

Verify Waste Android Application For Proper Waste Segregation

R.Hari Sankar⁽¹⁾, I.Bala Subbaiah⁽²⁾, V.Larisha Juhi Ackshaya⁽³⁾, S.Malathi⁽⁴⁾

^{(1) (2) (3)}UG Student, National Engineering College, Kovilpatti.

*⁽⁴⁾Assistant Professor in ECE Department, National Engineering College, Kovilpatti
National Engineering College, Kovilpatti*

Abstract:- Recycling and garbage separation are easy yet efficient strategies to cut down on the quantity of waste that ends up in our landfills. According to data, physical garbage sorting causes injuries (60.5%), eye problems (40%) respiratory difficulties (19%), dermatological disorders (22%), and skin problems (19%) among waste pickers. We create a model to distinguish between biodegradable and non-biodegradable garbage using a deep learning algorithm in order to mitigate the risk factors, hazards, and vulnerabilities that waste pickers encounter when gathering and sorting waste. To guarantee good accuracy, the approach must then be assessed by changing the optimization.

Keywords: Waste segregation, Convolutional neural network, Tensorflow

1.1 Introduction

In this area, it is of vital importance that computer algorithms are autonomously taught and developed. Deep learning, as opposed to machine learning, which uses simpler ideas, uses artificial neural networks, which are designed to mimic human thinking and learning processes. Until recently, the complexity of neural networks was limited by their processing power. Due to the development of Big Data Analytics, which makes it possible for computers to be more capable of monitoring, understanding and responding to complex situations in a way that humans couldn't, today we can achieve much higher levels of sophisticated neural networks. Deep learning has proven to be useful in speech recognition, language translation and the classification of images. It can run on its own, without being assisted by humans. Deep neural networks, or DNNs, are these kind of networks and each layer can perform a variety of complex tasks such as presentation and abstraction for the processing of text, sound and image data. The area of machine learning that has taken the fastest pace in terms of growth is considered to be deeper learning. Digital technology has the potential to fundamentally disrupt industries, which is why more and more enterprises are taking advantage of it in building new business models.

1.2 Objectives

'Verify Waste' is an Android application whose primary goal is to utilize one of deep learning's image classification techniques to accurately classify the user's input image and determine if it is biodegradable or not. Apart than sorting and separating rubbish photos, the 'Verify rubbish' app for Android seeks to raise awareness of environmental issues and support environmentally friendly waste disposal methods. It informs users on the significance of appropriate trash segregation and gives them feedback on how they dispose of their waste. By properly identifying biodegradable and non-biodegradable waste, this program will help reduce the amount of non-biodegradable waste ending up in landfills, thereby improving the quality of the environment.

2.1 Literature Survey

Concern should be expressed about the waste's rapid increase and the despicable disposal of it, which are endangering the ecology. This may be avoided and recycling made easy with the help of an automated trash segregator. Before waste is separated, its importance and economic value cannot be completely understood. There

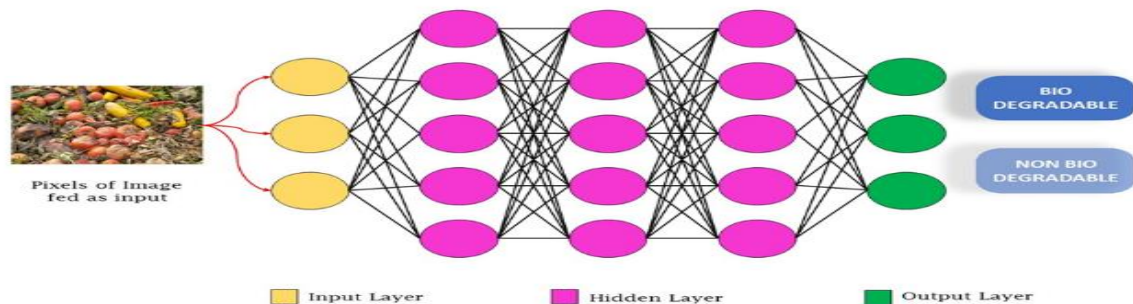
isn't currently a system in place to separate metal waste from other kinds of waste, including wet and dry waste. The concept proposes a spot automatic waste segregation device as a successful solution to this problem.

The RFID based Smart Dust Bin System is an early prototype of a future generation of dustbins that will be heavily equipped with sensors. The main focus of this concept is on security elements. A portion of the bombs detonated during the Delhi 2008 serial blasts were deposited in trash cans. Following the explosions, dustbins were taken out of every Delhi metro station. This allows explosives to be easily kept in dustbins for storage. We provide a workable solution to the problem of dustbins at metro stations in this paper. We have built a prototype model of this smart dustbin system using geared and servo motors, an RFID reader, an Arduino UNO, a Raspberry Pi, and a solar panel for power. The garbage will be monitored by the system through cloud-based monitoring tools. Utilizing a cloud-based solution eliminates the need for routine trash can inspections. In order to maintain Metro's carbon neutral footprint and keep the system ecologically friendly, we are using a modest solar panel as our power source.

