Classification of Recyclable Items (Group 17)

HoHuan Chang and **Shawn Chua** and **Jiayao Li** and **Victor Loh** and **Peck Kai Ting**
A0133370R, A0129612J, A0160257J, A0135808B, A0130651U

Abstract

In recent years, the idea of recycling and conservation has gained significant prominence globally, including within the technology sector. The key objective of this project is to utilize machine learning to perform multi-class classification on different types of garbage, in the hope that the final product can be integrated into larger systems to improve upon and ameliorate certain issues faced by existing recycling efforts. The strategies that will be discussed in this report are Convolutional Neural network, Random Forest and XGBoost.

1 Introduction and Project Description

Based on national statistics (NEA 2018), 7.70 million tonnes of waste was generated in 2017. Although this number has decreased from the previous year, the recycling rate remains largely unchanged at 61%. In a Channel Newsasia article, Lim highlighted several issues hampering the increase in recycling rates, amongst them the confusion faced by individuals in determining what can be recycled and what cannot.

Due to this lack of awareness, people often discard their garbage incorrectly, throwing them into recycling bins when they belong in the trash bin. This means that not all items collected from recycling bins are recyclable. As a result, additional manpower is needed to manually sort the recyclables from the non-recyclable items.

In our project, we aim to develop a multi-class classifier that is able to distinguish recyclable items from those that are not, and identify the recyclable material. Such a classifier can then be incorporated into automated garbage sorters for more efficient sorting processes. Although our project tends to target organizations which are able to do so on a larger scale, we also allude to possible improvements using more scalable variants, that may promote recycling awareness at the individual level in Section 7.

We will assess mainly Convolutional Neural Networks (CNNs) for image classification. However, we also apply ensemble methods such as random forest and Extreme Gradient Boosting (XGBoost). We then compare the different machine learning methods, and propose the most suitable one to accomplish the above-mentioned tasks.

2 Previous Work

Previous work done in the area of classification of recyclable materials include the work by Torres-García et al., which uses the k-Nearest Neighbours (kNN) algorithm and Özkan et al., which uses feature extraction techniques such as Principle Component Analysis (PCA), as well as Support Vector Machines (SVM). Their work has usually involved significant effort in image processing. Yet, there is another method for image classification, which is the multi-layer Convolutional Neural Network (CNN). Such methods have gained traction over the past years, proving to be efficient for image classification (Krizhevsky, Sutskever, and Hinton 2012).

Due to the high dimensionality of image vectors, training of image data is usually computationally expensive and requires an extensive amount of time. Hence, people often utilize existing pre-trained models for classification problems. In Section 4.1, we apply transfer learning on *InceptionV3*, a pre-trained model, and use this as a benchmark for our self-built models.

3 Dataset Preparation

For the purpose of this project, we make use of a (surrogate) dataset, titled *Trashnet*, from *GitHub*, collected by Thung. The dataset consists of 2527 512 $\times$ 384 pixel images from six categories: cardboard, glass, plastic, metal, paper and trash. There are approximately 450 images for each category, except trash, with only 150 images. Most images consist of a single object. Not all the images are centered, but this placement allows for some versatility in the training result. The background is single-colored for most samples.

Due to the relatively small dataset size, we performed data augmentation techniques on the original images. Additional training samples were generated by changing brightness controls, performing translations and scaling on the original images. These transformations allowed us to better maximize the usage of the original dataset, hence providing a better approximation to real world situations.

One thing to note about this dataset is that most of them were prepared in a clean environment, and each image only contains a single object. This allows the data to better exploit our choice of classification algorithms, and reduces the effort required in terms of image processing. However, such conditions are seldom observed in real trash. It may be more practical to generalize classification algorithms for images with batches of objects. Hence in future, we wish to work with images obtained under more generalized conditions to

better simulate real-life situations of trash in recycling facilities, as explained in Section 7.

4 Design and Strategy

For this project, we analyzed two approaches of tackling the image classification problem: Convolutional Neural Networks (CNNs) and ensemble methods (Random Forest and XGBoost).

