International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-10, Issue-4, July 2022
https://doi.org/10.55524/ijircst.2022.10.4.20
Article ID IRPV1048, Pages 169-177
www.ijircst.org

Comparison of Color Classification Using Computer Vision and Deep Neural Network

Mir Rahil[1], and Ravinder Pal Singh[2]

[1] M. Tech Scholar, Department of Electronics and Communication Engineering, RIMT University, Mandi Gobindgarh, Punjab, India
[2] Professor, Department of Electronics and Communication Engineering, RIMT University, Mandi Gobindgarh, Punjab, India

Correspondence should be addressed to Mir Rahil; mirrahil206@gmail.com

ABSTRACT- Research on artificial intelligence and machine learning is currently ongoing and is focused on real-world problems. Machine learning is used by computers to make predictions based on the provided data set or existing knowledge. The main goal of our project is to use machine learning to categorize different colors while separating CNN from computer vision. In this work, we used supervised learning to categorize different hues using a binary classification approach. Color detection is the technique of identifying a color. In this scenario, humans can recognize the hue and choose with ease. A computer, however, cannot quickly recognize color. It is challenging to get a computer to quickly detect the color. Given that, we decide to pursue this initiative. Pandas, OpenCV, and the Naive Bayes algorithm are all used in Python. Naive Bayes classifiers are models that assign category labels to issue occurrences that are represented as vectors of feature values, where the category labels are selected from a finite set. There isn't a single method for training these classifiers; rather, there is a family of algorithms built on the premise that, given a category variable, the value of one feature is independent of the value of the other feature. Open-Source Computer Vision Library OpenCV was designed to be computationally effective and with a major emphasis on real-time applications. specialized video encoding for the cloud. Panda may be a platform that runs in the cloud and provides infrastructure for encoding audio and video.

KEYWORDS- Open CV, Colour detection, CNN

I. INTRODUCTION

The project's main objective is to identify color colors using machine learning and vision methods and to distinguish between open CV and CNN models.
The objective is to recognize a color from a photograph taken with a camera and to test the predictions generated by the machine learning system under various lighting conditions. The development, testing, and fine-tuning of a learning model to spot patterns, predict test data, and assess the machine's predictions produced in both favourable and unfavourable lighting circumstances are additional objectives in addition to this experiment.
The Python project here is for color recognition. The program was created in Python using OpenCV libraries, which can automatically recognize the name of the color. Naive Bayes classifiers were used to get the results. We now have a dataset with names and values for many colors. Then each color's distance will be calculated to determine which color is closest. The field of computer vision emerged in the late 1960s as AI advanced. To increase the intelligence of the existing synthetic mechanisms and have them interpret what they saw in a way akin to human sensory systems, cameras were put into them.
Therefore, real-world 3D objects should be recognizable from 2D photos using computer vision. Every photograph tells a story, captures a moment in time, or depicts an ongoing event. Fig 1. There are available values for r, g, and b. We now need a new technique to separate the color name from the RGB value. We calculate a distance (d), which shows how close we are to selecting the choice with the least distance, before choosing the color name.
We'll evaluate the CNN strategy and contrast it with other approaches depending on how accurate they are.

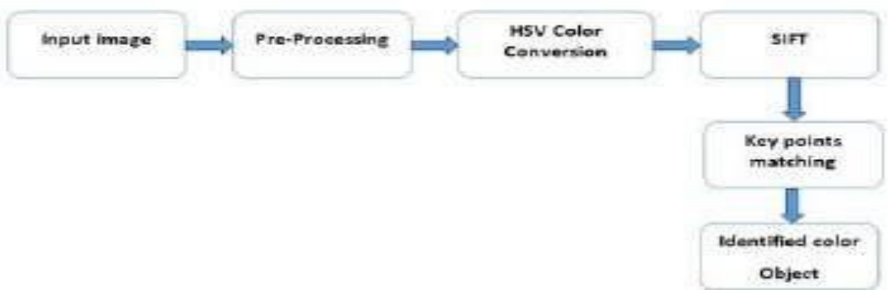


Figure 1: Process using computer vision techniques

II. LITERATURE REVIEW

As a result, "biohazard waste" refers to a category of garbage that requires expensive treatment, packing, and burning, which is predicted to cost eight times more per tonne than solid waste for the disposal procedure. A single technique of biological waste treatment or disposal, according to research in the literature, cannot completely remove all threats to the environment or people (Stankovic and Krall, 2007; Krall and Stankovic, 2006).

Therefore, in light of the risks to environmental health posed by improper management of medical waste, WHO is offering assistance to the relevant nations. There is a need for shared treatment and disposal facilities coordinated by the medical director and governed by the city authorities since each healthcare facility cannot have its treatment and disposal system (Celikyay).

The tools provided by the WHO aid the relevant nations' analysis and decision-making in the development of thorough guidelines and policies. This endeavor will serve as a basis for access to well-planned and thorough medical waste management. The U.S. Environmental Protection Agency (EPA), the Occupational Safety & Health Administration, and the U.S. Navy provided the current information for this Afloat Medical Waste Management Guide (OSHA). The manual also covers crucial elements of OSHA's Bloodborne Pathogens Standard as they relate to ships (e.g., personal exposure preventive measures, personal exposure control measures, and training requirements). Environmental Protection, Chief of Naval Operations, 1999.

