

# COLOR CLASSIFICATION USING MACHINE LEARNING

A Project

Presented to the faculty of the Department of Electrical and Electronic Engineering  
California State University, Sacramento

Submitted in partial satisfaction of  
the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Electronic Engineering

by

Mihir Rajendra Mahajan

Spring  
2020

© 2020

Mihir Rajendra Mahajan

ALL RIGHTS RESERVED

# COLOR CLASSIFICATION USING MACHINE LEARNING

A Project

by

Mihir Rajendra Mahajan

Approved by:

\_\_\_\_\_, Committee Chair  
Dr. Fethi Belkhouche

\_\_\_\_\_, Second Reader  
Dr. B. Preetham Kumar

\_\_\_\_\_  
Date

Student: Mihir Rajendra Mahajan

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

\_\_\_\_\_, Graduate Coordinator \_\_\_\_\_  
Dr. B. Preetham Kumar Date

Department of Electrical and Electronic Engineering

Abstract  
of  
COLOR CLASSIFICATION USING MACHINE LEARNING  
by  
Mihir Rajendra Mahajan

Machine learning and artificial intelligence continue to be active research fields focused on real-world problems. Machine learning uses computers to make predictions based on the provided data set or previous experience. Using machine learning methods such as supervised and unsupervised learning, we can process large data and solve classification problems. In this project, we have applied supervised learning, which is the most often used task-driven classification type of machine learning. The main goal of our project is classification of different color shades using machine learning under ideal and different non-ideal conditions. In this project, we used a binary classification technique of supervised learning to classify different colors.

\_\_\_\_\_, Committee Chair  
Dr. Fethi Belkhouche

\_\_\_\_\_  
Date

## ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my advisor Dr. Fethi Belkhouche for providing me the great opportunity to work on this project and for providing continuous guidance throughout the project. This project helped me to gain the knowledge of machine vision and machine learning algorithms and how we can use this in real world to solve the machine vision. And machine learning problems. I , again, thank him for guiding me in the correct direction for timely feedback and his invaluable suggestions to fulfill the project requirements.

Also, I would like to thank Dr. B. Preetham Kumar for his willingness to serve on the committee.

Finally, I would like to thank the entire faculty and staff of the Department of Computer Science Engineering at California State University, Sacramento.

## TABLE OF CONTENTS

	Page
Acknowledgements.....	vi
List of Tables .....	ix
List of Figures .....	x
Chapter	
1. INTRODUCTION .....	1
1.1 Problem and Purpose .....	1
2. BACKGROUND OF THE STUDY .....	2
2.1 Review of Research .....	2
2.2 Machine Learning.....	3
2.2.1 Different Types .....	5
2.2.1.1 Supervised Learning .....	5
2.2.1.2 Unsupervised Learning .....	6
2.2.1.3 Reinforcement Learning .....	7
2.3 Classification .....	8
2.4 Applications .....	9
2.5 Artificial Neural Network.....	10
3. LEARNING .....	12
3.1 CNN Neural Network Model.....	12
3.1.1 Convolution .....	12
3.1.2 Activation function .....	14
3.1.3 ReLU.....	14
3.1.4 Pooling.....	14

3.1.5 Fully connected layer.....	15
3.2 Model Implementation.....	16
3.2.1 Data analysis .....	16
3.2.2 Building the model .....	18
3.2.3 Compiling the model .....	20
3.3 Model Training and Prediction .....	22
3.3.1 Training the model.....	22
3.3.2 ImageDataGenerator function .....	22
3.3.3 Prediction.....	25
3.3.4 Load trained model for prediction .....	26
3.3.5 Confusion matrix .....	26
4. RESULT .....	28
4.1 Model Evaluation .....	28
4.1.1 Ideal Condition .....	28
4.1.2 Different Light Condition .....	29
4.1.2.1 Bright light condition .....	29
4.1.2.2 Low light condition.....	33
5. CONCLUSION AND FUTURE WORK .....	40
References.....	42



## LIST OF TABLES

Tables	Page
1. Confusion Matrix Summary – Bright light condition .....	32
2. Confusion Matrix Summary – Low light condition .....	36

## LIST OF FIGURES

Figures	Page
1. Data flywheel .....	3
2. Data Set .....	4
3. Types of Machine Learning.....	5
4. Supervised Learning Categories .....	6
5. Neural Network Layers.....	11
6. End-to-end structure of a CNN .....	12
7. Convolution Neural Network .....	13
8. Maxpool .....	15
9. Read image from dataset .....	17
10. Image size .....	17
11. Summary of the model .....	21
12. Early stopping .....	24
13. Output of summary function.....	28
14. Ideal olive color .....	28
15. Confusion Matrix – Bright light condition .....	30
16. Confusion Matrix – Low light condition .....	34
17. Orange color in bright-light condition .....	38
18. Light blue color in all the conditions .....	38
19. Orchid color in all the conditions .....	38
20. Dark violet color in all the conditions .....	38

## **Chapter 1**

### **INTRODUCTION**

#### **1.1 Problem and Purpose**

The problem addressed in the project consists of classification of color shades under different ideal and non-ideal light conditions using Machine Vision and Machine Learning algorithms. Examples of non-ideal light conditions include bright light, dark light, etc.

The purpose is to recognize a color from an image captured using a camera and also to experiment with the different light conditions to check the machine learning algorithm's prediction. Along with this experiment, the other purpose is to learn, build and tune the learning model to find a pattern, predict test data and study the difference between the machine's prediction under non-ideal condition, compared with the ideal light condition.

## **Chapter 2**

### **BACKGROUND OF THE STUDY**

#### **2.1 Review of Research**

Machine learning algorithms are sophisticated techniques that can be used to solve the color classification problem. Machine Vision field is moving from traditional statistical methods and algorithms to Neural Network methods [6].

Many challenges exist in Machine Vision. In general, neural network methods are very useful for specific problems compared to traditional algorithms [6]. Color classification using K-Nearest Neighbors Machine Learning algorithm with feature extraction is one traditional ways of color recognition. Whereas feature extraction is an important factor, feature extraction like Color Histogram, Color Correlogram, Color Moments can also be used [17]. Another traditional machine learning algorithm is the KMeans algorithm which can be used to extract colors from images to classify each image from a collection of images based on color space values, where any color space can be used like [17] RGB, CYMK, HSV, etc.

Artificial Neural Network allowed to create computational models that exceed the performance of artificial intelligence with traditional algorithms [1]. One example is Convolutional Neural Networks (CNN), which are used commonly and effectively to solve difficult image-driven pattern recognition problems [1]. Therefore, CNN are used for classification in this project.

## 2.2 Machine Learning

Machine Learning (ML) is one of the active areas of Artificial Intelligence. The objective of ML is to allow the machine to learn. We also have to give the machine some ability to respond to feedback [6]. The main difference between traditional programming and ML is that instead of instructions, we need to input data [17]. Also, instead of a predefined response, the goal of machine learning algorithms is to help the machine learn how to respond.

