***Capstone Project***

**Deep Learning-Based Smart Trash Can to Improve The**

**Recycling Behavior at Queens University of Charlotte**

**Student:** Alessandro Araujo

**Mentor:** Marco Scipioni, Assistant Professor of Physics at Queens University of Charlotte

**Abstract**

*Recycling is critical for saving our planet. The practice of recycling helps reduce pollution, greenhouse gas emissions, and the amount of waste that is disposed in landfills. The majority of Americans unfortunately does not embrace the three Rs (reduce, reuse, recycle). The lack of adequate knowledge for sorting and recycling materials is one of the biggest barriers to being green. Recycling is a behavior that can be improved through technology. This capstone project is centered around the creation of a smart trash can prototype designed to create awareness among the Queens University community on the importance of correctly sorting waste items. The smart-trash can has both a hardware and a software component. The author specifically focused on the software component of the prototype and developed a working deep learning (DL) model using the Python deep learning libraries Keras and TensorFlow. The model is able to correctly classify four different types of disposable and recyclable food service items (paper cups, paper boxes, paper trays, food containers) commonly found in the Queens University’s cafeteria. The classification is used by the hardware to provide a visual prompt to indicate the bin for a particular waste item. This can lead to improving the process of pre-sorting recyclable materials once the smart trash can is fully deployed on campus. The smart trash project has provided the author with the opportunity to learn about the sophisticated topics of deep learning and neural networks and use that knowledge to address the important of topic of recycling.*

1. **Introduction**

Many students at Queens University of Charlotte do not separate their waste properly despite the several recycling bins available on campus. Additionally, the majority of the collected waste material is not recycled and ends up in landfills. This unfortunate state of affairs has motivated the author of this capstone project to use using machine learning (ML), a branch of artificial intelligence, to improve the process of sorting recyclable items on campus through the creation of a ML image classification model. The ML model controls the action of a smart trash can which can be deployed to improve the waste sorting practice on campus.

Artificial Intelligence (AI) refers to the ability of machines to perform human-like tasks, such as learning from examples and experience, recognizing objects, understanding and responding to language, making decisions, solving problems, etc. Machine learning systems have the ability to automatically learn and improve from experience without being explicitly programmed. This capstone project focuses on the problem of image classification using a deep neural network, which is a supervised machine learning model, to tackle the problem of recycling.

The first part of this capstone report discusses the project’s hardware and framework. The training and test datasets, used to train and test the deep learning neural network, are introduced in the second part of this manuscript. The third part describes in detail the concept of deep learning and the theory of neural networks from a mathematical standpoint.

1. **Hardware**

We originally envisioned a way to both provide awareness to the user on what should be recycled versus what should not be and to correctly classify what the present object would be. In order to do this, we needed a *Raspberry Pi Model 3*, a Dorhea Raspberry Pi Mini Camera, PIR motion sensor, four different LEDs for each different class and four different recycling bins. The *Raspberry Pi* is the so-called “brains” of the operation as everything is saved and ran on its software. The *Raspberry Pi* will contain all of the code as well as the saved model; the *Raspberry Pi* would be attached to the rest of the hardware.

In order to take pictures of what objects need to be classified; a camera is used and positioned above the motion sensor. The PIR motion sensor was needed in order to sense any motion in front of the camera. This was done in order to provide a way for the raspberry pi to know when to snap a picture and run the model. Lastly, there are four LEDs (Red, Blue, Green, and Yellow) each for one of the different classes. The objective of the LEDs is to indicate which bin to place the object, this is accomplished by having the four different above a unique bin. This can be seen in the diagram below of the prototype that was used. The process of classifying is as followed:

a) An object is presented to the Raspberry pi

b) The Motion Sensor is triggered and notifies the Raspberry pi to take a picture

c) The Raspberry pi takes picture of the object and runs the loaded model

Once the model comes to a conclusion, internally the raspberry pi will return this statement “The

Classified object is a  \_\_\_\_\_” and light up the corresponding LED.

The user would then place the material in the bin with the corresponding LED, thus correctly sorting the material.

