Garbage Classification (keras + Transfer Learning)

SUBHAJIT CHATTERJEE

AD20216803

Table of Contents

Introduction

- Garbage management leads to the demolition of waste conducted by recycling and landfilling, it has had a significant impact on the increase in the environmental growth and pollution control.
- Garbage management is a daily task in urban areas, which requires a large amount of labor resources and affects natural, budgetary, efficiency, and social aspects.
- Many approaches have been proposed to optimize waste management, such as using the nearest neighbor, colony optimization, genetic algorithm etc. However results are not up to the mark.
- In this project, we designed an Xception transfer learning model that can classify garbage images.
- Transfer learning means that instead of our model learning everything from scratch, it uses another model that was trained on a similar problem, so that we can "transfer" the learned "knowledge" of the pretrained model to your model, and then learn some new features.

Initial Setup Libraries

- Keras : It is an open-source software library that provides a Python interface for artificial neural networks.
- Numpy : Fundamental package for scientific computing in python.
- Pandas : It an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.
- Matplotlib : Data animated visualization

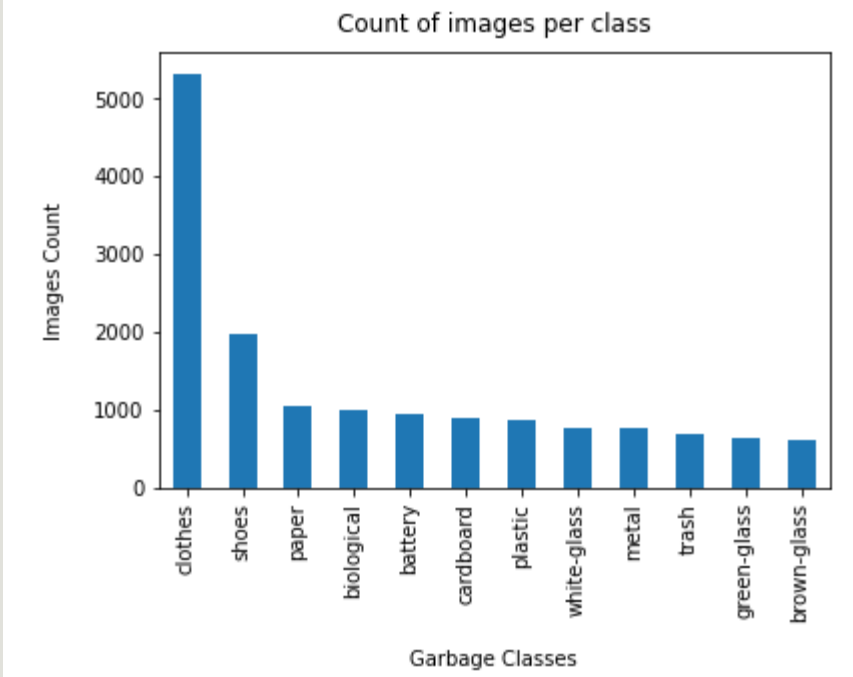
- Along with traditional libraries imported for tensor manipulation, mathematical operations and graphics, one scikit-learn module (confusion_matrix for performance metric assessment) and specific **Keras image preprocessing** (ImageDataGenerator) and **deep learning** objects (Sequential, Conv2D, MaxPooling2D, Flatten, Dense) are used in this exercise.

Dataset and Data Analysis

- Number of total images : 15515
- Dataset having 12 classes

- categories of 12 classes :

```
# Dictionary to save our 12 classes
categories = {0: 'paper', 1: 'cardboard', 2: 'plastic', 3: 'metal', 4: 'trash', 5: 'battery',
              6: 'shoes', 7: 'clothes', 8: 'green-glass', 9: 'brown-glass', 10: 'white-glass',
              11: 'biological'}
```



Visualize the Categories Distribution

Define Constants

IMAGE_WIDTH : 320
IMAGE_HEIGHT : 320
IMAGE_CHANNELS : 3 (R, G, and B)

```
IMAGE_WIDTH = 320
IMAGE_HEIGHT = 320
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS = 3
```

Critical hyperparameters

'epochs': Number of epochs (leaning iterations through which the whole dataset is exposed to the machine for weight updates);
'batch_size': Size of image batches;
'feature_maps': Reference number of feature maps generated by convolutional layers;
'channels' and 'mode': Number of channels utilized in the learning process. For colored RGB images, channels = 3 and mode = 'rgb‘.

