Libraries

```
!pip install tensorflow
!pip install scikit-learn
!pip install matplotlib
!pip install Pillow
!pip install keras
!pip install opencv-python
```

Show hidden output

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2
import shutil
import os
import random
import collections
import seaborn as sns
import zipfile
import os
import pandas as pd

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix
from keras.utils import load_img
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import GlobalAveragePooling2D
```

Load Data

```
from google.colab import files
uploaded = files.upload()
```

Choose Files Garbage Classification.zip

- **Garbage Classification.zip**(application/x-zip-compressed) - 85969666 bytes, last modified: 4/16/2025 - 100% done

Saving Garbage Classification.zip to Garbage Classification (1).zip

```
zip_path = "/content/Garbage Classification (1).zip"
extract_path = "/content/DataSet"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```

```
import os

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

```
['one-indexed-files-notrash_test.txt', 'garbage classification', 'one-indexed-files.txt', 'Garbage classification', 'zero-indexed-files.
```

Split Data

```
from sklearn.model_selection import train_test_split

original_dataset = "/content/DataSet/Garbage classification/Garbage classification"
base_dir = "/content/garbage_dataset"
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'val')
test_dir = os.path.join(base_dir, 'test')
```

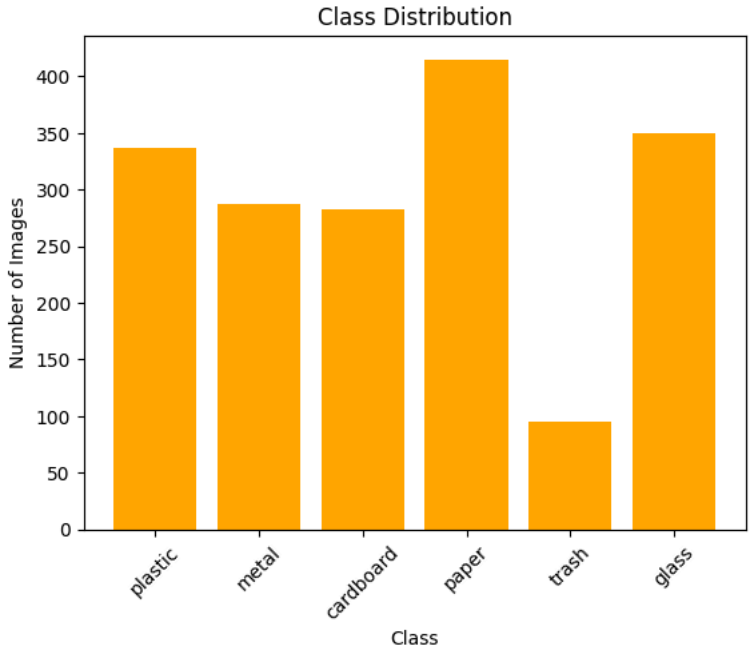
```
test_dir = os.path.join(base_dir, 'test')

for folder in [train_dir, val_dir, test_dir]:
    os.makedirs(folder, exist_ok=True)

for class_name in os.listdir(original_dataset):
    class_path = os.path.join(original_dataset, class_name)
    if not os.path.isdir(class_path):
        continue

    images = os.listdir(class_path)

    train_images, temp_images = train_test_split(images, test_size=0.3, random_state=42)
    val_images, test_images = train_test_split(temp_images, test_size=0.5, random_state=42)


    for img in train_images:
        src = os.path.join(class_path, img)
        dst = os.path.join(train_dir, class_name, img)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        shutil.copy(src, dst)


    for img in val_images:
        src = os.path.join(class_path, img)
        dst = os.path.join(val_dir, class_name, img)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        shutil.copy(src, dst)


    for img in test_images:
        src = os.path.join(class_path, img)
        dst = os.path.join(test_dir, class_name, img)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        shutil.copy(src, dst)

print("Data successfully split into:- Training: 70% (in", train_dir, "- Validation : 15% (in", val_dir, "- Testing : 15% (in", test_dir)
```

```
Data successfully split into:- Training: 70% (in /content/garbage_dataset/train - Validation : 15% (in /content/garbage_dataset/val - Te
```

Pre Processing

