```
!pip install keras_preprocessing
```

```
Collecting keras_preprocessing
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 1.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from keras_preprocessing) (1.23.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from keras_preprocessing) (1.16.0)
Installing collected packages: keras_preprocessing
Successfully installed keras_preprocessing-1.1.2
```

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

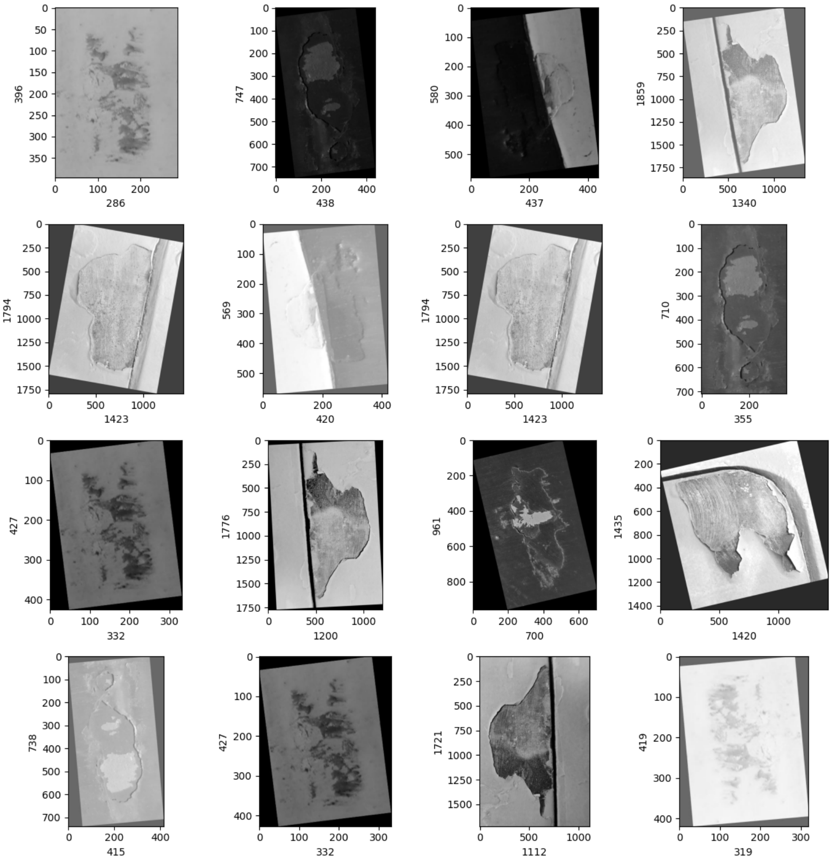
```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.image import imread
import cv2
import random
import os
from os import listdir
from PIL import Image
from sklearn.preprocessing import label_binarize, LabelBinarizer
from keras.preprocessing import image
from tensorflow.keras.optimizers import Adam
from keras_preprocessing.image import img_to_array
from keras.utils import to_categorical
# Your code using to_categorical here
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dropout, Dense
from sklearn.model_selection import train_test_split
from keras.models import model_from_json
```

```
!ls "/content/drive/MyDrive/AIRCRAFT_DATASET"
```

```
corrosion  dents  paint_irregularitie  scratches
```

```
plt.figure(figsize=(12,12))
path = "/content/drive/MyDrive/AIRCRAFT_DATASET/corrosion"
for i in range(1,17):
          plt.subplot(4,4,i)
          plt.tight_layout()
          rand_img = imread(path +'/'+ random.choice(sorted(os.listdir(path))))
          plt.imshow(rand_img)
          plt.xlabel(rand_img.shape[1], fontsize = 10)#width of image
          plt.ylabel(rand_img.shape[0], fontsize = 10)#height of image
```




```
#Converting Images to array
def convert_image_to_array(image_dir):
  try:
    image = cv2.imread(image_dir)
    if image is not None :
      image = cv2.resize(image, (256,256))
      #image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
      return img_to_array(image)
    else:
      return np.array([])
  except Exception as e:
    print(f"Error : {e}")
    return None
```

```
dir = "/content/drive/MyDrive/AIRCRAFT_DATASET"
root_dir = listdir(dir)
image_list, label_list = [], []
all_labels = ['corrosion', 'dents', 'paint_irregularitie' , 'scratches']
binary_labels= [0,1,2]
temp = 1



for directory in root_dir:
  aircraft_image_list = listdir(f"{dir}/{directory}")
  temp = 1
  for files in aircraft_image_list:
    image_path = f"{dir}/{directory}/{files}"
    image_list.append(convert_image_to_array(image_path))
    label_list.append(binary_labels[temp])
```

```
# Visualize the number of classes count
label_counts = pd.DataFrame(label_list).value_counts()
label_counts.head()
```

```
1    82
dtype: int64
```

```
#Next we will observe the shape of the image.