III. METHODOLOGY

A. CNN Neural Network Model

The structure of a convolutional neural network is shown below in Figure 2

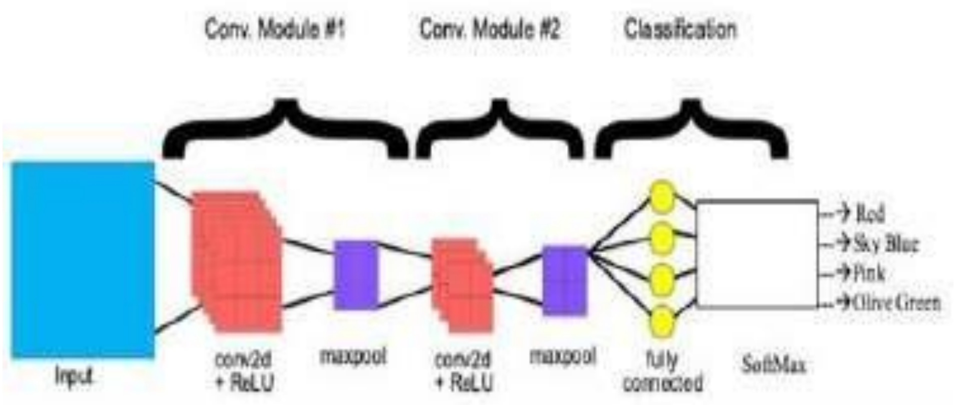


Figure 2: End-to-end structure of a convolution neural network [1]

Fully linked neural networks frequently perform poorly on visual data. This is because every pixel serves as an input, and the number of parameters keeps growing as we add more layers [1]. One picture can be distinguished from another by its unique structure [1]. Images place a great deal of importance on nearby or nearby places. A higher- and higher-level representation of the picture contents might be extracted using CNN.

1) Convolution

Input feature map, a three-dimensional matrix with the first two dimensions equal to the length and width of the pictures in pixels, is the initial type of data that CNN receives. The third dimension's (RGB color image's) size is 3 [1].

A convolution in CNN takes one tile (3x3 or 5x5) from the input feature map, applies filters (the same size as the tile) over them, and creates an output feature map or convolved feature.

In this convolution stage, as illustrated in Figure 3 below, the filters essentially glide across the grid of the input feature map from top to bottom and left to right, extracting each tile one pixel at a time [2].

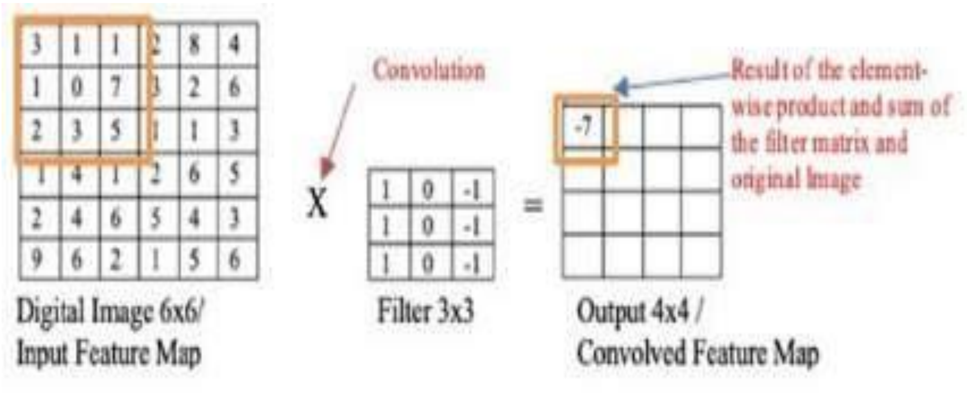

3 1 1 2 8 4
1 0 7 3 2 6
2 3 5 1 1 3
1 4 1 2 6 5
2 4 6 5 4 3
9 6 2 1 5 6
Convolution
X
1 0 -1
1 0 -1
1 0 -1
=
-7
Result of the element-wise product and sum of the filter matrix and original Image
Digital Image 6x6/ Input Feature Map
Filter 3x3
Output 4x4 / Convolved Feature Map

Figure 3: Convolution Neural Network [2]

The CNN multiplies the filter matrix and the tile matrix element by element for each filter tile, adding all the elements of the resultant matrix to obtain a single value. The dot product is comparable to this procedure. The convolved feature matrix then outputs each of the dot product results for every filter-tile pair.
The CNN "learns" during retraining the best filter matrices to use to extract useful features from the input feature map[2] The number of features that the CNN can extract rises along with the number of filters that are applied to the input feature map, but training time also increases as more filters are added to the CNN. Additionally, each additional filter that is introduced to the network adds less incremental value than the one before it. Therefore, we must build a network that employs the fewest possible filters to retrieve the characteristics required for precise picture categorization [3].

2) *Activation Functions*

The neural network has an activation function as one of its parts. The activation process determines whether or not a neuron fire. We need to make sure that these transfer functions are complex to guarantee that our network has some non - linearities [3]. As our input layer, yields an output of zero or one, we may utilize a point estimate. We have one neuronal if the output is greater than a certain level. The output is not fired and the value of the output is zero if it is less than the threshold [3].

3) *ReLU*

One of the most well-known Activation functions is the ReLU. For every value of x that is less than zero, the ReLU, or "Rectified Linear Unit," output is a zero. The function returns x for any x value that is higher than or equal to zero. to provide nonlinearity to the model, CNN adds a Rectified Linear Unit (ReLU) modification to the convolved feature after each convolution operation.

4) *Pooling*

The next stage following ReLU is pooling, in which the CNN decreases the sampling of the original image feature and the number of feature map axes while maintaining the most important feature data. One of the most used techniques is termed max pooling, and it is demonstrated in Figure 4 below.
Other pools include the average pool and the min pool. Max pooling operates in a similar way to convolution. It moves around the feature map and pulls out tiles of a particular size. The maximum value from each extracted tile is then outputted to a new feature map, and all other values are ignored. Two parameters are required for max-pooling operations [4]. The size of the max-pooling filter is typically 2x2 pixels.

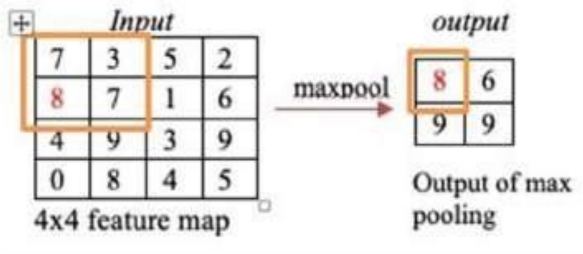


Figure 4: Max pool

The term "stride" refers to the pixel-based separation between each extracted tile. The stride in max-pooling controls the regions where each tile is retrieved, unlike convolution, where convolution filters glide over the feature map pixel by pixel. A stride of 2 indicates that the max pooling operation for a 2x2 filter size will remove all non-overlapping 2x2 tiles from the feature map.

