GARBAGE CLASSIFICATION USING CONVOLUTION NEURAL NETWORKS

Submitted to:

Dr. Rashmi Gupta
Deep Learning

Submitted by:

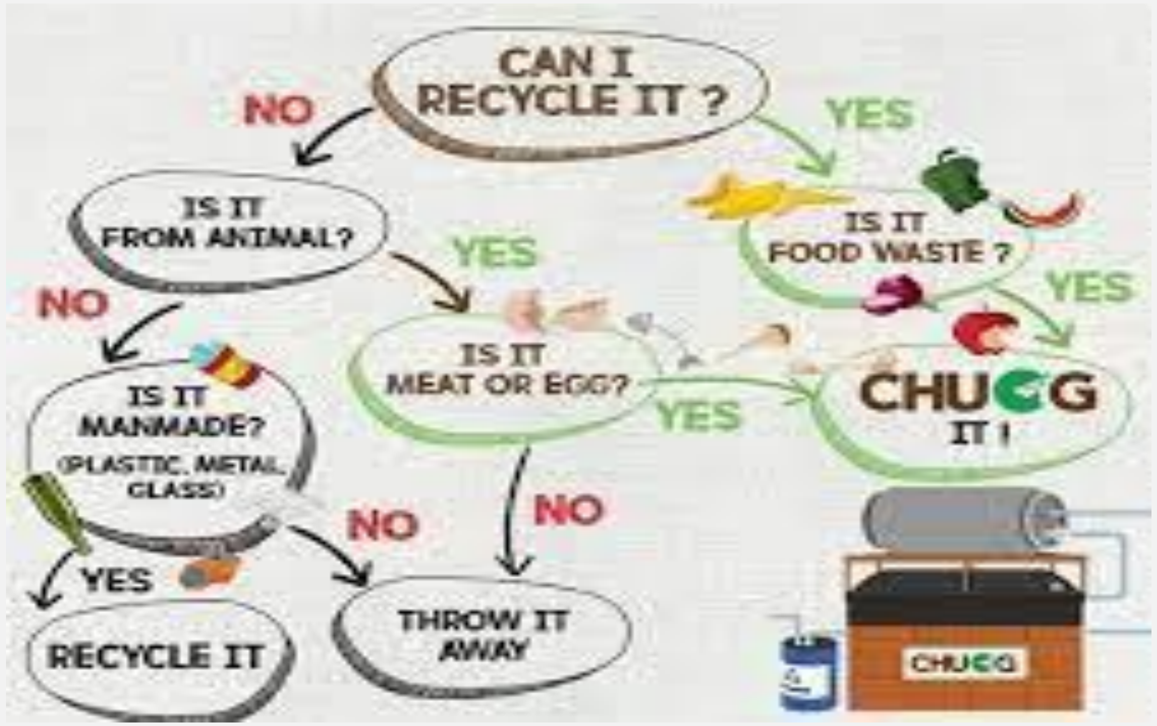
Naman

ECAM-2

2020UEA6598

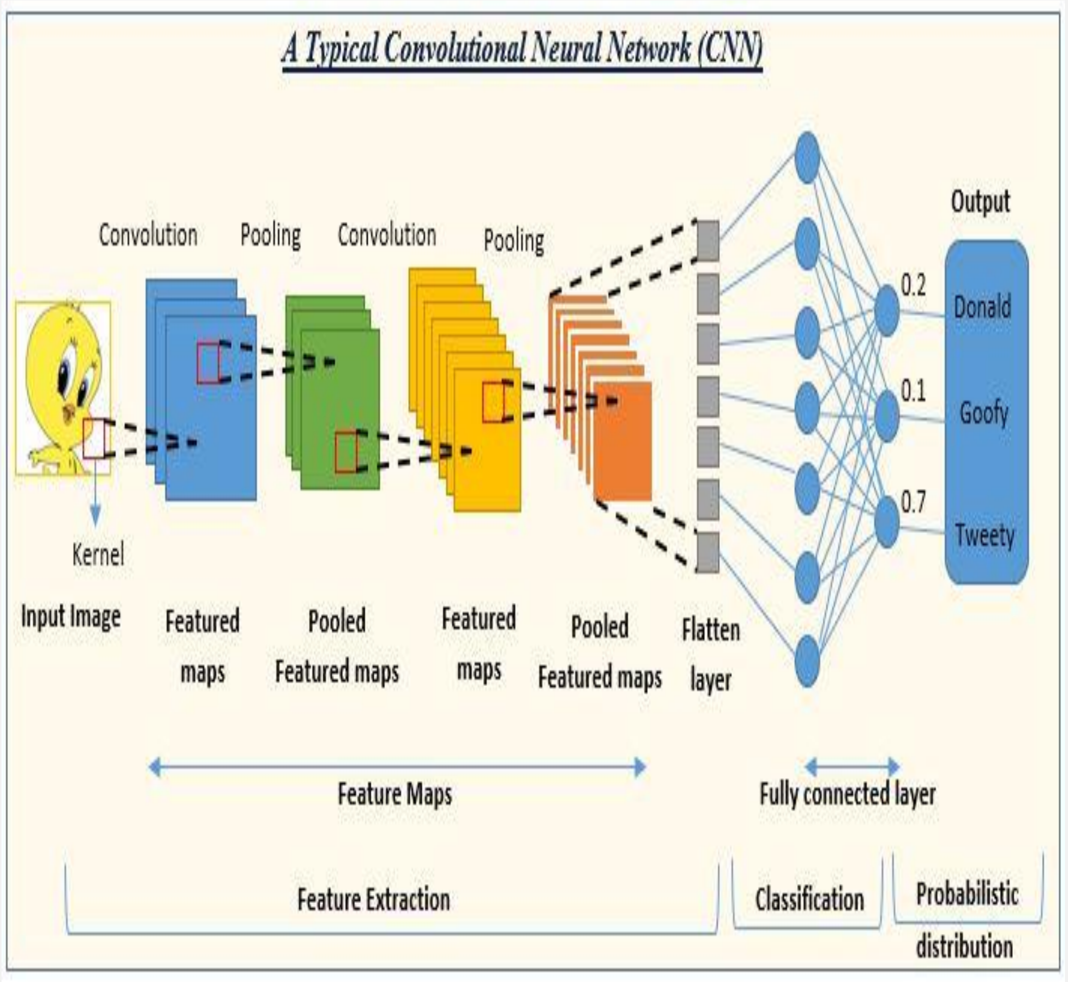
OBJECTIVE

- To develop a CNN model that can classify garbage images with high accuracy.
- To differentiate between recyclable and organic garbage.
- Recycling helps prevent waste of useful materials, reduces energy consumption, air pollution, and water pollution.
- To promote sustainable waste management practices through efficient garbage classification.



