University of Mumbai

Garbage Waste Segregation

using Low Light Image

Enhancement

Submitted at the end of semester VI in partial fulfillment of requirements For the degree of

Bachelors in Technology

by

Kashish Shah

1913025

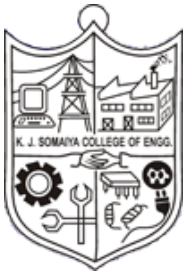
Jayesh Nirve

1913033

Guide

Prof Mahesh Warang

Prof Maruti Zalte



Department of Electronics and Telecommunication Engineering

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch 2019 -2023

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate

This is to certify that this report entitled **Garbage Waste Segregation using Low Light Image Enhancement** submitted by Kashish Shah and Jayesh Nirve at the end of semester VI of TY B. Tech is a bonafide record for partial fulfillment of requirements for the degree of Bachelors in Technology in Electronics and Telecommunication Engineering of University of Mumbai

Guide/Examiner1 Examiner2

Date:

Place: Mumbai-77

Department of Electronics and Telecommunication Engineering Semester VI 2019-23 Batch

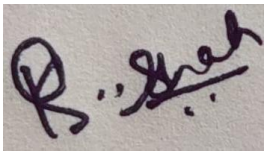
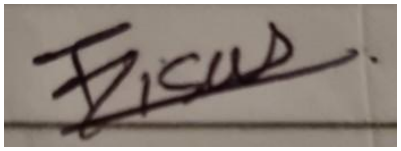
K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

DECLARATION

We declare that this written report submission represents the work done based on our and/or others' ideas with adequately cited and referenced the original source. We also declare that we have adhered to all principles of intellectual property, academic honesty, and integrity as we have not misinterpreted or fabricated, or falsified any idea/data/fact/source/original work/ matter in my submission.

We understand that any violation of the above will be cause for disciplinary action by the college and may evoke the penal action from the sources which have not been properly cited or from whom proper permission is not sought.

	
Signature of the Student	**Signature of the Student**
1913025	**1913033**
Roll No.	**Roll No.**

Date: 28/04/2022

Place: Mumbai-77

Abstract

Waste segregation refers to the separation of dry and wet garbage, which paves the way for other concepts of waste management like composting, recycling, and incineration. Its end goal is to reduce waste from landfills and eventually, prevent land, water, and air pollution.

India generates 62 million tonnes (MT) of waste every year, and only 43 MT is collected. Of the collected waste, close to 31 MT is dumped on landfill sites or water bodies, and only11.9 MT is scientifically treated. The world generates 2.01 billion tonnes of municipal solid waste annually, with at least 33 percent of that—extremely conservatively—not managed in an environmentally safe manner.

Segregating enormous amounts of garbage into these two categories is a difficult task. Often in India, human collectors are employed to do this task manually in unhygienic conditions which makes them susceptible to several diseases. Chemical and medical waste is hazardous and cannot be disposed of by hand. The proposed approach attempts to solve these problems by developing an automated garbage segregation model. For this, we take the input image and pass it through a convolutional neural network. Initially, it goes through a low light image enhancement algorithm that enhances the image quality. An image representation is obtained from the extracted deep features. This is then classified using various supervised learning techniques into 6 separate classes namely: (a) plastic; (b) metal; (c) cardboard; (d) paper; (e) glass; and (f) general trash.

Contents

List of Figures

List of Tables

1. Introduction

The world's increasing waste level poses a serious threat to biodiversity. Garbage can be broadly classified as biodegradable and non-biodegradable waste. Biodegradable waste decomposes by natural factors without polluting the environment. Non-biodegradable waste does not decompose naturally which causes it to accumulate and pollute the environment. Biodegradable waste can be used for fertilizer manufacturing and energy generation. Non-biodegradable waste is hazardous but it can be recycled and reused.

However, to take the benefits of this waste, we need to segregate them before use. Manual segregation of garbage is an unhygienic task. Those involved in manual segregation come from economically backward areas. They often with social discrimination and ostracization due to the nature of their job. Sometimes medical and chemical waste is dumped along with other garbage. This can be life-threatening to those involved in segregation since it is hazardous to handle such waste with bare hands. The main advantage of our proposed approach is that when coupled with some automated image capturing technique, it will minimize the human effort required to manually segregate garbage. This will enable the workers to live in dignity and reduce the dangers associated with direct access to this waste. In recent times, machine learning is becoming increasingly used to create environmental sustainability. Deep learning is a branch of machine learning that mimics the brain using artificial neural networks and uses the knowledge gained to make decisions. In the proposed approach, we have used deep learning to generate a representation of the image. For this, we have considered a Convolutional Neural Network (CNN) to extract features from the image.
Then, classification is done using several supervised machine learning approaches. This determines the object in the image. Using this information, we segregate the waste in the image as biodegradable and non-biodegradable. The rest of the paper is organized as follows: Section 2 provides a survey on several related works done in this field. The proposed methodology is described in Section 3. Section 4 gives the details of the dataset and the experimental results obtained while evaluating the proposed approach. Finally, Section 5 concludes the paper and provides a brief outline of the possible future work.

2. Literature Review

Most of the proposed approaches for garbage segregation use deep learning for classification. A transfer learning-based approach took a VGG 19 CNN pre-trained on Image Net and fine-tuned it to classify garbage[7]. Several other CNN architectures such as VGG, Inception, and ResNet were also used for the automatic classification of waste. In, the author focused on classifying the images into different categories using InceptionV3[6] architecture to train the model and deploy it on a web server developed using Flask Web Framework. Some workers also made modifications to the existing CNNs to optimize their performance. [4]Bobulski and Kubanek used a 16-layer CNN with an auto-encoder to classify the images. In, the author experimented with deep CNN architectures by reducing the number of parameters. Some used faster R-CNN and YOLO. The authors broadened the existing CNN to enhance the classification performance. A self-learning neural network is used to categorize the images based on their properties. A pre-trained CNN was combined with a network header to classify images taken from a mobile[9]. During our survey, we found out that many also incorporated hardware in their proposed approach. In, the authors proposed a garbage bin with a camera that took images of the garbage and classified them using an optimized ResNet-34 algorithm[5]. Anh H et. al developed a deep neural network model named DNN-TC for trash classification on the VN-trash dataset and TrashNet which fine-tuned the ResNext CNN. The authors proposed a waste classification model named WasteNet based on a CNN that can be deployed on a low power device such as a Jetson Nano. A combination of sound recording and a beat frequency oscillator metal detector was given as input to a machine-learning algorithm to identify the occurrence of glass and metal in trash bags with a test rig built to record sound and metal detector data from trash bags. In, the authors proposed to segregate hazardous waste using deep learning networks in real-time videos and used machine learning to categorize different recyclable materials. Ahmad et.[10] all extracted features from the fc1 layer of several CNNs which were pre-trained on ImageNet and combined their features using different methods such as genetic algorithms, particle swarm optimization, discriminant correlation analysis, and induced ordered weighting average[8]. In our proposed approach, we have extracted features from CNN and classified them using several supervised learning techniques.

3. Methodology

3.1 Introduction

This section presents the methods used for feature extraction and classification of garbage images. As seen in Fig 1., deep features are extracted from a CNN and classified

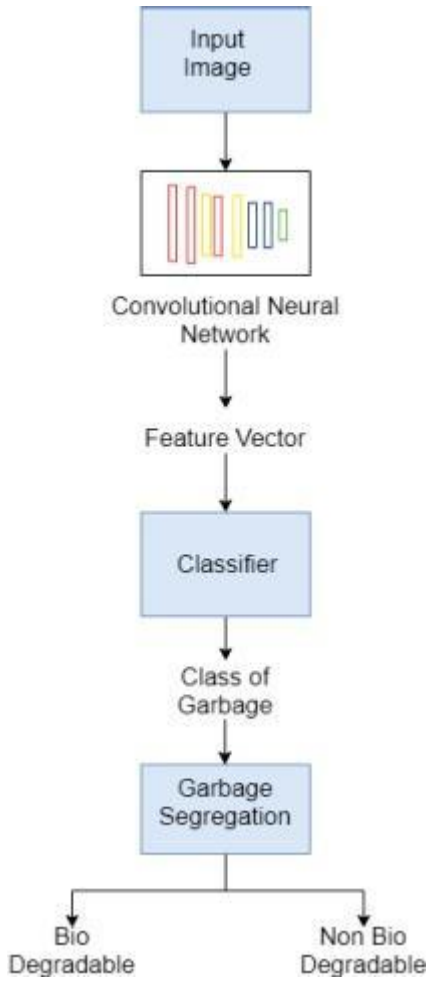


Fig 1. Flowchart of selected approach

3.2 Low Light Image Enhancement - MIRNet Model

To recover high-quality image content from its degraded version, image restoration enjoys numerous applications, such as in photography, security, medical imaging, and remote sensing [14]. In this example, we implement the MIRNet model for low-light image enhancement, a fully-convolutional architecture that learns an enriched set of features that combines contextual information from multiple scales, while simultaneously preserving the high-resolution spatial details.

We use 300 image pairs from the LoL Dataset's training set for training, and we use the remaining 185 image pairs for validation. We generate random crops of size 128 x 128 from the image pairs to be used for both training and validation[12].

Here are the main features of the MIRNet model:

- A feature extraction model that computes a complementary set of features across multiple spatial scales, while maintaining the original high-resolution features to preserve precise spatial details.
- A regularly repeated mechanism for information exchange, where the features across multi-resolution branches are progressively fused for improved representation learning.
- A new approach to fuse multi-scale features using a selective kernel network that dynamically combines variable receptive fields and faithfully preserves the original feature information at each spatial resolution.
- A recursive residual design that progressively breaks down the input signal to simplify the overall learning process, and allows the construction of very deep networks.

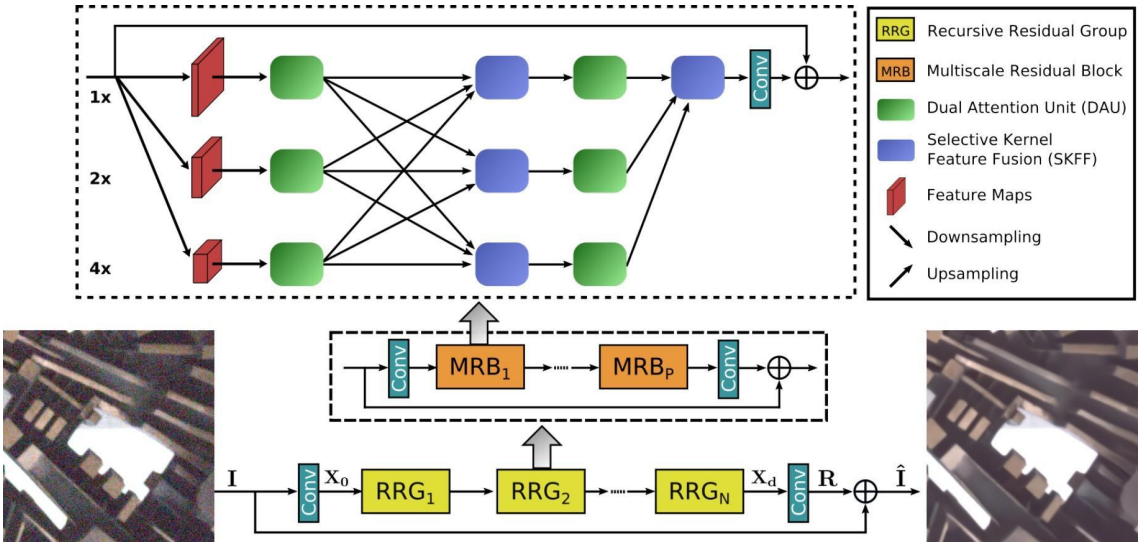


Fig 2. MIRNet Model

3.2.1 Selective Kernel Feature Fusion

The Selective Kernel Feature Fusion or SKFF module performs dynamic adjustment of receptive fields via two operations: **Fuse** and **Select**. The Fuse operator generates global feature descriptors

by combining the information from multi-resolution streams. The Select operator uses these descriptors to recalibrate the feature maps (of different streams) followed by their aggregation.