India, the country with the second-highest population in the world, still faces challenges to the advancement of garbage management. An estimated 10 million tons of waste are produced annually in India's main cities alone. This work suggests classifying rubbish and classifying its type using Convolutional Neural Networks (CNNs) based on a well-defined and annotated data set of images that comprises the following categories: plastic, paper, cardboard, and metals. Images are categorized using a self-learning neural network based on their properties. The intended classifier is trained using the provided image data. The classifier employs the supervised learning technique, whereby an algorithm learns from a set of annotated data. By using this technique, testing accuracy is achieved to be 76%.

3.1 Image Classification

Image classification is the process of categorizing images. This is achieved by using similar qualities seen in photographs belonging to different classes to recognize and label photos. Images can be classified with the help of neural networks. Neural networks are algorithms used in deep learning. The graphic below shows neural networks.



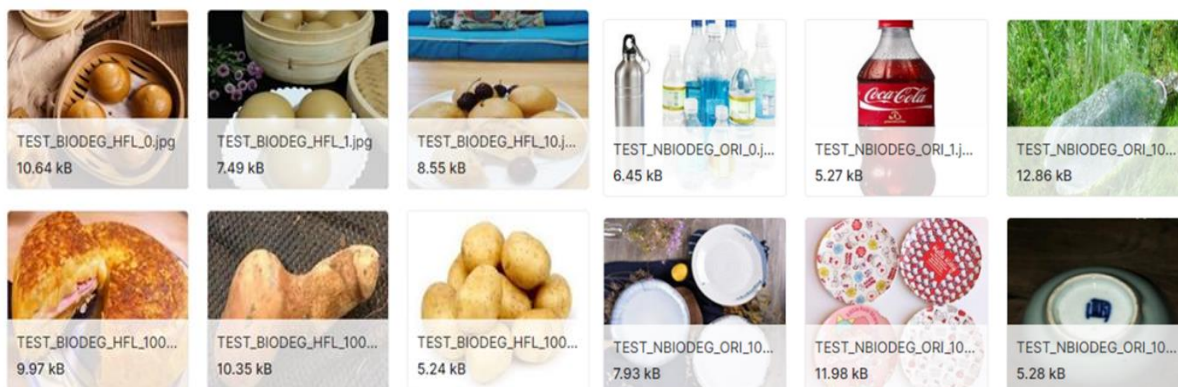
Neural networks use multiple layers of neurons to perform tasks like prediction and categorization. Up until the final output layer, every neuron in the layer below it transmits a signal to the neurons in that layer, which helps to improve the output. The layers of a neural network are as follows: Data input for your neural network is obtained at the input layer. The many neurons that make up the hidden layer process the data from the input layer. The output layer is the final layer in the network that provides you with the output after processing your input one last time. Using a deep learning package like Pytorch, Tensorflow, or Keras makes neural network implementation easy. You will be using Keras to implement your neural network.

3.2 Kaggle Image Classification Dataset

This collection's 256K images (156K raw data) are separated into biodegradable and non-biodegradable groups.

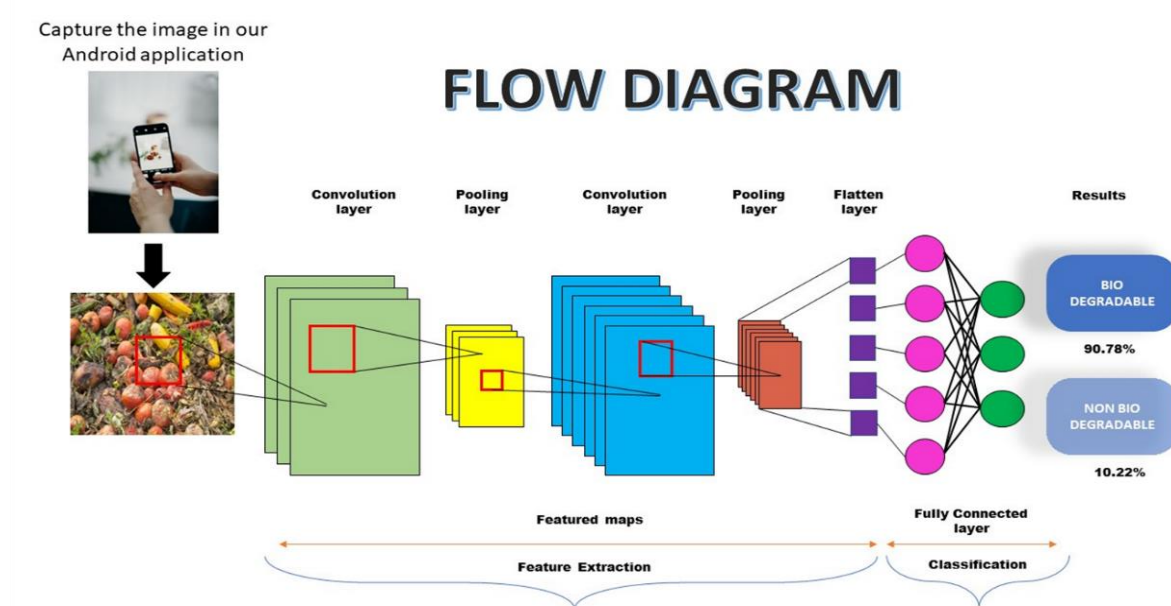
Foods, plants, fruits, and other biodegradable things can be broken down by bacteria in a natural way. The trash from this material can be turned into compost. Materials that are not biodegradable include plastics, metals,

