Garbage Classification with Convolutional Networks

1st Farooq A. Khan
Computer Science dept.
Ryerson University
Toronto, CA
farooq1.khan@ryerson.ca

2nd Harun Abdi
Computer Science dept.
Ryerson University
Toronto, CA
harun.abdi@ryerson.ca

3rd Kanishkan Kukarajah
Computer Science dept.
Ryerson University
Toronto, CA
kkukaraj@ryerson.ca

4th Jenil Vekaria
Computer Science dept.
Ryerson University
Toronto, CA
jverkaria@ryerson.ca

***Abstract*—This paper aims to do garbage classification using many computer vision and deep learning techniques. Our goal with this project is to propose a model that can be deployed on IoT devices, has a small footprint, and can be efficiently used to classify garbage. Garbage is classified into six major categories: Cardboard, Glass, Metal, Paper, Plastic, and Trash.**

We iterated through different classification techniques to create a reliable model, starting with a basic Neural Network (NN), then a Convolution Neural Network (CNN), to ResNet, and finally an Efficient Net. These different methods with added modifications were tested and evaluated to find the best model to classify images of garbage. Our most optimized model achieves 90% accuracy on the test set.

I. Introduction

A. Problem Description and Significance

In many cities in Ontario, residents are given multiple bins to separate their garbage, recycling, and compost but all of it ends up on the same truck and goes to the same place. Cities do this because they want to save a quick buck or hand over operations to privatized garbage services whilst ignoring the social and ethical issues of proper waste management.

Often recycled materials are not adequately classified and go into landfills [8]. The typical waste disposal pipeline is that recycled material goes in the blue bin and other types go in the green or black bin; however, humans are not good at making decisions quickly and consistently creating problems. Due to these problems, we must find a safe and reliable way to dispose of waste.

Our experiments attempted empirically to find the best classification network that classifies garbage efficiently and accurately. We created a baseline experimental model with traditional deep learning and gradually improved on to various deep convolution models. The goal of our project is to use image classification to determine types of garbage and separate them into their classifications using the dataset from Kaggle [2]. For this reason, an autonomous waste classification system will significantly improve waste management in many cities. It will introduce consistency at an early stage and further provide a decrease in recycled material going into landfills.

B. Challenge of Garbage Classification

Deep Convolutional Networks have shown highly favourable results for image classification problems; thus, we can leverage the strengths of deep learning by trans-piling our garbage classification problem into an image classification problem.

Unlike many image classification tasks, the challenging portion of this project is that we must consider the reliability and safety of a model and figure out how it would react in certain situations. Figure out where the model has its strengths and weaknesses and find ways to address them.

Another challenge we had to consider was the size and quality of the dataset. It can be difficult to train a CNN if there is not enough data or there are no good patterns that could be extracted from the dataset.

C. Prior Work

Reference [9] tackles the issue of waste classification as well. The authors tried a variety of pre-trained models like AlexNet, DenseNet, ResNet, and custom WasteNet. They achieved 97% accuracy on their custom WasteNet model. We want to see why normal NN will not work and to see if custom CNN models will work and gradually try ResNet and Efficient Net.

D. What is to Follow?

Section II: Explains the garbage classification problem.

Section III: Outlines the models we used, the justification for our evaluation metric, and the use of data processing pipelines.

Section IV: Shares experiments we ran with each model architecture and the results we attained.

Section V: Shares the implementations details and the libraries we used.

Section VI: Concludes the paper and provides suggestions for future work.

II. Problem Statement

We are doing an image classification task to classify six types of garbage: Cardboard, Glass, Metal, Paper, Plastic, and Trash. We got it from Kaggle [2]. Figure 1 shows the distribution of each class of data. With the exception of 'trash', the dataset is uniformly distributed.

The data is of images of garbage with dimensions (384, 512, 3). The labels are one-hot encoding of each class. This is a supervised learning and classification problem.

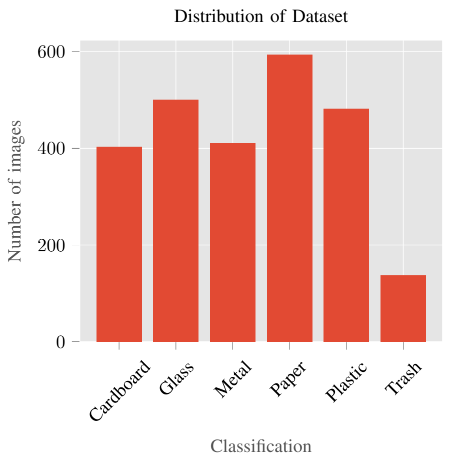


Fig. 1: Distribution of dataset: Image caption: The dataset contains 2532 images split into 6 categories: cardboard (393 images), glass (491 images), metal (400 images), paper (594), plastic (472) images, and general trash (127 images).