THEORY BEHIND A CNN

- Convolutional Neural Networks (CNNs) are a type of deep learning algorithm used for image and video recognition, natural language processing, and many other applications.
- The fundamental idea behind CNNs is to mimic the visual processing of the human brain, where neurons respond to specific regions of the visual field.
- CNNs are made up of multiple layers, each with a specific task



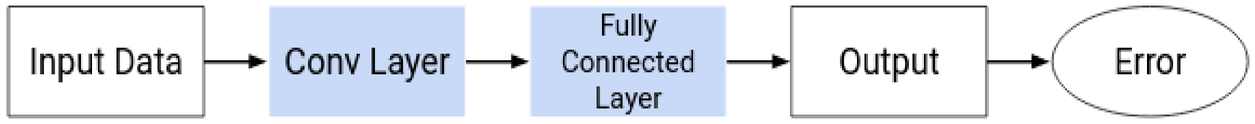
CNN ARCHITECTURE

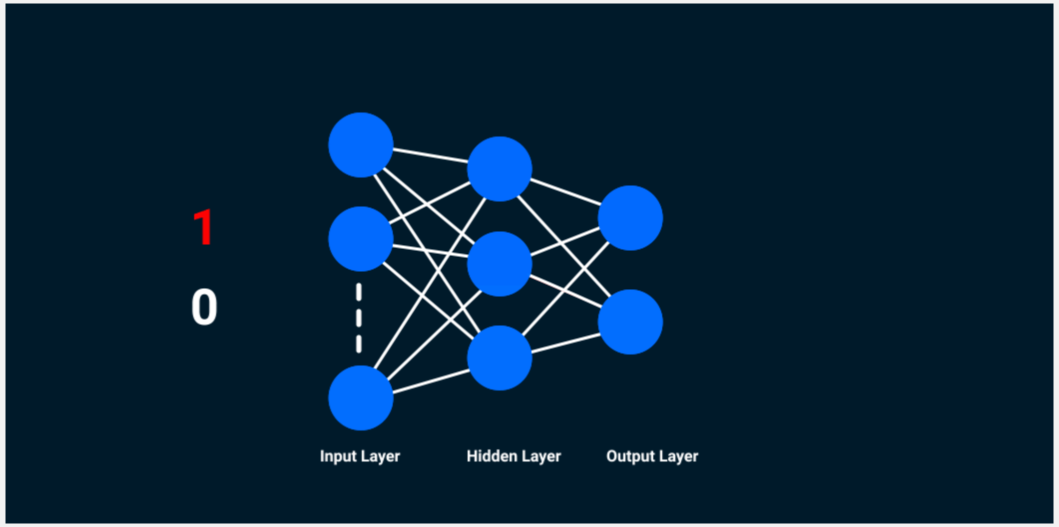
The convolutional neural network can be broken down into two parts:

- **The convolution layers**: Extracts features from the input
- **The fully connected (dense) layers**: Uses data from convolution layer to generate output

There are two important processes involved in the training of any neural network:

- **Forward Propagation:** Receive input data, process the information, and generate output
- **Backward Propagation:** Calculate error and update the parameters of the network

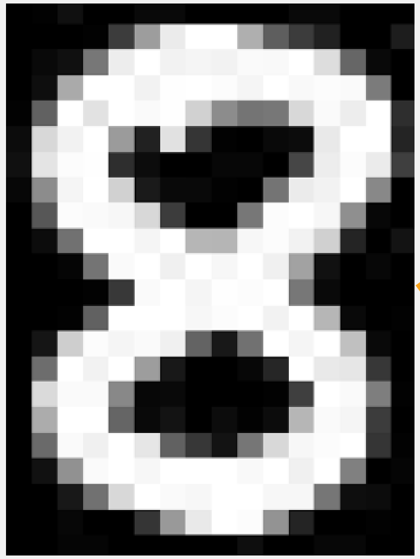


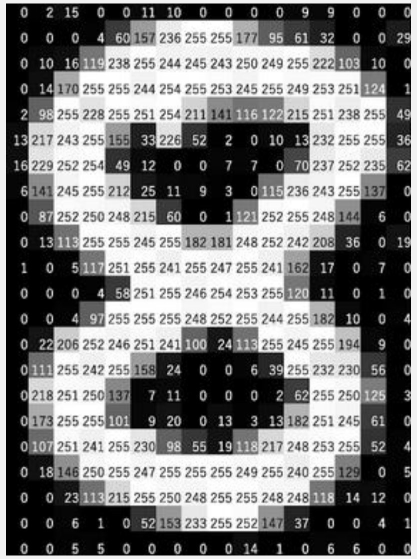


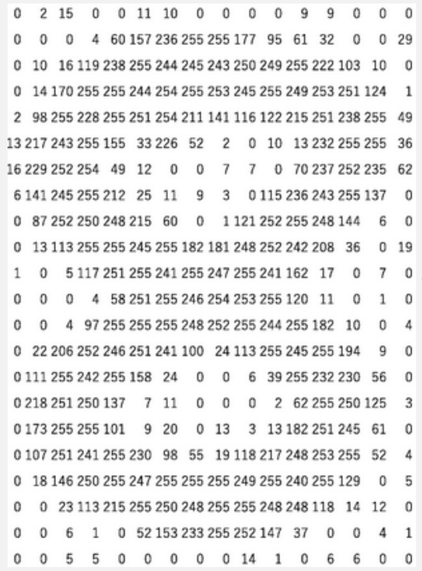
CONVOLUTION LAYER

A convolutional neural network looks at images and identify the object's shape and edges by comparing the pixel values.

- On right hand side is an image of the number 8 and the pixel values for this image.
- We can see that there is a significant difference between the pixel values around the edges of the number.
- Hence, a simple way to identify the edges is to compare the neighboring pixel value.



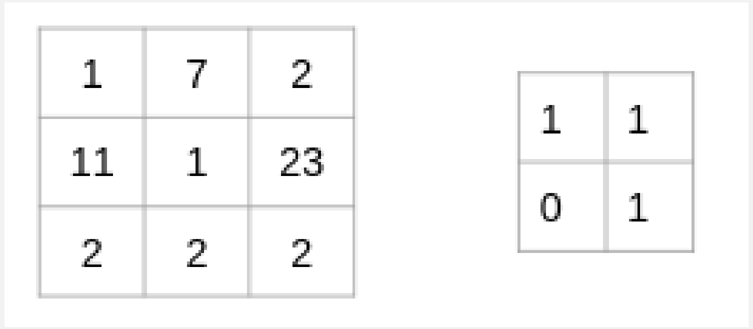




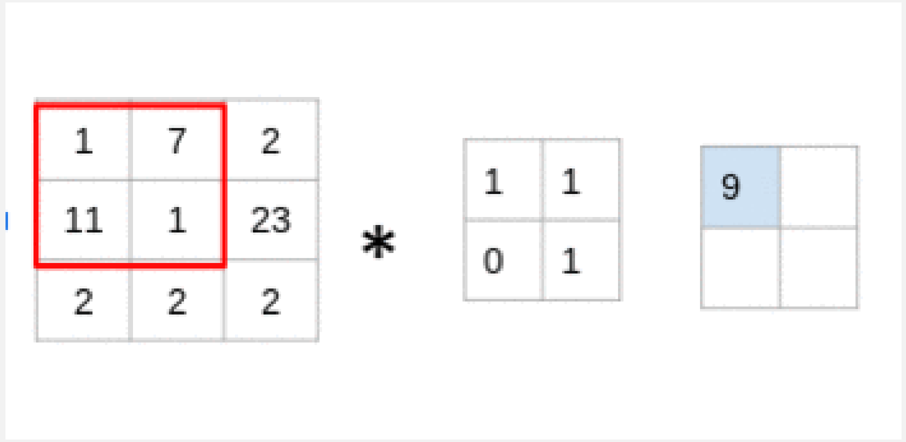
0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

MATH AND PROCESS BEHIND A CONVOLUTION OPERATION

- Consider that we have an image of size 3 x 3 and a filter of size 2 x 2:

