# CSCI 4150 Project - Garbage Classification


### Anushan Vimalathasan, Ken Zhou


In this project, we intend to build a model that is able to classify images of garbage into different classes. These classes are:

- cardboard
- glass
- metal
- paper
- plastic
- trash

The dataset we will be using can be found at https://www.kaggle.com/asdasdasasdas/garbage-classification

Some applications for garbage classification may be to automatically sort garbage so that garbage is placed in the correct bin. For recycling, different types of garbage may need to be recycled differently, so classifying them before recycling would help.

In [1]:
```
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.models as models
import tensorflow.keras.layers as layers
import tensorflow.keras.losses as losses
import tensorflow.keras.optimizers as optimizers
import cv2

import pandas as pd
import numpy as np

import matplotlib.pyplot as pl
import matplotlib.patches as patches
import matplotlib.image as mpimg
```

## Data Collection

To access the dataset in this notebook, we uploaded the dataset to google drive. In order to access it, we need to mount our drive.

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
root_path = "/content/gdrive/My Drive/CSCI4150U/"
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_r
emount=True).
```

The dataset contains a text file of 2 columns, which are the file name, and the class. We start by reading in the training dataset and the testing dataset into pandas DataFrames.

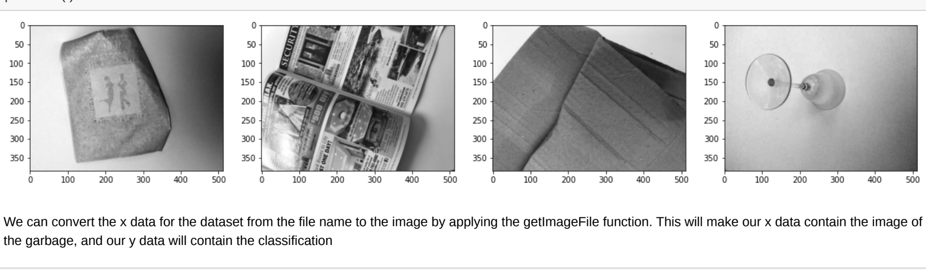
In [3]:
```
train_path = root_path + "archive/one-indexed-files-notrash_train.txt"
test_path = root_path + "archive/one-indexed-files-notrash_test.txt"
val_path = root_path + "archive/one-indexed-files-notrash_val.txt"

# Read data into a DataFrame
train_data = pd.read_csv(train_path, delimiter= " ", header=None)
test_data = pd.read_csv(test_path, delimiter= " ", header=None)
val_data = pd.read_csv(val_path, delimiter= " ", header=None)

# Split data into x and y
x_train_text = train_data[0]
y_train = train_data[1] - 1
x_test_text = test_data[0]
y_test = test_data[1] - 1
x_val_text = val_data[0]
y_val = val_data[1] - 1
```

Since our dataset only contains the file name of the image, we need to create a function that allows us to retrieve the actual image from the file name. The function getImageFile does this by finding the path to the image from the file name, and returning a matplotlib image.

In [4]:
```
# Function to get an image
def getImageFile(fileName):
  image_path = root_path + "/archive/Garbage classification/Garbage classification"

  # Add path to folder to correct image type
  if ("cardboard" in fileName):
    image_path += "/cardboard"
  elif ("glass" in fileName):
    image_path += "/glass"
  elif ("metal" in fileName):
    image_path += "/metal"
  elif ("paper" in fileName):
    image_path += "/paper"
  elif ("plastic" in fileName):
    image_path += "/plastic"
  elif ("trash" in fileName):
    image_path += "/trash"
  else:
    return "Error - File Not Found";

  image_path += "/" + fileName

  image = cv2.imread(image_path, cv2.IMREAD_COLOR)
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  #image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  return np.array(image)
```

We can show some images of the data by plotting the images using matplotlib

In [5]:
```
# Show some images
fig, (ax1, ax2, ax3, ax4) = pl.subplots(1, 4, figsize=(20,5))

ax1.imshow(getImageFile(x_test_text[0]), cmap='gray')
ax2.imshow(getImageFile(x_test_text[1]), cmap='gray')
ax3.imshow(getImageFile(x_test_text[2]), cmap='gray')
ax4.imshow(getImageFile(x_test_text[3]), cmap='gray')

