DETECTION OF RECOLORED IMAGES USING DEEP DISCRIMINATIVE MODEL

By:
Gattu Kamal Yeshodhar Shastry
GitHub: Recolored-Image-Detection
Reference: IEEE Document

CONTENTS

LIST OF FIGURES

ABSTRACT

Image recoloring is a technique that can transfer image color or theme and result in an imperceptible change in human eyes. Although image recoloring is one of the most important image manipulation techniques, there is no special method designed for detecting this kind of forgery. In this paper, we propose a trainable end-to-end system for distinguishing recolored images from natural images. The proposed network takes the original image and two derived inputs based on illumination consistency and inter-channel correlation of the original input into consideration and outputs the probability that it is recolored. Our algorithm adopts a CNN-based deep architecture, which consists of three feature extraction blocks and a feature fusion module. To train the deep neural network, we synthesize a dataset comprised of recolored images and corresponding ground truth using different recoloring methods. Extensive experimental results on the recolored images generated by various methods show that our proposed network is well generalized and much robust.

1.INTRODUCTION

1.1 MOTIVATION

Nowadays, millions of photographs are produced by various devices and distributed by newspapers, televisions, and websites every day. Many legal, governmental and scientific organizations use digital images as evidence of specific events to make critical decisions. Unfortunately, with the development of low-cost and high-resolution digital cameras and sophisticated photo editing softwares, it is simple to perform image manipulations and the detection of forged images is much difficult through human vision. This challenges the reliability of digital images/photographs as real-world events. Accordingly, image forensic techniques for forged images detection are necessary. Image recoloring, i.e., color transferring, is one of the most common image operations in photo editing. Usually, satisfying color transfer algorithms apply the color characteristic of a target image to a source image and generate a recolored result that human cannot distinguish. Although decent recolored images may leave no visual clues, they may alter the underlying image consistencies. Therefore, we design an approach for recoloring detection. The System takes advantages of two consistencies as well as the original input image to distinguish whether an image is recolored.

1.2 PROBLEM DEFINITION

Forgery detection in the images involves in detecting the discrepancies in the image. But there are no forensics methods specially designed for color transferring even if altering the color of an image is one of the most common tasks in image processing. Therefore, it is necessary to design approaches for recoloring detection. A deep discriminative network has to be trained for color transfer detection in images. Accordingly, we discuss the most relevant algorithms including forgery detection methods, color transfer approaches in this section. We import a large dataset of images and apply Color transfer algorithms on these images and then apply relevant algorithms to train the system and apply the trained model on test set.

1.2 OBJECTIVES

- System distinguishes recolored images from natural images.
- System analyses the inter-channel correlation and illumination consistency for natural images which may not hold after the color transfer operation. Based on these two properties, we propose a deep discriminative model for recoloring detection.
- System generates a large-scale and high-quality training dataset for training the proposed network and create a benchmark dataset consisting of skilfully recolored images and the corresponding original photographs for testing.

2. LITERATURE SURVEY

As per the surveys conducted, to the best of our knowledge, there are no forensics methods specially designed for color transferring even if altering the color of an image is one of the most common tasks in image processing. Therefore, it is necessary to design approaches for recoloring detection.

For example, Stamm et algorithm "Forensic detection of image manipulation using statistical intrinsic fingerprints," show that pixel value mappings leave behind artifacts and detect enhancement by observing the intrinsic fingerprints in the pixel value histogram. However, these state-of-the-art methods are limited by the hand-designed priors or heuristic cues which may be less effective for some images. For instance, the method proposed in Stamm et algorithm is not likely to detect tampered images if the pixel value histogram after tampering keeps smooth. Therefore, it is necessary to design approaches for recoloring detection.

2.1 EXISTING SYSTEM

Previous forged image detection approaches focus on statistical relationships of hand-crafted appearance features between the original and tampered images. For example, Stamm et al. show that pixel value mappings leave behind artifacts and detect enhancement by observing the intrinsic fingerprints in the pixel value histogram. However, these state-of-the-art methods are limited by the hand-designed priors or heuristic cues which may be less effective for some images. For instance, the method proposed in is not likely to detect tampered images if the pixel value histogram after tampering keeps smooth.

2.2 LIMITATIONS OF EXISTING SYSTEM

- These methods are limited by the hand-designed priors or heuristic cues which may be less effective for some images.
- These methods are not likely to detect tampered images if the pixel value histogram after tampering keeps smooth.
- These systems are not effective in detecting the changes in color concentration and unable to compare relationship between the color components in an image.

3.4 PROPOSED SYSTEM

The system developed is an end-to-end deep discriminative neural network to distinguish natural images from recolored images, which captures more comprehensive features. Our network employs inter-channel images and illumination map as well as the input image as the inputs for our propose network. We select these derived inter-channel images and illumination map as inputs since they have potential effectiveness for forgeries detection. Therefore, these derived inputs can provide additional information in addition to the original input. For training our proposed network, we use a color transfer method to automatically generate our training dataset. In addition, to evaluate our proposed model, we also generate a dataset in which the recolored images are generated by a variety of color transfer methods and establish a manual recolored image dataset. We analyze the inter-channel correlation and illumination consistency for natural images which may not hold after the color transfer operation. Based on these two properties, we propose a deep discriminative model for recoloring detection. We generate a large-scale and high-quality training dataset for training the proposed network and create a benchmark dataset consisting of skillfully recolored images and the corresponding original photographs for testing.

3.5 ADVANTAGES OF PROPOSED SYSTEM

- Inconsistencies in the illumination properties of images are detected.
- Forgeries concerning the changes in color properties are detected.
- Disruptions in the inter-channel correlation (relationship between each pixel of image) are detected

3.6 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.
Three key considerations involved in the feasibility analysis are

- Economic feasibility
- Technical feasibility
- Social feasibility

1. **Economic Feasibility:**

   Economic analysis is the most frequently used method for evaluating the effectiveness of the new system. More commonly known as cost/benefit analysis, the procedure is determining the benefits and savings that are expected from a candidate and compare them with costs. If benefits outweigh costs, then the decision is made to design and implement the system. And for deployment of the system, it requires only softwares which are basically open-sourced.

2. **Technical Feasibility:**

   The assessment is based on the outline design of system requirements in terms of Input, Process, Output, Fields, Programs, and Procedures. This can be quantified in terms of volumes of data, trends, frequency of updating, etc. in order to estimate where the new system will perform adequately or not. The proposed system doesn't require any additional equipment other than basic computer with internet. And the deployment requires software which is only open-sourced.

3. **Social Feasibility:**

   Determines whether the proposed system conflicts with legal requirements, e.g. - a dataset used for training and testing should not be under clause of any copyrights or patents and also the algorithms and APIs used for training the model are to be open-sourced.

3. SOFTWARE DEVELOPMENT LIFE CYCLE(SDLC)

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates. SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. A typical Software Development Life Cycle consists of the following stages:

Preliminary analysis: Begin with a preliminary analysis, propose alternative solutions, describe costs and benefits, and submit a preliminary plan with recommendations.

Systems analysis, requirements definition: Define project goals into defined functions and operations of the intended application. This involves the process of gathering and interpreting facts, diagnosing problems, and recommending improvements to the system.

Systems design: At this step, desired features and operations are described in detail, including screen layouts, business rules, process diagrams, pseudocode, and other documentation.

Development: The real code is written here.

Integration and testing: All the modules are brought together into a special testing environment, then checked for errors, bugs, and interoperability.

Acceptance, installation, deployment: This is the final stage of initial development, where the software is put into production and runs actual business.

Maintenance: During the maintenance stage of the SDLC, the system is assessed/evaluated to ensure it does not become obsolete. This is also where changes are made to initial software.

3.1 DATA ANALYTICS LIFE CYCLE

Data analysis is a process of inspecting, cleansing, transforming and modelling data with the goal of discovering useful information, informing conclusions and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, and is used in different business, science, and social science domains. It is a process to understand the data and apply statistics to get insights for a business objective.

Objective: First and foremost, Understand the Objective, the purpose. Why are you doing this? What outcome you want from it? If these questions are not clear, the rest is in vain. It is the same way that we do in SDLC (Software Development Life Cycle) model, If the requirement is not clear, then you might develop or test the software wrongly.

Understanding the Data: Once you have understood the "Objective", understanding the data is crucial. Suppose if you have bulk dataset, take a sample and then do some wear and tear to understand what it embraces. And most importantly, check if the data fulfils the objective or not.

Data Cleaning and Data Transformation: Data cleaning includes removing and replacing junk data, filling in some gaps if present. Whereas Data transformation consist of transforming the data as per your requirement to achieve the objective.

Data Enhancement: Data enhancement is adding value to the data given to you by looking for other external sources or non-traditional data.

Data Analytics: When you have all the data in desired format, you will perform Analytics which will give you the insights for the business and help in decision making. For this you can you use various algorithms to come to a conclusion and many more as per requirement.

Data Visualisation: Visualization of the analysis results makes the model more understandable. The way you present with the outcome of Analytics makes the result more readable.

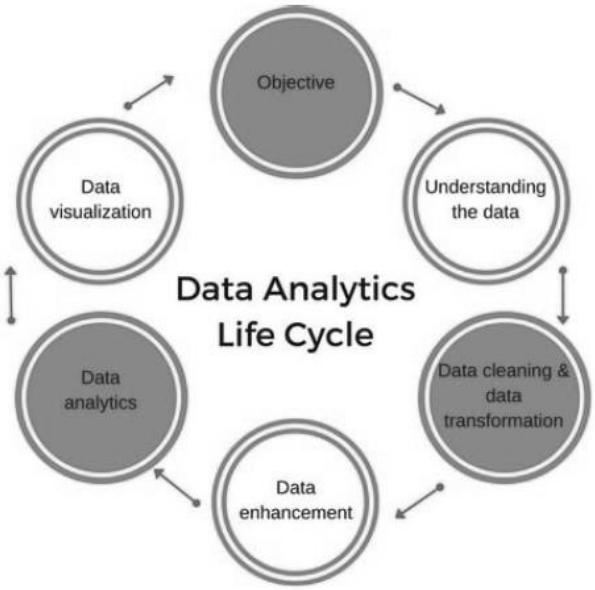

Fig 3.1 Data Analytics Life Cycle

4. SYSTEM ANALYSIS

4.1 SYSTEM REQUIREMENTS

Requirements analysis is done in order to understand the problem the software system is to solve. The emphasis in requirements analysis is on identifying *what* is needed from the system, not *how* the system will achieve its goals. This task is complicated by the fact that there are often at least two parties involved in software development-a client and a developer. The developer usually does not understand the client's problem domain and the client does not understand the issues involved in the system software systems developed by the developers.

4.1.1 FUNCTIONAL REQUIREMENTS

The functional requirements describe the interactions between the system and its environment independent of its implementation.

The proposed system should satisfy following requirements:

- Data collection
- Recoloring Algorithm
- Data Pre-processing
- Exploratory Data Analysis
- Training and Evaluating Model
- Model Evaluation
- Testing Model

Data Collection:

Collect appropriate picture data containing both indoor and outdoor pictures which contain the objects of all the types and preferentially containing various colors

Recoloring Algorithm:

Use an appropriate recoloring algorithm for creating a dataset for training the model.

Data Pre-processing:

- Importing the dataset.
- Categorizing the training images into Recolored (0) Original (1)
- Optimise images such that all are of same size.

Exploratory Data Analysis:

It involves in extraction of necessary features important for training the model.

Training and Evaluating Model:

- The model has to be trained using any efficient algorithm and API so that results are accurate and train the model for number of iterations to maintain accuracy.
- The model has to be evaluated continuously for each iteration and training has to be continued until the results are consistent and Saving the results

Testing the Model:

The model has to be applied to a new dataset for testing the accuracy.

4.1.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements describe user-visible aspects of the system that are not directly related to functionality of the system.

The proposed system has to satisfy the following requirements to make it efficient:

- Quality
- Robustness
- Security
- Acceptance
- Portability
- Documentation

Quality

Quality refers to how reliable, available and robust should the system be? While developing the system the developer must be able to guarantee the reliability transactions so that they will be processed completely and accurately.

Robustness

Robustness of system refers to the capability of system providing information when concurrent users requesting for information.

Security

Security and confidentiality are the top most concerns of the client. The proposed system should provide the following:

Acceptance

The developer will have to demonstrate and show to the user that the system works by testing with suitable test cases so that all conditions are satisfied.

Portability

It works under any environment of operating system i.e., independent of platform.

Documentations

The client is provided with an introductory help about the client interface and the user documentation has been developed.

4.1.3 SOFTWARE REQUIREMENTS

- Operating System : Windows 7 / Ubuntu 14 / Mac 10.13 or newer
- Programming Language : Python
- IDE : Anaconda3-Jupyter Notebook
- Software : Python 3.8
- Python Modules : NumPy, Pandas, Opencv2, Matplotlib, Scikit-learn, Keras

4.1.4 HARDWARE REQUIREMENTS

- ❖ Processor : Pentium core 2 duo or similar/advanced
- ❖ Hard Disk : 50 GB or more
- ❖ RAM : 4 GB or more

4.2 DATASET

A dataset is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question. In our case we use a dataset containing various categorised and non-categorised images for training and testing respectively from VOC PASCAL 2012 dataset.