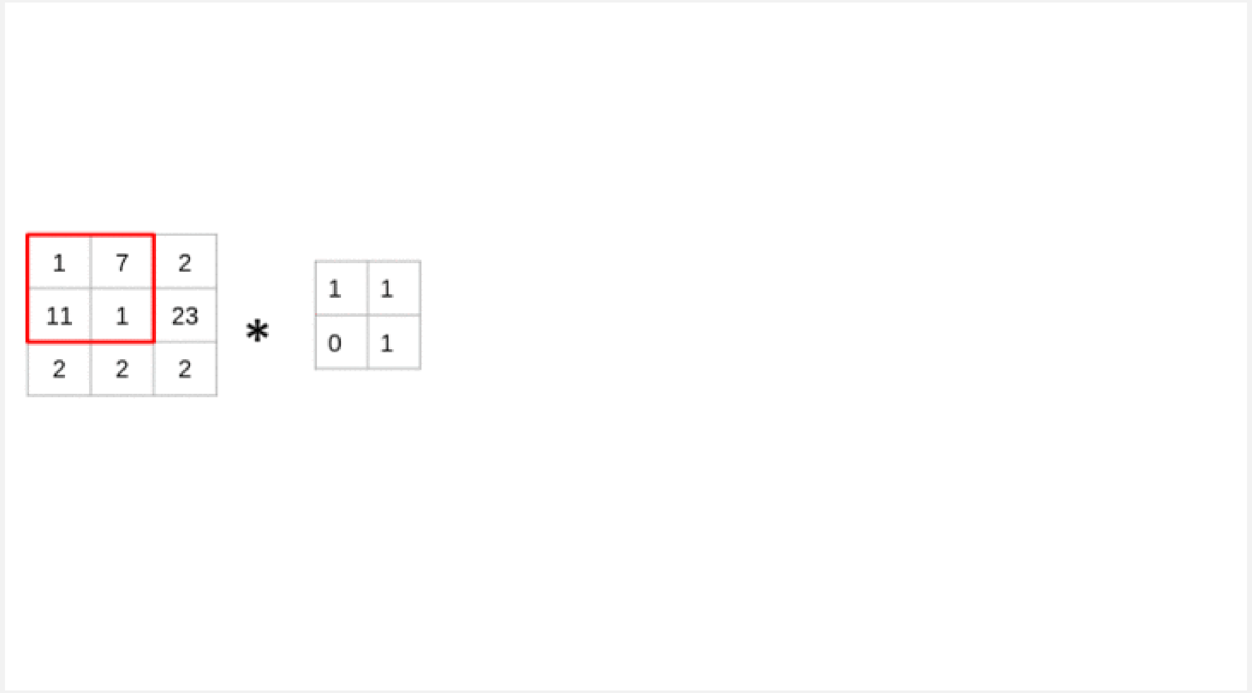


- The filter goes through the patches of images, performs an element-wise multiplication, and the values are summed up:

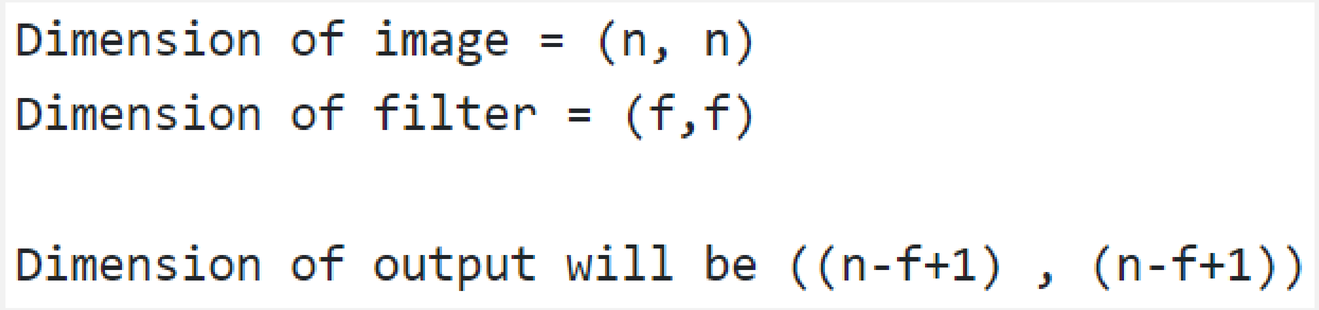


```
(1x1 + 7x1 + 11x0 + 1x1) = 9
(7x1 + 2x1 + 1x0 + 23x1) = 32
(11x1 + 1x1 + 2x0 + 2x1) = 14
(1x1 + 23x1 + 2x0 + 2x1) = 26
```

- The filter is considering a small portion of the image at a time. We can also imagine this as a single image broken down into smaller patches, each of which is convolved with the filter.



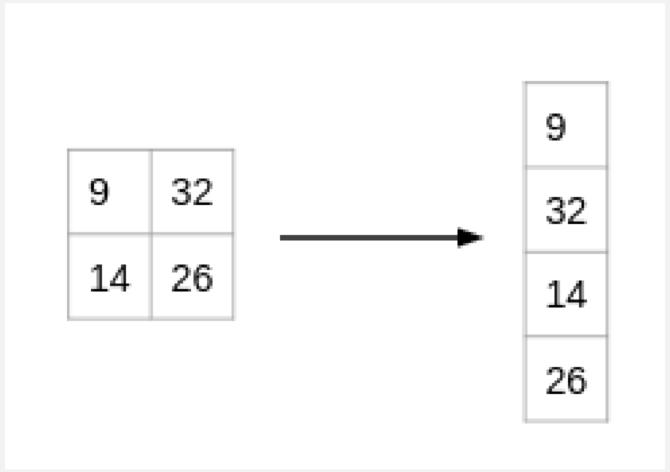
- To find the shape of an output for more complex inputs or filter dimensions, the formula goes as:



```
Dimension of image = (n, n)
Dimension of filter = (f,f)

Dimension of output will be ((n-f+1) , (n-f+1))
```

FULLY CONNECTED(DENSE) LAYER

- The output from the convolution layer was a 2D matrix.
- Ideally, we would want each row to represent a single input image. In fact, the fully connected layer can only work with 1D data. Hence, the values generated from the previous operation are first converted into a 1D format.



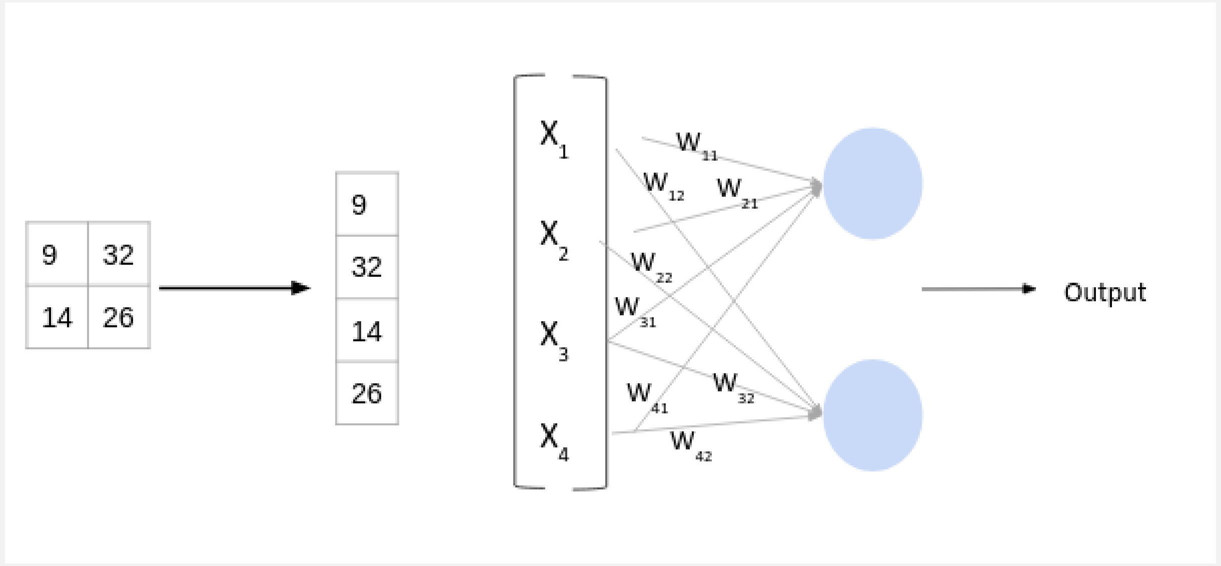
- Once the data is converted into a 1D array, it is sent to the fully connected layer. All of these individual values are treated as separate features that represent the image.
- The fully connected layer performs two operations on the incoming data – **a linear transformation and a non-linear transformation.**

1. LINEAR TRANSFORMATION

- The equation for linear transformation is:

$$Z = W^T.X + b$$

- Here, X is the input, W is weight, and b (called bias) is a constant.



$$Z = W^T.X + b$$

$$Z = \begin{bmatrix} W_{11} & W_{21} & W_{31} & W_{41} \\ W_{12} & W_{22} & W_{32} & W_{42} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$Z_{2x2} = \begin{bmatrix} W_{11}X_1 + W_{21}X_2 + W_{31}X_3 + W_{41}X_4 \\ W_{12}X_1 + W_{22}X_2 + W_{32}X_3 + W_{42}X_4 \end{bmatrix}$$

Considering the size of the matrix is (m, n) – m will be equal to the number of features or inputs for this layer. Since we have 4 features from the convolution layer, m here would be 4.