Data are also important in ML. If we plan to use ML, then it is crucial to obtain high quality & diverse datasets [17]. If the data set is better, then the algorithm and the product will be better. An example is shown below in Figure 1.

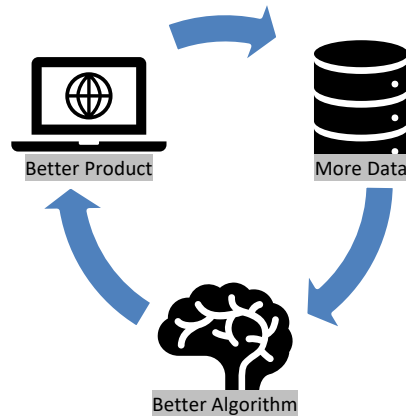


Figure 1: Data Flywheel [20]

In machine learning, we use methods to help our program find patterns in massive datasets.

As shown in figure 2, an original data set divided into a training set and test set, usually in a 70:30 ratio, respectively.

A training set is a smaller set that is used to tune the algorithms. In turn, with these algorithms' help, we can create a model that will work for the larger dataset of the original data [23].

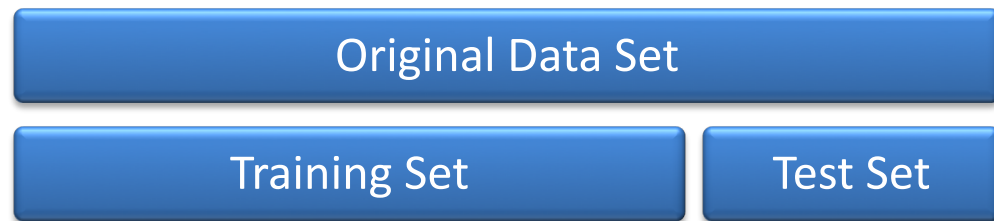


Figure 2: Data set

If we create a simple model, this might work for small training sets, but it is less flexible when we are looking for large data. This is typically called *underfitting* [23]. If you are underfitting your model to the data, it's not capturing enough information, so it makes an inaccurate prediction.

On the other hand, we could create a model that's flexible enough to work with the dataset but so complex and difficult to understand. This is typically called *overfitting* [23].

### 2.1.1 Different types of Machine Learning

There are three ways a machine can learn [6], as shown below in Figure 3.

- 1) Supervised Learning
- 2) Unsupervised Learning
- 3) Reinforcement Learning

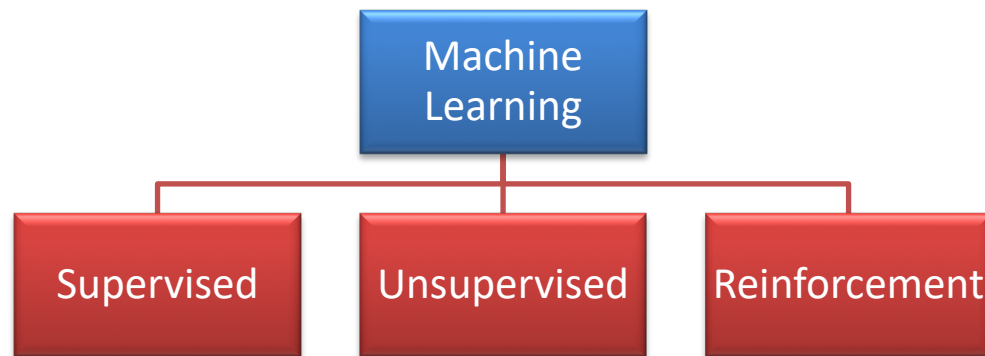


Figure 3: Types of Machine Learning [6]

#### 2.1.1.1 Supervised Learning

When we know enough about the data, we can use supervised learning. In supervised learning, we show the machine the connection between different variables and known outputs. In supervised ML, the variables consist of labeled sample data and known output called correct output. The labeled data is the input, which is the independent variable, and dependent variable would be the output. Supervised learning is task-driven (classification) with three classification types as shown below in Figure 4.

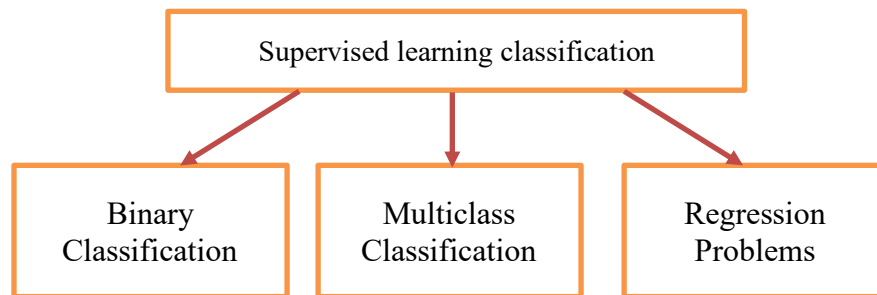


Figure 4: Supervised Learning Categories

#### 2.1.1.2 Unsupervised Learning

Learning and improving by trial and error is the key to unsupervised learning. Unlike supervised learning [19], we are not giving labeled data or showing the correct answer to the machine. Instead, we are using different algorithms to let the machine create connections by studying and observing the data.

Unsupervised learning is a data-driven (clustering) learning algorithm, the machine gets unlabeled data, by studying and observing the data. The machine clusters the data into different groups. The key thing with unsupervised learning is access to a massive amount of data. The more data, the easier it is for a machine to observe and study trends that might lead to a worthwhile cluster.



### **2.1.1.3 Reinforcement**

Reinforcement learning is totally different from supervised and unsupervised learning [17]. Reinforcement learning has the machine iterate to continuously improve the outcome. Overtime, the machine should zero-in and get closer and closer to high quality output [19].

In reinforcement learning the algorithm learns to react to an environment. You are reinforcing certain ways that you want the machine to behave. Instead of just study and observe, we are giving the machine a very clear goal.

Q-learning is a type of reinforcement learning, it is one of the most promising areas in machine learning. A machine can play games or run algorithms and then look at the results of players. If a positive event occurs, then the machine can learn or reinforce the algorithm [22] and keep learning.

## 2.3 Classification

As we saw above, classification is a supervised learning type of machine learning. It decides the class to which the data belong to. This technique is to categorize input data into respective and different number of classes and assign labels to every class [2].

Classification has two types [2]:

- 1) Binomial/Binary
- 2) Multi-Class

Classification can be used in following applications [2]:

- Is received email a spam or ham?
- Verify signature/handwriting recognition [15].
- What is the class of a given image?
- Document classification.
- Medical x-ray labelling [15].
- Face recognition from the photograph and label names [15].
- Classification of types of crops or soil.

## 2.4 Applications

This project deals with color classification using learning methods. Color classification using machine vision and machine learning has many applications [10]. Some applications are listed below.