5) *Fully Connected Layers*

An end of a convolutional neural network is one or more completely connected layers. When every node in the first layer is linked to every node in the second layer, two layers are fully connected [4]. Their task is to do classification using the feature that the convolutions have retrieved. The terminal is typically completely covered with neurons [5]. The SoftMax activation function, which is present in this last fully connected layer, provides an output probability value ranging from 0 to 1 for each categorization label that the model is attempting to predict [5]. The whole construction of a convolution neural network.

B. Model Implementation

As is well known, CNN classifies pictures using deep learning, the most popular kind of machine learning. The best Python package for this is Keras, which makes creating a CNN model quite easy.
Getting the dataset is the initial step. Download the dataset or make one of your own. we first need to import the picture files that make up the 3:1 ratio of training photos to test images to train the model.

1) *Data Analysis*

Let's examine the dataset image, as displayed in Figure 5 below. The "plot" function is required to plot the picture from the dataset, and the "shape" function is required to determine its size.

```
<matplotlib.image.AxesImage at 0x7f987e532358>
```

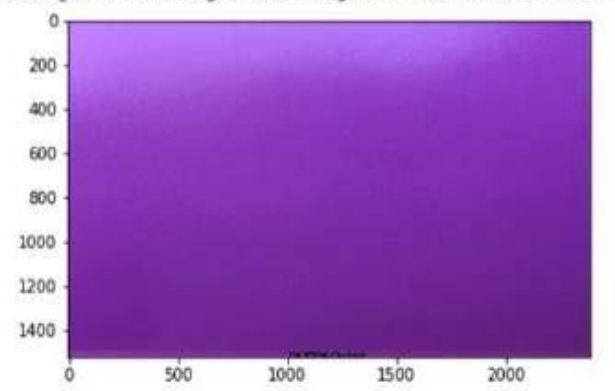
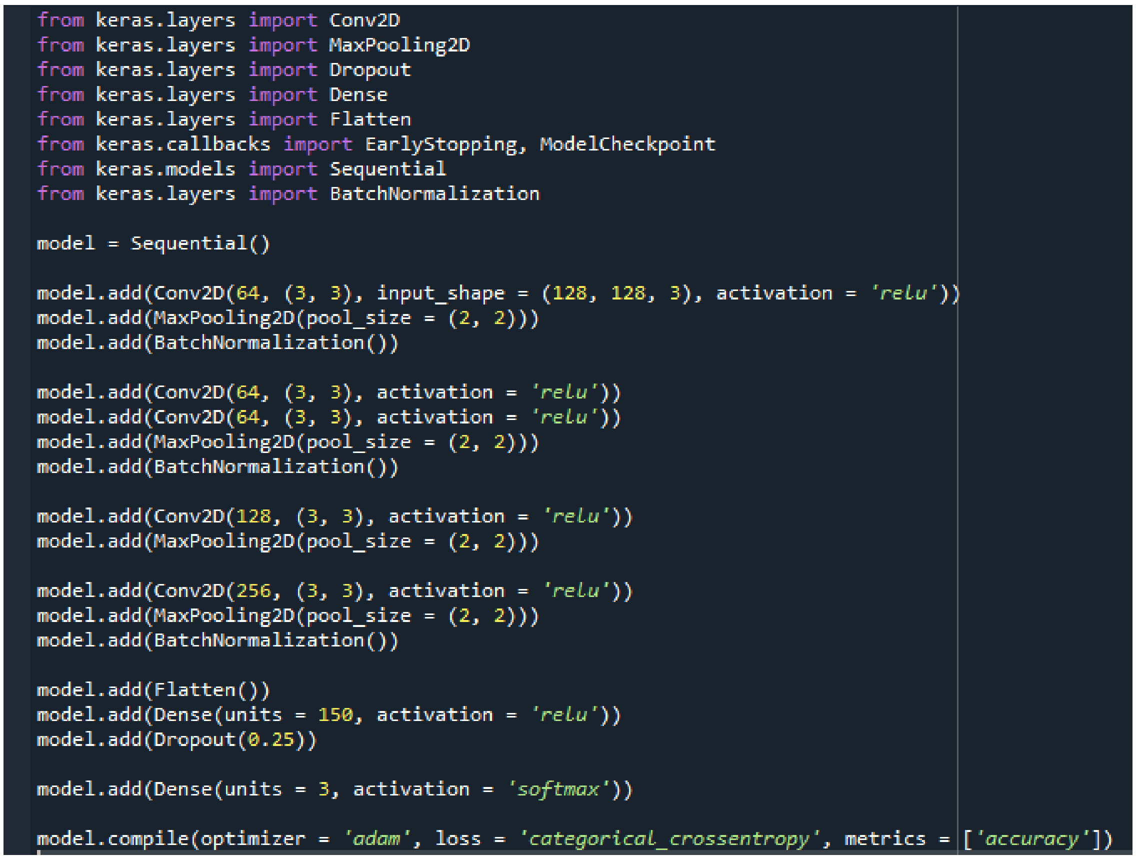


