IMAGE PROCESSING WITH DEEP NEURAL NETWORK FOR THE DETECTION OF TUBERCULOSIS

A project phase-II report submitted in the partial fulfillment of the requirements for the award of the degree in
BACHELOR OF TECHNOLOGY

By

K.Kiran Kumar (U14IT502)

(Declared u/s 3 of UGC Act, 195E

Bharath UNIVERSITY
173, Agaram Road, Selaiyur, Chennai-600073

Bharath Institute of Science and Technology

Department of Information Technology

Chennai-73

APRIL-2018

(Declared u/s 3 of UGC Act, 1956

Bharath UNIVERSITY
பாரத் பல்கலைக்கழகம்

(Declared Under Section 3 of UGC Act, 1956)

BHARATH INSTITUTE OF SCIENCE AND TECHNOLOGY

Selaiyur, Chennai-73

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project phase II report titled "**IMAGE PROCESSING WITH DEEP NEURAL NETWORK FOR THE DETECTION OF TUBERCULOSIS"** is the Bonafide work of **K.Kiran Kumar (U14IT502)** are IV year B.Tech (IT) who carried out the research under my supervision.

Project Guide

Mr.S.Thirunavukkarasu
Assistant Professor,
Department of IT,
Bharath University.

Head of the Department

DR.A.KUMARAVEL,
Professor and Head,
Department of IT,
Bharath University.

Internal Examiner

External Examiner

DECLARATION

I hereby declare that to the best of my knowledge the work reported here in does not from part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Name & Roll Number:
K.Kiran Kumar
U14IT502

Signature

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our respected Chancellor **Dr.J.SUNDEEP ANAND** and Managing Director **DR.SWETHA ANAND** for their valuable support and encouragement.

We take great pleasure in expressing our sincere thanks to our Vice chancellor **DR.M.PONNAVAIKKO** for backing us in this project.

We take great pleasure in expressing our sincere thanks to our Pro Vice chancellor **Dr.KANAGASABAI** for backing us in this project

We thank our Director **Mr.S.THYAGARAJAN** and **Dr.M.HAMMID HUSSIN** Dean of Engineering for providing sufficient facilities for the completion of this project.

We express our sincere thanks to our Head of the Department **Dr.A.KUMARAVEL,** for his valuable support and encouragement.

We would express our sincere thanks to our Project Guide **Mr.S.Thirunavukkarasu** for her encouragement to work on this project.

I wish to express my thanks to all the **STAFF MEMBERS** to the Dept. of IT, for their valuable support throughout this project. I also thank my **FAMILY AND FRIENDS** for their moral support and suggestion for this work.

Finally, I thank one and all those who have rendered help directly or indirectly at various stages of the project.

TABLE OF CONTENTS

DETECTION OF TUBERCLUSOSIS USING DEEP NEURAL NETWORK FOR IMAGE PROCESSING

ABSTRACT

Image segmentation is one of the most important tasks in medical image analysis and is often the first and the most critical step in many clinical applications. In brain MRI analysis, image segmentation is commonly used for measuring and visualizing the brain's anatomical structures, for analysing brain changes, for delineating pathological regions, and for surgical planning and image-guided interventions. In the last few decades, various segmentation techniques of different accuracy and degree of complexity have been developed and reported in the literature. In this paper we review the most popular methods commonly used for brain MRI segmentation. We highlight differences between them and discuss their capabilities, advantages, and limitations. To address the complexity and challenges of the brain MRI segmentation problem, we first introduce the basic concepts of image segmentation. Then, we explain different MRI pre -processing steps including image registration field correction, and removal of nonbrain tissue. Finally, after reviewing different brain MRI segmentation methods, we discuss the validation problem in brain MRI segmentation. mage segmentation has long been an important focus of research, yielding many automated methods to perform segmentation of different organs in the human body. Brain image segmentation is an important but inherently difficult problem in medical image processing. In general, it cannot be solved using straightforward, conventional image processing techniques. Among medical images, segmentation of brain images is a challenging problem that has received an enormous amount of attention recently, and this is because proper diagnosis of brain disorders greatly depends upon accurate segmentation of the complex structure of the brain.