For image vectors, machine learning methods such as SVMs suffer from their high dimensionality, due to high time and space complexity. In fact, when we first trained the images using an SVM using the original image size, we encountered out-of-memory situations; even with scaled-down images, SVMs still took much longer to train than CNNs, yet only yielded results slightly better than random guessing ($\approx 50\%$). Hence, the reason for our choice of CNNs is largely due to its ability to make use of images to make model training more efficient by constraining the architecture in a more sensible manner. As the chosen ensemble methods have relatively fast run-time, and yields fair results for classification problems in general, they are used for comparison against the CNN.

For the CNN approach, we first trained the data using a CNN architecture that we have built from scratch. ("Built from scratch" in the context of this report refers to the the construction of CNN by composition of different layers using existing libraries. We did not write the library). Optimization of the model hyper-parameters using Bayesian optimization was also carried out. Lastly, we applied transfer learning on a pre-trained model, *InceptionV3*, to provide a benchmark for the models. In Section 5, we evaluate both approaches.

For CNNs, the images are usually taken as inputs directly into the neural network, whereas for ensemble methods, images are first transformed into vectors of pixel values before being fed into the algorithm. As the dataset contains relatively large-sized images, this translates to high input feature dimensions for the image vectors. In order to prevent instances of insufficient memory and to limit the overall runtime, in some cases we used scaled-down images, generated using the *Image* library from *Python Imaging Library* (PIL).

4.1 Convolutional Neural Network (CNN)

As mentioned above, CNNs assumes explicitly that inputs are images, which allow them to utilize structural information for image classification. Instead of having fully-connected weights to every single pixel (which would result in a vast number of weights to compute at **every** layer), it makes use of different types of layers to extract information from images (Karpathy 2015). For instance, features of images such as their general shapes, as well as common lines or edges shared by certain objects can be easily extracted using convolutions. Such properties makes CNNs especially effective for image classification tasks, and allows it to learn more efficiently than, say, artificial neural network (ANN), whilst using far less parameters to do so.

In fact, CNNs has been successfully applied to many other computer vision problems, such as facial recognition and scene labeling (Coşkun et al. ; Yan et al. 2017). Their applications include gesture controls in mobile applications and image segmentation in autonomous driving (Cadence 2017).

CNN from scratch In this section, we will first explore the commonly used layers in CNNs. These layers were also used in our CNN architecture. The layers are as follows.

- **Convolution layers** make use of filters to extract features from images. Convolutions preserves the spatial relationship of each pixel by extracting a small localized section of the image using filters of size $F \times F$. Filters are moved across the entire image, in strides of S pixels. Here, the tunable parameters are F, S, and the number of filters N (this accounts for the depth of the output from this layer).
- **Pooling layers** down-sample image data extracted by the convolution layers to reduce dimensionality of the feature map, reducing processing time for future layers. A common form of pooling used is max pooling, which extracts the maximum value from subregions of the feature map and discards all other values. For this layer, there are 2 tunable hyper-parameters: the size of the pooling filter $F \times F$, as well as the stride S.
- **Flattening layers** simply flattens the input. If the input array is $(W \times H \times D)$, then the output of a Flatten layer will be of shape $(1 \times O)$, where $O = W \times H \times D$.
- **Dense layers** refer to fully connected network layers, which are similar to the layers in Artificial Neural Network(ANN). It is fully connected to all activations in the previous layer, and feeds all data from the previous layer to all its neurons.
- **Dropout** is a form of regularization in neural networks, which helps to reduce over-fitting, through reducing the interdependencies among neurons. By applying a rate of p, neurons are dropped with the probability of $1 - p$ and one would only train the data using this reduced network in that stage. Doing so would also improve training speed.

In our architecture, two different activation functions were used - **Rectified Linear units (ReLu)** and **Softmax**.

ReLu introduces non-linearity to the model. The function for ReLu is as follows:

$$R(x) = \max(0, x) \tag{1}$$

The ReLu function is simple and efficient (Occam's razor), compared to other activation functions such as Tanh, where optimizations for gradient updates can be computationally expensive. ReLu also solves the vanishing gradient problem.