Fuse: The SKFF[13] receives inputs from three parallel convolution streams carrying different scales of information. We first combine these multi-scale features using an element-wise sum, on which we apply Global Average Pooling (GAP) across the spatial dimension. Next, we apply a channel-downscaling convolution layer to generate a compact feature representation that passes through three parallel channel-upscaling convolution layers (one for each resolution stream) and provides us with three feature descriptors.

Select: This operator applies the SoftMax function to the feature descriptors to obtain the corresponding activations that are used to adaptively recalibrate multi-scale feature maps. The aggregated features are defined as the sum of the product of the corresponding multi-scale feature and the feature descriptor.

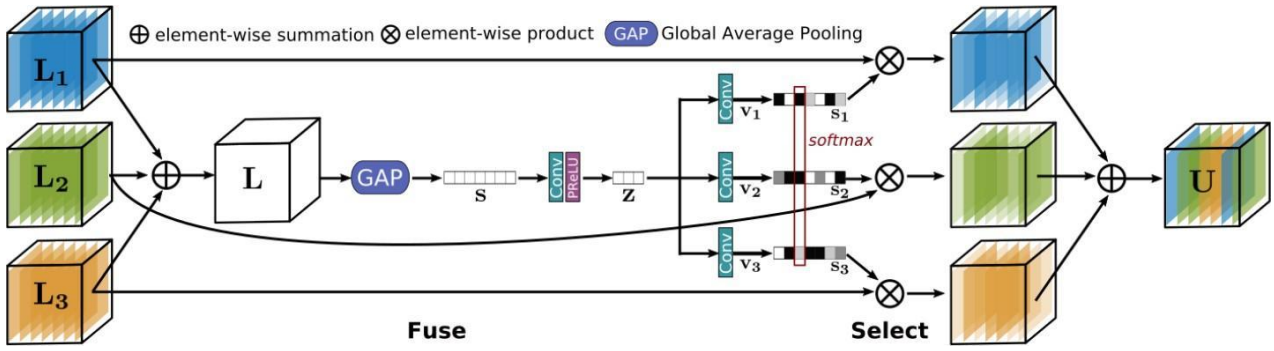


Fig 3. Selective Kernel Feature Fusion

3.2.2 Dual Attention Unit

The Dual Attention Unit or DAU is used to extract features in the convolutional streams. While the SKFF block fuses information across multi-resolution branches, we also need a mechanism to share information within a feature tensor, both along the spatial and the channel dimensions which is done by the DAU block. The DAU suppresses less useful features and only allows more informative ones to pass further. This feature recalibration is achieved by using **Channel Attention** and **Spatial Attention** mechanisms.

The **Channel Attention** branch exploits the inter-channel relationships of the convolutional feature maps by applying squeeze and excitation operations. Given a feature map, the squeeze operation applies Global Average Pooling across spatial dimensions to encode global context, thus yielding a feature descriptor. The excitation operator passes this feature descriptor through two convolutional layers followed by the sigmoid gating and generates activations. Finally, the output of the Channel Attention branch is obtained by rescaling the input feature map with the output activations.

The **Spatial Attention** branch is designed to exploit the inter-spatial dependencies of convolutional features. The goal of Spatial Attention is to generate a spatial attention map and use it to recalibrate the incoming features. To generate the spatial attention map, the Spatial Attention branch first independently applies Global Average Pooling and Max Pooling operations on input features along the channel dimensions and concatenates the outputs to form a resultant feature map which is then passed through convolution and sigmoid activation to obtain the spatial attention map. This spatial attention map is then used to rescale the input feature map.

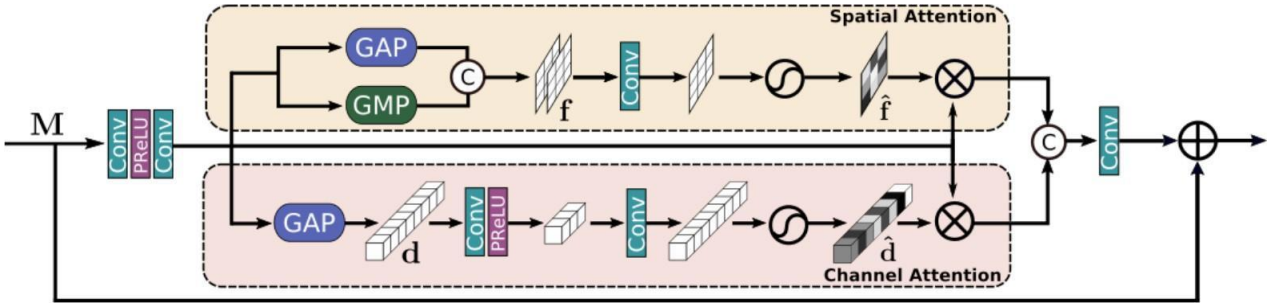
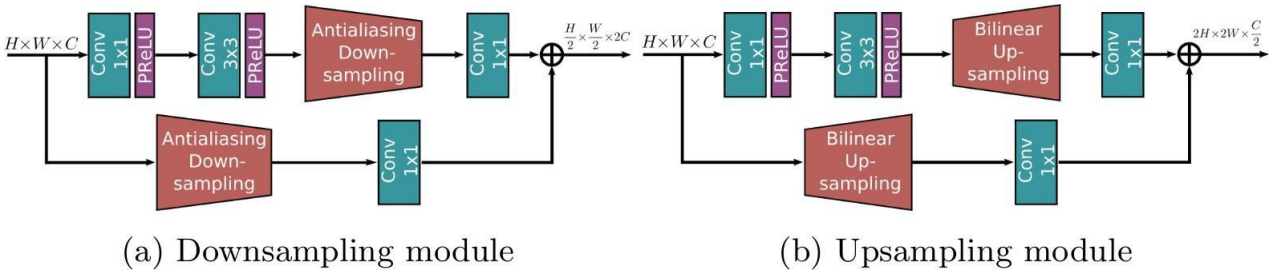


Fig 4. Dual Attention Unit

3.2.3 Multi-Scale Residual Block

The Multi-Scale Residual Block is capable of generating a spatially-precise output by maintaining high-resolution representations while receiving rich contextual information from low resolutions. The MRB consists of multiple (three in this paper) fully-convolutional streams connected in parallel. It allows information exchange across parallel streams to consolidate the high-resolution features with the help of low-resolution features, and vice versa. The MIRNet employs a recursive residual design (with skip connections) to ease the flow of information during the learning process. To maintain the residual nature of our architecture, residual resizing modules are used to perform downsampling and upsampling operations that are used in the Multi-scale Residual Block.



(a) Downsampling module (b) Upsampling module

Fig 5. Downsampling and Upsampling

3.2.4 Web Application

Both MIRNet and TrashNet are hosted on a separate port on the same server. Both have an individual web UI and can also be used as individual modules.

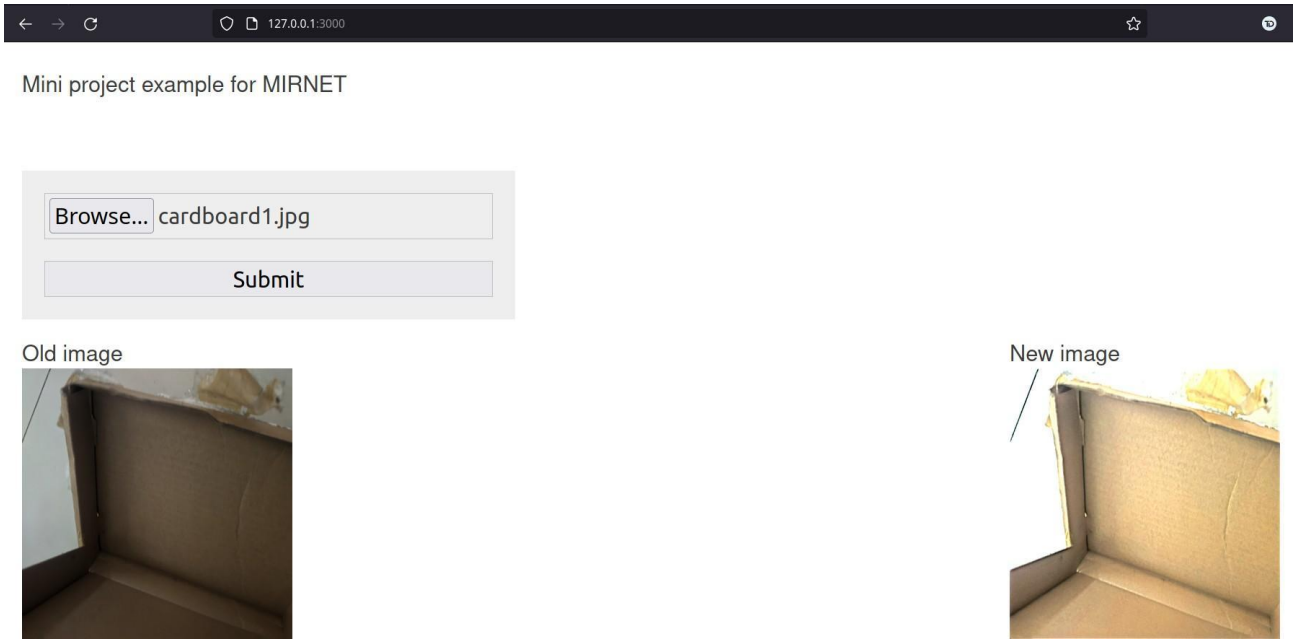


Fig 6. Web Interface of MIRNet Model

The image is uploaded to the server via this GUI which runs on port 3000. Apart from using the model in a web GUI, we can also only load a cli only version where once the image is uploaded the results are shown in the cli and can also be passed to the TrashNet model. We also have a

runonDirectory function for testing purposes, where we can run the model on a bunch of images in a directory and get how accurate the model is.

3.3 Garbage Classification - TrashNet Model

This repository contains the dataset that we collected. The dataset spans six classes: glass, paper, cardboard, plastic, metal, and trash. Currently, the dataset consists of 2527 images:

- 501 glass
- 594 paper
- 403 cardboard
- 482 plastic
- 410 metal
- 137 trash

The pictures were taken by placing the object on a white poster board and using sunlight and/or room lighting. The pictures have been resized down to 512 x 384, which can be changed in data/constants.py (resizing them involves going through step 1 in usage).

The convolutional neural network results on the poster are dated since we continued working after the end of the quarter and were able to achieve around 75% test accuracy (with 70/13/17 train/val/test split) after changing the weight initialization.