CHAPTER 1

INTRODUCTION

Over the last few decades, the rapid development of noninvasive brain imaging technologies has opened new horizons in analyzing and studying the brain anatomy and function. Enormous progress in accessing brain injury and exploring brain anatomy has been made using magnetic resonance imaging (MRI). The advances in brain MR imaging have also provided large amount of data with an increasingly high level of quality. The analysis of these large and complex MRI datasets has become a tedious and complex task for clinicians, who have to manually extract important information. This manual analysis is often time-consuming and prone to errors due to various inter- or intra-operator variability studies. These difficulties in brain MRI data analysis required inventions in computerized methods to improve disease diagnosis and testing. Nowadays, computerized methods for MR image segmentation, registration, and visualization have been extensively used to assist doctors in qualitative diagnosis.

Brain MRI segmentation is an essential task in many clinical applications because it influences the outcome of the entire analysis. This is because different processing steps rely on accurate segmentation of anatomical regions. For example, MRI segmentation is commonly used for measuring and visualizing different brain structures, for delineating lesions, for analyzing brain development, and for image-guided interventions and surgical planning. This diversity of image processing applications has led to development of various segmentation techniques of different accuracy and degree of complexity.

In this paper we review the most popular methods commonly used for brain MRI segmentation. We highlight differences between them and discuss their capabilities, advantages, and limitations. To introduce the reader to the complexity of the brain MRI segmentation problem and address its challenges, we first introduce the basic concepts of image segmentation. This includes defining 2D and 3D images, describing an image segmentation problem and image features, and introducing MRI intensity distributions of the brain tissue. Then, we explain different MRI preprocessing steps including image registration, bias field correction, and removal of nonbrain tissue. Finally, after reviewing different brain MRI segmentation methods, we discuss the validation problem in brain MRI segmentation.

CHAPTER 2

LITERATURE SURVEY

IMAGE PROCESSING FOR MEDICAL PURPOSES USING ALEX NET:

Pathology identification is performed by the image classification technique and then the treatment is planned based on the nature of abnormality. After treatment, it is highly essential to estimate the response of the patient to the treatment. In case of brain tumour abnormalities, the size of the tumour may decrease which indicates a positive effect and sometimes it may increase which shows a negative effect. In any case, it is important to perform a volumetric analysis on MR brain tumour images. Image segmentation covers this objective by extracting the abnormal portion from the image which is useful for analysing the size and shape of the abnormal region. This method is also called as "pixel based classification" since the individual pixels are clustered unlike the classification techniques which categorizes the whole image. Several research works are reported in the area of medical image segmentation. All the research works performed on image segmentation can be classified into two broad categories: (a) Non-AI techniques 18 and (b) AI techniques. Initially, a survey is performed on Non-AI techniques followed by the report on AI techniques.

2.1 Image segmentation based on AI techniques:

A model based on tumour segmentation technique was implemented by Nathan et al (2002). Modified Expectation Maximization (EM) algorithm is used in this work to differentiate the healthy and the timorous tissues. A set of tumour characteristics are presented in this paper which is highly essential for accurate segmentation. But the drawback of this work is the lack of quantitative analysis on the extracted tumour region. It have developed a level set method based tumour segmentation technique. This method involves the method of boundary detection with the seed point. Water-shed algorithm is also used to capture the weak edges. The main problem of this approach is the selection of seed point. Random selection of seed point may lead to inappropriate results and also consumes large convergence time period. A complete analysis of various types of brain tumours and the effect of MR image segmentation techniques on the treatment is studied by Sundeep et al (2006). The report concluded that the enhanced MR image segmentation techniques play a major role for brain tumour treatment. This study shows the requirement for an accurate and quick image segmentation technique. Habib et al (2006) have elaborated the merits and demerits of various statistical segmentation techniques. This work analysed the performance measures of histogram based method, EM technique and the Statistical Parameter Mapping (SPM2) package

