```python
import os

# Create a directory for Kaggle config
os.makedirs("/root/.kaggle", exist_ok=True)

# Upload `kaggle.json`
from google.colab import files
files.upload()  # Select and upload the downloaded kaggle.json file

# Move `kaggle.json` to the correct directory
!mv kaggle.json /root/.kaggle/

# Set permissions
!chmod 600 /root/.kaggle/kaggle.json

# Verify Kaggle API works
!kaggle datasets list
```

Choose Files kaggle.json
- **kaggle.json**(application/json) - 67 bytes, last modified: 3/20/2025 - 100% done
Saving kaggle.json to kaggle.json
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.7.4.2 / client 1.6.17)

```
ref                                                          title                                              size  las
-----------------------------------------------------------  -------------------------------------------------  ----- --
atharvasoundankar/chocolate-sales                            Chocolate Sales Data 📊 🍫                          14KB
atharvasoundankar/global-food-wastage-dataset-2018-2024      🌍 Global Food Wastage Dataset (2018-2024) 🍽️      106KB
abdulmalik1518/mobiles-dataset-2025                          Mobiles Dataset (2025)                             20KB  202
adilshamim8/student-depression-dataset                       Student Depression Dataset                        456KB  202
mahmoudelhemaly/students-grading-dataset                     Student Performance & Behavior Dataset            508KB  202
atharvasoundankar/global-water-consumption-dataset-2000-2024 Global Water Consumption Dataset (2000-2024) 🌍 💧  17KB
atharvasoundankar/global-energy-consumption-2000-2024        Global Energy Consumption (2000-2024) 🔥 ⚡        252KB
aniruddhawankhede/mental-heath-analysis-among-teenagers      Mental_Heath_Analysis_Among_Teenagers             173KB  202
salahuddinahmedshuvo/ecommerce-consumer-behavior-analysis-data Ecommerce Consumer Behavior Analysis Data        43KB  202
smayanj/netflix-users-database                               Netflix Users Database                            354KB  202
willianoliveiragibin/grocery-inventory                       Grocery Inventory                                  50KB  202
alikalwar/heart-attack-risk-prediction-cleaned-dataset       Heart Attack Risk Prediction Cleaned Dataset      671KB  202
atharvasoundankar/global-music-streaming-trends-and-listener-insights Global Music Streaming Trends & Listener Insights  95KB  202
atharvasoundankar/viral-social-media-trends-and-engagement-analysis 🚀 Viral Social Media Trends & Engagement Analysis 105KB  2
brsahan/genomic-data-for-cancer                              Genomic Data for Cancer                            9KB  202
amanrajput16/olympics-medal-list-1896-2024                   Olympic Medal List (1896-2024)                     11KB  202
miadul/brain-tumor-dataset                                   Brain Tumor Dataset                               852KB  202
adilshamim8/student-performance-on-an-entrance-examination   Student Performance on an Entrance Examination    4KB  202
anandshaw2001/video-game-sales                               Video Game Sales                                  381KB  202
atharvasoundankar/big-4-financial-risk-insights-2020-2025    Big 4 Financial Risk  Insights (2020-2025)        3KB  202
```

```python
!kaggle datasets download -d preetishah/waste-classificationorganic-and-recyclable
```

Warning: Looks like you're using an outdated API Version, please consider updating (server 1.7.4.2 / client 1.6.17)
Dataset URL: https://www.kaggle.com/datasets/preetishah/waste-classificationorganic-and-recyclable
License(s): apache-2.0
Downloading waste-classificationorganic-and-recyclable.zip to /content
 36% 9.00M/25.0M [00:00<00:00, 94.0MB/s]
100% 25.0M/25.0M [00:00<00:00, 157MB/s]

```python
import zipfile

with zipfile.ZipFile("waste-classificationorganic-and-recyclable.zip", 'r') as zip_ref:
    zip_ref.extractall("waste_classification")
```

```python
import os
print(os.listdir("/content/"))
```

['.config', 'waste-classificationorganic-and-recyclable.zip', 'waste_classification', 'sample_data']

```python
train_folder = "/content/waste_classification/wasteclassification/train"
test_folder = "/content/waste_classification/wasteclassification/test"
```

```python
import pandas as pd
import numpy as np
import glob
import os
from datetime import datetime
from packaging import version
```

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.callbacks import ModelCheckpoint, History
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input

from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, Lambda, MaxPooling2D, Dense, Dropout, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.densenet import preprocess_input
from tensorflow.keras.utils import to_categorical

from skimage.io import imread, imshow
from skimage.transform import resize
from IPython import display
import matplotlib.pyplot as plt
import seaborn as sns
from seaborn import heatmap
from sklearn.metrics import confusion_matrix
```

```python
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,  # Use DenseNet-specific preprocessing
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')


test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)


train_generator = train_datagen.flow_from_directory(
    train_folder,
    target_size=(224, 224),  # Matches DenseNet input size
    batch_size=32,
    class_mode='binary'  # Use 'categorical' for multi-class classification
)
```

⮒ Found 666 images belonging to 2 classes.