inorganic elements, and other materials that do not decompose on their own. Waste from this substance will be recycled to make new materials. We include augmented data that has been improved by original data manipulation to offset the unequal class. In order to manipulate photos, rotations of 90 degrees CW and 90 degrees CCW are applied, along with flipping images both horizontally and vertically. Training and assessment sets comprise the two subsets of the data in this dataset. Some technical concerns led to the training set being divided into four pieces. It is imperative to acknowledge that a portion of the training set's data exhibits poor distribution. Therefore, refrain from sending every component directly to your model. Create a single dataset by combining all the components. See Quickstart for more information.



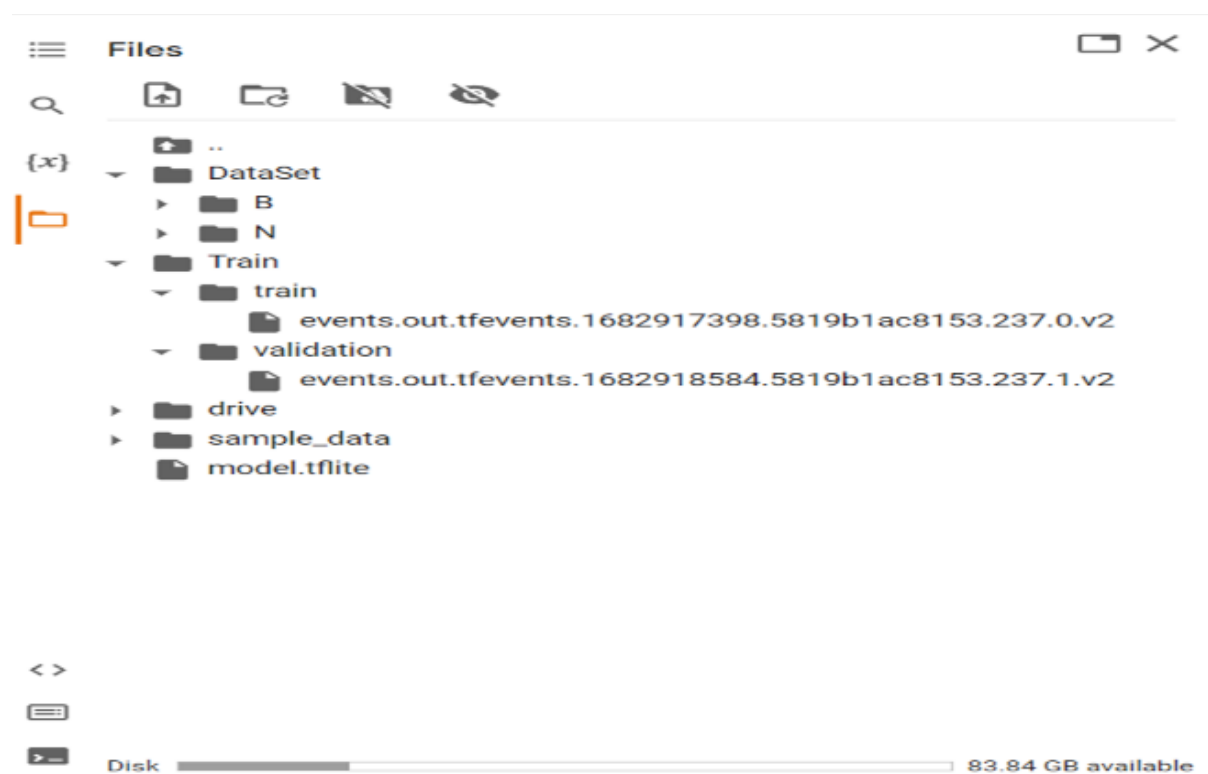
This dataset's data files are named uniquely to avoid overwriting one another during concatenation. The filename reference is below. This is necessary in order to filter this dataset.

3.3 Block Diagram



3.4 Loading the Data

To unzip the dataset zip file that we want to classify, use the following Python code. The dataset contains two folders, each of which has images that are either biodegradable or non biodegradable.



Three libraries are imported: Matplotlib, NumPy, and TensorFlow.

Neural networks can be constructed and trained using the open-source TensorFlow machine learning framework. NumPy is a Python numerical computing toolkit that supports matrices and arrays and makes computations on big datasets quick and easy. A plotting library called Matplotlib offers capabilities for making graphs and visualizations. These libraries can be imported so that the code can use their functions and methods to accomplish a variety of tasks involving data visualization, numerical computation, and machine learning. Python code that makes use of the `image_dataset_from_directory()` function in the TensorFlowKeraslibrary to construct an image dataset from a directory. The path to the directory holding the image files is the only argument that the function accepts. The directory path in this instance is `"/content/DataSet"`. The function automatically generates labels for each image based on the subdirectory that the image is stored in. the directory `"/content/DataSet"` contains subdirectories `"Bio Degradable"` and `"Non Bio degradable"` and each subdirectory contains images of Bio Degradable and Non Bio Degradable images, respectively, then the function will generate labels of `"Bio degradable"` and `"Non Bio Degradable"` for each corresponding image.

