CHAPTER 1

INTRODUCTION

In the recent times, the amount of data computers generate has grown exponentially, and the need to store them has become paramount. The industry has started to develop specialised category of softwares for handling what is now called as Big Data. Human intervention for data segregation is no more feasible and efficient for such large scale data. Images are one of the most generated and used type of data and it needs to be handled intelligently as well as efficiently. Hence image segregation is a necessity that software engineers need to address.

Our project aims to segregate image files efficiently using machine learning techniques. It carries out image file segregation based on the similar and dissimilar features present in it. The aim of this project is to map a given set of files based on their similar and dissimilar features i.e. map similar ones nearer and vice versa to the dissimilar ones. The above project falls under the category of system software. Currently we have options like classification of files based on type, date, size and so on. We do not have a system software that classifies the files based on the content.

This is achieved using dimensionality reduction algorithms like T-SNE (t-distributed stochastic neighbour embedding). This falls under Unsupervised Machine Learning wherein the programmer does not manually teach the program what exactly the file contents are, rather the program itself picks certain features and maps it nearer to some files and farther to some other files based on the similarities and dissimilarities.

However in the process, we use convolutional neural networks that needs to be trained (supervised machine learning), but this is only for practical convenience and greater efficiency and not a requirement or a flaw.

In this project we are using image files as one can easily identify how the images have been classified. Images are basically made up of pixels and each pixel has a certain RGB value and a set of pixels making up a part of an image may constitute a feature. And every image is considered as a high dimensional dataset space.

However, passing raw RGB pixel values may not be the most favourable way for dimensionality reduction algorithms. Hence we use a pre trained convolutional neural network of an image classifier. Instead of obtaining the labels for the image(s) we extract values from the penultimate layer which contains feature sets of the image. The feature sets are basically array of floating point values.

This large array of floating points contain the feature sets is provided to the T-SNE algorithm which maps them into lower dimensions as specified which maybe a cluster of images (3D), 2D Map or linear collection(1D). T-SNE returns what is called as embedding. Embedding(s) are basically floating point values which effectively represent the contents of a dataset in a lower dimension.

We need to take into account the fact that reducing the mappings to lower dimensions results in significant loss of spatial information. However the accuracy of mappings can be improved by training the image classifier CNN with larger and more diverse dataset.

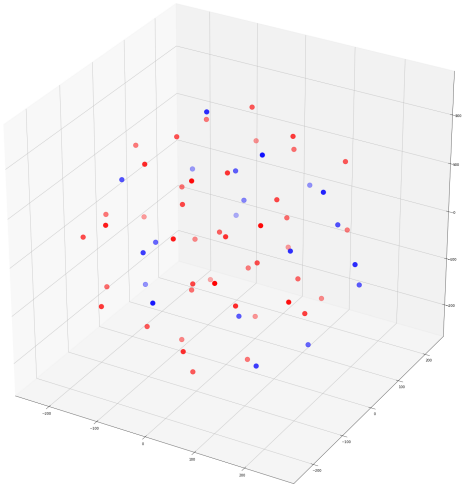

Fig. 1.1 Representation of images a points in 2D plane on a 3D plot

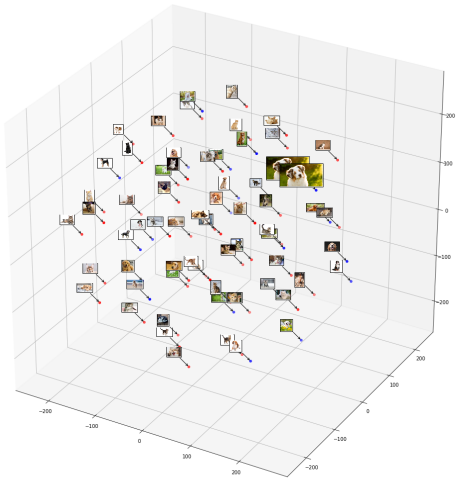

Fig. 1.2 Representation of images in 2D plane 3D plot

OBJECTIVE

- To sort or segregate image files based on their relative similarities and dissimilarities in order to help the users organise their file system in a more systematic way.
- This project aims to change the organisational paradigm of user's file system.
- It enables quick and easy way of sorting large sets of images.
- Machine learning picks up many patterns and features that a person might have missed (colour distribution etc.) i.e. it gives a thorough comparison and segregation.

CHAPTER 2

LITERATURE SURVEY

A literature review is a text of a scholarly paper, which includes the current knowledge including substantive findings, as well as theoretical and methodological contributions to a particular topic.

2.1 Convolutional Neural Network

In machine learning, a convolutional neural network (CNN, or Convent) is a class of deep, feed forward artificial neural networks that has successfully been applied to analysing visual imagery.

CNNs use a variation of multilayer perceptron's designed to require minimal pre-processing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems and Natural language processing.

CNN in this project is used for extracting feature sets from an image. Feature sets identified by a CNN depends on how well and how diversely it has been trained. As mentioned earlier we extract values from the penultimate layer which contains feature sets of the image. The feature sets are basically array of floating point values.

