```
#import image datagenerator library
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Image Data Augmentation

```
#setting parameter  for image data augmentation to the training data.
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
```

```
#image data augmentation to the testing data.
val_datagen = ImageDataGenerator(rescale = 1./255)
```

Loading Our Data And Performing Data Augmentaion

```
train_transform = train_datagen.flow_from_directory(r"I:\SmartBridge Projects\Garbage-waste-prediction\trainset",
                                                    target_size=(128,128),
                                                    batch_size=64,
                                                    class_mode='categorical')
```

```
Found 2527 images belonging to 6 classes.
```

```
test_transform = val_datagen.flow_from_directory(r"I:\SmartBridge Projects\Garbage-waste-prediction\testset",
                                                 target_size=(128,128),
                                                 batch_size=64,
                                                 class_mode='categorical')
```

```
Found 782 images belonging to 6 classes.
```

Importing Necessary Libraries

```
#to define linear initializations import Sequential
from tensorflow.keras.models import Sequential
#To add layers import Dense
from tensorflow.keras.layers import Dense
# to create a convolution kernel import Convolution2D
from tensorflow.keras.layers import Convolution2D
# Adding Max pooling Layer
from tensorflow.keras.layers import MaxPooling2D
# Adding Flatten Layer
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import Adam
```

```
# Initializing the model
model=Sequential()
```

```
# Initializing the model
model=Sequential()
```

```
#First Convolution layer and pooling
model.add(Convolution2D(32,(3,3),input_shape=(128,128,3),activation='relu'))
model.add(MaxPooling2D(2,2))
```

```
#Second Convolution layer and pooling
model.add(Convolution2D(64,(3,3),padding='same',activation='relu'))

#input shape is going to be the pooled feature maps from the previous convolution.
model.add(MaxPooling2D(pool_size=2))
```

```
#Third Convolution layer and pooling
model.add(Convolution2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
```

```
#Fourth Convolution layer and pooling
model.add(Convolution2D(32,(3,3), padding='same',activation='relu'))

#input shape is going to be the pooled feature maps from the previous convolution.
model.add(MaxPooling2D(pool_size=2))
```

```
#Flattening the layers
model.add(Flatten())
```

Adding Fully Connected Layer

```
# Adding 1st hidden layer
model.add(Dense(kernel_initializer='uniform',activation='relu',units=150))
```

```
# Adding 2nd hidden layer
model.add(Dense(kernel_initializer='uniform',activation='relu',units=68))
```

```
model.add(Dense(kernel_initializer='uniform',activation='softmax',units=6))
```

Summary of the Model

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 127008)            0
_________________________________________________________________
dense (Dense)                (None, 150)               19051350
_________________________________________________________________
dense_1 (Dense)              (None, 68)                10268
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 414
=================================================================
Total params: 19,062,928
Trainable params: 19,062,928
Non-trainable params: 0
```

Compiling the Model

```
#Compiling the CNN Model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