pl.show()
```



We can convert the x data for the dataset from the file name to the image by applying the getImageFile function. This will make our x data contain the image of the garbage, and our y data will contain the classification

In [6]:
```
x_test = x_test_text.apply(getImageFile)
x_train = x_train_text.apply(getImageFile)
x_val = x_val_text.apply(getImageFile)
```

In [7]:
```
# Stack values of the series
x_test = np.stack(x_test.values)
x_train = np.stack(x_train.values)
x_val = np.stack(x_val.values)
```

In [8]:
```
#normalize the values
x_train = x_train / 255.
x_test = x_test / 255.
x_val = x_val / 255.
```

In [9]:
```
x_train = tf.convert_to_tensor(x_train)
x_test = tf.convert_to_tensor(x_test)
```

We create functions that will help us visualize how our model performs.

predictTrash function will display the test images as well as the type of trash the model has predicted along with the actual type of trash in the image.

In [10]:
```
def predictTrash(model):
  trashTypes = ['Glass', 'Paper', 'Cardboard', 'Plastic', 'Metal', 'Trash']
  prediction = model.predict(x_test)

  pl.figure(figsize=(25,40))

  for i in range(50):
    ax = pl.subplot(10, 5, i+1)
    pl.imshow(getImageFile(x_test_text[i]), cmap='gray')
    pl.title("Predicted: " + trashTypes[np.argmax(prediction[i])] + " | Actual: " + trashTypes[y_test[i]])
    pl.axis('off')
```

percentDist function will display the probability distribution of an image and show the trash type of the highest probability as well as the actual trash type of the image.

In [11]:
```
def percentDist(model):
  prediction = model.predict(x_test)
  trashTypes = ['Glass', 'Paper', 'Cardboard', 'Plastic', 'Metal', 'Trash']

  pl.figure(figsize=(30,55))

  for i in range(20):
    ax = pl.subplot(10, 5, i+1)
    pl.bar(trashTypes, prediction[i])
    pl.title("Predicted: " + trashTypes[np.argmax(prediction[i])] + " | Actual: " + trashTypes[y_test[i]])
    pl.xlabel("Trash Types")
    pl.ylabel("Percentage")
```

## Modelling

Now that we have the data ready, we can create a network that can be used to classify the different types of garbage.

The first model will have an input and reshape model for the data, followed by 4 16 filter convolution2D layers using a 3 by 3 kernel and 2 by 2 max pooling layer, followed by a 32 filter conv2d with another 2 by 2 max pooling layer. It will then be flattened using a flatten layer and connected using two dense layers.

In [12]:
```
model1 = models.Sequential([
                            layers.Input(shape=(384, 512)), # The images are 384x512
                            layers.Reshape((384, 512, 1)),

                            layers.Conv2D(16, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 192 x 256

                            layers.Conv2D(16, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 96 x 128

                            layers.Conv2D(16, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 48 x 64

                            layers.Conv2D(16, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 24 x 32

                            layers.Conv2D(32, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 12 x 16

                            layers.Flatten(),
                            layers.Dense(64, activation='relu'),
                            layers.Dense(6, activation='softmax')
], name="Model_1")
```

In [13]:
```
model1.compile(loss=losses.SparseCategoricalCrossentropy(), optimizer=optimizers.Adam(), metrics=['acc'])
```

In [14]:
```
fitting1 = model1.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))
```

```
Epoch 1/10
56/56 [==============================] - 186s 3s/step - loss: 1.6836 - acc: 0.2729 - val_loss: 1.6413 - val_acc: 0.32
93
Epoch 2/10
56/56 [==============================] - 185s 3s/step - loss: 1.4755 - acc: 0.3868 - val_loss: 1.4824 - val_acc: 0.40
55
Epoch 3/10
56/56 [==============================] - 184s 3s/step - loss: 1.3398 - acc: 0.4515 - val_loss: 1.3946 - val_acc: 0.47
26
Epoch 4/10
56/56 [==============================] - 184s 3s/step - loss: 1.2477 - acc: 0.5056 - val_loss: 1.2926 - val_acc: 0.52
13
Epoch 5/10
56/56 [==============================] - 184s 3s/step - loss: 1.0473 - acc: 0.6184 - val_loss: 1.2977 - val_acc: 0.52
13
Epoch 6/10
56/56 [==============================] - 185s 3s/step - loss: 0.8982 - acc: 0.6978 - val_loss: 1.2601 - val_acc: 0.53
96
Epoch 7/10
56/56 [==============================] - 184s 3s/step - loss: 0.6924 - acc: 0.7639 - val_loss: 1.3209 - val_acc: 0.57
62
Epoch 8/10
56/56 [==============================] - 184s 3s/step - loss: 0.5850 - acc: 0.7909 - val_loss: 1.5311 - val_acc: 0.55
18
Epoch 9/10
56/56 [==============================] - 184s 3s/step - loss: 0.3911 - acc: 0.8615 - val_loss: 1.6103 - val_acc: 0.55
18
Epoch 10/10
56/56 [==============================] - 185s 3s/step - loss: 0.2498 - acc: 0.9146 - val_loss: 1.6938 - val_acc: 0.54
57
```

In [15]:
```
model1.evaluate(x_test, y_test)
```