### Agriculture

- To differentiate between crop and soil when using robotics to identify and cut crops.
- To determine ripe fruits.
- To identify and discard rotten vegetables/fruits using robotic arm.
- To find nutrient deficiency based on leaf yellowing.

### Autonomous vehicles

- Traffic signal detection and recognition [15].

### Road and Safety

- Vehicle color identification in different light conditions

### Carpentry and household services

- Identify wood type based on color
- Determine wall paint color for repainting

### Cosmetology

- Determine skin tone

### Landscape architecture

- For landscape observations from aerial view and classification of land, forest, water areas [10].

## 2.5 Artificial Neural Networks

As we know machine learning algorithms are designed to seek out different patterns in the data. Artificial Neural Networks are an extremely robust way for machines to find the patterns. This means we can classify several thousand images in an instant, or we can translate to different languages and transcribe audio into text files [22]. Because artificial neural networks are a type of machine learning, we still need very large datasets.

We can use supervised learning with a small training set and then have the network find patterns. Then we use these patterns to run against our test data. We can also use unsupervised learning, where we let the neural network find patterns in unlabeled data. Our network might find new clusters that we haven't considered. This works particularly well for larger data sets [22].

There are a couple of things we want to keep in mind with artificial neural networks. The first is same as machine learning [22], we're going to rely on a massive amount of high-quality data. That's how artificial neural networks find new patterns. If we don't supply the data, then our artificial neural network will not have the opportunity to learn new things [22]. The second thing to keep in mind is that when [22] we're working on an artificial neural network, we're going to be using an empirical approach. That means that we'll run small experiments to try and fine-tune our network to get better results. The modern artificial neural network gives us plenty of opportunities to change our configurations. These are called the networks' hyper-parameters.

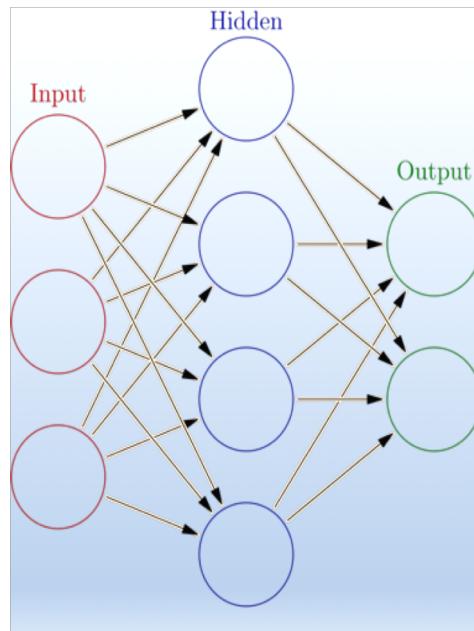


Figure 5: Neural Network Layers [14]

Artificial Neural Networks use much of the same language as neuroscience. Our brain is filled with neurons. We learn new things when the cells form networks of connections. In a sense, within our brain, there's a network of connections that help us identify and classify new things. An artificial neural network also uses neurons, except these neurons have a numerical value as a way to hold information [5]. Much like our biological brain, an artificial neural network gets a lot of its power from the connections between different neurons [5]. An artificial neural network organizes these neurons into layers [5]. There are input layers, several hidden layers and an output layer as shown in figure 5. A neural network can have many different hidden layers [5]. In fact, more layers we have the more processing we can do to find a complex pattern. This artificial neural network often called deep learning. It can learn many new things because we have an artificial neural network that is several layers deep.

## Chapter 3

### LEARNING

#### 3.1 CNN Neural Network Model

The structure of a convolutional neural network is shown below in Figure 6.

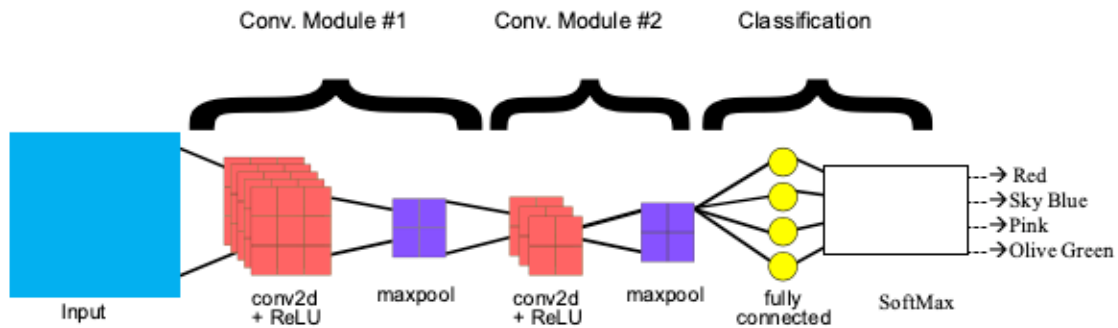


Figure 6: End-to-end structure of a convolution neural network [18].

Fully connected neural networks typically don't work well on images. This is because each pixel is an input, as we add more layers, the amount of parameters increases exponentially [19]. What makes one image distinguishable from another is its special structure [19]. Areas close to each other or vicinity areas are highly significant in images. CNN could be used to extract a higher- and higher-level representation of image contents.

##### 3.1.1 Convolution

At first, CNN gets input feature map which is a three-dimensional matrix where the size of the first two dimensions corresponds to the length and width of the images in pixels. The size of the third dimension is 3 (RGB color image) [18].

In CNN, a convolution extracts one tile (3x3 or 5x5 pixels) of the input feature map and applies filters (same size as the tile) over them and produces an output feature map or convolved feature.

In this convolution step, as shown below in Figure 7, the filters effectively slide over the input feature map's grid left to right and top to bottom, one pixel at a time [15], extracting each respective tile [18].

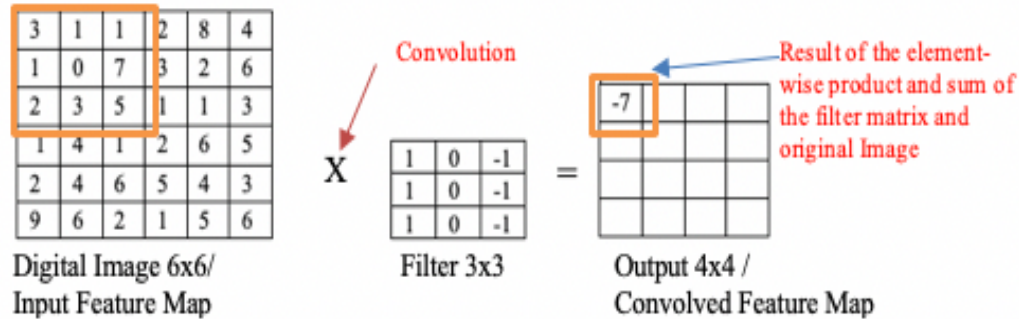


Figure 7: Convolution Neural Network [18]

For each filter-tile, the CNN does the element-wise multiplication of the filter matrix and the tile matrix, and then add all the elements of the resulting matrix to get a single value. This operation is similar to the dot product. Each of resulting values from this dot product for every filter-tile pair is then output in the convolved feature matrix.

