```
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Warning: Looks like you're using an outdated API Version, please consider updating (server 1.7.4.2 / client 1.6.17)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  la
---------------------------------------------------------------------  -------------------------------------------------  -----  --
atharvasoundankar/chocolate-sales                                      Chocolate Sales Data 📊 🍫                            14KB
abdulmalik1518/mobiles-dataset-2025                                    Mobiles Dataset (2025)                              20KB  20
mahmoudelhemaly/students-grading-dataset                               Student Performance & Behavior Dataset             508KB  20
atharvasoundankar/global-water-consumption-dataset-2000-2024           Global Water Consumption Dataset (2000-2024) 🌍 💧    17KB
adilshamim8/student-depression-dataset                                 Student Depression Dataset                         456KB  20
atharvasoundankar/global-food-wastage-dataset-2018-2024                🌍 Global Food Wastage Dataset (2018-2024) 🍽️      106KB
atharvasoundankar/global-energy-consumption-2000-2024                  Global Energy Consumption (2000-2024) 🔥 ⚡           252KB
parsabahramsari/wdi-education-health-and-employment-2011-2021          WDI: Education, Health & Employment (2011-2021)    133KB  20
bhargavchirumamilla/netflix-movies-and-tv-shows-till-2025              Netflix Movies and TV shows till 2025                6MB  20
aniruddhawankhede/mental-heath-analysis-among-teenagers                Mental_Heath_Analysis_Among_Teenagers              173KB  20
salahuddinahmedshuvo/ecommerce-consumer-behavior-analysis-data         Ecommerce Consumer Behavior Analysis Data           43KB  20
smayanj/netflix-users-database                                         Netflix Users Database                             354KB  20
willianoliveiragibin/grocery-inventory                                 Grocery Inventory                                   50KB  20
atharvasoundankar/global-music-streaming-trends-and-listener-insights  Global Music Streaming Trends & Listener Insights   95KB  20
atharvasoundankar/viral-social-media-trends-and-engagement-analysis    🚀 Viral Social Media Trends & Engagement Analysis  105KB  2
anandshaw2001/imdb-movies-and-tv-shows                                 IMDb Movies and TV Shows                             2MB  20
brsahan/genomic-data-for-cancer                                        Genomic Data for Cancer                              9KB  20
amanrajput16/olympics-medal-list-1896-2024                             Olympic Medal List (1896-2024)                      11KB  20
miadul/brain-tumor-dataset                                             Brain Tumor Dataset                                852KB  20
adilshamim8/student-performance-on-an-entrance-examination             Student Performance on an Entrance Examination       4KB  20
```

```
!kaggle datasets download -d jaiharish11499/wastedata
```

```
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/jaiharish11499/wastedata
License(s): CC0-1.0
Downloading wastedata.zip to /content
 98% 66.0M/67.5M [00:02<00:00, 43.3MB/s]
100% 67.5M/67.5M [00:02<00:00, 31.8MB/s]
```

```
import zipfile

with zipfile.ZipFile("wastedata.zip", 'r') as zip_ref:
    zip_ref.extractall("waste_data")
```

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

```
['.config', 'wastedata.zip', 'waste_data', 'sample_data']
```

```
train_folder = "/content/waste_data/d/Train"
test_folder = "/content/waste_data/d/Test"
```

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

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
```

```
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 336 images belonging to 2 classes.
```

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

```
Found 64 images belonging to 2 classes.
```

```
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: 4.666666666666667, 1: 0.56}
```

```
# 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
```

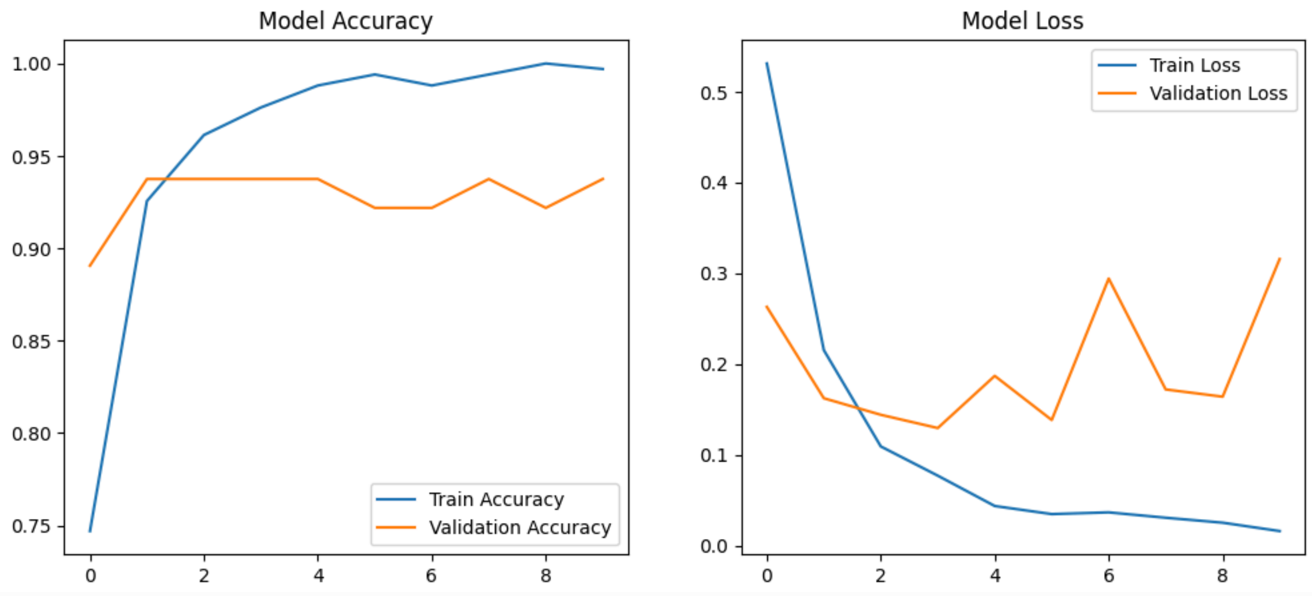
```
x = GlobalAveragePooling2D()(x)  # Convert feature maps into a single vector
x = Dense(256, activation='relu')(x)  # Add a fully connected layer
outputs = Dense(1, activation='sigmoid')(x)  # Output layer for binary classification