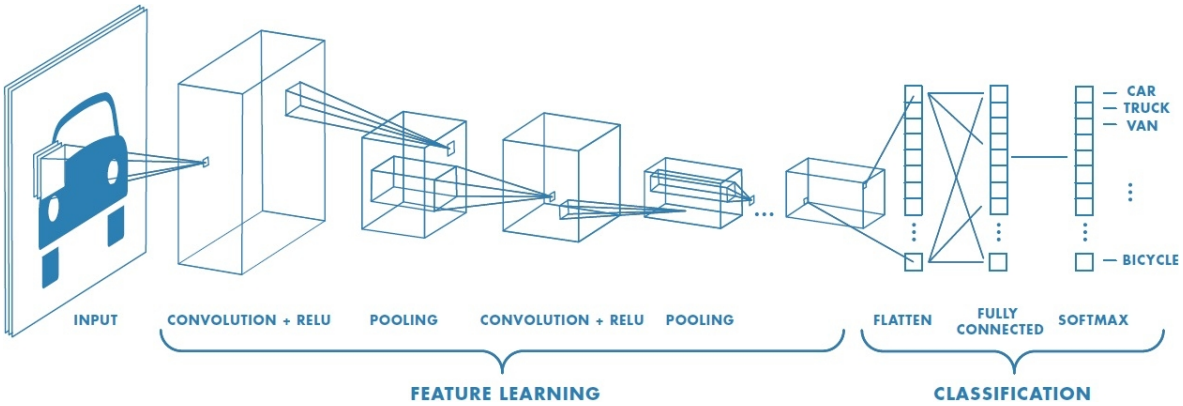

Fig 2.1 Representation of Convolution Neural Network.

2.2 High dimensional dataset

Suppose we have *n* data points and *p* features (attributes for every data point).

It is said to be a high dimensional dataset when p>n, but usually p>>n. Meaning, the number of features per data point is very much more than the number of data points.

The adjoining figure shows a pictorial representation of a high dimensional dataset.

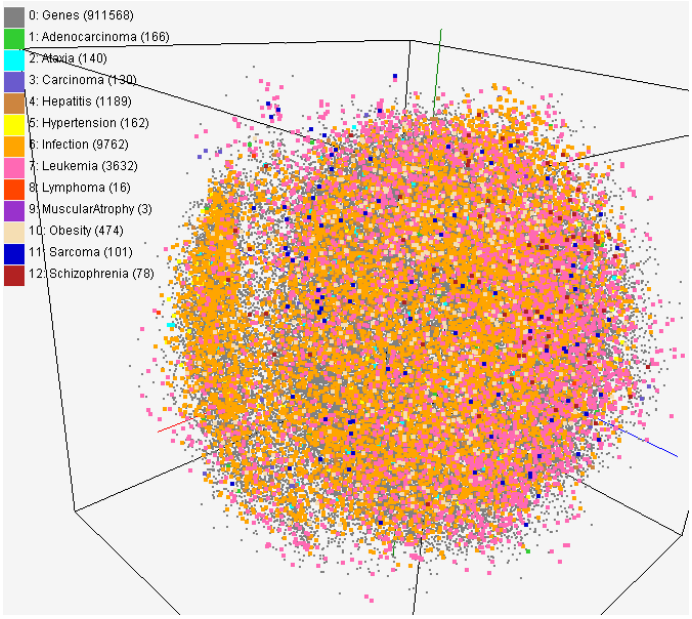

Fig 2.2 High Dimensional Dataset

2.3 t-distributed stochastic neighbour embedding (t-SNE)

It is a machine learning algorithm for dimensionality reduction was developed by Geoffrey Hinton and Laurens van der Maaten. It is a nonlinear dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modelled by nearby points and dissimilar objects are modelled by distant points.

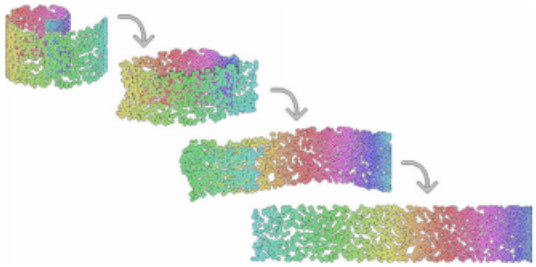

Fig. 2.3 Reducing higher dimension data to lower dimensions

The t-SNE algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked, whilst dissimilar points have an extremely small probability of being picked. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence between the two distributions with respect to the locations of the points in the map. Note that whilst the original algorithm uses the Euclidean distance between objects as the base of its similarity metric, this should be changed as appropriate.

t-SNE has been used in a wide range of applications, including **computer security** research, analysis, cancer, bioinformatics, and biomedical signal processing. It is often used to visualize high-level representations learned by an artificial neural network.

2.4 Caffe Library

Caffe is a machine learning library developed by BVLC (Berkley Vision and Learning Centre) at UC (University of California) Berkeley as a part of their research on machine learning. It is

completely written in C++ and hence prioritize on deployment than research unlike most other machine learning libraries. This library is used for edge case, air gapped (no internet) computer deployments. This library is widely adopted by tech giants like Facebook.

This mini project uses caffe's models for feature extraction. The model used is called bvlc_reference_caffenet. This extracts features i.e. elements, colours and colour gradients and various other entities present in an image that can be used to for deducing similarities and differences between images present in the dataset.