Fig 7. Classes: a) Cardboard b) Glass c) Metal d) Paper e) Plastic f) General Trash

3.3.1 Image Augmentation

Data Augmentation is a technique that can be used to artificially expand the size of a training set by creating modified data from the existing one. It is a good practice to use DA if you want to prevent overfitting if the initial dataset is too small to train on, or even if you want to squeeze better performance from your model. Data Augmentation is also good for enhancing the model's performance. You can augment:

- Audio
- Text
- Images
- Any other types of data

We can apply various changes to the initial data. For example, for images we can use:

- Geometric transformations – you can randomly flip, crop, rotate or translate images, and that is just the tip of the iceberg

- Color space transformations – change RGB color channels, intensify any color
- Kernel filters – sharpen or blur an image
- Random Erasing – delete a part of the initial image
- Mixing images mix images. Might be counterintuitive but it works

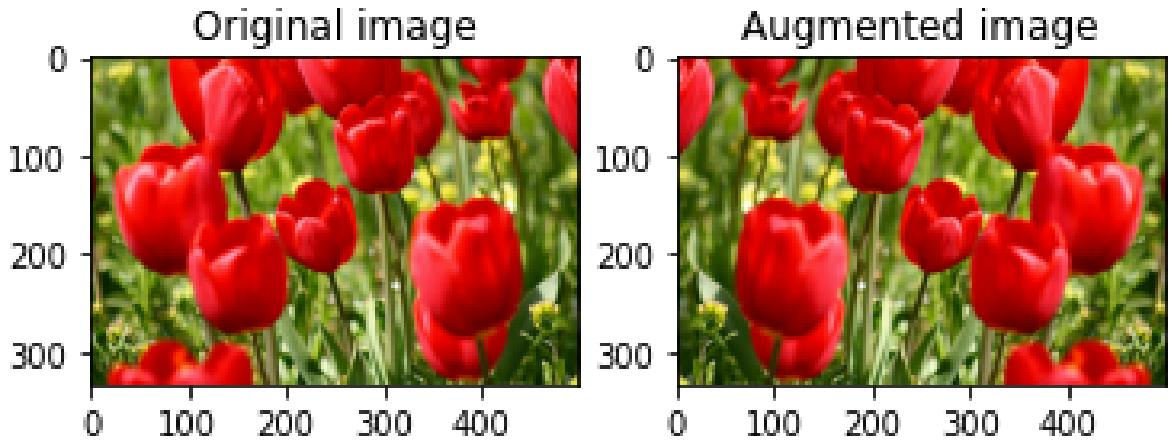


Fig 8. Image being flipped from left to right.

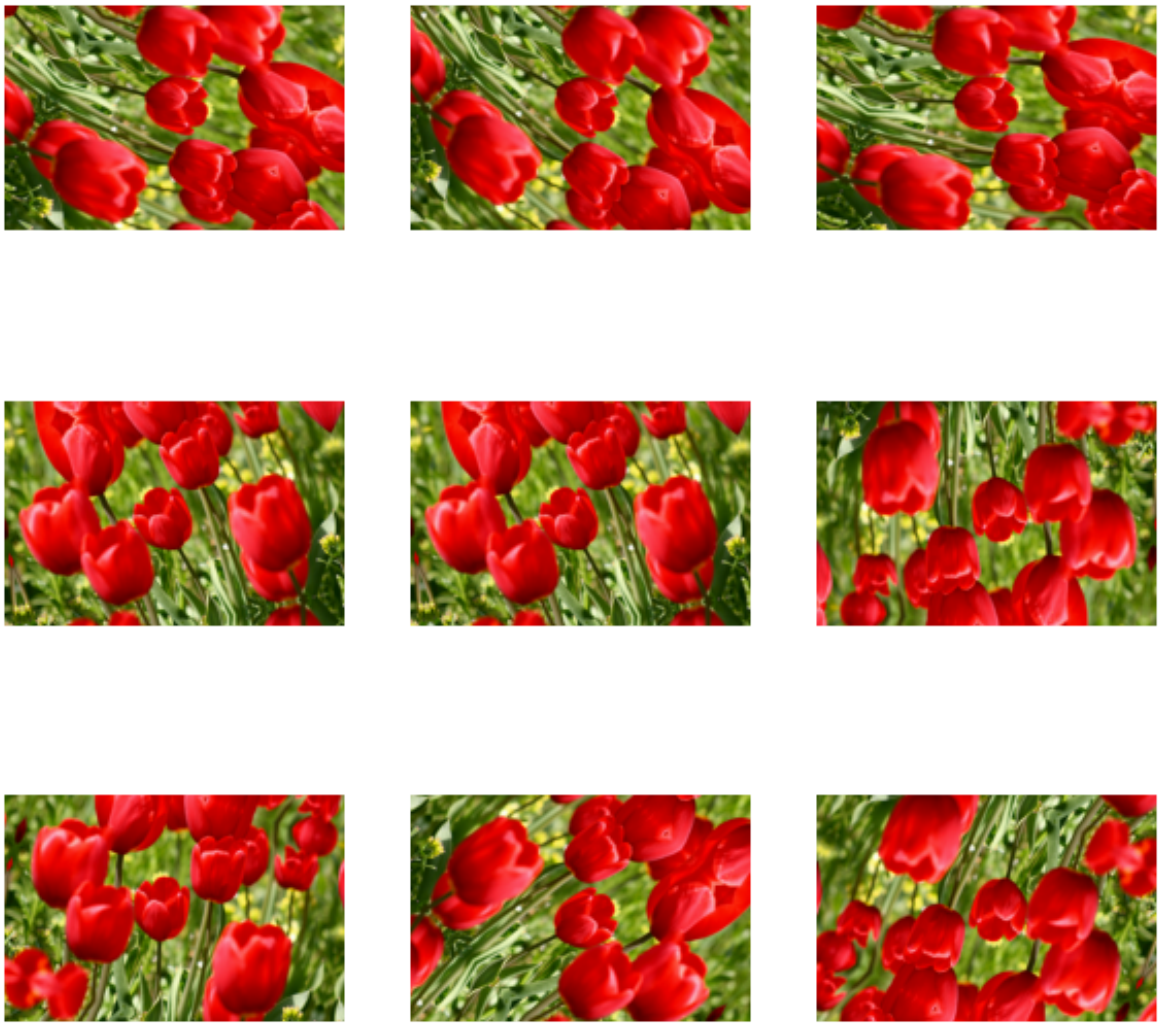


Fig 9. Image being augmented using various functions.

3.3.2 Building CNN & Saving Keras model

A CNN usually has 3 types of layers:

1) Convolutional Layer (CONV)

2) Pooling Layer (POOL)

3) Fully Connected Layer (FC)

Each of these layers is described in detail:

Convolutional Layer

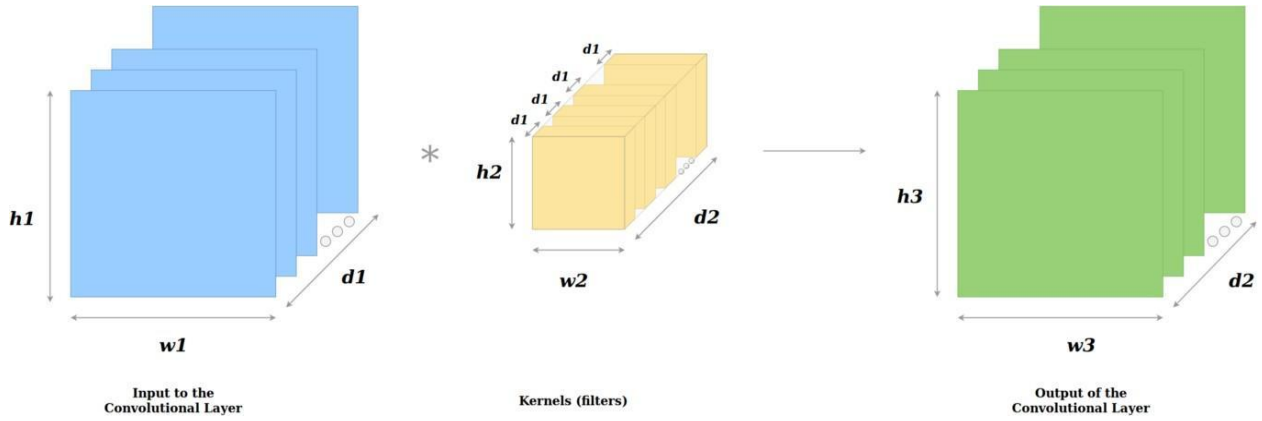


Fig 10. Convolutional Layer Convolutional Layer is the first layer in a CNN.

It gets as input a matrix of the dimensions [h1 * w1 * d1], which is the blue matrix in the above image.

Next, we have kernels (filters).

Kernels: A kernel is a matrix with the dimensions [h2 * w2 * d1], which is one yellow cuboid of the multiple cuboids (kernels) stacked on top of each other (in the kernels layer) in the above image.

For each convolutional layer, there are multiple kernels stacked on top of each other, this is what forms the yellow 3-dimensional matrix in Fig 10., which is of dimensions [h2 * w2 * d2], where d2 is the number of kernels.

For each kernel, we have its respective bias, which is a scalar quantity.

And then, we have an output for this layer, the green matrix in Fig 10., which has dimensions [h3 * w3 * d2].

Observations from the above content:

1) The depth (d1) (or the number of channels) of the input and one kernel is the same.

2) The depth (d2) of the output is equal to the number of kernels (i.e. the depth of the orange 3-dimensional matrix).

Alright, so we have inputs, kernels, and outputs. Now let's look at what happens with a 2D input and a 2D kernel, i.e. d1=1.

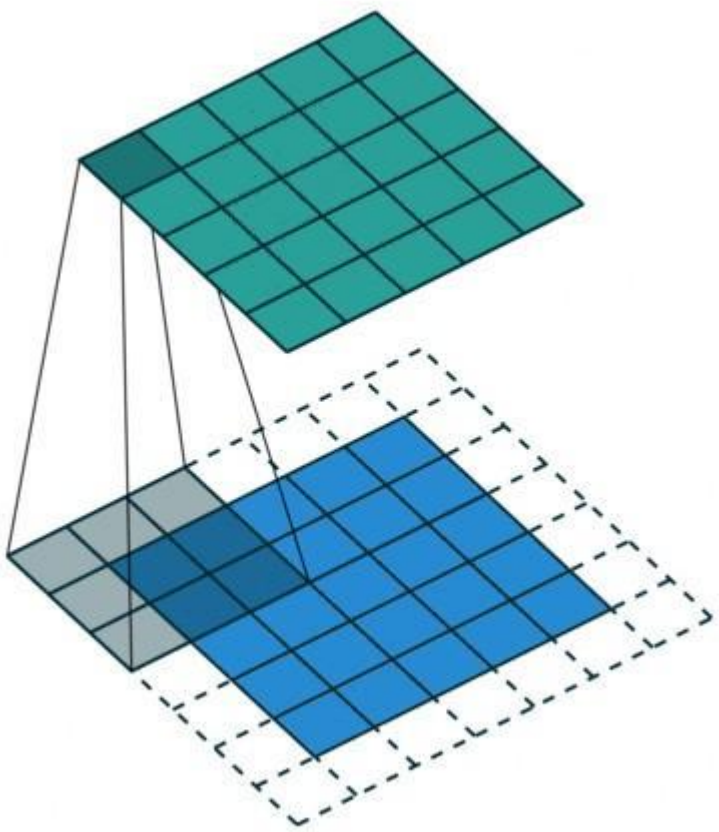


Fig 11. Calculation of output using 2D Convolution

For each position of the kernel on the image, each number on the kernel gets multiplied with the corresponding number on the input matrix (blue matrix) and then they all are summed up for the value in the corresponding position in the output matrix (green matrix).

With d1 > 1, the same thing occurs for each of the channels and then they are added up together and then summed up with the bias of the respective filter and this forms the value in the corresponding position of the output matrix.

