)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
# Define the model
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(2, activation='softmax')
])
 #Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=10, validation_data=(X_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
```

```
# Load and preprocess the image
img_path = '/content/drive/MyDrive/mountains/mountain 2.jpg'
# Get input shape of the first layer
input_shape = model.input_shape
print(f"Expected input shape: {input_shape}")
# Resize image to match expected input shape
img = image.load_img(img_path, target_size=input_shape[1:3])  # Resize to (width, height)
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Expand dimensions to match the model's input shape
img_array /= 255.0  # Rescale the image (same as during training)
# Predict the class of the image
prediction = model.predict(img_array)
predicted_class = 1 if prediction[0][0] > 0.5 else 0
confidence_score = prediction[0][0]
# Define the actual class (you need to know this)
actual_class = 1  # Replace with the actual class of the image
# Display the actual and predicted class
print(f"Actual Class: {actual_class}")
print(f"Predicted Class: {predicted_class}")
print(f"Confidence Score: {confidence_score:.4f}")
# Optional: Map the class indices to human-readable class names if available
class_names = {0: 'Class 0', 1: 'Class 1'}
print(f"Actual Class Name: {class_names[actual_class]}")
print(f"Predicted Class Name: {class_names[predicted_class]}")
#Display the image
import matplotlib.pyplot as plt
plt.imshow(img)
plt.title(f"Actual: {class_names[actual_class]}, Predicted: {class_names[predicted_class]}")
plt.axis('off')  # Hide the axis
plt.show()
```