Fig 4.1 Dataset Sample

4.3 ALGORITHMS

Machine learning algorithms are programs (math and logic) that adjust themselves to perform better as they are exposed to more data. It is a program with a specific way to adjusting its own parameters, given feedback on its previous performance making predictions about a dataset. We have to use appropriate algorithms for training the model to obtain maximum accuracy.

Supervised Classification:

In supervised classification the user or image analyst "supervises" the image classification process. The user specifies the various concentration values or spectral signatures that should be associated with each class. This is done by selecting representative sample sites of a known cover type called Training Sites or Areas. The computer algorithm then uses the spectral signatures from these training areas to classify the whole image. Ideally, the classes should not overlap or should only minimally overlap with other classes.

A supervised classification algorithm requires a training sample for each class, that is, a collection of data points known to have come from the class of interest. The classification is thus based on how "close" a point to be classified is to each training sample. We shall not attempt to define the word "close" other than to say that both Geometric and statistical distance measures are used in practical pattern recognition algorithms. The training samples are representative of the known classes of interest to the analyst. Classification methods that relay on use of training patterns are called supervised classification methods.

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

A system architecture or system's architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system.

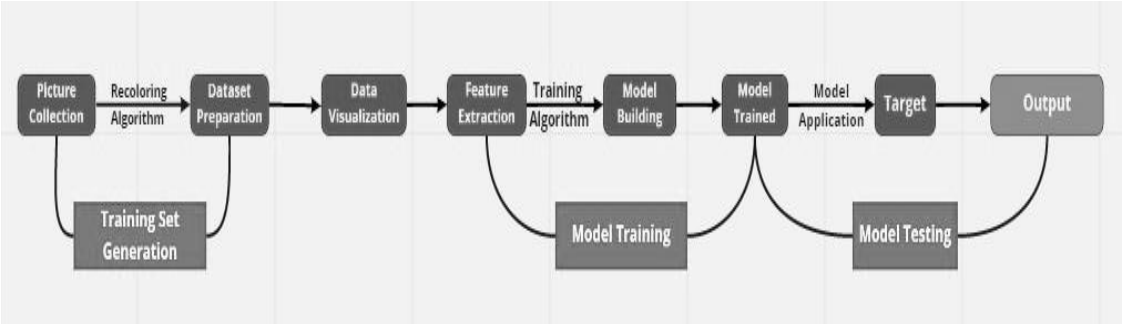

Fig 5.1 System Architecture of Network

The system for the detection of recolored images using a deep discriminative model involves following steps:

- Picture Collection
  - Collection of pictures from a source for preparing the dataset.
- Dataset Preparation
  - Applying Color Transfer Algorithms on the pictures and create a training dataset.
- Data Visualization
  - Pre-processing the raw data and visualizing the dataset.
- Feature Extraction
  - Extraction of the image features like color concentration etc which effects the category of the data.
- Model Building
  - Apply the machine learning algorithm on the dataset and train the model and fit the model with the results.

- Testing/Model Application
  - Apply the model result to testing dataset to exact the result of the system as Image "Recolored or Original".

Apart from the architecture of the system, it is also important to know about the architecture of training model in a Data Analysis/Machine Learning Problem.

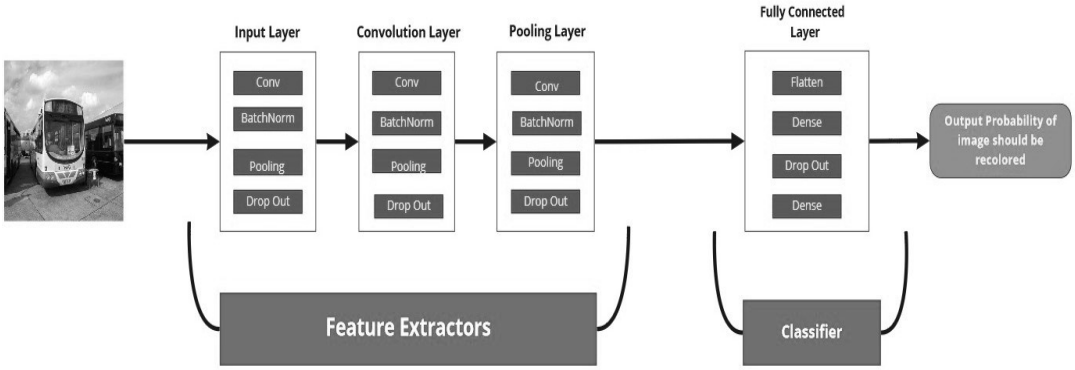

Fig 5.2 Architecture of Training Model

The Keras Sequential Training Model is a layer oriented Deep Learning API. Hence the System is trained in a step by step process to classify the data.

- **Input Layer:** It represent input image data. It will reshape image into single dimension array. Example your image is 64x64 = 4096, it will convert to (4096,1) array.
- **Convolution Layer:** This layer will extract features from image.
- **Pooling Layer:** This layer reduces the spatial volume of input image after convolution.
- **Fully Connected Layer:** It connect the network from a layer to another layer
- **Output Layer:** It is the predicted values layer.

5.2 DESIGN ANALYSIS

Dataset Creation:

We use images from VOS PASCAL 2012 Image Collection for system training. We apply a Palette Based Color Transfer Algorithm on these Images to create a set of recolored images. We use this manually created dataset for training the system.

Data Pre-processing:

The images in the dataset are cleaned and set such that all images are of same size to avoid discrepancies in training. We use Supervised Learning for training the model hence objects in each class is appended with 0 or 1 (where 0 and 1 denotes Original and Recolored Images respectively).

Feature Data Extraction:

We extract the properties (Illumination Consistency Inter-Channel Correlation) by which the class of the images in the set dependent.

Model Training and Evaluation:

We use Keras Sequential API to train our model by using Supervised Classification. We collect the dependant variables of each object and correlate this aspect to the pre-defined class of the object and using Sequential Model we train the system in multiple iterations to obtain a threshold value of dependant variables in each class. We evaluate the system by cross validating the result in each step of iteration to obtain maximum accuracy.

Model Testing:

We use the threshold value obtained in the training model to test the originality of the image. For testing, we create a new dataset containing a collection of images taken from various sources. This contain original images, recolored images from various image editing softwares available online. The image properties are extracted and checked with the Training Threshold of the system and the class of the image is decided by checking whether the value is less than or greater than the threshold value.

Visualization:

The properties of the dataset are visualised before the training. The results of the training of model is visualised. And the results of testing the prepared model is visualised in end.

5.3 UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering.

The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software.

In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Security Alert for Crime Prone Areas.

Goals: The Primary goals in the design of the UML are as follows:

- ✓ Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.
- ✓ Provide extendibility and specialization mechanisms to extend the core concepts.
- ✓ Be independent of particular programming languages and development process.
- ✓ Provide a formal basis for understanding the modelling language.

The list of UML diagrams is as follows:

- ➢ Use case diagram
- ➢ Sequence diagram
- ➢ Class diagram
- ➢ Activity diagram

5.3.1 Use Case Diagram:

UML provides the use case diagram to facilitate the process of requirements gathering. The use case diagram models the interactions between the system's external clients and the use cases of the system. Each use case represents a different capability that the system provides the client.

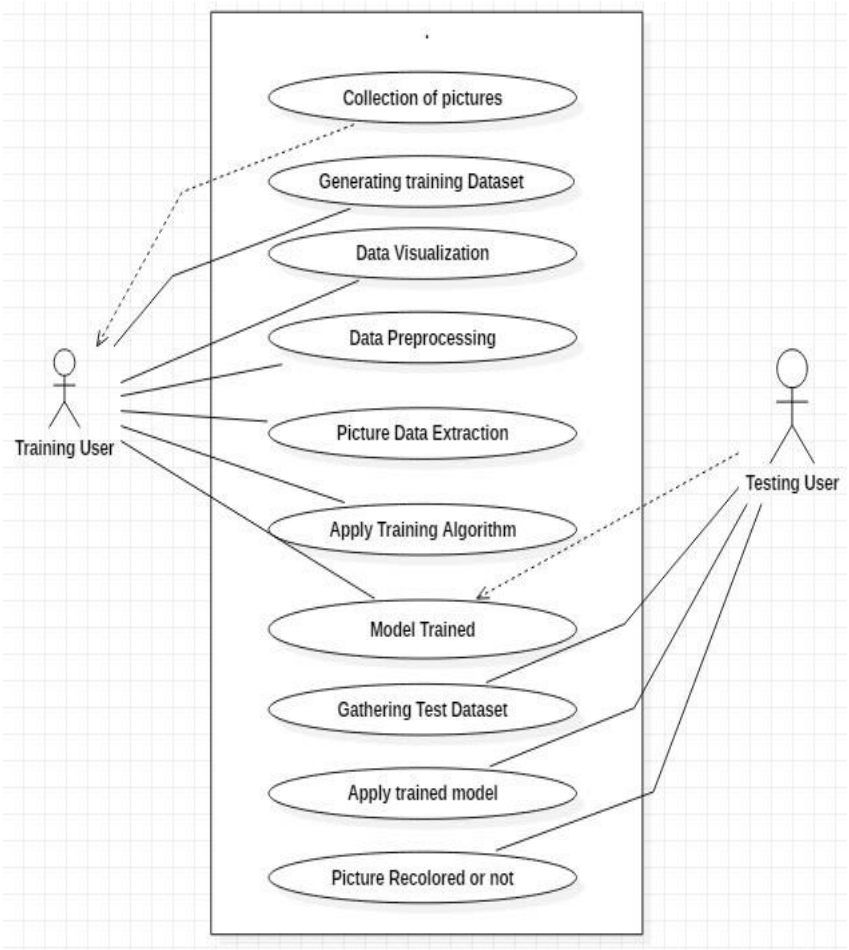

Fig 5.3 Use Case Diagram

Use case diagram of Detection of Recolored Images using Deep Discriminative Model explains about the properties and actions of Training user and Testing user

(i) User: Training User

Use case:

- Collection of pictures
- Generating training dataset

- Data Visualization
- Data Pre-processing
- Picture Data Extraction
- Apply Training Algorithm
- Model Trained

(ii) User: Testing User

Use case:

- Gathering Testing Dataset
- Apply Trained Model
- Picture Recolored or not

5.3.2 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. This shows the interactions between the objects step by step allowing one to understand the system clearly.

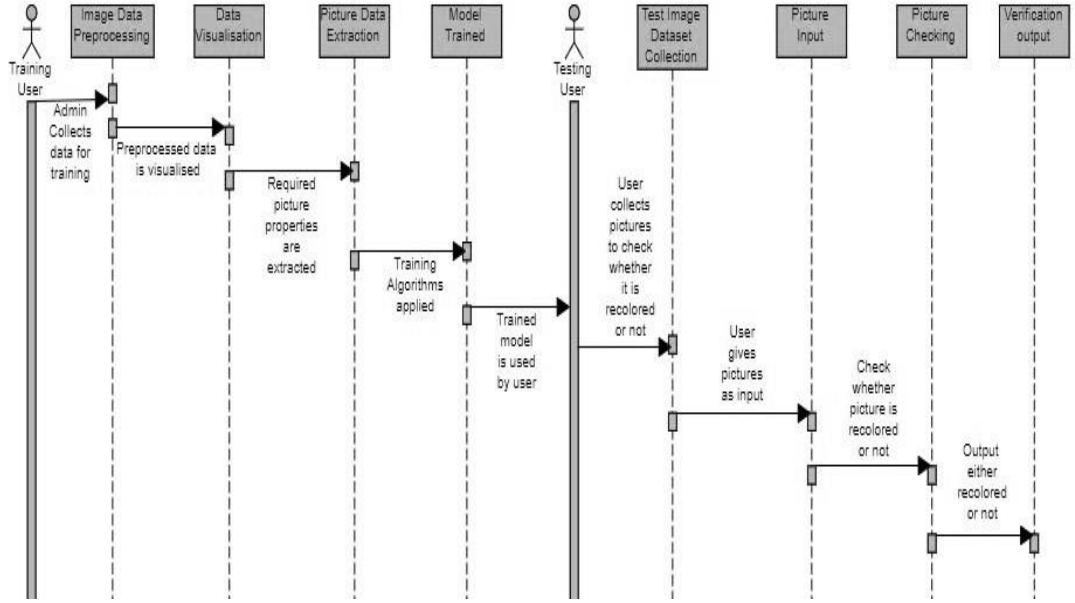

Fig 5.4 Sequence Diagram

System Action Sequence

- Training User collects large set of images.
- Color transfer algorithms are applied on images to create training set.
- Images in dataset are visualised.
- Model training algorithms are applied on the training set.
- Model is trained and train data is stored.
- Testing user collects a dataset- of images of different types.
- Image features are checked with training results.
- Output is given whether image is recolored or original.