A picture containing indoor

Description automatically generated

**Figure 1: Smart trashcan prototype showing the system’s camera and LED indicators for each different waste bin.**

1. **Dataset**

The Recycling Dataset consists of a collection of images each from four different classes: Paper Cup, Paper Box/Bin, Paper Tray, Food Container. The dataset contains a total of 156 images split equally, consequently 48 images per recyclable class. These classes of images are necessary in order to train a model to distinguish the different recycling materials. Therefore, each class has a set of training images and testing images, with a 75/25 split (36 used for training and 12 images used for testing). The images are a collection of photos taken during the earlier stages along with a few stock images. The stock images are present in both Testing and Training Data. The images personally gathered, contain both the object and the noisy background; this was intentionally done in order to simulate what pictures a working prototype would be given. The stock images were later added in order to increase the accuracy of the model.

Four classes were decided upon observing which materials were disposed of in the most common area of campus. Initially, the coffee house and library were considered, but did not provide as much data with disposables as the cafeteria did. The idea would be that a prototype would be displayed there allowing for people to correctly sort the soon-to-be recycled objects. This exhibit would, in hope, reduce the amount of potentially recycled objects in landfills.

1. **Training and Testing images**

The quality of the training dataset was increased by including high quality images of the four waste items and as well as images of the same items under different orientation and lighting. The reasoning behind this was to simulate the condition that the objects could plausibly have, when presented to the prototype. This could be done by simply training a photo of a folded paper plate along with a photo of perfect plate as seen below in figure 2. This was done to all four classes to minimize error, simulate a natural, working environment, and to procure a high-quality dataset.

A picture containing white, dishware, tableware, ceramic ware

Description automatically generated A person holding a white plate

Description automatically generated with medium confidence A picture containing text, person, indoor, green

Description automatically generated

.

**Figure 2: Different images (orientation, shape, etc.) of plates from the training dataset to improve the training of the DL model.**

1. **Neural Networks and Deep Learning**

The term “Neural Network” is inspired by the biological neurons in the human body. Biological neurons are sophisticated nerve cells that use neurotransmitters to communicate and relay electrical impulses with other cells. A simplified neuron is displayed in Figure 3. An artificial neural network (ANN) is fundamentally a computer program that emulates the behavior and inner workings of the much more complex biological neural networks.

Diagram

Description automatically generated

**Figure 3: Sketch of communicating biological neurons.**

A diagram of a neural network can be seen below in Figure 4.

Arrow

Description automatically generated

.

**Figure 4: Diagram of a simple neural network. The arrows represent the neurons’ connections.**

All neural networks are constructed in layers with each layer containing a number of neurons (nodes) which are computational units. The neurons in same layer are not interconnected. Neurons are generally organized in three types of layers: the input layer, hidden layers, and the output layer. The term “deep learning” refers to neural networks comprising an input layer, an output layer, and multiple (more than 3) inner hidden layers.

In regard to our project, the input signal fed into the input layer is represented by the image taken by camera on the Raspberry Pi board. The output layer provides the classification result from the ML model. The hidden layers perform all the calculations that generate the output classification. Within the hidden layers, the neurons on each layer receive inputs from the previous layer sending their processed outputs to the neurons in the following layer. This process continues until the data reaches the output later. A sketch of a small neural network is provided in Figure 5.

All neurons in the hidden layers have associated weights and bias. Weights are the coefficients of the equation which aids in the image classification and solving equation [4]. When a neural network is trained on the training set, the values for the weights and bias strongly affect the result. The bias term is simply a constant value that is added to the product of inputs and weights, similar to adding a constant c to the solution of an integral problem in calculus [4]. This will be explained in greater detail later in the paper.

Diagram, schematic

Description automatically generated

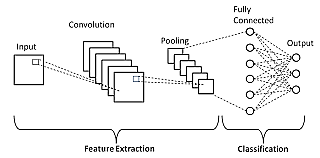
**Figure 5: Diagram of a 3-layer deep learning network. The three inner layers are the hidden layers, which are not visible.**

The early layers process the raw input data, and each subsequent layer is able to use information from neurons on the previous layer to process larger chunks of data [1]. For instance, if the prototype’s Neural Network receives a picture of a paper plate; the first layer looks at the individual pixels. The next layer would examine groups of pixels, and the succeeding layer would notice certain patterns (lines, curves). That data is given to the subsequent layer where that layer would reveal the image as a whole, this process would form lines, determine curvature, and would pick up general structures each layer. Eventually, a much later layer might reveal that the whole image shows a plate. When dealing with image recognition, convolutional neural networks typically obtains great results.