```
def plot_class_distribution(directory):
    class_counts = {}
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            class_counts[class_name] = len(os.listdir(class_path))

    plt.bar(class_counts.keys(), class_counts.values(), color='orange')
    plt.title('Class Distribution')
    plt.xlabel('Class')
    plt.ylabel('Number of Images')
    plt.xticks(rotation=45)
    plt.show()

print("Train Distribution:")
plot_class_distribution('/content/garbage_dataset/train')
```

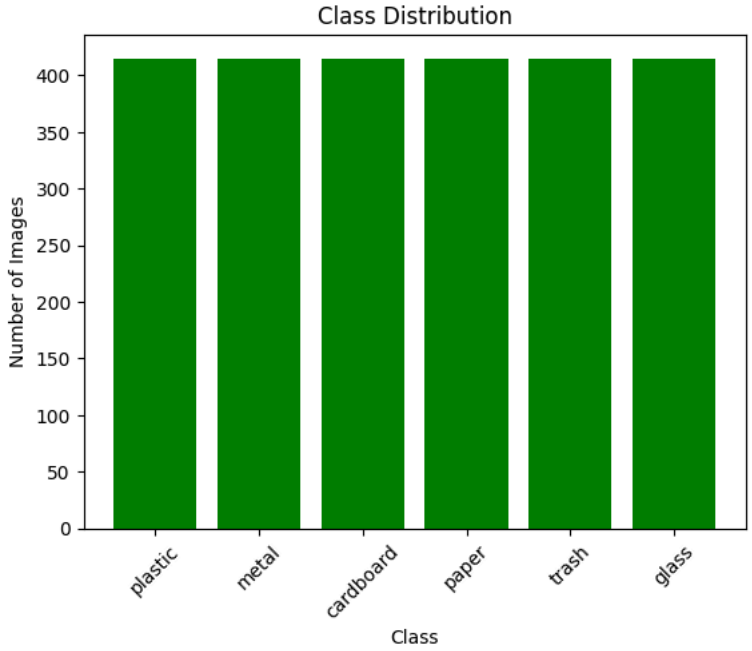
Train Distribution:




```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from PIL import Image
import random
import numpy as np

train_dir = '/content/garbage_dataset/train'
class_counts = {cls: len(os.listdir(os.path.join(train_dir, cls))) for cls in os.listdir(train_dir)}
max_images = max(class_counts.values())

aug = ImageDataGenerator(
    rotation_range=40,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest"
)

for class_name, count in class_counts.items():
    folder = os.path.join(train_dir, class_name)
    images = os.listdir(folder)
    current_count = len(images)

    while current_count < max_images:
        img_name = random.choice(images)
        img_path = os.path.join(folder, img_name)

        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)

        gen = aug.flow(image, batch_size=1)
        new_image = next(gen)[0].astype(np.uint8)

        save_path = os.path.join(folder, f"aug_{current_count}.jpg")
        Image.fromarray(new_image).save(save_path)

        current_count += 1



def plot_class_distribution(directory):
    class_counts = {}
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            class_counts[class_name] = len(os.listdir(class_path))
```

```
    plt.bar(class_counts.keys(), class_counts.values(), color='green')
    plt.title('Class Distribution')
    plt.xlabel('Class')
    plt.ylabel('Number of Images')
    plt.xticks(rotation=45)
    plt.show()

print("Train Distribution:")
plot_class_distribution('/content/garbage_dataset/train')
```

```
Train Distribution:
```




```
def custom_normalization(img):
    img = img / 255.0
    img = img - 0.5
    return img

train_datagen = ImageDataGenerator(
    preprocessing_function=custom_normalization,
    validation_split=0.3,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(
    preprocessing_function=custom_normalization,
    validation_split=0.3
)

test_datagen = ImageDataGenerator(
    preprocessing_function=custom_normalization
)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
```

```
)

val_generator = val_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

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

```
Found 1746 images belonging to 6 classes.
Found 744 images belonging to 6 classes.
Found 383 images belonging to 6 classes.
```

```
from sklearn.utils.class_weight import compute_class_weight

y_labels = train_generator.classes

class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_labels),
    y=y_labels
)

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(1.0), 1: np.float64(1.0), 2: np.float64(1.0), 3: np.float64(1.0), 4: np.float64(1.0), 5: np.float64(1.0)}
```

Build Model

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout, GlobalAveragePooling2D, BatchNormalization, Activation
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2


base_model = DenseNet201(include_top=False, input_shape=(224, 224, 3), weights='imagenet')
base_model.trainable = False

inputs = Input(shape=(224, 224, 3))
x = base_model(inputs, training=False)
x = GlobalAveragePooling2D()(x)

x = Dense(256, kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

x = Dense(128, kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

x = Dense(32, kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.3)(x)

outputs = Dense(6, activation='softmax')(x)
```

```
model = Model(inputs, outputs)

model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_9 (InputLayer)	(None, 224, 224, 3)	0
densenet201 (Functional)	(None, 7, 7, 1920)	18,321,984
global_average_pooling2d_4 (GlobalAveragePooling2D)	(None, 1920)	0
dense_7 (Dense)	(None, 256)	491,776
batch_normalization_6 (BatchNormalization)	(None, 256)	1,024
activation_5 (Activation)	(None, 256)	0
dropout_5 (Dropout)	(None, 256)	0
dense_8 (Dense)	(None, 128)	32,896
batch_normalization_7 (BatchNormalization)	(None, 128)	512
activation_6 (Activation)	(None, 128)	0
dropout_6 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 32)	4,128
batch_normalization_8 (BatchNormalization)	(None, 32)	128
activation_7 (Activation)	(None, 32)	0
dropout_7 (Dropout)	(None, 32)	0
dense_10 (Dense)	(None, 6)	198