5.3.3 Class Diagram

A class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

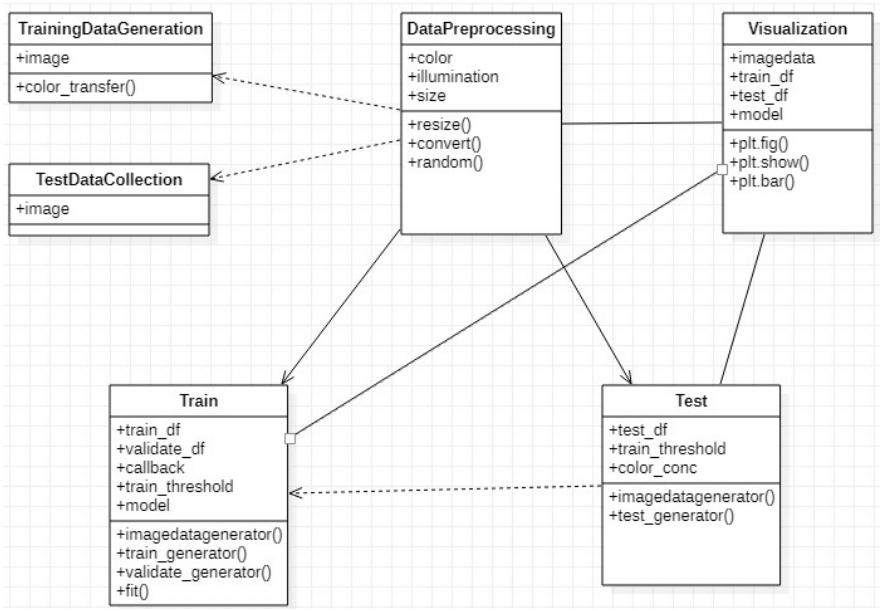

Fig 5.5 Class Diagram

The network developed have four major classes and two supporting classes.

- Training Data Generation
- Test Data Collection
- Data Pre-processing
- Train
- Test
- Data Visualization

5.3.3 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, concurrency and overall flow of control.

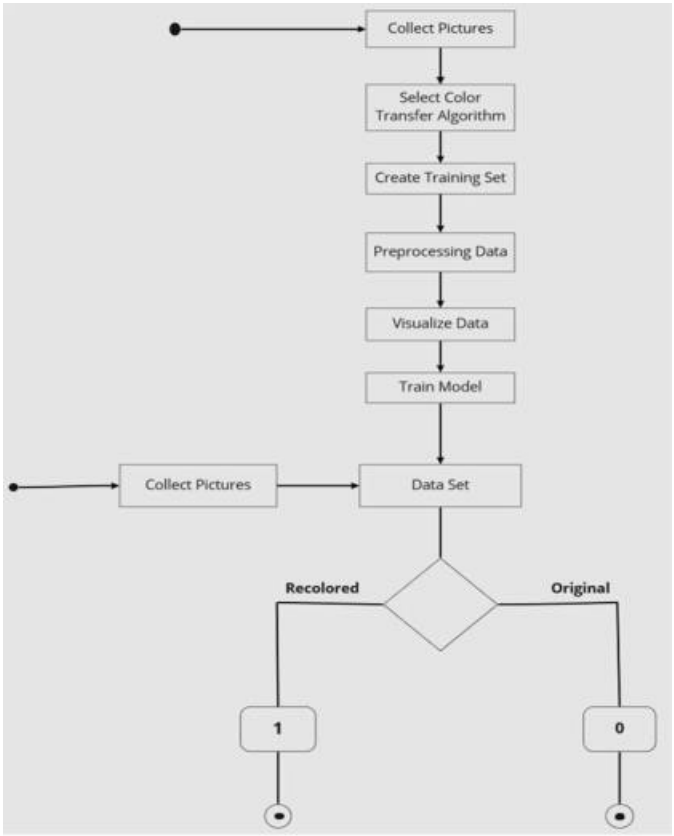

Fig 5.6 Activity Diagram

System Workflow:

- A large set of images is collected for training the model.
- Color transfer algorithms are applied on images to create training set.
- The image data is pre-processed and cleaned
- Images in dataset are visualised.
- Model training algorithms are applied on the training set.
- A new set of images is collected for testing the trained model.
- Image features are checked with results of trained model and output is given as whether image is recolored or original.

6. IMPLEMENTATION

6.1 INTRODUCTION TO ENVIRONMENT

6.1.1 Programming Language - Python

History of Python:

The programming language Python was conceived in the late 1980s and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to ABC capable of exception handling and interfacing with the Amoeba operating system.

Python 3.0, a major, backwards-incompatible release, was released on December 3, 2008 after a long period of testing. Many of its major features have also been backported to the backwards-compatible, while by now unsupported, Python 2.6 and 2.7

Features of Python:

1.Easy to code:

Python is high level programming language. It is very easy to code in python language and anybody can learn python basic in few hours or days. It is also developer-friendly language.

2. Free and Open Source:

Python language is freely available at official website. Since, it is open-source, this means that source code is also available to the public. So, you can download it as, use it as well as share it.

3.Object-Oriented Language:

One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation etc.

4. High-Level Language:

Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

5. Python is Portable language:

Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platform such as Linux, Unix and Mac then we do not need to change it, we can run this code on any platform.

6. Interpreted and dynamically typed Language:

Python is an Interpreted Language i.e. python code is executed line by line at a time. like other language c, c++, java etc., there is no need to compile python code this makes it easier to debug our code.

7. Large Standard Library

Python has a large standard library which provides rich set of module and functions so you do not have to write your own code for every single thing.

6.1.2 Integrated Development Environment – Anaconda Jupyter Notebook

History of Jupyter Notebook

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. It is developed and maintained by

Anaconda, Inc. The distribution includes data-science packages suitable for Windows, Linux, and macOS.

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter. The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results.

Notebook documents:

Notebook documents contains the inputs and outputs of an interactive session as well as additional text that accompanies the code but is not meant for execution. In this way, notebook files can serve as a complete computational record of a session, interleaving executable code with explanatory text, mathematics, and rich representations of resulting objects.

Installation of Jupyter Notebook:

Prerequisites:

While Jupyter runs code in many programming languages, Python is a requirement (Python 3.3 or greater, or Python 2.7) for installing the Jupyter Notebook. It is recommended to use the Anaconda distribution to install Python and Jupyter.

Installing Jupyter using Anaconda and conda:

It is highly recommended to install Anaconda. Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for data science.

Steps:

- ✓ Download Anaconda. We recommend downloading Anaconda's latest Python 3 version (currently Python 3.7).
- ✓ Install the version of Anaconda which you downloaded, following the instructions on the download page.
- ✓ Jupyter Notebook is installed as a part of Anaconda Package.

Installing Jupyter with *pip*:

Jupyter can also be installed using Python's package manager, pip, instead of Anaconda.

Steps

- ✓ First, ensure that you have the latest pip:

```
pip3 install --upgrade pip
```

- ✓ Then install the Jupyter Notebook using:

```
pip3 install jupyter
```

Starting the Notebook Server:

Notebook server can be started from the command line (using Terminal on Mac/Linux, Command Prompt on Windows) by running:

```
jupyter notebook
```

This will print some information about the notebook server in your terminal, including the URL of the web application (by default, http://localhost:8888):

```
$ jupyter notebook
[I 08:58:24.417 NotebookApp] Serving notebooks from local directory: D\Rec ImgDet
[I 08:58:24.417 NotebookApp] 0 active kernels
[I 08:58:24.417 NotebookApp] The Jupyter Notebook is running at: http://loca lhost:8888/
[I 08:58:24.417 NotebookApp] Use Control-C to stop this server and shut dow n all kernels (twice to skip confirmation).
```

It will then open your default web browser to this URL. When the notebook opens in your browser, you will see the Notebook Dashboard, which will show a list of the notebooks, files, and subdirectories in the directory where the notebook server was started.

Notebook Dashboard:

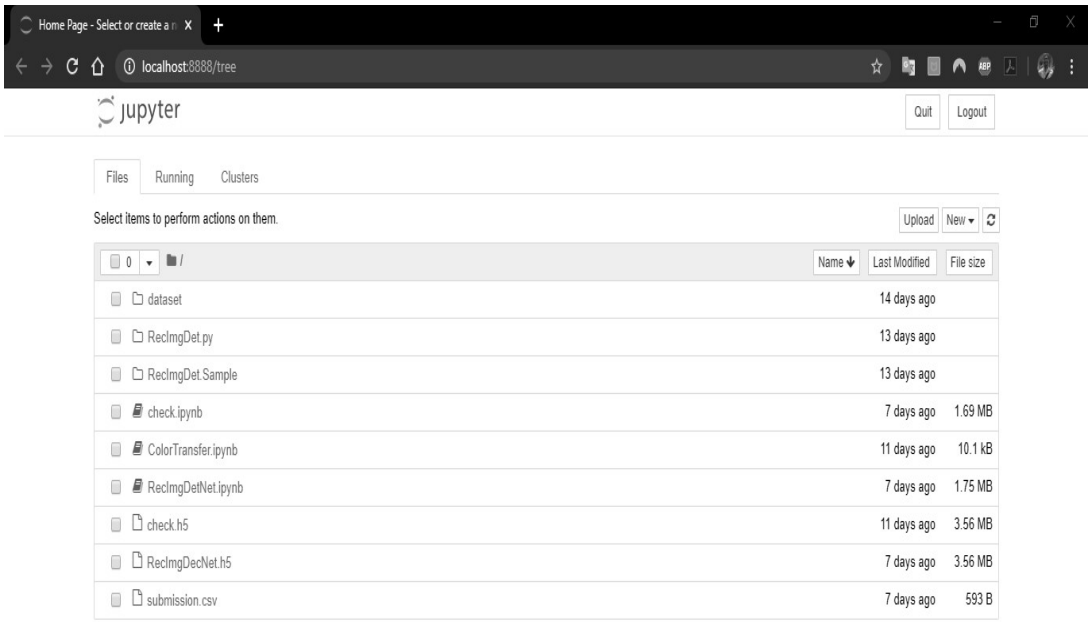

Fig 6.1 Jupyter Notebook Dashboard

Creating A Jupyter Notebook:

Any Python code for an application or any algorithm, we can create a new Jupyter Notebook from dashboard of Jupyter.

Steps:

- ✓ Open Jupyter Notebook Dashboard.
- ✓ Click on New
- ✓ Select Python3 from Notebook dropdown
- ✓ Start coding cell wise
- ✓ The file containing all the operations is stored with a .ipnyb file which can be converted into .py file.

6.2 INTRODUCTION TO PYTHON LIBRARIES:

6.2.1 Python OS

This module provides a portable way of using operating system dependent functionality. For example, if you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module.

The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system. OS Library of the python is installed by default when we install Python.

6.2.2 Python NumPy

NumPy stands for Numerical Python. NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Installation of NumPy:

In most use cases the best way to install NumPy on your system is by using a pre-built package for your operating system. Python comes with an inbuilt package management system, pip. Pip can install, update, or delete any official package.

You can install packages via the command line by entering:

```
python -m pip install –user numpy
```

6.2.3 Python Pandas

Pandas stands for "Python Data Analysis Library". Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. Pandas takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data frame that looks very similar to table in a statistical software. It is built on the NumPy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables.

Installation of Pandas:

The easiest way for the majority of users to install pandas is to install it as part of the Anaconda distribution, a cross platform distribution for data analysis and scientific computing. This is the recommended installation method for most users.

Installation using pip

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large "on-line repository" termed as Python Package Index (PyPI). Pandas can be installed using PIP by the use of the following command: pip install pandas

```
pip install pandas
```

6.2.4 Python OpenCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

Installation of OpenCV

Prerequisites:

- ✓ Install matplotlib
- ✓ Install NumPy

Install using pip

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large "on-line repository" termed as Python Package Index (PyPI). Pandas can be installed using PIP by the use of the following command:

```
pip install opencv-python
```

6.2.5 Python Scikit-learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. Scikit-learn integrates well with many other Python libraries, such as matplotlib and plotly for plotting, numpy for array vectorization, pandas dataframes, SciPy, and many more. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. As such, the module provides learning algorithms and is named scikit-learn. The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as ease of use, code quality, collaboration, documentation and performance..

Installation of Scikit-learn:

Installation using pip:

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large "on-line repository" termed as Python Package Index (PyPI). Pandas can be installed using PIP by the use of the following command:

```
pip install -U scikit-learn
```

6.2.6 Python Keras

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Use Keras if you need a deep learning library that: Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility). Supports both convolutional networks and recurrent networks, as well as combinations of the two. Runs seamlessly on CPU and GPU.

Installation of Keras:

Before installing Keras, please install one of its backend engines: TensorFlow, Theano, or CNTK. We recommend the TensorFlow backend.

Install using pip

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large “on-line repository” termed as Python Package Index (PyPI). Pandas can be installed using PIP by the use of the following command:

```
pip install keras
```

6.2.7 Python Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Installation of Matplotlib:

Before Matplotlib's plotting functions can be used, Matplotlib needs to be installed. Depending on which distribution of Python is installed on your computer, the installation methods are slightly different.

Install Matplotlib with pip

Matplotlib can also be installed using the Python package manager, pip. To install Matplotlib with pip,

- ✓ Open a terminal window and type:

  $ pip install matplotlib

6.3 INTRODUCTION TO THE DATA SOURCE

A data set (or dataset) is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Each value is known as a datum. Data sets can also consist of a collection of documents or files. In the open data discipline, data set is the unit to measure the information released in a public open data repository. The European Open Data portal aggregates more than half a million data sets. In this field other definitions have been proposed, but currently there is not an official one.

For each variable, the values are normally all of the same kind. However, there may also be missing values, which must be indicated in some way. In statistics, data sets usually come from actual observations obtained by sampling a statistical population, and each row corresponds to the observations on one element of that population. Data sets may further be generated by algorithms for the purpose of testing certain kinds of software.