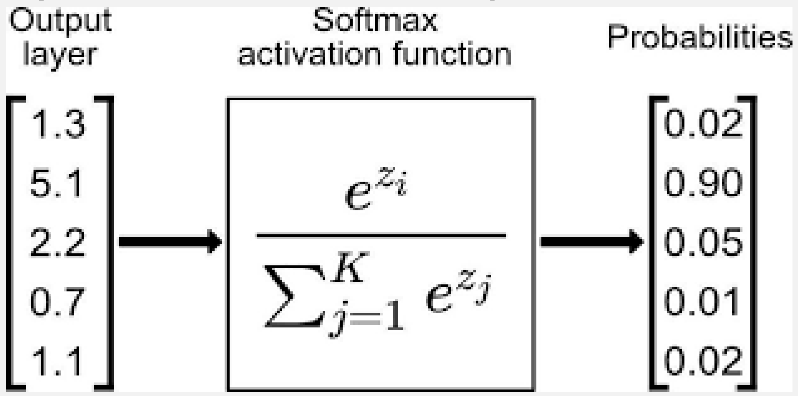
The value of n will depend on the number of neurons in the layer. For instance, if we have two neurons, then the shape of weight matrix will be (4, 2)

2. NON - LINEAR TRANSFORMATION

- The linear transformation alone cannot capture complex relationships. Thus, we introduce an additional component in the network which adds non-linearity to the data.
- This new component in the architecture is called the "**Activation Function**".
- The one we are working with in our model is:

SOFTMAX ACTIVATION FUNCTION:

- It is especially used for multiclass classification problems.
- It is applied to the output layer of a neural network, and its purpose is to transform the raw output values into a probability distribution over the predicted classes.

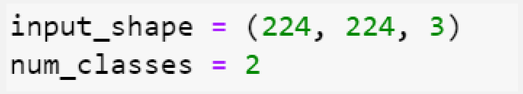

Output layer
1.3
5.1
2.2
0.7
1.1
Softmax activation function
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
Probabilities
0.02
0.90
0.05
0.01
0.02

$\vec{z}$	The input vector to the softmax function, made up of (z0, ... zK)
z_i	All the zi values are the elements of the input vector to the softmax function, and they can take any real value, positive, zero or negative. For example a neural network could have output a vector such as (-0.62, 8.12, 2.53), which is not a valid probability distribution, hence why the softmax would be necessary.
e^{z_i}	The standard exponential function is applied to each element of the input vector. This gives a positive value above 0, which will be very small if the input was negative, and very large if the input was large. However, it is still not fixed in the range (0, 1) which is what is required of a probability.
$\sum_{j=1}^{K} e^{z_j}$	The term on the bottom of the formula is the normalization term. It ensures that all the output values of the function will sum to 1 and each be in the range (0, 1), thus constituting a valid probability distribution.
K	The number of classes in the multi-class classifier.

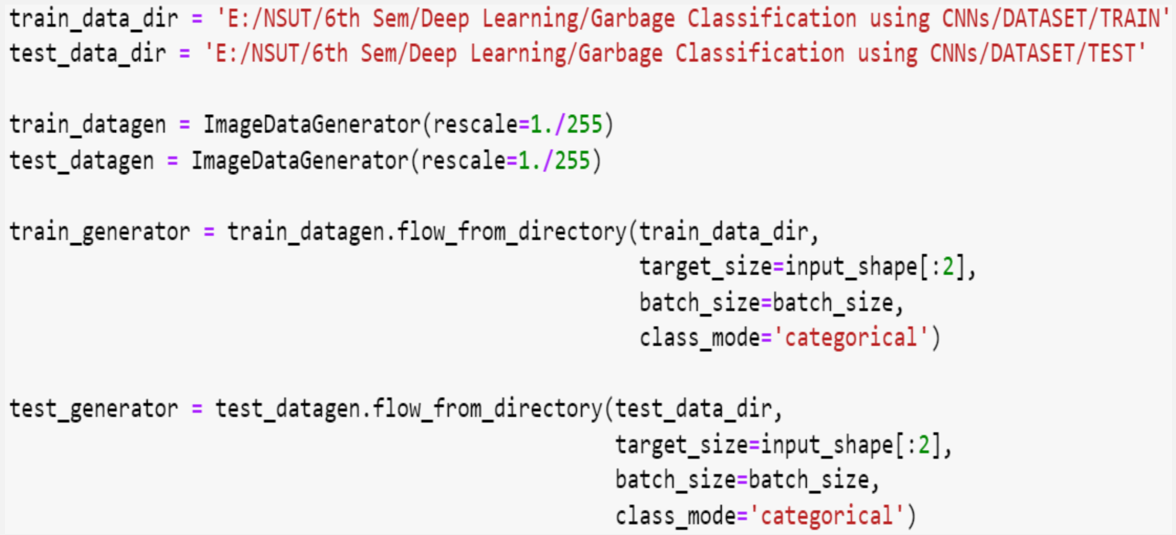
ABOUT PROJECT CODE

DATA PREPROCESSING

- Specify input image size and number of classes in the input data.
- Creating Data Generators:
  - Set rescale parameter to 1./255 to normalize pixel values to be between 0 and 1.
  - Resize images to dimensions specified in target_size.
  - Convert images to batches of data with number of images equal to batch_size parameter.
  - Set class_mode parameter to 'categorical' to represent labels in one-hot encoded format.



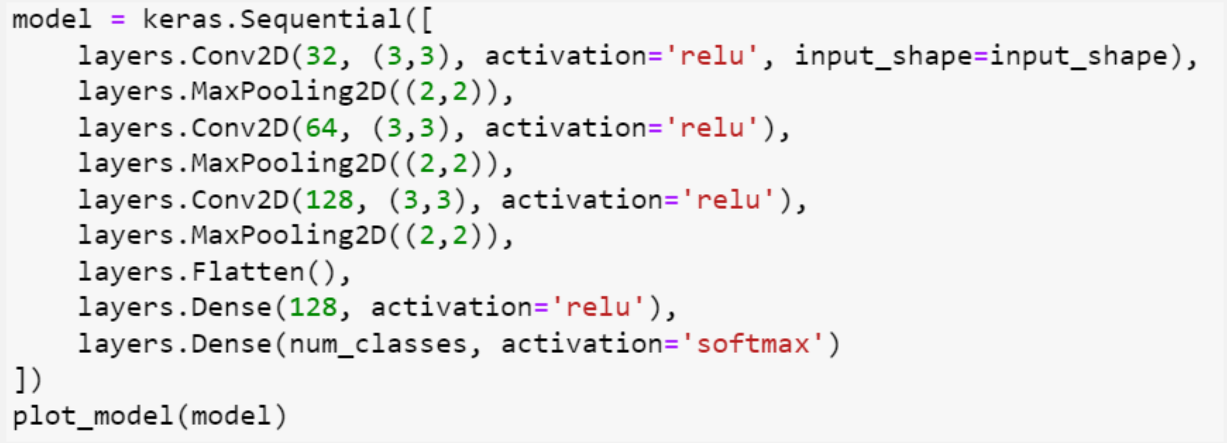
```
input_shape = (224, 224, 3)
num_classes = 2
```



