Code:

```
# Here Imported libraries
import os
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout
from tensorflow.keras.applications import ResNet50
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

# Here imported the dataset
cracked_dir =
os.path.join("/root/.cache/kagglehub/datasets/arunrk7/surface-crack-detection/versions/
1", 'Positive')
non_cracked_dir =
os.path.join("/root/.cache/kagglehub/datasets/arunrk7/surface-crack-detection/versions/
1", 'Negative')

# Here performing the preprocessing
def load_images_from_folder(folder, label, limit=None):
    images = []
    labels = []
    for i, filename in enumerate(os.listdir(folder)):
        if limit and i >= limit:
            break
        img_path = os.path.join(folder, filename)
        img = tf.keras.utils.load_img(img_path, target_size=(34, 34))  # Reduced to 34x34
        img_array = tf.keras.utils.img_to_array(img) / 255.0  # Normalize pixel values
        images.append(img_array)
        labels.append(label)
    return images, labels
#Here  loaded limited number of images

limit_per_class = 1000  # Adjust this limit as needed
```

```
cracked_images, cracked_labels = load_images_from_folder(cracked_dir, 1,
limit=limit_per_class)
non_cracked_images, non_cracked_labels =
load_images_from_folder(non_cracked_dir, 0, limit=limit_per_class)

# Here Combining the datasets
images = np.array(cracked_images + non_cracked_images)
labels = np.array(cracked_labels + non_cracked_labels)

# Here Converting the  labels to categorical
labels = to_categorical(labels, num_classes=2)

#Here splitting the dataset for training , testing.
X_train, X_temp, y_train, y_temp = train_test_split(images, labels, test_size=0.3,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)

# Data augmentation
datagen = ImageDataGenerator(rotation_range=15, width_shift_range=0.1,
height_shift_range=0.1, horizontal_flip=True)
datagen.fit(X_train)

# Here performing the ResNet50-based model
def create_resnet50():
    # Load the ResNet50 model without the top classification layer
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(34,
34, 3))

    # Here Freezing the  all layers of the base model
    base_model.trainable = False

    # Adding customer layers on the top of the model
    inputs = base_model.input
    x = Flatten()(base_model.output)
    x = Dense(128, activation='relu')(x)  # Reduced dense layer size
    x = Dropout(0.5)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.5)(x)
```

```
    # here is the Output Layer
    outputs = Dense(2, activation='softmax')(x)

    model = Model(inputs, outputs)
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

# here Creating the model
model = create_resnet50()
model.summary()

# Training
history = model.fit(datagen.flow(X_train, y_train, batch_size=16),  # Smaller batch size
                    validation_data=(X_val, y_val),
                    epochs=10,  # Training for fewer epochs initially
                    verbose=1)

# Evaluating the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")

# here is the Classification metrics and confusion matrix
y_pred = np.argmax(model.predict(X_test), axis=1)
y_true = np.argmax(y_test, axis=1)

print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=["Non-Cracked", "Cracked"]))

# Confusion Matrix Visualization
conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Non-Cracked", "Cracked"], yticklabels=["Non-Cracked", "Cracked"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

# here is the Training and validation metrics visualization
```

```
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```