Create Data Frame

Data frame that has in one column the filenames of all images and in the other column the corresponding category.

Open the directories in the dataset one by one, save the filenames in the filenames_list and add the corresponding category in the categories_list

```
def add_class_name_prefix(df, col_name):
    df[col_name] = df[col_name].apply(lambda x: x[:re.search("\d",x).start()] + '/' + x)
    return df

# list conatining all the filenames in the dataset
filenames_list = []
# list to store the corresponding category, note that each folder of the dataset has one class of data
categories_list = []

for category in categories:
    filenames = os.listdir(base_path + categories[category])

    filenames_list = filenames_list  +filenames
    categories_list = categories_list + [category] * len(filenames)

df = pd.DataFrame({
    'filename': filenames_list,
    'category': categories_list
})

df = add_class_name_prefix(df, 'filename')

# Shuffle the dataframe
df = df.sample(frac=1).reset_index(drop=True)

print('number of elements = ' , len(df))
```

Model

The steps are:

1. Create a xception model without the last layer and load the ImageNet pretrained weights
2. Add a pre-processing layer
3. Add a pooling layer followed by a softmax layer at the end

Trainable parameters are the number of, well, trainable elements in our network; neurons that are affected by backpropagation.

Non-trainable params ones are e.g. Batch Normalization or when you lock/freeze layers e.g. during transfer learning.

```
model = Sequential()
model.add(keras.Input(shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))

#create a custom layer to apply the preprocessing
def xception_preprocessing(img):
  return xception.preprocess_input(img)

model.add(Lambda(xception_preprocessing))

model.add(xception_layer)
model.add(tf.keras.layers.GlobalAveragePooling2D())
model.add(Dense(len(categories), activation='softmax'))

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 320, 320, 3)       0
_________________________________________________________________
xception (Functional)        (None, 10, 10, 2048)      20861480
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 12)                24588
=================================================================
Total params: 20,886,068
Trainable params: 24,588
Non-trainable params: 20,861,480
_________________________________________________________________
```

Data Split

Here, data split the training set into three separate sets:

The training set : Used to train the model.

The validation set : used to double check that the model is not overfitting the training set, i.e. it can also generalize to other data other than the train data

The Test set : Used to estimate the accuracy of the model on new data.

They split the data set as follows: 80% train set, 10% validation set, and 10% test set.

First, dataset is divided into 80-20 as training and validation.

Last, validation set is divided into 50-50 as validation and test set.

```
df["category"] = df["category"].replace(categories)

# We first split the data into two sets and then split the validate_df to two sets
train_df, validate_df = train_test_split(df, test_size=0.2, random_state=42)
validate_df, test_df = train_test_split(validate_df, test_size=0.5, random_state=42)

train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
test_df = test_df.reset_index(drop=True)

total_train = train_df.shape[0]
total_validate = validate_df.shape[0]

print('train size = ', total_validate , 'validate size = ', total_validate, 'test size = ', test
_df.shape[0])
```

Training

.fit_generator is used when either we have a huge dataset to fit into our memory or when data augmentation needs to be applied.

EPOCHS : 100

steps_per_epoch : It is the **quotient of total training samples by batch size chosen**. As the batch size for the dataset increases the steps per epoch reduce simultaneously and vice-versa. The total number of steps before declaring one epoch finished and starting the next epoch.

start = datetime.datetime.now() and end = datetime.datetime.now() : Calculation training time.

```
EPOCHS = 100

import datetime
import time
start = datetime.datetime.now()

history = model.fit_generator(
    train_generator,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,
    callbacks=callbacks
)

end = datetime.datetime.now()
elapsed = end-start
print('Time :',elapsed)
```