in detail. This report aimed at differentiating four different types of brain tissues. Experiments are carried out on simulated brain images. But the statistical techniques fail in the case of large deformations. An enhanced version of symmetry analysis which also incorporates the deformable models is reported by Hassan et al (2007). The segmentation efficiencies reported in this approach is 19 very low and the report also concluded that the proposed approach is a failure in case of symmetrical tumour across the mid-sagittal plane. Mathematical morphology based segmentation is implemented by Abdul (2007). The experimental results suggested the usage of Skeleton by Influence Zones detection (SKIZ) for brain image segmentation. This technique involves the initialization of several parameters which is one of the demerits of this approach. henry (2007) have revealed the various software available for medical image segmentation. This work also suggested appropriate techniques for various types of segmentation methods. An analysis on evaluation procedure for segmented images is performed by Ranjith et al (2007). Few conventional algorithms such as EM algorithm mean shift filtering algorithm, etc. are experimented in this work. But the comparative analysis between various performance measures is not reported in this work. Pierre et al (2007) have implemented a topology preserving tissue classification on MR brain images. The advantages of statistical techniques and image registration are combined in this technique which is also suitable for noisy images. But the requirement for high computational time period is the major drawback of this approach. Symmetry based brain tumour extraction is performed by Nilan(2008). The ability of this approach is limited since it can detect only the densely packed tumour tissues. Chin(2008) have demonstrated the application of pseudo-conditional random fields for brain tumour segmentation. This technique is implemented on images of different tumour size. This system also claimed to be highly accurate and much faster than other conventional techniques. The mode of training used in this approach is patient - specific training which is one of the limitations of this technique. Jason et al (2008) have implemented a Bayesian model based tissue segmentation technique for tumour detection. This method proved to be computationally efficient besides yielding improved results over the conventional techniques. K-Nearest Neighbour technique based MR brain image classification is performed by Petronella et al (2008). An extensive comparative analysis is 20 performed with other techniques. The dependency on threshold values for accurate output is the drawback of this approach. A volumetric image analysis based on mesh and level set method is illustrated by Alouietal (2009). This work concentrated on 3D image processing and employed on different tumour types. But the results yielded by this approach are inferior and not comparable to the other pixel based classifiers. Zhen et al (2009) have performed a survey on various medical image segmentation algorithms and analysed the merits and demerits of these techniques. The drawbacks of several techniques are clearly illustrated in this report and also suggested suitable techniques for tumour segmentation.

2.2 Image processing based on Artificial Deep Neural Networks:

Another group of researchers depend on AI techniques for brain image segmentation. Among the AI techniques, ANN and Fuzzy theory are the predominantly used methodologies for segmentation. ANN is preferred by the researchers because of its adaptive nature, accuracy, etc. Java1997) have demonstrated the application of Learning Vector Quantization (LVQ) for brain image segmentation. A comparative analysis is performed with the Back Propagation Network (BPN) and the experimental results proved the superior nature of LVQ. The report also concluded that the ANN is faster than the conventional classifiers. Constantino et al (2000) have implemented Self Organizing Map based segmentation technique on MR brain images. Though the proposed system is faster, the segmentation efficiency is comparatively low since it employs unsupervised mode of training. Automated brain image segmentation using conventional LVQ is proposed by Carlos et al (2003). The convergence rate of this approach is high but this system failed to distinguish the outer layers of brain which is normally seen in MR brain images. Also the results are not quantitatively analysed in this report. Raquel et al (2004) have developed a combined Radial Basis Function neural network (RBF) and contour model based MR image segmentation technique. The contour model is used as a pre-segmentation step by developing the clear boundaries between the different tissues. RBF neural network is then used to segment the various brain 21 tissues into different groups. A modified version of SOM with Markov random field model is suggested by Yane(2005). Extra spatial constraints are added in this algorithm for weight adjustment between the input layer and the output layer. But this method is highly prone to noise and applicable for only noise-free images. Back propagation neural network based tumour segmentation is performed by Martin et al (2006). A comparative analysis is done with the Inverse Laplace Transform based technique. Sylvian et al (2006) have developed a modified Bidirectional associative memory for grey level pattern classification. A self-convergent iterative learning is performed in this approach with a non-linear function. This technique is more applicable for noisy images. But this approach suffers from the drawback of handling non linear separable problems. A fast neural network suitable for real time applications is implemented by Guang-Binetal (2006). An iterative-free training approach is followed in this network using the Huang's neural network. The convergence time period is considerably reduced since the weights are determined analytically rather than through conventional weight adjustment procedure. Lange et al (2006) have presented a comparative study between the neural classifiers and another AI technique such as fuzzy classifier. SOM is used as a neural representative and fuzzy n-means algorithm is used as the fuzzy representative. Experimental results revealed the inability of SOM to detect small tumour areas. A modified neural network with neighbourhood information is proposed by Siddhartha et al (2008). This work aimed at minimizing the false classifications of the images. The efficiency of this approach is revealed through experimental results and the report also revealed the applicability of this method for binary, multilevel and colour intensity images. Weietal (2008) analysed the functional MRI (fMRI) data using the SOM architecture. clustering technique has been incorporated in this approach to yield better results. An enhanced version of LVQ neural network is implemented by Jinn-Yi et al (2008). The concept of GA is incorporated in this technique to improve the performance of 22 conventional LVQ. An analysis in terms of segmentation efficiency

