This notebook demonstrates how to leverage transfer learning to use your own image dataset to build and train an image classification model using MXNet and Amazon SageMaker.

We use, as an example, the creation of a trash classification model which, given some image, classifies it into one of three classes: compost, landfill, recycle. This is based on the Show Before You Throw (https://www.youtube.com/watch?v=Ut1VGG6TOOw) project from an AWS DeepLens hackathon and the Smart Recycle Arm (https://www.youtube.com/watch?v=QF0QjRjBwFs) project presented at the AWS Public Sector Summit 2019

Prequisites

- Amazon Sagemaker notebook should have internet access to download images needed for testing this notebook. This is turned ON by default. To explore aoptions review this link : Sagemaker routing options (https://aws.amazon.com/blogs/machine-learning/understanding-amazon-sagemaker-notebook-instance-networking-configurations-and-advanced-routing-options/)
- The IAM role assigned to this notebook should have permissions to create a bucket (if it does not exist)
  - IAM role for Amazon Sagemaker (https://docs.aws.amazon.com/glue/latest/dg/create-an-iam-role-sagemaker-notebook.html)
  - S3 create bucket permissions (https://docs.aws.amazon.com/AmazonS3/latest/dev/using-with-s3-actions.html#using-with-s3-actions-related-to-buckets)

Permissions and environment variables

Here we set up the linkage and authentication to AWS services. There are 2 parts to this:

- The roles used to give learning and hosting access to your data. This will automatically be obtained from the role used to start the notebook
- The Amazon sagemaker image classification docker image which need not be changed

In [46]:
```
import os
import urllib.request
import boto3, botocore


import sagemaker
from sagemaker import get_execution_role

import mxnet as mx
mxnet_path = mx.__file__[ : mx.__file__.rfind('/')]
print(mxnet_path)

role = get_execution_role()
print(role)

sess = sagemaker.Session()

print(sess)
```

```
/home/ec2-user/anaconda3/envs/mxnet_p36/lib/python3.6/site-packages/mxnet
arn:aws:iam::084089099477:role/service-role/AmazonSageMaker-ExecutionRole-20210518T145960
<sagemaker.session.Session object at 0x7f68d9d51710>
```

Amazon S3 bucket info

Enter your Amazon S3 Bucket name where your data will be stored, make sure that your SageMaker notebook has access to this S3 Bucket by granting `S3FullAccess` in the SageMaker role attached to this instance. See here (https://docs.aws.amazon.com/sagemaker/latest/dg/gs-config-permissions.html) for more info.

DeepLens-compatible buckets must start with `deeplens`

In [47]:
```
BUCKET = 'deeplens-wastescan'
PREFIX = 'deeplens-trash-test'
```

In [48]:
```
from sagemaker.amazon.amazon_estimator import get_image_uri
training_image = get_image_uri(sess.boto_region_name, 'image-classification', repo_version="latest")
print (training_image)
```

```
The method get_image_uri has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
Defaulting to the only supported framework/algorithm version: 1. Ignoring framework/algorithm version: latest.

813361260812.dkr.ecr.eu-central-1.amazonaws.com/image-classification:1
```

We are going to check if we have the right bucket and if we have the right permissions.

Please make sure that the result from this cell is "Bucket access is Ok"

In [49]:
```
test_data = 'TestData'
s3 = boto3.resource('s3')
object = s3.Object(BUCKET, PREFIX+"/test.txt")
try:
    object.put(Body=test_data)
except botocore.exceptions.ClientError as e:
    if e.response['Error']['Code'] == "AccessDenied":
        #cannot write on the bucket
        print("Bucket "+BUCKET+"is not writeable, make sure you have the right permissi
ons")
    else:
        if e.response['Error']['Code'] == "NoSuchBucket":
            #Bucket does not exist
            print("Bucket"+BUCKET+" does not exist")
        else:
            raise
else:
    print("Bucket access is Ok")
    object.delete(Bucket=BUCKET, Key=PREFIX+"/test.txt")
```

```
Bucket access is Ok
```

Prepare data

It is assumed that your custom dataset's images are present in an S3 bucket and that different classes are separated by named folders, as shown in the following directory structure:

```
|-deeplens-bucket
   |-deeplens-trash

    |-images

        |-Compost

        |-Landfill

        |-Recycle
```

Since we are providing the data for you in this example, first we'll download the example data, unzip it and upload it to your bucket.