Training Time :

```
Epoch 93/100
193/193 [==============================] - 85s 441ms/step - loss: 0.0011 - categorical_accuracy: 0.9998 - val_loss: 0.2149 - val_categorical_accuracy: 0.9557
Epoch 94/100
193/193 [==============================] - 84s 437ms/step - loss: 0.0012 - categorical_accuracy: 0.9998 - val_loss: 0.2147 - val_categorical_accuracy: 0.9551
Epoch 95/100
193/193 [==============================] - 85s 438ms/step - loss: 0.0012 - categorical_accuracy: 0.9997 - val_loss: 0.2104 - val_categorical_accuracy: 0.9577
Epoch 96/100
193/193 [==============================] - 84s 437ms/step - loss: 0.0012 - categorical_accuracy: 0.9997 - val_loss: 0.2284 - val_categorical_accuracy: 0.9557
Epoch 97/100
193/193 [==============================] - 85s 439ms/step - loss: 0.0011 - categorical_accuracy: 0.9998 - val_loss: 0.2234 - val_categorical_accuracy: 0.9544
Epoch 98/100
193/193 [==============================] - 84s 436ms/step - loss: 0.0011 - categorical_accuracy: 0.9997 - val_loss: 0.2118 - val_categorical_accuracy: 0.9570
Epoch 99/100
193/193 [==============================] - 84s 434ms/step - loss: 0.0010 - categorical_accuracy: 0.9998 - val_loss: 0.2192 - val_categorical_accuracy: 0.9564
Epoch 100/100
193/193 [==============================] - 85s 438ms/step - loss: 0.0011 - categorical_accuracy: 0.9997 - val_loss: 0.2174 - val_categorical_accuracy: 0.9577
Time : 2:22:10.082857
```

Saving model weight :

```
model.save_weights("model.h5")
```

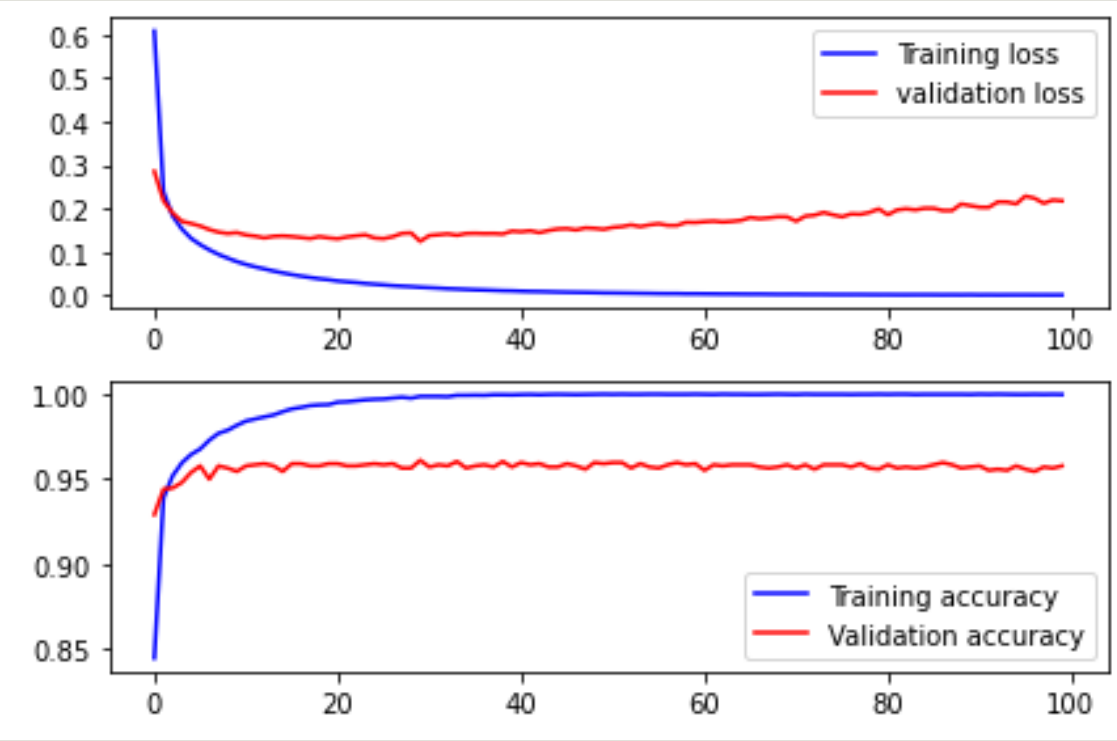
Using save_weights () method we can simply save the weights of all the layers using the save_weights () method.

It saves the weights of the layers contained in the model. It is advised to use the save () method to save h5 models instead of save_weights () method for saving a model using tensorflow.