1. **Convolutional Neural Networks (CNN)**

**6.1 Convolutional Layer**

Convolutional neural networks (CNNs) emulate the way the human brain processes images. The visual classification of objects in the environment is performed by a sensory region of our brains called the visual cortex which receives the physical input signals and preprocess them. The visual cortex discards unnecessary information, adjusts/normalizes the input data and identifies relevant features and patterns (edges, spatial features, etc.) that will trigger neurons downstream. CNNs are fundamentally as an artificial application of these concepts.



A CNN is an artificial neural network that start with section dedicated to convolutional operations for the purpose of automatic feature extraction. The convolutional section of a CNN uses a set of convolutional filters which are applied to the input image to extract relevant features such as shapes, color area, and decrease the complexity of the information later passed to the neurons in the network. See Figure 6.

**Figure 6: Diagram of a 3-layer deep learning network. The three inner layers are the hidden layers, which are not visible.**

A CNN can have multiple convolutional layers that iteratively apply a transformation to the data through convolutional filters in matrix form. The Convolutional filters in each different convolutional layer capture specific higher-level features in the input image by looking at the relationships between each pixel and its neighboring pixels. The first convolutional layer and its filters capture low-level features such as edges, color gradients, and orientation, while the downstream convolutional layers detect more complex features such as objects, sizes, distances.

In general, a CNN can also include one or more pooling layers. Pooling layers reduce the dimensionality of the data by selecting the dominant features extracted by the upstream convolutional layers, Finally, the matrix of features extracted by the convolutional layers from the original data is flattened, i.e. converted to a 1D array, and passed an input to a fully connected neural network where a multitude of neurons performs the image classification.

**6.2 Convolution**

As mentioned, convolutional layers are named after the mathematical operation of convolution performed within a CNN. Convolution is a special type of product between two signals. In our case, the original input signal (an image) is convolved with a smaller 2D array (the convolutional filter). Each different filter is responsible for detecting a particular and different feature in the image.

To clarify these concepts, let’s consider an imagine of size 100x100 composed of a total of 10,000 pixels. Each individual pixel represents a feature that can be used to predict the object in the image. Directly feeding a 100-pixel image into the fully connected neural network can be computational very expensive since it would involve an input later with 10,000 input neurons!

Convolutional layers preprocess the input image, automatically extract the relevant features, and provide a lower dimensional data as input to the fully connected network can use to generate the prediction.

Each convolution operation is carried out by a 2D discrete matrix, the kernel, which shifts over different regions of the image and calculates an inner product with the pixels in that regions. The end result is a matrix that collects the results of this operation. Different kernels detect a variety of different features, such as lines and patterns. The various kernels are automatically designed and constructed by the model during the phase of training (automatic feature engineering). These patterns can make up a portion of a paper cup or any of the other four classes. When kernels are used in this way, it is referred to as filtering the input. The kernels slide across the width and height of the inputted image and extracts high level features.

Considering everything is a number when dealing with computers, the inputs are converted into numerical values. Imagine someone built a kernel to detect a straight-line segment; the kernel filter would resemble a 3X3 matrix with the second column having a value of 1 while the other columns would have a value of zero. The kernel filter would then slide across the image that it was given and multiply the values. This results in a new feature map; with a summation of the values, the output is determined by the highest sum. In diagram 5, the top filter is built to detect four separate lines while the bottom detects a continuous line. Unlike linear algebra, where the matrix multiplication involves multiplying the values according to their rows and columns, the values are simply multiplied by the value it is hovering. In other words, values are just multiplied element by element. This can be observed in the diagram below, where the sum of the second filter is higher indicating that there is a vertical line present in the input.

