Classifying Recyclables

Michael Kulinich & Brandon Kleinman
2328386   & 2291703

The Problem

According to the EPA only 27% of glass and 9% of plastic actually gets recycled. Almost all of the other recyclables end up in the ocean or the atmosphere (through incineration).

Goal: Successfully recognize various recyclables in order to sort

Garbage Patch in the Pacific

Importance

Although most plastics can only be recycled once or twice, glass can be recycled indefinitely. If there is a way to identify recyclables with high accuracy, keeping recyclable waste out of oceans and landfills is an attainable goal.

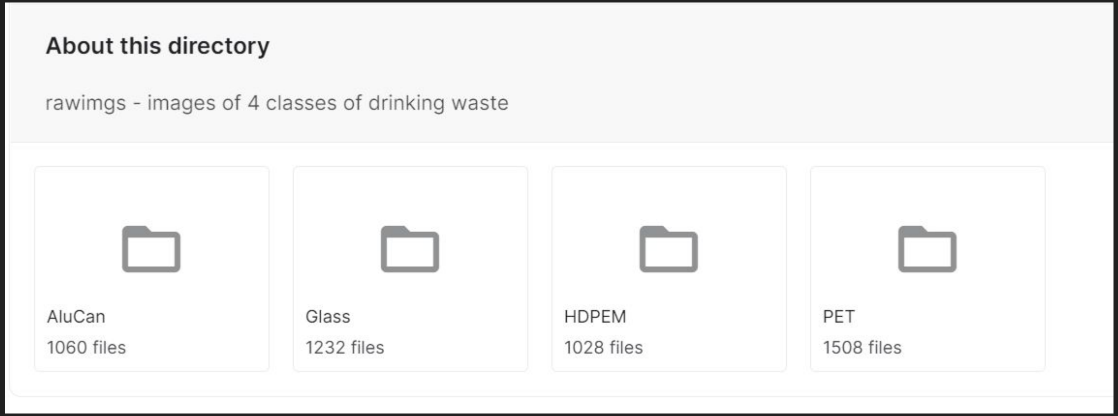
Recycling Facility

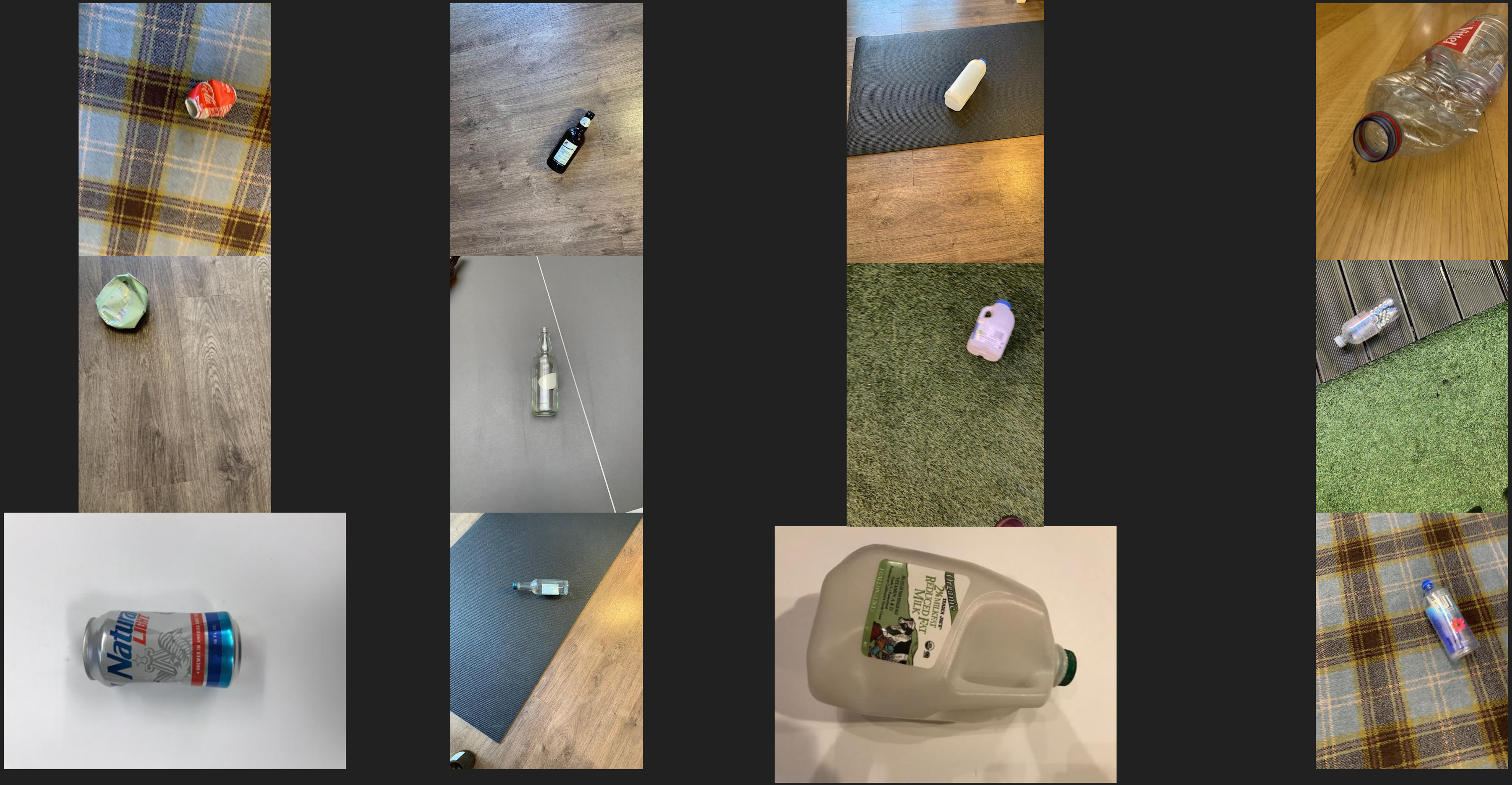
Dataset & Analysis

Our dataset is comprised of images from four different classes: Glass bottles, Aluminum Cans, PET water bottles, and HDPE bottles. We combined two different datasets to create a large enough dataset to train the CNN. The first dataset is from Kaggle and the second is from Portland State University.