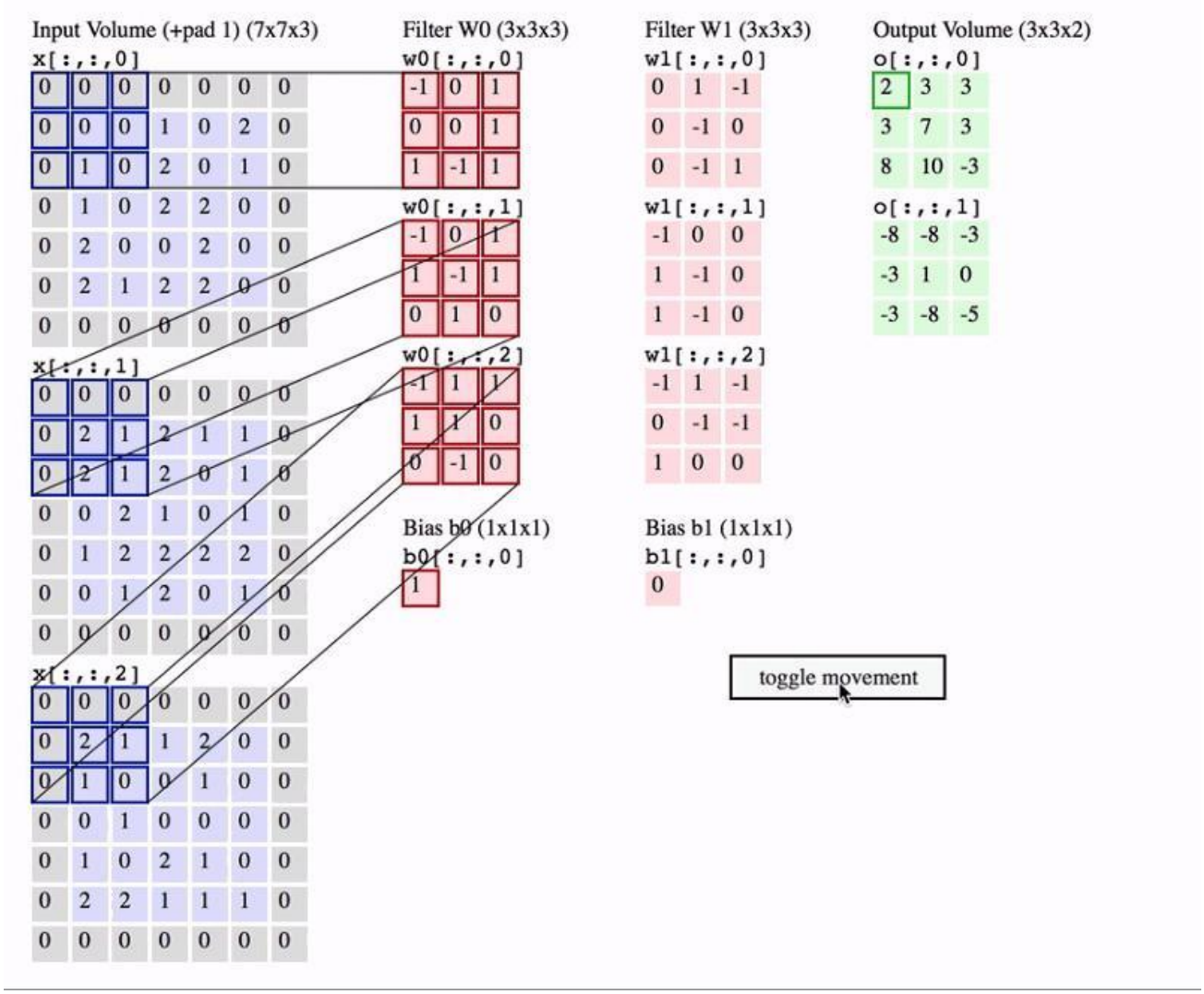


Fig 12. An example of how Inputs are mapped to Outputs

And this forms one (of d2) matrix of the output layer.

This entire process is repeated with all the d2 kernels which form the d2 channels in the output layer.

Pooling Layer

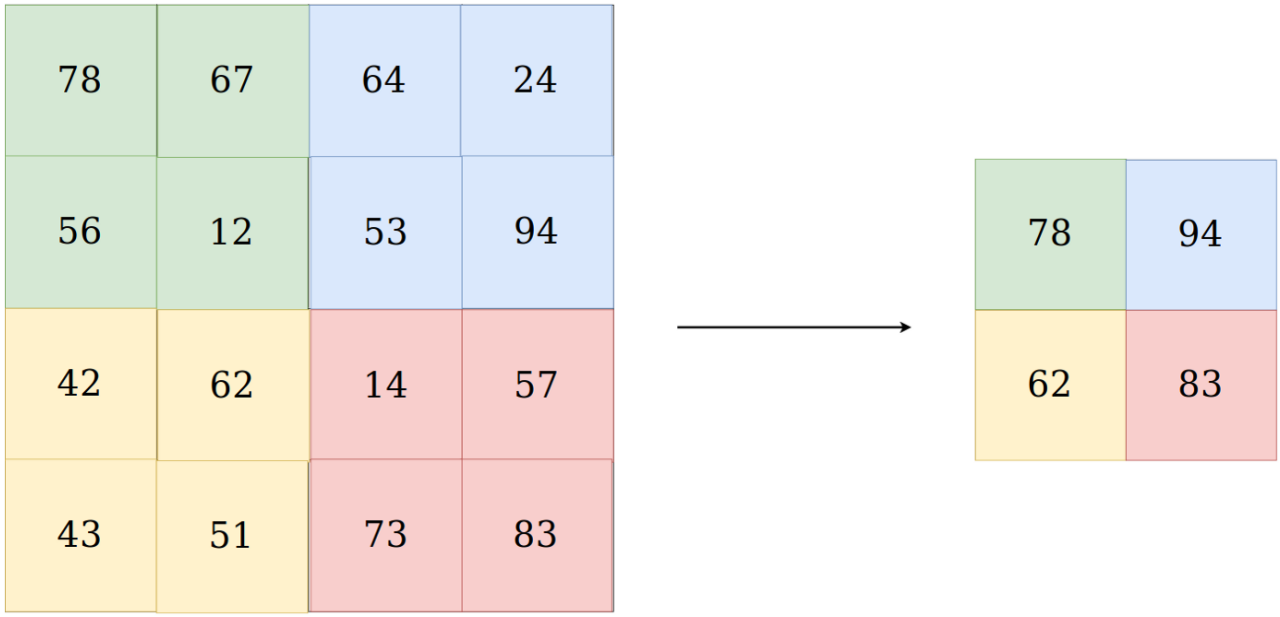


Fig 13. Max Pooling Layer

There are two types of pooling:

1) Max Pooling

2) Average Pooling

The main purpose of a pooling layer is to reduce the number of parameters of the input tensor and thus:

- Helps reduce overfitting
- Extract representative features from the input tensor
- Reduces computation and thus aids efficiency The input to the Pooling layer is tensor.

In the case of Max Pooling, an example of which is shown in Fig 13., a kernel of size n*n (2x2 in the above example) is moved across the matrix, and for each position, the max value is taken and put in the corresponding position of the output matrix.

In the case of Average Pooling, a kernel of size n*n is moved across the matrix, and for each position, the average is taken of all the values and put in the corresponding position of the output matrix.

This is repeated for each channel in the input tensor. And so, we get the output tensor.

So, a thing to note is, that Pooling downsamples the image in its height and width but the number of channels(depth) stays the same.

Fully Connected Layer

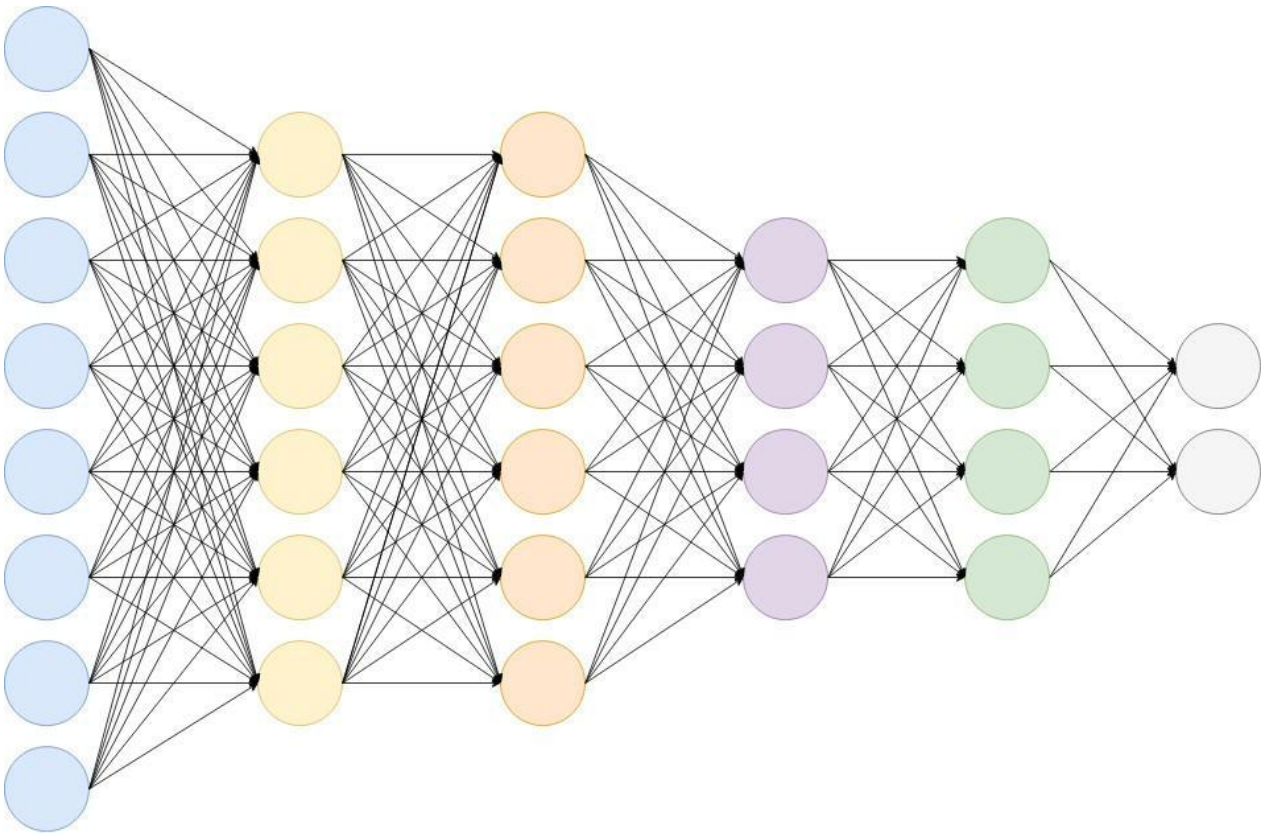


Fig 14. Fully Connected Network

Fully Connected Layer simply, feeds forward neural networks. Fully Connected Layers form the last few layers in the network.

The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

Flattened: The output from the final (and any) Pooling and Convolutional Layer is a 3-dimensional matrix, to flatten that is to unroll all its values into a vector.