```
C:\Users\smartbridge\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1844: UserWarning: `Model.fit_generator`
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/30
41/41 [==============================] - 45s 1s/step - loss: 1.7747 - acc: 0.2061 - val_loss: 1.4693 - val_acc: 0.4132
Epoch 2/30
41/41 [==============================] - 53s 1s/step - loss: 1.5144 - acc: 0.3893 - val_loss: 1.3684 - val_acc: 0.4410
Epoch 3/30
41/41 [==============================] - 43s 1s/step - loss: 1.3444 - acc: 0.4624 - val_loss: 1.2694 - val_acc: 0.5122
Epoch 4/30
41/41 [==============================] - 49s 1s/step - loss: 1.2176 - acc: 0.5210 - val_loss: 1.1758 - val_acc: 0.5469
Epoch 5/30
41/41 [==============================] - 47s 1s/step - loss: 1.2179 - acc: 0.4982 - val_loss: 1.0858 - val_acc: 0.5694
Epoch 6/30
41/41 [==============================] - 48s 1s/step - loss: 1.1780 - acc: 0.5352 - val_loss: 1.0922 - val_acc: 0.5868
Epoch 7/30
41/41 [==============================] - 50s 1s/step - loss: 1.0955 - acc: 0.5836 - val_loss: 1.0062 - val_acc: 0.6111
Epoch 8/30
41/41 [==============================] - 53s 1s/step - loss: 1.0096 - acc: 0.6127 - val_loss: 1.0593 - val_acc: 0.5885
Epoch 9/30
41/41 [==============================] - 52s 1s/step - loss: 1.0005 - acc: 0.6200 - val_loss: 0.8735 - val_acc: 0.6701
Epoch 10/30
41/41 [==============================] - 50s 1s/step - loss: 0.9507 - acc: 0.6571 - val_loss: 0.8716 - val_acc: 0.6806
Epoch 11/30
41/41 [==============================] - 47s 1s/step - loss: 0.8568 - acc: 0.6822 - val_loss: 0.8149 - val_acc: 0.7083
Epoch 12/30
41/41 [==============================] - 47s 1s/step - loss: 0.8062 - acc: 0.7054 - val_loss: 0.7221 - val_acc: 0.7500
Epoch 13/30
41/41 [==============================] - 51s 1s/step - loss: 0.7450 - acc: 0.7262 - val_loss: 0.6855 - val_acc: 0.7465
Epoch 14/30
41/41 [==============================] - 54s 1s/step - loss: 0.7229 - acc: 0.7402 - val_loss: 0.6877 - val_acc: 0.7465
Epoch 15/30
41/41 [==============================] - 49s 1s/step - loss: 0.6481 - acc: 0.7739 - val_loss: 0.5801 - val_acc: 0.8038
```

Fit the Model ¶

```
res = model.fit_generator(train_transform,steps_per_epoch=2527//64,validation_steps=782//64,epochs=30,
                          validation_data=test_transform)
```

```
Epoch 18/30
41/41 [==============================] - 52s 1s/step - loss: 0.6438 - acc: 0.7567 - val_loss: 0.5998 - val_acc: 0.7587
Epoch 19/30
41/41 [==============================] - 51s 1s/step - loss: 0.6158 - acc: 0.7782 - val_loss: 0.4134 - val_acc: 0.8594
Epoch 20/30
41/41 [==============================] - 52s 1s/step - loss: 0.4948 - acc: 0.8250 - val_loss: 0.4548 - val_acc: 0.8455
Epoch 21/30
41/41 [==============================] - 50s 1s/step - loss: 0.4633 - acc: 0.8358 - val_loss: 0.3721 - val_acc: 0.8472
Epoch 22/30
41/41 [==============================] - 50s 1s/step - loss: 0.4586 - acc: 0.8412 - val_loss: 0.3870 - val_acc: 0.8594
Epoch 23/30
41/41 [==============================] - 49s 1s/step - loss: 0.4216 - acc: 0.8470 - val_loss: 0.3280 - val_acc: 0.8819
Epoch 24/30
41/41 [==============================] - 45s 1s/step - loss: 0.3834 - acc: 0.8611 - val_loss: 0.3725 - val_acc: 0.8819
Epoch 25/30
41/41 [==============================] - 44s 1s/step - loss: 0.3818 - acc: 0.8644 - val_loss: 0.2590 - val_acc: 0.9062
Epoch 26/30
41/41 [==============================] - 42s 1s/step - loss: 0.3715 - acc: 0.8593 - val_loss: 0.2497 - val_acc: 0.9271
Epoch 27/30
41/41 [==============================] - 45s 1s/step - loss: 0.3030 - acc: 0.8954 - val_loss: 0.2159 - val_acc: 0.9323
Epoch 28/30
41/41 [==============================] - 43s 1s/step - loss: 0.2889 - acc: 0.9078 - val_loss: 0.1847 - val_acc: 0.9479
Epoch 29/30
41/41 [==============================] - 48s 1s/step - loss: 0.2697 - acc: 0.9050 - val_loss: 0.1973 - val_acc: 0.9410
Epoch 30/30
41/41 [==============================] - 43s 1s/step - loss: 0.2579 - acc: 0.9134 - val_loss: 0.1926 - val_acc: 0.9306
```

Saving the Model