Total params: 18,852,646 (71.92 MB)
Trainable params: 529,830 (2.02 MB)
Non-trainable params: 18,322,816 (69.90 MB)

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

lr_schedule = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=10,
    verbose=1,
    min_lr=1e-6
)

checkpoint_path = "best_model_Version_1.h5"
checkpoint = ModelCheckpoint(
    checkpoint_path,
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=1
)

optimizer = Adam(learning_rate=1e-4)
model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=200,
```

```
    callbacks=[lr_schedule, checkpoint],
    class_weight=class_weight_dict
)
```

```
Epoch 187/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 471ms/step - accuracy: 0.9426 - loss: 0.4480
Epoch 187: val_accuracy did not improve from 0.95430
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 525ms/step - accuracy: 0.9426 - loss: 0.4480 - val_accuracy: 0.9530 - val_loss: 0.4101 - learning_rate:
Epoch 188/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 473ms/step - accuracy: 0.9484 - loss: 0.4352
Epoch 188: val_accuracy did not improve from 0.95430
55/55 ━━━━━━━━━━━━━━━━━━━━ 31s 568ms/step - accuracy: 0.9484 - loss: 0.4352 - val_accuracy: 0.9530 - val_loss: 0.4080 - learning_rate:
Epoch 189/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 474ms/step - accuracy: 0.9467 - loss: 0.4238
Epoch 189: val_accuracy did not improve from 0.95430
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 528ms/step - accuracy: 0.9467 - loss: 0.4239 - val_accuracy: 0.9516 - val_loss: 0.4016 - learning_rate:
Epoch 190/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 472ms/step - accuracy: 0.9575 - loss: 0.4050
Epoch 190: val_accuracy did not improve from 0.95430
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 526ms/step - accuracy: 0.9575 - loss: 0.4050 - val_accuracy: 0.9516 - val_loss: 0.4075 - learning_rate:
Epoch 191/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 558ms/step - accuracy: 0.9447 - loss: 0.4229
Epoch 191: val_accuracy did not improve from 0.95430
55/55 ━━━━━━━━━━━━━━━━━━━━ 34s 611ms/step - accuracy: 0.9448 - loss: 0.4230 - val_accuracy: 0.9476 - val_loss: 0.4158 - learning_rate:
Epoch 192/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 473ms/step - accuracy: 0.9506 - loss: 0.4147
Epoch 192: val_accuracy improved from 0.95430 to 0.95565, saving model to best_model_Version_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is con
55/55 ━━━━━━━━━━━━━━━━━━━━ 33s 599ms/step - accuracy: 0.9505 - loss: 0.4149 - val_accuracy: 0.9556 - val_loss: 0.3972 - learning_rate:
Epoch 193/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 479ms/step - accuracy: 0.9634 - loss: 0.3940
Epoch 193: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 533ms/step - accuracy: 0.9633 - loss: 0.3943 - val_accuracy: 0.9556 - val_loss: 0.3906 - learning_rate:
Epoch 194/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 475ms/step - accuracy: 0.9408 - loss: 0.4528
Epoch 194: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 530ms/step - accuracy: 0.9409 - loss: 0.4527 - val_accuracy: 0.9503 - val_loss: 0.3997 - learning_rate:
Epoch 195/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 472ms/step - accuracy: 0.9372 - loss: 0.4134
Epoch 195: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 526ms/step - accuracy: 0.9374 - loss: 0.4132 - val_accuracy: 0.9530 - val_loss: 0.3974 - learning_rate:
Epoch 196/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 473ms/step - accuracy: 0.9519 - loss: 0.4168
Epoch 196: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 527ms/step - accuracy: 0.9518 - loss: 0.4171 - val_accuracy: 0.9489 - val_loss: 0.4029 - learning_rate:
Epoch 197/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 483ms/step - accuracy: 0.9538 - loss: 0.4012
Epoch 197: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 30s 537ms/step - accuracy: 0.9538 - loss: 0.4011 - val_accuracy: 0.9530 - val_loss: 0.3921 - learning_rate:
Epoch 198/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 473ms/step - accuracy: 0.9395 - loss: 0.4355
Epoch 198: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 527ms/step - accuracy: 0.9395 - loss: 0.4356 - val_accuracy: 0.9530 - val_loss: 0.4008 - learning_rate:
Epoch 199/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 468ms/step - accuracy: 0.9399 - loss: 0.4246
Epoch 199: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 31s 563ms/step - accuracy: 0.9400 - loss: 0.4243 - val_accuracy: 0.9543 - val_loss: 0.3909 - learning_rate:
Epoch 200/200
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 475ms/step - accuracy: 0.9544 - loss: 0.3981
Epoch 200: val_accuracy did not improve from 0.95565
55/55 ━━━━━━━━━━━━━━━━━━━━ 29s 529ms/step - accuracy: 0.9543 - loss: 0.3984 - val_accuracy: 0.9543 - val_loss: 0.3861 - learning_rate:
```