In [50]:
```
!wget https://deeplens-public.s3.amazonaws.com/samples/deeplens-trash/trash-images.zip
```

```
--2021-05-26 04:29:58--  https://deeplens-public.s3.amazonaws.com/samples/deeplens-tras
h/trash-images.zip
Resolving deeplens-public.s3.amazonaws.com (deeplens-public.s3.amazonaws.com)... 52.21
6.237.107
Connecting to deeplens-public.s3.amazonaws.com (deeplens-public.s3.amazonaws.com)|52.21
6.237.107|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 253515836 (242M) [application/zip]
Saving to: ‘trash-images.zip’

trash-images.zip    100%[===================>] 241.77M  16.4MB/s    in 15s

2021-05-26 04:30:14 (15.9 MB/s) - ‘trash-images.zip’ saved [253515836/253515836]
```

In [51]:
```
!rm -rf data/ && mkdir -p data
!mkdir -p data/images
!unzip -qq trash-images.zip -d data/images
!rm trash-images.zip
```

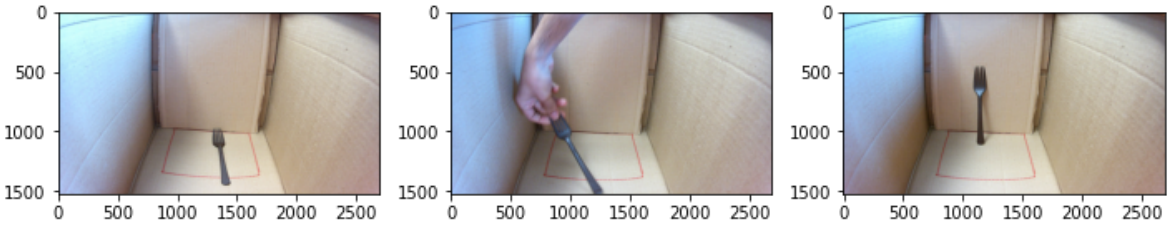
In [52]:
```
ls data/images
```

```
Compost/  Landfill/  Recycling/
```

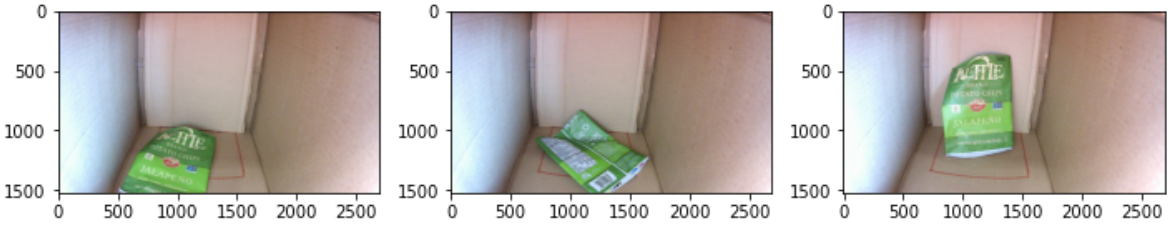
In [53]:
```
import matplotlib.pyplot as plt
%matplotlib inline

def show_images(item_name, images_to_show=-1):
    _im_list = !ls $IMAGES_DIR/$item_name

    NUM_COLS = 3
    if images_to_show == -1:
        IM_COUNT = len(_im_list)
    else:
        IM_COUNT = images_to_show

    print('Displaying images category ' + item_name + ' count: ' + str(IM_COUNT) + ' im
ages.')

    NUM_ROWS = int(IM_COUNT / NUM_COLS)
    if ((IM_COUNT % NUM_COLS) > 0):
        NUM_ROWS += 1

    fig, axarr = plt.subplots(NUM_ROWS, NUM_COLS)
    fig.set_size_inches(10.0, 10.0, forward=True)

    curr_row = 0
    for curr_img in range(IM_COUNT):
        # fetch the url as a file type object, then read the image
        f = IMAGES_DIR + item_name + '/' + _im_list[curr_img]
        a = plt.imread(f)

        # find the column by taking the current index modulo 3
        col = curr_img % NUM_ROWS
        # plot on relevant subplot
        if NUM_ROWS == 1:
            axarr[curr_row].imshow(a)
        else:
            axarr[col, curr_row].imshow(a)
        if col == (NUM_ROWS - 1):
            # we have finished the current row, so increment row counter
            curr_row += 1

    fig.tight_layout()
    plt.show()

    # Clean up
    plt.clf()
    plt.cla()
    plt.close()
```

In [54]:
```
IMAGES_DIR = 'data/images/'
show_images("Compost", images_to_show=3)
show_images("Landfill", images_to_show=3)
show_images("Recycling", images_to_show=3)
```

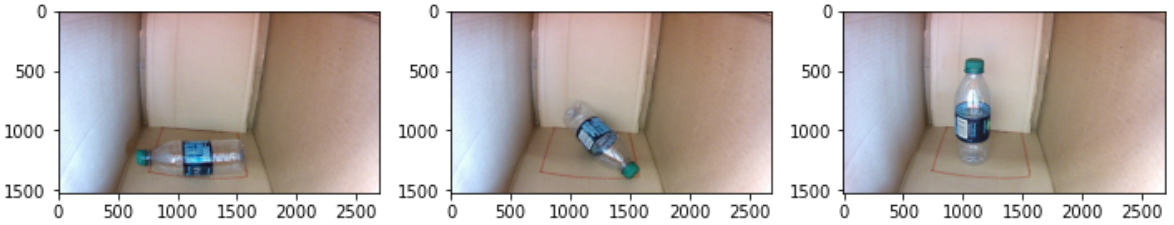
```
Displaying images category Compost count: 3 images.
```



```
Displaying images category Landfill count: 3 images.
```



```
Displaying images category Recycling count: 3 images.
```



In [55]:
```
DEST_BUCKET = 's3://'+BUCKET+'/'+PREFIX+'/images/'
```

In [56]:
```
!aws s3 cp --recursive data/images $DEST_BUCKET --quiet
```

Ensure that the newly created directories containing the downloaded data are structured as shown at the beginning of this tutorial.