Visualize the training process

Training and validation loss over time

Training and validation accuracy over time



Evaluate the test

Recognition rate :

```
filenames = test_generator.filenames
nb_samples = len(filenames)

_, accuracy = model.evaluate_generator(test_generator, nb_samples)

print('accuracy on test set = ',  round((accuracy * 100),2 ), '% ')
```

```
accuracy on test set =  94.65 %
```

Classification report :

	precision	recall	f1-score	support
battery	0.95	0.99	0.97	95
biological	0.98	0.97	0.97	96
brown-glass	0.93	0.90	0.92	61
cardboard	0.96	0.93	0.94	97
clothes	0.99	0.99	0.99	500
green-glass	0.78	0.83	0.80	52
metal	0.88	0.88	0.88	101
paper	0.95	0.95	0.95	105
plastic	0.81	0.80	0.81	99
shoes	0.98	0.96	0.97	199
trash	0.97	0.98	0.98	62
white-glass	0.86	0.89	0.88	85
accuracy			0.95	1552
macro avg	0.92	0.92	0.92	1552
weighted avg	0.95	0.95	0.95	1552

- The table above shows among other info the F1 score of each category.
- In the bottom of the F1 score column notice two numbers accuracy and macro avg.
- Accuracy is the same accuracy that we evaluated above for the test set, it is a weighted average.

Training second time :

```
Epoch 94/100
145/145 [==============================] - 65s 450ms/step - loss: 6.9641e-04 - categorical_accuracy: 1.0000 - val_loss: 0.1931 - val_categorical_accuracy: 0.9577
Epoch 95/100
145/145 [==============================] - 65s 449ms/step - loss: 6.5277e-04 - categorical_accuracy: 1.0000 - val_loss: 0.1905 - val_categorical_accuracy: 0.9570
Epoch 96/100
145/145 [==============================] - 65s 448ms/step - loss: 6.2785e-04 - categorical_accuracy: 1.0000 - val_loss: 0.1916 - val_categorical_accuracy: 0.9590
Epoch 97/100
145/145 [==============================] - 65s 448ms/step - loss: 6.0137e-04 - categorical_accuracy: 1.0000 - val_loss: 0.1851 - val_categorical_accuracy: 0.9596
Epoch 98/100
145/145 [==============================] - 65s 447ms/step - loss: 5.7062e-04 - categorical_accuracy: 1.0000 - val_loss: 0.1911 - val_categorical_accuracy: 0.9596
Epoch 99/100
145/145 [==============================] - 65s 450ms/step - loss: 5.5061e-04 - categorical_accuracy: 1.0000 - val_loss: 0.1914 - val_categorical_accuracy: 0.9577
Epoch 100/100
145/145 [==============================] - 65s 448ms/step - loss: 5.2129e-04 - categorical_accuracy: 1.0000 - val_loss: 0.1952 - val_categorical_accuracy: 0.9590
Time : 1:50:05.353373
```

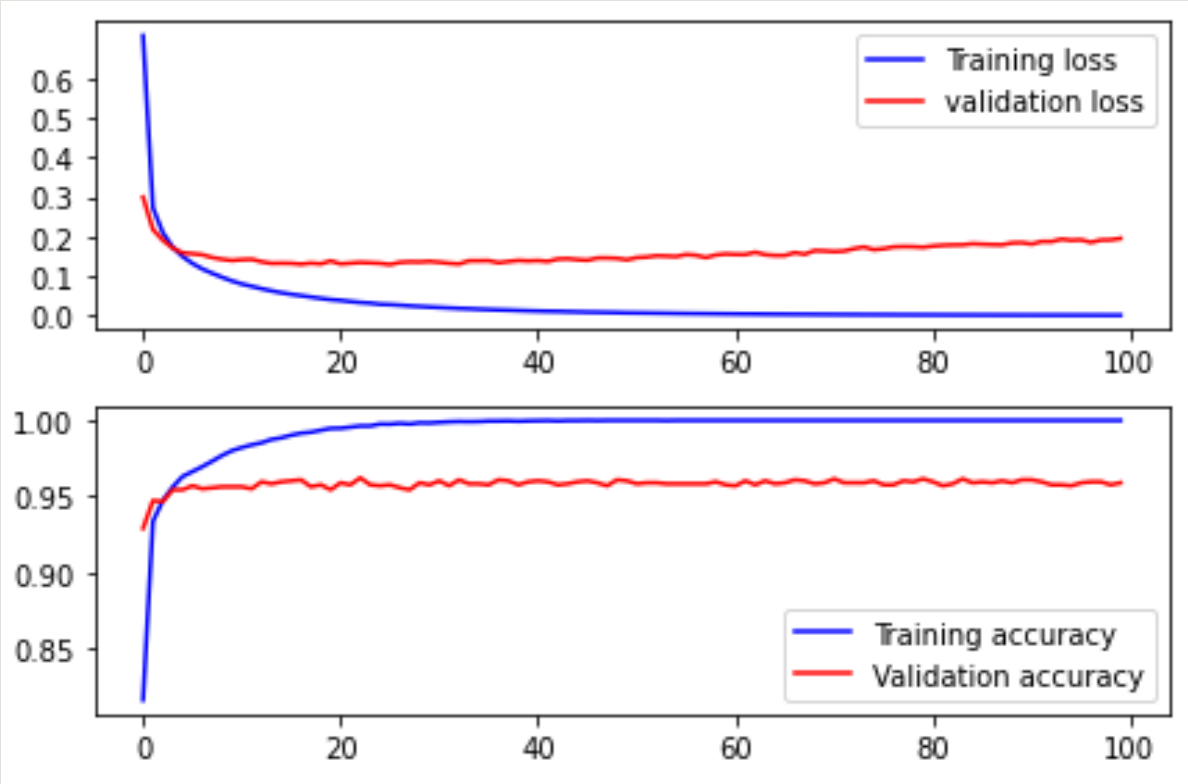
Saving model weight :

```
model.save_weights("model_new.h5")
```