```
model.save('Garbage1.h5')
```

```
#import numpy library
import numpy as np
#import load_model method to load our saved model
from tensorflow.keras.models import load_model
#import image from keras.preprocessing
from tensorflow.keras.preprocessing import image
#loading our saved model file
model = load_model("Garbage1.h5")
```

```
img = image.load_img(r"I:\SmartBridge Projects\Garbage-waste-prediction\glass17.jpg",
                     target_size=(128,128))

x=image.img_to_array(img) #converting in to array format

x=np.expand_dims(x,axis=0) #changing its dimensions as per our requirement
#img_data=preprocess_input(x)
#img_data.shape
```

```
a=np.argmax(model.predict(x), axis=1)
```

```
index=['0', '1', '2', '3', '4','5']
result = str(index[a[0]])
result
```

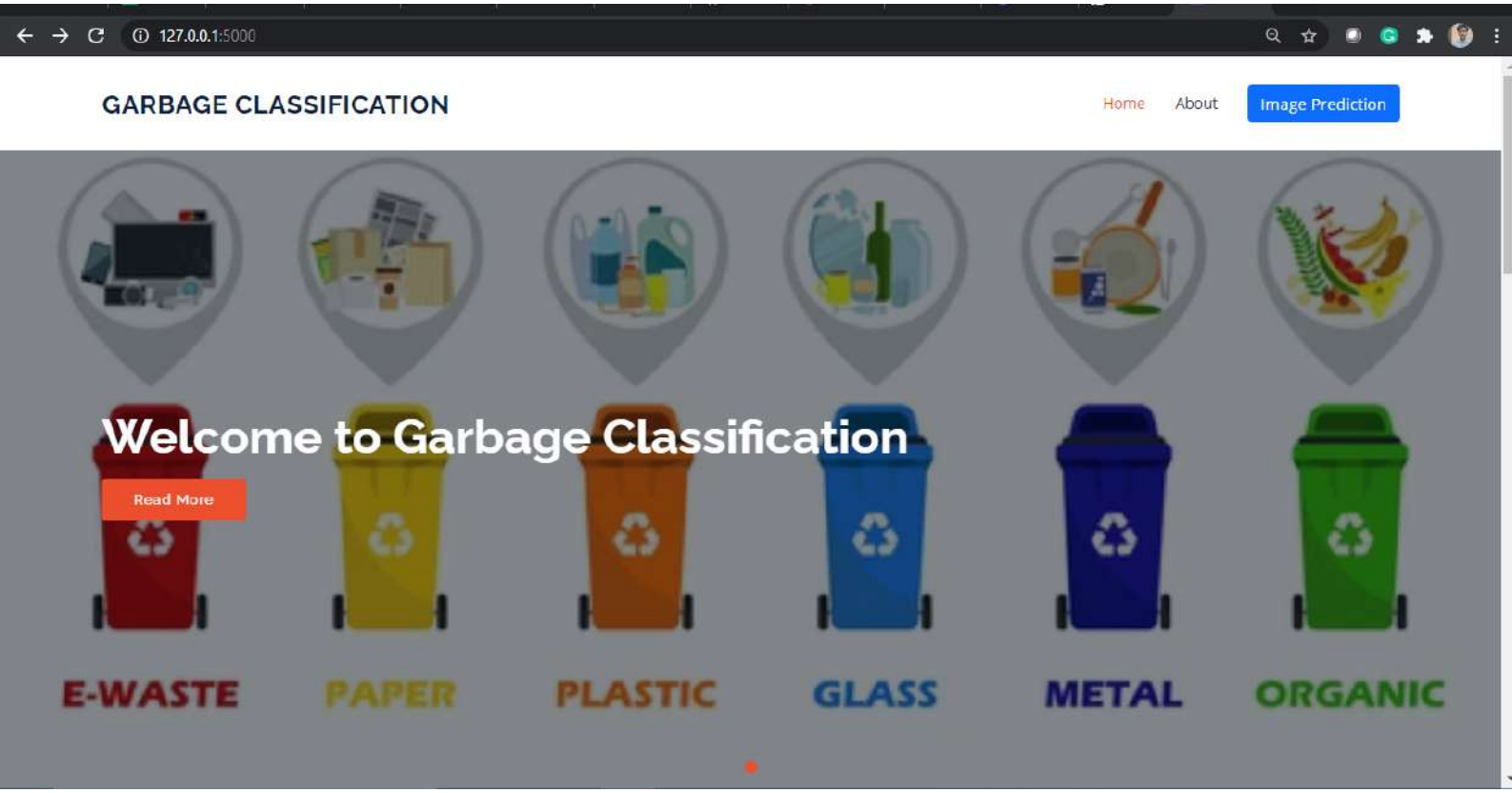
```
'3'
```