```python
test_generator = test_datagen.flow_from_directory(
    test_folder,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False)
```

⮒ Found 32 images belonging to 2 classes.

```python
from sklearn.utils.class_weight import compute_class_weight

# Get class labels from train_generator
class_labels = np.array(train_generator.classes)

# Compute class weights
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(class_labels), y=class_labels)

# Convert to dictionary format required for model.fit()
class_weight_dict = {i: class_weights[i] for i in range(len(class_weights))}

print("Class Weights:", class_weight_dict)
```

⮒ Class Weights: {0: np.float64(0.8740157480314961), 1: np.float64(1.168421052631579)}

```python
# Load the pre-trained DenseNet model without the classification layer
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model
base_model.trainable = False

# Define the new model structure
```

```
inputs = Input(shape=(224, 224, 3))  # Explicitly define input
x = base_model(inputs, training=False)  # Ensure frozen base model is applied correctly
x = GlobalAveragePooling2D()(x)  # Convert feature maps into a single vector
x = Dense(256, activation='relu')(x)  # Add a fully connected layer
outputs = Dense(1, activation='sigmoid')(x)  # Output layer for binary classification

# Create the final model
model = Model(inputs=inputs, outputs=outputs)  # Make sure inputs and outputs are linked correctly

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

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_...
29084464/29084464 ———————————— 0s 0us/step

```
# Train model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator,
    class_weight=class_weight_dict)
```

/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
Epoch 1/10
21/21 ———————————— 152s 6s/step - accuracy: 0.8479 - loss: 0.3110 - val_accuracy: 1.0000 - val_loss: 0.0045
Epoch 2/10
21/21 ———————————— 129s 6s/step - accuracy: 1.0000 - loss: 0.0096 - val_accuracy: 1.0000 - val_loss: 0.0049
Epoch 3/10
21/21 ———————————— 149s 7s/step - accuracy: 0.9977 - loss: 0.0069 - val_accuracy: 1.0000 - val_loss: 0.0012
Epoch 4/10
21/21 ———————————— 138s 7s/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 5/10
21/21 ———————————— 134s 6s/step - accuracy: 1.0000 - loss: 5.8860e-04 - val_accuracy: 1.0000 - val_loss: 8.5770e-04
Epoch 6/10
21/21 ———————————— 132s 6s/step - accuracy: 1.0000 - loss: 4.9483e-04 - val_accuracy: 1.0000 - val_loss: 6.2216e-04
Epoch 7/10
21/21 ———————————— 136s 6s/step - accuracy: 1.0000 - loss: 7.2648e-04 - val_accuracy: 1.0000 - val_loss: 6.8992e-04
Epoch 8/10
21/21 ———————————— 137s 6s/step - accuracy: 1.0000 - loss: 2.6747e-04 - val_accuracy: 1.0000 - val_loss: 6.2045e-04
Epoch 9/10
21/21 ———————————— 143s 6s/step - accuracy: 1.0000 - loss: 3.1907e-04 - val_accuracy: 1.0000 - val_loss: 5.7042e-04
Epoch 10/10
21/21 ———————————— 132s 6s/step - accuracy: 1.0000 - loss: 4.6311e-04 - val_accuracy: 1.0000 - val_loss: 3.9180e-04

```
# Plot accuracy and loss
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].plot(history.history['accuracy'], label='Train Accuracy')
axes[0].plot(history.history['val_accuracy'], label='Validation Accuracy')
axes[0].set_title('Model Accuracy')
axes[0].legend()
axes[1].plot(history.history['loss'], label='Train Loss')
axes[1].plot(history.history['val_loss'], label='Validation Loss')
axes[1].set_title('Model Loss')
axes[1].legend()
plt.show()
```