2.5 QT

Qt is a cross-platform application framework that is used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with native capabilities and speed. Qt is currently being developed both by The Qt Company, a publicly listed company, and the Qt Project under open-source governance, involving individual developers and firms working to advance Qt.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

Hardware requirements:

- Architecture: intel i386, x86_64
- Processor: intel core i3
- RAM: 4GB DDR3 SDRAM
- Secondary storage: 512GB SATA
- Graphics: 256 Mb on board graphics or above
- Mouse, Keyboard and LCD Display

Software requirements

- Operating System: Linux x86_64 – 3.16 and above or Windows 7 and above
- Image viewer and File explorer
- Caffe 1.0

CHAPTER 4

DESIGN

The UI consists of two windows one the Main Window, and an Output Window. The Main Window is the window that opens when the user launches this software. The Main Window consists of a User's Manual, 3 Buttons, a Drop down Menu and a List View.

The User's Manual consists of various aspects of the software like

- What Fort is?
- What Fort is not?
- How to use the Software?

3 Buttons are

- Select Image - which allows the user to multiple image files.
- Select Folder - which allows the user to select the folders that contain image files.
- Segregate - this buttons on click, calls the back end and finally displays the output window.

Dropdown Menu - has various options like low, medium, high that is used to define the precision with which the user wants his segregation to be done.

The List View space is used to display all the files selected by the user in a list.

On clicking the segregate button the function extract embedding is called to which the list of selected files have been passed. The extract embedding after the completion of execution returns a vector of vectors which contains a pair of x y coordinates for each image file.
The Output Window is a Dialog which acts as 2-D graph, that is used to display the images, at their respective x y coordinates, obtained based on their feature set.

CHAPTER 5

IMPLEMENTATION

The code in FSORT is completely written in C++ mainly because of its speed, as we have many images to process.

In the front end we have buttons that have certain functionalities

1. Get input files

(I) select image - creates a list of image files selected by the user. Code snippet is as follows.

```
void MainWindow::on_imageFiles_released()
        {
          int i;
                    QStringList temp=QFileDialog::getOpenFileNames(this,"Select Image Files","","Images (*.png *.xpm *.jpg *.JPG *.PNG *.XPM)");
          no_of_files+=temp.size();
          files.append(temp);
          for(i=0;i<temp.size();i++)
          {
             QFileInfo fileinfo(temp.at(i));
             QFileIconProvider iconprovider;
             QIcon icon = iconprovider.icon(fileinfo);
             QStandardItem *s_item=new QStandardItem(icon,temp.at(i));
```

```
            model->appendRow(s_item);
            image_files.push_back((temp.at(i)).toStdString());
        }
        ui->listView->setModel(model);
    }
```

ii.select folder - creates a list of image files from the folder selected by the user.

```
void MainWindow::on_selectFolder_released()
{
    int i;
    QString dir = QFileDialog::getExistingDirectory(this, tr("Open Directory"),
    "Desktop",QFileDialog::ShowDirsOnly | QfileDialog::DontResolveSymlinks);
    QDir directory(dir);
    QStringList temp=directory.entryList(QStringList() << "*.jpg" << "*.JPG" <<"*.png"
    <<"*.PNG" <<"*.xpm" <<"*.XPM",QDir::Files);
    no_of_files+=temp.size();
    for(i=0;i<temp.size();i++)
     {
        files.append(dir+"/"+temp.at(i));
        QFileInfo fileinfo(files.at(i));
        QFileIconProvider iconprovider;
        QIcon icon = iconprovider.icon(fileinfo);
```

```
            QStandardItem *s_item=new QStandardItem(icon,temp.at(i));
            model->appendRow(s_item);
            image_files.push_back((dir+"/"+temp.at(i)).toStdString());
        }
        ui->listView->setModel(model);
}
```

2. DropDown list - selects the level of accuracy - low, medium and high. Where low is assigned 0, medium with 1, and high with 2.

3. Segregate Button - send the list of files selected by user to extract embeddings after returning fom the backend opens a new window, which is the output window. Code snippet is as follows.

```
void MainWindow::on_fsortfunc_released()
{
 int i;
 points=(extract_embeddings(image_files,accuracy_level,false));
 for(i=0;i<no_of_files;i++)
     for(int j=0;j<2;j++)
         points[i][j]*=1000;
 qInfo() << "\n"<<points<<"\nPlotting images..";
 QMap<QString,QVector<double>> map;
 for(i=0;i<no_of_files;i++)
```

```
    map.insert(QString::fromStdString(image_files.at(i)),QVector<double>::fromStdVect
or(points[i]));

  output_window=new Mappings(this,&map);

  output_window->show();

  hide();

}
```

The backend is where the actual processing of the images is done. We have defined various functions breaking down the complete functionalities into small functional units.