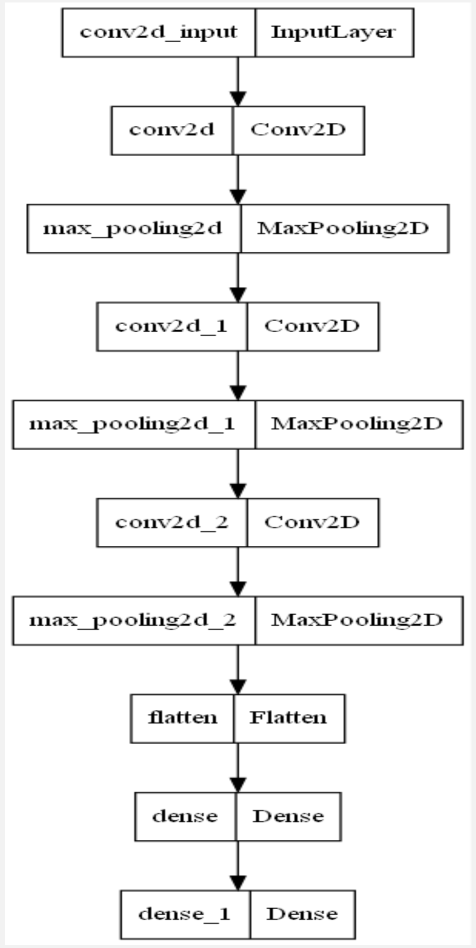
```
train_data_dir = 'E:/NSUT/6th Sem/Deep Learning/Garbage Classification using CNNs/DATASET/TRAIN'
test_data_dir = 'E:/NSUT/6th Sem/Deep Learning/Garbage Classification using CNNs/DATASET/TEST'

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_data_dir,
                                                     target_size=input_shape[:2],
                                                     batch_size=batch_size,
                                                     class_mode='categorical')

test_generator = test_datagen.flow_from_directory(test_data_dir,
                                                   target_size=input_shape[:2],
                                                   batch_size=batch_size,
                                                   class_mode='categorical')
```

MODEL ARCHITECTURE

- Conv2D layer creates a convolutional layer with 32, 64, and 128 filters of size 3x3.
- Activation parameter is set to 'relu' for applying the rectified linear unit activation function to each neuron output.
- MaxPooling2D layer creates a pooling layer that downsamples convolutional layer output by taking the maximum value in each 2x2 region.
- Flatten layer flattens the output of convolutional layers into a 1D vector, which is passed to fully connected layers.
- Dense layers create fully connected layers with 128 and num_classes neurons.
- Activation parameter is set to 'relu' for the first dense layer and 'softmax' for the second dense layer.
- 'Softmax' activation function outputs a probability distribution over possible classes used for multi-class classification.

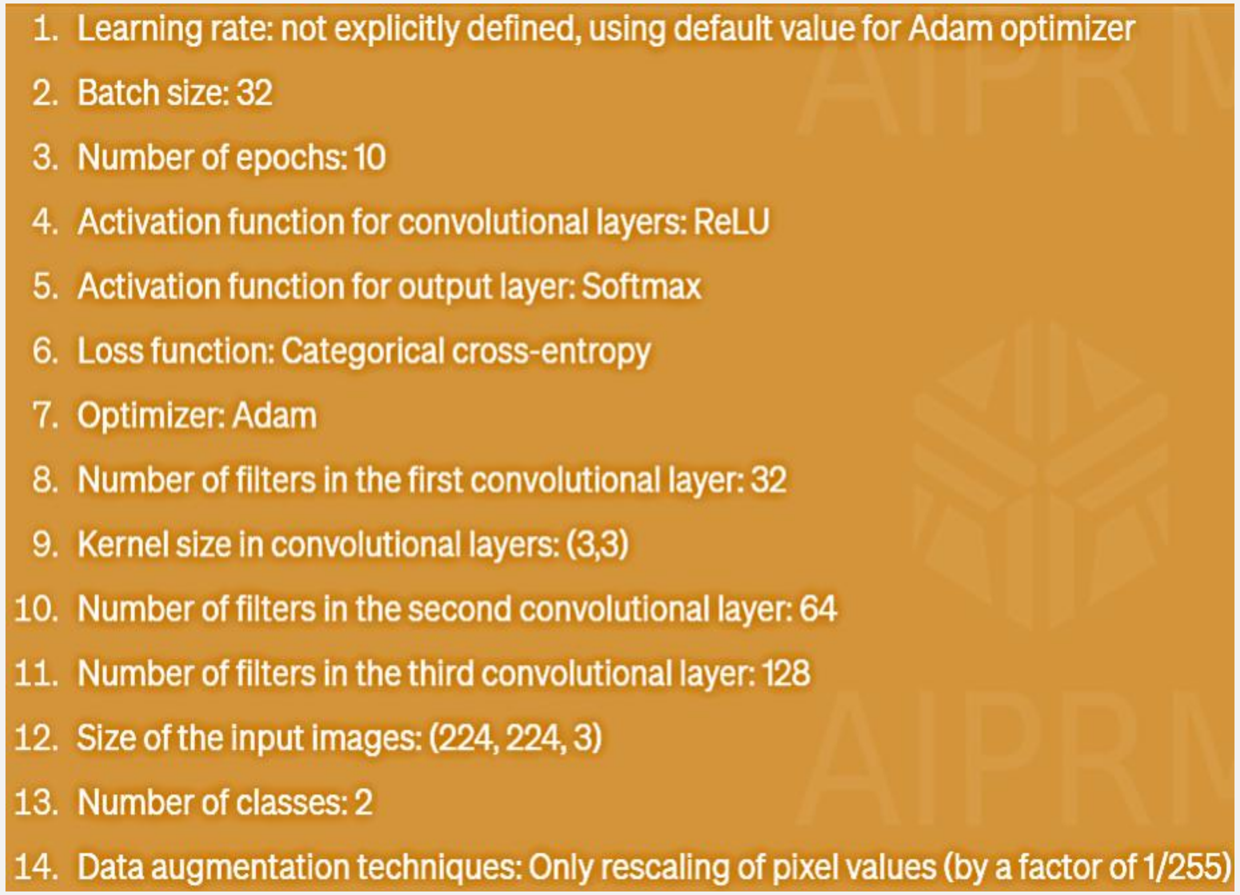


```
model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])
plot_model(model)
```


conv2d_input | InputLayer
conv2d | Conv2D
max_pooling2d | MaxPooling2D
conv2d_1 | Conv2D
max_pooling2d_1 | MaxPooling2D
conv2d_2 | Conv2D
max_pooling2d_2 | MaxPooling2D
flatten | Flatten
dense | Dense
dense_1 | Dense

HYPERPARAMETERS CHOSEN

- We chose the following hyperparameters;
  - input_shape:The input image size to the CNN model.
  - num_classes:The number of output classes.
  - batch_size: The number of samples that will be propagated through the network in each iteration during training.
  - epochs: The number of times the training dataset is iterated over while training the model.
  - optimizer: The algorithm used to optimize the weights of the model during training.
  - loss: The loss function used to measure the difference between predicted and actual output values during training.
  - metrics: The performance metric used to evaluate the performance of the model during training and testing.
  - learning_rate: The rate at which the model's weights are updated during training.
  - kernel_size: The size of the convolutional kernel used in the convolutional layers.
  - activation: The activation function used in the layers of the CNN model.