Softmax function transforms a vector $\mathbf{x} \in \mathbb{R}^k$ into another K-dimensional vector of real values in the range (0, 1). The vector components sums up to 1. It reacts differently (in terms of probability) to different levels of stimulation, hence its usefulness.

The CNN architecture was built using Keras, with the *RMSProp* gradient descent algorithm as an optimizer. The CNN has the following structure:

1. Input layer of shape $[height \times width \times depth] = [150, 150, 3]$

2. Convolution layer with filter size 6×6, stride of 1, and N filters, with *ReLu* activation, followed by a Max Pooling layer of size 2×2, stride of 2
3. Convolution with filter size of 3×3, stride of 1, and M filters, with *ReLu* activation, followed by a Max Pooling of size 2×2, stride of 2
4. K number of dense layers with 256 output nodes, ReLu activation, with a Dropout of rate 0.5
5. Dense Layer with 6 output nodes, with *softmax* activation. This is also the output layer.

Here, N = number of filters for the first convolution layer, M = number of filters for the second convolution layer and K = number of dense layers.

We used the *SciKit-Optimize* library to perform Bayesian optimization on the CNN to tune the values for N, M, K, *batch size* $\in \mathbb{Z}$ and the learning rate of the optimizer η. The range constraints used for each parameter were as follows: $N \in [32, 64]$, $M \in [64, 96]$, $K \in [1, 5]$, *batch size* $\in [32, 64]$ and $\eta \in [0.0001, 0.1]$.

Transfer Learning (Pre-trained Model)

As a benchmark to our CNN model built from scratch, we also applied transfer learning on a pre-trained model *InceptionV3*. InceptionV3 is a pre-trained model, trained using images from *ImageNet*, a vast image database consisting of over 12 million images, to classify images from 1000 categories. We replaced the topmost layer with a dense layer with 6 output nodes (for the six classes in our classification problem) and the *softmax* activation function. Here, we are concerned with training a feature extraction layer using weights from the earlier pre-trained layers, in order to do multi-class classification for our 6 targeted categories.

Given the large amount of data used to train *InceptionV3*, it is likely that the earlier pre-trained layers may contain information or features useful for our problem. We do not perform Bayesian Optimization here, instead leaving the learning rate and the batch size constant at 0.001 and 32 respectively, as we expect such a pre-trained model to produce similar or better results than the CNN built in Section 4.1.

4.2 Random Forest and XGBoost

Our team also experimented with ensemble methods: random forest and XGBoost. Such methods typically combine many base learners to form stronger classifiers.

Here, we used scaled-down images of sizes 64 x 48 and 128 x 96. Additional data preprocessing steps involve a transformation of the images into vectors of pixel values, followed by a normalization process where applicable.

Random Forest

Definition 1.1 A *random forest* is a classifier consisting of a collection of independent and identically-distributed (i.i.d) decision trees, whereby for a given input $\mathbf{x}$, each tree casts a unit vote for the most popular class (final output for $\mathbf{x}$).

In Breiman's paper, a theoretical proof was given for this rather promising result: as the number of trees T increases, random forests do not overfit; rather, the generalization error converges to a limiting value. In addition, random forests are robust to noise, and are usually faster to train compared to neural networks. This provides us with a potentially useful comparison against the CNN models.

In training the data, we used the *SciKit-Learn* package from Python. For training each tree, training samples are drawn with replacement (bootstrap samples) from the entire set. Typically, in each bootstrap, about two-thirds of samples are retained as training samples. The remaining are used to compute out-of-bag estimates. In the splitting steps of the tree, we consider a maximum of $f = \sqrt{M}$ features, randomly permuted in each split, where M = total number of features. The quality of splits are measured using the Gini impurity index at each node, given by Equation 2.

$$I_G(p) = 1 - \sum_{i=1}^{K} {p_i}^2 \tag{2}$$

where p_i = probability of the classification i, based on the training set. Bayesian optimization and grid search were used to tune the hyper-parameters here. The parameters (integer-valued) and their ranges are as follows: number of trees $T \in [1, 10^5]$ and minimum number of observations at each leaf $k \in [1, 100]$. It was noted from Breiman's paper that changing f usually does little for improving the generalization accuracy, hence we do not optimize for this parameter here.