A picture containing text, electronics

Description automatically generated

**Input         Kernel Filter       Output**

Applying convolutional layers over an image results in a smaller image, this is important because this makes the image easier to compile as it is smaller and will require less computing power. Imagine there is an image of size 28 X 28, after applying a convolutional layer, there will be a new feature map of size 24 X 24. The more convolutional layers are applied, the more reduced the size of the image will be. A small amount of data is lost as a result of applying the convolutional layers; however, the sum of the kernel filters carries the weight of the data that was loss. To better explain, imagine there is an image with an original size of 7X7. During the convolution phase, a 3X3 filter is applied to the image, the filter is moved across the image, both vertically and horizontally. As the filter moves across the image it applies the element-by-element multiplication previously mentioned and stores the results in one of the features in the new features map. This exact diagram is provided below to further aid with understanding. Though data is typically lost with convolution, the weight from the filter is carried to the new feature map. The next step is to train the model.

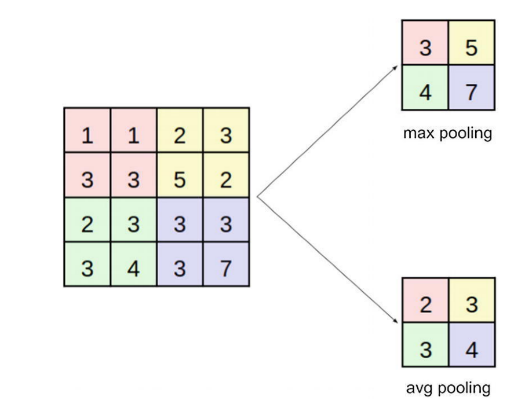
**Figure 7: From left to right: Imagine the first array is what was inputted, the Kernel filter, the second array, in this case could be looking for something resembling a line, on the far right is the output.**

**A picture containing text, crossword puzzle

Description automatically generated**

**Figure 8: In this diagram, a 3X3 filter, in red, is being applied to an image, in blue, and the results are being stored in specific values in the new feature map in the center. The original size of the image was 7X7 and the size of the new feature map is 5X5.**

**6.3 Pooling layer**

The output of a convolutional layer is connected to a pooling layer which is intended to reduce the dimensionality of the data without losing any relevant information needed for a correct classification. Two pooling functions are most commonly used:

Max pooling: Select the maximum/ highest value in the underlying subset of the input tensor.

Average pooling: Select the average value in the underlying subset of the input tensor.

**Figure 9: Pooling operations (max pooling and average pooling)**

**6.4 Fully Connected Layer**

The last part of a CNN is the fully connected neural network which receives the flattened matrix from the convolutional layers as input. The fully connected layer contains as many nodes in the output layer as the number of classes that the model classifies (4 nodes in our case). The dropout technique is often applied to the fully connected layer of a CNN to prevent overfitting

1. **Training the DL Model**

When a model is being trained, there are two methods that are typically used, supervised learning and unsupervised learning. Supervised learning is when a model is being trained and every sample has an associated label describing the class that we have manually assigned to it [1]. Unsupervised learning is the opposite, where there are no labels that are assigned; the model then works on its own to discover patterns and information. In the case of this project, supervised learning was used for the training. The training data have the labels associated with each picture, while the testing data does not.

The samples are given one at a time, in preassigned batches, to the model. For each sample, the model analyzes the features of the picture and predict its class. If the prediction is correct, the model moves on to the next image, if the prediction is not correct, the model adjust the algorithm used. This idea of learning from the mistake is known as backpropagation. The diagram below represents this concept visually (Diagram 7). Each time the models run through the entire training set, the model has trained for one epoch. It is important not to exceed a certain number of epochs as this can lead to overfitting.

**7.1 Underfitting, Ideal Fitting, Overfitting**

Having a model with too many epochs could result in overfitting, which is the production of an analysis that matches too closely or exactly to a particular set of data. The problem is that the system might be exploiting subtle relationships in the training data that aren’t true for data in general [1]. These subtle relationships could be finding a curve line and automatically assuming that the object is a plate. Another way to look at overfitting is the idea that the system has learned to cheat. In actuality the model learned a shortcut that produced the correct results. In this case the system is identifying the correct objects but not based on most of the features of the object in the picture.