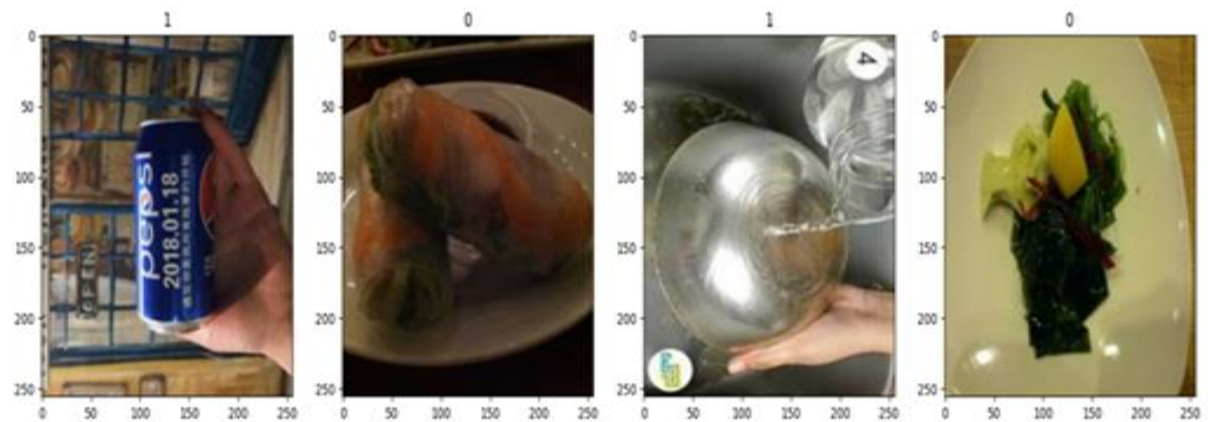
Once generated, the dataset object is returned and can be utilized for additional processing, like machine learning model training. Python code that shows the first four photographs in a batch of data shown with the labels that go with them. The data object is a dataset that has been split up into separate training, validation, and testing sets. Use the `as_numpy_iterator()` function on the data object to create a data iterator that returns the data as NumPy arrays. The next batch of data, which is kept in the batch variable, is obtained by calling the `next()` method on the data iterator.

A figure with four columns of subplots, each with a width and height of 20, is created using the `subplots()` method. Each subplot's axes and figure are represented, respectively, by the resulting `fig` and `ax` objects.

The batch of data's first four photos are iterated over using the `enumerate()` method. The current image's index is stored in the `idx` variable, while the image data is stored in the `img` variable as a NumPy array.

To display an image, its corresponding subplot axis `ax[idx]` is called with the `imshow()` method. To convert the pixel values to integers, the `astype()` method is invoked on the picture data with the input `int`. To set the title of

the subplot to the relevant label for the image, which is retrieved from the batch[1] variable using the idx variable as the index, the title.set_text() method is invoked on the subplot axis.



3.5 Scale the Data

Python code that uses the map() method to apply a lambda function to each element of the data object. The image data and the associated label are represented by the two arguments that the lambda function accepts, x and y, respectively. The original label y and the picture data divided by 255 (to normalize the pixel values) make up the tuple that the lambda function returns. A changed dataset with each element being a tuple comprising a normalized image and its matching label is the resultant data object. To generate a data iterator that returns the data as NumPy arrays, call the as_numpy_iterator() method on the data object. The next batch of data, which is kept in the batch variable, is obtained by calling the next() method on the data iterator. The first components of the updated data object are included in this batch; all picture pixel values have been normalized to fall between 0 and 1.

3.6 Divide Information for testing and training

```
train_size = int(*.7)
```

```
test_size = int(*.1)
```

```
val_size = int(*.2)
```

```
train_size = int(*.1)
```

The Python method mentioned above determines, as a percentage of the dataset's overall size, the sizes of the training, validation, and testing sets. We apply the len() function to the dataset data in order to get the total number of samples in the dataset. Multiply this amount by a decimal number to determine each set's required size. The size of the training set will be determined by taking the integer value of 70% of the overall dataset size, which is what the train_size variable is set to. The size of the validation set will be determined by taking the integer value of 20% of the overall dataset size, which is what the val_size variable is set to. The size of the testing set will be decided by using the test_size variable, which is set to an integer value equal to 10% of the size of the entire dataset. By partitioning the dataset into separate training, validation, and testing sets, the technique may evaluate how well a machine learning model performs on untrained data. This can help prevent overfitting and ensure that the model can be used with new data. Python code that partitions a dataset into separate training, validation, and testing sets based on the sizes decided in the code block before it. The take() function builds a new dataset train with the initial train_size number of samples using the existing dataset data. This dataset is meant to be used as machine learning model training material. Next, using the train_size argument, the skip() method is performed on the original dataset data, skipping the first train_size amount of samples. This subset of the original dataset is again subjected to the take() method, yielding a new dataset val containing the subsequent val_size number of samples. The machine learning model's performance during training will be verified using this dataset. Lastly, the first train_size+val_size number of samples is skipped when the skip() method is invoked again on the original

dataset data, this time with the input `train_size+val_size`. This subset of the original dataset is used to invoke the `take()` method, which generates a new dataset `test` with the `test_size` number of samples that remain. After training, this dataset will be used to evaluate the machine learning model's performance.