III. Method and Models

A. Preprocessing

During the image processing step we applied the traditional Computer Vision image augmentation and image normalization steps. Firstly, the image pixel value is normalized to be between 0 and 1 to improve gradient computation. Secondly, the images were augmented during the training step. The code below shows the types of image augmentation transformations we performed.

```
tf.keras.models.Sequential(
    [ tf.keras.layers.RandomRotation(factor=0.15)
    , tf.keras.layers.RandomTranslation(
    height_factor=0.1, width_factor=0.1)
    , tf.keras.layers.RandomFlip()
    , tf.keras.layers.RandomContrast(factor=0.1)
    ], name='img_augmentation',
)
```

We applied a random set of rotations, translation, flips, and contrast changes. Figure 2 shows an example of cardboard that has been augmented. Since we were dealing with images of trash and not text there was no constraint on the types of transformations we could not do that would pollute the dataset.

The data was split into training and testing sets. The images were then processed using the file name, using one-hot encoding to determine which classification each image belonged to.

B. Deep Neural Network

We tried a simple neural network to create ground truth for our deep neural network approach. This allowed us to reject any more complex models but did not have a better testing score.

C. Convolutional Neural Network (CNN)

CNN is most commonly used for image classification for feature extraction to observe spatial patterns in datasets. It is a hierarchical model that allows you to build a network by stacking Convolutional layers, ReLU activation, max pooling, and passing the output to a fully-connected layer, classifying images into multiple classes.

For this project, we developed our own CNN architecture where the pre-processed images are taken and passed into the first Convolutional layer of size 32, and each kernel is 3x3. Then it applies ReLU activation on the output from the first layer and goes through max pooling, which downsamples the image by half. The same steps repeat for another three layers of sizes 64, 128, and 512. The output from the final max pooling is flattened into a vector, then passed into a dropout layer which randomly sets an input unit to 0, reducing over-fitting. The result from dropout goes through a single dense layer of size 64 and outputs a feature vector. This is then passed into the final dense layer, which applies softmax to classify the image.

D. ResNet

The Residual Network framework was created to ease the training of very deep neural networks. These networks are easy to optimize and can gain accuracy from increased depths [5]. This is a great model to test on the garbage collection dataset

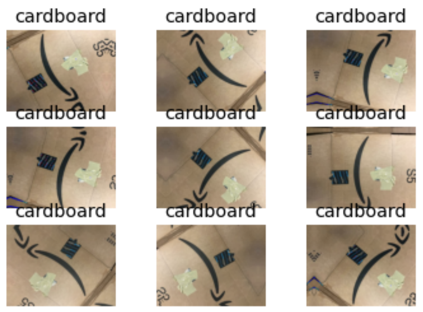


Fig. 2: Image augmentation example

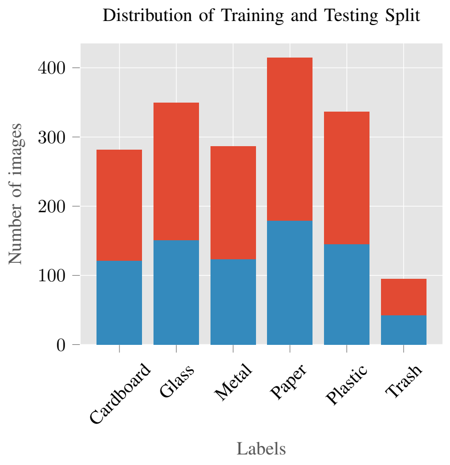


Fig. 3: Distribution and split of the Datset. Blue bars are testing set, and orange bars are training set.

as residual networks work well for deep neural networks. Network depth is very significant; however, continuously stacking layers onto a network can be problematic and result in unwanted properties like high training error [6].

Optimizing very deep neural networks can also be challenging as when the model begins converging, the training accuracy gets saturated and begins to degrade rapidly [5]. Using residual neural networks will allow us to efficiently train and optimize very deep neural networks without adding complexity or degrading training accuracy.