For training the model to classify the images there is necessity of an image set containing images of different types, images of different environments, images of different scenarios. The network trained uses a comprehensive collection of images named VOC PASCAL 2012 for training.

VOC PASCAL 2012

VOC PASCAL is a challenge conductor based on data analysis on images. It provides standardised image data sets for object class recognition. It provides a common set of tools for accessing the data sets and annotations. Enables evaluation and comparison of different methods. It runs challenges evaluating performance on object class recognition. It provides a data set of images of various types like persons, animals, birds, vehicles, trees, objects in both outdoor and indoor environment.

The training data provided consists of a set of images; each image has an annotation file giving a bounding box and object class label for each object in one of the twenty classes present in the image. Note that multiple objects from multiple classes may be present in the same image. Annotation was performed according to a set of guidelines distributed to all annotators. A subset of images are also annotated with pixel-wise segmentation of each object present, to support the segmentation competition. Images for the action classification task are disjoint from those of the classification/detection/segmentation tasks. They have been partially annotated with people, bounding boxes, reference points and their actions. Annotation was performed according to a set of guidelines distributed to all annotators. Images for the person layout taster, where the test set is disjoint from the main tasks, have been additionally annotated with parts of the people (head/hands/feet).

We use few images from this dataset to train the system by applying Recoloring Algorithms and Color Transfer algorithms to create a synthetic dataset. Since edit propagation and palette based recoloring methods require artificial manipulation and are inappropriate for generating a large number of training data, only example-based recoloring methods are used to generate training data.

As our recoloring detection is a binary classification task, we need a balance between the positive and negative examples in training data. In this work, given an original photograph I, one recoloring method is randomly selected to generate the recolored image. Therefore, the ratio between the positive and negative examples is 1, which is the most appropriate for binary classification using the neural network.

6.4 CREATION OF TRAINING SET

We divide the above collected images into two folders so as to create a set of recolored images by transferring the color properties of each image in the first folder to each image in the second folder using below explained color transfer algorithm and create a dataset of images containing collection of both source and transfer images.

COLOR TRANSFER ALGORITHM

One of the most common tasks in image processing is to alter an image's color. Often, this means removing a dominant and undesirable color cast, such as the yellow in photos taken under incandescent illumination. This article describes a method for a more general form of color correction that borrows one image's color characteristics from another. Figure 1 shows an example of this process, where we applied the colours of a sunset photograph to a daytime Computer graphics rendering.

When a typical three channel image is represented in any of the most well-known color spaces, there will be correlations between the different channels' values. For example, in RGB space, most pixels will have large values for the red and green channel if the blue channel is large. This implies that if we want to change the appearance of a pixel's color in a coherent way, we must modify all color channels in tandem. This complicates any color modification process. What we want is an orthogonal color space without correlations between the axes.

Decorrelated color space

Because our goal is to manipulate RGB images, which are often of unknown phosphor chromaticity, we first show a reasonable method of converting RGB signals to Ruderman et al.'s perception-based color space lαβ. Because lαβ is a transform of LMS cone space, we first convert the image to LMS space in two steps. The first is a conversion from RGB to XYZ tristimulus values. This conversion depends on the phosphors of the monitor that the image was originally intended to be displayed on. Because that information is rarely available, we use a device-independent conversion that maps white in the chromaticity diagram (CIE xy) to white in RGB space and vice versa. Because we define white in the chromaticity diagram as $x = X/(X + Y + Z) = 0.333$, $y = Y/(X + Y + Z) = 0.333$, we need a transform that maps $X = Y = Z = 1$ to $R = G = B = 1$. To achieve this, we modified the XYZitu601-1 (D65) standard conversion matrix to have rows that add up to 1. The International Telecommunications Union standard matrix is

$$M_{itu} = \begin{bmatrix} 0.4306 & 0.3415 & 0.1784 \\ 0.2220 & 0.7067 & 0.0713 \\ 0.0202 & 0.1295 & 0.9394 \end{bmatrix} \quad (1)$$

By letting $M_{itu}x = (111)^T$ and solving for x, we obtain a vector x that we can use to multiply the columns of matrix M_{itu}, yielding the desired RGB to XYZ conversion:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.5141 & 0.3239 & 0.1604 \\ 0.2651 & 0.6702 & 0.0641 \\ 0.0241 & 0.1228 & 0.8444 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (2)$$

Compared with M_{itu}, this normalization procedure constitutes a small adjustment to the matrix's values. Once in device-independent XYZ space, we can convert the image to LMS space using the following conversion:

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3897 & 0.6890 & -0.0787 \\ -0.2298 & 1.1834 & 0.0464 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (3)$$

Combining these two matrices gives the following transformation between RGB and LMS cone space:

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3811 & 0.5783 & 0.0402 \\ 0.1967 & 0.7244 & 0.0782 \\ 0.0241 & 0.1288 & 0.8444 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (4)$$

The data in this color space shows a great deal of skew, which we can largely eliminate by converting the data to logarithmic space:

$$\begin{aligned} \boldsymbol{L} &= \log L \\ \boldsymbol{M} &= \log M \\ \boldsymbol{S} &= \log S \end{aligned} \quad (5)$$

We can compute maximal decorrelation between the three axes using principal components analysis (PCA), which effectively rotates them. The three resulting orthogonal principal axes have simple forms and are close to having integer coefficients.

$$\begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{L} \\ \mathbf{M} \\ \mathbf{S} \end{bmatrix} \quad (6)$$

If we think of the L channel as red, the Mas green, and the S as blue, we can see that this is a variant of many opponent-color models:

$$\begin{aligned} &\text{Achromatic} && \propto r+g+b \\ &\text{Yellow–blue} && \propto r+g-b \\ &\text{Red–green} && \propto r-g \end{aligned} \quad (7)$$

Thus, the l axis represents an achromatic channel, while the α and β channels are chromatic yellow–blue and red–green opponent channels. The data in this space are symmetrical and compact, while we achieve decorrelation to higher than second order for the set of natural images tested. Our color-correction method operates in this lαβ space because decorrelation lets us treat the three color channels separately, simplifying the method. After color processing, which we explain in the next section, we must transfer the result back to RGB to display it. For convenience, here are the inverse operations. We convert from lαβ to LMS using this matrix multiplication:

$$\begin{bmatrix} \mathbf{L} \\ \mathbf{M} \\ \mathbf{S} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{3} & 0 & 0 \\ 0 & \frac{\sqrt{6}}{6} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} \quad (8)$$

Then, after raising the pixel values to the power ten to go back to linear space, we can convert the data from LMS to RGB using:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 4.4679 & -3.5873 & 0.1193 \\ -1.2186 & 2.3809 & -0.1624 \\ 0.0497 & -0.2439 & 1.2045 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix} \quad (9)$$

Statistics and color correction

The goal of our work is to make a synthetic image take on another image's look and feel. Thus, we compute these measures for both the source and target images. Note that we compute the means and standard deviations for each axis separately in lαβ space. First, we subtract the mean from the data points

$$\begin{aligned} l^* &= l - \langle l \rangle \\ \alpha^* &= \alpha - \langle \alpha \rangle \\ \beta^* &= \beta - \langle \beta \rangle \end{aligned} \qquad (10)$$

Then, we scale the data points comprising the synthetic image by factors determined by the respective standard deviations:

$$\begin{aligned} l' &= \frac{\sigma_t^l}{\sigma_s^l} l^* \\ \alpha' &= \frac{\sigma_t^\alpha}{\sigma_s^\alpha} \alpha^* \\ \beta' &= \frac{\sigma_t^\beta}{\sigma_s^\beta} \beta^* \end{aligned} \qquad (11)$$

Next, instead of adding the averages that we previously subtracted, we add the averages computed for the photograph. Finally, we convert the result back to RGB via log LMS, LMS, and XYZ color spaces using Equations 8 and 9.

We apply the above color transfer algorithm on the images in the collected dataset to create a training dataset containing both recolored images, original images but named with different names for images of each class, so that model can distinguish between the classes (we use supervised learning for training the system).

Examples of applications of algorithm:

Fig 6.2: We applied (a) the atmosphere of Vincent van Gogh's Cafe Terrace on the Place du Forum, Arles, at Night to (b) a photograph of Legnica Castle near Brno in the Czech Republic. (c) We matched the blues of the sky in both images, the yellows of the cafe and the castle, and the browns of the tables at the cafe and the people at the castle separately.

Fig 6.3: Rendered forest (image courtesy of Oliver Deussen);(b) old photograph of a tuna processing plant at Sancti Petri, referred to as El Bosque, or the forest(image courtesy of the NOAA Photo Library);and (c) corrected rendering.

Fig 6.4: Two images are taken from VOC PASCAL 2012 dataset and Color properties pf first image are transferred to that of second image to get a recolored image (third image above).

Fig 6.2 Color Transfer Demo 1

Fig 6.3 Color Transfer Demo 2

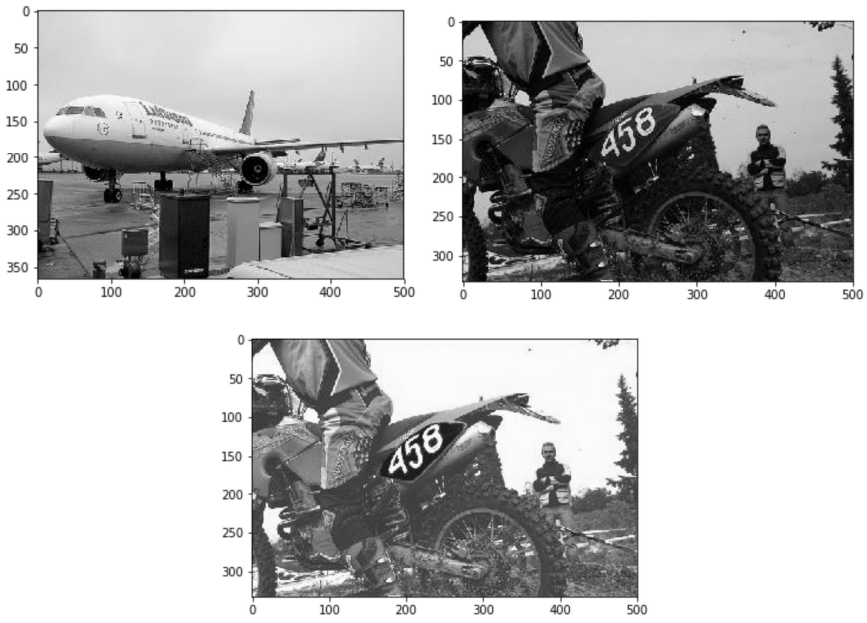

Fig 6.4 Color Transfer Output Sample

6.5 MODEL TRAINING – TRAINING ALGORITHMS

The process of training an ML model involves providing an ML algorithm (that is, the learning algorithm) with training data to learn from. The term ML model refers to the model artifact that is created by the training process. The training data must contain the correct answer, which is known as a target or target attribute. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict), and it outputs an ML model that captures these patterns.

To train an ML model, you need to specify the following:

- Input training data source
- Name of the data attribute that contains the target to be predicted
- Required data transformation instructions
- Training parameters to control the learning algorithm

Typically, machine learning algorithms accept parameters that can be used to control certain properties of the training process and of the resulting ML model. In Amazon Machine Learning, these are called training parameters. You can set these parameters using the Amazon ML console, API, or command line interface (CLI). If you do not set any parameters, Amazon ML will use default values that are known to work well for a large range of machine learning tasks.

You can specify values for the following training parameters:

- Maximum model size
- Maximum number of passes over training data
- Shuffle type
- Regularization type
- Regularization amount

Larger input data sets do not necessarily result in larger models because models store patterns, not input data; if the patterns are few and simple, the resulting model

will be small. Input data that has a large number of raw attributes (input columns) or derived features will likely have more patterns found and stored during the training process. Picking the correct model size for your data and problem is best approached with a few experiments.

In general, data sets with only a few observations typically require more passes over the data to obtain higher model quality. Larger data sets often contain many similar data points, which eliminates the need for a large number of passes. The impact of choosing more data passes over your data is two-fold: model training takes longer, and it costs more.

You must shuffle your training data even if you chose the random split option when you split the input datasource into training and evaluation portions. The random split strategy chooses a random subset of the data for each datasource, but it doesn't change the order of the rows in the datasource. For more information about splitting your data, see Splitting Your Data.

The predictive performance of complex ML models (those having many input attributes) suffers when the data contains too many patterns. As the number of patterns increases, so does the likelihood that the model learns unintentional data artifacts, rather than true data patterns. In such a case, the model does very well on the training data, but can’t generalize well on new data. This phenomenon is known as *overfitting* the training data.

Regularization helps prevent linear models from overfitting training data examples by penalizing extreme weight values. L1 regularization reduces the number of features used in the model by pushing the weight of features that would otherwise have very small weights to zero. L1 regularization produces sparse models and reduces the amount of noise in the model. L2 regularization results in smaller overall weight values, which stabilizes the weights when there is high correlation between the features. You can control the amount of L1 or L2 regularization by using the Regularization amount parameter. Specifying an extremely large Regularization amount value can cause all features to have zero weight.

Selecting and tuning the optimal regularization value is an active subject in machine learning research. You will probably benefit from selecting a moderate amount of L2 regularization. Advanced users can choose between three types of regularization (none, L1, or L2) and amount.

