```
# 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 InceptionV3
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 Sequential, load_model
from tensorflow.keras.layers import Conv2D, Lambda, MaxPooling2D, Dense, Dropout, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
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: 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"


from tensorflow.keras.applications.inception_v3 import preprocess_input
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,  # InceptionV3-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'
)



# No augmentation for validation/test
test_datagen = ImageDataGenerator(rescale=1./255)


# Load dataset
train_generator = train_datagen.flow_from_directory(
    train_folder,
    target_size=(299, 299),
    batch_size=32,
    class_mode='binary')
```

```
Found 999 images belonging to 2 classes.
```

```
test_generator = test_datagen.flow_from_directory(
    test_folder,
    target_size=(299, 299),
    batch_size=32,
    class_mode='binary',
    shuffle=False)
```

```
Found 1234 images belonging to 2 classes.
```

```
from sklearn.utils.class_weight import compute_class_weight
# Compute class weights to address imbalance
class_labels = np.array(train_generator.classes)
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(class_labels), y=class_labels)
class_weight_dict = {i: class_weights[i] for i in range(len(class_weights))}
```

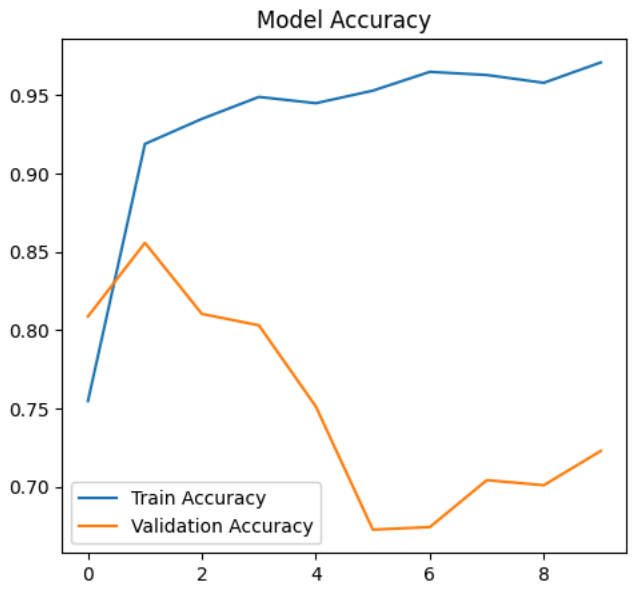
```
# Load InceptionV3 base model (pre-trained on ImageNet)
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))  # 299x299 input size
base_model.trainable = False
```

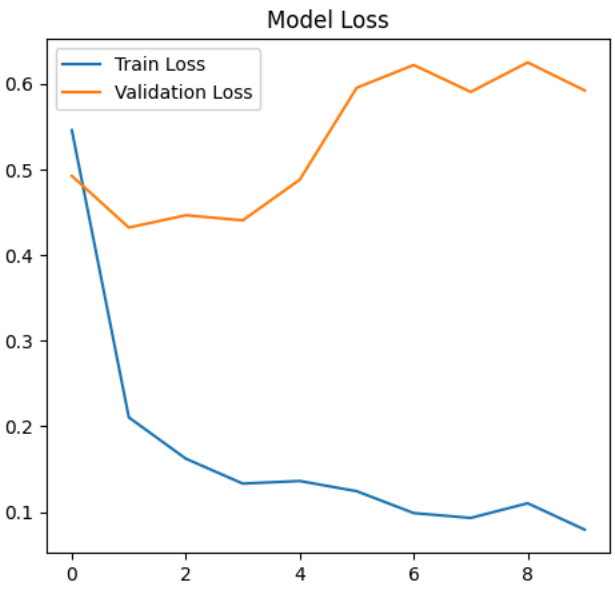
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
# Build model
model = keras.Sequential([
    base_model,
    keras.layers.GlobalAveragePooling2D(),  # Efficient feature extraction
    keras.layers.Dense(128, activation='relu'),
    keras.layers.BatchNormalization(),  # Improves stability
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation='sigmoid')  # Binary classification
])