Figure 5: Read image from the dataset

2) *Building the Model*

Once we load our image files, we are ready to build our model. The code is below:



```
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import Dense
from keras.layers import Flatten
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import Sequential
from keras.layers import BatchNormalization

model = Sequential()

model.add(Conv2D(64, (3, 3), input_shape = (128, 128, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(128, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(256, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(units = 150, activation = 'relu'))
model.add(Dropout(0.25))

model.add(Dense(units = 3, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Figure 6: Code

Let's import the Keras library first as we are utilizing it in Python [5]. We are utilizing a sequencing kind of model. The simplest way to create a model in Keras is with a sequential model type. Layer by layer, a model may be built using this Keras package. We are adding model layers one at a time, as you can see in the code above. To add a layer to our CNN model, utilize the add () method.

The first two layers in our CNN model are Conv2D layers. To handle our input photos, Conv2D is a convolution layer [6]. since the 2-D matrix that makes up our input picture is. We require a filter matrix for our convolution, and Kernel determines its size. The Kernel size in this case is 20. We will have a 20x20 filter matrix if the kernel size is 20 [6].

The layer activation function is called "Activation" in this case [6] ReLU, short for Rectified Linear Activation, is the activation function we're utilizing for the top two layers. With neural networks, this ReLU activation function performs well.

An input shape is also sent to the first conv2D layer. Therefore, we are establishing an input shape of 128,128,3, where the 3 denotes that the photos are in color rather than grayscale.

To determine the maximum value in each Kernel patch, a pooling operation may be performed using the MaxPool2D layer. By simply adding Keras' MaxPooling2D or MaxPooling2D layer, we may add a maximum convolution layer to the model [7].

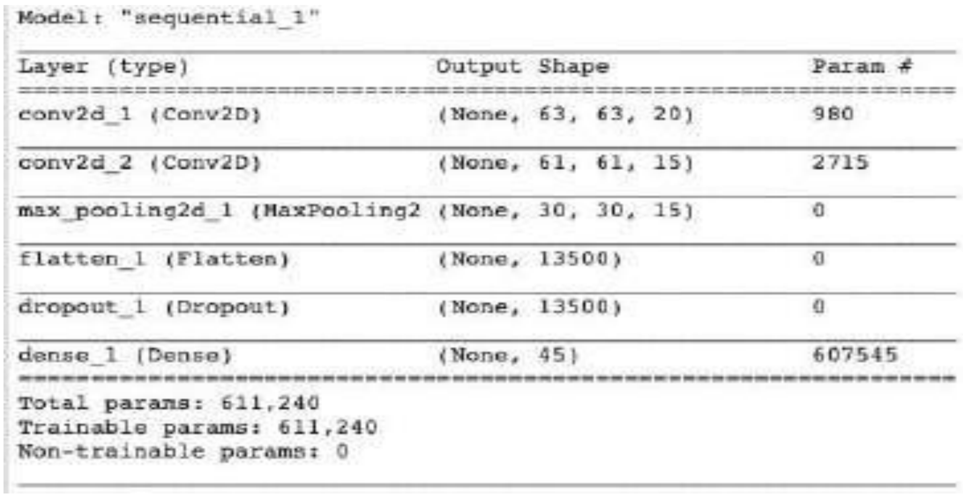
There is a "Flatten" layer sandwiched between the MaxPool2D layers and the Dense layer. This flattened layer lies between the dense layer and the MaxPool or Convolution layer. Between two layers, it acts as a connector [7].

We will utilize the standard layer type "dense" for the output layer because it is frequently used for neural networks [8]. In our output layer, there will be 45 nodes, one for each potential result for each color (0-44). "SoftMax" is the final but most significant activation. The outputs add up to one when the SoftMax is activated in the dense layer [8]. so that we may examine the output's probability. Based on the highest likelihood, the model predicts the color [8].

3) Compiling the Model

We built our model, now we need to compile it. To compile the model, we need three parameters

- **Optimizer**: The optimizer manages the pace of learning (LR). In this instance, we are utilizing the optimizer "adam" [8]. This optimizer modifies the training's learning rate. Adam lr=0.0001 was employed here throughout the training.
  How quickly or slowly the ideal weights for the model are determined depends on the learning rate. Although a slower learning rate results in more accurate weights, the computation of the model's weights will take longer because of the slower learning rate.
- **Loss**: We are using ***'binary_crossentropy'*** for our loss function [8].
- **Metric**: We use the "accuracy" measure to view the accuracy score on the validation set when we train the model [7] to make interpretation simple.
  Let's now summarize our model. We must use the summary () function to translate the model.
  Figure 7 provides a summary of our model.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 63, 63, 20)        980
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 15)        2715
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 15)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 13500)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 13500)             0
_________________________________________________________________
dense_1 (Dense)              (None, 45)                607545
=================================================================
Total params: 611,240
Trainable params: 611,240
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Summary of the model

C. *Model Training and Prediction Training the Model*

We have created and assembled our model thus far. It's time to train the model right now. One of the issues is the necessity for a sizable dataset of image and video files in machine learning and machine vision [8]. Because we probably ran into a circumstance where our system didn't have enough RAM when importing and processing a big number of photos or video data sets [8]. Therefore, to import and analyze pictures in the Keras library, we need to develop

D. *Data Generators*

1) *Image Data Generator Function*

The Image-Data-Generator class is quite helpful in our application for picture categorization. Depending on the approach we wish to take, we may use this data generator in a variety of ways [8]. Here, the flow from the directory method [8] is used. As seen in figure 8, this technique requires a path to the directory containing the photos and the amplification parameters.

We must first import the libraries needed for the data generator. Then we develop an image-augmented data generator.

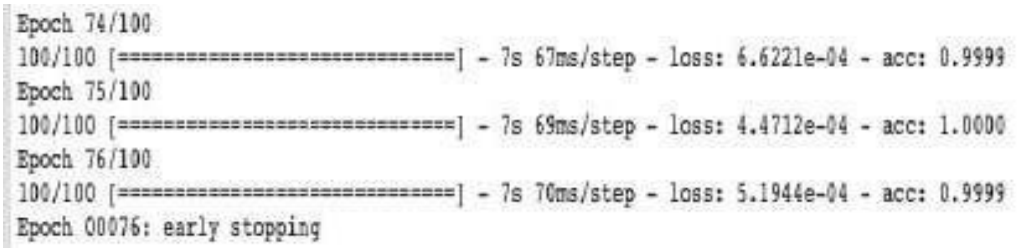
We utilized the ImageDataGenerator class's flow from the directory function.

image-pre-processing library Keras. The flow from the directory method accepts the augmentation of the picture as a parameter [8].

Path, color mode, target size, and batch size are the arguments of the procedure [8]. To indicate the picture's path, use the path parameter. To define the color mode of an input image, use the color mode option. The target size parameter determines the size of the output picture, while the batch size parameter indicates how many photos will be processed in a batch.

The model will cycle through the data according to the number of epochs we provide because each epoch is similar to an iteration [9]. Our model will get better as there are more epochs, but only up to a limit, beyond which it will cease becoming better with each epoch [9]. We are utilizing

"Early Stopping(es)" as a result. As a result, training terminates [9] if there is no discernible progress after a few additional epochs.