In [57]:
```
!aws s3 ls $DEST_BUCKET
```

```
                           PRE Compost/
                           PRE Landfill/
                           PRE Recycling/
```

Prepare "list" files with train-val split

The image classification algorithm can take two types of input formats. The first is a RecordIO format (https://mxnet.apache.org/api/faq/recordio) (content type: application/x-recordio) and the other is a Image list format (.lst file). These file formats allows for efficient loading of images when training the model. In this example we will be using the Image list format (.lst file). A .lst file is a tab-separated file with three columns that contains a list of image files. The first column specifies the image index, the second column specifies the class label index for the image, and the third column specifies the relative path of the image file. The RecordIO file contains the actual pixel data for the images.

To be able to create the .rec files, we first need to split the data into training and validation sets (after shuffling) and create two list files for each. Here our split into train, validation and test (specified by the `0.7` parameter below for test). We keep 0.02% to test the model.

The image and lst files will be converted to RecordIO file internally by the image classification algorithm. But if you want do the conversion, the following cell shows how to do it using the im2rec (https://github.com/apache/incubator-mxnet/blob/master/tools/im2rec.py) tool. Note that this is just an example of creating RecordIO files. We are ***not*** using them for training in this notebook.

In [58]:
```
!python $mxnet_path/tools/im2rec.py --list --recursive --test-ratio=0.02 --train-ratio
0.7 trash data/images
```

```
Compost 0
Landfill 1
Recycling 2
```

Save lst files to S3

Training models is easy with Amazon SageMaker. When you're ready to train in SageMaker, simply specify the location of your data in Amazon S3, and indicate the type and quantity of SageMaker ML instances you need. SageMaker sets up a distributed compute cluster, performs the training, outputs the result to Amazon S3, and tears down the cluster when complete. To use Amazon Sagemaker training we must first transfer our input data to Amazon S3.

In [59]:
```
s3train_lst = 's3://{}/{}/train_lst/'.format(BUCKET, PREFIX)
s3validation_lst = 's3://{}/{}/validation_lst/'.format(BUCKET, PREFIX)

# upload the lst files to train_lst and validation_lst channels
!aws s3 cp trash_train.lst $s3train_lst --quiet
!aws s3 cp trash_val.lst $s3validation_lst --quiet
```

Retrieve dataset size

Let's see the size of train, validation and test datasets

In [60]:
```
f = open('trash_train.lst', 'r')
train_samples = sum(1 for line in f)
f.close()
f = open('trash_val.lst', 'r')
val_samples = sum(1 for line in f)
f.close()
f = open('trash_test.lst', 'r')
test_samples = sum(1 for line in f)
f.close()
print('train_samples:', train_samples)
print('val_samples:', val_samples)
print('test_samples:', test_samples)
```

```
train_samples: 384
val_samples: 155
test_samples: 10
```

This marks the end of the data preparation phase.

Train the model

Training a good model from scratch can take a long time. Fortunately, we're able to use transfer learning to fine-tune a model that has been trained on millions of images. Transfer learning allows us to train a model to recognize new classes in minutes instead of hours or days that it would normally take to train the model from scratch. Transfer learning requires a lot less data to train a model than from scratch (hundreds instead of tens of thousands).

Fine-tuning the Image Classification Model

Now that we are done with all the setup that is needed, we are ready to train our trash detector. To begin, let us create a `sageMaker.estimator.Estimator` object. This estimator will launch the training job.

Training parameters

There are two kinds of parameters that need to be set for training. The first one are the parameters for the training job. These include:

- **Training instance count**: This is the number of instances on which to run the training. When the number of instances is greater than one, then the image classification algorithm will run in distributed settings.
- **Training instance type**: This indicates the type of machine on which to run the training. Typically, we use GPU instances for these training
- **Output path**: This the s3 folder in which the training output is stored

In [61]:
```
s3_output_location = 's3://{}/{}/output'.format(BUCKET, PREFIX)
ic = sagemaker.estimator.Estimator(training_image,
                                   role,
                                   instance_count=1,
                                   instance_type='ml.p2.xlarge',
                                   volume_size = 50,
                                   max_run = 360000,
                                   input_mode= 'File',
                                   output_path=s3_output_location,
                                   base_job_name='ic-trash')
# sagemaker_session = sess?
```

Apart from the above set of parameters, there are hyperparameters that are specific to the algorithm. These are:

- **num_layers**: The number of layers (depth) for the network. We use 18 in this samples but other values such as 50, 152 can be used.
- **use_pretrained_model**: Set to 1 to use pretrained model for transfer learning.
- **image_shape**: The input image dimensions,'num_channels, height, width', for the network. It should be no larger than the actual image size. The number of channels should be same as the actual image.
- **num_classes**: This is the number of output classes for the new dataset. For us, we have
- **num_training_samples**: This is the total number of training samples. It is set to 15240 for caltech dataset with the current split.
- **mini_batch_size**: The number of training samples used for each mini batch. In distributed training, the number of training samples used per batch will be N * mini_batch_size where N is the number of hosts on which training is run.
- **epochs**: Number of training epochs.
- **learning_rate**: Learning rate for training.
- **top_k**: Report the top-k accuracy during training.
- **resize**: Resize the image before using it for training. The images are resized so that the shortest side is of this parameter. If the parameter is not set, then the training data is used as such without resizing.
- **precision_dtype**: Training datatype precision (default: float32). If set to 'float16', the training will be done in mixed_precision mode and will be faster than float32 mode

In [62]:
```
ic.set_hyperparameters(num_layers=18,
                               use_pretrained_model=1,
                               image_shape = "3,224,224",
                               num_classes=3,
                               mini_batch_size=128,
                               epochs=10,
                               learning_rate=0.01,
                               top_k=2,
                               num_training_samples=train_samples,
                               resize = 224,
                               precision_dtype='float32')
```

Input data specification

Set the data type and channels used for training

In [63]:
```
s3images = 's3://{}/{}/images/'.format(BUCKET, PREFIX)

train_data = sagemaker.inputs.TrainingInput(s3images, distribution='FullyReplicated',
                        content_type='application/x-image', s3_data_type='S3Prefix')
validation_data = sagemaker.inputs.TrainingInput(s3images, distribution='FullyReplicate
d',
                               content_type='application/x-image', s3_data_type='S3Prefi
x')
train_data_lst = sagemaker.inputs.TrainingInput(s3train_lst, distribution='FullyReplica
ted',
                        content_type='application/x-image', s3_data_type='S3Prefix')
validation_data_lst = sagemaker.inputs.TrainingInput(s3validation_lst, distribution='Fu
llyReplicated',
                               content_type='application/x-image', s3_data_type='S3Prefi
x')

data_channels = {'train': train_data, 'validation': validation_data,
                 'train_lst': train_data_lst, 'validation_lst': validation_data_lst}
```