```
train_transform.class_indices
```

```
{'cardboard': 0, 'glass': 1, 'metal': 2, 'paper': 3, 'plastic': 4, 'trash': 5}
```

```
index1=['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash ']
result1=str(index1[a[0]])
result1
```

```
'paper'
```


127.0.0.1:5000
GARBAGE CLASSIFICATION
Home
About
Image Prediction
Welcome to Garbage Classification
Read More
E-WASTE
PAPER
PLASTIC
GLASS
METAL
ORGANIC

GARBAGE CLASSIFICATION

Home About Image Prediction

ABOUT

ABOUT PROJECT

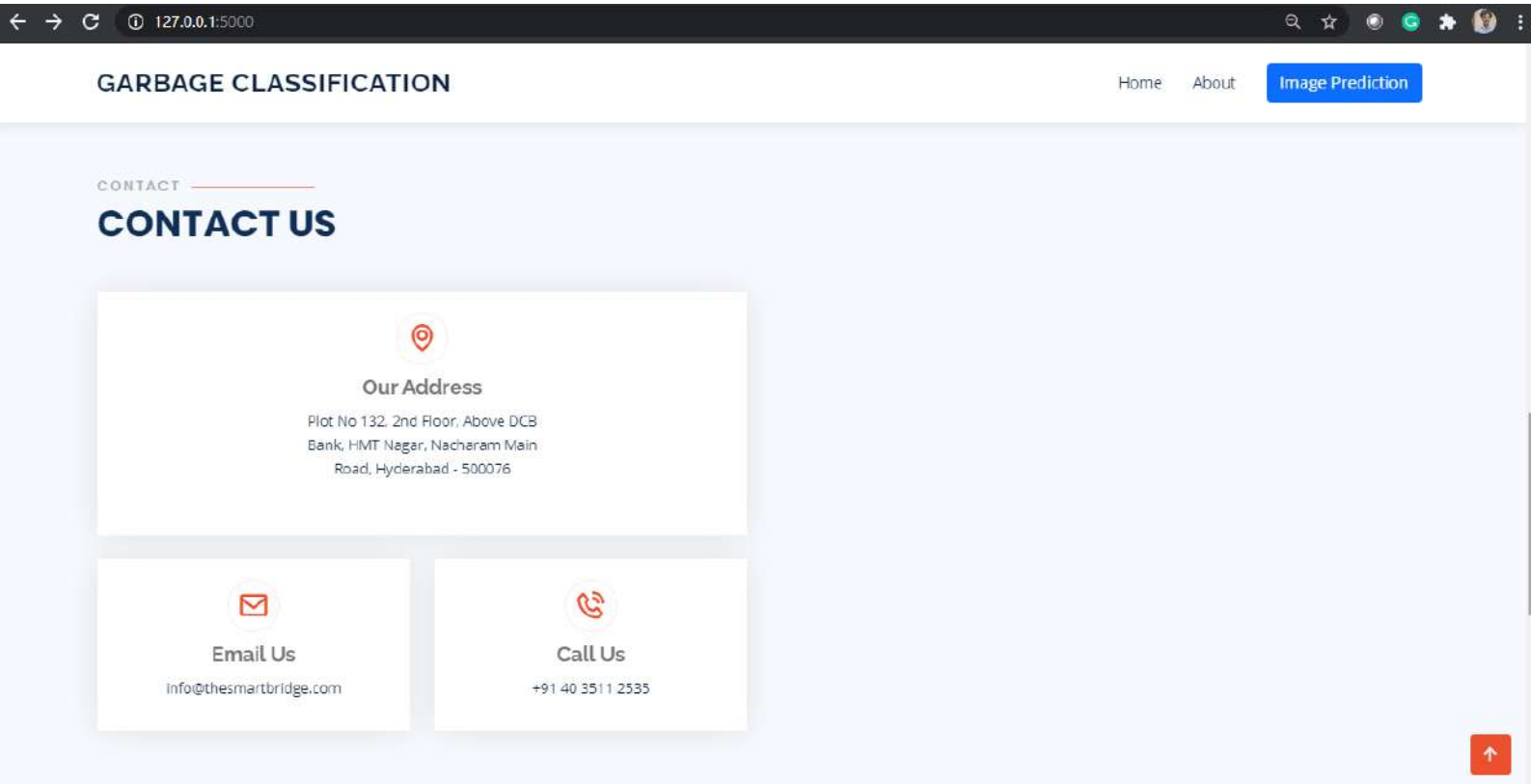
Problem:

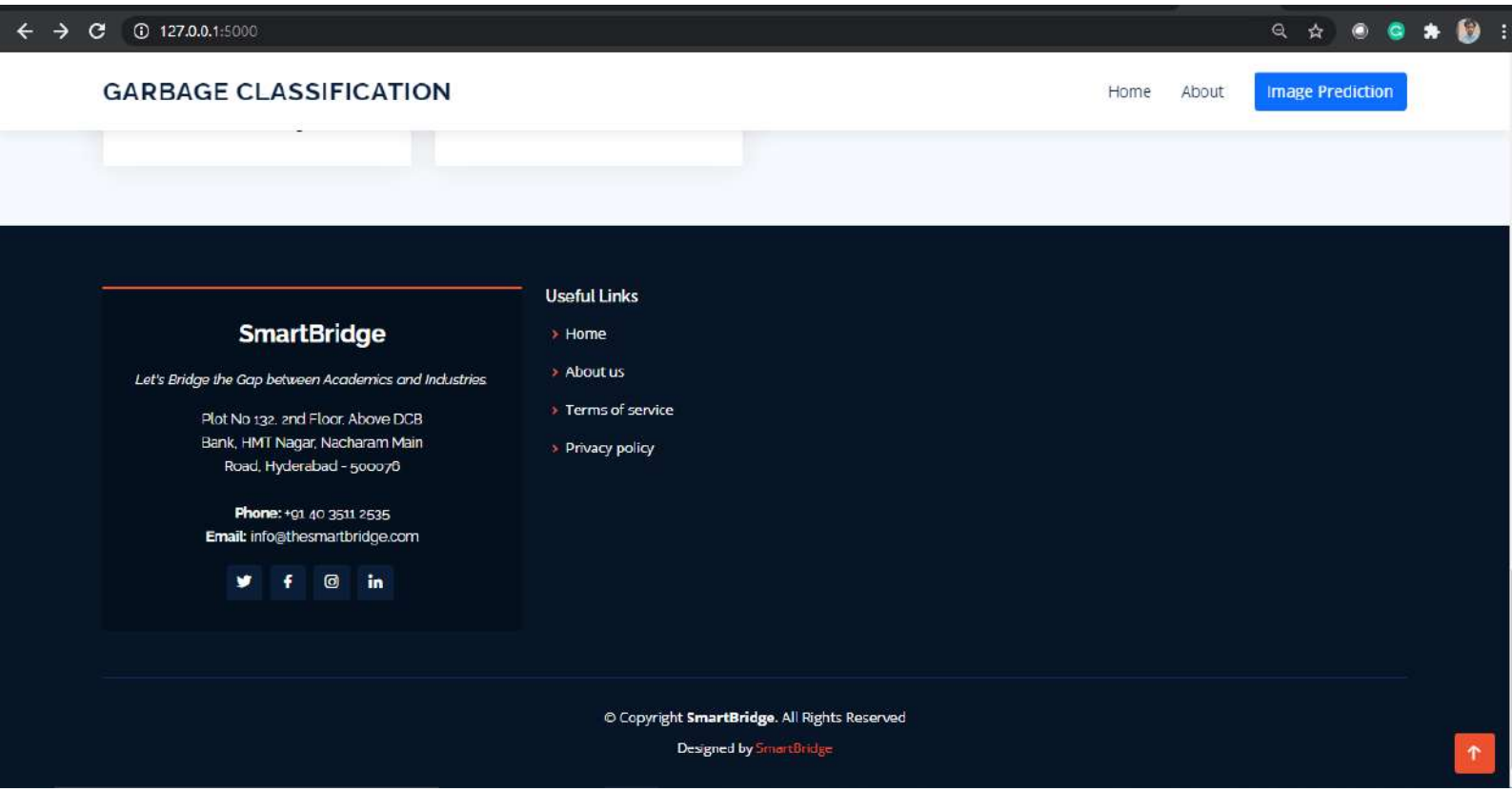
"The accumulation of solid waste in the urban area is becoming a great concern, and it would result in environmental pollution and may be hazardous to human health if it is not properly managed. It is important to have an advanced/intelligent waste management system to manage a variety of waste materials. One of the most important steps of waste management is the separation of the waste into the different components and this process is normally done manually by hand-picking. To simplify the process, we