```
14/14 [==============================] - 14s 976ms/step - loss: 1.9443 - acc: 0.5429
```

Out[15]:
```
[1.9443359375, 0.5429234504699707]
```

Following our first model, we reduce the number of filters of the later convolution layers to see what effect it would have on the

In [16]:
```
model2 = models.Sequential([
                            layers.Input(shape=(384, 512)), # The images are 384x512
                            layers.Reshape((384, 512, 1)),

                            layers.Conv2D(16, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 192 x 256

                            layers.Conv2D(16, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 96 x 128

                            layers.Conv2D(8, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 48 x 64

                            layers.Conv2D(8, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 24 x 32

                            layers.Conv2D(4, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 12 x 16

                            layers.Flatten(),
                            layers.Dense(64, activation='relu'),
                            layers.Dense(6, activation='softmax')
], name="Model_2")
model2.compile(loss=losses.SparseCategoricalCrossentropy(), optimizer=optimizers.Adam(), metrics=['acc'])
fitting2 = model2.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))
```

```
Epoch 1/10
56/56 [==============================] - 182s 3s/step - loss: 1.7553 - acc: 0.2156 - val_loss: 1.6236 - val_acc: 0.30
18
Epoch 2/10
56/56 [==============================] - 182s 3s/step - loss: 1.5561 - acc: 0.3500 - val_loss: 1.5289 - val_acc: 0.39
02
Epoch 3/10
56/56 [==============================] - 182s 3s/step - loss: 1.4661 - acc: 0.4120 - val_loss: 1.4301 - val_acc: 0.42
38
Epoch 4/10
56/56 [==============================] - 183s 3s/step - loss: 1.3591 - acc: 0.4524 - val_loss: 1.3739 - val_acc: 0.48
48
Epoch 5/10
56/56 [==============================] - 182s 3s/step - loss: 1.2620 - acc: 0.5274 - val_loss: 1.2873 - val_acc: 0.51
52
Epoch 6/10
56/56 [==============================] - 185s 3s/step - loss: 1.1299 - acc: 0.5767 - val_loss: 1.2830 - val_acc: 0.52
74
Epoch 7/10
56/56 [==============================] - 182s 3s/step - loss: 1.0901 - acc: 0.5893 - val_loss: 1.2909 - val_acc: 0.52
74
Epoch 8/10
56/56 [==============================] - 183s 3s/step - loss: 0.9720 - acc: 0.6370 - val_loss: 1.3656 - val_acc: 0.53
35
Epoch 9/10
56/56 [==============================] - 183s 3s/step - loss: 0.8605 - acc: 0.6928 - val_loss: 1.3473 - val_acc: 0.58
84
Epoch 10/10
56/56 [==============================] - 183s 3s/step - loss: 0.6904 - acc: 0.7373 - val_loss: 1.3224 - val_acc: 0.57
62
```

In [17]:
```
model2.evaluate(x_test, y_test)
```

```
14/14 [==============================] - 14s 964ms/step - loss: 1.4088 - acc: 0.5406
```

Out[17]:
```
[1.408792495727539, 0.5406032204627991]
```

We also want to see how increasing the number of filters the first model will affect the output of our model

In [18]:
```
model3 = models.Sequential([
                            layers.Input(shape=(384, 512)), # The images are 384x512
                            layers.Reshape((384, 512, 1)),

                            layers.Conv2D(32, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 192 x 256

                            layers.Conv2D(32, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 96 x 128

                            layers.Conv2D(32, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 48 x 64

                            layers.Conv2D(32, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 24 x 32

                            layers.Conv2D(64, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 12 x 16

                            layers.Flatten(),
                            layers.Dense(64, activation='relu'),
                            layers.Dense(6, activation='softmax')
], name="Model_3")
model3.compile(loss=losses.SparseCategoricalCrossentropy(), optimizer=optimizers.Adam(), metrics=['acc'])
history3 = model3.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))
```