1. Learning rate: not explicitly defined, using default value for Adam optimizer
2. Batch size: 32
3. Number of epochs: 10
4. Activation function for convolutional layers: ReLU
5. Activation function for output layer: Softmax
6. Loss function: Categorical cross-entropy
7. Optimizer: Adam
8. Number of filters in the first convolutional layer: 32
9. Kernel size in convolutional layers: (3,3)
10. Number of filters in the second convolutional layer: 64
11. Number of filters in the third convolutional layer: 128
12. Size of the input images: (224, 224, 3)
13. Number of classes: 2
14. Data augmentation techniques: Only rescaling of pixel values (by a factor of 1/255)

HYPERPARAMETER OPTIMIZATION

1.

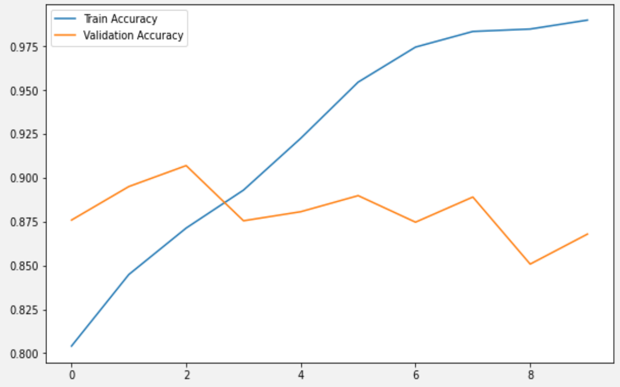
NO. OF CONVOLUTION LAYERS: **3**

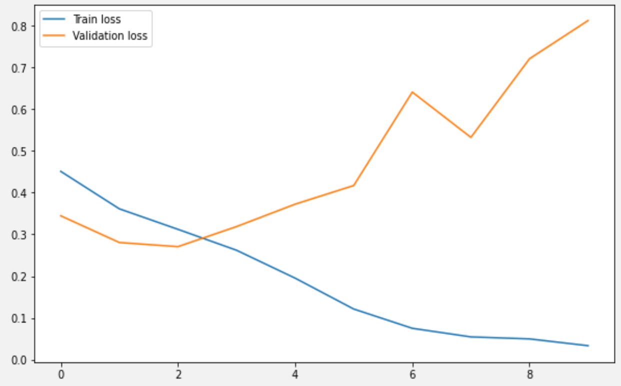
BATCH SIZE: **32**

NO. OF EPOCHS: **10**

OPTIMIZER USED: **ADAM**

LOSS: 0.8447 - ACCURACY: 0.8846





2.

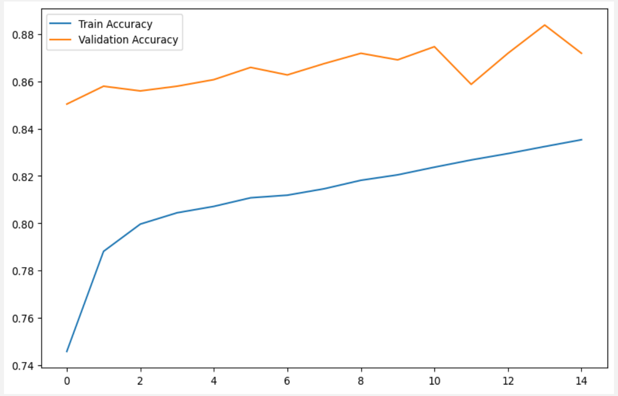
NO. OF CONVOLUTION LAYERS: **4**

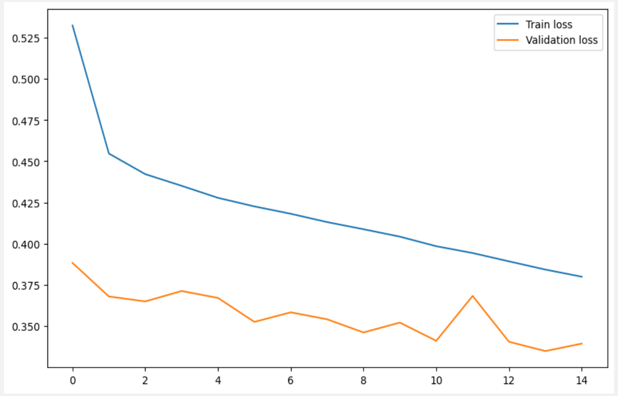
BATCH SIZE: **64**

NO. OF EPOCHS: **15**

OPTIMIZER USED: **ADAGRAD**

LOSS: 0.3395 - ACCURACY: 0.8719





3.