and convergence time period is provided in the report. (2009) have developed a hybrid expert system based SOM for interpretation of MR brain images. This system proved to be much superior than the individual ANN since the accuracy is increased to a higher extent. This work also highlighted the application of neuro-fuzzy approach for high quality results. But the lack of availability of an expert knowledge database for all the applications is the major drawback of this system. A novel neural network such as Incremental Supervised Neural Network (ISNN) is proposed by Zafer et al (2010). This method depends on Continous Wavelet Transform (CWT) and Zernike moments for the analysis of six different types of brain tissues. Experimental results suggested the applicability of this network for noisy environment. Jaya et al (2011) have used the pixel similarity based technique for tumour segmentation in abnormal MR brain images. Kohonen neural network has been used for tumour segmentation by Riadetal (2011). But, the level of misclassification rate reported in this work is significantly high. SOM based brain image segmentation methodologies are also reported by Shi et al (2010) and Valarmathi (2011). An improved version of SOM based brain image segmentation is implemented by Logeswari(2010). Another modified approach such as Echo State Neural Network (ESNN) is used for brain image segmentation is reported by Suganthi (2011).

CHAPTER 3

OBJECTIVES OF THE PROJECT

Objectives: for tuberclosis

Tuberculosis is a very deadly disease if not diagnosed in the early stage and hence here we aim to speeden the process of diagnosis

- Size of the extent
- Location of the tuberculosis
- Presence of swell
- Overall health, and medical history

3.1 Problem Statement:

- Diagnosis of tuberculosis from xray scan:

If patients with primary tuberculosis undergo imaging, a conventional chest radiograph may be sufficient for diagnosis in the appropriate clinical setting. (A case of primary pulmonary tuberculosis is depicted in the image below.) In patients with progressive primary or postprimary tuberculosis, computed tomography scanning is often performed, in addition to chest radiography. Magnetic resonance imaging may be used to evaluate complications of thoracic disease, such as the extent of thoracic wall involvement with empyema, but is of limited value in the evaluation of patients with pulmonary tuberculosis.
Concerning characterization of the infection as active or not, CT is more sensitive than radiography, and fluorodeoxyglucose positron emission tomography/CT (FDG PET/CT) has yielded promising results.