During training the model using data we use various logical and statistical methods to correlate interconnection between the data and use these relationships to find the class of the data objects. These are called Machine Learning Algorithms. Machine learning algorithms are programs (math and logic) that adjust themselves to perform better as they are exposed to more data. The "learning" part of machine learning means that those programs change how they process data over time, much as humans change how they process data by learning. So, a machine-learning algorithm is a program with a specific way to adjusting its own parameters, given feedback on its previous performance making predictions about a dataset.

There are so many algorithms that it can feel overwhelming when algorithm names are thrown around and you are expected to just know what they are and where they fit. There are basically two ways to think about and categorize the algorithms you may come across in the field.

- The first is a grouping of algorithms by their learning style.
- The second is a grouping of algorithms by their similarity in form or function.

There are different ways an algorithm can model a problem based on its interaction with the experience or environment or whatever we want to call the input data. It is popular in machine learning and artificial intelligence textbooks to first consider the learning styles that an algorithm can adopt. There are only a few main learning styles or learning models that an algorithm can have and we'll go through them here with a few examples of algorithms and problem types that they suit. This taxonomy or way of organizing machine learning algorithms is useful because it forces you to think about the roles of the input data and the model preparation process and select one that is the most appropriate for your problem in order to get the best result.

Instance-based learning model

Instance-based learning model is a decision problem with instances or examples of training data that are deemed important or required to the model. Such methods typically build up a database of example data and compare new data to the database using a similarity measure in order to find the best match and make a prediction. For this reason, instance-based methods are also called winner-take-all methods and memory-based learning. Focus is put on the representation of the stored instances and similarity measures used between instances.

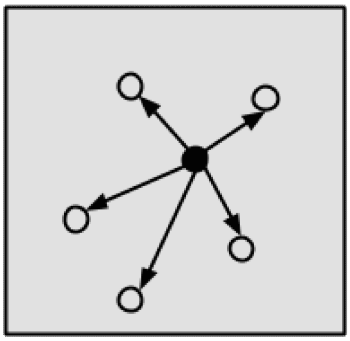

Fig 6.5 Instance Based Algorithms

Deep Learning Algorithms

Deep Learning methods are a modern update to Artificial Neural Networks that exploit abundant cheap computation. They are concerned with building much larger and more complex neural networks and, as commented on above, many methods are concerned with very large datasets of labelled analog data, such as image, text. audio, and video.

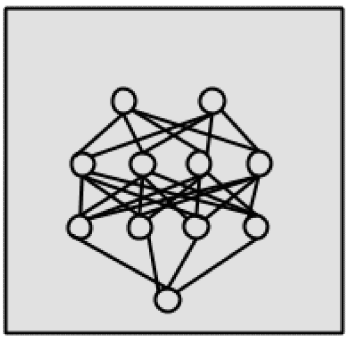

Fig 6.6 Deep Learning Algorithms

Classification Algorithm:

We use the training dataset to get better boundary conditions that could be used to determine each target class. Once the boundary conditions are determined, the next task is to predict the target class. The whole process is known as classification.

Image Classification:

Image classification is a procedure to automatically categorize all pixels in an Image of a terrain into land cover classes. Normally, multispectral data are used to Perform the classification of the spectral pattern present within the data for each pixel is used as the numerical basis for categorization. This concept is dealt under the Broad subject, namely, Pattern Recognition. Spectral pattern recognition refers to the Family of classification procedures that utilizes this pixel-by-pixel spectral information as the basis for automated land cover classification. Spatial pattern recognition involves the categorization of image pixels on the basis of the spatial relationship with pixels surrounding them. Image classification techniques are grouped into two types, namely supervised and unsupervised. The classification process may also include features, such as, land surface elevation and the soil type that are not derived from the image. Two categories of classification are contained different types of techniques can be seen in fig

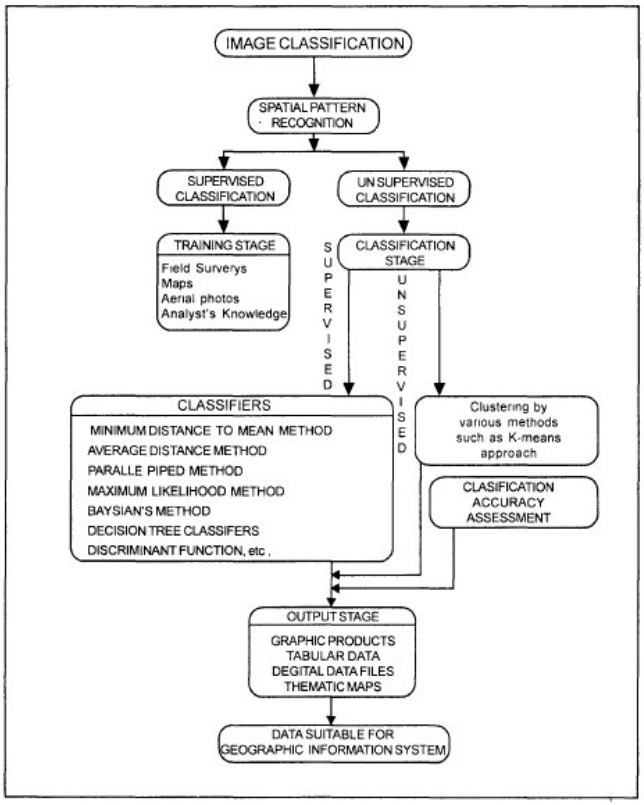

Fig 6.7 Image Classification Procedure

Supervised Image Classification:

In supervised classification the user or image analyst "supervises" the pixel classification process. The user specifies the various pixels values or spectral signatures that should be associated with each class. This is done by selecting representative sample sites of a known cover type called Training Sites or Areas. The computer algorithm then uses the spectral signatures from these training areas to classify the whole image. Ideally, the classes should not overlap or should only minimally overlap with other classes.

A supervised classification algorithm requires a training sample for each class, that is, a collection of data points known to have come from the class of interest. The classification is thus based on how "close" a point to be classified is to each training sample. We shall not attempt to define the word "close" other than to say that both Geometric and statistical distance measures are used in practical pattern recognition algorithms. The training samples are representative of the known classes of interest to

the analyst. Classification methods that relay on use of training patterns are called supervised classification methods. The three basic steps involved in a typical supervised classification procedure are as follows:

(i) Training stage: The analyst identifies representative training areas and develops numerical descriptions of the spectral signatures of each land cover type of interest in the scene.

(ii) The classification stage: Each pixel in the image data set IS categorized into the land cover class it most closely resembles. If the pixel is insufficiently similar to any training data set it is usually labelled 'Unknown'.

(iii) The output stage: The results may be used in a number of different ways. Three typical forms of output products are thematic maps, tables and digital data files which become input data for GIS. The output of image classification becomes input for GIS for spatial analysis of the terrain. Fig. 2 depicts the flow of operations to be performed during image classification of remotely sensed data of an area which ultimately leads to create database as an input for GIS. Plate 6 shows the land use/ land cover color coded image, which is an output of image

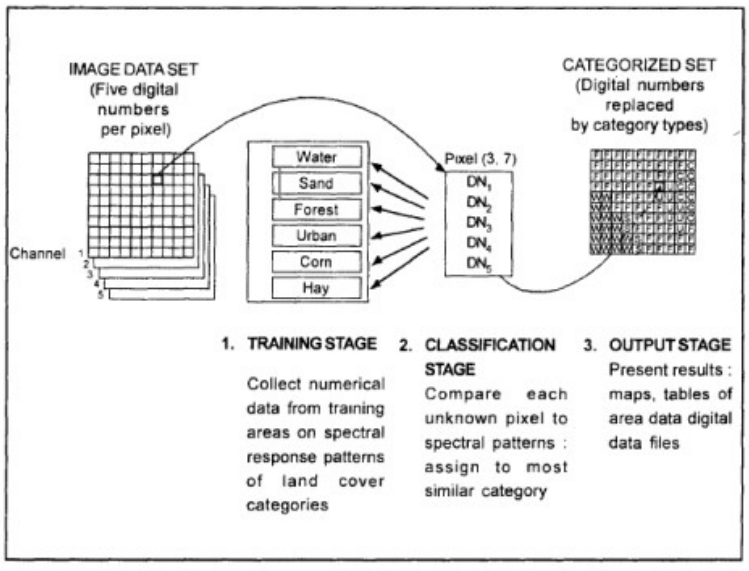

Fig 6.8 Supervised Image Classification Procedure

Supervised Algorithm for Image Classification:

The principles and working algorithms of all these supervised classifiers are derived as follows:

Parallelepiped Classification

Parallelepiped classification, sometimes also known as box decision rule, or level-slice procedures, are based on the ranges of values within the training data to define regions within a multidimensional data space. The spectral values of unclassified pixels are projected into data space; those that fall within the regions defined by the training data are assigned to the appropriate categories. In this method a parallelepiped-like (i.e., hyper-rectangle) subspace is defined for each class. Using the training data for each class the limits of the parallelepiped subspace can be defined either by the minimum and maximum pixel values in the given class, or by a certain number of standard deviations on either side of the mean of the training data for the given class. The pixels lying inside the parallelepipeds are tagged to this class. Figure depicts this criterion in cases of two-dimensional feature space for three classes using two spectral bands.

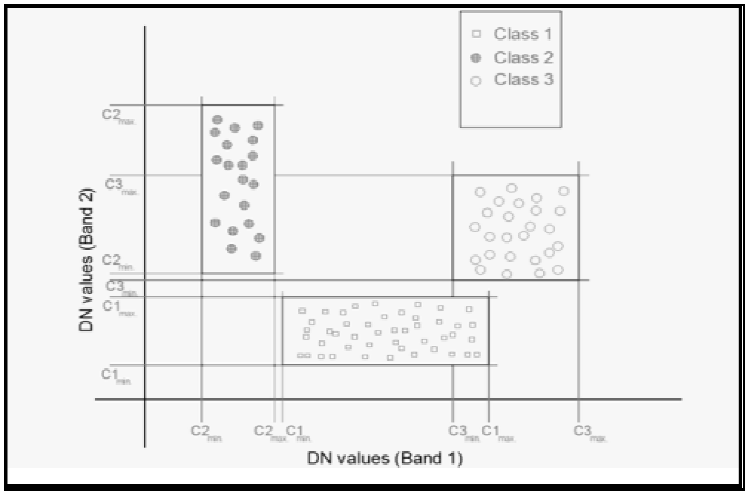

Fig 6.9 Parallelepiped Classification

Minimum Distance Classification

For supervised classification, these groups are formed by values of pixels within the training fields defined by the analyst. Each cluster can be represented by its centroid, often defined as its mean value. As unassigned pixels are considered for assignment to

one of the several classes, the multidimensional distance to each cluster centroid is calculated, and the pixel is then assigned to the closest cluster. Thus, the classification proceeds by always using the "minimum distance" from a given pixel to a cluster centroid defined by the training data as the spectral manifestation of an informational class. Minimum distance classifiers are direct in concept and in implementation but are not widely used in remote sensing work. In its simplest form, minimum distance classification is not always accurate; there is no provision for accommodating differences in variability of classes, and some classes may overlap at their edges. It is possible to devise more sophisticated versions of the basic approach just outlined by using different distance measures and different methods of defining cluster centroids.

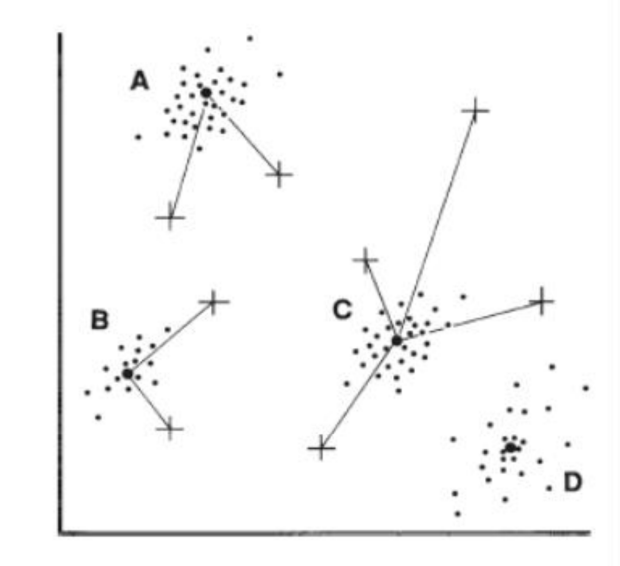

Fig 6.10 Minimum Distance Classification

The Euclidean distance is the most common distance metric used in low dimensional data sets. It is also known as the L2 norm. The Euclidean distance is the usual manner in which distance is measured in real world. In this sense, Manhattan distance tends to be more robust to noisy data.

Euclidean distance = $\sqrt{\Sigma_i (X_i - Y_i)^2}$ (1)

Where x and y are m-dimensional vectors and denoted by x = (x1, x2, x3… xm) and y = (y1, y2, y3… ym) represent the m attribute values of two classes. While

Euclidean metric is useful in low dimensions, it doesn't work well in high dimensions and for categorical variables.