```
Epoch 1/10
56/56 [==============================] - 358s 6s/step - loss: 1.8014 - acc: 0.2285 - val_loss: 1.5469 - val_acc: 0.36
89
Epoch 2/10
56/56 [==============================] - 359s 6s/step - loss: 1.5506 - acc: 0.3586 - val_loss: 1.5137 - val_acc: 0.34
45
Epoch 3/10
56/56 [==============================] - 355s 6s/step - loss: 1.4427 - acc: 0.3872 - val_loss: 1.4625 - val_acc: 0.42
99
Epoch 4/10
56/56 [==============================] - 356s 6s/step - loss: 1.3653 - acc: 0.4516 - val_loss: 1.4219 - val_acc: 0.42
99
Epoch 5/10
56/56 [==============================] - 356s 6s/step - loss: 1.3126 - acc: 0.4719 - val_loss: 1.3766 - val_acc: 0.49
39
Epoch 6/10
56/56 [==============================] - 357s 6s/step - loss: 1.1240 - acc: 0.5675 - val_loss: 1.3455 - val_acc: 0.46
95
Epoch 7/10
56/56 [==============================] - 361s 6s/step - loss: 0.9855 - acc: 0.6303 - val_loss: 1.3312 - val_acc: 0.50
00
Epoch 8/10
56/56 [==============================] - 355s 6s/step - loss: 0.7796 - acc: 0.7269 - val_loss: 1.5880 - val_acc: 0.46
65
Epoch 9/10
56/56 [==============================] - 357s 6s/step - loss: 0.6510 - acc: 0.7767 - val_loss: 1.4748 - val_acc: 0.49
70
Epoch 10/10
56/56 [==============================] - 356s 6s/step - loss: 0.4606 - acc: 0.8495 - val_loss: 1.7483 - val_acc: 0.51
52
```

In [19]:
```
model3.evaluate(x_test, y_test)
```

```
14/14 [==============================] - 22s 2s/step - loss: 1.7312 - acc: 0.5197
```

Out[19]:
```
[1.731215476989746, 0.5197215676307678]
```

Finally, we tried reducing the number of filters on every convolution layer in our model to see what effect it would have

In [20]:
```
model4 = models.Sequential([
                            layers.Input(shape=(384, 512)), # The images are 384x512 with 3 colours
                            layers.Reshape((384, 512, 1)),

                            layers.Conv2D(8, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 192 x 256

                            layers.Conv2D(8, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 96 x 128

                            layers.Conv2D(8, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 48 x 64

                            layers.Conv2D(8, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 24 x 32

                            layers.Conv2D(8, (3,3), padding='same'),
                            layers.MaxPooling2D((2,2)), # This will make the shape 12 x 16

                            layers.Flatten(),
                            layers.Dense(64, activation='relu'),
                            layers.Dense(6, activation='softmax')
], name="Model_4")
```

In [21]:
```
model4.compile(loss=losses.SparseCategoricalCrossentropy(), optimizer=optimizers.Adam(), metrics=['acc'])
```

In [22]:
```
fitting4 = model4.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))
```

```
Epoch 1/10
56/56 [==============================] - 135s 2s/step - loss: 1.7400 - acc: 0.2463 - val_loss: 1.6516 - val_acc: 0.30
18
Epoch 2/10
56/56 [==============================] - 134s 2s/step - loss: 1.6138 - acc: 0.3303 - val_loss: 1.5225 - val_acc: 0.39
02
Epoch 3/10
56/56 [==============================] - 134s 2s/step - loss: 1.5101 - acc: 0.3791 - val_loss: 1.5956 - val_acc: 0.36
89
Epoch 4/10
56/56 [==============================] - 134s 2s/step - loss: 1.4081 - acc: 0.4504 - val_loss: 1.4239 - val_acc: 0.41
77
Epoch 5/10
56/56 [==============================] - 134s 2s/step - loss: 1.3488 - acc: 0.4416 - val_loss: 1.3753 - val_acc: 0.45
12
Epoch 6/10
56/56 [==============================] - 134s 2s/step - loss: 1.2512 - acc: 0.5234 - val_loss: 1.2984 - val_acc: 0.46
95
Epoch 7/10
56/56 [==============================] - 134s 2s/step - loss: 1.1333 - acc: 0.5728 - val_loss: 1.2651 - val_acc: 0.50
30
Epoch 8/10
56/56 [==============================] - 134s 2s/step - loss: 1.0799 - acc: 0.6059 - val_loss: 1.1975 - val_acc: 0.53
96
Epoch 9/10
56/56 [==============================] - 134s 2s/step - loss: 0.9399 - acc: 0.6697 - val_loss: 1.2629 - val_acc: 0.53
66
Epoch 10/10
56/56 [==============================] - 134s 2s/step - loss: 0.8550 - acc: 0.7001 - val_loss: 1.2875 - val_acc: 0.56
40
```

In [23]:
```
model4.evaluate(x_test, y_test)
```

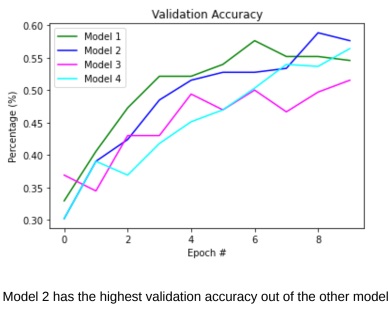
```
14/14 [==============================] - 12s 838ms/step - loss: 1.3010 - acc: 0.5290
```

Out[23]:
```
[1.3010255098342896, 0.52900230884552]
```

## Observations

We want to pick the model that is best suited for our study. The model we chose would be the one with the highest validation accuracy.

In [33]:
```
fitList = [fitting1, fitting2, history3, fitting4]
color = ["green", "blue", "magenta", "cyan"]