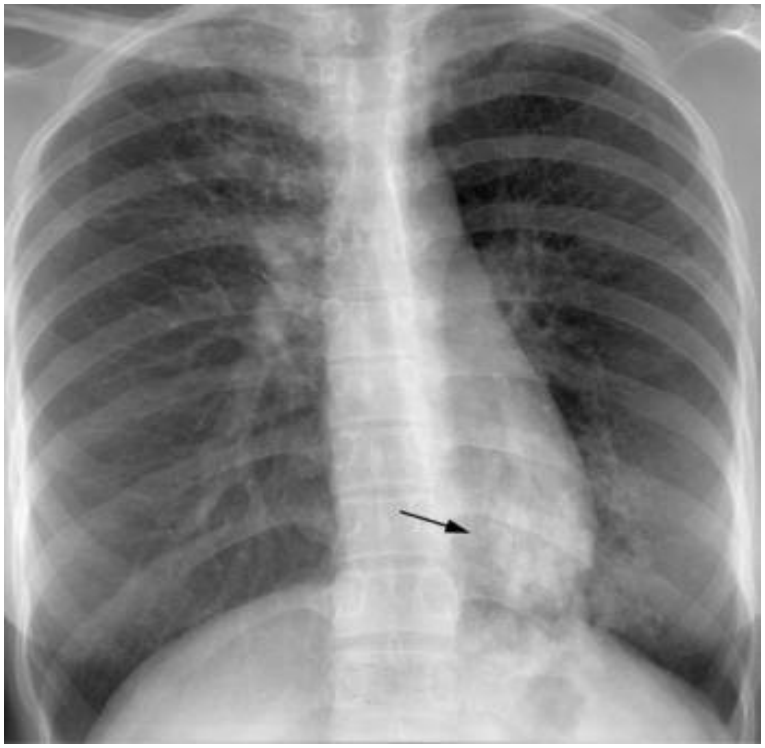


FIG 1: X-RAY IMAGE SHOWING TUBERCULOSIS

- Alexnet being a deep neural network, has 25 convolution layers. integrating the image processing for X- ray scans of tuberclusis with alexnet increases the speed, efficiency and accuracy of the disease being disgnosed in a patient.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 Hardware Specifications:

4.1.1 Computer machine: One

4.1.2 Hard Disk: Preferably SSD (Solid state Drive)

4.1.3 Ram: 8GB

4.1.4 Processor: 3.5 GHz

4.1.5 GPU: Nvidia GeForce GTH 1050 Ti with 768 CUDA CORES

4.2 Software Specifications:

4.2.1 MATLAB 9.2.0.538062(R2017a)
4.2.2 Windows Operating System 10
4.2.3 CUDA 9.1 (for deep network training on GPU)

TECHNOLOGIES USED
MATLAB 9.2.0.538062 (R2017a)

The MATLAB ® high-performance language for technical computing integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization Scientific and engineering graphics

- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. It allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. You can add on toolboxes for signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many other areas.

INTRODUCTION TO MATLAB 9.2.0.538062 (R2017a)

The MATLAB® System

The MATLAB system consists of these main parts:

Desktop Tools and Development Environment

This is the set of tools and facilities that help you use and become more productive with MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, a code analyzer and other reports, and browsers for viewing help, the workspace, files, and the search path.

Mathematical Function Library

This is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

The Language

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create large and complex application programs.

Graphics

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

External Interfaces

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), for calling MATLAB as a computational engine, and for reading and writing MAT-files.

WORKING OF MATLAB® Session Starting a MATLAB® Session

On Microsoft® Windows® platforms, start the MATLAB® program by double-clicking the MATLAB R2014a shortcut on your Windows desktop.

On Apple® Macintosh® platforms, start MATLAB by double-clicking the MATLAB 7.6 icon on your Macintosh desktop.

On The Open Group UNIX® platforms, start MATLAB by typing matlab at the operating system prompt.

There are alternative ways to start MATLAB on Windows and Macintosh platforms, and you can customize MATLAB startup. For example, you can change the directory in which MATLAB starts or automatically execute MATLAB statements upon startup.

For More Information See Startup and Shutdown in the Desktop Tools and Development Environment documentation.