During training, the CNN “learns” the optimal values for the filter matrices that enable it to extract meaningful features from the input feature map [18]. As the number of filters applied to the input feature map increases, so does the number of features the CNN can extract but training time also increases as more filters are added on CNN. Also, every new filter added to the network provides less incremental value than the previous one. Thus, we need to construct a network that uses the minimum number of filters needed to extract the features necessary for accurate image classification [18].

### 3.1.2 Activation functions

An activation function is a component of the neural network. The activation function decides if a neuron fires or not. To ensure that there is some nonlinearity in our network, we need to make sure that these activation functions are nonlinear [19]. We can use a step function as our activation function, which provides an output of zero or one. If the output is above a certain threshold, then neuron is fired, and we have one. If the value of the output is less than the threshold, then it is not fired, and we have a zero [19].

### 3.1.3 ReLU

The ReLU is one of the most well-known Activation functions. The ReLU or “Rectified Linear Unit” output, a zero for any value of  $x$  that is less than zero. For any value of  $x$  equal to or greater than zero, the function returns  $x$ . After each convolution operation of CNN, the network applies a Rectified Linear Unit (ReLU) transformation to the convolved feature, in order to introduce nonlinearity into the model.

### 3.1.4 Pooling

Pooling is the next step after ReLU, in which the CNN reduces the sampling of the convolved feature, and the number of dimensions of feature map, while still preserving the most critical feature information. This process is called max pooling, as shown in Figure 8 below, and it is one of the most common algorithms.

There are other pools as well, like the average pool and min pool. Max pooling works in the same manner as convolution. It slides over the feature map and extracts tiles of a specified size. Then from each extracted tile, the maximum value out of that tile is outputted to a new feature map, and



all other values are discarded. Max pooling operations take two parameters [18]. The size of max-pooling filter is typically 2x2 pixels.

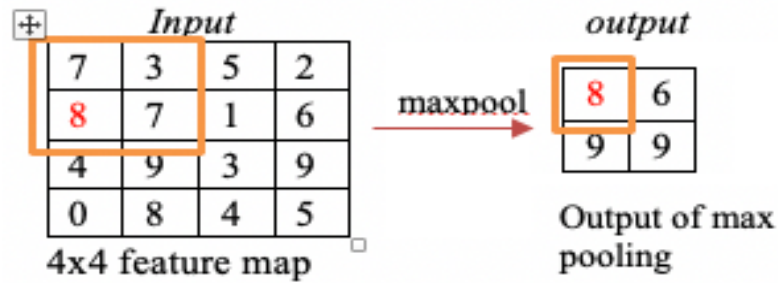


Figure 8: maxpool [18]

The distance, in pixel, separating each extracted tile is called stride. It's different from convolution, wherein convolution filters slide over the feature map pixel by pixel, but in max pooling, the stride determines the locations where each tile is extracted. For a 2x2 filter size, a stride of 2 specifies that the max pooling operation will extract all non-overlapping 2x2 tiles from the feature map.

### 3.1.5 Fully Connected Layers

One or more fully connected layers are at the end of a convolutional neural network. Two layers are fully connected when every node in the first layer is connected to every node in the second layer [18]. Their job is to perform classification based on the feature extracted by the convolutions. Mostly, the end is fully connected with neurons [18]. This final fully connected layer contains a SoftMax activation function, which gives an output probability value from 0 to 1 for each of the classification labels the model is trying to predict [18]. Figure 8 illustrates the end-to-end structure of a convolution neural network.

## 3.2 Model Implementation

As we know, deep learning is the most common type of machine learning used to classify images using CNN. For this, the best library in Python is Keras, which makes it pretty simple to build a CNN model.

The first step is to obtain the dataset. Either download the dataset or create your own dataset. We need 3:1 proportion of training image and test images respectively, and then we need to load those image files to train the model.

Here is the definition to load image files:

```
import os
def loadImages(path, folder):
    '''Put files into lists and return them as one list with all
    images in the folder'''
    image_files = sorted([os.path.join(path, folder, file)
                          for file in os.listdir(path+folder+'')
                          if (file.endswith('.jpg') or file.endswith('.jpeg'))])
    return image_files
```

### 3.2.1 Data analysis

Let's check the image from the dataset, as shown below in Figure 9. To plot the image from dataset we need 'plot' function, and to check its size we need to use 'shape' function.

```
Import matplotlib.pyplot as plt
folder_path = "drive/My Drive/" #folder path

testimage = loadImages(folder_path, 'Test_Low_Light')
img = imread(testimage[33]) #read image number 33 from dataset
#plot the first image
plt.imshow(img)
```

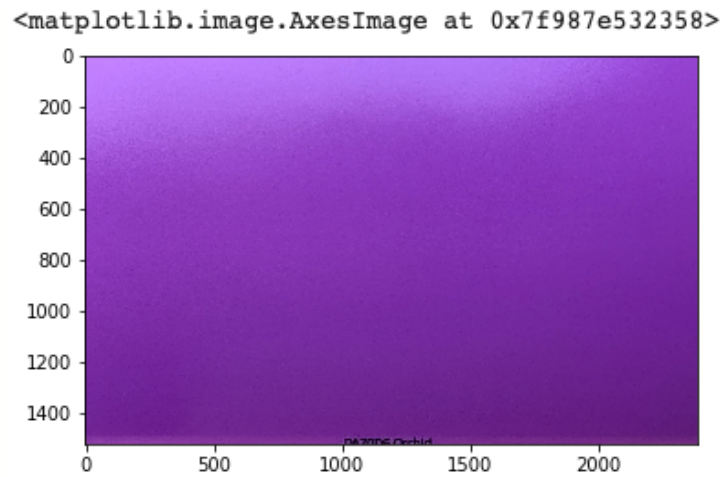


Figure 9: Read image from dataset

Let's check the size of image above by using *shape* function.

```
#check shape of our image  
testimage[0].shape
```

Figure 10 below is the 2D shape of the above image, with the 3 signifying that the image is RGB

```
(1510, 2361, 3)
```

Figure 10: Image size

### 3.2.2 Building the model

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

```
#Import Libraries
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, BatchNormalization, Conv2D
, MaxPool2D, Flatten
from keras import optimizers
from keras.callbacks import EarlyStopping, ModelCheckpoint

#Create model
cnn_model=Sequential()

#add model layers
cnn_model.add(Conv2D(input_shape=(128, 128, 3), filters=20, kernel_size=4,
                        strides=2, padding='valid',activation='relu',data_format='channel
                        s_last'))

cnn_model.add(Conv2D(filters=15, kernel_size=3, strides=1, padding='valid',
                        activation='relu',data_format='channels_last'))

cnn_model.add(MaxPool2D(pool_size=2, data_format='channels_last'))
cnn_model.add(Flatten())
cnn_model.add(Dropout(0.2))
cnn_model.add(Dense(45, activation='softmax'))

cnn_model.compile(optimizer=optimizers.Adam(lr=0.0001),loss='binary_crossentropy',
                  metrics=['accuracy'])
#get summary of created model
cnn_model.summary()
```

We are using Keras library in python so let's import that first [13]. The model type we are using is sequential type. Sequential model type is the easiest method to build a model in Keras. Using this *Keras* library we can build a model layer by layer. You can see above in the code that we are adding model layers one by one. The *add()* function is used to add layer to our CNN model.