# 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_k
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 1s 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
11/11 ━━━━━━━━━━━━━━━━━━━━ 98s 8s/step - accuracy: 0.6625 - loss: 0.7341 - val_accuracy: 0.8906 - val_loss: 0.2630
Epoch 2/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 73s 7s/step - accuracy: 0.9229 - loss: 0.2217 - val_accuracy: 0.9375 - val_loss: 0.1623
Epoch 3/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 83s 7s/step - accuracy: 0.9600 - loss: 0.1238 - val_accuracy: 0.9375 - val_loss: 0.1440
Epoch 4/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 73s 7s/step - accuracy: 0.9898 - loss: 0.0614 - val_accuracy: 0.9375 - val_loss: 0.1294
Epoch 5/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 81s 7s/step - accuracy: 0.9845 - loss: 0.0449 - val_accuracy: 0.9375 - val_loss: 0.1869
Epoch 6/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 76s 7s/step - accuracy: 0.9960 - loss: 0.0403 - val_accuracy: 0.9219 - val_loss: 0.1383
Epoch 7/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 76s 7s/step - accuracy: 0.9882 - loss: 0.0359 - val_accuracy: 0.9219 - val_loss: 0.2942
Epoch 8/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 83s 8s/step - accuracy: 0.9984 - loss: 0.0349 - val_accuracy: 0.9375 - val_loss: 0.1720
Epoch 9/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 77s 7s/step - accuracy: 1.0000 - loss: 0.0218 - val_accuracy: 0.9219 - val_loss: 0.1640
Epoch 10/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 85s 8s/step - accuracy: 0.9919 - loss: 0.0191 - val_accuracy: 0.9375 - val_loss: 0.3158
```

```
# 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()
```

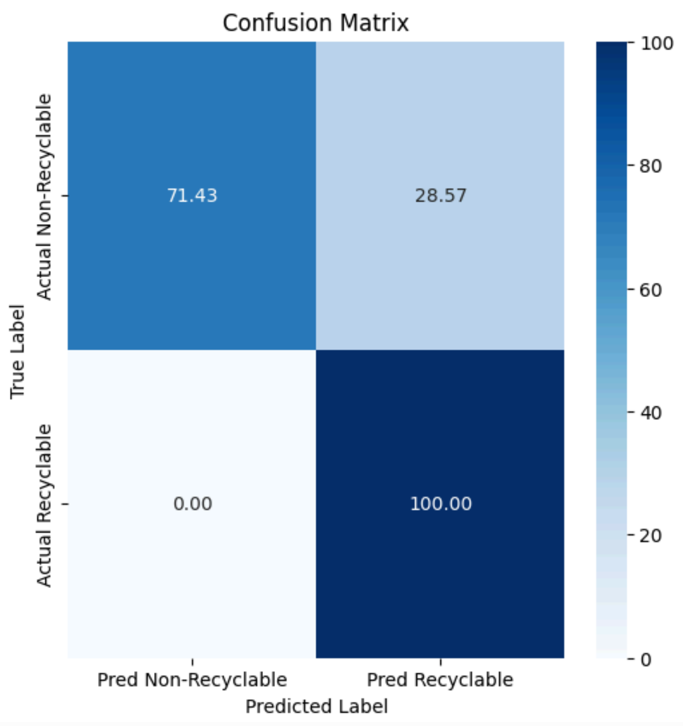


```
# Confusion matrix
y_true = test_generator.classes
y_pred = model.predict(test_generator) > 0.5
cm = confusion_matrix(y_true, y_pred)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 16s 6s/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      0.71      0.83        14
    Recyclable       0.93      1.00      0.96        50

      accuracy                           0.94        64
     macro avg       0.96      0.86      0.90        64
  weighted avg       0.94      0.94      0.93        64
```

```
# 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]))
```

```
Final Training Accuracy: 99.70%
Final Validation Accuracy: 93.75%
Final Training Loss: 1.58%
Final Validation Loss: 31.58%
```