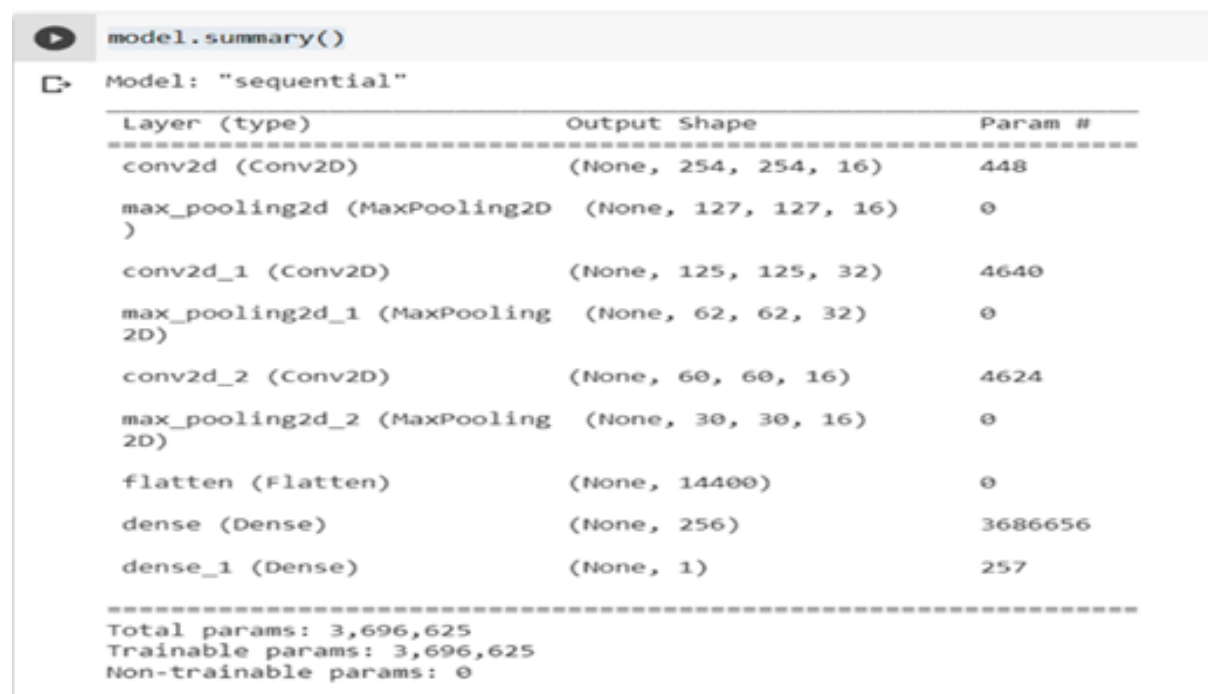
3.7 Build Deep Learning Neural Network

A convolutional neural network (CNN) model is created using Python code that makes use of the high-level neural networks API Keras. Using the `Sequential()` function, the layers of the model are built in a linear stack. We use the `Conv2D()` method to add a 2D convolutional layer to the model. With a 256x256 RGB image, the input shape of the layer is (256,256,3), as indicated by the `input_shape` option. The `MaxPooling2D()` function adds a max pooling layer to the model. This layer reduces the spatial sizes of the feature maps in the convolutional layer. Two further convolutional layers and max pooling layers with settings similar to the first convolutional layer are added to the model. Use the `Flatten()` function to turn the last convolutional layer's output into a 1D array. Two fully connected layers are added to the model by using the `Dense()` function. The first dense layer uses the ReLU activation function with 256 neurons. The sigmoid activation function, suited in the second dense layer with a single neuron, is employed for binary classification applications. For binary classification problems, including image classification tasks where the objective is to categorize an image as belonging to one of two classes, this model—a straightforward CNN architecture—can be applied. Next, we go on to the compilation phase. The model's learning procedure is configured using the `compile()` method. Three parameters are required:

Optimizer: the model's training optimization algorithm. Here, the Adam optimizer is applied.

Loss: The loss function is used to evaluate the model's performance. In this case, the binary cross-entropy loss function is used. This is a typical loss function for binary classification issues.