```
Epoch 74/100
100/100 [==============================] - 7s 67ms/step - loss: 6.6221e-04 - acc: 0.9999
Epoch 75/100
100/100 [==============================] - 7s 69ms/step - loss: 4.4712e-04 - acc: 1.0000
Epoch 76/100
100/100 [==============================] - 7s 70ms/step - loss: 5.1944e-04 - acc: 0.9999
Epoch 00076: early stopping
```

Figure 8: Early stopping

We defined 100 steps/epoch for 100 epochs in our model. Additionally, we defined min delta=0.0001 and patience=10 in "Early Stopping(es)" [9]. This signifies that training will finish after the last 10 epochs if there isn't a minimum accuracy improvement of 0.0001. We don't want to continue training if there isn't an improvement.

Once the model has been generated, we must utilize the "fit generator ()" function with the trained generator, steps per epoch, and epoch parameters. The constructed model is then fitted using the fit generator technique [10].

We store the learned model using the same method so that we may utilize it for the forecast.

To import all of the photos from a specified path and process them according to the augmentation parameter in batches for image classification applications, use Image-Data-Generator.

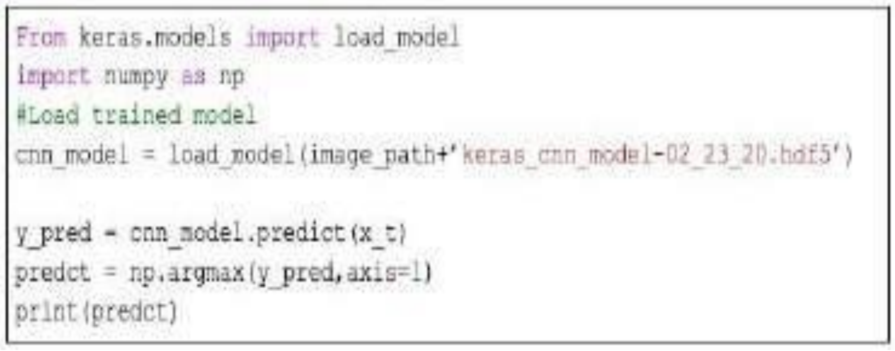
2) *Prediction*

Use the previous load Image () procedure to load the photos now to perform tests or make predictions.

Before submitting original photos to a trained model for prediction, they are required to be scaled. We can resize all test photos to 128X128 using the CV2 computer vision library. The next step is to use the
NumPy library to add each picture to a single array.

3) *Load Trained Model for Prediction*

We employ the load model function of the Image Data-Generator class [10] to load the learned model. A trained model will be loaded from the appropriate place. The trained model is loaded for prediction using the load model method [10], which accepts a route for the model. Then, using the provided trained model, the predict technique is utilized to forecast each test picture [10].



```
From keras.models import load_model
import numpy as np
#Load trained model
cnn_model = load_model(image_path+'keras_cnn_model-02_23_20.hdf5')