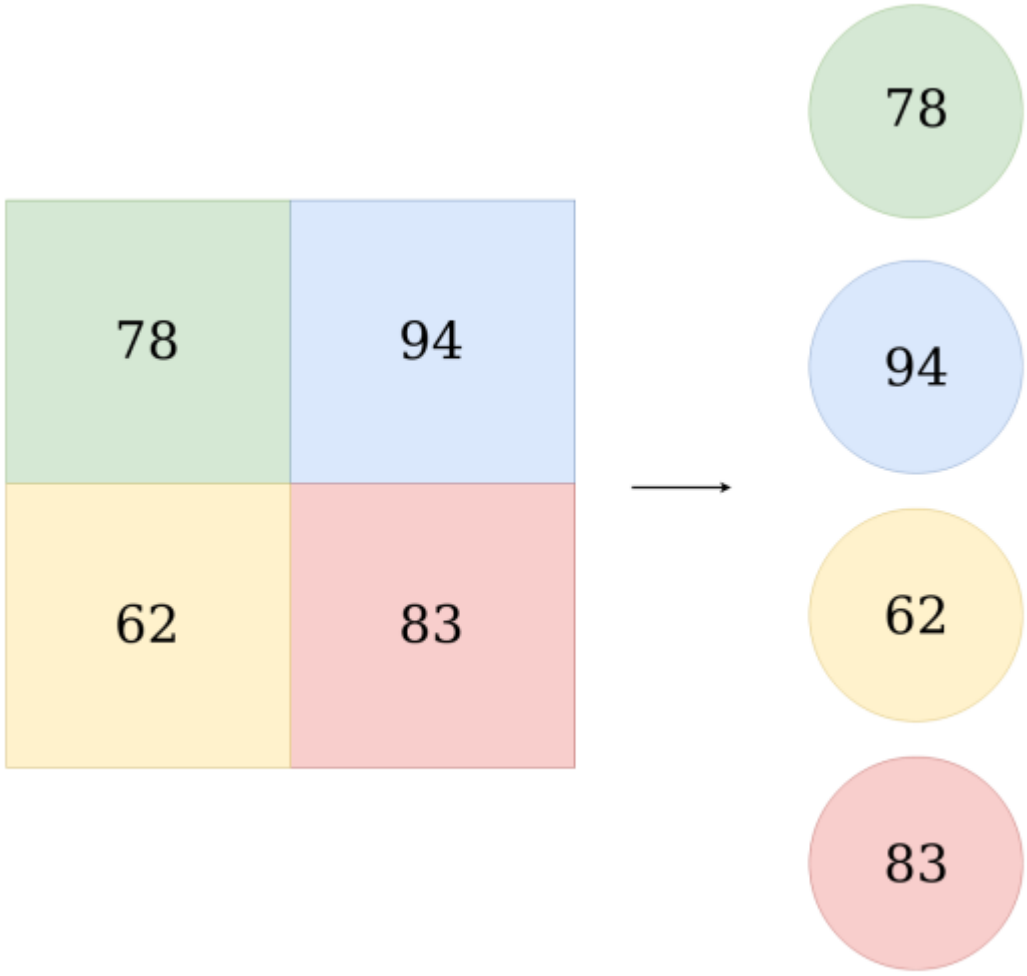


Fig 15. Flattening

This Flattened vector is then connected to a few fully connected layers which are the same as Artificial Neural Networks and perform the same mathematical operations.

For each layer of the Artificial Neural Network, the following calculation takes place

Fig 16. ANN Calculation for each layer

where,

x — is the input vector with dimension [p_l, 1]

W — Is the weight matrix with dimensions [p_l, n_l] where p_l is the number of neurons in the previous layer and n_l is the number of neurons in the current layer.

b — Is the bias vector with dimension [p_l, 1]

g — Is the activation function, which is usually ReLU. This calculation is repeated for each layer.

After passing through the fully connected layers, the final layer uses the SoftMax activation function (instead of ReLU) which is used to get probabilities of the input being in a particular class (classification).

And so finally, we have the probabilities of the object in the image belonging to the different classes. **Input images get classified as labels.**

The process to calculate the dimensions of the output tensor from the input tensor:

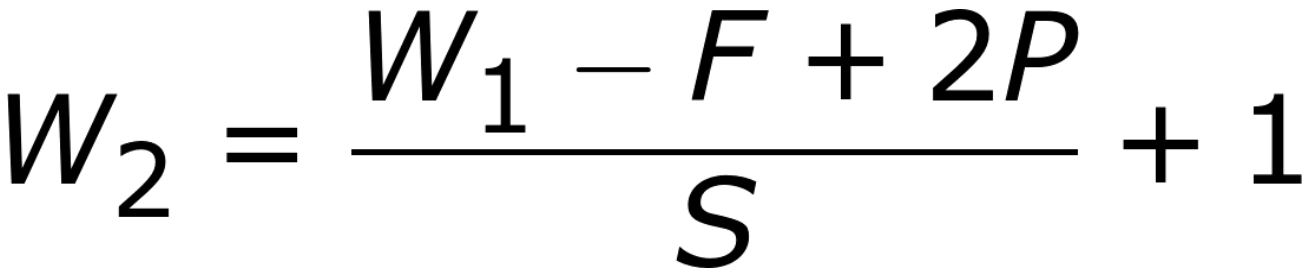


Fig 17. Output Dimension Calculations from Input Dimensions

where,

W1 — is the width/height of the input tensor F — is the width/height of the kernel

P — is the padding S — is the stride

W2 — is the output width/height

Padding: Usually, the input matrices are padded around with zero so that the kernel can move over the matrix uniformly and the resultant matrix has the desired dimension. So, P=1 means that all sides of the input matrix are padded with 1 layer of zeros as in Fig 12. with the dotted extras around the input (blue) matrix.

Stride: The kernel moves over the matrix 1 pixel at a time (Fig 12.). So, this is said to have a stride of 1. We can increase the stride to 2 so that the kernel moves over the matrix 2 pixels at a time. This will, in turn, affect the dimensions of the output tensor and helps reduce overfitting.