In our CNN model, the first two layers are **Conv2D** layers. **Conv2D** is a convolution layer to deal with our input images [18]. Because our input image is a 2-Dimensional matrix.

For our convolution we need a filter matrix, the size of filter matrix is defined by Kernel. So, here the **Kernel\_size** is 20. A Kernel size of 20 means we will have 20x20 filter matrix [13].

Here **Activation** is the activation function for the layers [18]. For the first two layers we are using the activation function ReLU or Rectified Liner Activation. This ReLU activation function works well with neural networks.

The first conv2D layer also takes in an input shape. So, we are defining an input shape of each input image 128,128,3, where the 3 means that the images are color not grayscale.

**MaxPool2D** layer allows to perform a pooling operation to calculate the maximum value in each Kernel patch. We can add maximum pooling operation to the model by just adding **MaxPooling2D** or **MaxPool2D** layer provided by **Keras** [13].

In between the **MaxPool2D** layers and the **Dense** layer, there is a '**Flatten**' layer. This flatten layer is a connection between the MaxPool or Convolution layer and dense layer. It serves as a connection between two layers [18].

'**Dense**' is a standard layer type used in many cases for neural networks, which we will use for the output layer [3]. We will have 45 nodes in our output layer, each possible outcome for each color (0-44). The last but most important activation is '**softmax**'. Softmax activation in the dense layer makes the outputs sum up to 1 [3]. So, that we can check the probabilities for the output. The model predicts the color based on the highest probability [3].

### 3.2.3 Compiling the model

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

```
Cnn_model.compile(optimizer=optimizers.Adam(lr=0.0001),  
                  loss='binary_crossentropy', metrics=['accuracy'])
```

- 1) **Optimizer:** The optimizer controls the learning rate(**lr**). In our case, we are using '**adam**' as an optimizer [3]. This optimizer adjusts the learning rate of training. Here we used **adam** **lr=0.0001** throughout the training.

The learning rate decides how fast or slow the optimal weights for the model are calculated. Slow optimal weight calculation by smaller learning rate leads to more accurate weights, but as it's slow the time to compute the weights for the model will be longer.

- 2) **Loss:** We are using '**binary\_crossentropy**' for our loss function [3].
- 3) **Metric:** To interpret easily, we are using the '**accuracy**' metric to see the accuracy score on the validation set when we train the model [21].

Now let's get a summary of our model. To create a summary of the model we need to call a **summary()** function which is shown below.

```
#get summary of build model  
cnn_model.summary()
```

Summary of our model is shown in figure 11.

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 (MaxPooling2D)	(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 11: Summary of the model

### 3.3 Model Training and Prediction

#### 3.3.1 Training the model

So far, we built and compiled our model. Now it's time to train the model. In the field of machine learning and machine vision we need a large dataset of image and video files, and that's one of the challenges [7]. Because while loading and processing a large number of images or video data set, we probably encountered a situation of not having enough memory in our machine [7]. So, for loading and processing images in Keras library, we need to build *Data Generators*.

#### 3.3.2 ImageDataGenerator function

In our application of image classification, the *ImageDataGenerator* class is very useful. We can use this data generator in multiple ways, depending on the method we want to use [7]. Here we are using *flow\_from\_directory* method [13]. This method takes a path to the directory which contains the images and the augmentation parameters as shown in figure 12.

First, we need to import the libraries required to for data generator. Then we create a data generator with image augmentation.



```

From keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    # This is the target directory
    image_path, color_mode='rgb',
    # All images will be resized to 128x128
    target_size=(128,128), batch_size=15,
    #Since we use binary_crossentropy loss,we need binary labels
)

# simple early stopping
es = [EarlyStopping(monitor='loss', mode='auto', min_delta=0.0001,
    patience=10, verbose=1)]

test_datagen = ImageDataGenerator(rescale=1./255)

```

We used the *flow\_from\_directory* method from the *ImageDataGenerator* class of *keras.preprocessing.image* library. The augmentation of the image is provided as an argument to the *flow\_from\_directory* method [7].

The parameters of the method are as follows: **path**, **color\_mode**, **target\_size**, **batch\_size** [13].

The *path* argument is needed to specify the path of the image, *color\_mode* is for specifying the color mode of an input image. The *target\_size* parameter is to set the size of the output image, and the *batch\_size* parameter is to specify the number of images per batch going to be processed.

The epoch is like an iteration, so the number of epochs we define is the number of times the model will cycle through the data [3]. The greater the number of epochs, the more our model will improve, but up to a certain point only, then the model will stop improving during each epoch [7]. That's why we are using '*EarlyStopping(es)*.' So that training stops if there is no significant improvement in few new epochs [7], as shown below in Figure 12.

```

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 12: Early stopping

In our model we defined *100 steps/epoch* for *100 epochs*. Also, in '*EarlyStopping(es)*' we defined *min\_delta=0.0001* and *patience=10* [13]. This means in the last 10 epochs, if there is no minimum improvement of 0.0001 in accuracy then training will stop, as we don't want to further spend time on training if there is no improvement.

```

history = cnn_model.fit_generator(train_generator,
                                steps_per_epoch=100, epochs=100, verbose=1, callbacks=es)

# Give name to model here. Model will be saved on given path.
Cnn_model.save(image_path+'keras_cnn_model-02_23_20.hdf5')
train_generator.class_indices

```

Once the model is created, we need to use the '*fit\_generator()*' method with the following parameters: *train\_generator*, *steps\_per\_epoch*, *epochs*. Then run the *fit\_generator* method to fit the created model [7].

Using the *save* method we save the trained model, so that we can use this model for prediction.

That's why *ImageDataGenerator* is an easy way to load all images from the specific path and process them as per the augmentation parameter in batches for image classification application.

### 3.3.3 Prediction

Now, let's load the images using the previous *loadImage()* function for tests or prediction.

```
Testimage = loadImages(folder_path, 'Test_Bright_Light') # give test
image folder name.
```

Original images needed to be resized before giving them to a trained model for prediction. Using the CV2 computer vision library we can resize all test images to 128X128. Then we need to append all images to one array using the NumPy library.