XGBoost

XGBoost is a variation of Gradient Tree Boosting (GTB), a popular machine learning technique used for supervised learning problems. Its scalability and speed contribute significantly to its success and is widely used even in competitions (Chen and Guestrin 2016). Similar to *random forest*, it is an ensemble method, but uses instead an ensemble of regression trees (CART) $f_k \in \mathcal{F}$. Each tree has a number of leaf nodes with leaf weights w. Mathematically, it represents a function mapping an input $\mathbf{x}$ to a leaf node, and outputs the weight (score) attached to that leaf. Given $\mathbf{x}$, the output of the ensemble is the sum of the scores from all the trees. More precisely,

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), f_k \in \mathcal{F} \tag{3}$$

The goal is to find the set of trees, by minimization of a standard regularized objective:

$$L^{(t)} = \sum_i l(y_i, \hat{y_i}) + \sum_k \Omega(f_k) \tag{4}$$

where l is a differentiable loss function and Ω is the regularization term which prevents overfitting of the model.

The model is trained iteratively: Fix what we have learned up to this point, and add the tree that improves the objective the most. A greedy algorithm is used for finding splits for the trees, either by enumerating over all possible splits on all features, or finding proposed splits based on percentiles of feature distribution. Gradient statistics are aggregated for candidate splits to evaluate the best splits.

One important hyper-parameter to tune is the maximum depth of the tree d as well as the learning rate η. Jain stated that typical values are $d \in [3, 10]$ and $\eta \in [0.01, 2]$. Trying out several values, we settled on $d = 7$ and $\eta = 0.05$, as this parameter setting achieved better results compared to others.

4.3 Bayesian Optimization

For the methods described in Section 4.1 and 4.2, each model has a different set of tunable hyper-parameters, and finding an optimal set of hyper-parameters is a challenge in itself. Common ways of hyper-parameter optimization include grid search (a type of exhaustive searching), random search and Bayesian optimization.

For our experiment, we performed mainly Bayesian optimization on our models, as it is often hard to determine a suitable subset of hyper-parameters to iterate over if grid search was used instead.

Typically in Bayesian optimization, a Gaussian Process Prior is assumed for the function f(**x**) to be maximized (in our case, the validation accuracy), and a posterior distribution is maintained for the observations during the procedure, so that information from previous evaluations are also used for determining the next best point **x** (parameter setting) to evaluate at, by maximizing a function known as the *acquisition function*. More precisely, the posterior distribution is

Algorithm 1: Bayesian optimization procedure

for *t = 1,2,...* **do**
  Find $\mathbf{x}_t$ by optimizing the acquisition function;
  Compute the value of the objective function value (with Gaussian noise), y_t;
  Augment the data (set of observations) with $(\mathbf{x}_t, y_t)$ and update the posterior;

characterized by mean $\mu(\mathbf{x})$ variance $\sigma(\mathbf{x})^2$. Commonly-used acquisition functions are the Expected Improvement (EI) and Upper Confidence Bound (UCB). In Bayesian optimization, one often has to find a balance between exploitation (choosing **x** which gives a high value of $\mu(\mathbf{x})$) versus exploration (choosing **x** with a high variance $\sigma(\mathbf{x})$). We illustrate this idea using the UCB as shown in Equation 5.

$$UCB(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}) \tag{5}$$

where κ is a tunable parameter to balance exploitation and exploration.

Algorithm 1 summarizes the optimization procedure. As mentioned in Sections 4.1 and 4.2, we attempted to optimize for the *number of trees* in the random forest, and the *filter sizes*, *number of dense layers*, *batch size* and *learning rate* (without decay) for the CNN.

5 Results and Findings

5.1 Convolutional Neural Network

For the CNN, we used a consistent number of epochs and dataset split across all of our trained models. We ran the models on 150 epochs and with a dataset split training/validation/test split of 0.7/0.15/0.15. For our inputs to the CNN built from scratch, our input shape for the images are scaled to $150 \times 150 \times 3$ to reduce memory load. For the InceptionV3 model with transfer learning, we use resized images of size $299 \times 299 \times 3$. This is consistent with the input size for *InceptionV3*.