Diagram

Description automatically generated

**Figure 10: In this diagram, the training set on the far left are ran through the loop individually. In this case, “Features” represents the image being used, and classifier is the model. Backpropagation in this diagram is noted in red.**

**7.2 Backpropagation**

Imagine someone decides to go for a run; however, they decided to run barefoot. When the run is over, they notice that their foot is full of cuts and bruises. They decide that the next time they decide to go for a run, they will use the proper shoes. This concept of learning from your mistakes is similar to the concept of Backpropagation. Backpropagation is a commonly used algorithm for supervised learning of artificial neural networks using gradient descent.

Gradient descent is simply used to find the values of a function's parameters or coefficients that minimize a cost function as far as possible [2]. The cost function, also known as the error function, is derived from the weights and bias of the neural network. To understand gradient descent, it is important to understand some of the notations that will be used.

Backpropagation works backwards, so the notation is based on the final layer (output), this will be denoted as *L* (layer), and the earlier layers are annotated with respect to *L* (*L* – 1*, L* – 2, . . . *L – n*) [5]. The weights, biases, and outputs from functions are subscripted appropriately with this same notation. ​To calculate the final layer by multiplying the preceding layer’s activation () by the weight *(*) and bias ) terms to produce the function of the final layer . The final layer is then passed through an activation function to produce a value between 1 and 0, this will later be probability that the object is one of the outputs. In diagram 8, a Neural network with the equation for the first neuron in the first layer or .

Diagram

Description automatically generated

**Figure 11: Diagram is providing a visual aid of the equation of the first neuron in the first layer ()**

Each weight is notated as such , where *L* is the respected layer, while k is the next neuron and j is the previous. This notation was done to indicate that the weight was being applied to the previous neuron and the result will be added to the bias and stored in the next neuron. All things considered, a ‘simple’ linear function can used to calculate the output of a simple three-layer neural network (input, hidden and output), Z = . That function is then passed to a non-linear function, called an activation function, to return a value between 1 and 0. The error function can be calculated by squaring the difference between the result from the last neuron and the output. These series of equations can be seen in diagram 9 below.

Text, letter

Description automatically generated

B.1 Represents the linear function used to calculate the output of a simple 3-layer neural network

B.2 Represents the activation function that returns a value between 1 and 0.

B.3 Represents the error function

In order to find the best possible weight value, that given any input would produce the correct output, having a near zero error value is critical. In diagram 10, Gradient descent, or the process of finding the minimum of the error, can be seen visually. The diagram below is a very simplified version of this as there is only one minimum value and as it is depicted in 2-D. When dealing with a complex neural network, finding the minimum value becomes more complicated as there will be local minimums, the more layers the more error values. In a neural network, Gradient descent is the process in which the model looks for a global minimum.

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 12: Diagram is displaying gradient descent on one error value.**

Backpropagation is an expression for the partial derivative ∂C/∂w of the cost function, represented by C, with respect to any weight (w) or bias (b) in the network [3]. In other words, backpropagation is calculus at its’ finest and also where the model truly learns. The purpose of backpropagation is to figure out the partial derivatives of the error function with respects to each individual weight in the network, that can be used during gradient descent. Recall the functions in diagram 9, it is important to remember that those functions are representing the outputs, due to backpropagation running back-to-front. From the output layer, the total error of the system is propagated backwards [5]. This function can be seen in diagram 11, this value will be notated as .

A picture containing text, watch, gauge

Description automatically generated

The gradient of the total error from the preceding layer

In order to update the weights in the output layer *L*, use the value from the error of the cost function, which is function B.3, in the equation for the derivative of the cost function with respect to the weights in layer *L*. This results in the function below B.6, in diagram 12.

Letter

Description automatically generated with low confidence

Since is equivalent to *∂C/∂* the equation can be further simplified

A picture containing text, watch, clock, gauge

Description automatically generated

This function is basically the relative amount by which the weights at layer *L* affect the total cost function. This value is used to update the weights at this layer. The next step is to continue down the rest of the layers. Layer *L* – 1 can be seen in diagram 13.