```
import numpy as np
import cv2

x_cv = testimage
x_t = []
images = []
for x in x_cv:
    img = imread(x)
    img = img[:,:,:3]
    images.append(img)
    #scale_percent = 60 # percent of original size
    width = 128
    height = 128
    dim = (width, height)
    # resize image
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
    x_t.append(resized)
x_t = np.array(x_t)
x_t = x_t/255
```

### 3.3.4 Load Trained Model for Prediction

To load the trained model, we use the *load\_model* method of the *ImageDataGenerator* class [7]. This will load a trained model from the respected location. This *load\_model* method takes a path for the model and loads the trained model for prediction [7]. Then *predict* method is used to predict each test image using the given trained model [7].

```
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)
```

### 3.3.5 Confusion Matrix

To summarize the performance and prediction result of our classification algorithm, we used a confusion matrix technique. As we have more than two classification classes, classification accuracy alone can be misleading if we have an unequal number of observations in many classes in the dataset [11].

To get a better idea of what our classification model is predicting, calculating a confusion matrix and representing it visually is a good option. The total number of correct and incorrect prediction results is summarized and then broken down by class.

The *Confusion\_Matrix()* function will calculate the confusion matrix and return the result as an array, which shows how our classification model is confused with which class when it makes predictions [11]. It gives insight not only on what errors are being made by the classifier but more specifically the types of errors. We can print this array or plot it using *matplotlib.pyplot* and interpret the results.

```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FormatStrFormatter
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import svm, datasets

# Plot a confusion matrix.
# cm is the confusion matrix, names are the names of the classes.
Def plot_confusion_matrix(cm, names, title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(names))
    plt.xticks(tick_marks, names, rotation=90)
    plt.yticks(tick_marks, names,)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm = confusion_matrix(y_true, predict)