This is the architecture of the built neural network:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_55 (Conv2D)           (None, 300, 300, 32)      896
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 150, 150, 32)      0
_________________________________________________________________
conv2d_56 (Conv2D)           (None, 150, 150, 64)      18496
_________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 75, 75, 64)        0
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 75, 75, 32)        18464
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 37, 37, 32)        0
_________________________________________________________________
flatten_17 (Flatten)         (None, 43808)             0
_________________________________________________________________
dense_41 (Dense)             (None, 64)                2803776
_________________________________________________________________
dropout_27 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_42 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_28 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_43 (Dense)             (None, 6)                 198
=================================================================
Total params: 2,843,910
Trainable params: 2,843,910
Non-trainable params: 0
```

3.3.3 Web Application

Both MIRnet and TrashNet are hosted on a separate port on the same server. Both have an individual web UI and can also be used as individual modules.

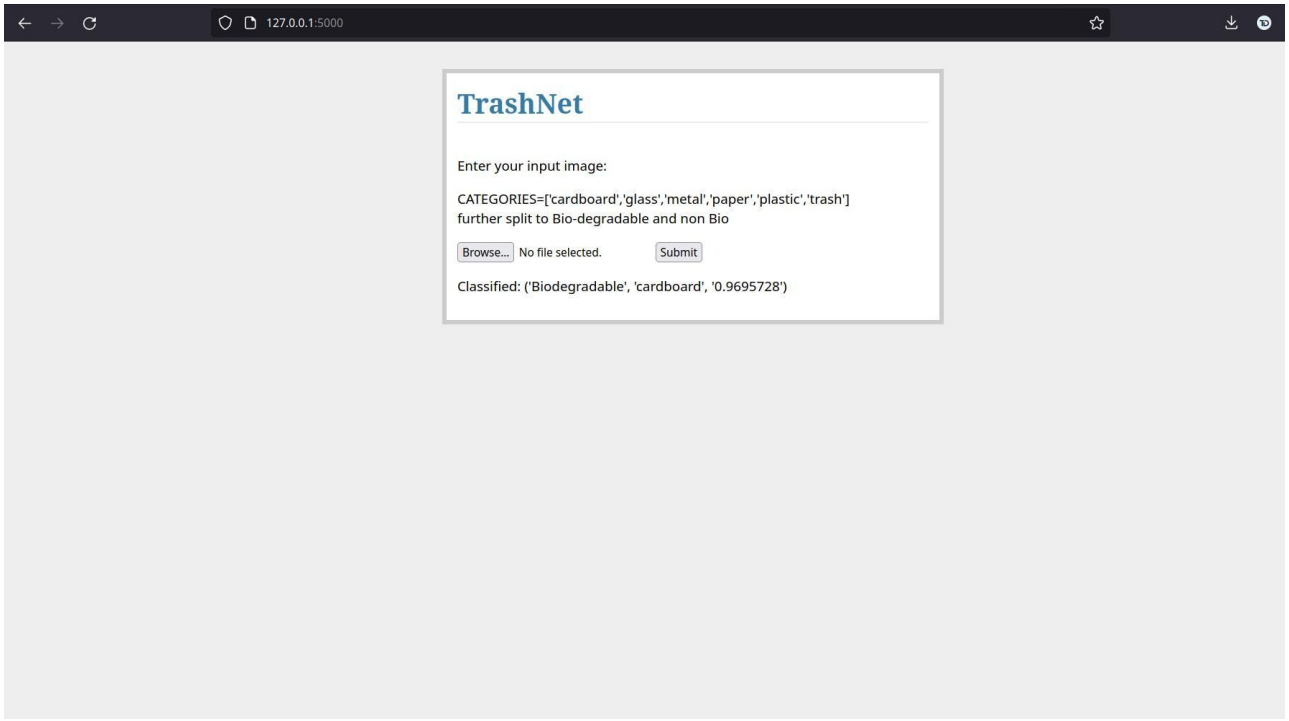


Fig 18. Web Interface of TrashNet

The image is uploaded to the server via this GUI on port 5000, the model is loaded when the GUI is started for the first time, and then data from the image is used to predict the category and its type. Depending on the predicted class we can assign a biodegradable or non-biodegradable category with a simple map function.

Apart from using the model in a web GUI, we can also only load a cli only version where once the image is uploaded the results are shown in the cli and can be passed on to the required machine/hardware to segregate the garbage. We also use this for joining the MIRNet and TrashNet models.

3.4 Creating a Pipeline for MIRNet and TrashNet Model

To join the MIRNet and TrashNet models, we use their cli modes. After loading the image in MIRNet and getting the new image we can send this to TrashNet via a simple POST request to port 5000 (the port where TrashNet is hosted), once TrashNet gets an image on its `/submit` endpoint it does not care whether the image was from the web GUI or someone has sent it anonymously. The model then extracts the data and returns us an object which has the class and its type (biodegradable/non-biodegradable).

4. Performance Evaluation

4.1 Dataset

We have used the TrashNet dataset created by Mindy Yang and Gary Thung[1] at Stanford University which consists of six classes with 2527 images. Photos were taken by placing the object on a white poster board using sunlight with sufficient lighting in the room. Images are resized to 512 x 384.
The details of the dataset are provided in Table I.

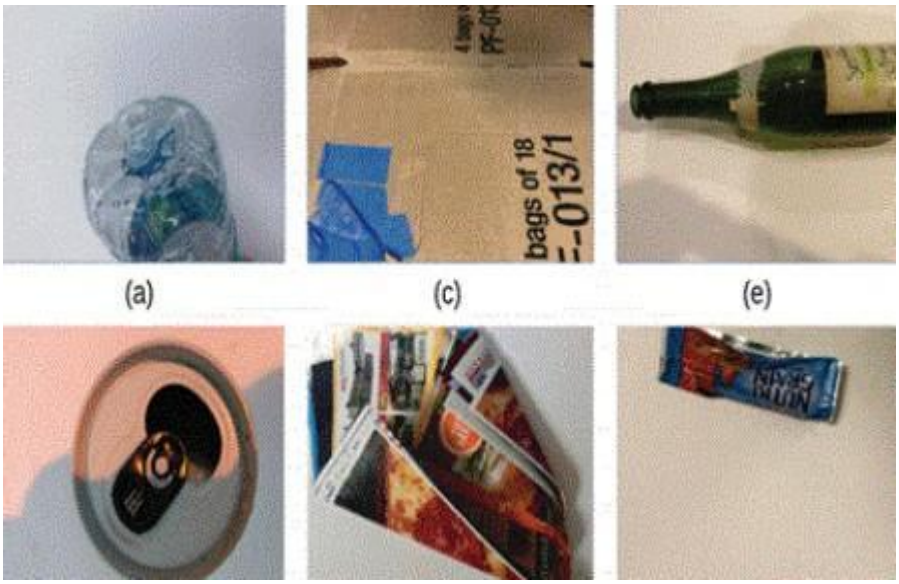


Fig 19. Sample images of waste classes in Trash Net dataset: (a) plastic; (b) metal; (c) cardboard; (d) paper; (e) glass; and (f) general trash.

CLASSES	TRAINING IMAGES	TEST IMAGES	TOTAL
Cardboard	323	80	403
Metal	328	82	410
Glass	404	101	505

Plastic	385	97	482
Paper	475	119	594
Trash	109	28	137

TABLE I. DETAILS OF THE DATASET USED FOR GARBAGE SEGREGATION

4.2 Experimental Results

MIRNet results -

MIRNet runs on port 3000 and sends output images to TrashNet, here, for example, we have uploaded an image of cardboard taken by us (not a part of the dataset). The output of the image can be seen brightened.

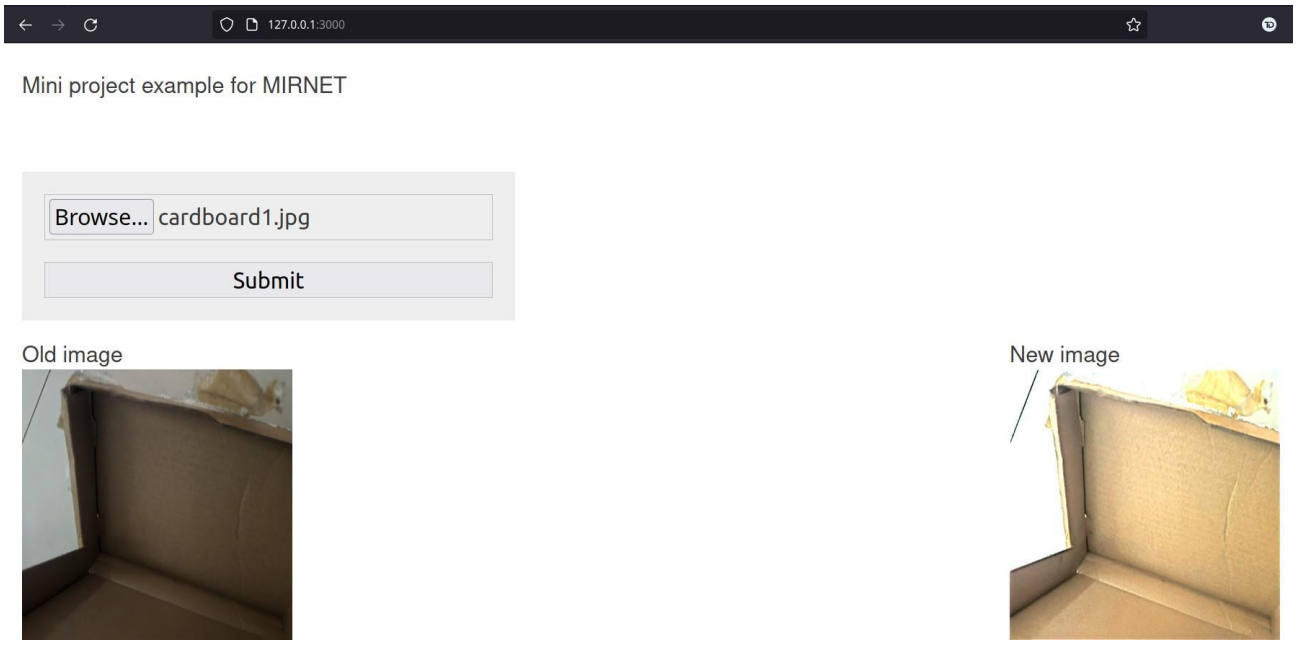


Fig 20. Web Interface of MIRNet

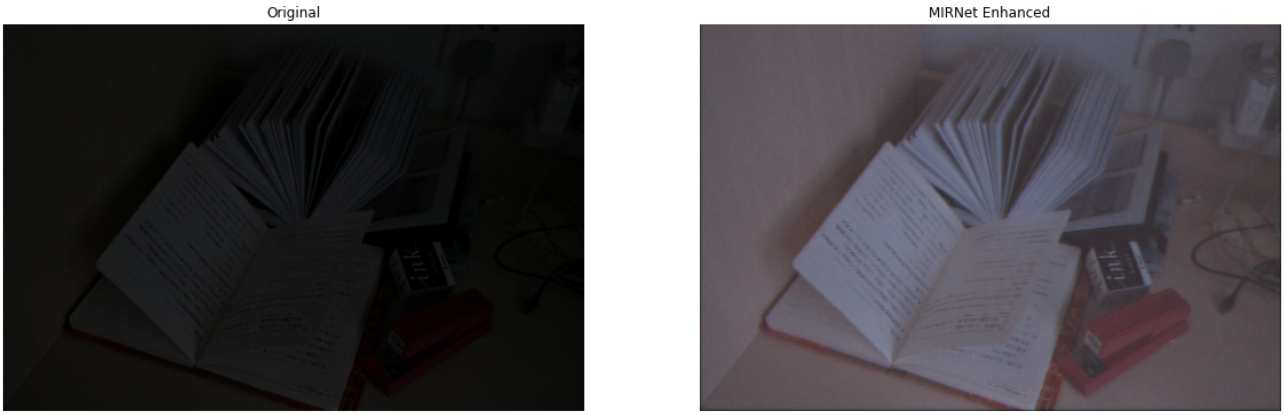


Fig 21. Output Images of MIRNet

Fig 22. Output Images of MIRNet

Fig 23. Output Images of MIRNet

TrashNet results -

TrashNet running on port 5000 on a server, the output image of MIRNet is uploaded to the TrashNet web interface and in this case, it detects the image object like cardboard with an accuracy of 95%.

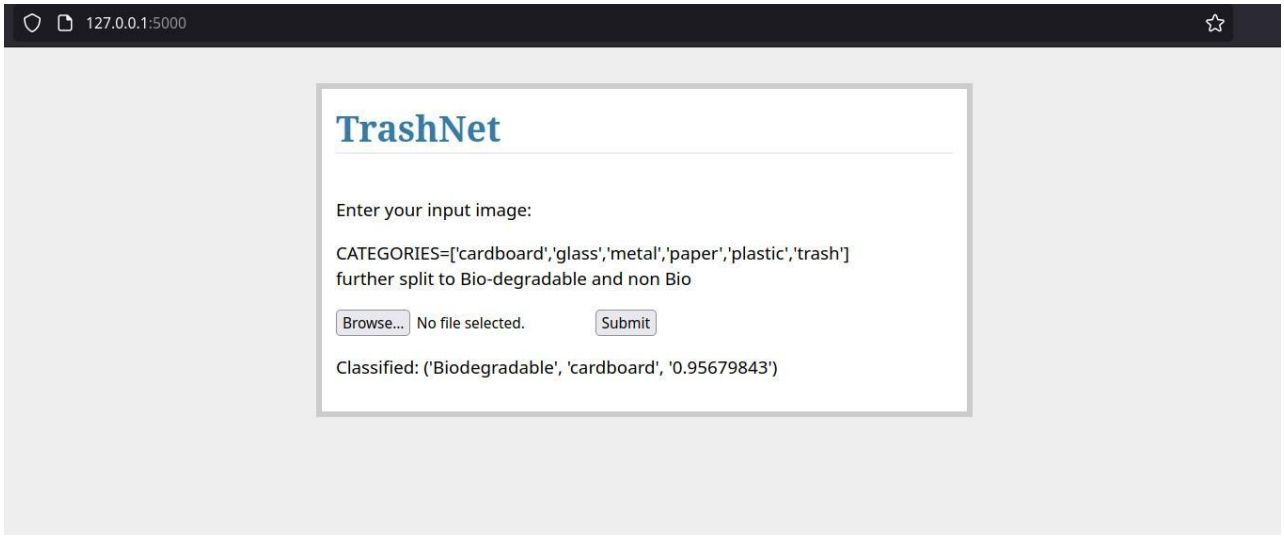


Fig 24. Output of TrashNet

```
p.shape: (1, 6)
prob 0.9695728
classified label: cardboard
('Biodegradable', 'cardboard', '0.9695728')
127.0.0.1 - - [04/Apr/2022 01:43:59] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [04/Apr/2022 01:43:59] "GET /static/style.css HTTP/1.1" 3
04 -
```

Fig 25. Output Results

4.3 Results and discussion

We can classify photos into six categories using our TrashNet model. (waste, cardboard, paper, plastic, glass, metal) Using clean, well-lit photos, we were able to achieve an accuracy of roughly 80%. Validation was done with the remaining 251 photos from the Gary-Mindy dataset.

We also tested this with 15 photographs of various items from our houses in a dark environment, and while the accuracy decreases to approximately 36%, feeding the same set of 15 images through the MIRNET model and then running TrashNet on the batch gives us an accuracy of around 73%.

5. Conclusion and Scope for Further Work

5.1 Conclusion

In the proposed approach, features extracted from CNNs are used for the automatic classification of garbage. The MIRNet Model is used for low light image enhancement. The input image goes through the MIRNet Model and the image features are enhanced here. The output image of the MIRNet Model is used as the input for the TrashNet Model. Here, the image objects get detected and classified into 6 classes and finally 2 main classes namely Biodegradable and NonBiodegradable.

5.2 Scope for Further Work

In the future, we would like to extend this project to identify and classify multiple objects from a single picture or video data. This could help recycling facilities more by processing a stream of recycling rather than single objects.

Bibliography

[1] Yang, Mindy, and Gary Thung. "Classification of trash for recyclability status." CS229 Project Report 2016 (2016).

[2] Kennedy, Tom. "OscarNet: Using transfer learning to classify disposable waste." CS230 Report: Deep Learning. Stanford University, CA, Winter (2018).

[3] Awe, Oluwasanya, Robel Mengistu, and Vikram Sreedhar. "Smart trash net: Waste localization and classification." arXiv preprint (2017).

[4] Bobulski, Janusz, and Mariusz Kubanek. "PROJECT OF SORTING SYSTEM FOR PLASTIC GARBAGE IN SORTING PLANT BASED ON ARTIFICIAL INTELLIGENCE."

[5] Kang, Zhuang, Jie Yang, Guilan Li, and Zeyi Zhang. "An Automatic Garbage Classification System Based on Deep Learning." IEEE Access 8 (2020): 140019-140029.

[6] Vinodha, D., J. Sangeetha, B. Cynthia Sherin, and M. Renukadevi. "Smart Garbage System with Garbage Separation Using Object Detection." International Journal of Research in Engineering, Science and Management (2020).

[7] Bircanolu, Cenk, Meltem Atay, Fuat Beer, Özgün Genç, and Merve Ayyüce Kzrak. "RecycleNet: Intelligent waste sorting using deep neural networks." In 2018 Innovations in Intelligent Systems and Applications (INISTA), pp. 1-7. IEEE, 2018.

[8] Shaikh, Farzana, Nagma Kazi, Farheen Khan, and Zaid Thakur. "Waste Profiling and Analysis using Machine Learning." In 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), pp. 488-492. IEEE, 2020.

[9] Thung, G. "Trashnet." GitHub repository (2016). [13] Vo, Anh H., Minh Thanh Vo, and Tuong Le. "A novel framework for trash classification using deep transfer learning." IEEE Access 7 (2019): 178631-178639.

[10] Ahmad, Kashif, Khalil Khan, and Ala Al-Fuqaha. "Intelligent fusion of deep features for improved waste classification." IEEE Access 8 (2020): 96495-96504.

[11] Funch, Oliver Istad, Robert Marhaug, Sampsa Kohtala, and Martin Steinert. "Detecting glass and metal in consumer trash bags during waste collection using convolutional neural networks." Waste Management 119 (2021): 30-38.

[12] Hua, Dorothy, Julia Gao, Roger Mayo, Albert Smedley, Piyush Puranik, and Justin Zhan. "Segregating Hazardous Waste Using Deep Neural Networks in Real-Time Video." In 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), pp. 1016-1022. IEEE,

2020
[13] Despotovski A., Despotovski F., Lameski J., Zdravevski E., Kulakov A., Lameski P. (2020) Towards Cleaner Environments by Automated Garbage Detection in Images. In: Dimitrova V., Dimitrovski I. (eds) ICT Innovations 2020. Machine Learning and Applications. ICT Innovations 2020. Communications in Computer and Information Science, vol 1316. Springer, Cham.
[14] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”. Int. Conf. on Learning Representations, San Diego, CA, 2015

Appendix

I. MIRNet Model