Measures: A collection of evaluation metrics used to monitor the model's performance throughout training and testing. In this case, only the accuracy metric is used. Once the model is constructed, it may be trained on a dataset using the `fit()` function.



```
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 16)	448
max_pooling2d (MaxPooling2D)	(None, 127, 127, 16)	0
conv2d_1 (Conv2D)	(None, 125, 125, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 32)	0
conv2d_2 (Conv2D)	(None, 60, 60, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 16)	0
flatten (Flatten)	(None, 14400)	0
dense (Dense)	(None, 256)	3686656
dense_1 (Dense)	(None, 1)	257
Total params: 3,696,625		
Trainable params: 3,696,625		
Non-trainable params: 0		

3.8 Train the Model

Before you train the model, during the model fitting phase, decide where to store the training as well as testing data. The variable `logdir` specifies the path where TensorBoard logs will be stored during training. The directory

path can be changed to a new location if needed. The program that trains the previously developed CNN model using the Keras API. The fit() function is used to train the model on a dataset.

Train: The model will be trained using the data that has been collected. The number of training epochs, or iterations over the entire dataset, that need to be completed.

Validation_data: The validation data that will be utilized to track the model's performance throughout training.

Callbacks: A set of callback operations to be carried out throughout the training process. In this instance, the logs are saved to the directory indicated by logdir using a TensorBoard callback function.

When the training is complete, Fit() creates a History object containing information about the training procedure, including the loss and accuracy values for each epoch. This item is kept in the hist variable by this code. The output of the CNN model's last training epoch is displayed in the figure below.

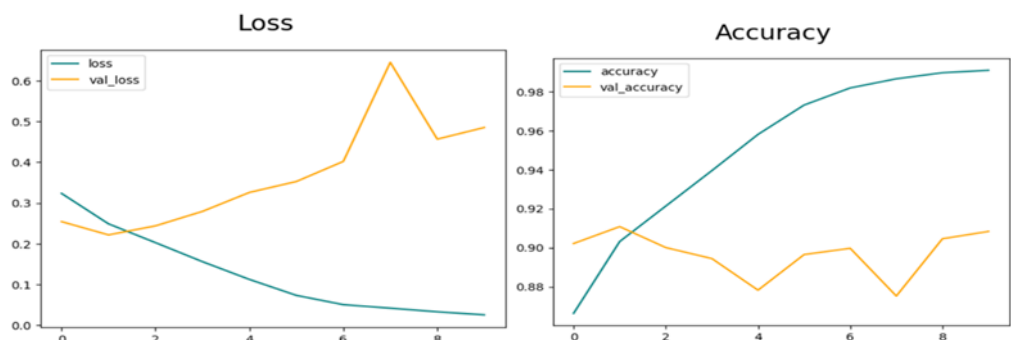
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
hist = model.fit(train, epochs=10, validation_data=val, callbacks=[tensorboard_callback])
```

```
Epoch 1/10
581/581 [=====] - 1328s 2s/step - loss: 0.3234 - accuracy: 0.8663 - val_loss: 0.2542 - val_accuracy: 0.9021
Epoch 2/10
581/581 [=====] - 1265s 2s/step - loss: 0.2487 - accuracy: 0.9031 - val_loss: 0.2215 - val_accuracy: 0.9108
Epoch 3/10
581/581 [=====] - 1300s 2s/step - loss: 0.2025 - accuracy: 0.9213 - val_loss: 0.2437 - val_accuracy: 0.9000
Epoch 4/10
581/581 [=====] - 1303s 2s/step - loss: 0.1559 - accuracy: 0.9395 - val_loss: 0.2792 - val_accuracy: 0.8944
Epoch 5/10
581/581 [=====] - 1292s 2s/step - loss: 0.1122 - accuracy: 0.9580 - val_loss: 0.3258 - val_accuracy: 0.8782
Epoch 6/10
581/581 [=====] - 1304s 2s/step - loss: 0.0730 - accuracy: 0.9731 - val_loss: 0.3527 - val_accuracy: 0.8965
Epoch 7/10
581/581 [=====] - 1306s 2s/step - loss: 0.0502 - accuracy: 0.9819 - val_loss: 0.4020 - val_accuracy: 0.8997
Epoch 8/10
581/581 [=====] - 1282s 2s/step - loss: 0.0417 - accuracy: 0.9866 - val_loss: 0.6452 - val_accuracy: 0.8752
Epoch 9/10
581/581 [=====] - 1290s 2s/step - loss: 0.0327 - accuracy: 0.9897 - val_loss: 0.4566 - val_accuracy: 0.9046
Epoch 10/10
581/581 [=====] - 1295s 2s/step - loss: 0.0252 - accuracy: 0.9909 - val_loss: 0.4852 - val_accuracy: 0.9083
```

It shows the accuracy and loss figures for both the training and validation datasets. In this case, the model yielded a loss value of 0.0252 and an accuracy of 0.9909 on the training dataset, along with a validation accuracy of 0.9083 and validation loss of 0.4852.

3.9 Plotting Accuracy

To plot the accuracy of the model, create the plot_accuracy_loss() function. The information that your neural network returns also includes the final accuracy and model loss. It depicts the accuracy of the training set and validation set for each epoch so you may understand the variance in your performance. Along with this, it also graphs the validation loss and loss. Examine the accuracy and loss of your model with the test data that is supplied below. The charts demonstrate how the accuracy of the model increases with each epoch for both the training and testing sets. With every epoch, the loss of your model decreases as it learns and gets better.