Start the training

Start training by calling the fit method in the estimator

In [64]:
```
ic.fit(inputs=data_channels, logs=True)
```

```
2021-05-26 04:30:34 Starting - Starting the training job...
2021-05-26 04:30:58 Starting - Launching requested ML instancesProfilerReport-162200343
4: InProgress
......
2021-05-26 04:31:58 Starting - Preparing the instances for training.........
2021-05-26 04:33:30 Downloading - Downloading input data...
2021-05-26 04:34:01 Training - Downloading the training image......
2021-05-26 04:34:59 Training - Training image download completed. Training in progress.
Docker entrypoint called with argument(s): train
[05/26/2021 04:34:57 INFO 140660128433984] Reading default configuration from /opt/amaz
on/lib/python3.7/site-packages/image_classification/default-input.json: {'use_pretraine
d_model': 0, 'num_layers': 152, 'epochs': 30, 'learning_rate': 0.1, 'lr_scheduler_facto
r': 0.1, 'optimizer': 'sgd', 'momentum': 0, 'weight_decay': 0.0001, 'beta_1': 0.9, 'bet
a_2': 0.999, 'eps': 1e-08, 'gamma': 0.9, 'mini_batch_size': 32, 'image_shape': '3,224,2
24', 'precision_dtype': 'float32'}
[05/26/2021 04:34:57 INFO 140660128433984] Merging with provided configuration from /op
t/ml/input/config/hyperparameters.json: {'num_classes': '3', 'top_k': '2', 'num_trainin
g_samples': '384', 'resize': '224', 'use_pretrained_model': '1', 'precision_dtype': 'fl
oat32', 'num_layers': '18', 'epochs': '10', 'image_shape': '3,224,224', 'learning_rat
e': '0.01', 'mini_batch_size': '128'}
[05/26/2021 04:34:57 INFO 140660128433984] Final configuration: {'use_pretrained_mode
l': '1', 'num_layers': '18', 'epochs': '10', 'learning_rate': '0.01', 'lr_scheduler_fac
tor': 0.1, 'optimizer': 'sgd', 'momentum': 0, 'weight_decay': 0.0001, 'beta_1': 0.9, 'b
eta_2': 0.999, 'eps': 1e-08, 'gamma': 0.9, 'mini_batch_size': '128', 'image_shape': '3,
224,224', 'precision_dtype': 'float32', 'num_classes': '3', 'top_k': '2', 'num_training
_samples': '384', 'resize': '224'}
[05/26/2021 04:34:57 INFO 140660128433984] Searching for .lst files in /opt/ml/input/da
ta/train_lst.
[05/26/2021 04:34:57 INFO 140660128433984] Creating record files for trash_train.lst
[05/26/2021 04:35:11 INFO 140660128433984] Done creating record files...
[05/26/2021 04:35:11 INFO 140660128433984] Searching for .lst files in /opt/ml/input/da
ta/validation_lst.
[05/26/2021 04:35:11 INFO 140660128433984] Creating record files for trash_val.lst
[05/26/2021 04:35:16 INFO 140660128433984] Done creating record files...
[05/26/2021 04:35:16 INFO 140660128433984] use_pretrained_model: 1
[05/26/2021 04:35:16 INFO 140660128433984] multi_label: 0
[05/26/2021 04:35:16 INFO 140660128433984] Using pretrained model for initializing weig
hts and transfer learning.
[05/26/2021 04:35:16 INFO 140660128433984] ---- Parameters ----
[05/26/2021 04:35:16 INFO 140660128433984] num_layers: 18
[05/26/2021 04:35:16 INFO 140660128433984] data type: <class 'numpy.float32'>
[05/26/2021 04:35:16 INFO 140660128433984] epochs: 10
[05/26/2021 04:35:16 INFO 140660128433984] image resize size: 224
[05/26/2021 04:35:16 INFO 140660128433984] optimizer: sgd
[05/26/2021 04:35:16 INFO 140660128433984] momentum: 0.9
[05/26/2021 04:35:16 INFO 140660128433984] weight_decay: 0.0001
[05/26/2021 04:35:16 INFO 140660128433984] learning_rate: 0.01
[05/26/2021 04:35:16 INFO 140660128433984] num_training_samples: 384
[05/26/2021 04:35:16 INFO 140660128433984] mini_batch_size: 128
[05/26/2021 04:35:16 INFO 140660128433984] image_shape: 3,224,224
[05/26/2021 04:35:16 INFO 140660128433984] num_classes: 3
[05/26/2021 04:35:16 INFO 140660128433984] augmentation_type: None
[05/26/2021 04:35:16 INFO 140660128433984] kv_store: device
[05/26/2021 04:35:16 INFO 140660128433984] top_k: 2
[05/26/2021 04:35:16 INFO 140660128433984] checkpoint_frequency not set, will store the
best model
[05/26/2021 04:35:16 INFO 140660128433984] --------------------
[04:35:16] /opt/brazil-pkg-cache/packages/AIAlgorithmsMXNet/AIAlgorithmsMXNet-1.3.x_ecl
_Cuda_10.1.x.6753.0/AL2_x86_64/generic-flavor/src/src/nnvm/legacy_json_util.cc:209: Loa
ding symbol saved by previous version v0.8.0. Attempting to upgrade...
[04:35:16] /opt/brazil-pkg-cache/packages/AIAlgorithmsMXNet/AIAlgorithmsMXNet-1.3.x_ecl
_Cuda_10.1.x.6753.0/AL2_x86_64/generic-flavor/src/src/nnvm/legacy_json_util.cc:217: Sym
bol successfully upgraded!
[05/26/2021 04:35:16 INFO 140660128433984] Setting number of threads: 3
[04:35:24] /opt/brazil-pkg-cache/packages/AIAlgorithmsMXNet/AIAlgorithmsMXNet-1.3.x_ecl