propose an intelligent waste material classification system, which is developed by using the 50-layer residual net pre-train (ResNet-50) Convolutional Neural Network model which is a machine learning tool and serves as the extractor, and Support Vector Machine (SVM) which is used to classify the waste into different groups/types such as glass, metal, paper, and plastic etc. The proposed system is tested on the trash image dataset which was developed by Gary Thung and Mindy Yang, and is able to achieve an accuracy of 87% on the dataset. The separation process of the waste will be faster and intelligent using the proposed waste material classification system without or reducing human involvement."

Solution:

"The present way of separating waste/garbage is the hand-picking method, whereby someone is employed to separate out the different objects/materials. The person, who separate waste, is prone to diseases due to the harmful substances in the garbage. With this in mind, it motivated us to develop an automated system which is able to sort the waste, and this system can take short time to sort the waste, and it will be more accurate in sorting than the manual way. With the system in place, the beneficial separated waste can still be recycled and converted to energy and fuel for the growth of the economy. The system that is developed for the separation of the accumulated waste is based on the combination of Convolutional Neural Network."

Learn More


127.0.0.1:5000
GARBAGE CLASSIFICATION
Home
About
Image Prediction
CONTACT
CONTACT US
Our Address
Plot No 132, 2nd Floor, Above DCB Bank, HMT Nagar, Nacharam Main Road, Hyderabad - 500076
Email Us
info@thesmartbridge.com
Call Us
+91 40 3511 2535


127.0.0.1:5000
GARBAGE CLASSIFICATION
Home
About
Image Prediction
SmartBridge
Let's Bridge the Gap between Academics and Industries
Plot No 132, 2nd Floor, Above DCB Bank, HMT Nagar, Nacharam Main Road, Hyderabad - 500076
Phone: +91 40 3511 2535
Email: info@thesmartbridge.com
Useful Links
Home
About us
Terms of service
Privacy policy
© Copyright SmartBridge. All Rights Reserved
Designed by SmartBridge

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