```
from tensorflow.keras.models import load_model

model = load_model("best_model_Version_1.h5")

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.save('garbage_classification_model_one.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

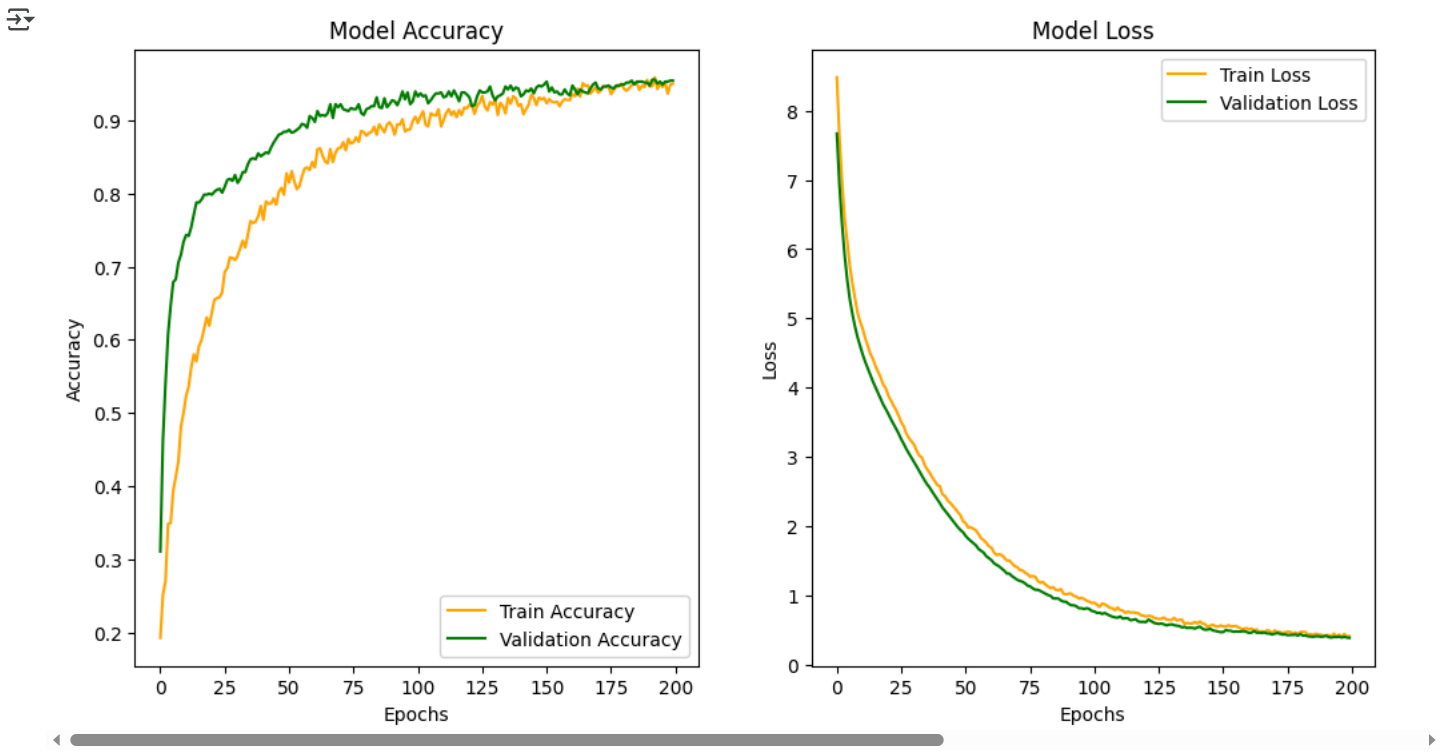
```
train_loss, train_accu = model.evaluate(train_generator)
val_loss, val_accu = model.evaluate(val_generator)
```

```
test_loss, test_accu = model.evaluate(test_generator)

print("final train accuracy = {:.2f}, validation accuracy = {:.2f}, testing accuracy = {:.2f}"
      .format(train_accu*100, val_accu*100, test_accu*100))
```

```
55/55 ━━━━━━━━━━━━━━━━━━━━ 59s 716ms/step - accuracy: 0.9945 - loss: 0.2725
24/24 ━━━━━━━━━━━━━━━━━━━━ 15s 632ms/step - accuracy: 0.9741 - loss: 0.3438
12/12 ━━━━━━━━━━━━━━━━━━━━ 24s 2s/step - accuracy: 0.9420 - loss: 0.4534
final train accuracy = 99.43, validation accuracy = 95.56, testing accuracy = 91.64
```