_Cuda_10.1.x.6753.0/AL2_x86_64/generic-flavor/src/src/operator/nn/./cudnn/./cudnn_algor
eg-inl.h:97: Running performance tests to find the best convolution algorithm, this can
take a while... (setting env variable MXNET_CUDNN_AUTOTUNE_DEFAULT to 0 to disable)
[05/26/2021 04:35:32 INFO 140660128433984] Epoch[0] Train-accuracy=0.364583
```

```
[05/26/2021 04:35:32 INFO 140660128433984] Epoch[0] Train-top_k_accuracy_2=0.744792
[05/26/2021 04:35:32 INFO 140660128433984] Epoch[0] Time cost=7.654
[05/26/2021 04:35:32 INFO 140660128433984] Epoch[0] Validation-accuracy=0.460938
[05/26/2021 04:35:32 INFO 140660128433984] Storing the best model with validation accur
acy: 0.460938
[05/26/2021 04:35:33 INFO 140660128433984] Saved checkpoint to "/opt/ml/model/image-cla
ssification-0001.params"
[05/26/2021 04:35:35 INFO 140660128433984] Epoch[1] Train-accuracy=0.726562
[05/26/2021 04:35:35 INFO 140660128433984] Epoch[1] Train-top_k_accuracy_2=0.963542
[05/26/2021 04:35:35 INFO 140660128433984] Epoch[1] Time cost=2.143
[05/26/2021 04:35:35 INFO 140660128433984] Epoch[1] Validation-accuracy=0.562500
[05/26/2021 04:35:36 INFO 140660128433984] Storing the best model with validation accur
acy: 0.562500
[05/26/2021 04:35:36 INFO 140660128433984] Saved checkpoint to "/opt/ml/model/image-cla
ssification-0002.params"
[05/26/2021 04:35:38 INFO 140660128433984] Epoch[2] Train-accuracy=0.901042
[05/26/2021 04:35:38 INFO 140660128433984] Epoch[2] Train-top_k_accuracy_2=0.994792
[05/26/2021 04:35:38 INFO 140660128433984] Epoch[2] Time cost=2.134
[05/26/2021 04:35:39 INFO 140660128433984] Epoch[2] Validation-accuracy=0.632812
[05/26/2021 04:35:39 INFO 140660128433984] Storing the best model with validation accur
acy: 0.632812
[05/26/2021 04:35:39 INFO 140660128433984] Saved checkpoint to "/opt/ml/model/image-cla
ssification-0003.params"
[05/26/2021 04:35:41 INFO 140660128433984] Epoch[3] Train-accuracy=0.986979
[05/26/2021 04:35:41 INFO 140660128433984] Epoch[3] Train-top_k_accuracy_2=0.997396
[05/26/2021 04:35:41 INFO 140660128433984] Epoch[3] Time cost=2.121
[05/26/2021 04:35:42 INFO 140660128433984] Epoch[3] Validation-accuracy=0.828125
[05/26/2021 04:35:42 INFO 140660128433984] Storing the best model with validation accur
acy: 0.828125
[05/26/2021 04:35:42 INFO 140660128433984] Saved checkpoint to "/opt/ml/model/image-cla
ssification-0004.params"
[05/26/2021 04:35:44 INFO 140660128433984] Epoch[4] Train-accuracy=0.986979
[05/26/2021 04:35:44 INFO 140660128433984] Epoch[4] Train-top_k_accuracy_2=1.000000
[05/26/2021 04:35:44 INFO 140660128433984] Epoch[4] Time cost=2.120
[05/26/2021 04:35:45 INFO 140660128433984] Epoch[4] Validation-accuracy=0.882812
[05/26/2021 04:35:45 INFO 140660128433984] Storing the best model with validation accur
acy: 0.882812
[05/26/2021 04:35:45 INFO 140660128433984] Saved checkpoint to "/opt/ml/model/image-cla
ssification-0005.params"
[05/26/2021 04:35:48 INFO 140660128433984] Epoch[5] Train-accuracy=1.000000
[05/26/2021 04:35:48 INFO 140660128433984] Epoch[5] Train-top_k_accuracy_2=1.000000
[05/26/2021 04:35:48 INFO 140660128433984] Epoch[5] Time cost=2.162
[05/26/2021 04:35:48 INFO 140660128433984] Epoch[5] Validation-accuracy=0.859375
[05/26/2021 04:35:51 INFO 140660128433984] Epoch[6] Train-accuracy=1.000000
[05/26/2021 04:35:51 INFO 140660128433984] Epoch[6] Train-top_k_accuracy_2=1.000000
[05/26/2021 04:35:51 INFO 140660128433984] Epoch[6] Time cost=2.130
[05/26/2021 04:35:51 INFO 140660128433984] Epoch[6] Validation-accuracy=0.835938

2021-05-26 04:36:05 Uploading - Uploading generated training model[05/26/2021 04:35:54
 INFO 140660128433984] Epoch[7] Train-accuracy=1.000000
[05/26/2021 04:35:54 INFO 140660128433984] Epoch[7] Train-top_k_accuracy_2=1.000000
[05/26/2021 04:35:54 INFO 140660128433984] Epoch[7] Time cost=2.124
[05/26/2021 04:35:54 INFO 140660128433984] Epoch[7] Validation-accuracy=0.867188
[05/26/2021 04:35:57 INFO 140660128433984] Epoch[8] Train-accuracy=1.000000
[05/26/2021 04:35:57 INFO 140660128433984] Epoch[8] Train-top_k_accuracy_2=1.000000
[05/26/2021 04:35:57 INFO 140660128433984] Epoch[8] Time cost=2.147
[05/26/2021 04:35:58 INFO 140660128433984] Epoch[8] Validation-accuracy=0.875000
[05/26/2021 04:36:00 INFO 140660128433984] Epoch[9] Train-accuracy=1.000000
[05/26/2021 04:36:00 INFO 140660128433984] Epoch[9] Train-top_k_accuracy_2=1.000000
[05/26/2021 04:36:00 INFO 140660128433984] Epoch[9] Time cost=2.115
[05/26/2021 04:36:01 INFO 140660128433984] Epoch[9] Validation-accuracy=0.867188

2021-05-26 04:36:19 Completed - Training job completed
Training seconds: 169
Billable seconds: 169
```