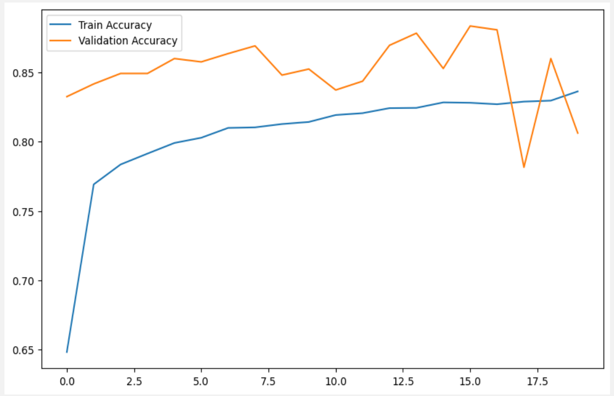
NO. OF CONVOLUTION LAYERS: **6**

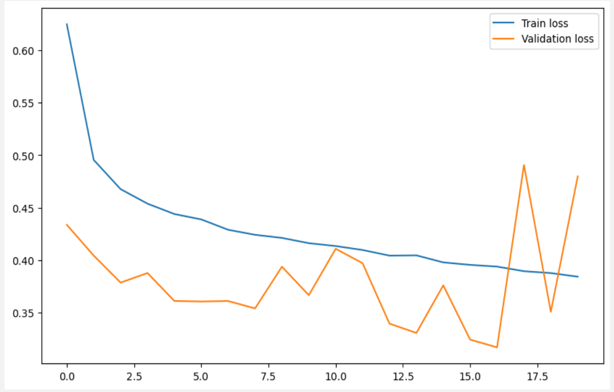
BATCH SIZE: **128**

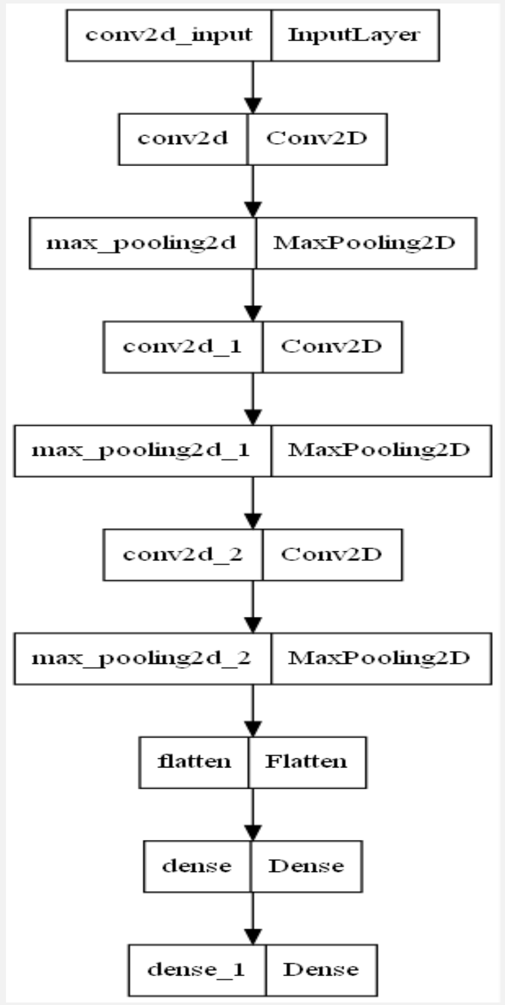
NO. OF EPOCHS: **20**

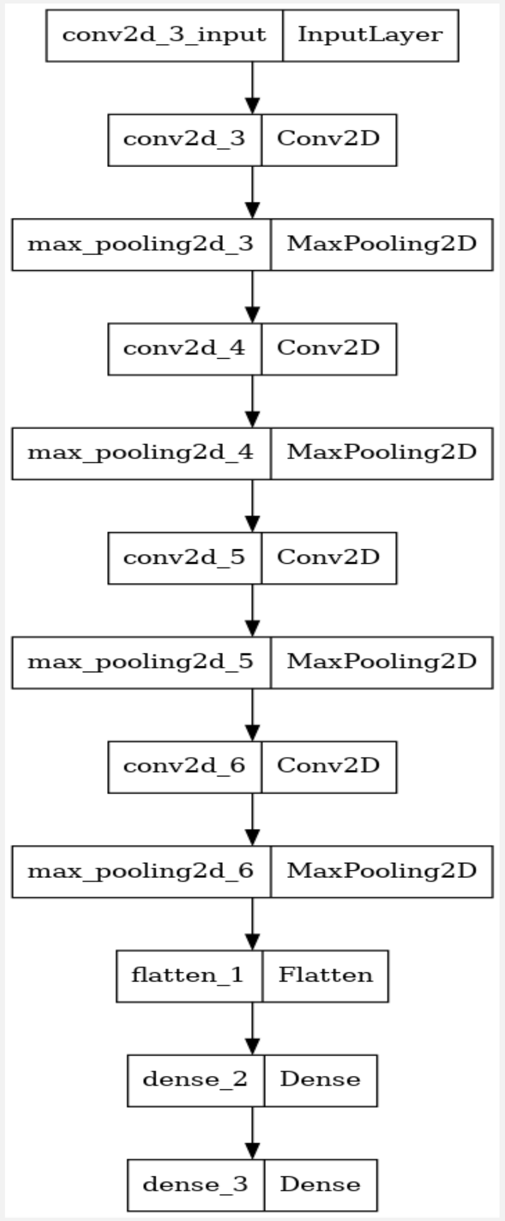
OPTIMIZER USED: **SGD**

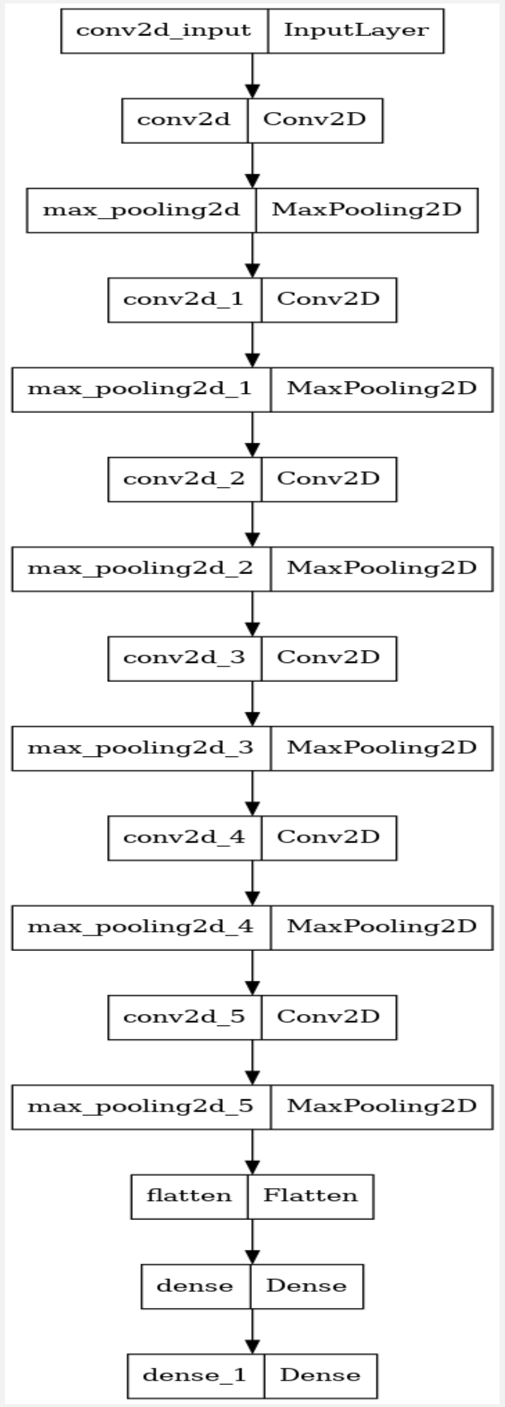
LOSS: 0.4797 - ACCURACY: 0.8062



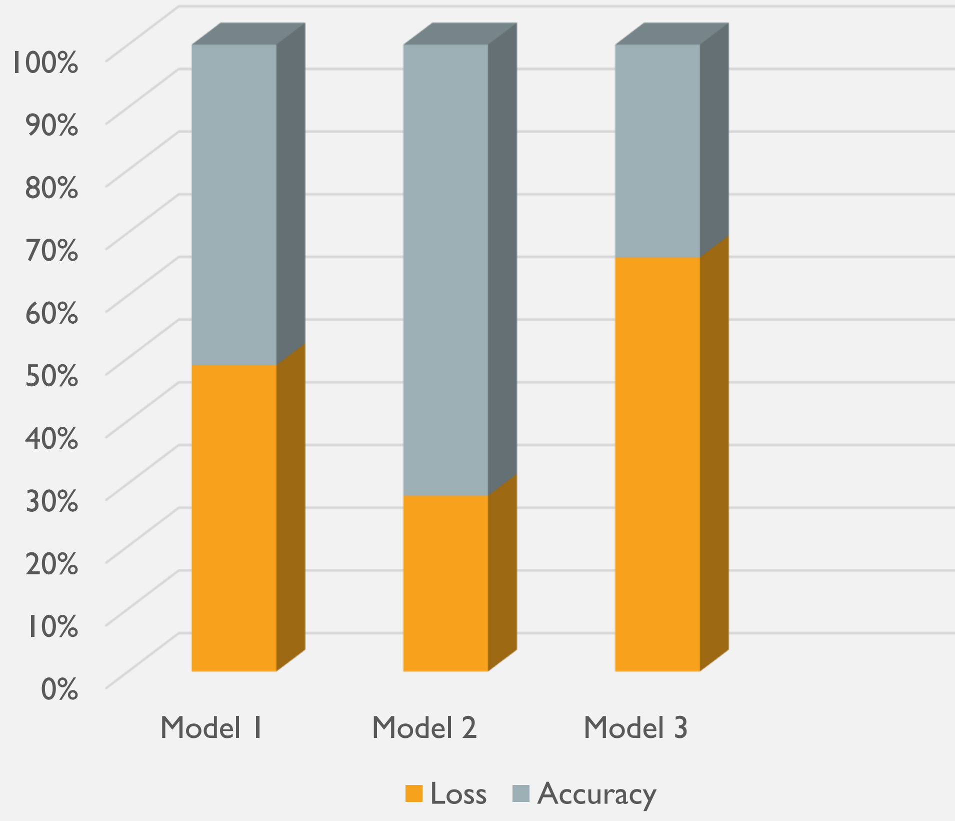



conv2d_input | InputLayer
conv2d | Conv2D
max_pooling2d | MaxPooling2D
conv2d_1 | Conv2D
max_pooling2d_1 | MaxPooling2D
conv2d_2 | Conv2D
max_pooling2d_2 | MaxPooling2D
flatten | Flatten
dense | Dense
dense_1 | Dense


conv2d_3_input | InputLayer
conv2d_3 | Conv2D
max_pooling2d_3 | MaxPooling2D
conv2d_4 | Conv2D
max_pooling2d_4 | MaxPooling2D
conv2d_5 | Conv2D
max_pooling2d_5 | MaxPooling2D
conv2d_6 | Conv2D
max_pooling2d_6 | MaxPooling2D
flatten_1 | Flatten
dense_2 | Dense
dense_3 | Dense


conv2d_input | InputLayer
conv2d | Conv2D
max_pooling2d | MaxPooling2D
conv2d_1 | Conv2D
max_pooling2d_1 | MaxPooling2D
conv2d_2 | Conv2D
max_pooling2d_2 | MaxPooling2D
conv2d_3 | Conv2D
max_pooling2d_3 | MaxPooling2D
conv2d_4 | Conv2D
max_pooling2d_4 | MaxPooling2D
conv2d_5 | Conv2D
max_pooling2d_5 | MaxPooling2D
flatten | Flatten
dense | Dense
dense_1 | Dense