Backend

1.Extract Embeddings - this is the function that is called from the Main Window when segregate button is clicked. It receives a list of files, absolute path of the images image, selected by the user as input. This function is wrapper that wraps null character at the end of each file name and then passed it on to extract features function. The function returns a vector of vectors which contains the embeddings i.e. the features represented in terms of floating points, which can be used to draw images at x y. Code snippet is as follows.

```
vector<vector<double>> extract_embeddings(vector<string> image_files,int perplexity,int
threads,bool print_tsnepts,float amplify,bool normalize)
    {
      vector<vector<double>> embeddings;
      if(image_files.size()<1)
        return embeddings;

      ofstream file_list;
      tsne_wrapper tsne_obj;
      file_list.open("file_list.txt");
      for(vector<string>::iterator i=image_files.begin();i!=image_files.end();i++)
```

```
        file_list<<*i<<" 0\n";
    file_list.close();

    cout<<image_files.size()<<" input image files";
    vector<vector<float>>features = extract_features(image_files.size());
    //display_vector_<float>(features);
    if(features.size()<1)
        return embeddings;
    embeddings=tsne_obj.run(features,2,1000,3,perplexity,0.5,amplify,normalize);
    if(print_tsnepts)
        display_vector_<double>(embeddings);
    return embeddings;
}
```

2. Extract Features - this function on receiving the list of files from the extract embeddings function, starts processing the images and creates a Convolutional Neural Network of the images, which has many layers. We are interested in the penultimate layer which contains the feature set with which we are segregating the images. We extract the feature set from the penultimate layer. Pass this to the t-sne function. The unsupervised learning comes in to play at this function where the computer will decide which feature is best to extract in order to segregate the images and extracts only such features. Code snippet is as follows.

```
vector<vector<float>> extract_features(int num_img_files)
{
    int num_mini_batches = num_img_files;
    vector<vector<float>> features_vec;

    const char *
binaryproto="ml_data/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemod
el";
    string feature_extraction_proto("ml_data/imagenet_val.prototxt");
```

```
std::string pretrained_binary_proto(binaryproto);
::google::InitGoogleLogging(binaryproto);

if(num_img_files<1)
{
   LOG(ERROR)<<"oo few files!"<<endl;
   return features_vec;
}
if(!exists("file_list.txt"))
{
   LOG(ERROR)<<"\nfile_list.txt not found!"<<endl;
   return features_vec;
}
if(!exists(binaryproto))
{
   LOG(ERROR)<<"File "<<binaryproto<<" not found!"<<endl;
   return features_vec;
}
LOG(ERROR)<<"Found "<<binaryproto<<endl;
if(!exists(feature_extraction_proto.c_str()))
{
   LOG(ERROR) <<"File "<<feature_extraction_proto<<" not found!"<<endl;
   return features_vec;
}
LOG(ERROR)<<"Found "<<feature_extraction_proto<<endl;

Caffe::set_mode(Caffe::CPU);

LOG(ERROR)<<"Extracting features"<<endl;
```

```
    boost::shared_ptr<Net<float> > feature_extraction_net(new
Net<float>(feature_extraction_proto, caffe::TEST));
    feature_extraction_net->CopyTrainedLayersFrom(pretrained_binary_proto);
    std::string feature_blob_name("fc7");

    if(!feature_extraction_net->has_blob(feature_blob_name))
    {
        LOG(ERROR)<< "Unknown feature name "<< feature_blob_name<<endl;
        return features_vec;
    }

    for (int batch_index = 0; batch_index < num_mini_batches; ++batch_index) {
        feature_extraction_net->Forward();
        const boost::shared_ptr<Blob<float> > feature_blob = feature_extraction_net-
>blob_by_name(feature_blob_name);

        int batch_size = feature_blob->num();
        int dim_features = feature_blob->count() / batch_size;

        const float* feature_blob_data;
        vector<float> batch_fvec;
        for (int n = 0; n < batch_size; ++n)
        {
            feature_blob_data = feature_blob->cpu_data()+feature_blob->offset(n);
            vector<float> dim_fvec {feature_blob_data,feature_blob_data+dim_features};
            batch_fvec.insert(batch_fvec.end(),dim_fvec.begin(),dim_fvec.end());
        }
        features_vec.push_back(batch_fvec);
    }
```

```
    LOG(ERROR)<<"Done extracting
features :"<<features_vec.size()<<"x"<<features_vec[0].size()<<endl<<endl;
    return features_vec;
}
```

3. T-SNE - in this function the algorithm segregates the features based on the obtained floating points (which represent the feature set) from the extract features function. This returns a vector of vectors to extract embeddings which is later on passed to the main function. Code snippet is as follows.

```
vector<vector<double>> tsne_wrapper::run(vector<vector<float>> data,
                                         int op_dims, int max_iters,
                                         int num_threads, double perplexity,
                                         double theta, double amplify,
                                         bool normalize) {
  iters = 0;
  perplexity = 30;
  this->data = data;
  this->op_dims = op_dims;
  this->perplexity = perplexity;
  this->theta = theta;
  this->normalize = normalize;

  samples = data.size();
  input_dims = data[0].size();
  cout << "\n\ntsne wrapper:Received Feature vector of size " << samples << "x"
       << input_dims << "\n";
  if (samples - 1 < 3 * perplexity) {
    cout << "Warning: too few samples for expected accuracy";
    perplexity = (samples - 1) / 3;
```

```
}
if(amplify==0)amplify=1;
cout<<"\nAmplification factor:"<<amplify;

inp_data = (double *)malloc(input_dims * samples * sizeof(double));
op_data = (double *)malloc(op_dims * samples * sizeof(double));

k = 0;
for (i = 0; i < samples; i++)
  for (j = 0; j < input_dims; j++)
    inp_data[k++] = data[i][j];
tsne_run_double(inp_data, samples, input_dims, op_data, op_dims, perplexity,
                theta, 4, max_iters);

if (normalize) {

  if (min_.size() != op_dims)
    min_.resize(op_dims);
  if (max_.size() != op_dims)
    max_.resize(op_dims);
  for (i = 0; i < op_dims; i++) {
    min_[i] = numeric_limits<double>::max();
    max_[i] = numeric_limits<double>::min();
  }
}

tsne_embeddings.clear();
k = 0;
for (i = 0; i < samples; i++) {
  vector<double> tsne_points;
```

```
  tsne_points.resize(op_dims);
  for (j = 0; j < op_dims; j++) {
    tsne_points[j] = amplify*op_data[k++];
    if (normalize) {
      if (tsne_points[j] < min_[j])
        min_[j] = tsne_points[j];
      if (tsne_points[j] > max_[j])
        max_[j] = tsne_points[j];
    }
  }
  tsne_embeddings.push_back(tsne_points);
 }

 // normalize if requested
 if (normalize) {
   for (i = 0; i < tsne_embeddings.size(); i++) {
     for (j = 0; j < op_dims; j++) {
       tsne_embeddings[i][j] =
           (tsne_embeddings[i][j] - min_[j]) / (max_[j] - min_[j]);
     }
   }
 }

 return tsne_embeddings;
}
```