image_list[0].shape
```

```
(256, 256, 3)
```

```
image_list=np.array(image_list)
image_list.shape
```

```
(82, 256, 256, 3)
```

```
label_list = np.array(label_list)
label_list.shape
```

```
(82,)
```

```
import numpy as np
from sklearn.model_selection import train_test_split

# Creating dummy data (replace this with your actual data)
num_samples = 100
# Perform train-test split
x_train, x_test, y_train, y_test = train_test_split(image_list, label_list, test_size=0.2, random_state=10)

# Check the shapes of the resulting arrays
print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (65, 256, 256, 3)
x_test shape: (17, 256, 256, 3)
y_train shape: (65,)
y_test shape: (17,)
```

```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
```

```
from sklearn.model_selection import train_test_split
import numpy as np
x_train, x_test, y_train, y_test = train_test_split(np.array(image_list), np.array(label_list), test_size=0.2, random_state=10)
x_train = np.array(x_train, dtype=np.float16) / 255.0  # Normalize pixel values
x_test = np.array(x_test, dtype=np.float16) / 255.0

x_train = x_train.reshape(-1, 256, 256, 3)

import tensorflow as tf

# Assuming you have integer labels in y_train
num_classes = 3  # Change this to the actual number of classes

# Convert integer labels to one-hot encoded vectors
y_train = tf.keras.utils.to_categorical(y_train, num_classes=num_classes)

y_test = to_categorical(y_test)
model = Sequential()
model.add(Conv2D(32, (3, 3), padding="same",input_shape=(256,256,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Conv2D(16, (3, 3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(8, activation="relu"))
model.add(Dense(3, activation="softmax"))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2D  (None, 85, 85, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 85, 85, 16)        4624

 max_pooling2d_1 (MaxPooling  (None, 42, 42, 16)       0
 2D)

 flatten (Flatten)           (None, 28224)             0
```

```
 dense (Dense)               (None, 8)                 225800

 dense_1 (Dense)             (None, 3)                 27

=================================================================
Total params: 231,347
Trainable params: 231,347
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss = 'categorical_crossentropy', optimizer = Adam(0.0001),metrics=['accuracy'])
```

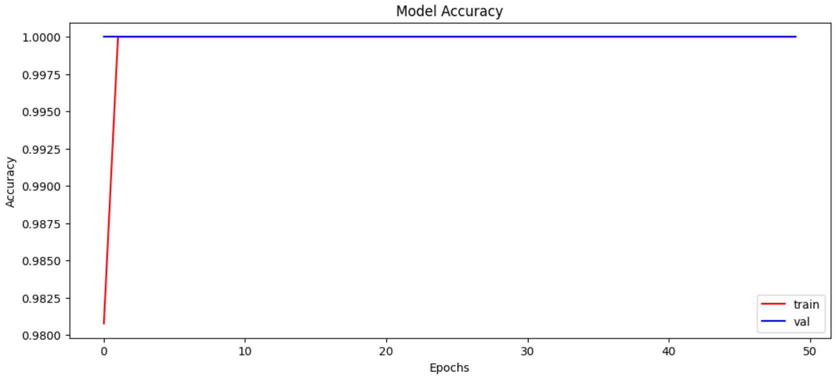
```
#Next we will split the dataset into validation and training data.
