```
#code
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential

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

dataset_path1='/content/drive/MyDrive/train'
dataset_path2='/content/drive/MyDrive/validation'
dataset_path3='/content/drive/MyDrive/test'

batch_size = 16
image_size = (128, 128)

train_datagen = ImageDataGenerator(rescale=1/255)
validation_datagen = ImageDataGenerator(rescale=1/255)

train_generator = train_datagen.flow_from_directory(
    dataset_path1,  # Use dataset_path here
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary'

)

validation_generator = validation_datagen.flow_from_directory(
    dataset_path2,  # Use dataset_path here
    target_size=image_size,
```

```
    batch_size=batch_size,
    class_mode='binary'

)

model = Sequential()

model.add(layers.Conv2D(32, (3, 3), input_shape=(128,128, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))



model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.25))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()

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

```
history = model.fit(train_generator,
                    steps_per_epoch = 50,
                    epochs=100,
                    validation_data=validation_generator)

model.save('image_forgery')

from sklearn.metrics import confusion_matrix, classification_report
# Assuming that you have already trained your model and have a validation generator
y_true = validation_generator.classes  # True labels
y_pred = (model.predict(validation_generator) > 0.5).astype(int).flatten()  # Predicted labels

# Compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(confusion)

import numpy as np
import matplotlib.pyplot as plt

# Assuming you have the confusion matrix as a 2D NumPy array
confusion_matrix = np.array([[334, 266], [218, 169]])

# Create a figure and axis for the plot
fig, ax = plt.subplots()

# Display the confusion matrix with boxes and labels
cax = ax.matshow(confusion_matrix, cmap=plt.cm.Greens)

# Add labels for the boxes
```

```
for i in range(confusion_matrix.shape[0]):
    for j in range(confusion_matrix.shape[1]):
        plt.text(j, i, str(confusion_matrix[i, j]), va='center', ha='center')

# Set axis labels
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')

# Add a colorbar to represent values
plt.colorbar(cax)

# Show the plot
plt.show()



import matplotlib.pyplot as plt

# Assuming you have already trained the model and have the history object
# ...

# Access the training and validation loss and accuracy from the history object
training_loss = history.history['loss']
validation_loss = history.history['val_loss']
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']

# Create plots for loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
```

```
plt.title('Loss')
plt.xlabel('Epoch')
plt.legend()

# Create plots for accuracy
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.tight_layout()
plt.show()



from tensorflow.keras.preprocessing import image
import numpy as np

# Load and preprocess the test image
test_image_path = '/content/drive/MyDrive/test/test.au/Au_art_30474.jpg'  # Provide the path to your test image
test_image = image.load_img(test_image_path, target_size=image_size)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
test_image = test_image / 255.0  # Normalize the pixel values

# Use the model for prediction
prediction = model.predict(test_image)

if prediction > 0.5:
```

```
    print("The image is manipulated.")
else:
    print("The image is authentic.")



from tensorflow.keras.preprocessing import image
import numpy as np

# Load and preprocess the test image
test_image_path = '/content/drive/MyDrive/test/test.tp/Tp_S_NNN_S_N_txt00073_txt00073_01286.tif'  # Provide the path to your test image
test_image = image.load_img(test_image_path, target_size=image_size)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
test_image = test_image / 255.0  # Normalize the pixel values

plt.imshow(test_image[0])
plt.title('Test Image')
plt.show()

# Use the model for prediction
prediction = model.predict(test_image)

if prediction > 0.5:
    print("The image is Tampered.")
else:
    print("The image is authentic.")
```