Ouput Window - This window displays the images at their x y coordinates which is obtained from the extract embeddings in a 2D coordinate system. Code snippet is as follows.

```
Mappings::Mappings(QWidget *parent,QMap<QString,QVector<double>> *map) :
QDialog(parent),  ui(new Ui::Mappings)
```

```
{
    ui->setupUi(this);
    //set the scene
    scene = new QGraphicsScene(this);
    ui->graphicsView->setScene(scene);
    //add  background image
    pixmap=new QPixmap(":/Images/Background_image.jpg");
    QGraphicsPixmapItem* BackGround=new QGraphicsPixmapItem(*pixmap);
    scene->addItem(BackGround);
    //add the images at respective x y coordinates
    for(i=map->begin();i!=map->end();i++)
    {
        pixmap=new QPixmap(i.key());
        *pixmap=pixmap->scaled(100,100);
        item=new QGraphicsPixmapItem(*pixmap);
        scene->addItem(item);
        item->setPos(i.value()[0],i.value()[1]);    }}
```

CHAPTER 6

TESTING

Testing is the process of verification and validation of actual output yielded by the software and the expected output for a given input. This helps us determine efficiency and accuracy of the given software.

TESTING CASES:

Sl No	Test case	Description	Expected result	Status
1.	Unit Testing: File Dialog - Images/Folder selection	Checking whether the ui gets the details of the selected images	The function should return list of strings of absolute paths for selected files or the same of all image files in a given directory.	Passed
2.	Unit Testing: Extract Features	Getting a vector of features present in	1D vector of size 4096 of type float	Passed
3.	Unit Testing - Creating 2D Plot	Displaying the image files at the respective coordinates.	Creates a new window with images at their respective coordinates.	Passed
4.	Integration Testing - UI and Back-End Integration	Passing the List of files selected to the Back-End	The function returns a Vector of Vectors having 2 x y points in each vector	Passed.
5.	Validation Testing	Output Window Test	Displays images at (x,y)	Passed.