for i in range(len(fitList)):
  pl.plot(fitList[i].history['val_acc'], label="Model " + str(i + 1), color=color[i])

pl.xlabel("Epoch #")
pl.ylabel("Percentage (%)")
pl.title("Validation Accuracy")
pl.legend()
pl.show()
```

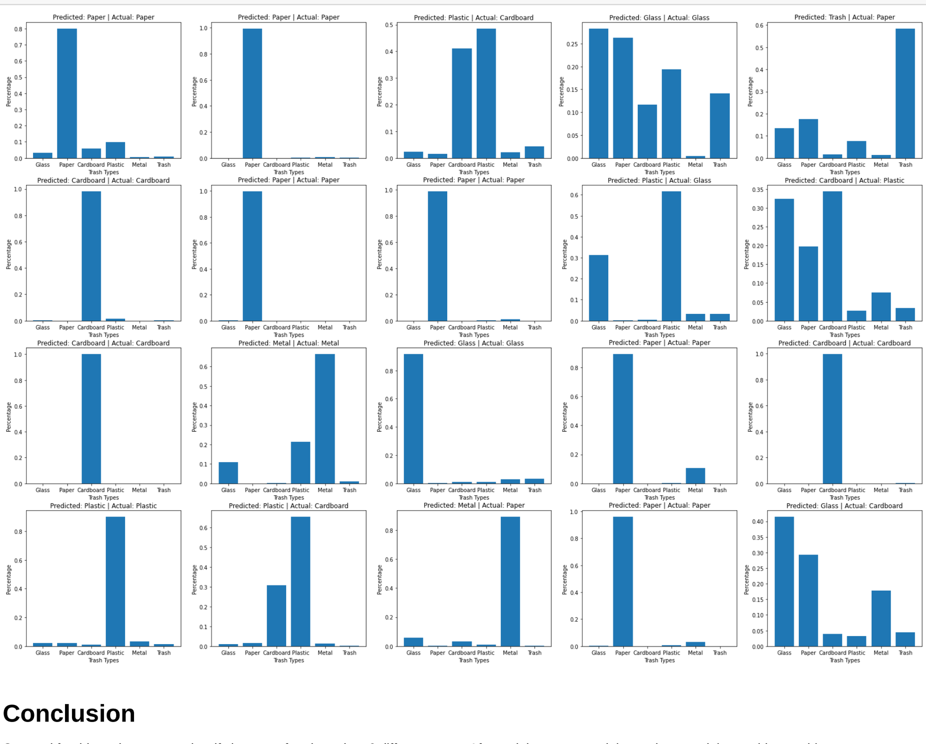



Model 2 has the highest validation accuracy out of the other models.

We want to visualize how our model performs on the test dataset. The predictTrash function will show us an image of garbage with the predicted trash type and the actual trash type. The percentDist function will show us the probability distribution of the types of trash for a test and give us the predicted trash type and the actual trash type.

In [34]:
```
predictTrash(model2)
```


In [35]:
```
percentDist(model2)
```




## Conclusion

Our goal for this project was to classify images of garbage into 6 different types. After training some models, our best model was able to achieve an accuracy of: 54.06% on the test data.