Text, letter

Description automatically generated

The total error is being backpropagated. Since is equivalent to *∂C/∂* the equation can be further simplified:

Text

Description automatically generated

After finding the gradient of the cost with respect to the weights at this layer *L* – 1, the following equation is left:

Text

Description automatically generated

After substituting *δL–1* for *∂C/∂* and taking the derivatives of the other terms, the function below is left:

Text

Description automatically generated

A neural network is largely a massive composite function, so each layer of a feed forward neural network can be represented by a single function. The inputs are the weight vales and the bias associated with each. An example of such a function is seen below, where σ is the activation function. This can also be represented in matrix notation as seen in diagram 15. A diagram with the updated knowledge can be seen in diagram 16.

*a*11 = σ(*w*111 *a*10 + *w*122 *a*20 + *w*133 *a*30 + *b*11)​

**Diagram 15:**

​Text, letter

Description automatically generated

In this example, Z represents the outputs, w represents the weights, a represents the neurons and b represents the bias.

Diagram

Description automatically generated

This diagram shows a Neural network the weights connecting a layer with four neurons, in layer 2, to a layer with three neurons (layer 3) made up a 4X3 matrix.

1. **Future Work**

The project would benefit by increasing the quantity of the classes in the dataset. This would be part of a more complex project that has the target of obtaining a model that can identify a much wider array of objects from images. The idea is that the model would be able to predict the materials Queens University of Charlotte pre-assigns, thus increasing the sustainability here at Queens University

The overall speed of the raspberry Pi could be improved in order to produce a result faster. Currently the raspberry Pi takes roughly 30 seconds to take a photo and run the saved model. This may not seem like much; however, we felt that if a user would have to wait 30 seconds just to receive a classification, it would not be practical. This is important because once the raspberry pi takes a picture of the object, we want to ensure that the user does not become impatient waiting for a result. This would be counter-productive because it could lessen the usage of the prototype.

1. **References**
2. Glassner, Andrew S. *Deep Learning: a Visual Approach*. No Starch Press, Inc., 2021.
3. Donges, Niklas. “Gradient Descent: An Introduction to 1 of Machine Learning's Most Popular Algorithms.” *Built In*, Niklas Donges, 16 June 2019, builtin.com/data-science/gradient-descent.
4. Nielsen, Michael A. “Neural Networks and Deep Learning.” *Neural Networks and Deep Learning*, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/chap2.html.
5. Malik, Farhad. “Neural Networks Bias And Weights.” *Medium*, FinTechExplained, 4 Mar. 2021, medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da.
6. Orland, Paul. *Math for Programmers: 3D Graphics, Machine Learning, and Simulations with Python*. Manning Publications Co. LLC, 2021.
7. Manganiello, Fabio. Computer Vision with Maker Tech (pp. 123-124). Apress. Kindle Edition.

1. **Source Code**
2. Graphical user interface, text

   Description automatically generatedThe required libraries are imported:
3. A picture containing logo

   Description automatically generatedA list with the four different classes is created:
4. Data Augmentation (training images are augmented to provide extra image variety to the model during the training phase):