For the CNN built from scratch, we initially used the default hyper-parameter settings of $N = 32, M = 64, K = 2$, *batch size* b $= 32$ and *learning rate* $\eta = 0.01$. This yielded a final test accuracy of 74.1%. Figure 1 shows the confusion matrix for this run. After 30 iterations of Bayesian optimization under the Gaussian Process Prior, we arrive at the following set of optimized hyper-parameters: $N = 55$, $M = 70$, $K = 1$, $b = 32$, $\eta = 0.0001$. Figure 2 shows the confusion matrix for this run. This gave a final test accuracy of 77.8%, about 3% improvement from the model prior to optimization. Lastly, how did our self-built and trained model fare against a transfer learnt model? Surprisingly, the results obtained previously are relatively good compared to the pre-trained model. Figure 3 shows the confusion matrix for the model with transfer learning on *InceptionV3*. This model yielded a final test accuracy of 79.8%, which is considerably close to the self-built CNN model.

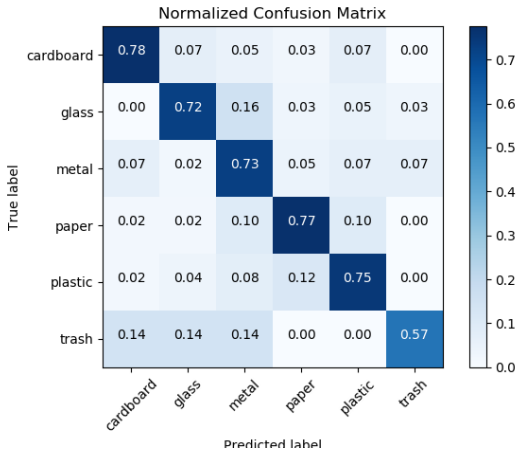

Figure 1: Confusion Matrix for the self-built CNN

5.2 Random Forest and XGBoost

Besides CNNs, we also ran the ensemble methods described in Section 4.2. We first ran the random forest algorithm on the image vectors with number of trees $T \in [100, 1000]$. 5-fold cross validation was used, and we consider the average validation error from the 5 folds. The results are shown in Figure 4.

With this as a reference, we optimized for the parameters using Bayesian optimization methods, with a larger range of T for the algorithm to search through. The acquisition function used was Expected Improvement (EI). Although this method yielded a lower generalization accuracy than the CNNs, it is still noteworthy that this relatively simpler model managed to achieve an average out-of-bag estimate of 65% (out of 5 folds) and improved generalization accuracy of 68.8% with optimized parameters: 4629 trees, and