Maximum Likelihood Classification

In nature the classes that we classify exhibit natural variation in their spectral patterns. Further variability is added by the effects of haze, topographic shadowing, system noise, and the effects of mixed pixels. As a result, remote sensing images seldom record spectrally pure classes; more typically, they display a range of brightness's in each band. The classification strategies considered thus far do not consider variation that may be present within spectral categories and do not address problems that arise when frequency distributions of spectral values from separate categories overlap. The maximum likelihood (ML) procedure is the most common supervised method used with remote sensing. It can be described as a statistical approach to pattern recognition where the probability of a pixel belonging to each of a predefined set of classes is calculated; hence the pixel is assigned to the class with the highest probability MLC is based on the Bayesian probability formula.

K-Nearest Neighbour Algorithm:

K-Nearest Neighbour Algorithm is an algorithm which measures dependant variables and classifies the objects to its defined class based on the distance of the value of dependant variable from the mean value of the variable in the class dataset. In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression: In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until function evaluation. Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the

average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor. The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

Scikit-learn implements two different nearest neighbors classifiers: **K Neighbors Classifier** implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user. **Radius Neighbors Classifier** implements learning based on the number of neighbors within a fixed radius r of each training point, where r is a floating-point value specified by the user.

In cases where the data is not uniformly sampled, radius-based neighbors classification in **RadiusNeighborsClassifier** can be a better choice. The user specifies a fixed radius r, such that points in sparser neighbourhoods use fewer nearest neighbors for the classification. For high-dimensional parameter spaces, this method becomes less effective due to the so-called “curse of dimensionality”.

The basic nearest neighbors classification uses uniform weights: that is, the value assigned to a query point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit. This can be accomplished through the weights keyword. The default value, weights = 'uniform', assigns uniform weights to each neighbor. weights = 'distance' assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied to compute the weights.

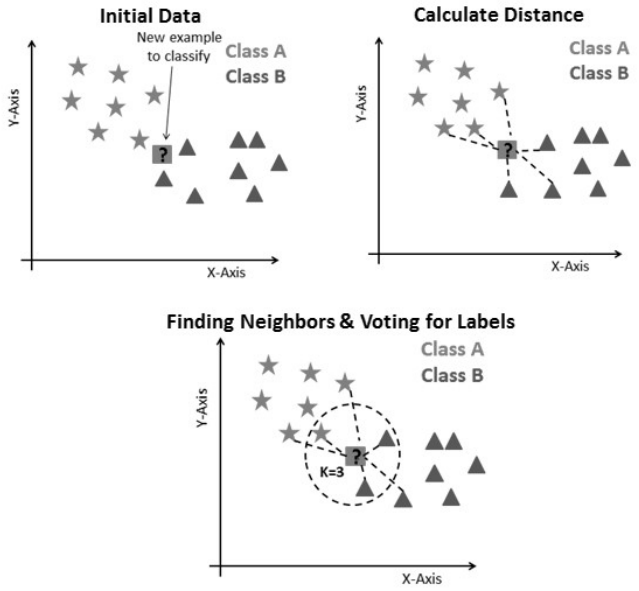

Fig 6.11 Classification using KNN Algorithm

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

Keras-Sequential

The sequential API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs. Sequence models can be augmented using an attention mechanism. This algorithm will help your model understand where it should focus its attention given a sequence of inputs. The Sequential model API is great for developing deep learning models in most situations, but it also has some limitations. For example, it is not straightforward to define models that may have:

- Multiple different input sources,
- Produce multiple output destinations, or
- Models that re-use layers.

The Sequential model is a linear stack of layers. You can create a Sequential model by passing a list of layer instances to the constructor:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

You can also simply add layers via the .add() method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Specifying the input shape

The model needs to know what input shape it should expect. For this reason, the first layer in a Sequential model (and only the first, because following layers can do automatic shape inference) needs to receive information about its input shape. There are several possible ways to do this:

- Pass an input_shape argument to the first layer. This is a shape tuple (a tuple of integers or None entries, where None indicates that any positive integer may be expected). In input_shape, the batch dimension is not included.
- Some 2D layers, such as Dense, support the specification of their input shape via the argument input_dim, and some 3D temporal layers support the arguments input_dim and input_length.
- If you ever need to specify a fixed batch size for your inputs (this is useful for stateful recurrent networks), you can pass a batch_size argument to a layer. If you pass both batch_size=32 and input_shape=(6, 8) to a layer, it will then expect every batch of inputs to have the batch shape (32, 6, 8).
- As such, the following snippets are strictly equivalent:

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')

# For custom metrics
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

Compilation

Before training a model, you need to configure the learning process, which is done via the compile method. It receives three arguments:

- An optimizer. This could be the string identifier of an existing optimizer (such as rmsprop or adagrad), or an instance of the Optimizer class. See: optimizers.
- A loss functions. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as categorical_crossentropy or mse), or it can be an objective function. See: losses.
- A list of metrics. For any classification problem you will want to set this to metrics=['accuracy']. A metric could be the string identifier of an existing metric or a custom metric function. See: metrics.

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')

# For custom metrics
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

Training

Keras models are trained on Numpy arrays of input data and labels. For training a model, you will typically use the fit function.

```
# For a single-input model with 2 classes (binary classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

fit() method:

Trains the model for a fixed number of epochs (iterations on a dataset).

```
fit(x=None, y=None, batch_size=None, epochs=1, verbose=1,callbacks=None, validation_split=0.0,
validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0,
steps_per_epoch=None, validation_steps=None, validation_freq=1, max_queue_size=10)
```

6.6 Model Evaluation

A model should always be evaluated to determine if it will do a good job of predicting the target on new and future data. Because future instances have unknown target values, you need to check the accuracy metric of the ML model on data for which you already know the target answer, and use this assessment as a proxy for predictive accuracy on future data. To properly evaluate a model, you hold out a sample of data that has been labelled with the target (ground truth) from the training datasource. Evaluating the predictive accuracy of an ML model with the same data that was used for training is not useful, because it rewards models that can "remember" the training data, as opposed to generalizing from it. Once you have finished training the ML model, you send the model the held-out observations for which you know the target values. You then compare the predictions returned by the ML model against the known target value. Finally, you compute a summary metric that tells you how well the predicted and true values match.

Preventing Overfitting:

When creating and training an ML model, the goal is to select the model that makes the best predictions, which means selecting the model with the best settings (ML model settings or hyperparameters). Overfitting occurs when a model has memorized patterns that occur in the training and evaluation datasource, but has failed to generalize the patterns in the data. It often occurs when the training data includes all of the data used in the evaluation. An overfitted model does well during evaluations, but fails to make accurate predictions on unseen data. To avoid selecting an overfitted model as the best model, you can reserve additional data to validate the performance of the ML model. For example, you can divide your data into 60 percent for training, 20 percent for evaluation, and an additional 20 percent for validation. After selecting the model parameters that work well for the evaluation data, you run a second evaluation with the validation data to see how well the ML model performs on the validation data. If the model meets your expectations on the validation data, then the model is not overfitting the data. To solve this problem, you can perform cross-validation.

Cross Validation:

Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, ie, failing to generalize a pattern.

The following diagram shows an example of the training subsets and complementary evaluation subsets generated for each of the four models that are created and trained during a 4-fold cross-validation. Model one uses the first 25 percent of data for evaluation, and the remaining 75 percent for training. Model two uses the second subset of 25 percent (25 percent to 50 percent) for evaluation, and the remaining three subsets of the data for training, and so on.

Fig 6.12 Cross Validation of a data source

Each model is trained and evaluated using complementary datasource - the data in the evaluation datasource includes and is limited to all of the data that is not in the training datasource.

Adjusting Your Models

After you have cross-validated the models, you can adjust the settings for the next model if your model does not perform to your standards. This can be done using Callbacks in Sequential API

Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. You can pass a list of callbacks (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes. The relevant methods of the callbacks will then be called at each stage of the training.

```
keras.callbacks.callbacks.Callback()
```

Few of the callbacks used are as follows:

History

Callback that records events into a History object. This callback is automatically applied to every Keras model. The History object gets returned by the fit method of models.

```
keras.callbacks.callbacks.History()
```

EarlyStopping

Stop training when a monitored quantity has stopped improving.

keras.callbacks.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=0, mode='auto', baseline=**None**, restore_best_weights=**False**)

ReduceLROnPlateau

Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

Keras.callbacks.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10, verbose=0, mode='auto', min_delta=0.0001, cooldown=0, min_lr=0)

6.7 System Build Code

6.7.1 Creation of Training Set using Color Transfer Algorithm

```
# ### Creating a dataset of Recolored Images and Original Images for training the Model

import numpy as np

import cv2

#  > **Color Transfer Algorithm**

def color_transfer(source, target):

    # convert color space from BGR to L*a*b color space

    ## L* for the lightness from black to white, a* from green to red, and b* from blue to yellow.

    # note - OpenCV expects a 32bit float rather than 64bit

    source                =                cv2.cvtColor(source, cv2.COLOR_BGR2LAB).astype("float32")

    target                =                cv2.cvtColor(target, cv2.COLOR_BGR2LAB).astype("float32")

    # compute color stats for both images

    (lMeanSrc, lStdSrc, aMeanSrc, aStdSrc, bMeanSrc, bStdSrc) = image_stats(source)

    (lMeanTar, lStdTar, aMeanTar, aStdTar, bMeanTar, bStdTar) = image_stats(target)

    # split the color space

    (l, a, b) = cv2.split(target)

    # substarct the means from target image

    l -= lMeanTar

    a -= aMeanTar

    b -= bMeanTar

    # scale by the standard deviation

    l = (lStdTar/lStdSrc)*l
```

```
    a = (aStdTar/aStdSrc)*a
    b = (bStdTar/bStdSrc)*b
    # add the source mean
    l += lMeanSrc
    a += aMeanSrc
    b += bMeanSrc
    # clipping the pixels between 0 and 255(0 denotes black and
255 denotes white)
    l = np.clip(l, 0, 255)
    a = np.clip(a, 0, 255)
    b = np.clip(b, 0, 255)
    # merge the channels
    transfer = cv2.merge([l, a, b])
    # converting back to BGR
    transfer      =      cv2.cvtColor(transfer.astype("uint8"),
cv2.COLOR_LAB2BGR)
    return transfer
def image_stats(image):
    # compute mean and standard deviation of each channel
    (l, a, b) = cv2.split(image)
    (lMean, lStd) = (l.mean(), l.std())
    (aMean, aStd) = (a.mean(), a.std())
    (bMean, bStd) = (b.mean(), b.std())
    return (lMean, lStd, aMean, aStd, bMean, bStd)
def show_image(title, image, width=720):
    r = width/float(image.shape[1])
    dim = (width, int(image.shape[0]*r))
```

```
    resized = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
    cv2.imshow(title, resized)


source = cv2.imread("dataset/source/source (1).jpg")
target = cv2.imread("dataset/target/target (1).jpg")
# transfer of color
transfer = color_transfer(source, target)
# display of image
show_image("Source", source)
show_image("Target", target)
show_image("Transfer", transfer)
cv2.waitKey(0)
# > - Applying Color transfer to some of the images(including both indoor and outdoor images) taken from VOC PASCAL 2012 dataset
from os import listdir
from os.path import isfile, join
import numpy
import cv2
import os


mypath1='dataset/source/'
mypath2='dataset/target/'


onlyfiles1 = [ f for f in listdir(mypath1) if isfile(join(mypath1,f)) ]
onlyfiles2 = [ f for f in listdir(mypath2) if isfile(join(mypath2,f)) ]
```

```
print(len(onlyfiles1))
print(len(onlyfiles2))
images1 = numpy.empty(len(onlyfiles1), dtype=object)
images2 = numpy.empty(len(onlyfiles2), dtype=object)
for n in range(0, len(onlyfiles1)):
    images1[n] = cv2.imread( join(mypath1,onlyfiles1[n]) )
    images1[n] = cv2.cvtColor(images1[n], cv2.COLOR_BGR2RGB)
    images1[n]=cv2.resize(images1[n],(500,500))

    images2[n] = cv2.imread( join(mypath2,onlyfiles2[n]) )
    images2[n] = cv2.cvtColor(images2[n], cv2.COLOR_BGR2RGB)
    images2[n]=cv2.resize(images2[n],(500,500))
    # transfer of color
    transfer = color_transfer(images1[n], images2[n])
    #write images in a folder
    path2='dataset/trainingset'
    cv2.imwrite(os.path.join(path2 ,
'img.{}.jpg'.format(n)),transfer)##for labeling the recolor
images images
    cv2.imwrite(os.path.join(path2 ,
'pic.{}.jpg'.format(n)),images2[n])#for labeling the original
images
    #different labelling for recolored and original images is
done for supervised learning
```

6.7.2 Network for Detection of Recolored Images

```
# # DETECTION OF RECOLORED IMAGES USING  DEEP DISCRIMINATIVE MODEL

# ## Training The Model

import numpy as np # linear algebra

import pandas as pd # data processing

from keras.preprocessing.image import ImageDataGenerator, load_img

from keras.utils import to_categorical

from sklearn.model_selection import train_test_split

#scikit-learn and keras for training and testing the model

import matplotlib.pyplot as plt # visualizing the model

import random

import os

print(os.listdir("dataset"))


FAST_RUN = False

IMAGE_WIDTH=128

IMAGE_HEIGHT=128

IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)

