```
# Mount to Google Drive for downloading dataset file

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

```
Mounted at /content/gdrive/
```

```
# Unzip the dataset file

!unzip /content/gdrive/MyDrive/deep-learning-recycle-item-classification-main.zip
```

```
Archive:  /content/gdrive/MyDrive/deep-learning-recycle-item-classification-main.zip
33ae657e01683187bb19c4351555cd56aa5329d3
   creating: deep-learning-recycle-item-classification-main/
  inflating: deep-learning-recycle-item-classification-main/LICENSE
  inflating: deep-learning-recycle-item-classification-main/README.md
   creating: deep-learning-recycle-item-classification-main/code/
  inflating: deep-learning-recycle-item-classification-main/code/deep-learning-real-life-item-classification.ipynb
  inflating: deep-learning-recycle-item-classification-main/code/deep-learning-real-life-item-classification.pdf
   creating: deep-learning-recycle-item-classification-main/dataset/
  inflating: deep-learning-recycle-item-classification-main/dataset/Dataset.zip
   creating: deep-learning-recycle-item-classification-main/images/
  inflating: deep-learning-recycle-item-classification-main/images/accuracy-validation.png
  inflating: deep-learning-recycle-item-classification-main/images/confusion-matrix.png
  inflating: deep-learning-recycle-item-classification-main/images/item-classification-deep-learning.png
  inflating: deep-learning-recycle-item-classification-main/images/loss-validation.png
```

```
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
```

```
!unzip /content/deep-learning-recycle-item-classification-main/dataset/Dataset.zip
```

```
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_92.jpg
  inflating: Dataset/Train/Non-recyclable/O_86.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_86.jpg
  inflating: Dataset/Train/Non-recyclable/O_51.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_51.jpg
  inflating: Dataset/Train/Non-recyclable/O_45.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_45.jpg
  inflating: Dataset/Train/Non-recyclable/O_79.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_79.jpg
  inflating: Dataset/Train/Non-recyclable/O_41.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_41.jpg
  inflating: Dataset/Train/Non-recyclable/O_55.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_55.jpg
  inflating: Dataset/Train/Non-recyclable/O_69.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_69.jpg
  inflating: Dataset/Train/Non-recyclable/O_82.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_82.jpg
  inflating: Dataset/Train/Non-recyclable/O_96.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_96.jpg
  inflating: Dataset/Train/Non-recyclable/O_264.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_264.jpg
  inflating: Dataset/Train/Non-recyclable/O_270.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_270.jpg
  inflating: Dataset/Train/Non-recyclable/O_258.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_258.jpg
  inflating: Dataset/Train/Non-recyclable/O_476.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_476.jpg
  inflating: Dataset/Train/Non-recyclable/O_310.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_310.jpg
  inflating: Dataset/Train/Non-recyclable/O_304.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_304.jpg
  inflating: Dataset/Train/Non-recyclable/O_462.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_462.jpg
  inflating: Dataset/Train/Non-recyclable/O_338.jpg
  inflating: __MACOSX/Dataset/Train/Non-recyclable/._O_338.jpg
  inflating: Dataset/Train/Non-recyclable/O_6.jpg
```

```
# Define train and test image folder path

train_folder = "/content/Dataset/Train"
test_folder = "/content/Dataset/Test"
```

```
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 999 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 1234 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)
```

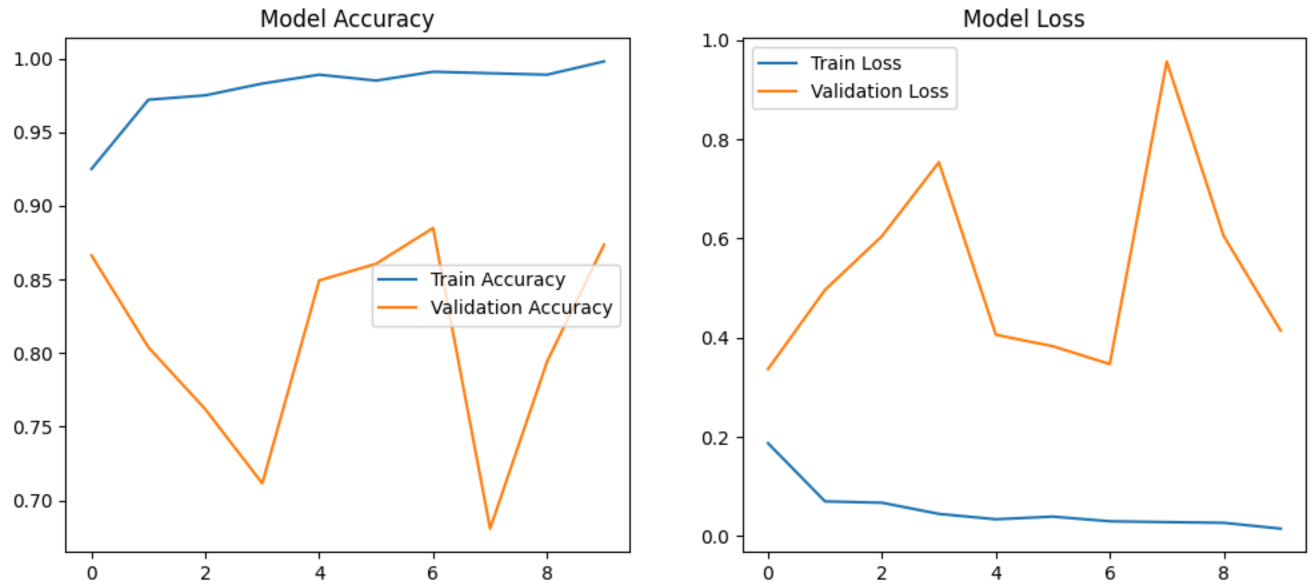
```
Class Weights: {0: 1.001002004008016, 1: 0.999}
```