Kaggle Dataset

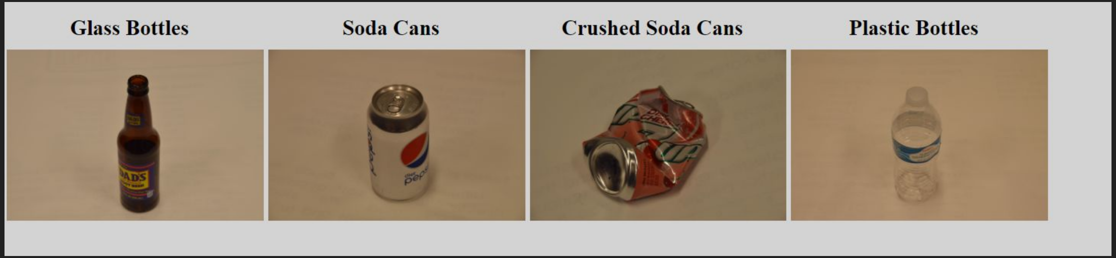
- Pictures were taken on 12 mp camera
- Stored as .jpg




REDUCED FAT
MILK

Portland State University Dataset

- 1840 samples of each
- Stored as .npy (numPy array file on disk)



Data Augmentation

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=70,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
val_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    classes=['AluCan', 'Glass', 'HDPEM', 'PET'],
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical')
validation_generator = val_datagen.flow_from_directory(
    validation_dir,
    classes=['AluCan', 'Glass', 'HDPEM', 'PET'],
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical')
```

The Model

- Sequential
- Convolutional Neural Network
- input shape: 128 x 128
- 6 total layers
- 4 convolutional layers
    - 3x3 kernels
    - Relu activation function
    - Batch Norm
- 2 dense layers
    - Dropout of .5
    - Relu and softmax
    - Batch Norm
- RMSprop optimizer
    - Learning rate .001