The skip connection allows the alternate shortcut path for the gradient to flow through, solving the degrading gradient problem. The neural network can be trained end-to-end by Stochastic Gradient descent with back propagation, and can be implemented using common libraries [5].

We used the pre-built ResNet50 and ResNet101 models trained on the Imagenet dataset. We added dense layers and a final output layer for model classification, to have a gradual decline of nodes in each dense layer.

E. *Efficient Net*

Unlike many other image classification papers that propose new ideas to improve the task, the Efficient Net architecture was designed to tackle the problem of scaling up a CNN. This was the main reason we chose the Efficient Net model. Often the most straightforward solutions can yield the best results. The EfficientNet B7 model has surpassed many state-of-the-art CNNs by using 8.4x fewer parameters and running 6.1x faster on inference [3]. If we can achieve similar gains with our garbage classification dataset using transfer learning, it will help to provide a fast implementation on IoT devices.

When training a CNN, to get better performance, the number of layers, depth, and resolution of an image are scaled up arbitrarily. The problem of arbitrarily scaling a CNN depends on the use case of the problem set. Different problems will require different CNN architecture, and it is difficult to figure out that exact model.

Intuitively, if the resolution of an image increases, so should the depth and width of the network to capture more ”fine-grained patterns” [3]. This leads to the coordination of balanced scaling rather than single-dimensional scaling.[1]

Much like the ResNet model we used transfer learning to train the Efficient Net B0 up to B4 model. To get a more detailed picture of the B0 Network read [3].

F. *Experiments*

Since we are dealing with Deep Networks, the hyper parameters we need to consider are the learning rate, regularization rate, number of epochs, batch size, number of hidden layers, number of units per layer, and image size.

[1]This intuition has been empirically shown to be true in the paper. Deeper networks are also more challenging to train due to the vanishing gradient problem.

1) CNN: In the CNN experiment, multiple architectures were designed and we tuned hyper parameters such as the number of Convolutional layers, max pooling, and dense layers. In addition, we experimented with different image sizes (100x100, 200x200, 224x224), and epochs (10, 50, 100, 200). To further improve our model, we added image augmentation layer, more convolution layers, dense dropout layer and regularizer.

2) ResNet and EfficientNet: We used Keras Tuner to tune the pre-trained models along with the custom dense layers. Adjusted the parameters mentioned above. Only difference is the gradual decline of the number of units per layer. The formula below as used to calculate the number of layers,

```
units_count = np.linspace(10, 900, hp_dense_count)
    [::-1]
```

where `hp_dense_count` is the number of dense layers. The pretrained models we used were the ResNet50, ResNet101, and Efficient Net B0 to B4.

The hyper-parameters we tuned were the learnining rate, the regularization rate, and the number of dense layers. We used the Hyperband tuning method.

G. *Post Processing and Metrics*

No complicated post processing steps were performed. All outputs to the Networks were softmax activation layers of length 6.

We use validation accuracy to evaluate model performance and categorical loss to obtain the gradients to train the models. An Adam optimizer was used for all models.

The best model has the best testing accuracy. We use accuracy because it is a good metric suited for datasets that are well balanced, aside from the trash class. Other metrics considered were precision, recall, and f-score; however, it would have added extra complexity. We also used confusion matrices the model outputs against its true label.

IV. Results and Discussion

A. *Initial Deep Neural Network (DNN) Model*

The DNN is a a flattened layer that takes as input an image of size (1, 50176). During experimentation, we ran the DNN for 10, 20, 100, and 1000 epochs with a batch size of 16.

The results were not very good. The various amount of epochs the models ran on had little effect on model performance. Validation accuracy was relatively high, and the validation loss was very low, showing us that there was heavy bias and over-fitting in our model. The validation accuracy floated around 23.8% percent, and the training accuracy was around 23.4%. Tweaking the batch image size, additional hidden layers, and more epochs did not improve the model. The model yielded both high bias and high variance. It was time to move on.