```
# 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'])
```

```
# 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
32/32 ━━━━━━━━━━━━━━━━━━━━ 446s 14s/step - accuracy: 0.8672 - loss: 0.3081 - val_accuracy: 0.8663 - val_loss: 0.3368
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 457s 15s/step - accuracy: 0.9745 - loss: 0.0684 - val_accuracy: 0.8039 - val_loss: 0.4963
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 455s 14s/step - accuracy: 0.9767 - loss: 0.0562 - val_accuracy: 0.7618 - val_loss: 0.6047
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 460s 15s/step - accuracy: 0.9843 - loss: 0.0410 - val_accuracy: 0.7115 - val_loss: 0.7536
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 463s 13s/step - accuracy: 0.9916 - loss: 0.0267 - val_accuracy: 0.8493 - val_loss: 0.4059
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 415s 13s/step - accuracy: 0.9835 - loss: 0.0443 - val_accuracy: 0.8606 - val_loss: 0.3828
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 415s 13s/step - accuracy: 0.9900 - loss: 0.0322 - val_accuracy: 0.8849 - val_loss: 0.3469
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 416s 13s/step - accuracy: 0.9914 - loss: 0.0253 - val_accuracy: 0.6807 - val_loss: 0.9567
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 456s 15s/step - accuracy: 0.9822 - loss: 0.0334 - val_accuracy: 0.7942 - val_loss: 0.6050
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 420s 13s/step - accuracy: 0.9988 - loss: 0.0140 - val_accuracy: 0.8736 - val_loss: 0.4142
```

```
# 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()
```

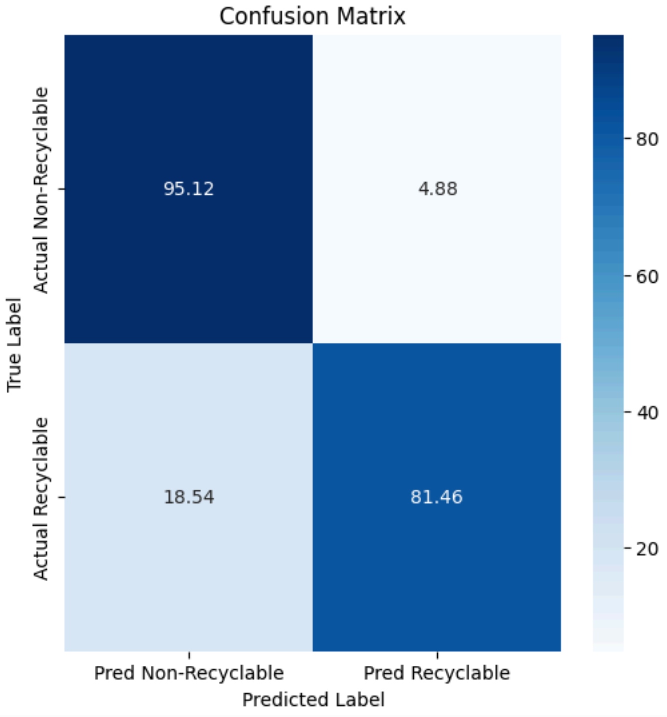


```
# Confusion matrix
y_true = test_generator.classes
y_pred = model.predict(test_generator) > 0.5
cm = confusion_matrix(y_true, y_pred)
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 236s 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       0.80      0.95      0.87       533
    Recyclable       0.96      0.81      0.88       701

      accuracy                           0.87      1234
     macro avg       0.88      0.88      0.87      1234
  weighted avg       0.89      0.87      0.87      1234
```

```
# 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: 99.80%
Final Validation Accuracy: 87.36%
Final Training Loss: 1.52%
Final Validation Loss: 41.42%
```