print('Plot of Confusion Matrix')
plt.figure(figsize=(15,10))
plot_confusion_matrix(cm,true_lables)
plt.grid(which='both')
plt.show()
print(classification_report(y_true, predict))
```

## Chapter 4

### RESULTS

#### 4.1 Model Evaluation

This is the summary of the implemented model. As we are using a sequential type of model, summary title says that the model is *sequential\_1*. We can get this model summary, as shown in Figure 13 below, by using *summary()* function at the end of the 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 (MaxPooling2D)	(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 13: Output of summary function

##### 4.1.1 Ideal Condition

We provide a set of real color images, as shown in Figure 14 below, to the machine to predict colors. For that, we have created a set of color images using color code in the paint tool and saved them as a .jpg file. Later, we fed the images to a machine to learn and predict under ideal condition.



Figure 14: Ideal olive color

#### **4.1.2 Different Light Condition**

Here we are creating two different (non- ideal) light conditions to test what the algorithm predicts from its previous learning. We will check the result/prediction when there are different light conditions, and what difference is predicted when we compared with the actual color (ideal) condition of the same color. Also, we are using our machine learnings methods to differentiate two cases. For that, we are using the confusion matrix method to compare the result of different light conditions with ideal light conditions.

We are creating two different light conditions as follows:

- 1) Bright-light condition,
- 2) Low-light condition.

##### **4.1.2.1 Bright light condition**

Bright light condition is when a color (colored image here) is exposed to bright light, and the image is captured and provided to the algorithm to predict the color based on the previous learning based on ideal condition.

In our test, we are considering morning light as a bright light scenario. We exposed 45 ideal color shade images to this bright light. After that, we again captured all 45 images, which are now under bright light condition.

Now we have our model trained under ideal condition. We fed the 45 captured images as an input to the trained model. Here are the predicted results shown in the confusion matrix in figure 15.

Figure 15: Confusion Matrix – Bright light condition

In this confusion matrix, we can see that the algorithm confused between the ideal condition and bright light condition images. We are expecting that different light conditions will change the prediction. The overall accuracy report is shown below.



**Classification report:**

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	1
2	1.00	1.00	1.00	1
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	1
5	0.33	1.00	0.50	1
6	0.50	1.00	0.67	1
7	0.00	0.00	0.00	1
8	0.50	1.00	0.67	1
9	0.50	1.00	0.67	1
10	0.00	0.00	0.00	1
11	0.00	0.00	0.00	1
12	0.00	0.00	0.00	1
13	1.00	1.00	1.00	1
14	0.25	1.00	0.40	1
15	1.00	1.00	1.00	1
16	0.50	1.00	0.67	1
17	0.00	0.00	0.00	1
18	1.00	1.00	1.00	1
19	1.00	1.00	1.00	1
20	0.50	1.00	0.67	1
21	0.33	1.00	0.50	1
22	1.00	1.00	1.00	1
23	1.00	1.00	1.00	1
24	0.25	1.00	0.40	1
25	1.00	1.00	1.00	1
26	0.33	1.00	0.50	1
27	0.00	0.00	0.00	1
28	1.00	1.00	1.00	1
29	0.00	0.00	0.00	1
30	0.00	0.00	0.00	1
31	0.00	0.00	0.00	1
32	0.00	0.00	0.00	1
33	0.00	0.00	0.00	1
34	1.00	1.00	1.00	1
35	0.00	0.00	0.00	1
36	0.00	0.00	0.00	1
37	0.00	0.00	0.00	1
38	1.00	1.00	1.00	1
39	0.50	1.00	0.67	1
40	0.00	0.00	0.00	1
41	0.00	0.00	0.00	1
42	0.00	0.00	0.00	1
43	0.00	0.00	0.00	1
44	0.00	0.00	0.00	1
accuracy			0.53	45
macro avg	0.39	0.53	0.43	45
weighted avg	0.39	0.53	0.43	45

The above classification report gives accuracy in percent. Here the accuracy is 53%. It means 47% of the time, predicated wrong under bright light conditions. You can look at the confusion matrix to see which color is predicted wrong. This is also show in the summary Table 1 below.

### Confusion matrix summary













































Sr. no	Actual color	Predicted color	Sr. no	Actual color	Predicted color
1	 Black	 Gray	13	 Snow	 Light Gray
2	 Blue	 Dark Blue	14	 White	 Light Gray
3	 Dark Green	 Lime Green	15	 Olive	 Dark Olive Green
4	 Green Yellow	 Lime Green	16	 Orange	 Tomato
5	 Dark Orange	 Chocolate	17	 Orchid	 Dark Violet
6	 Orange Red	 Chocolate	18	 Violet	 Dark Violet
7	 Dark Red	 Fire Brick	19	 Purple	 Indigo
8	 Maroon	 Fire Brick	20	 Royal Blue	 Sky Blue
9	 Red	 Fire Brick	21	 Tomato	 Indian Red
10	 Dark Violet	 Indigo	22	 Yellow	 Dark Khaki
11	 Purple	 Indigo			
12	 Linen	 Light Gray			

Table 1: Confusion Matrix Summary – Bright light condition

This summary gives the overall comparison with the actual color and what the algorithm predicts the given bright light condition image. Let's discuss a few examples from Table 1. It looks like the algorithm predicted gray color instead of black, and lime green instead of dark green.

Also, we can observe that the algorithm made other mistakes. The algorithm predicted the same color for three different colors of the same shade. Linen, snow, white color from white shade all got predicted as a light gray color. Orchid and violet got predicted as dark violet. One more example of machine's confusion is, for dark violet and purple, the machine predicted indigo color.

#### **4.1.2.2 Low light condition**

Low light condition is when a color (colored image here) is exposed to Low light/Low-intensity light, and machine vision captures this and provides it to the algorithm to predict the color based on its previous learning based on ideal conditions.

So, in our test, we are considering indoor evening light as a low light scenario. Considering evening 5 pm indoor light as a low light condition, we exposed 45 Ideal color shade images to this low light. After that, we again captured all 45 images, which are now under a low light condition.

We have our model trained under ideal conditions. We fed this 45 captured images as an input. The predicted results are shown in the confusion matrix in figure 16.

Figure 16: Confusion Matrix – Low light condition

In this confusion matrix, we can see that the machine confused with the ideal condition and low light condition images. In low light, data looks scattered more than the previous non-ideal condition. It means the accuracy is less here. Let's see the overall accuracy report, which is shown below.

**Classification report:**

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1
1	0.50	1.00	0.67	1
2	0.00	0.00	0.00	1
3	1.00	1.00	1.00	1
4	0.00	0.00	0.00	1
5	0.00	0.00	0.00	1
6	0.50	1.00	0.67	1
7	0.00	0.00	0.00	1
8	0.00	0.00	0.00	1
9	0.33	1.00	0.50	1
10	0.00	0.00	0.00	1
11	0.00	0.00	0.00	1
12	0.00	0.00	0.00	1
13	0.00	0.00	0.00	1
14	0.00	0.00	0.00	1
15	0.00	0.00	0.00	1
16	0.00	0.00	0.00	1
17	0.00	0.00	0.00	1
18	0.00	0.00	0.00	1
19	0.00	0.00	0.00	1
20	0.33	1.00	0.50	1
21	0.33	1.00	0.50	1
22	0.00	0.00	0.00	1
23	0.00	0.00	0.00	1
24	0.00	0.00	0.00	1
25	0.33	1.00	0.50	1
26	0.33	1.00	0.50	1
27	0.00	0.00	0.00	1
28	0.00	0.00	0.00	1
29	0.33	1.00	0.50	1
30	0.00	0.00	0.00	1
31	0.00	0.00	0.00	1
32	0.00	0.00	0.00	1
33	0.00	0.00	0.00	1
34	0.00	0.00	0.00	1
35	0.00	0.00	0.00	1
36	0.00	0.00	0.00	1
37	0.00	0.00	0.00	1
38	0.00	0.00	0.00	1
39	0.33	1.00	0.50	1
40	0.00	0.00	0.00	1
41	0.00	0.00	0.00	1
42	0.00	0.00	0.00	1
43	0.00	0.00	0.00	1
44	0.00	0.00	0.00	1
accuracy			0.22	45
macro avg	0.10	0.22	0.13	45
weighted avg	0.10	0.22	0.13	45

The above classification report gives the accuracy in percent. Here the accuracy is 22%. It means 88% of the time, predicated wrong under low light conditions. You can look at the confusion matrix to see which color is predicted wrong. This is also show in the summary Table 2 below.

#### Confusion matrix summary

Sr. no	Actual color	Predicted color	Sr. no	Actual color	Predicted color
1	 Antique White	 Light Gray	18	 Deep Pink	 Purple
2	 Royal Blue	 Light Gray	19	 Dark Violet	 Indigo
3	 AquaMarine	 Aqua	20	 Purple	 Indigo
4	 Blue	 Dark Blue	21	 Fire Brick	 Indian Red
5	 Chocolate	 Saddle Brown	22	 Tomato	 Indian Red
6	 Orange Red	 Saddle Brown	23	 Yellow	 Dark Khaki
7	 Dark Green	 Lime Green	24	 Gold	 Dark Khaki
8	 Green Yellow	 Lime Green	25	 Gay	 Sky Blue
9	 Dark Khaki	 Light Green	26	 Light Blue	 Sky Blue
10	 Khaki	 Light Green	27	 Hot Pink	 Orchid
11	 Green	 Dark Olive Green	28	 Linen	 Light Blue
12	 Olive	 Dark Olive Green	29	 Light Gray	 Light Blue
13	 Dark Orange	 Chocolate	30	 Orchid	 Dark Violet
14	 Dark Red	 Maroon	31	 Violet	 Dark Violet
15	 Saddle Brown	 Maroon	32	 Magenta	 Dark Violet
16	 Red	 Fire Brick	33	 Snow	 Royal Blue
17	 Orange	 Fire Brick	34	 White	 Royal Blue

Table 2: Confusion Matrix Summary – Low light condition

This summary above gives the overall comparison with the actual/ideal color and what the machine predicts with the given low light condition. It looks like the machine got confused with multiple colors. Let's discuss a few examples from Table 2. From this table, most confused colors are bright. For example, light blue was predicted as sky blue. Then for light gray and linen, the machine is predicting light blue. The olive color is getting a predicted dark olive color.