IMAGE_CHANNELS=3 # RGB color


# > - Use the dataset created by RecImDet.ipnyb for training the model


filenames = os.listdir("dataset/trainingset")

categories = []

for filename in filenames:
```

```
    category = filename.split('.')[0]
    if category == 'pic':
        categories.append(1)
    else:
        categories.append(0)
df = pd.DataFrame({'filename': filenames,'category': categories})
print(df.shape)
df.head()
df.tail()
df['category']=df['category'].astype(str)
df.dtypes
sample = random.choice(filenames)
image = load_img("dataset/trainingset/"+sample)
plt.imshow(image)
df['category'].value_counts().plot.bar()


from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization
model = Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
```

```
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

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

model.summary()


# **Callbacks**

from keras.callbacks import EarlyStopping, ReduceLROnPlateau

earlystop = EarlyStopping(monitor='val_loss',patience=10)

learning_rate_reduction                                          =
ReduceLROnPlateau(monitor='val_loss',patience=2, verbose=1,

                                          factor=0.05,
min_lr=0.00001)


callbacks = [earlystop,learning_rate_reduction]

train_df,  validate_df  =  train_test_split(df,  test_size=0.20,
random_state=42)

train_df = train_df.reset_index(drop=True)
```

```
validate_df = validate_df.reset_index(drop=True)

total_train = train_df.shape[0]

total_validate = validate_df.shape[0]

batch_size=4

print(train_df.shape)

print(validate_df.shape)

train_df['category'].value_counts().plot.bar()

train_datagen = ImageDataGenerator(

rotation_range=15,rescale=1./255,shear_range=0.1,zoom_range=0.
2,

horizontal_flip=True,width_shift_range=0.1,height_shift_range=
0.1)

train_generator = train_datagen.flow_from_dataframe(

    train_df,"dataset/trainingset", x_col='filename',

    y_col='category',target_size=IMAGE_SIZE,

    class_mode='binary',batch_size=batch_size)

validation_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_dataframe(

validate_df,"dataset/trainingset",x_col='filename',y_col='cate
gory',

target_size=IMAGE_SIZE,class_mode='binary',batch_size=batch_si
ze)


# **Fit the model**

epochs=3 if FAST_RUN else 20

history = model.fit_generator(
```

```
train_generator,epochs=epochs,validation_data=validation_gener
ator,


validation_steps=total_validate//batch_size,steps_per_epoch=to
tal_train//batch_size,)


trainmodel=model.predict_generator(train_generator,
steps=np.ceil(train_df.shape[0]/batch_size))

trainmean=sum(trainmodel)/len(trainmodel)

train_threshold=trainmean[0]

train_threshold

# **Save the model**


model.save_weights("RecImgDecNet.h5")

# **Visualize Training**

fig, (ax1) = plt.subplots(1, 1, figsize=(8, 5))

ax1.plot(history.history['loss'],  color='b',  label="Training
loss")

ax1.plot(history.history['val_loss'],               color='r',
label="Validation loss")

ax1.plot(history.history['accuracy'],               color='y',
label="Training accuracy")

ax1.plot(history.history['val_accuracy'],
color='g',label="Validation accuracy")

legend = plt.legend(loc='best', shadow=True)

plt.tight_layout()

plt.show()


# ## Testing The Model
```

```
# **Preparing Test Data**

test_filenames = os.listdir("dataset/testingset")

test_df = pd.DataFrame({

    'filename': test_filenames

})

nb_samples = test_df.shape[0]

print(nb_samples)


# **Creating Test Generator**

test_gen = ImageDataGenerator(rescale=1./255)

test_generator = test_gen.flow_from_dataframe(

    test_df, "dataset/testingset",x_col='filename', y_col=None,


class_mode=None,target_size=IMAGE_SIZE,batch_size=batch_size,s
huffle=False)


# - For categoral classication the prediction will come with
probability of each category.

color_conc      =      model.predict_generator(test_generator,
steps=np.ceil(nb_samples/batch_size))

print(color_conc)

threshold = train_threshold

test_df['color_concentration'] = color_conc

test_df['category'] = np.where(test_df['color_concentration'] >
threshold, 1,0)

test_df['category'].value_counts().plot.bar()


# **Visualize the Test Results**

sample_test = test_df #.head(5)
```

```
sample_test.head()

plt.figure(figsize=(12, 24))

for index, row in sample_test.iterrows():

    filename = row['filename']

    category = row['category']

    color_concentration = row['color_concentration']

    img        =        load_img("dataset/testingset/"+filename,
target_size=IMAGE_SIZE)

    plt.subplot(11, 3, index+1)

    plt.imshow(img)

    plt.xlabel(filename + '(' + "{}".format(category) + ')' '('
+ "{}".format(round(color_concentration, 2)) + ')')

plt.figtext(1,1,'Category(0):RECOLORED,Category(1):ORIGINAL',f
ontsize='large')

plt.tight_layout()

plt.show()


# **Submission of Test Results to a CSV**

submission_df = test_df.copy()

submission_df['id']                                          =
submission_df['filename'].str.split('.').str[0]

submission_df['label'] = submission_df['category']

submission_df.drop(['filename',      'category'],      axis=1,
inplace=True)

submission_df.to_csv('submission.csv', index=False)

print('\n \n Category(0):RECOLORED,Category(1):ORIGINAL \n \n')

print(test_df)
```

7. SYSTEM TESTING

Testing is a process, which reveals errors in the program. It is the major quality measure employed during software development. During testing, the program is executed with a set of conditions known as test cases and the output is evaluated to determine whether the program is performing as expected. In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development are:

7.1 TYPES OF TESTING

Black Box Testing:

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been uses to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structure or external database access
- Performance errors
- Initialization and termination errors.

In this testing only the output is checked for correctness. The logical flow of the data is not checked.

White Box Testing:

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been uses to generate the test cases in the following cases:

- Guarantee that all independent paths have been executed.
- Execute all logical decisions on their true and false sides.
- Execute all loops at their boundaries and within their operational
- Execute internal data structures to ensure their validity.

7.2 LEVELS OF TESTING

Unit Testing:

Unit Testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements.

Integration Testing:

Integration testing ensures that software and subsystems work together as a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together.

System Testing:

Involves in-house testing of the entire system before delivery to the user. Its aim is to satisfy the user the system meets all requirements of the client's specifications.

Acceptance Testing:

It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors.

Validation Testing:

The system has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirements specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed.

Compile Testing:

It was a good idea to do our stress testing early on, because it gave us time to fix some of the unexpected deadlocks and stability problems that only occurred when components were exposed to very high transaction volumes.

Execution Test:

This program was successfully loaded and executed. Because of good programming there was no execution error.

Output Test:

The successful output screens are placed in the output screens section. Testing is the process of finding differences between the expected behavior specified by system models and the observed behavior of the system.

7.3 TESTING A MACHINE LEARNING MODEL

Traditional testing techniques are based on fixed inputs. This is not true in machine learning systems. The output is not fixed. It will change over time as we know more and as the model on which the machine learning system is built evolves as it is fed more data. This forces the testing professional to think differently and adopt test strategies that are very different from traditional testing techniques.

Mentioned below are critical activities essential to test machine learning systems:

- Developing test data sets: This is a dataset that is intelligently built to test all the possible combinations and estimates how well your model is trained. The model will be fine-tuned based on the results of the test data set.
- Developing validation test suites based on algorithms and test datasets.
- The key to building validation suites is to understand the algorithm. This is based on calculations that create a model from the training data.
- The model is refined as the number of iterations and the richness of the data increase.
- Communicating test results in statistical terms.

7.4 Testing the Trained Model of RecDeNet

To further demonstrate the practicability of our framework, we synthesize a new dataset including few authentic photographs and corresponding synthetic recolored images. We first crawl the original photographs from the website, then employ a variety of color transferring methods to generate the recolored images. Note that edit propagation and palette based recoloring methods are also used to generate recolored images in this section for testing the generalization ability of the RecDeNet. Most of the recolored images in the synthetic dataset appear quite realistic, it is hard to distinguish them from authentic photographs by human vision. We utilize our trained RecDeNet to judge the images in the synthetic dataset. The accuracy reaches 83:50% and the AUC achieves 90:75%, which reveals that our RecDeNet is effective for various types of color transferring methods.

8.INPUT AND RESULT SCREENS

```
from os import listdir
from os.path import isfile, join
import numpy
import cv2
import os

mypath1='dataset/source/'
mypath2='dataset/target/'

onlyfiles1 = [ f for f in listdir(mypath1) if isfile(join(mypath1,f)) ]
onlyfiles2 = [ f for f in listdir(mypath2) if isfile(join(mypath2,f)) ]

print(len(onlyfiles1))
print(len(onlyfiles2))
images1 = numpy.empty(len(onlyfiles1), dtype=object)
images2 = numpy.empty(len(onlyfiles2), dtype=object)
for n in range(0, len(onlyfiles1)):
    images1[n] = cv2.imread( join(mypath1,onlyfiles1[n]) )
    images1[n] = cv2.cvtColor(images1[n], cv2.COLOR_BGR2RGB)
    images1[n]=cv2.resize(images1[n],(500,500))

    images2[n] = cv2.imread( join(mypath2,onlyfiles2[n]) )
    images2[n] = cv2.cvtColor(images2[n], cv2.COLOR_BGR2RGB)
    images2[n]=cv2.resize(images2[n],(500,500))
    # transfer of color
    transfer = color_transfer(images1[n], images2[n])
    #write images in a folder
    path2='dataset/trainingset'
    cv2.imwrite(os.path.join(path2 , 'img.{}.jpg'.format(n)),transfer)##for labeling the recolor images images
    cv2.imwrite(os.path.join(path2 , 'pic.{}.jpg'.format(n)),images2[n])#for labeling the original images
    #different labelling for recolored  and original images is done for supervised learning
```

Fig 8.1 Input of Images for Dataset Creation Using Color Transfer

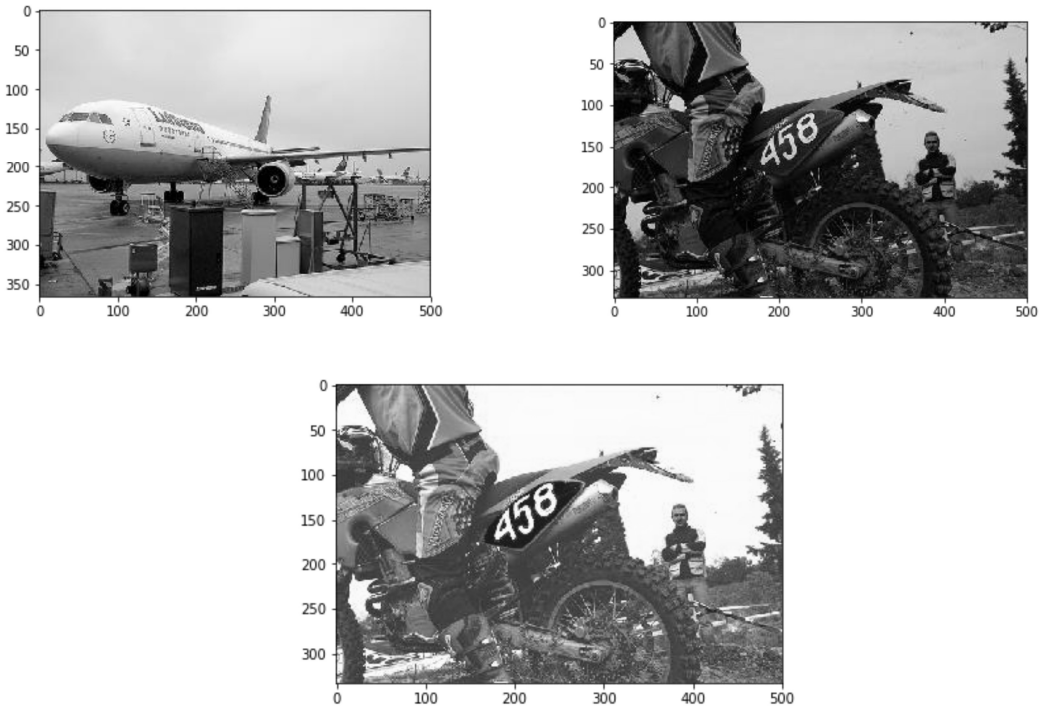

Fig 8.2 Color Properties Of 1st Image are Transferred to 2nd Image to Get 3rd Image as Output

```
#we use supervised learning
filenames = os.listdir("dataset/trainingset")
categories = []
for filename in filenames:
    category = filename.split('.')[0]
    if category == 'pic':
        categories.append(1)
    else:
        categories.append(0)

df = pd.DataFrame({
    'filename': filenames,
    'category': categories
})
```

```
print(df.shape)
```

```
(100, 2)
```

```
df.head()
```

	filename	category
0	img.0.jpg	0
1	img.1.jpg	0
2	img.10.jpg	0
3	img.11.jpg	0
4	img.12.jpg	0

Fig 8.3 Feeding the Above Created Dataset as Input for Training

```
1 sample = random.choice(filenames)
2 image = load_img("dataset/trainingset/"+sample)
3 plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x24da295a2e8>
```

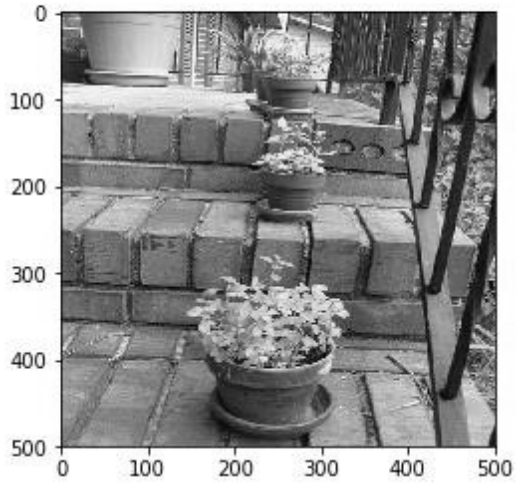

Fig 8.4 Loading A Random Image from Dataset