The Desktop

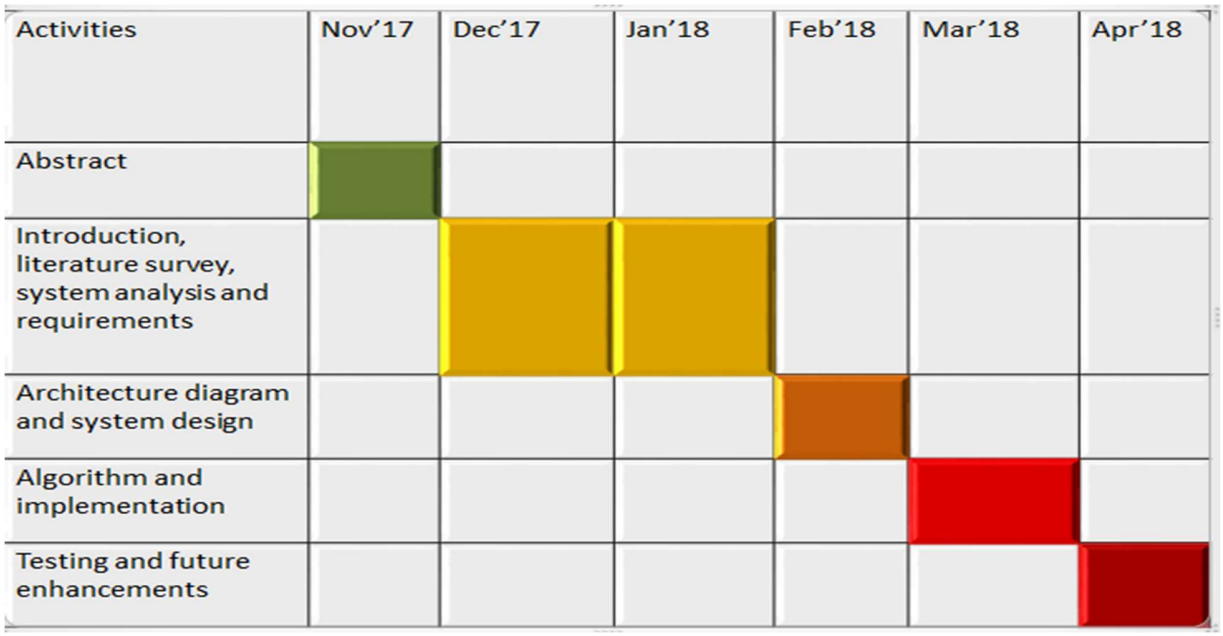
When you start MATLAB, the desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The following illustration shows the default desktop. You can customize the arrangement of tools and documents to suit your needs. For more information about the desktop tools, see Desktop Tools and Development Environment.

Quitting the MATLAB® Program

To end your MATLAB session, select File > Exit MATLAB in the desktop, or type quit in the Command Window. You can run a script file named finish.m each time MATLAB quits that, for example, executes functions to save the wor

4.3Gantt chart(figure 2)



CHAPTER 5

EXISTING SYSTEM VS PROPOSED SYSTEM

In the existing system, the training and processing of data for a deep neural network is done over the CPU. But in a proposed system, the training and processing of data is done over the GPU.

Deep learning involves huge amount of matrix multiplications and other operations which can be massively parallelized and thus sped up on GPU-s.

A single GPU might have thousands of cores while a CPU usually has no more than 12 cores. Although GPU cores are slower than CPU cores, they more than make up for that with their large number and faster memory if the operations can be parallelized. Sequential code is still faster on CPUs.

CPUs are designed for more general computing workloads. GPUs in contrast are less flexible, however GPUs are designed to compute in parallel the same instructions. Deep Neural Networks (DNN) are structured in a very uniform manner such that at each layer of the network thousands of identical artificial neurons perform the same computation. Therefore the structure of a DNN fits quite well with the kinds of computation that a GPU can efficiently perform.

GPUs have additional advantages over CPUs, these include having more computational units and having a higher bandwidth to retrieve from memory. Furthermore, in applications requiring image processing (i.e. Convolution Neural Networks) GPU graphics specific capabilities can be exploited to further speed up calculations.

The primary weakness of GPUs as compared to CPUs is memory capacity on GPUs are lower than CPUs. The highest known GPU contains 24GB of RAM, in contrast, CPUs can reach 1TB of RAM. A secondary weakness is that a CPU is required to transfer data into the GPU card. This takes place through the PCI-E connector which is much slower than CPU or GPU memory. One final weakness is GPU clock speeds are 1/3rd that of high end CPUs, so on sequential tasks a GPU is not expected to perform comparatively well.

In summary, GPUs work well with DNN computations because (1) GPUs have many more resources and faster bandwidth to memory (2) DNN computations fit well with GPU architecture. Computational speed is extremely important because training of Deep Neural Networks can range from days to weeks. In fact, many of the successes of Deep Learning may have not been discovered if it were not for the availability of GPUs.