Text, letter

Description automatically generated

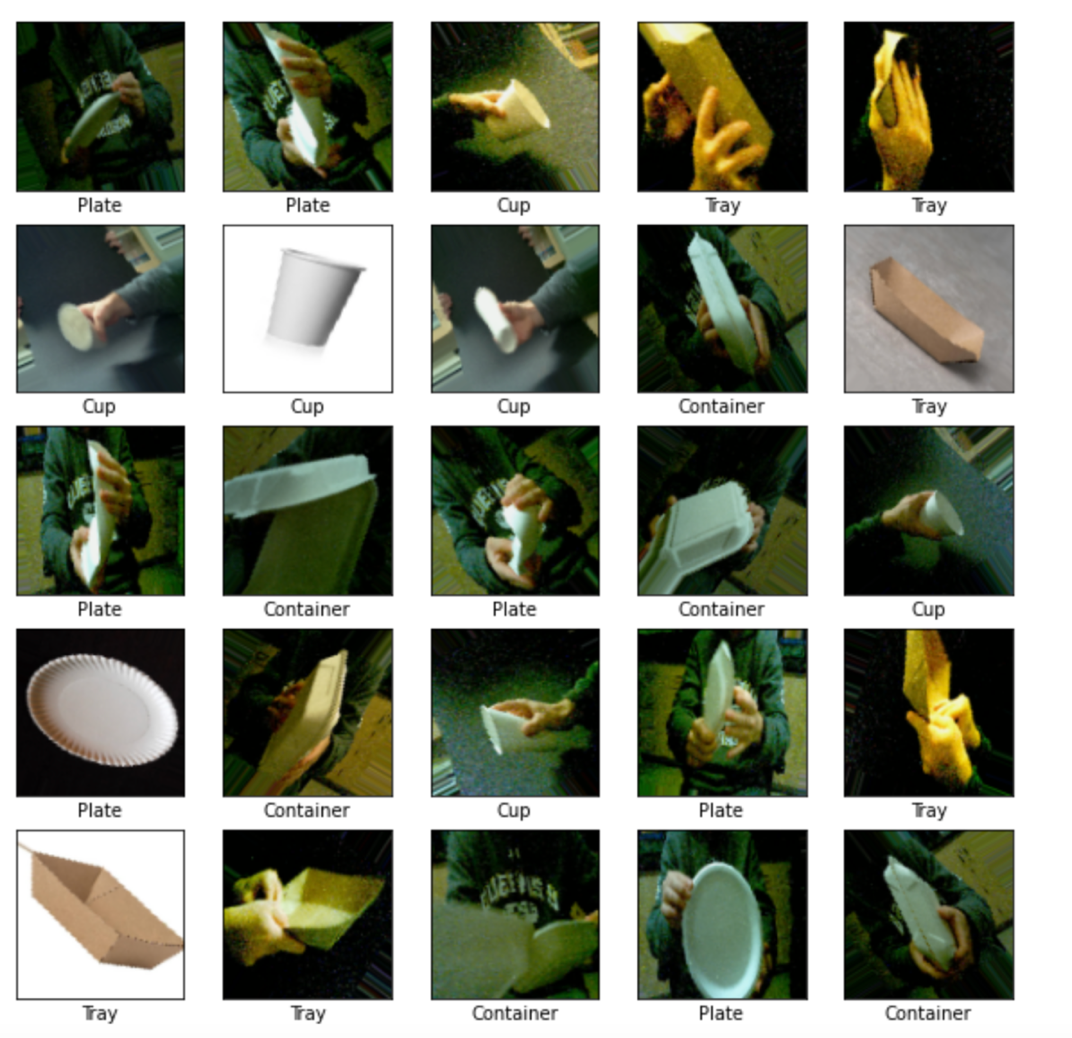
Graphical user interface, text, application

Description automatically generated

1. A subset of the images in the training dataset are visualized:

Graphical user interface, text, application

Description automatically generated



1. Text

   Description automatically generatedA sequential Keras model is created (two conv layers, two max pooling layers, one fully connected layer):
2. Graphical user interface, text, application, email

   Description automatically generatedThe model is trained through 10 epochs. Loss and accuracy are calculated at the end of the training phase:

Table

Description automatically generated

1. The model’s accuracy is graphically displayed after the training phase.

Chart, line chart

Description automatically generated

1. The model is tested on number of test images from the test dataset:

Graphical user interface, text, application, email

Description automatically generated

The mode correctly predicts the majority of the test images it was presented with (blue bars indicate correct predictions; red bars indicate an incorrect prediction):

Graphical user interface, application

Description automatically generated

The model is saved (architecture+weights) with the name *“Recycling.model.h5”* to be later deployed on new unseen images.

1. The saved model is loaded and visualized next:

Table

Description automatically generated

1. The saved model is applied to a new unseen image to test its performance. The test image is the image of a plate.

Graphical user interface, text, application

Description automatically generated

The model correctly predicts the image to be the image of a plate with 90% confidence.

Graphical user interface, text, application, email

Description automatically generated