```python
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(128, 128, 3)))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(layers.Dense(4, activation='softmax'))
model.compile(loss='categorical_crossentropy',
optimizer=optimizers.RMSprop(lr=1e-3),
metrics=['acc'])
return model
```

The Model

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 126, 126, 32)	896
batch_normalization_9 (Batch	(None, 126, 126, 32)	128
max_pooling2d_16 (MaxPooling	(None, 63, 63, 32)	0
batch_normalization_10 (Batc	(None, 63, 63, 32)	128
conv2d_19 (Conv2D)	(None, 61, 61, 64)	18496
batch_normalization_11 (Batc	(None, 61, 61, 64)	256
max_pooling2d_17 (MaxPooling	(None, 30, 30, 64)	0
batch_normalization_12 (Batc	(None, 30, 30, 64)	256
conv2d_20 (Conv2D)	(None, 28, 28, 128)	73856
batch_normalization_13 (Batc	(None, 28, 28, 128)	512
max_pooling2d_18 (MaxPooling	(None, 14, 14, 128)	0
batch_normalization_14 (Batc	(None, 14, 14, 128)	512
conv2d_21 (Conv2D)	(None, 12, 12, 128)	147584
batch_normalization_15 (Batc	(None, 12, 12, 128)	512
max_pooling2d_19 (MaxPooling	(None, 6, 6, 128)	0
batch_normalization_16 (Batc	(None, 6, 6, 128)	512
flatten_4 (Flatten)	(None, 4608)	0
dropout_6 (Dropout)	(None, 4608)	0
dense_8 (Dense)	(None, 512)	2359808
batch_normalization_17 (Batc	(None, 512)	2048
dense_9 (Dense)	(None, 4)	2052

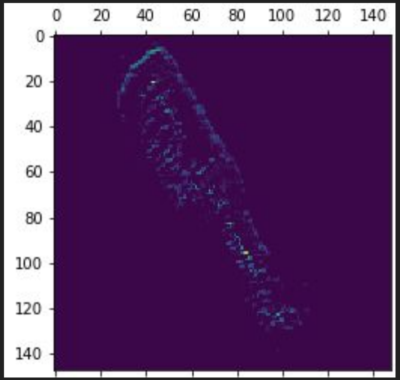
Total params: 2,607,556
Trainable params: 2,605,124
Non-trainable params: 2,432

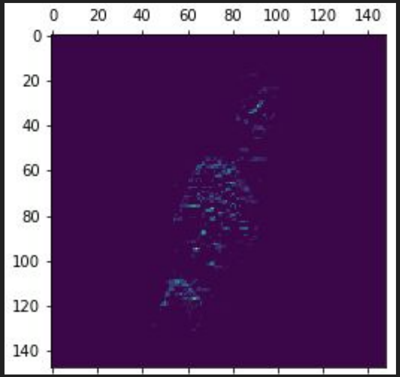
Training The Model

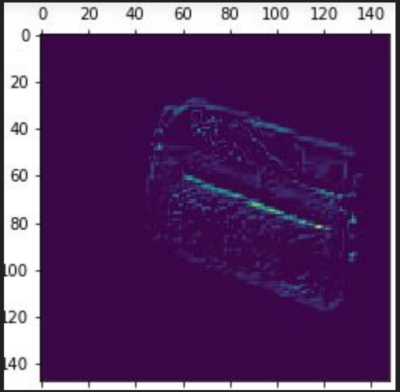
```
filepath="weights.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]