3.10 Using TFLite, save the model

The trained Keras model is converted into a TensorFlow Lite model during this process. Using the `from_keras_model()` method, we first generate a TFLiteConverter object from the Keras model. The Keras model is then converted to a TensorFlow Lite model by calling the `convert()` method on this converter. Finally, write the produced TensorFlow Lite model to a file named "model.tflite" using the `write()` method.

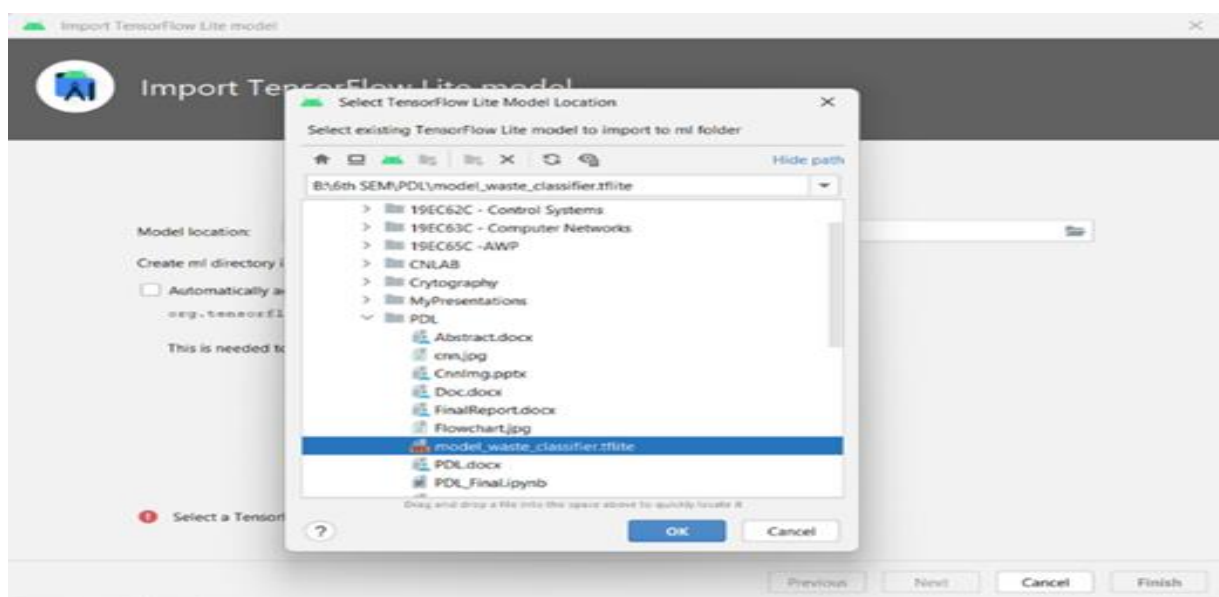
4.1 Put the Model

To import the model that was downloaded earlier from Google Colab into Android Studio, take the following actions:

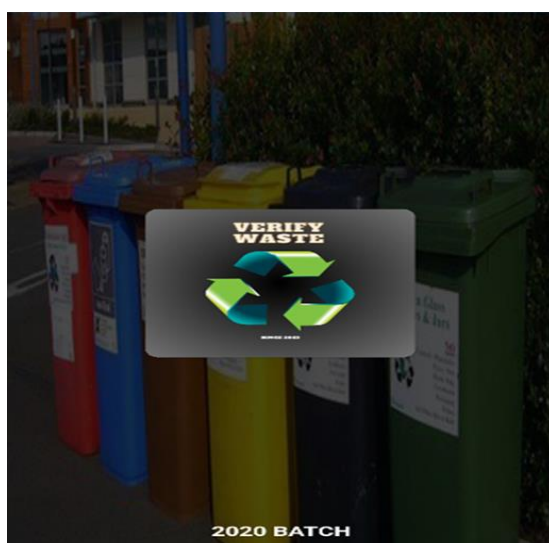
To open Android Studio, choose the app with a right-click.

To import the model, select "Other" and then click "Tensorflow Lite Model".

Use the imported model as the backend in your code.