The output from the above command will have the model accuracy and the time it took to run the training.

You can also view these details by navigating to `Training -> Training Jobs -> job_name -> View logs` in the

The model trained above can now be found in the `s3://<YOUR_BUCKET>/<PREFIX>/output` path.

In [65]:
```
MODEL_PATH = ic.model_data
print(MODEL_PATH)
```

```
s3://deeplens-wastescan/deeplens-trash-test/output/ic-trash-2021-05-26-04-30-34-638/out
put/model.tar.gz
```

Deploy to a Sagemaker endpoint

After training your model is complete, you can test your model by asking it to predict the class of a sample trash image that the model has not seen before. This step is called inference.

Amazon SageMaker provides an HTTPS endpoint where your machine learning model is available to provide inferences. For more information see the Amazon SageMaker documentation (https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-hosting.html).

In [66]:
```
ic_infer = ic.deploy(initial_instance_count=1, instance_type='ml.p2.xlarge')
```

```
----------------!
```

In [69]:
```
print (ic_infer)
```

```
<sagemaker.predictor.Predictor object at 0x7f6840e77198>
```

Test the images against the endpoint

We will use the test images that were kept aside for testing.

In [70]:
```
object_categories = ['Compost', 'Landfill', 'Recycling']
```

In [74]:

```
from IPython.display import Image, display
import json
import numpy as np


def test_model():
    preds = []
    acts  = []
    num_errors = 0
    with open('trash_test.lst', 'r') as f:
        for line in f:
            stripped_line = str(line.strip()).split("\t")
            file_path = stripped_line[2]
            category = int(float(stripped_line[1]))
            with open(IMAGES_DIR + stripped_line[2], 'rb') as f:
                payload = f.read()
                payload = bytearray(payload)

                #ic_infer.content_type = 'application/x-image'
                result = json.loads(ic_infer.predict(payload, initial_args={'ContentTyp
e':'application/x-image'} ))
            # the result will output the probabilities for all classes
            # find the class with maximum probability and print the class index
            index = np.argmax(result)
            act = object_categories[category]
            pred = object_categories[index]
            conf = result[index]
            print("Result: Predicted: {}, Confidence: {:.2f}, Actual: {} ".format(pred,
conf, act))
            acts.append(category)
            preds.append(index)
            if (pred != act):
                num_errors += 1
                print('ERROR on image -- Predicted: {}, Confidence: {:.2f}, Actual: {}'
.format(pred, conf, act))
            display(Image(filename=IMAGES_DIR + stripped_line[2], width=100, height=100
))

    return num_errors, preds, acts
num_errors, preds, acts = test_model()
```

Result: Predicted: Landfill, Confidence: 0.59, Actual: Compost
ERROR on image -- Predicted: Landfill, Confidence: 0.59, Actual: Compost

Result: Predicted: Compost, Confidence: 0.77, Actual: Compost

Result: Predicted: Compost, Confidence: 0.93, Actual: Compost

Result: Predicted: Landfill, Confidence: 0.91, Actual: Landfill

Result: Predicted: Compost, Confidence: 0.97, Actual: Compost

Result: Predicted: Compost, Confidence: 0.50, Actual: Compost

Result: Predicted: Recycling, Confidence: 1.00, Actual: Recycling

Result: Predicted: Landfill, Confidence: 0.44, Actual: Compost
ERROR on image -- Predicted: Landfill, Confidence: 0.44, Actual: Compost

Result: Predicted: Landfill, Confidence: 0.99, Actual: Landfill

Result: Predicted: Compost, Confidence: 0.97, Actual: Compost

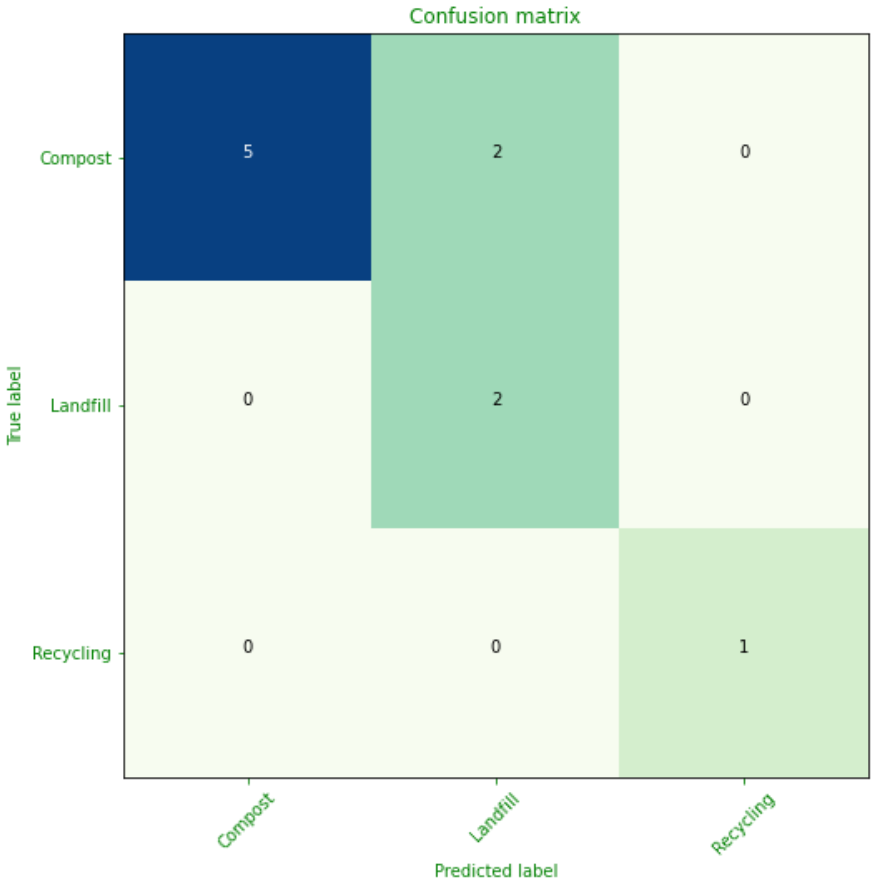
In [75]:
```
from sklearn.metrics import confusion_matrix
import numpy as np
import itertools
COLOR = 'green'
plt.rcParams['text.color'] = COLOR
plt.rcParams['axes.labelcolor'] = COLOR
plt.rcParams['xtick.color'] = COLOR
plt.rcParams['ytick.color'] = COLOR
def plot_confusion_matrix(cm, classes,
                          class_name_list,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.GnBu):
    plt.figure(figsize=(7,7))
    plt.grid(False)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
                                  range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.gca().set_xticklabels(class_name_list)
    plt.gca().set_yticklabels(class_name_list)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

def create_and_plot_confusion_matrix(actual, predicted):
    cnf_matrix = confusion_matrix(actual, np.asarray(predicted),labels=range(len(object
_categories)))
    plot_confusion_matrix(cnf_matrix, classes=range(len(object_categories)), class_name
_list=object_categories)
```