CHAPTER 7

SCREENSHOTS

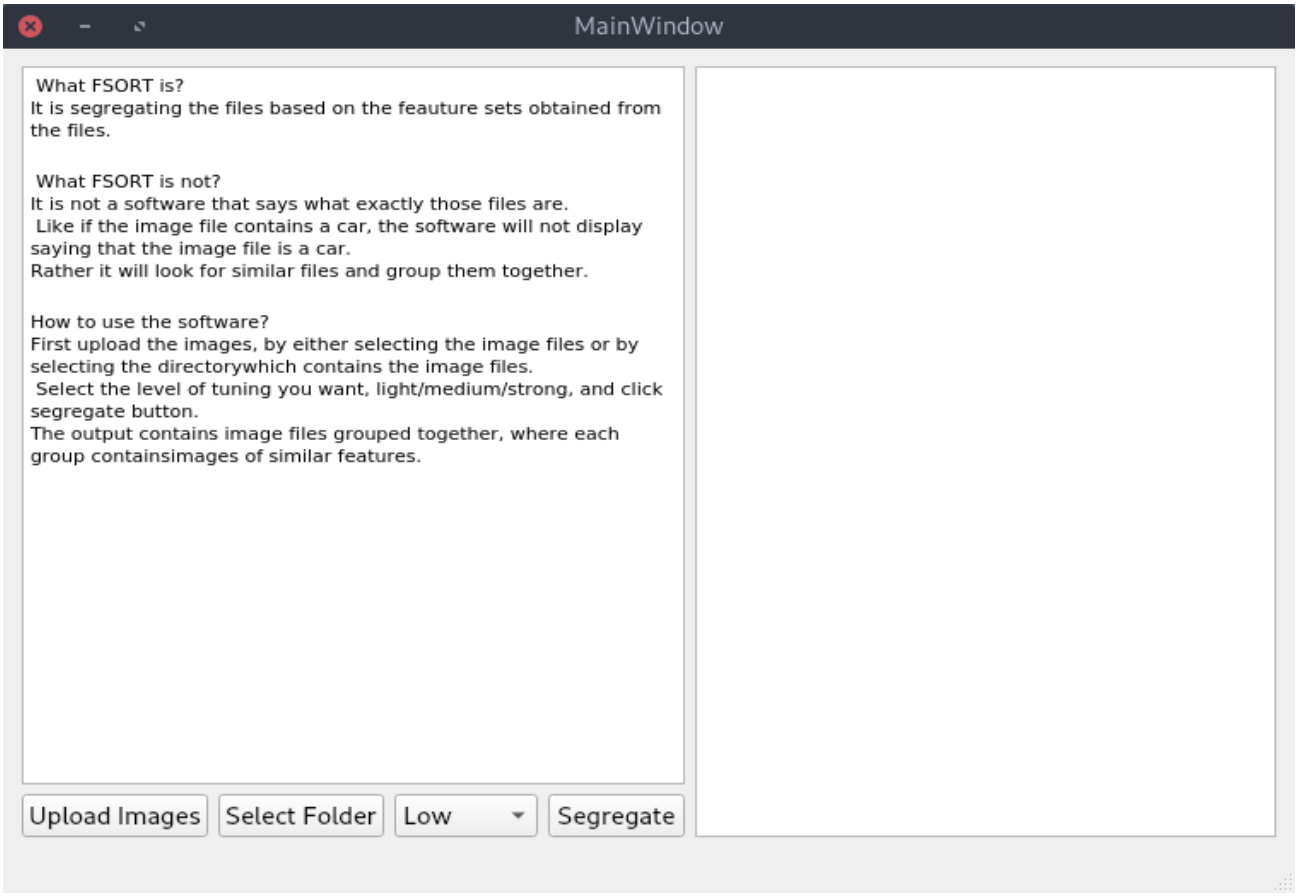

Fig. 7.1 Launch Window

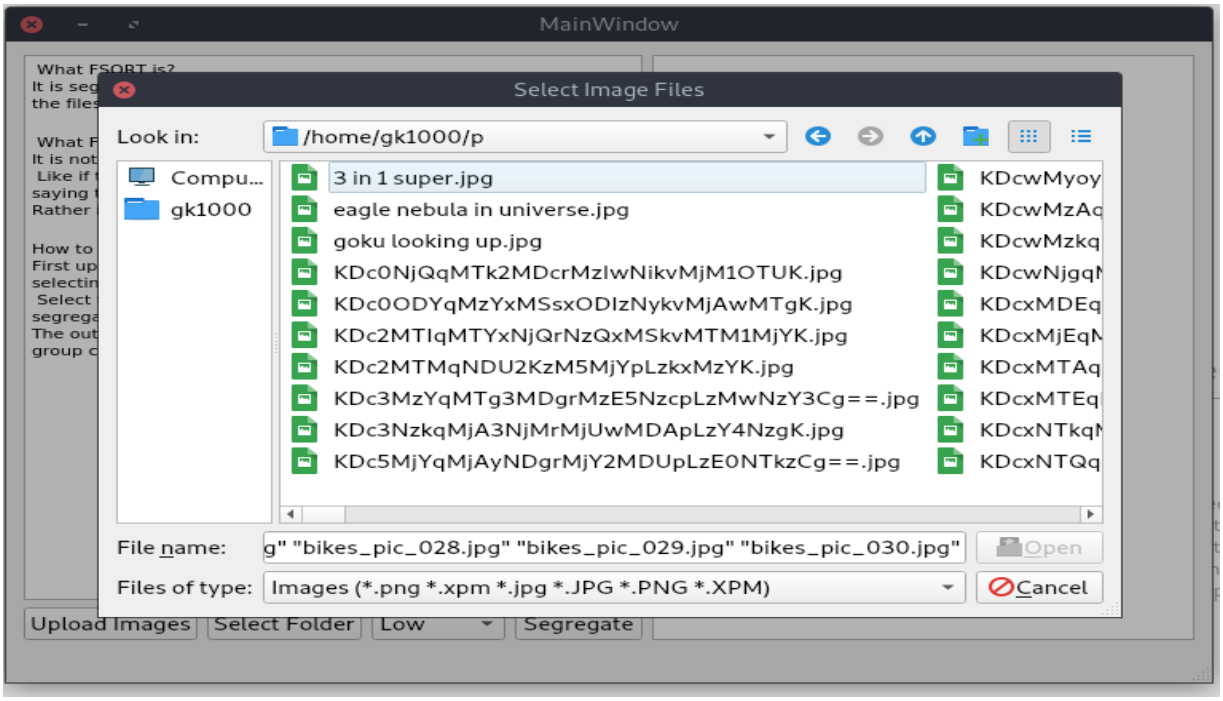

Fig. 7.2 Input Selection – Image Files

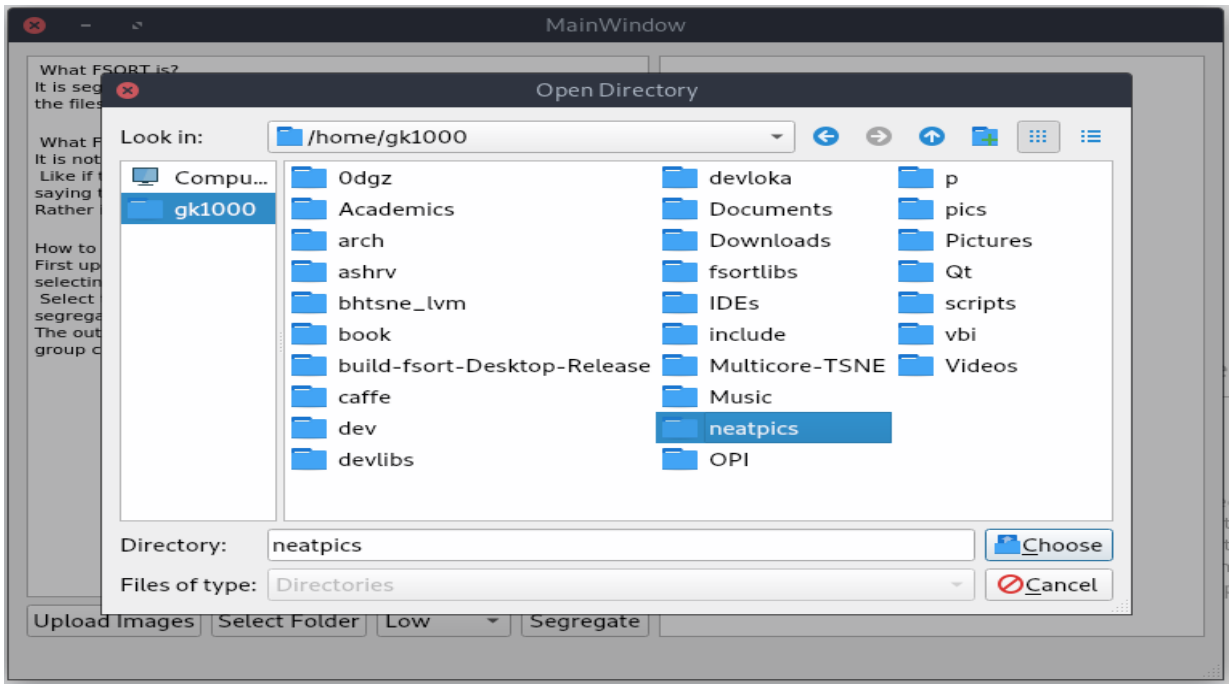

Fig. 7.3 Selection of Folder Containing Images

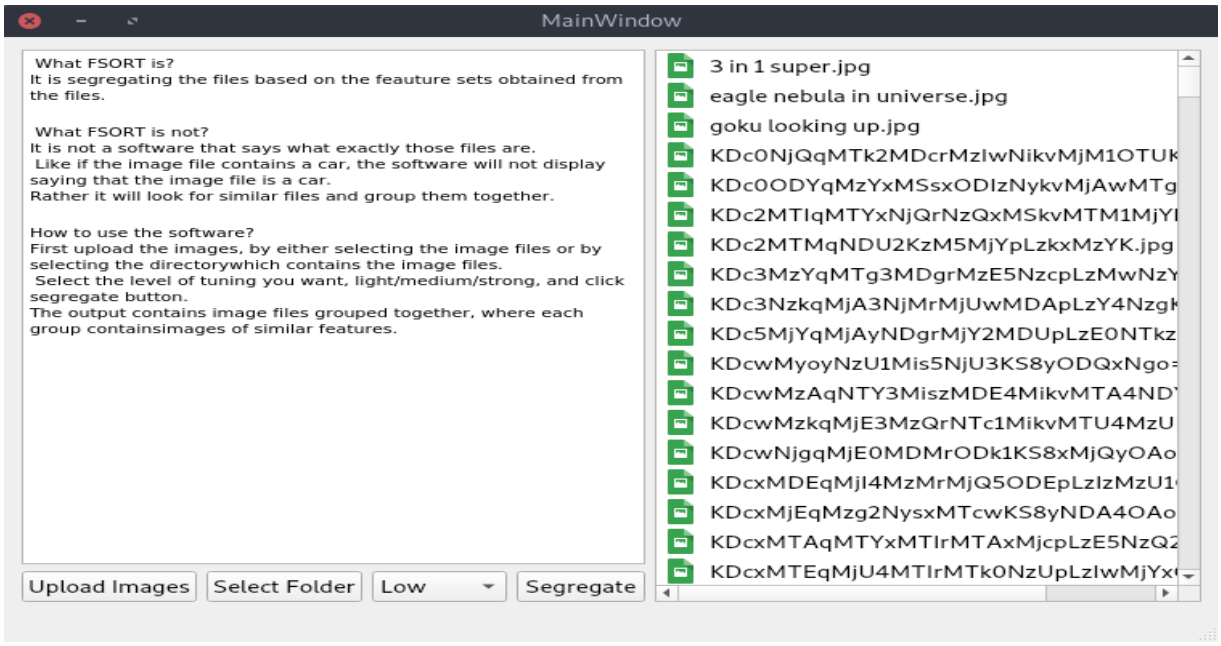

Fig. 7.4 List of Images selected by Users

Fig. 7.5 Feature Extraction Started

```
Starting /home/gk1000/dev/fsort/src/ui/build-fsort-Desktop-Release/fsort...
E0423 21:29:56.771653 24020 feature_extract.cpp:60] Found ml_data/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
E0423 21:29:56.772195 24020 feature_extract.cpp:66] Found ml_data/imagenet_val.prototxt
E0423 21:29:56.772265 24020 feature_extract.cpp:70] Extracting features
E0423 21:30:33.295871 24020 feature_extract.cpp:99] Done extracting features :365x4096
```