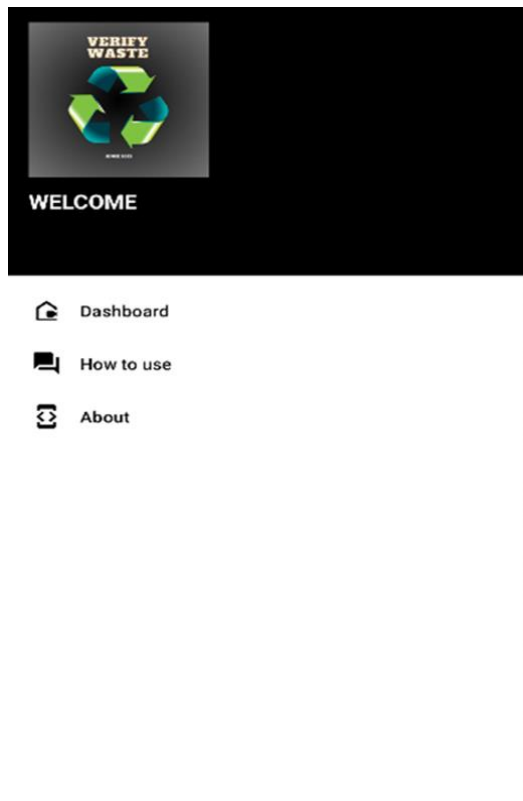
4.2 User Interface Front End



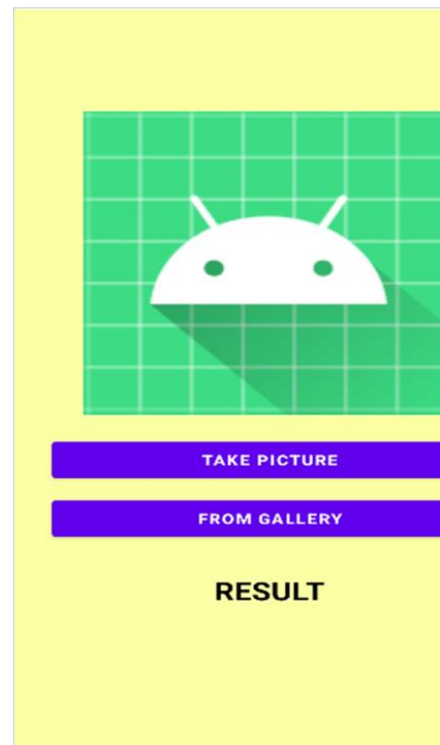
Welcome Activity



DashBoard Fargment



Navigation Drawer



Main Activity



App Logo

These are all main pages of our android application. That are created using XML language.

4.3 Functionality (Backend By Kotlin)

The MainActivity class, which implements AppCompatActivity and offers a layout with buttons, an image view, and text views, is defined in the Kotlin code. The onCreate() function sets click handlers to the buttons and

initializes these UI components so that users can choose between taking a picture or choosing one from their device's gallery. Firstly, before moving on to the activity of our code, we define our package name as kotlin code. It that includes several imports and defines an AppCompatActivity that provides a layout with buttons, an image view, and text views. The code uses the Android MediaStore to allow users to select an image from their device's gallery, and then processes and classifies the image using a TensorFlow Lite model named ModelWasteClassifier. A TensorFlow Lite model called ModelWasteClassifier is used by the classifyImage() function to process and categorize the chosen image. The layout presents the user with the classification findings. The chosen image is retrieved by the onActivityResult() method, which resizes it to 256 by 256 pixels before passing it to classifyImage() for classification.



References

- [1] R. Sultana, R. D. Adams, Y. Yan, P. M. Yanik and M. L. Tanaka, "Trash and Recycled Material Identification using Convolutional Neural Networks (CNN)," 2020 SoutheastCon, Raleigh, NC, USA, 2020, pp.1-8
- [2] Tripathi, C. Pandey, A. Narwal and D. Negi, "Cloud Based Smart Dustbin System Metro Station," 2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU), Bhimtal, India, 2018, pp. 1-4
- [3] S. Murugaanandam, V. Ganapathy and R. Balaji, "Efficient IOT Based Smart Bin for Clean Environment," 2018 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2018, pp. 0715-0720
- [4] R. K. Singhvi, R. L. Lohar, A. Kumar, R. Sharma, L. D. Sharma and R. K. Saraswat, "IoT Based Smart Waste Management System: India prospective," 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU), Ghaziabad, India, 2019, pp. 1-6
- [5] S. Dubey, M. K. Singh, P. Singh and S. Aggarwal, "Waste Management of Residential Society using Machine Learning and IoT Approach," 2020 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2020, pp. 293-297
- [6] M. Jayson, S. Hiremath and L. H.R., "SmartBin-Automatic waste segregation and collection," 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAEECC), Bangalore, India, 2018, pp. 1-4
- [7] Muniandi, B., Huang, C., Kuo, C., Yang, T., Chen, K., Lin, Y., Lin, S., & Tsai, T. (2019). A 97% maximum efficiency fully automated control turbo boost topology for battery chargers. IEEE Transactions on Circuits and Systems I-regular Papers, 66(11), 4516–4527. <https://doi.org/10.1109/tcsi.2019.2925374>
- [8] M. G. C. P, S. Yadav, A. Shanmugam, H. V and N. Suresh, "Waste Classification and Segregation: Machine Learning and IOT Approach," 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2021, pp. 233-238
- [9] N. S. Gupta, V. Deepthi, M. Kunnath, P. S. Rejeth, T. S. Badsha and B. C. Nikhil, "Automatic Waste Segregation," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2018, pp. 1688-1692
- [10] S. R., R. P., V. S., K. R. and G. M., "Deep Learning based Smart Garbage Classifier for Effective Waste Management," 2020 5th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2020, pp. 1086-1089
- [11] H. Dessai and D. Miranda, "Garbage Segregation Using Images," 2021 2nd International Conference for Emerging Technology (INCET), Belagavi, India, 2021, pp.1-4