Visualize the training process

Training and validation loss over time

Training and validation accuracy over time



Evaluate the test

Recognition rate :

```
filenames = test_generator.filenames
nb_samples = len(filenames)

_, accuracy = model.evaluate_generator(test_generator, nb_samples)

print('accuracy on test set = ',  round((accuracy * 100),2 ), '% ')
```

```
accuracy on test set =  95.21 %
```

	precision	recall	f1-score	support
battery	0.98	0.99	0.98	293
biological	0.98	0.96	0.97	323
brown-glass	0.90	0.90	0.90	186
cardboard	0.95	0.95	0.95	281
clothes	0.99	0.99	0.99	1566
green-glass	0.89	0.87	0.88	199
metal	0.87	0.90	0.89	214
paper	0.94	0.93	0.94	336
plastic	0.90	0.83	0.86	279
shoes	0.96	0.98	0.97	569
trash	0.97	0.97	0.97	188
white-glass	0.85	0.89	0.87	221
accuracy			0.95	4655
macro avg	0.93	0.93	0.93	4655
weighted avg	0.95	0.95	0.95	4655

Classification report

```
[[ 289    0    0    0    0    0    4    0    0    0    0    0]
 [   0  311    2    0    2    0    1    1    0    5    1    0]
 [   0    1  168    0    0   11    2    0    3    0    0    1]
 [   0    0    0  267    1    0    2   10    1    0    0    0]
 [   0    0    0    2 1548    0    0    0    0   15    0    1]
 [   0    2   11    1    0  174    1    0    3    0    0    7]
 [   4    0    2    1    0    2  193    2    6    0    0    4]
 [   2    2    0    8    4    0    4  313    2    1    0    0]
 [   0    2    1    2    4    6    9    2  232    0    0   21]
 [   0    0    1    1    6    0    1    1    0  558    0    1]
 [   0    0    0    0    1    0    1    0    4    0  182    0]
 [   0    0    1    0    1    3    4    4    7    0    4  197]]
```

Confusion matrix

Conclusion

- In this work, a transfer learning model has been used to classify garbage. The proposed algorithm presents many advantages, as compared to the old garbage classification methods.
- Transfer learning model has proven useful for increasing the speed of training the model and for enhancing the accuracy of the predictions.
- Proposed xception model has achieved 95.21% of accuracy.
- However, the distribution of data in each class is not balanced, accuracy can be improved if we can balance the dataset or apply data augmentation techniques for better classification results.

References

- https://www.kaggle.com/subhajitchatterjee/garbage-classification-keras-transfer-learning
- https://www.hindawi.com/journals/wcmc/2020/6138637/
- Yu Y, Grammenos R. Towards artificially intelligent recycling Improving image processing for waste classification. arXiv preprint arXiv:2108.06274. 2021 Aug 9.
- Rahman MW, Islam R, Hasan A, Bithi NI, Hasan MM, Rahman MM. Intelligent waste management system using deep learning with IoT. Journal of King Saud University-Computer and Information Sciences. 2020 Sep 5.

Thank You