history = model.fit(
    train_generator,
    steps_per_epoch=7900 // 32,
    epochs=100,
    validation_data=validation_generator,
    validation_steps=1750 // 32,
    callbacks=callbacks_list,
    verbose = 2)
```

- Check Points

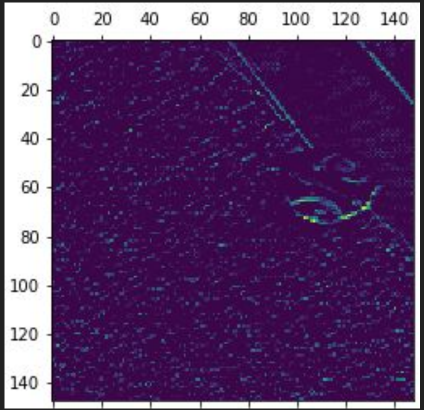
What the CNN “sees”

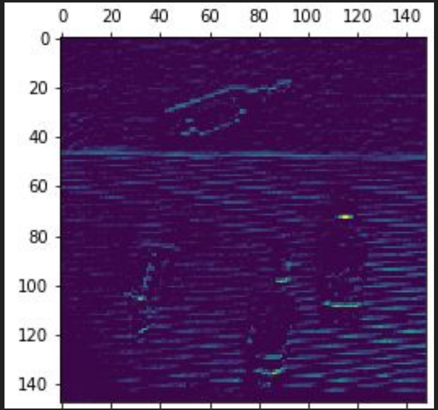


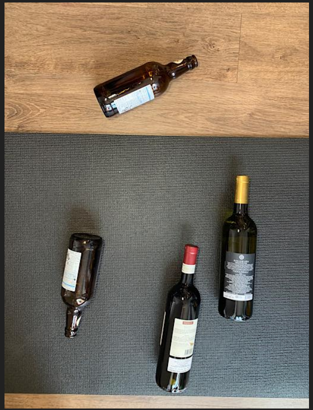




More first layer activations





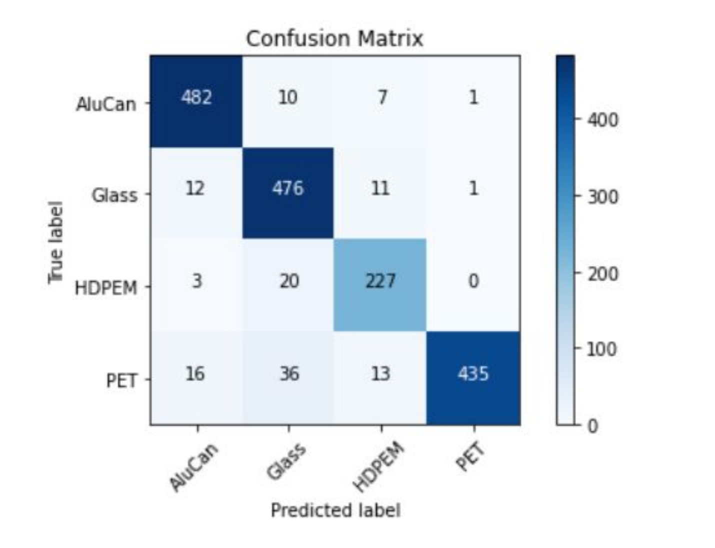


Results

- Epoch 93:
  - loss: 0.0685 - acc: 0.9790 - val_loss: 0.0696 - val_acc: 0.9780

Validation Data

	precision	recall	f1-score	support
AluCan	0.99	0.97	0.98	500
Glass	0.97	0.98	0.98	500
HDPEM	0.93	0.99	0.96	250
PET	1.00	0.97	0.98	500
accuracy			0.98	1750
macro avg	0.97	0.98	0.98	1750
weighted avg	0.98	0.98	0.98	1750

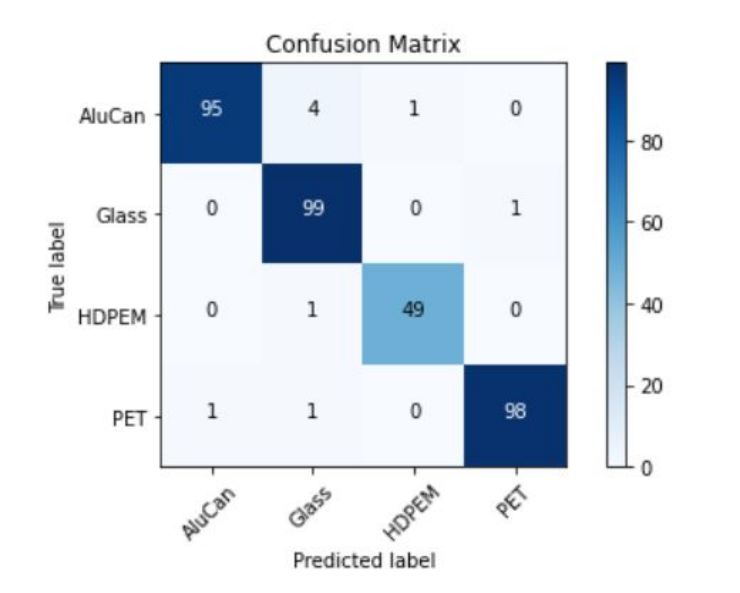


Results

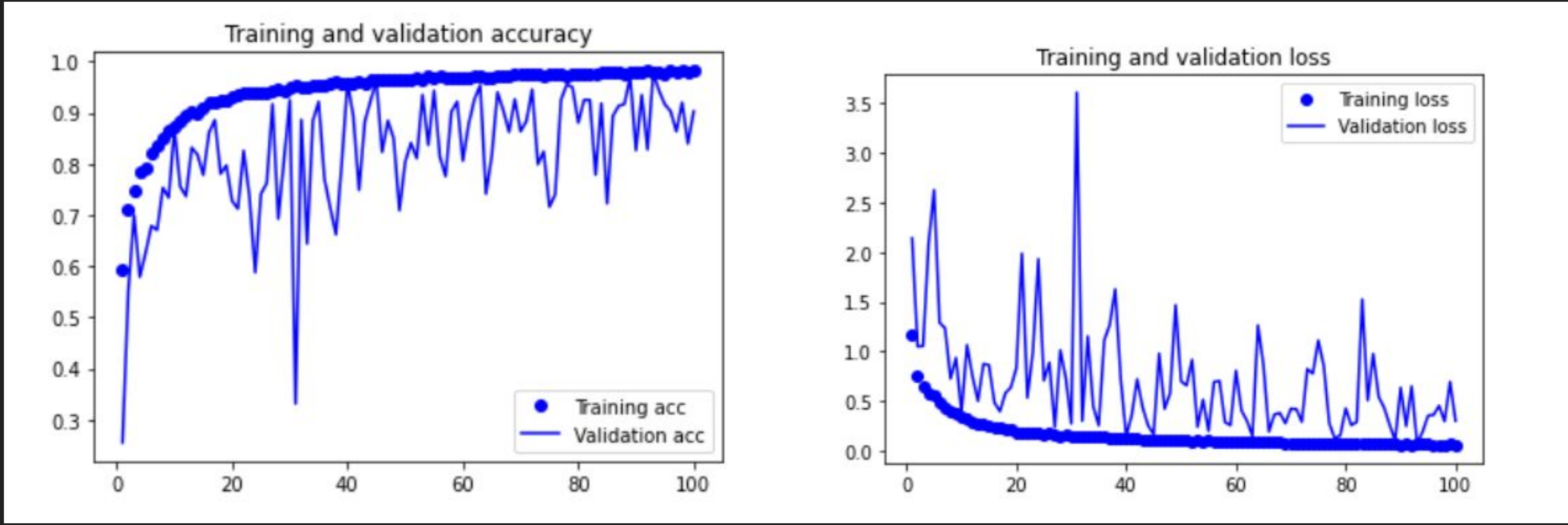
- Accuracy On Test Data:
  - model, accuracy: 97.43%

Test Data

	precision	recall	f1-score	support
AluCan	0.99	0.95	0.97	100
Glass	0.94	0.99	0.97	100
HDPEM	0.98	0.98	0.98	50
PET	0.99	0.98	0.98	100
accuracy			0.97	350
macro avg	0.98	0.97	0.98	350
weighted avg	0.97	0.97	0.97	350



Results



Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100
Training acc
Validation acc
Training and validation loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
Training loss
Validation loss

Future Steps for Improvement

- More data
- More variety in data
- Use Keras Tuner
- Use various amounts of layers and parameters
- More stable backgrounds
- We could have used VGG16 model

References

https://keras.io/

https://www.tensorflow.org/

https://www.kaggle.com/arkadiyhacks/drinking-waste-classification?fbclid=IwAR2uhNnZOSJ5MdyKuLeKJI7UEiTYcV4pUW_9ZMSM_zO8yIHFtWMIRju-DWc

http://web.cecs.pdx.edu/~singh/rcyc-web/index.html

https://deeplizard.com/learn/video/bfQBPNDy5EM

(Buy AMC)