CHAPTER 6

ARCHITECTURAL DESIGN SPECIFICAITION

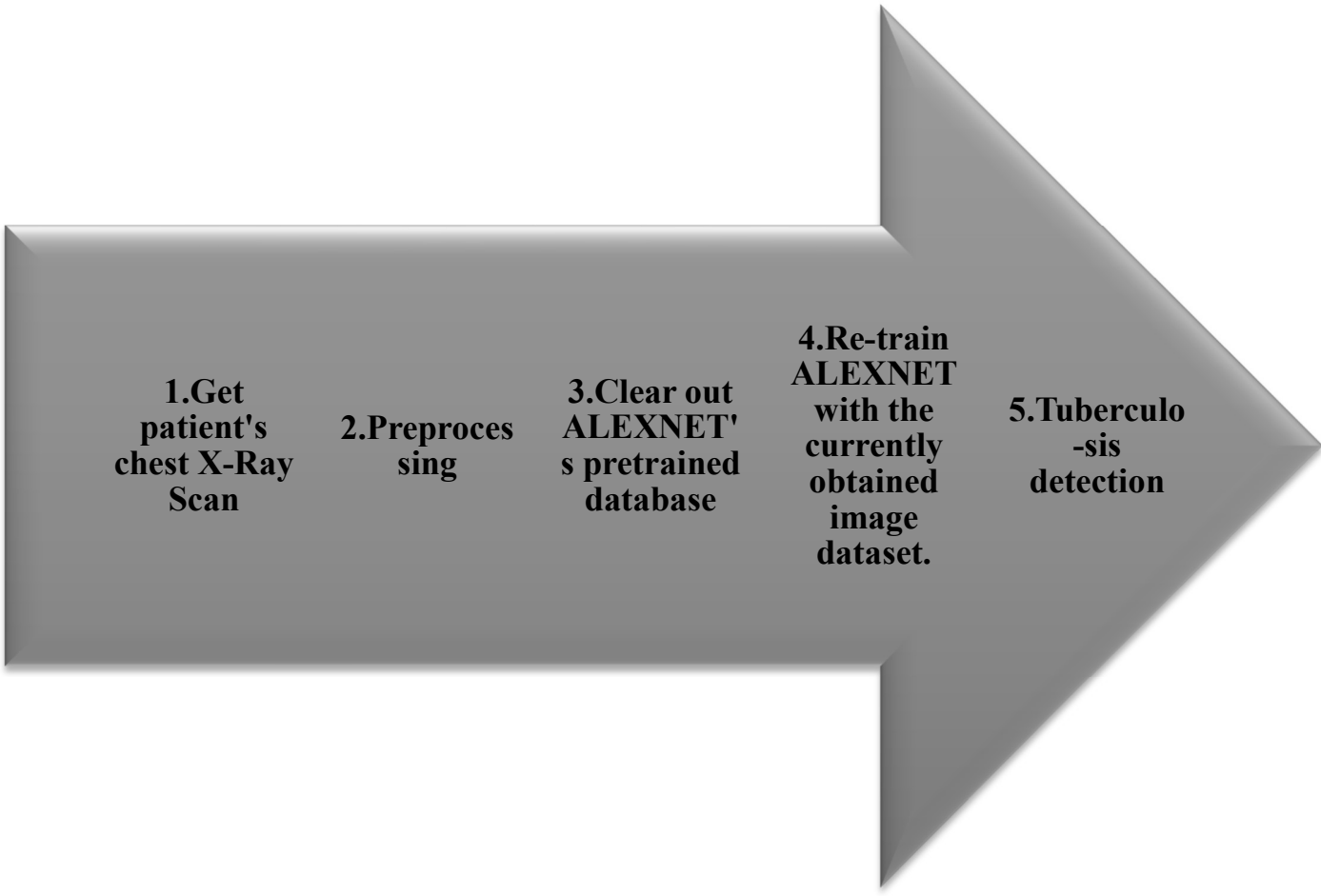


FIGURE 3:

CHAPTER 7

ALEXNET

Syntax

net = alexnet

Description

net = alexnet returns a pretrained AlexNet model. This model is trained on a subset of the ImageNet database , which is used in ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) . The model is trained on more than a million images and can classify images into 1000 object categories. For example, keyboard, mouse, pencil, and many animals. As a result, the model has learned rich feature representations for a wide range of images.

This function requires Neural Network Toolbox™ Model for AlexNet Network support package. If this support package is not installed, the function provides a download link.