```python
def selective_kernel_feature_fusion(
    multi_scale_feature_1, multi_scale_feature_2, multi_scale_feature_3
):
    channels = list(multi_scale_feature_1.shape)[-1]
    combined_feature = layers.Add()(
        [multi_scale_feature_1, multi_scale_feature_2, multi_scale_feature_3]
    )
    gap = layers.GlobalAveragePooling2D()(combined_feature)
    channel_wise_statistics = tf.reshape(gap, shape=(-1, 1, 1, channels))
    compact_feature_representation = layers.Conv2D(
        filters=channels // 8, kernel_size=(1, 1), activation="relu"
    )(channel_wise_statistics)
    feature_descriptor_1 = layers.Conv2D(
        channels, kernel_size=(1, 1), activation="softmax"
    )(compact_feature_representation)
    feature_descriptor_2 = layers.Conv2D(
        channels, kernel_size=(1, 1), activation="softmax"
    )(compact_feature_representation)
    feature_descriptor_3 = layers.Conv2D(
        channels, kernel_size=(1, 1), activation="softmax"
    )(compact_feature_representation)
    feature_1 = multi_scale_feature_1 * feature_descriptor_1
    feature_2 = multi_scale_feature_2 * feature_descriptor_2
    feature_3 = multi_scale_feature_3 * feature_descriptor_3
    aggregated_feature = layers.Add()([feature_1, feature_2, feature_3])
    return aggregated_feature


def spatial_attention_block(input_tensor):
    average_pooling = tf.reduce_max(input_tensor, axis=-1)
    average_pooling = tf.expand_dims(average_pooling, axis=-1)
    max_pooling = tf.reduce_mean(input_tensor, axis=-1)
    max_pooling = tf.expand_dims(max_pooling, axis=-1)
    concatenated = layers.Concatenate(axis=-1)([average_pooling, max_pooling])
    feature_map = layers.Conv2D(1, kernel_size=(1, 1))(concatenated)
    feature_map = tf.nn.sigmoid(feature_map)
    return input_tensor * feature_map


def channel_attention_block(input_tensor):
    channels = list(input_tensor.shape)[-1]
    average_pooling = layers.GlobalAveragePooling2D()(input_tensor)
    feature_descriptor = tf.reshape(average_pooling, shape=(-1, 1, 1, channels))
    feature_activations = layers.Conv2D(
        filters=channels // 8, kernel_size=(1, 1), activation="relu"
    )(feature_descriptor)
    feature_activations = layers.Conv2D(
        filters=channels, kernel_size=(1, 1), activation="sigmoid"
    )(feature_activations)
    return input_tensor * feature_activations


def dual_attention_unit_block(input_tensor):
    channels = list(input_tensor.shape)[-1]
    feature_map = layers.Conv2D(
        channels, kernel_size=(3, 3), padding="same", activation="relu"
    )(input_tensor)
    feature_map = layers.Conv2D(channels, kernel_size=(3, 3), padding="same")(
        feature_map
    )
    channel_attention = channel_attention_block(feature_map)
    spatial_attention = spatial_attention_block(feature_map)
    concatenation = layers.Concatenate(axis=-1)([channel_attention, spatial_attention])
    concatenation = layers.Conv2D(channels, kernel_size=(1, 1))(concatenation)
    return layers.Add()([input_tensor, concatenation])


  # Recursive Residual Modules
```

Training

```python
import os
import cv2
import random
import numpy as np
from glob import glob
from PIL import Image, ImageOps
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


random.seed(10)

IMAGE_SIZE = 128
BATCH_SIZE = 4
MAX_TRAIN_IMAGES = 300


def read_image(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=3)
    image.set_shape([None, None, 3])
    image = tf.cast(image, dtype=tf.float32) / 255.0
    return image


def random_crop(low_image, enhanced_image):
    low_image_shape = tf.shape(low_image)[:2]
    low_w = tf.random.uniform(
        shape=(), maxval=low_image_shape[1] - IMAGE_SIZE + 1, dtype=tf.int32
    )
    low_h = tf.random.uniform(
        shape=(), maxval=low_image_shape[0] - IMAGE_SIZE + 1, dtype=tf.int32
    )
    enhanced_w = low_w
    enhanced_h = low_h
    low_image_cropped = low_image[
        low_h : low_h + IMAGE_SIZE, low_w : low_w + IMAGE_SIZE
    ]
    enhanced_image_cropped = enhanced_image[
        enhanced_h : enhanced_h + IMAGE_SIZE, enhanced_w : enhanced_w + IMAGE_SIZE
    ]
    return low_image_cropped, enhanced_image_cropped


def load_data(low_light_image_path, enhanced_image_path):
    low_light_image = read_image(low_light_image_path)
    enhanced_image = read_image(enhanced_image_path)
    low_light_image, enhanced_image = random_crop(low_light_image, enhanced_image)
    return low_light_image, enhanced_image


def get_dataset(low_light_images, enhanced_images):
    dataset = tf.data.Dataset.from_tensor_slices((low_light_images, enhanced_images))
    dataset = dataset.map(load_data, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
    return dataset


train_low_light_images = sorted(glob("./lol_dataset/our485/low/*"))[:MAX_TRAIN_IMAGES]
train_enhanced_images = sorted(glob("./lol_dataset/our485/high/*"))[:MAX_TRAIN_IMAGES]

val_low_light_images = sorted(glob("./lol_dataset/our485/low/*"))[MAX_TRAIN_IMAGES:]
val_enhanced_images = sorted(glob("./lol_dataset/our485/high/*"))[MAX_TRAIN_IMAGES:]

test_low_light_images = sorted(glob("./lol_dataset/eval15/low/*"))
test_enhanced_images = sorted(glob("./lol_dataset/eval15/high/*"))


train_dataset = get_dataset(train_low_light_images, train_enhanced_images)
val_dataset = get_dataset(val_low_light_images, val_enhanced_images)


print("Train Dataset:", train_dataset)
print("Val Dataset:", val_dataset)
```

```python
def down_sampling_module(input_tensor):
    channels = list(input_tensor.shape)[-1]
    main_branch = layers.Conv2D(channels, kernel_size=(1, 1), activation="relu")(
        input_tensor
    )
    main_branch = layers.Conv2D(
        channels, kernel_size=(3, 3), padding="same", activation="relu"
    )(main_branch)
    main_branch = layers.MaxPooling2D()(main_branch)
    main_branch = layers.Conv2D(channels * 2, kernel_size=(1, 1))(main_branch)
    skip_branch = layers.MaxPooling2D()(input_tensor)
    skip_branch = layers.Conv2D(channels * 2, kernel_size=(1, 1))(skip_branch)
    return layers.Add()([skip_branch, main_branch])


def up_sampling_module(input_tensor):
    channels = list(input_tensor.shape)[-1]
    main_branch = layers.Conv2D(channels, kernel_size=(1, 1), activation="relu")(
        input_tensor
    )
    main_branch = layers.Conv2D(
        channels, kernel_size=(3, 3), padding="same", activation="relu"
    )(main_branch)
    main_branch = layers.UpSampling2D()(main_branch)
    main_branch = layers.Conv2D(channels // 2, kernel_size=(1, 1))(main_branch)
    skip_branch = layers.UpSampling2D()(input_tensor)
    skip_branch = layers.Conv2D(channels // 2, kernel_size=(1, 1))(skip_branch)
    return layers.Add()([skip_branch, main_branch])


# MRB Block
def multi_scale_residual_block(input_tensor, channels):
    # features
    level1 = input_tensor
    level2 = down_sampling_module(input_tensor)
    level3 = down_sampling_module(level2)
    # DAU
    level1_dau = dual_attention_unit_block(level1)
    level2_dau = dual_attention_unit_block(level2)
    level3_dau = dual_attention_unit_block(level3)
    # SKFF
    level1_skff = selective_kernel_feature_fusion(
        level1_dau,
        up_sampling_module(level2_dau),
        up_sampling_module(up_sampling_module(level3_dau)),
    )
    level2_skff = selective_kernel_feature_fusion(
        down_sampling_module(level1_dau), level2_dau, up_sampling_module(level3_dau)
    )
    level3_skff = selective_kernel_feature_fusion(
        down_sampling_module(down_sampling_module(level1_dau)),
        down_sampling_module(level2_dau),
        level3_dau,
    )
    # DAU 2
    level1_dau_2 = dual_attention_unit_block(level1_skff)
    level2_dau_2 = up_sampling_module((dual_attention_unit_block(level2_skff)))
    level3_dau_2 = up_sampling_module(
        up_sampling_module(dual_attention_unit_block(level3_skff))
    )
    # SKFF 2
    skff_ = selective_kernel_feature_fusion(level1_dau_2, level3_dau_2, level3_dau_2)
    conv = layers.Conv2D(channels, kernel_size=(3, 3), padding="same")(skff_)
    return layers.Add()([input_tensor, conv])
```

```python
def recursive_residual_group(input_tensor, num_mrb, channels):
    conv1 = layers.Conv2D(channels, kernel_size=(3, 3), padding="same")(input_tensor)
    for _ in range(num_mrb):
        conv1 = multi_scale_residual_block(conv1, channels)
    conv2 = layers.Conv2D(channels, kernel_size=(3, 3), padding="same")(conv1)
    return layers.Add()([conv2, input_tensor])


def mirnet_model(num_rrg, num_mrb, channels):
    input_tensor = keras.Input(shape=[None, None, 3])
    x1 = layers.Conv2D(channels, kernel_size=(3, 3), padding="same")(input_tensor)
    for _ in range(num_rrg):
        x1 = recursive_residual_group(x1, num_mrb, channels)
    conv = layers.Conv2D(3, kernel_size=(3, 3), padding="same")(x1)
    output_tensor = layers.Add()([input_tensor, conv])
    return keras.Model(input_tensor, output_tensor)


model = mirnet_model(num_rrg=3, num_mrb=2, channels=64)
print(model.layers)
print(len(model.layers))


def charbonnier_loss(y_true, y_pred):
    return tf.reduce_mean(tf.sqrt(tf.square(y_true - y_pred) + tf.square(1e-3)))


def peak_signal_noise_ratio(y_true, y_pred):
    return tf.image.psnr(y_pred, y_true, max_val=255.0)


optimizer = keras.optimizers.Adam(learning_rate=1e-4)
model.compile(
    optimizer=optimizer, loss=charbonnier_loss, metrics=[peak_signal_noise_ratio]
)

history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=1,
    callbacks=[
        keras.callbacks.ReduceLROnPlateau(
            monitor="val_peak_signal_noise_ratio",
            factor=0.5,
            patience=5,
            verbose=1,
            min_delta=1e-7,
            mode="max",
        )
    ],
)


def plot_results(images, titles, figure_size=(12, 12)):
    fig = plt.figure(figsize=figure_size)
    for i in range(len(images)):
        fig.add_subplot(1, len(images), i + 1).set_title(titles[i])
        _ = plt.imshow(images[i])
        plt.axis("off")
    plt.show()


def infer(original_image):
    image = keras.preprocessing.image.img_to_array(original_image)
    image = image.astype("float32") / 255.0
    image = np.expand_dims(image, axis=0)
    output = model.predict(image)
    output_image = output[0] * 255.0
    output_image = output_image.clip(0, 255)
    output_image = output_image.reshape(
        (np.shape(output_image)[0], np.shape(output_image)[1], 3)
    )
    output_image = Image.fromarray(np.uint8(output_image))
    original_image = Image.fromarray(np.uint8(original_image))
    return output_image