```
1 train_df['category'].value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x24da3bf0400>
```

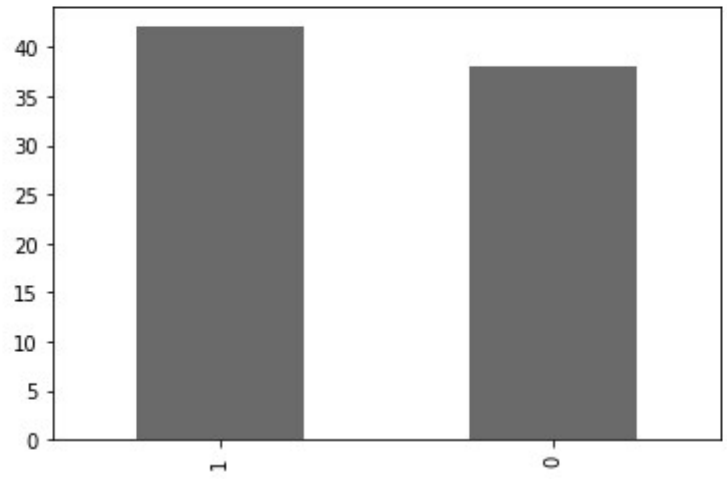

Fig 8.5 Plot Representing Statistics of Recolored And Original Images

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model = Sequential()

model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

model.summary()
```

Fig 8.6 Image Features Extraction

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 64)      1792
_________________________________________________________________
batch_normalization_1 (Batch (None, 126, 126, 64)      256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 63, 63, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 64)        36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 61, 61, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                802880
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 879,617
Trainable params: 879,105
Non-trainable params: 512
_________________________________________________________________
```

Fig 8.7 Summary of Model

```
epochs=3 if FAST_RUN else 20
history = model.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,

)
```

```
Epoch 1/20
20/20 [==============================] - 14s 722ms/step - loss: 1.2293 - accuracy: 0.4875 - val_loss: 1.5006 - val_accuracy: 0.
6000
Epoch 2/20
20/20 [==============================] - 9s 455ms/step - loss: 0.8131 - accuracy: 0.5000 - val_loss: 1.3886 - val_accuracy: 0.6
000
Epoch 3/20
20/20 [==============================] - 9s 448ms/step - loss: 0.9692 - accuracy: 0.4500 - val_loss: 0.7565 - val_accuracy: 0.6
000
Epoch 4/20
20/20 [==============================] - 9s 442ms/step - loss: 0.8906 - accuracy: 0.5500 - val_loss: 0.8238 - val_accuracy: 0.6
000
Epoch 5/20
20/20 [==============================] - 9s 474ms/step - loss: 0.8339 - accuracy: 0.5625 - val_loss: 0.7645 - val_accuracy: 0.6
000
Epoch 6/20
20/20 [==============================] - 10s 481ms/step - loss: 0.8401 - accuracy: 0.6000 - val_loss: 0.9460 - val_accuracy: 0.
6000
Epoch 7/20
20/20 [==============================] - 10s 485ms/step - loss: 0.7928 - accuracy: 0.5250 - val_loss: 2.5248 - val_accuracy: 0.
6000
Epoch 8/20
20/20 [==============================] - 9s 459ms/step - loss: 0.7327 - accuracy: 0.6375 - val_loss: 0.9307 - val_accuracy: 0.6
```

Fig 8.8 Model Fitting Through Multiple Epochs

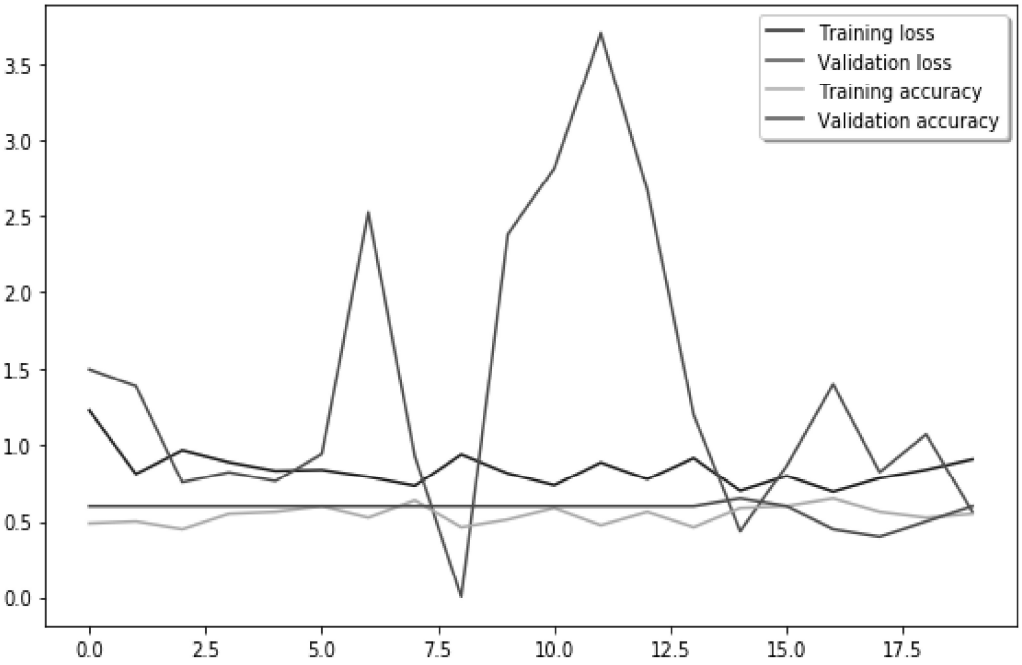

Fig 8.9 Graph Representing Trends in Training and Validation Accuracy

Preparing Test Data

```
test_filenames = os.listdir("dataset/testingset")
test_df = pd.DataFrame({
    'filename': test_filenames
})
nb_samples = test_df.shape[0]
```

```
print(nb_samples)
```

```
32
```

Creating Test Generator

```
test_gen = ImageDataGenerator(rescale=1./255)
test_generator = test_gen.flow_from_dataframe(
    test_df,
    "dataset/testingset",
    x_col='filename',
    y_col=None,
    class_mode=None,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    shuffle=False
)
```

```
Found 32 validated image filenames.
```

Fig 8.10 Preparing Test Dataset

```
color_conc = model.predict_generator(test_generator, steps=np.ceil(nb_samples/batch_size))
```

```
threshold = train_threshold
test_df['color_concentration'] = color_conc
test_df['category'] = np.where(test_df['color_concentration'] > threshold, 1,0)
```

Fig 8.11 Checking of Predict Value Against Train Threshold

```
1 test_df['category'].value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x24da6c7bb38>
```

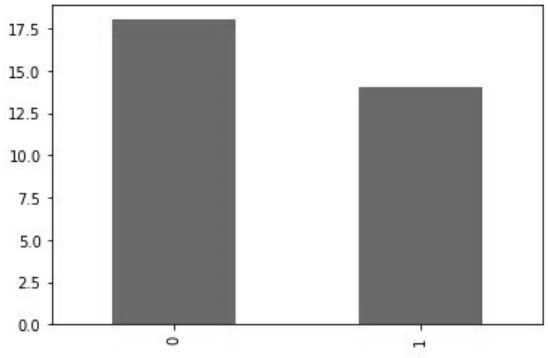

Fig 8.12 Plot Representing Statistics of Recolored And Original Images in Test Set After Evaluating

Category(0):RECOLORED,Category(1):ORIGINAL

	filename	color_concentration	category
0	img.0.jpg	0.792879	1
1	img.1.jpg	0.626490	1
2	img.10.jpg	0.635062	1
3	img.11.jpg	0.123868	0
4	img.12.jpg	0.290647	0
5	img.13.jpg	0.409377	0
6	img.14.jpg	0.357703	0
7	img.16.jpg	0.533659	1
8	img.17.jpg	0.557282	1
9	img.18.jpg	0.199183	0
10	img.19.jpg	0.675524	1
11	img.2.jpg	0.339402	0
12	img.20.jpg	0.161175	0
13	img.21.jpg	0.292490	0
14	img.22.jpg	0.555307	1
15	img.3.jpg	0.618577	1
16	img.30.jpg	0.253419	0
17	img.31.jpg	0.084953	0
18	img.32.jpg	0.485570	1
19	img.33.jpg	0.179625	0
20	img.34.jpg	0.447080	1
21	img.35.jpg	0.163576	0
22	img.36.jpg	0.243802	0
23	img.37.jpg	0.545495	1
24	img.38.jpg	0.359321	0
25	img.39.jpg	0.788567	1
26	img.4.jpg	0.155721	0
27	img.5.jpg	0.686802	1
28	img.6.jpg	0.331426	0
29	img.7.jpg	0.750866	1
30	img.8.jpg	0.404011	0
31	img.9.jpg	0.209784	0

Fig 8.13 Table Representing Test Results

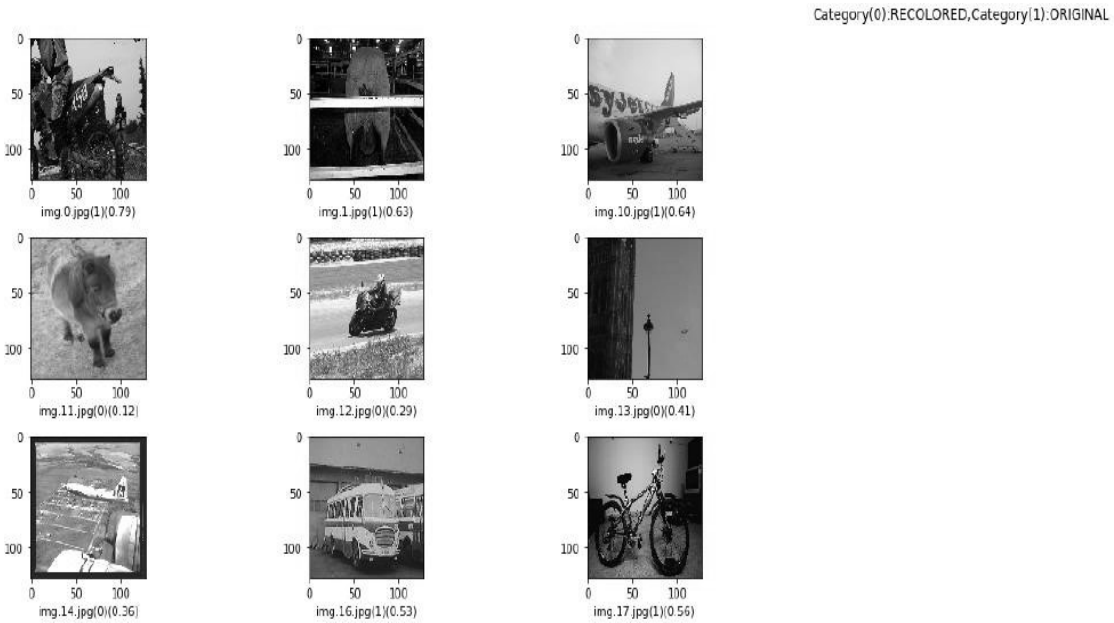

Fig 8.14 Visualising Test Results

9. CONCLUSION

9.1 PROJECT CONCLUSION

In this work, we present a novel deep learning approach for recolored image detection. Both the inter-channel correlation and the illumination consistency are employed to help the feature extraction. We elaborate the design principle of our RecDeNet and systematically validate the rationality by running a number of experiments. Furthermore, two recolored datasets with different sources are created to test the performance of our System and the high performance (8 out of 10 images are identified accurately) of our RecDeNet demonstrates the effectiveness of the model. We hope our simple yet effective RecDeNet will serve as a solid baseline and help future research in recolored images detection. We elaborated the design principle of our System and systematically validated the rationality by running a number of experiments.

9.2 FUTURE ENHANCEMENTS

The training set is comprised of images recolored using one algorithm, hence results of the network may not be very accurate for all recolored images. So, in future the training dataset will have to be enhanced by using images formed using different algorithms. The dependent variables in the training of the system can be increased to improve accuracy.

An application can be built using this network to dynamically upload a picture and know genuineness of image. This network can be attuned to hardware devices so to check a scanned image.

10. BIBLIOGRAPHY

[1] Yanyang Yan, Wenqi Ren, Xiaochun Cao, "Recolored Image Detection via a Deep Discriminative Model," IEEE Transactions On Information Forensics And Security, Vol. XX, No. X, July 2017

[2] VOC PASCAL 2012 Image Dataset

[3] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," IEEE Computer Graphics Applications, vol. 21, no. 5, pp. 34–41, 2001.

[4] M. C. Stamm and K. J. R. Liu, "Forensic detection of image manipulation using statistical intrinsic fingerprints," IEEE Transactions on Information Forensics and Security, vol. 5, no. 3, pp. 492–506, 2010.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Computer Science, 2014

[6] Keras-Sequential Model