Model Accuracy | Model Loss
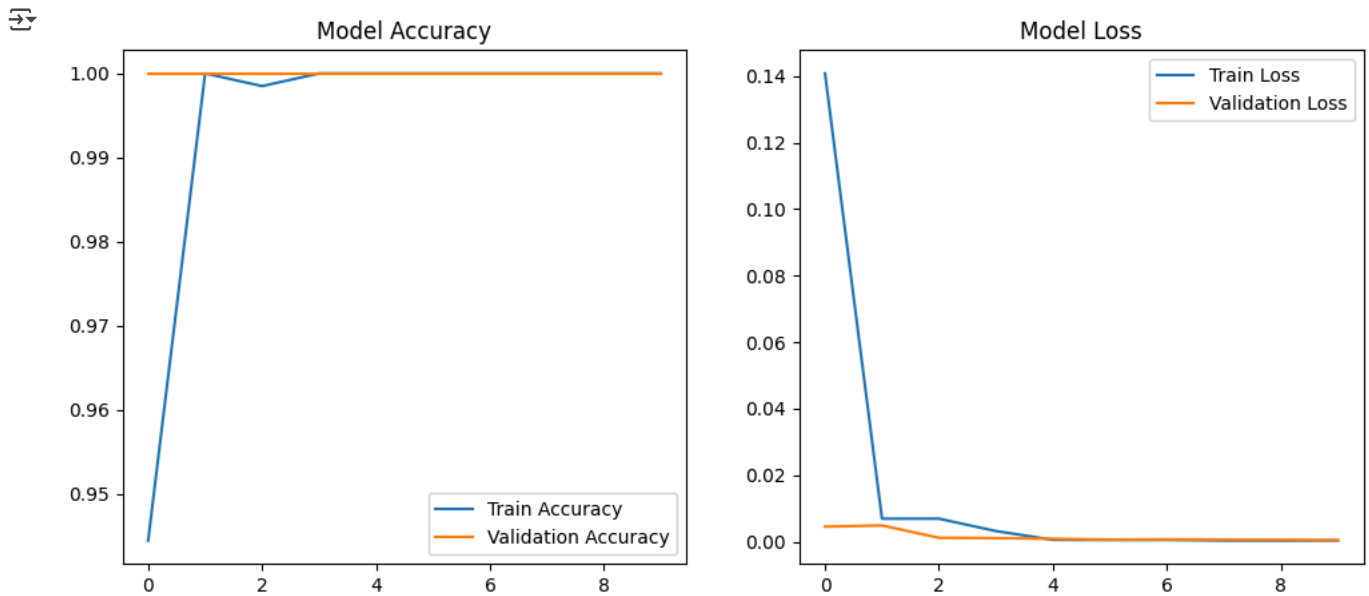
```
# Confusion matrix
y_true = test_generator.classes
y_pred = model.predict(test_generator) > 0.5
cm = confusion_matrix(y_true, y_pred)
```

1/1 ─────────────── **11s** 11s/step

```
# Display confusion matrix with labels and percentages
fig, ax = plt.subplots(figsize=(6, 6))
cm_percent = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] * 100
sns.heatmap(cm_percent, annot=True, fmt='.2f', cmap='Blues', xticklabels=['Pred Non-Recyclable', 'Pred Recyclable'],
            yticklabels=['Actual Non-Recyclable', 'Actual Recyclable'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
```

Text(0.5, 1.0, 'Confusion Matrix')
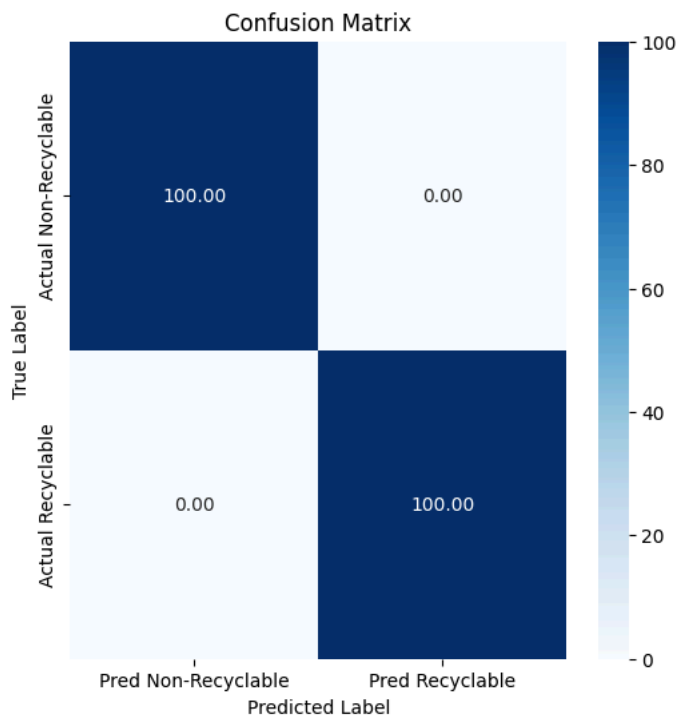

```
from sklearn.metrics import classification_report
# Classification report
print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=['Non-Recyclable', 'Recyclable']))
```

Classification Report:
                   precision   recall f1-score   support

Non-Recyclable	1.00	1.00	1.00	17
Recyclable	1.00	1.00	1.00	15
accuracy			1.00	32
macro avg	1.00	1.00	1.00	32
weighted avg	1.00	1.00	1.00	32

```
# Convert accuracy and loss to percentage
train_acc = [x * 100 for x in history.history['accuracy']]
val_acc = [x * 100 for x in history.history['val_accuracy']]
train_loss = [x * 100 for x in history.history['loss']]
val_loss = [x * 100 for x in history.history['val_loss']]
# Print accuracy and loss values
print("Final Training Accuracy: {:.2f}%".format(train_acc[-1]))
print("Final Validation Accuracy: {:.2f}%".format(val_acc[-1]))
print("Final Training Loss: {:.2f}%".format(train_loss[-1]))
print("Final Validation Loss: {:.2f}%".format(val_loss[-1]))
```