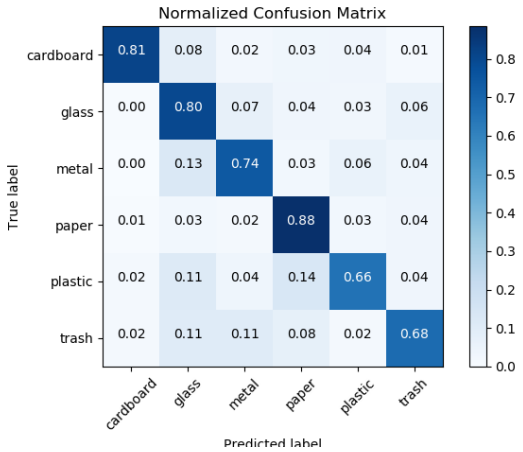

Figure 2: Confusion matrix for the self-built CNN after Bayesian Optimisation

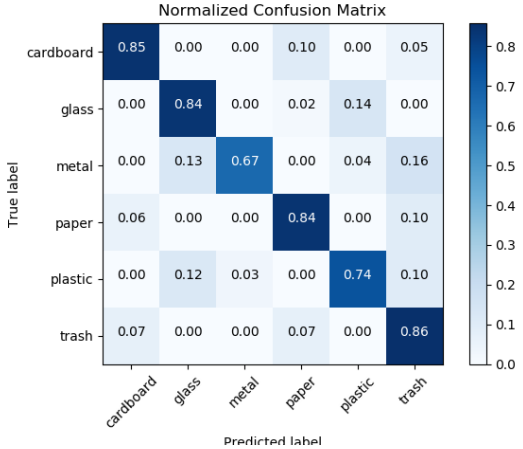

Figure 3: Confusion matrix for *InceptionV3* with transfer learning

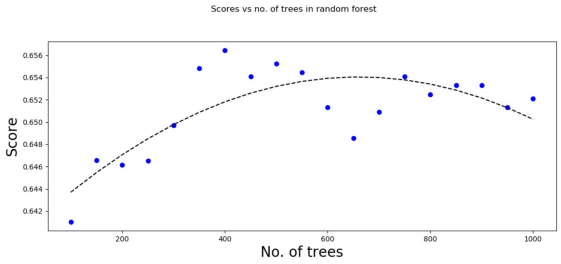

Figure 4: No. of trees against validation accuracy score

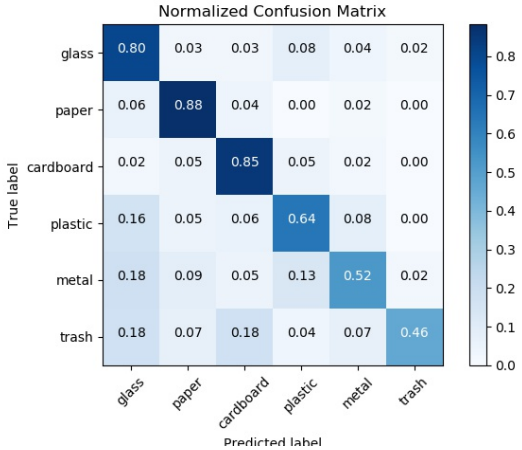

Figure 5: Confusion matrix of random forest with 4629 trees

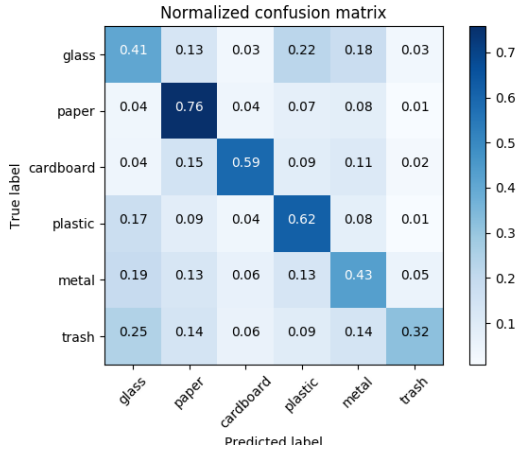

Figure 6: Confusion matrix of XGBoost model

a minimum of 1 observation per leaf. The confusion matrix for one of the folds is as shown in Figure 5.

For XGBoost, the results are shown in Figure 6.

Based on the results in Sections 6 and 5, the ensemble methods did not score as well in validation accuracy, compared to the CNNs. A possible explanation for this is that XGBoost is more suited for training tabular data, rather than images. Random forests also tend to train less well on highly detailed data, and may not capture certain properties of images in general. These ensemble algorithms do not model the spatial temporal locality in images (Chen 2016), unlike CNNs. According to Rohrer, different aspects and features of an image are captured better using CNNs. Lower layers are adept at capturing the edges and bright spots in an image while high layers represent more complicated aspects like the shapes and patterns.

6 Conclusion

In our work, we were able to obtain empirical results which demonstrate the advantage of using CNNs compared to other

methods such as ensemble and boosting methods. Since there are certain properties of images (compared to tabular data) that can be encoded into CNNs but not other methods, the result did not come across as surprising.

In Section 1, we described the aim of our work - to develop a classifier that is capable of classifying objects into specific types of recyclables and trash - in the context of our proposed real-life application. Taking our analysis of different machine learning techniques for multi-class classification, as well as our insights based on empirical analysis, we propose the CNN method for such a classifier. However, in our experiment, there were also certain constraints, such as the size of the dataset and computational power. In Section 7, we describe possible improvements to our work, so as to better generalize to and emulate real-life situations.