Fig. 7.6 Feature Extraction Done

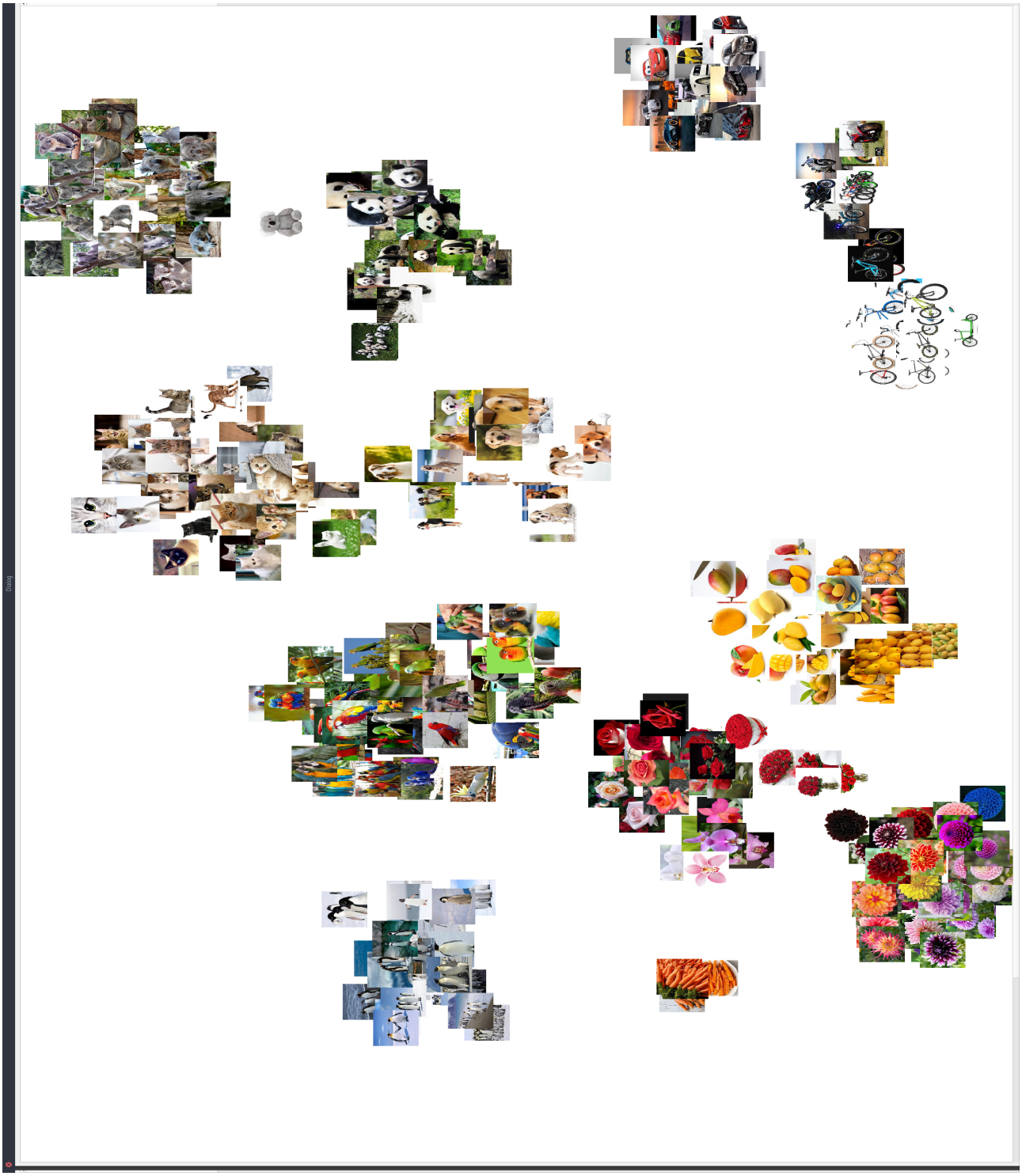

Fig. 7.7 Final Segregation of Images.

APPLICATIONS

In today's world, we are dealing with a lot of data. It takes a toll on, the people dealing with such large amounts of data. Our software is exactly build to solve such problems. We have developed a software that segregates images based on the features rather than the content of the image itself (which is done by most of the image classifiers present in these days). FSORT has various applications in almost every field from space research to health care and many such important fields.

- FSORT can be used to segregate all the pictures clicked by satellites, and group all related images, galaxies, stars and other objects.
- We can use FSORT in Health Care Department, where we can group the scans of similar of diseases together.

CONCLUSION

We have developed a software that segregates files based on the features, which are used to identify similar ones and contrasting ones. We have used various machine learning techniques in order to segregate the images based on features such as Unsupervised Learning. In this software we have implemented the concept only for the image files, we are looking forward to update the software.

In the updated version, we are looking forward include text files, video files apart from image files. In this way we can extend the usage of the files into various other fields. Then we can use this software in order to check plagiarism and other such patent, copy right issues.

REFERENCES

- Caffe C++ Library - http://caffe.berkeleyvision.org
- Machine Learning - https://www.coursera.org/learn/machine-learning
- QT framework - https://www.qt.io