Display confusion matrix showing 'true' and 'predicted' labels

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It's a table with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table). The diagonal values in the table indicate a match between the predicted class and the actual class.

For more details go to Confusion matrix (https://en.wikipedia.org/wiki/Confusion_matrix) (Wikipedia)

In [76]:
```
create_and_plot_confusion_matrix(acts, preds)
```



Approximate costs

As of 03/11/2020 and based on the pricing information displayed on the page: https://aws.amazon.com/sagemaker/pricing/ (https://aws.amazon.com/sagemaker/pricing/), here's the costs you can expect in a 24 hour period:

- Notebook instance cost **$6** Assuming you choose ml.t3.xlarge ($0.233/hour) instance. This can vary based on the size of instance you choose.
- Training costs **$1.05** : Assuming you will run about 10 training runs in a 24 hour period using the sample dataset provided. The notebook uses a p2.xlarge ($1.26/hour) instance
- Model hosting **$6.72** : Assuming you use the ml.m4.xlarge ($0.28/hour) instance running for 24 hours.

NOTE : To save on costs, stop your notebook instances and delete the model edpoint when not in use

(Optional) Clean-up

If you're ready to be done with this notebook, please run the cell below. This will remove the hosted endpoint you created and avoid any charges from a stray instance being left on.

In [ ]:
```
sess.delete_endpoint(ic_infer.endpoint)
print("Completed")
```

Rename model to deploy to AWS DeepLens

The MxNet model that is stored in the S3 bucket contains 2 files: the params file and a symbol.json file. To simplify deployment to AWS DeepLens, we'll modify the params file so that you do not need to specify the number of epochs the model was trained for.

In [77]:
```
import glob
```

In [78]:
```
!rm -rf data/$PREFIX/tmp && mkdir -p data/$PREFIX/tmp
!aws s3 cp $MODEL_PATH data/$PREFIX/tmp
!tar -xzvf data/$PREFIX/tmp/model.tar.gz -C data/$PREFIX/tmp
```

```
download: s3://deeplens-wastescan/deeplens-trash-test/output/ic-trash-2021-05-26-04-30-
34-638/output/model.tar.gz to data/deeplens-trash-test/tmp/model.tar.gz
image-classification-0005.params
image-classification-symbol.json
model-shapes.json
```

In [79]:
```
params_file_name = glob.glob('./data/' + PREFIX + '/tmp/*.params')[0]
```

In [80]:
```
!mv $params_file_name data/$PREFIX/tmp/image-classification-0000.params
```

In [81]:
```
!tar -cvzf ./model.tar.gz -C data/$PREFIX/tmp ./image-classification-0000.params ./imag
e-classification-symbol.json
```

```
./image-classification-0000.params
./image-classification-symbol.json
```

In [82]:
```
!aws s3 cp model.tar.gz $MODEL_PATH
```

```
upload: ./model.tar.gz to s3://deeplens-wastescan/deeplens-trash-test/output/ic-trash-2
021-05-26-04-30-34-638/output/model.tar.gz
```

Next steps

At this point, you have completed:

- Training a model with Amazon Sagemaker using transfer learning

Next you'll deploy this model to AWS DeepLens. If you have started this notebook as part of a tutorial, please go back to the next step in the tutorial. If you have found this notebook through other channels, please go to awsdeeplens.recipes (http://awsdeeplens.recipes) and select the Trash Detector tutorial to continue.

In [ ]:
```
```