B. *CNN*

Using a basic CNN architecture that we developed initially with very few Convolutional layers and max pooling, the results indicated high variance and high bias. The graph seen

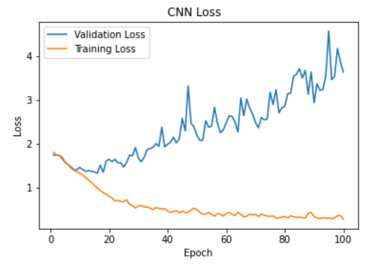
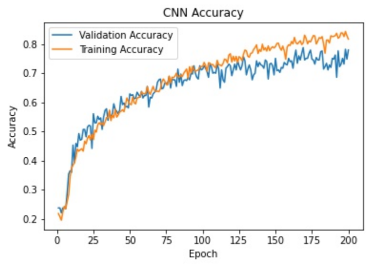
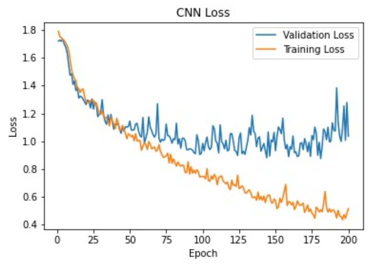


Fig. 4: CNN Training with small network

in Figure 4 is one of the examples of a small CNN model trained with 100 epochs. The model yielded an accuracy of 87% on training and 48% on validation. To combat the high bias we decided to scale up the number of convolutions in the CNN model and to combat the over-fitting we decided to use dropout in our next iteration.



(a) Expanded Network: CNN Accuracy



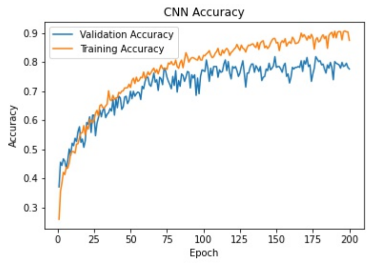
(b) Expanded Network: CNN Loss

Fig. 5: CNN Training with new architecture.

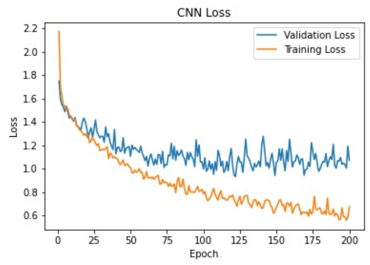
The graph in Figure 5 is the result of our scaled-up CNN. Both the training and validation accuracy are approximately 80% and 76% showing an improvement from our initial model. In terms of the loss graph, there is still high bias and high variance.

We observed slightly more variance on the loss graph from Figure 5, so we added an L2-Regularizer to the dense layers. This technique helps prevent over-fitting as well. This can be seen in Figure 6.

We then tested our modified model (L2-Regularizer) on the test dataset and produced a confusion matrix seen in Figure 7. The model got a test accuracy of 81%.



(a) Expanded Network with Regularizer: CNN Accuracy



(b) Expanded Network with Regularizer: CNN Loss

Fig. 6: CNN Training with add L2-Regularizer

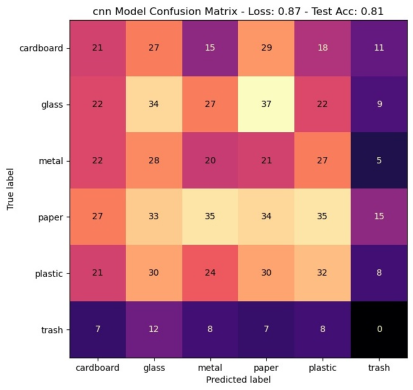


Fig. 7: Confusion matrix: added L2-Regularizer

C. ResNet

Our custom CNN model had high bias and we wanted a target accuracy of above 85%. We had to scale up our model in order to decrease the bias and achieve this target. Training a large model requires a lot time and for that reason we decided to use pre-trained models.

Both ResNet50 and ResNet101 are constructed of 3-layer bottleneck blocks resulting in 50-layer and 101-layer ResNets respectively. After hyper-parameter tuning we received the optimal parameters for training, then trained the model from scratch using these hyper-parameters. ResNet50 performed better than ResNet101 as it resulted in 86% testing accuracy while ResNet101 resulted in 84% testing accuracy.[2] It has given us a high ranking model as shown in Table I.