Also, the algorithm predicted the same color for three different colors of the same shade. Orchid, violet, and magenta were predicted as dark violet. One more good example of machine confusion is for red, and orange, the algorithm predicted fire brick color for both of them. It seems that the algorithm is predicting darker shade than the actual color in low light conditions.

### **Overall observation**

Many times, we easily recognize the color by just seeing it. Sometimes we can guess the color shade family but can't easily tell which exact color it is. So, even for us to remember or to guess, we have a lookup table in our mind, which helps us to identify the color. So, it is the same with ML, we provide the machine with a lookup table for all color and will let the machine learn. Next time when a trained machine gets the input images, it uses the lookup table.

So, here we can see a few examples to visualize—figures 17-20 show orange, light blue, orchid, dark violet ideal color images, and also bright and low light condition images. You can see how much difference in the real/ideal color image and the non-ideal images.



Figure 17: Orange color in all the conditions



Figure 18: Light blue color in all the conditions



Figure 19: Orchid color in all the conditions

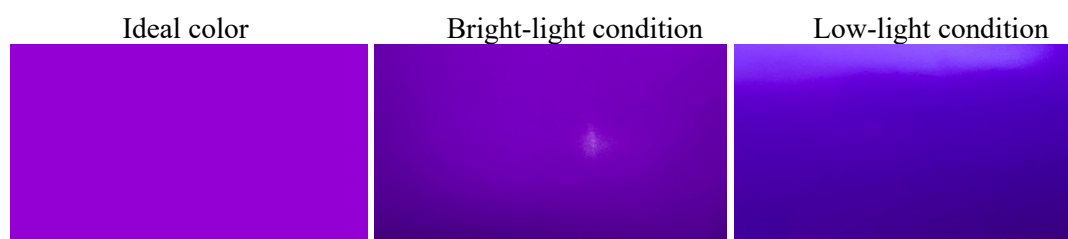


Figure 20: Dark violet color in all the conditions



From these figures, we can relate how the machine learning model failed to make the right prediction.

If you see the summary Tables 1-2, it says that orchid color in both non-ideal conditions was predicted as dark violet color. Because, in figure 19, the orchid color's bright light and low light condition image look like Figure 20's ideal color, which is dark violet.

## Chapter 5

### CONCLUSION AND FUTURE WORK

Using Machine Vision and Machine Learning algorithms, we are able to classify different color shades under different ideal and non-ideal light conditions like bright light, low/dark light, etc. We considered natural daylight to be a bright light source, and evening light/sunset light to be a low light source. This natural light source was needed in this project to create the non-ideal condition for a machine learning model. We trained and tested the machine learning model based on the data to classify different color shades.

The result of the low light condition gives us enough knowledge and visual output to conclude that in the low light condition, the predictions were darker than the actual color. Hence, the machine learning model predicts the relatively darker color shade of that actual color.

In addition, from the observation of the bright light condition, we can conclude that the machine learning model predicts a dark color relatively less dark and bright color relatively more bright shade of that actual color.

Based on the results of both non-ideal conditions, we conclude that different light conditions can alter the colors that machine vision captured and processed further, which can lead to differing color predictions.

Many future works can be possible in this project area. We can extend this project work by adding a few other non-ideal light conditions like, 1) Sharp/direct focused light, 2) Direct sunlight/natural light, 3) Defused white/warm light 4)Using UV filters to the camera.

Also, we can change photo image material like 1) Matt finished images 2) Glossy finished image.

All we need is an excellent image data set to train any model. We can take this project work to the next level by using the same methodology, but just differently. It's like a recurring process for a few days to gather image data for creating your own image dataset. We can capture images on fixed intervals all day long for multiple days, each day, capture a new color, from sunrise to sunset at a fixed location. It's like a time-lapse of color images along with the time stamp, and the intensity of the light using hardware/sensor.

Create a data set of all these images, timestamp, and sensor data together. We can give this vast data set to the ML model to predict the color difference and, based on that, learn again using reinforcement learning and predict correct color based on different light conditions with light intensity. This extensive future work can be implemented to solve real-world problems using AI and ML.

## REFERENCES

[1] “Artificial neural network - Wikipedia ”, [Online].

Available: [https://en.wikipedia.org/wiki/Artificial\\_neural\\_network](https://en.wikipedia.org/wiki/Artificial_neural_network) [Accessed Oct 10,2019]

[2] “Classification – Machine Learning”, [Online].

Available: <https://www.simplilearn.com/classification-machine-learning-tutorial>

[Accessed Oct 10, 2019].

[3] Eijaz Allibhai, “Building A Deep Learning Model using Keras - Towards Data”, [Online].

Available:<https://towardsdatascience.com/building-a-deep-learning-model-using-keras-1548ca149d37> [Accessed Apr 14, 2020].

[4] Esposito, Floriana & Malerba, Donato. (2010). Editorial: Machine learning in computer vision.

Applied Artificial Intelligence: An International Journal. 15. 693-705.  
10.1080/088395101317018546.

[5] “Find complex patterns - lynda.com”, [Online]. Available: [https://www.lynda.com/Data-](https://www.lynda.com/Data-Science-tutorials/Find-complex-patterns/601799/729679-4.html)

[Science-tutorials/Find-complex-patterns/601799/729679-4.html](https://www.lynda.com/Data-Science-tutorials/Find-complex-patterns/601799/729679-4.html) [Accessed Oct 10, 2019].

[6] Hunter Heidenreich, “what-are-the-types-of-machine-learning”, [Online].

Available: <https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f>

[Accessed Oct 10, 2020].

- [7] Ilya Michlin, “Keras data generators and how to use them”, [Online].  
Available: <https://towardsdatascience.com/keras-data-generators-and-how-to-use-them-b69129ed779c> [Accessed Nov 04, 2019].
- [8] J. Sainui and P. Pattanasatean, "Color Classification based on Pixel Intensity Values," 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Busan, 2018, pp. 302-306.
- [9] James Currier, “What Makes Data Valuable: The Truth About Data Network Effects” [Online]. Available: <https://www.nfx.com/post/truth-about-data-network-effects/> [Accessed Oct 10, 2019].
- [10] Jason Brownlee, “9 Application of Deep Learning for Computer Vision” [Online].  
<https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/>  
[Accessed Apr 14, 2020]
- [11] Jason Brownlee, “What is a Confusion Matrix in Machine Learning” [Online].  
<https://machinelearningmastery.com/confusion-matrix-machine-learning/>  
[Accessed Oct 10, 2019]
- [12] Karan Bhanot, “Color Identification in Images”, [Online].  
Available: <https://towardsdatascience.com/color-identification-in-images-machine-learning-application-b26e770c4c71> [Accessed Nov 04, 2019].

[13] “Keras Documentation - Image Processing”, [Online].

Available: <https://keras.io/preprocessing/image/> [Accessed Mar 09, 2020].

[14] Marco Peixeiro, “Step-by-step Guide to Building Your Own Neural Network From Scratch”,

[Online]. Available: <https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e> [Accessed Nov 04, 2019].

[15] Madhav, “Object Detection using Python OpenCV”, [Online]. Available:

<https://circuitdigest.com/tutorial/object-detection-using-python-opencv> [Accessed Apr 14, 2020].

[16] Mandy Sidana, “Intro to types of classification algorithms in Machine Learning”, [Online].

Available: <https://medium.com/@Mandysidana/machine-learning-types-of-classification-9497bd4f2e14> [Accessed Nov 04, 2019].

[17] “Machine Learning | Paldesk”, [Online]. Available: <https://www.paldesk.com/what-is-machine-learning/> [Accessed Apr 14, 2020].

[18] “ML Practicum: Image Classification”, [Online]. Available:

<https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?hl=en> [Accessed Nov 07, 2019].

- [19] “Neural Networks and Convolutional Neural Networks Essential Training | LinkedIn Learning, formerly Lynda.com”, [Online]. Available: <https://www.linkedin.com/learning/neural-networks-and-convolutional-neural-networks-essential-training/activation-functions> [Accessed Nov 04, 2019].
- [20] O'Shea, Keiron & Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. ArXiv e-prints.
- [21] Randerson, “Classify Hand-Written Digits Using. Python and Convolutional Neural Networks”, [Online]. Available: <https://itnext.io/classify-hand-written-digits-using-python-and-convolutional-neural-networks-26ccfc06b95c> [Accessed Nov 04, 2019].
- [22] “Use a neural network”, [Online]. Available: <https://www.lynda.com/Data-Science-tutorials/Use-neural-network/601799/729675-4.html> [Accessed Oct 10, 2019].
- [23] Will Koehrsen, “Overfitting vs. Underfitting: A complete Example” [Online]. <https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765> [Accessed Oct 10, 2019]