y_pred = cnn_model.predict(x_t)
predct = np.argmax(y_pred,axis=1)
print(predct)
```

Figure 9: Load trained model for prediction

4) *Confusion Matrix*

We employed a confusion matrix approach to analyze the performance and prediction outcome of our classification system. Given that there are more than two classification classes, classification accuracy alone may be deceptive if the dataset has an uneven distribution of observations across several classes [10].
Calculating a confusion matrix and graphically visualizing it can help you understand what our classification algorithm is forecasting. The entire number of accurate and inaccurate prediction outcomes is tallied and then divided by class.
The confusion matrix, which demonstrates how our classification model is confused with which class when it makes predictions, is calculated by the Confusion Matrix () method and returned as an array [11]. It provides information about the sorts of errors, rather than just the kinds of mistakes the classifier is making. This array may be printed or plotted with matplotlib pyplot so that we can analyze the findings.

IV. SYSTEM ARCHITECTURE

A. Implementing the Open CV Model

The r, g, and b values are available. We now require a different method that will extract the color name from the RGB value. To choose the color name, we compute a distance (d), which indicates how close we are to selecting the option with the smallest distance.

1) *Working on Colour Detection with OpenCV*

Just lately, I began working on picture color detection using Python, deep learning, and OpenCV. When I discovered [11] OpenCV, which enables the import and manipulation of pictures in Python as illustrated in Fig. 10, I started to consider if and how knowledge may be drawn from these images using machine learning. We've all observed how we do online searches using various filters, one of which is color. I was motivated to create code that will extract colors from photographs and filter out images that support those colors. In this research article, I describe how I learned the basics of OpenCV, used the Naive Bayes technique to extract colors from photos, then selected images from a set of images that supported RGB color values. The repository has the complete notebook available.

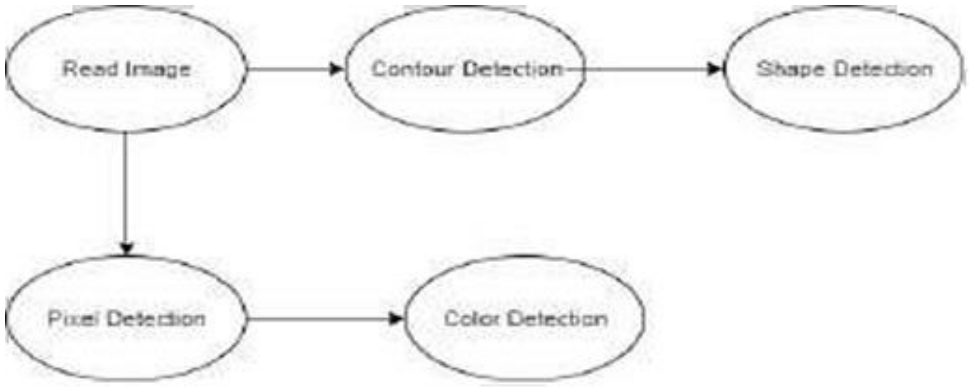


Figure 10: Flow Chart for color detection with OpenCV

2) Naïve Bayes Classification on Dataset

It is a Bayes theorem-based classification-based approach. To put it simply, a Naive Bayes classifier assumes that a certain characteristic is present in Figure 11. The Class has nothing to do with whether a feature is present

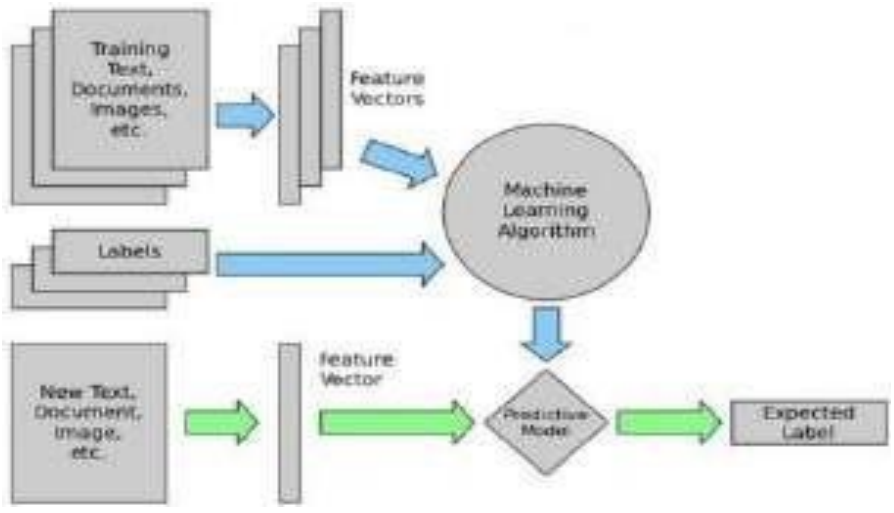

Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Figure 11: Naïve Bayes Classifier Working on Dataset

3) Is Naïve Bayes Efficient for Dataset Predictions?

The most effective algorithm for data mining and machine learning is naive Bayes. Because the conditional independence assumption, on which it is based, is always valid in reality, its competitive classification performance is unexpected.

4) implement Naïve Bayes Classifier?

the prior probability for the specified class labels. Find the chance probabilities calculated using each characteristic. Use the Bayes Formula [12] to determine the loglikelihood of using these values. To use this method, we employ two alternative tools. We use this method with a dataset.

5) The Dataset

The three main RGB values that makeup colors are 0 to 255, with 0 to 255 in each of the three columns. The total values are calculated using the formula [12] 256*256*256 = 16,777,216. These approaches can be represented in about 16.5 million distinct ways. 864 color names are present in our collection. We'll be utilizing a dataset that lists the names of the RGB values with the values themselves.

6) The Scope and Drawbacks of this Research Area

In real-time color sensor detection and image processing, color information is crucial. This has an impact on the video segmentation findings and the accurate real-time temperature value. The dominant color is initially identified using the RGB color space's color information. Choosing a color space is the first stage in the segmentation of a color picture.

The well-known color model [12] includes components like RGB, HSI, HSV, CMYK, CIE, YUV, and others. The most used color model for color processing is the HIS model, followed by the RGB model (Fig. 12). They are frequently used in image processing software.

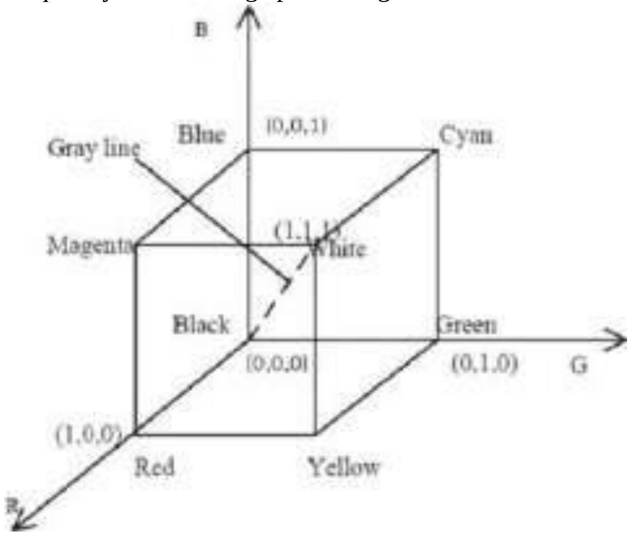
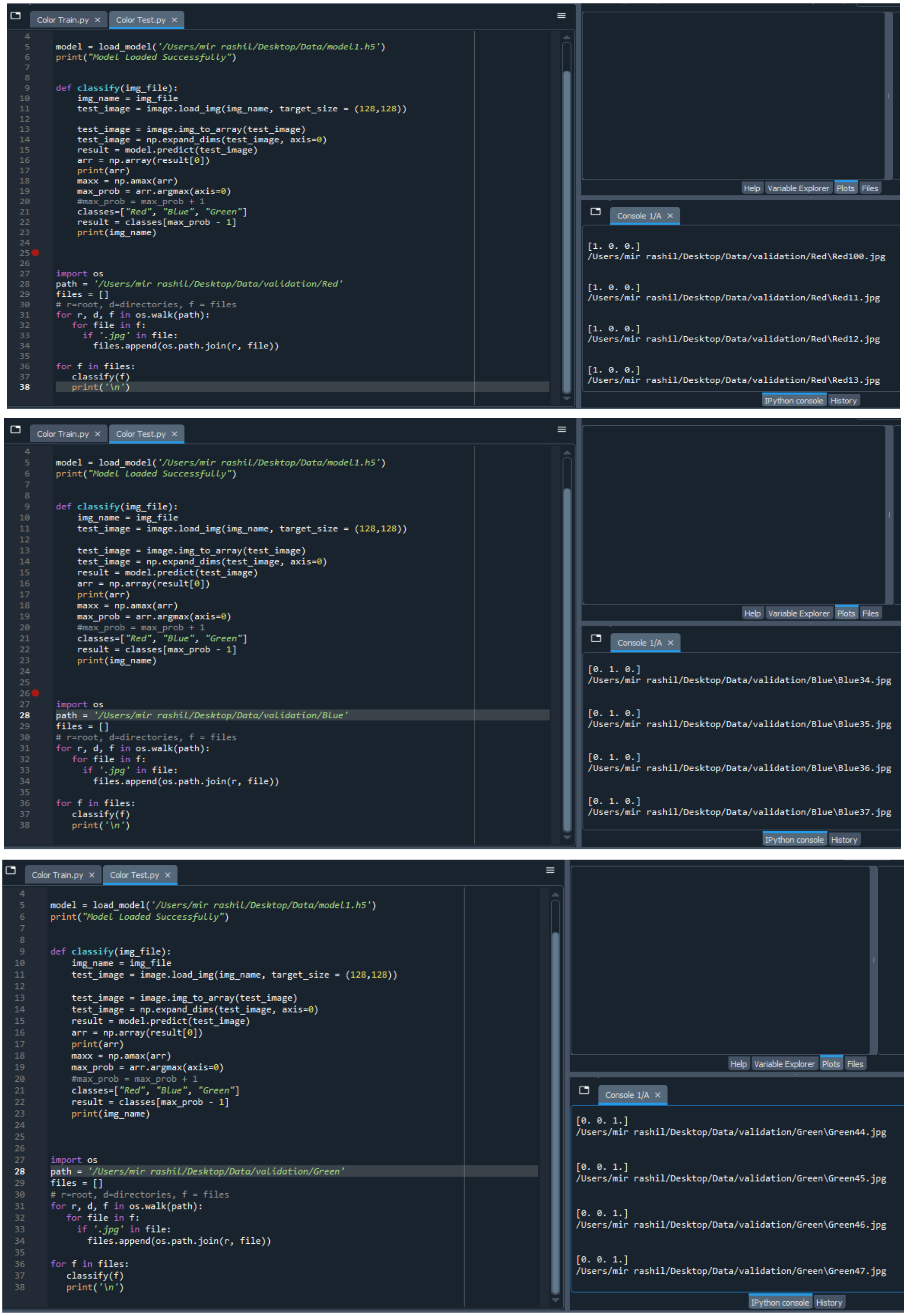


Figure 12: RGB Color Mode

Color detection technology is virtually as bad as picture and video browsing quality [12]. Everybody these days wants the nicest colors in their images and films, but despite the use of several algorithms and approaches, the results are often subpar due to color segmentation. However, to achieve the best results, we use the Python libraries for OpenCV and the Naive Bayes classifier.

V. SIMULATION AND RESULTS

Result analysis on an online dataset using CNN and Open CV algorithm below(Figure 13 and Figure 14) is the result discussion of the CNN and open CV algorithm in detail:



```python

model = load_model('/Users/mir rashil/Desktop/Data/model1.h5')
print("Model Loaded Successfully")


def classify(img_file):
    img_name = img_file
    test_image = image.load_img(img_name, target_size = (128,128))

    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)
    arr = np.array(result[0])
    print(arr)
    maxx = np.amax(arr)
    max_prob = arr.argmax(axis=0)
    #max_prob = max_prob + 1
    classes=["Red", "Blue", "Green"]
    result = classes[max_prob - 1]
    print(img_name)



import os
path = '/Users/mir rashil/Desktop/Data/validation/Red'
files = []
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
   for file in f:
     if '.jpg' in file:
       files.append(os.path.join(r, file))

for f in files:
   classify(f)
   print('\n')
```

```
[1. 0. 0.]
/Users/mir rashil/Desktop/Data/validation/Red\Red100.jpg


[1. 0. 0.]
/Users/mir rashil/Desktop/Data/validation/Red\Red11.jpg


[1. 0. 0.]
/Users/mir rashil/Desktop/Data/validation/Red\Red12.jpg


[1. 0. 0.]
/Users/mir rashil/Desktop/Data/validation/Red\Red13.jpg
```

```python

model = load_model('/Users/mir rashil/Desktop/Data/model1.h5')
print("Model Loaded Successfully")


def classify(img_file):
    img_name = img_file
    test_image = image.load_img(img_name, target_size = (128,128))

    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)
    arr = np.array(result[0])
    print(arr)
    maxx = np.amax(arr)
    max_prob = arr.argmax(axis=0)
    #max_prob = max_prob + 1
    classes=["Red", "Blue", "Green"]
    result = classes[max_prob - 1]
    print(img_name)



import os
path = '/Users/mir rashil/Desktop/Data/validation/Blue'
files = []
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
   for file in f:
     if '.jpg' in file:
       files.append(os.path.join(r, file))

for f in files:
   classify(f)
   print('\n')
```

```
[0. 1. 0.]
/Users/mir rashil/Desktop/Data/validation/Blue\Blue34.jpg


[0. 1. 0.]
/Users/mir rashil/Desktop/Data/validation/Blue\Blue35.jpg


[0. 1. 0.]
/Users/mir rashil/Desktop/Data/validation/Blue\Blue36.jpg


[0. 1. 0.]
/Users/mir rashil/Desktop/Data/validation/Blue\Blue37.jpg
```

```python

model = load_model('/Users/mir rashil/Desktop/Data/model1.h5')
print("Model Loaded Successfully")


def classify(img_file):
    img_name = img_file
    test_image = image.load_img(img_name, target_size = (128,128))