# Splitting the training data set into training and validation data sets
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = 0.2)
```

```
epochs = 50
batch_size = 128
history = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs,
validation_data = (x_val, y_val))
```

```
Epoch 1/50
1/1 [==============================] - 7s 7s/step - loss: 0.9269 - accuracy: 0.9808 - val_loss: 0.5325 - val_accuracy: 1.000
Epoch 2/50
1/1 [==============================] - 4s 4s/step - loss: 0.5868 - accuracy: 1.0000 - val_loss: 0.3333 - val_accuracy: 1.000
Epoch 3/50
1/1 [==============================] - 11s 11s/step - loss: 0.3818 - accuracy: 1.0000 - val_loss: 0.2169 - val_accuracy: 1.0
Epoch 4/50
1/1 [==============================] - 5s 5s/step - loss: 0.2544 - accuracy: 1.0000 - val_loss: 0.1482 - val_accuracy: 1.000
Epoch 5/50
1/1 [==============================] - 3s 3s/step - loss: 0.1744 - accuracy: 1.0000 - val_loss: 0.1067 - val_accuracy: 1.000
Epoch 6/50
1/1 [==============================] - 3s 3s/step - loss: 0.1231 - accuracy: 1.0000 - val_loss: 0.0807 - val_accuracy: 1.000
Epoch 7/50
1/1 [==============================] - 5s 5s/step - loss: 0.0895 - accuracy: 1.0000 - val_loss: 0.0638 - val_accuracy: 1.000
Epoch 8/50
1/1 [==============================] - 3s 3s/step - loss: 0.0669 - accuracy: 1.0000 - val_loss: 0.0524 - val_accuracy: 1.000
Epoch 9/50
1/1 [==============================] - 3s 3s/step - loss: 0.0513 - accuracy: 1.0000 - val_loss: 0.0443 - val_accuracy: 1.000
Epoch 10/50
1/1 [==============================] - 3s 3s/step - loss: 0.0403 - accuracy: 1.0000 - val_loss: 0.0384 - val_accuracy: 1.000
Epoch 11/50
1/1 [==============================] - 5s 5s/step - loss: 0.0323 - accuracy: 1.0000 - val_loss: 0.0339 - val_accuracy: 1.000
Epoch 12/50
1/1 [==============================] - 3s 3s/step - loss: 0.0264 - accuracy: 1.0000 - val_loss: 0.0303 - val_accuracy: 1.000
Epoch 13/50
1/1 [==============================] - 3s 3s/step - loss: 0.0219 - accuracy: 1.0000 - val_loss: 0.0274 - val_accuracy: 1.000
Epoch 14/50
1/1 [==============================] - 3s 3s/step - loss: 0.0184 - accuracy: 1.0000 - val_loss: 0.0251 - val_accuracy: 1.000
Epoch 15/50
1/1 [==============================] - 5s 5s/step - loss: 0.0158 - accuracy: 1.0000 - val_loss: 0.0231 - val_accuracy: 1.000
Epoch 16/50
1/1 [==============================] - 5s 5s/step - loss: 0.0136 - accuracy: 1.0000 - val_loss: 0.0214 - val_accuracy: 1.000
Epoch 17/50
1/1 [==============================] - 5s 5s/step - loss: 0.0119 - accuracy: 1.0000 - val_loss: 0.0199 - val_accuracy: 1.000
Epoch 18/50
1/1 [==============================] - 3s 3s/step - loss: 0.0105 - accuracy: 1.0000 - val_loss: 0.0186 - val_accuracy: 1.000
Epoch 19/50
1/1 [==============================] - 5s 5s/step - loss: 0.0094 - accuracy: 1.0000 - val_loss: 0.0175 - val_accuracy: 1.000
Epoch 20/50
1/1 [==============================] - 3s 3s/step - loss: 0.0084 - accuracy: 1.0000 - val_loss: 0.0165 - val_accuracy: 1.000
Epoch 21/50
1/1 [==============================] - 3s 3s/step - loss: 0.0076 - accuracy: 1.0000 - val_loss: 0.0157 - val_accuracy: 1.000
Epoch 22/50
1/1 [==============================] - 3s 3s/step - loss: 0.0069 - accuracy: 1.0000 - val_loss: 0.0149 - val_accuracy: 1.000
Epoch 23/50
1/1 [==============================] - 5s 5s/step - loss: 0.0064 - accuracy: 1.0000 - val_loss: 0.0142 - val_accuracy: 1.000
Epoch 24/50
1/1 [==============================] - 3s 3s/step - loss: 0.0059 - accuracy: 1.0000 - val_loss: 0.0136 - val_accuracy: 1.000
Epoch 25/50
1/1 [==============================] - 3s 3s/step - loss: 0.0054 - accuracy: 1.0000 - val_loss: 0.0130 - val_accuracy: 1.000
Epoch 26/50
1/1 [==============================] - 3s 3s/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 0.0125 - val_accuracy: 1.000
Epoch 27/50
1/1 [==============================] - 5s 5s/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0120 - val_accuracy: 1.000
Epoch 28/50
1/1 [==============================] - 3s 3s/step - loss: 0.0045 - accuracy: 1.0000 - val_loss: 0.0116 - val_accuracy: 1.000
Epoch 29/50
```

```
#Plot the training history
plt.figure(figsize=(12, 5))
plt.plot(history.history['accuracy'], color='r')
plt.plot(history.history['val_accuracy'], color='b')
plt.title('Model Accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'val'])
plt.show()
```




```
y_pred = model.predict(x_test)
```

```
1/1 [==============================] - 1s 1s/step
```

1s completed at 9:35 AM