from tensorflow.keras.optimizers import Adam
#Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), 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 ━━━━━━━━━━━━━━━━━━━━ 586s 18s/step - accuracy: 0.6425 - loss: 0.7728 - val_accuracy: 0.8088 - val_loss: 0.4926
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 537s 17s/step - accuracy: 0.9105 - loss: 0.2172 - val_accuracy: 0.8558 - val_loss: 0.4323
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 537s 17s/step - accuracy: 0.9275 - loss: 0.1755 - val_accuracy: 0.8104 - val_loss: 0.4467
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 529s 17s/step - accuracy: 0.9409 - loss: 0.1618 - val_accuracy: 0.8031 - val_loss: 0.4407
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 531s 17s/step - accuracy: 0.9515 - loss: 0.1236 - val_accuracy: 0.7512 - val_loss: 0.4882
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 560s 17s/step - accuracy: 0.9552 - loss: 0.1153 - val_accuracy: 0.6726 - val_loss: 0.5956
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 528s 17s/step - accuracy: 0.9644 - loss: 0.0930 - val_accuracy: 0.6742 - val_loss: 0.6222
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 527s 17s/step - accuracy: 0.9594 - loss: 0.0858 - val_accuracy: 0.7042 - val_loss: 0.5907
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 528s 17s/step - accuracy: 0.9487 - loss: 0.1259 - val_accuracy: 0.7010 - val_loss: 0.6252
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 568s 18s/step - accuracy: 0.9642 - loss: 0.0967 - val_accuracy: 0.7229 - val_loss: 0.5925
```

```
# 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()
```



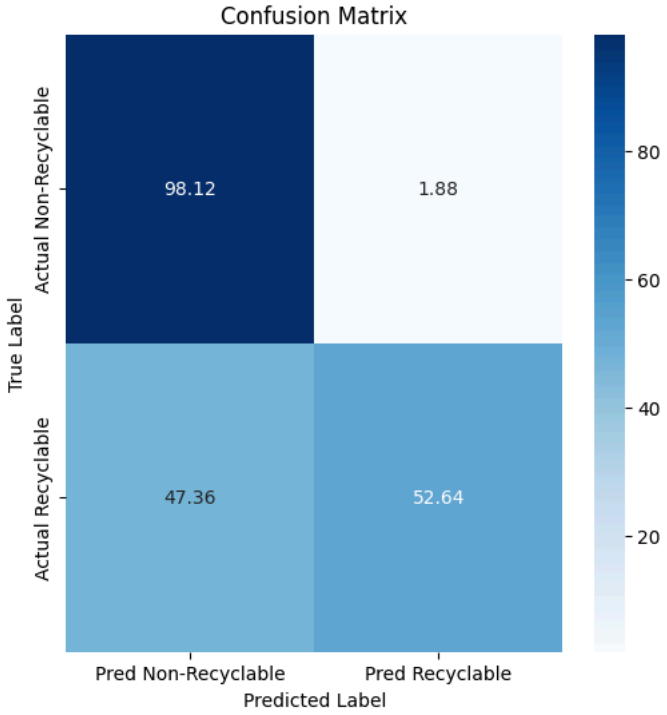



```
# Confusion matrix
y_true = test_generator.classes
y_pred = model.predict(test_generator) > 0.5
cm = confusion_matrix(y_true, y_pred)
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 289s 7s/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       0.61      0.98      0.75       533
    Recyclable       0.97      0.53      0.68       701

      accuracy                           0.72      1234
     macro avg       0.79      0.75      0.72      1234
  weighted avg       0.82      0.72      0.71      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: 97.10%
Final Validation Accuracy: 72.29%
Final Training Loss: 7.95%
Final Validation Loss: 59.25%
```