```python
@app.route('/', methods=['GET'])
def index():
    # Main page
    return render_template('index.html')

@app.route('/Image',methods=['POST','GET'])
def prediction(): # route which will take you to the prediction page
    return render_template('base.html')
```

```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

```
img = image.load_img(r"I:\SmartBridge Projects\Garbage-waste-prediction\glass17.jpg",
                     target_size=(128,128))

x=image.img_to_array(img) #converting in to array format

x=np.expand_dims(x,axis=0) #changing its dimensions as per our requirement
#img_data=preprocess_input(x)
#img_data.shape
```

```
a=np.argmax(model.predict(x), axis=1)
```

```
index=['0', '1', '2', '3', '4','5']
result = str(index[a[0]])
result
```

```
'3'
```

```
train_transform.class_indices
```

```
{'cardboard': 0, 'glass': 1, 'metal': 2, 'paper': 3, 'plastic': 4, 'trash': 5}
```

```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

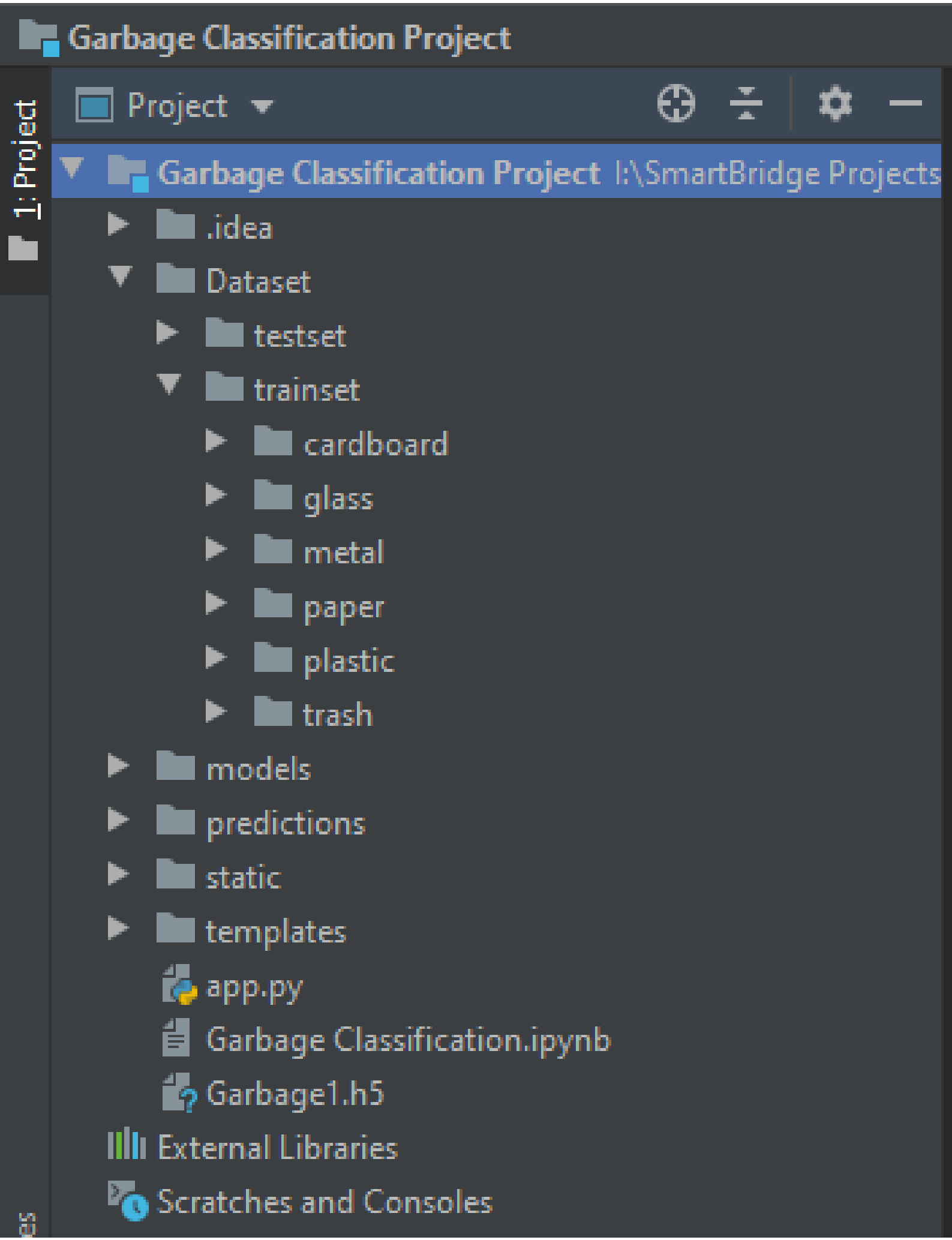
```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

```python
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text

if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```


Garbage Classification Project
Project
1: Project
Garbage Classification Project I:\SmartBridge Projects
.idea
Dataset
testset
trainset
cardboard
glass
metal
paper
plastic
trash
models
predictions
static
templates
app.py
Garbage Classification.ipynb
Garbage1.h5
External Libraries
Scratches and Consoles