[2]See Figure 8 for the accuracy

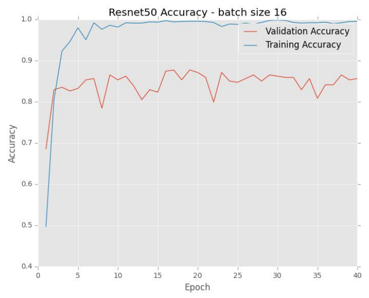
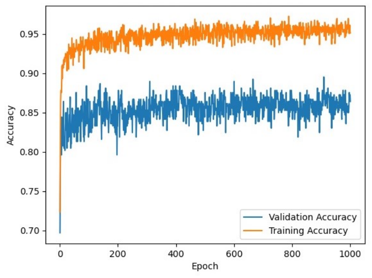
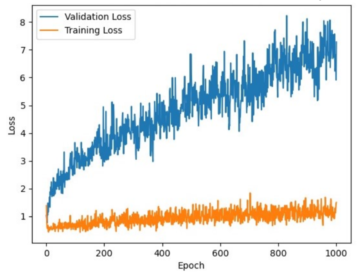


Fig. 8: ResNet best testing accuracy after Hyper Tuning



(a) EfficientNet-B0 Accuracy



(b) EfficientNet-B0 Accuracy

Fig. 9: EfficientNet Trianing with 1k Epochs - Batch Size 16

D. *Efficient Net*

We ran the EfficinetNet-B0 model for 1000 epochs, we achieved a validation accuracy of 89% and testing accuracy of 70%.[3] We learned running for longer epochs does not necessarily mean better performance in the long run, over-fitting can occur very quickly,and as the model learns to overfit the variance in the model will increase. See Figure 9 for the evidence of our conclusions.

The natural fix for this would be regularization; however, intuitively, there seemed to be something else wrong with the network. After consulting with our TA, we developed a modification to the model by ensuring that the output of the EfficientNet and the final softmax layer had a gradual decline. The output shape of the final layer of the pre-trained B0 network is 1280. We introduced a gradual decline of the output shape by adding three dense layers of nodes 800, 600, and 100.

[3]This model by far took the longest to run and showed little performance gains. It took over 5 hours to run 1000 epochs, each epoch taking 18-20s $\pm$ validation evaluation time.

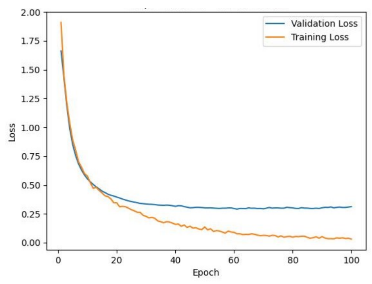


Fig. 10: Improved Efficient NetB0 Loss

Model	**Test Accuracy**
Efficient Net B0 with custom Dense Layers	*0.74*
Efficient Net B1 with custom Dense Layers	*0.76*
Efficient Net B2 with custom Dense Layers	*0.78*
Efficient Net B2 with custom Dense Layers	*0.78*
Our Custom CNN Regularized	*0.81*
Resnet50 Net	*0.86*
Efficient Net B0 with custom Dense Layers - regularized	*0.86*
Efficient Net B3 with custom Dense Layers	*0.8857*
Efficient Net B4 with custom Dense Layers	*0.9014*

TABLE I: Testing Accuracy of all Trained Models

This, in tandem with decreasing the learning rate of the Adam optimizer from 1e-2 to 1e-5 stabilized the model.[4]

This proved to be a valuable addition to our model, seeing as the loss (when trained on 100 epochs) fixed the over-fitting issue.[5]

We experimented with the B1 model to see how much our testing accuracy would increase with a uniform increase in-depth, with, and image resolution. The image resolution was 240 for the B1 model.

Figure 11 outlines the difference in how well the model scales with a decrease in variance and bias on the training and validation accuracy. The number of training epochs was set to 100 for the B0 model and 50 for the B1 model. This shows that a uniform increase has a positive effect on the improvement of the model's classification. The testing accuracy of the B1 model was 76%.[6]

We trained the Efficient Net models B1, B2, B3 and B4. This allowed us to have some of the top-performing models in our collection. See Table I for the results of the B4 model.

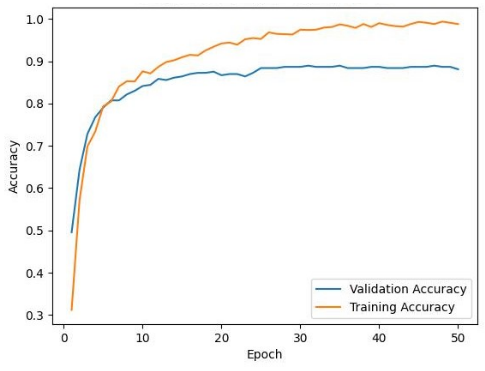
V. Implementation and Code

We used Tensorflow's Keras library for developing the DNNs and using their pre-trained models on the ImageNet dataset. Keras' tutorials were used as a starting point for