# testing
for low_light_image in random.sample(test_low_light_images, 6):
    original_image = Image.open(low_light_image)
    enhanced_image = infer(original_image)
    plot_results(
        [original_image, enhanced_image],
        ["Original", "MIRNet Enhanced"],
        (20, 12),
    )
```

API -

```
app.post("/submit", (req, res) => {
  let form = new formidable.IncomingForm(formOptions);
  form.parse(req, async (err, fields, files) => {
    if (err) {
      res.send("Incorrect File Format");
      console.log("\n" + err + "\n");
    } else {
      console.log("sending File: " + files.image.name);
      // res.sendFile(files.image.path);
      try {
        let toSend = await predict(
          files.image.path,
          path.join(__dirname, "public", "responseImages") + files.image.name
        );
        if (toSend === true) {
          res.sendFile(
            path.join(__dirname, "public", "responseImages") + files.image.name
          );
        } else {
          res.status(501).send(toSend);
        }
      } catch (err) {
        res.status(500).send(err);
      }
    }
  });
  const allowedFiles = ["image/jpg", "image/jpeg", "image/png"];
  try {
    form.on("fileBegin", function (name, file) {
      if (!allowedFiles.includes(file.type)) {
        // throw new Error("Incorrect File Type");
        form._error(new Error("Incorrect File Type"));
        return new Error("Incorrect File Type");
      } else {
        file.path =
          path.join(__dirname, "uploads") + "/" + Date.now() + "-" + file.name;
      }
    });
  } catch (err) {
    form._error(err);
    console.log(err);
    res.send("Incorrect File Type");
  }
});

module.exports = app;
```

Prediction -

```
async function loadModel() {
  try {
    // Warm up the model
    if (!mirNetModel) {
      modelInfo = await tf.node.getMetaGraphsFromSavedModel("./model");
      console.log(await modelInfo);

      mirNetModel = await tf.node.loadSavedModel("./model");
      return await mirNetModel;
    }
  } catch (error) {
    console.log(error);
  }
}

const predict = async (imgPath, responseImagePath) => {
  try {
    mirNetModel = await loadModel();
    console.log(mirNetModel);
    console.log("Inside predict");
    let image = fs.readFileSync(imgPath);
    let imageTensor = await tf.node.decodePng(image, 3);
    imageTensor = tf.image.resizeBilinear(
      imageTensor,
      (size = [imageSize, imageSize])
    );
    imageTensor = tf.cast(imageTensor, "float32");
    imageTensor = tf.div(imageTensor, tf.scalar(255.0));

    let input = imageTensor.expandDims(0);

    // Feed the image tensor into the model for inference.
    const startTime = tf.util.now();

    let outputTensor = mirNetModel.predict(input);

    const endTime = tf.util.now();
    console.log(endTime - startTime);
    console.log("After Predict");

    outputTensor = tf.reshape(outputTensor, [512, 512, 3]);
    outputTensor = tf.mul(outputTensor, tf.scalar(255.0));
    outputTensor = tf.clipByValue(outputTensor, 0, 255);

    outputTensor = await tf.node.encodePng(outputTensor);
    fs.writeFileSync(responseImagePath, outputTensor);

    return true;
  } catch (error) {
    console.log(error);
    return false;
  }
};
```

Running on a directory -

```
const runOnDirectory = async () => {
  const fileNames = fs.readdirSync("./trash");
  for (file in fileNames) {
    await predict(`./trash/${fileNames[file]}`);
  }
};
```

II. TrashNet Model Training -

```
import numpy as np
import cv2
from keras.callbacks import ModelCheckpoint,EarlyStopping
from keras.layers import Conv2D, Flatten, MaxPooling2D,Dense,Dropout,SpatialDropout2D
from keras.models  import Sequential
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img, array_to_img
import random,os,glob
import matplotlib.pyplot as plt

train=ImageDataGenerator(horizontal_flip=True,
                         vertical_flip=True,
                         validation_split=0.1,
                         rescale=1./255,
                         shear_range = 0.1,
                         zoom_range = 0.1,
                         width_shift_range = 0.1,
                         height_shift_range = 0.1,)

test=ImageDataGenerator(rescale=1/255,
                        validation_split=0.1)

train_generator=train.flow_from_directory(dir_path,
                                          target_size=(300,300),
                                          batch_size=32,
                                          class_mode='categorical',
                                          subset='training')

test_generator=test.flow_from_directory(dir_path,
                                        target_size=(300,300),
                                        batch_size=32,
                                        class_mode='categorical',
                                        subset='validation')

labels = (train_generator.class_indices)
print(labels)

labels = dict((v,k) for k,v in labels.items())
print(labels)

model=Sequential()
#Convolution blocks

model.add(Conv2D(32,(3,3), padding='same',input_shape=(300,300,3),activation='relu'))
model.add(MaxPooling2D(pool_size=2))
#model.add(SpatialDropout2D(0.5)) # No accuracy

model.add(Conv2D(64,(3,3), padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
#model.add(SpatialDropout2D(0.5))

model.add(Conv2D(32,(3,3), padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))

#Classification layers
model.add(Flatten())

model.add(Dense(64,activation='relu'))
#model.add(SpatialDropout2D(0.5))
model.add(Dropout(0.2))
model.add(Dense(32,activation='relu'))

model.add(Dropout(0.2))
model.add(Dense(6,activation='softmax'))

filepath="trained_model.h5"
checkpoint1 = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint1]
```

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['acc']) # RMS PROP - No accuracy


history = model.fit_generator(train_generator,
                              epochs=100,
                              steps_per_epoch=2276//32,
                              validation_data=test_generator,
                              validation_steps=251//32,
                              workers = 4,
                              callbacks=callbacks_list)

# predicting
from keras.preprocessing import image

img_path = '../input/garbage classification/Garbage classification/plastic/plastic75.jpg'

img = image.load_img(img_path, target_size=(300, 300))
img = image.img_to_array(img, dtype=np.uint8)
img=np.array(img)/255.0

plt.title("Loaded Image")
plt.axis('off')
plt.imshow(img.squeeze())

p=model.predict(img[np.newaxis, ...])

#print("Predicted shape",p.shape)
print("Maximum Probability: ",np.max(p[0], axis=-1))
predicted_class = labels[np.argmax(p[0], axis=-1)]
print("Classified:",predicted_class)

# saving
import tensorflow as tf
import keras
file = "Garbage.h5"
keras.models.save_model(model,file)
converter = tf.lite.TFLiteConverter.from_keras_model_file(file)
tflite_model=converter.convert()
open("garbage.tflite",'wb').write(tflite_model)
```

API -

```python
from prediction import *
import os, shutil
import cv2
from flask import Flask, render_template, request,jsonify
from PIL import Image
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

app=Flask(__name__)
UPLOAD_FOLDER = os.path.basename('.')
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


@app.route("/", methods=['GET', 'POST'])
def application():
    file=""
    answer = None
    error=""
    if request.method=="POST":
        try:
            file= request.files["image"]
            if file:
                f = os.path.join(app.config['UPLOAD_FOLDER'], file.filename) #upload
                file.save(f)
                # Deleting from path after uploading
                result=predict(file.filename)
                print(result)
                #os.remove(filename)
                if result=="":
                    error="Sorry!"

        except(SyntaxError) as e:
            error ="Could not understand"
            print("Error:" + str(e))

        try:
            if result!="Sorry!":
                answer=result
        except Exception as e:
            print(e)

    return render_template('index.html', file=file,
                           answer=answer, error=error)
```

Prediction in TrashNet model -

```python
def predict(img_path):
     labels={0: 'cardboard', 1: 'glass', 2: 'metal', 3: 'paper', 4: 'plastic', 5: 'trash'}

     img = image.load_img(img_path, target_size=(300, 300))
     img = image.img_to_array(img, dtype=np.uint8)
     img=np.array(img)/255.0

     model = tf.keras.models.load_model("trained_model.h5")
     p=model.predict(img[np.newaxis, ...])
     pro=np.max(p[0], axis=-1)
     print("p.shape:",p.shape)
     print("prob",pro)
     predicted_class = labels[np.argmax(p[0], axis=-1)]
     os.remove(img_path)
     print("classified label:",predicted_class)
     if predicted_class in ['cardboard','paper']:
          category = "Biodegradable"
          predicted_class = str(predicted_class)
          probability = str(pro)
          return category,predicted_class,probability
     elif predicted_class in ['metal','glass','plastic']:
          category = "Non-Biodegradable"
          predicted_class = str(predicted_class)
          probability = str(pro)
          return category,predicted_class,probability
     else:
          category = "Categorizing Difficult"
          predicted_class = str(predicted_class)
          probability = str(pro)
          return category,predicted_class,probability
```

Running on multiple images and getting accuracy in a directory -

```python
def getAcc():
    root_src_dir = 'output'
    root_dst_dir = 'output_testing'
    for src_dir, dirs, files in os.walk(root_src_dir):
        dst_dir = src_dir.replace(root_src_dir, root_dst_dir, 1)
        if not os.path.exists(dst_dir):
            os.makedirs(dst_dir)
        for file_ in files:
            src_file = os.path.join(src_dir, file_)
            dst_file = os.path.join(dst_dir, file_)
            if os.path.exists(dst_file):
                os.remove(dst_file)
            shutil.copy(src_file, dst_dir)
    directory = os.fsencode(root_dst_dir)
    count = 0
    for file in os.listdir(directory):
        filename = os.fsdecode(file)
        ans = filename.rsplit('.')[0][:-1]
        try:
            if file:
                f = os.path.join(root_dst_dir, filename)
                result=predict(f)
                print(result[1], f)
                if ans == result[1]:
                    count+=1

                if result=="":
                    error="Sorry!"

        except(SyntaxError) as e:
            error ="Could not understand"
            print("Error:" + str(e))

        try:
            if result!="Sorry!":
                answer=result
        except Exception as e:
            print(e)
    print(f"running on {root_src_dir}")
    print((count/11)*100)
```

III. Pipeline for MIRNet and TrashNet Models

Acknowledgements

I would like to express my gratitude to the HOD Prof. Bharati Singh and my professors Prof. Mahesh Warang and Prof. Maruti Zalte for their kind cooperation and encouragement which helped me in the completion of this project. I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

My thanks and appreciation also go to my colleague in developing the project and the people who have willingly helped me out with their abilities.