    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)
    arr = np.array(result[0])
    print(arr)
    maxx = np.amax(arr)
    max_prob = arr.argmax(axis=0)
    #max_prob = max_prob + 1
    classes=["Red", "Blue", "Green"]
    result = classes[max_prob - 1]
    print(img_name)



import os
path = '/Users/mir rashil/Desktop/Data/validation/Green'
files = []
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
   for file in f:
     if '.jpg' in file:
       files.append(os.path.join(r, file))

for f in files:
   classify(f)
   print('\n')
```

```
[0. 0. 1.]
/Users/mir rashil/Desktop/Data/validation/Green\Green44.jpg


[0. 0. 1.]
/Users/mir rashil/Desktop/Data/validation/Green\Green45.jpg


[0. 0. 1.]
/Users/mir rashil/Desktop/Data/validation/Green\Green46.jpg


[0. 0. 1.]
/Users/mir rashil/Desktop/Data/validation/Green\Green47.jpg
```

Figure 13: Result using CNN

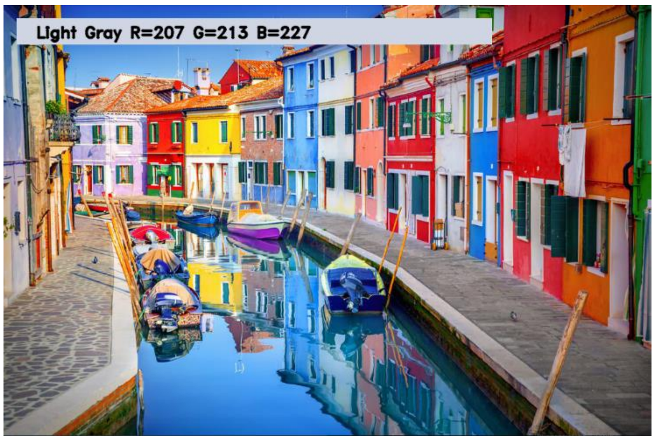


Figure 14: Result using Open CV

VI. CONCLUSION

Using machine vision and convolutional neural networks, we can classify different color hues in a variety of ideal and non-ideal lighting situations, such as bright light, low/dark light, etc. Twilight and sunset light was perceived as low light sources, but natural daylight was recognized as a powerful light source. This natural light source has to be used for this project to provide a less-than-ideal environment for a machine learning model. We trained and assessed the machine learning model to categorize distinct color hues using the dataset. In contrast, as part of the study area and for all research methods, we employed a computer vision model where we learned about colors and how to extract color RGB values and the color name of a pixel.

We learned how to read ARFF files with Weka, CSV files with Pandas, and how to manipulate data using the OpenCV library. Additionally, we learned how to manage operations like double-clicking a window. Several drawing and image-editing tools frequently utilize this. In this case, humans can quickly recognize the hue and make a choice. A computer, however, cannot quickly recognize color. It is challenging to get a computer to quickly detect the color. We use a range of techniques to do this task. Python uses the OpenCV package and the Naive Bayes algorithm. Naive Bayes classifiers are models that assign category labels to issue occurrences that are represented as vectors of feature values, where the category labels are selected from a finite set. The open CV approach has a percent accuracy rate compared to the CNN Model's percent accuracy rate.

REFERENCES

[1] Brennan Gillis, Bob Kenney, Martin Gillis, Mike Wilkinson, Michelle Adams, Nicole Perry, Carla Hill, Rochelle Owen - Waste Management Practices: Literature Review pg:2

[2] Tran Anh Khoa, Cao Hoang Phuc, Pham Duc Lam, Le Mai Bao Nhu, Nguyen Minh Trong, Nguyen Thi Hoang Phuong, Nguyen Van Dung, Nguyen Tan-Y, Hoang Nam Nguyen, and Dang Ngoc Minh Duc - Waste Management System Using IoT-Based Machine Learning in University

[3] Saurabh Jotwani1, Avesh Sheikh2, Prof. Urvashi Agrawal3, Dr. Narendra Bawane4 - To develop a Garbage detection Drone. Journal of Interdisciplinary Cycle Research Research,441

[4] Anjali Pradipbhai Anadkat, B V Monisha, Manasa Puthineedi, Ankit Kumar Patnaik, Dr. Shekhar R, Riyaz Syed - Drone-based Solid Waste Detection using Deep Learning & Image Processing. Alliance International Conference on Artificial Intelligence and Machine Learning (AICAAM), (2019) 362.

[5] Kellow Pardini, Joel J.P.C. Rodrigues, Ousmane Diallo, Ashok Kumar Das, Victor Hugo C. de Albuquerque, and Sergei A. Kozlov - A Smart Waste Management Solution Geared towards Citizens. Sensors (2020), 20, 2380. Pg: 12 of 15

[6] Parkash, Prabu V - IoT Based Waste Management for Smart City. DOI: 10.15680/IJIRCCE.2016. 04020 Pg: 1267

[7] Praveen Kumar Gupta, Vidhya Shree, Lingayya Hiremath, and Sindhu Rajendran - The Use of Modern Technology in Smart Waste Management and Recycling: Artificial Intelligence and Machine Learning

[8] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos and Dionisis Kandris - A Review of Machine Learning and IoT in Smart Transportation. Future Internet 11(4) (2019) 94.

[9] Olugboja Adedeji, Zenghui Wang - Intelligent Waste Classification System Using Deep Learning Convolutional Neural Network

[10] M. A. Viraj J. Muthugala, s. M. Bhagya p. Samarakoon, and Mohan Rajesh Elara - Tradeoff Between Area Coverage and Energy Usage of a Self-Reconfigurable Floor Cleaning Robot Based on User Preference

[11] Subham Chakraborty, Sayan Chowdhury, Soumyadip Das, Sayansri Ghosh, Rimona Dutta, Sayan Roy Chaudhuri - Segregable Smart Moving Trash Bin

[12] Bobulski, J., and Kubanek, M - Waste Classification System Using Image Processing and Convolutional Neural Networks. Springer International. (2019)