Evaluation

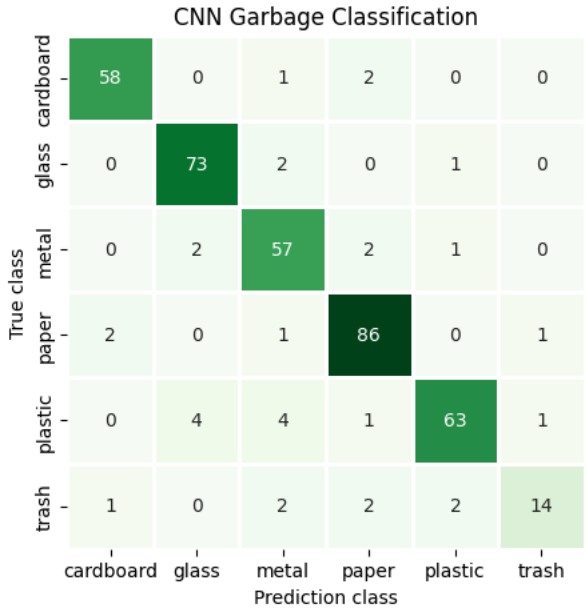
```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', color='orange')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='green')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', color='orange')
plt.plot(history.history['val_loss'], label='Validation Loss', color='green')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
train_loss, train_accu = model.evaluate(train_generator)
val_loss, val_accu = model.evaluate(val_generator)
test_loss, test_accu = model.evaluate(test_generator)

print("final train accuracy = {:.2f}, validation accuracy = {:.2f}, testing accuracy = {:.2f}"
      .format(train_accu*100, val_accu*100, test_accu*100))


Y_pred = model.predict(test_generator)
y_pred = np.argmax(Y_pred, axis=1)

label = ['cardboard','glass','metal','paper','plastic','trash']
labels = {0 : 'cardboard', 1 : 'glass', 2 : 'metal', 3 : 'paper', 4 : 'plastic',5 :'trash'}
```

```
cm = confusion_matrix(test_generator.classes, y_pred)
cm_df = pd.DataFrame(cm, index = label,
                     columns = label
                    )


plt.figure(figsize = (5,5))
sns.heatmap(cm_df, annot = True,cmap='Greens',cbar=False,linewidth=2,fmt='d')
plt.title('CNN Garbage Classification')
plt.ylabel('True class')
plt.xlabel('Prediction class')
plt.show()
```

```
55/55 ━━━━━━━━━━━━━━━━━━━━ 27s 486ms/step - accuracy: 0.9972 - loss: 0.2721
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 120ms/step - accuracy: 0.9741 - loss: 0.3438
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 125ms/step - accuracy: 0.9420 - loss: 0.4534
final train accuracy = 99.37, validation accuracy = 95.56, testing accuracy = 91.64
12/12 ━━━━━━━━━━━━━━━━━━━━ 40s 2s/step
```



Test

```
def choose_image_and_predict(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (224, 224))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)
    img = np.stack([img] * 3, axis=-1)
    pred = model.predict(img)
    label = np.argmax(pred, axis=1)[0]
    return labels[label]

images = ["cardboard.jpg", "glass.jpg", "metal.jpg", "paper.jpg", "plastic.jpg", "trash.jpg"]

fig = plt.figure(figsize=(12, 8))
rows, columns = 2, 3

for idx, image_path in enumerate(images):
    fig.add_subplot(rows, columns, idx + 1)
    plt.imshow(load_img(image_path))
    plt.axis('off')
    title = choose_image_and_predict(image_path)
    plt.title(title, fontsize=12)

plt.tight_layout()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
```

cardboard

glass

metal

paper

plastic

trash