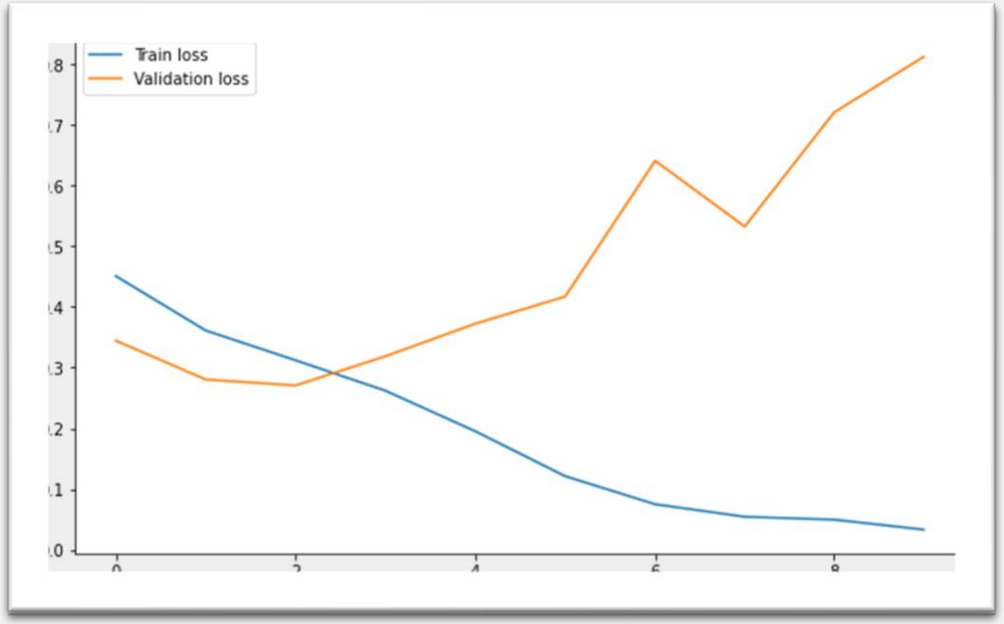
MODEL PERFORMANCE COMPARISON

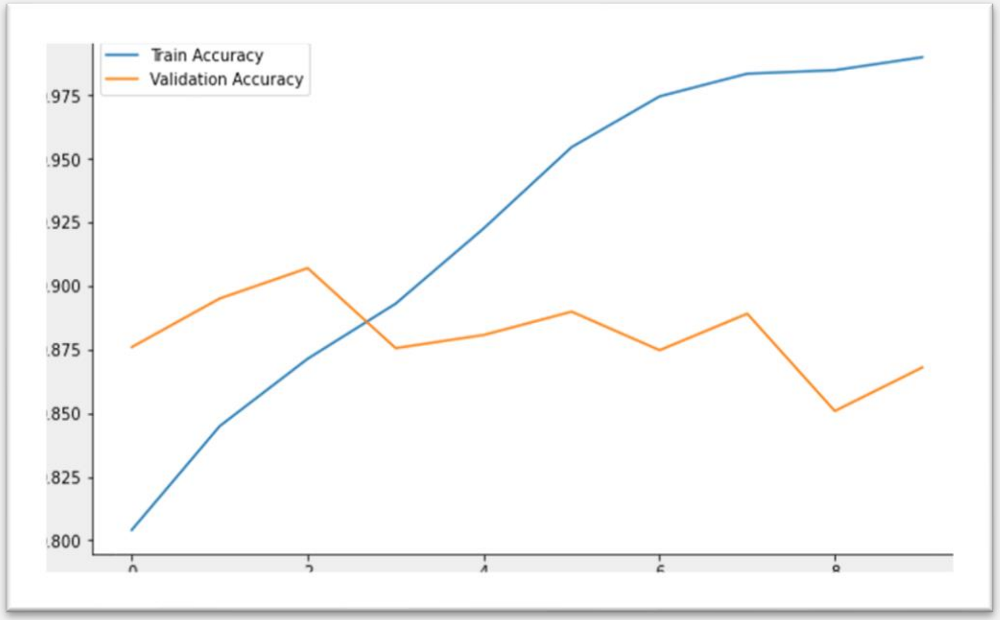


MODEL PERFORMANCE

- The model achieved 89% validation accuracy after 10 epochs, with decreasing training and validation loss indicating improvement.

- The test accuracy was around 90%. Nonetheless, the model's overall performance may vary based on the dataset and problem being tackled.





RESULTS & CONCLUSION

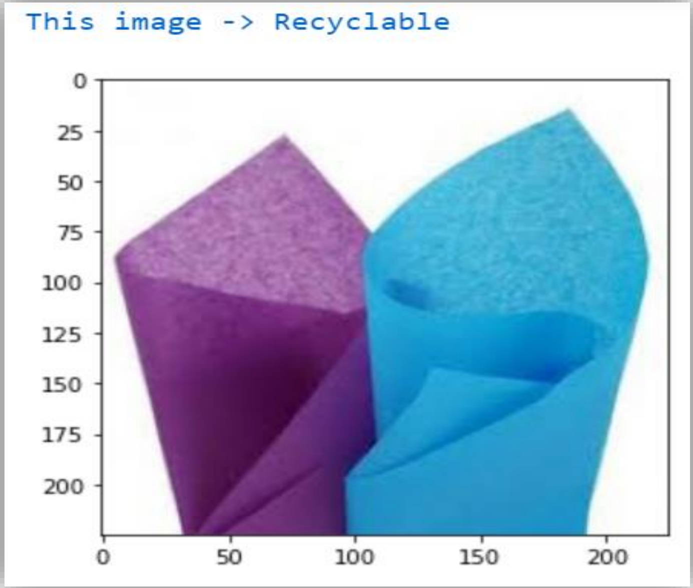
- Successfully able to classify input image of garbage into: Organic or Recyclable, with test accuracy up to 89%.

- Scope of improvements;
  - Increasing the dataset size
  - Fine-tuning hyperparameters
  - Adding more convolutional layers
  - Experimenting with different optimizers and learning rates
  - Use pre-trained architectures such as ResNet, DenseNet or AlexNet etc.

```
This image -> Organic
```

```
test_loss, test_acc = model.evaluate(test_generator, steps=len(test_generator))
print('Test accuracy:', test_acc)
```

```
79/79 [==============================] - 31s 388ms/step - loss: 0.8447 - accuracy: 0.8846
Test accuracy: 0.8846001029014587
```



```
This image -> Recyclable
```

REFERENCES

KAGGLE.COM
DATASET

TOWADSDATASCIENCE.COM

DATASEARCH.
RESEARCH.GOOGLE.
COM

ANALYTICS VIDHYA
WEB LINK