7 Future Work

Based on what has been done, there are a couple of potential improvements. Firstly, an observation we have made is that different machine learning models perform differently on identifying particular classes. Hence, combining or averaging results of different classification methods can possibly pave a breakthrough, perhaps improving the generalization accuracy of the models. Secondly, transfer learning performs the best by far, which is not surprising given that it was trained on a much larger set of data. Thus, it may be feasible to look into different pretrained models and find the one which performs best on garbage classification.

Thirdly, it is important to achieve accurate classification on garbage in real time on real world objects instead of just images. A potential application of this would be the building of large automation systems in recycling facilities, to reduce manpower cost and increase production efficiency. Lastly, in Section 3, it was noted that images may not always be generated in a clean environment to produce data that can better exploit machine learning algorithms. It is also impractical to capture every item individually in a large-scale sorter. Hence, a more viable option may be to generalize such classifiers to classify batches of objects.

Another area we are looking into is the extension of our application to devices with less computational power, such as mobile devices. There are existing pre-trained models designed specifically to meet such constraints, such as *MobileNets*. By doing transfer learning on these models, we can potentially achieve a small, low latency and power consuming model which can make acceptably accurate classifications. With this, our application may be helpful also for consumers recycling trash, on top of its utility in automated sorters. This may potentially ameliorate the confusion faced by individuals, as highlighted in our introduction.

8 The Team

Throughout the project, the effort and contribution of every team member has been invaluable. Jiayao, Victor and HoHuan did extensive research and experiments on the CNN. Jiayao and HoHuan worked on the self-built CNN, while Victor experimented with transfer learning on pre-trained models and data augmentation. Shawn and Kai Ting worked on ensemble-based algorithms, and conducted related experiments. As our team comprised of members from different disciplines, we had varied strengths and weaknesses, which helped to generate a large pool of ideas. HoHuan provided opinions from the Engineering perspective; Victor and Jiayao, being more adept at programming, steered the programming areas. Shawn and Kai Ting used their knowledge from Mathematics / Statistics to understand the more theoretical aspects. During experimentation and analysis, we learnt about the sensitivities and suitability of different methods. With the team's combined effort, we complemented one another's weaknesses to work towards the project's completion.

References

[Breiman] Breiman, L. 2001. Random forests. *Machine learning* 45(1):5–32.

[Cadence] Cadence. 2017. Introduction to convolutional neural networks (cnn).

[Chen and Guestrin] Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. *CoRR* abs/1603.02754.

[Chen] Chen, T. 2016. Tianqi chen's answer to "why is xgboost given so much less attention than deep learning despite its ubiquity in winning kaggle solutions?".

[Coşkun et al.] Coşkun, M.; Uçar, A.; Yıldırım, Ö.; and Demir, Y. Face recognition based on convolutional neural network.

[Jain] Jain, A. 2016. Complete guide to parameter tuning in xgboost (with codes in python).

[Karpathy] Karpathy, A. 2015. Cs231n convolutional neural networks for visual recognition.

[Krizhevsky, Sutskever, and Hinton] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[Lim] Lim, L. 2017. Why is singapore's household recycling rate stagnant?

[NEA] NEA. 2018. Waste statistics and overall recycling.

[Özkan et al.] Özkan, K.; Ergin, S.; Işık, Ş.; and Işıklı, İ. 2015. A new classification scheme of plastic wastes based upon recycling labels. *Waste management* 35:29–35.

[Rohrer] Rohrer, B. 2016. How do convolutional neural networks work?

[Thung] Thung, G. 2017. Github, trashnet.

[Torres-García et al.] Torres-García, A.; Rodea-Aragón, O.; Longoria-Gandara, O.; Sánchez-García, F.; and González-Jiménez, L. E. 2015. Intelligent waste separator. *Computación y Sistemas* 19(3):487–500.

[Yan et al.] Yan, K.; Huang, S.; Song, Y.; Liu, W.; and Fan, N. 2017. Face recognition based on convolution neural network. In *Control Conference (CCC), 2017 36th Chinese*, 4077–4081. IEEE.