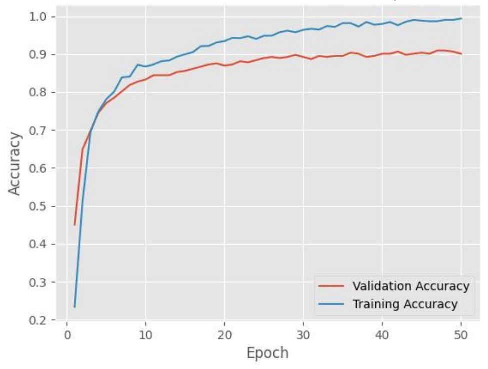
[4]This insight shaped our hyper-parameterization for our dense layers in our RenNet and EfficientNet models.

[5]The validation accuracy stayed around 89% while the testing accuracy bumped to 73%. The loss of the improved model can be seen in Figure 10.

[6]This is an improvement of 2% from B0.



(a) EfficientNet-B0 Accuracy



(b) EfficientNet-B0 Accuracy

Fig. 11: Comparison of the Variance in Accuracy of The B0 and B1 models.

the code. Jupyter Notebooks were used for experimentation. Afterwards, all the code was combined into clean reusable python scripts. This created a pipeline for training multiple models and provided good consistency with the different types of models we were training.[7]

VI. Conclusion and Future Work

One might wonder, why waste time doing image classification with basic NNs? Basic NN allowed us to both understand and develop scalable code for more complex models. Without it we would not have been able to understand the inner working of how an NN learns and what benefits keras provided.

Transfer learning has a fine-tuning step along with the training step. We did not perform the fine tuning step line in [9]. When performing fine-tuning it changes the frozen weights in the pre-trained models.

The types of transformations we performed could also be considered as hyper-parameters to fine tune with. We kept it fixed across all models due to our focus on the more important matters.

All the confusion matrices we produced have unanimously performed poorly when classifying trash. This made us look into the trash pictures, and we realized that the dataset represented the trash classification poorly.

[7]To see implementation details see the GitHub https://github.com/Farooq-azam-khan/cps803-course-project

For example, an empty plastic bag would be classified as plastic; however, a plastic bag with some food would be trash. That makes it very hard for the model to distinguish between the two. The trash images are dragging down the uniformity of our dataset as shown in Figure 1. All classifications have more than 400 images; however, only the trash classification has below 200 image. If we were to repeat these experiments, we would remove the trash classification.

References

[1] Khan F., Abdi H., Vekaria J., Kukarajah K. https://github.com/Farooq-azam-khan/cps803-course-project

[2] CCHANG, “Garbage Classification.” Kaggle, 2018, doi: 10.34740/KAGGLE/DS/81794.

[3] Mingxing Tan and Quoc V. Le, ”EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” Cornell University, 2019.

[4] fchollet, ”Image classification from scratch ,” Updated: 2020/04/28. https://keras.io/examples/vision/image_classification_from_scratch/

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, ”Deep Residual Learning for Image Recognition,” Microsoft Research, 2015.

[6] Mujtaba, H., 2020. What is Resnet or Residual Network — How Resnet Helps?. [online] GreatLearning Blog: Free Resources what Matters to shape your Career!. Available at: https://www.mygreatlearning.com/blog/resnet/ [Accessed 3 December 2021].

[7] Maladkar, K., 2021. Why ResNets Are A Major Breakthrough In Image Processing. [online] Analytics India Magazine. Available at: https://analyticsindiamag.com/why-resnets-are-a-major-breakthrough-in-image-processing/ [Accessed 3 December 2021].

[8] T. Simmons, “Garbage, recycling in the same truck? why it happens, and why some say it’s concerning — CBC news,” CBCnews, 17-Jun-2019. [Online]. Available: https://www.cbc.ca/news/canada/toronto/private-waste-services-garbage-recycling-1.5175874. [Accessed: 03-Dec-2021].

[9] Gary White, Christian Cabrera, Andrei Palade, Fan Li, Siobhan Clarke. WasteNet: Waste Classification at the Edge for Smart Bins.

Acknowledgment

We would like to acknowledge our TA, Reggie McLean and our professor Dr. Nariman Farsad who played a significant role in helping us complete this project. Reggie guided us throughout the semester, giving us feedback on how to move forward with our research project. He helped us immensely with technical and moral support, guiding us with new ideas and a positive attitude.