Examples

Type alexnet at the command line.

alexnet

If Neural Network Toolbox Model for AlexNet Network support package is not installed, then the function provides a link to the required support package in the Add-On Explorer. To install the support package, click the link, and then click Install. Check that the installation is successful by typing alexnet at the command line.

alexnet

ans =

SeriesNetwork with properties:

Layers: [25×1 nnet.cnn.layer.Layer]

If the required support package is installed, then the function returns a SeriesNetwork object.

Load Pretrained AlexNet Convolutional Neural Network

Load a pretrained AlexNet convolutional neural network and examine the layers and classes.

Load the pretrained AlexNet network using alexnet. The output net is a SeriesNetworkobject.

net = alexnet

net =

SeriesNetwork with properties:

Layers: [25×1 nnet.cnn.layer.Layer]

Using the Layers property, view the network architecture. The network comprises of 25 layers. There are 8 layers with learnable weights: 5 convolutional layers, and 3 fully connected layers.

net.Layers

ans =

25x1 Layer array with layers:

1 'data' Image Input 227x227x3 images with 'zerocenter' normalization
2 'conv1' Convolution 96 11x11x3 convolutions with stride [4 4] and padding [0 0]
3 'relu1' ReLU ReLU
4 'norm1' Cross Channel Normalization cross channel normalization with 5 channels per element
5 'pool1' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0]
6 'conv2' Convolution 256 5x5x48 convolutions with stride [1 1] and padding [2 2]
7 'relu2' ReLU ReLU
8 'norm2' Cross Channel Normalization cross channel normalization with 5 channels per element
9 'pool2' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0]
10 'conv3' Convolution 384 3x3x256 convolutions with stride [1 1] and padding [1 1]
11 'relu3' ReLU ReLU
12 'conv4' Convolution 384 3x3x192 convolutions with stride [1 1] and padding [1 1]
13 'relu4' ReLU ReLU
14 'conv5' Convolution 256 3x3x192 convolutions with stride [1 1] and padding [1 1]
15 'relu5' ReLU ReLU
16 'pool5' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0]
17 'fc6' Fully Connected 4096 fully connected layer
18 'relu6' ReLU ReLU
19 'drop6' Dropout 50% dropout

```
   20   'fc7'      Fully Connected             4096 fully connected layer
   21   'relu7'    ReLU                        ReLU
   22   'drop7'    Dropout                     50% dropout
   23   'fc8'      Fully Connected             1000 fully connected layer
   24   'prob'     Softmax                     softmax
   25   'output'   Classification Output       crossentropyex with 'tench', 'goldfish', and 998 other
classes
```

You can view the names of the classes learned by the network by viewing the ClassNames property of the classification output layer (the final layer). View the first 10 classes by selecting the first 10 elements.

```
net.Layers(end).ClassNames(1:10)

ans =

  1×10 cell array

  Columns 1 through 4

    'tench'    'goldfish'    'great white shark'    'tiger shark'

  Columns 5 through 9

    'hammerhead'    'electric ray'    'stingray'    'cock'    'hen'

  Column 10

    'ostrich'
```

Classify an Image Using AlexNet

Read, resize, and classify an image using AlexNet. First, load a pretrained AlexNet model.

```
net = alexnet;
```

Read the image using imread.

```
I = imread('peppers.png');
figure
imshow(I)
```

FIGURE 3(a):USE OF ALEXNET IN MATLAB

The pretrained model requires the image size to be the same as the input size of the network. Determine the input size of the network using the InputSize property of the first layer of the network.

```
sz = net.Layers(1).InputSize
```

```
sz =

   227   227     3
```

Crop the image to the input size of the network. Alternatively, you can resize the image using imresize (Image Processing Toolbox™).

```
I = I(1:sz(1),1:sz(2),1:sz(3));
figure
imshow(I)
```

FIGURE 3(b):USE OF ALEXNET IN MATLAB

Classify the image using classify.

```
label = classify(net,I)

label =

  categorical

     bell pepper
```

Show the image and classification result together.

```
figure
imshow(I)
title(char(label))
```

FIGURE 3(c):USE OF ALEXNET IN MATLAB

Feature Extraction using Alexnet:

This example shows how to extract learned features from a pretrained convolutional neural network, and use those features to train an image classifier. Feature extraction is the easiest and fastest way use the representational power of pretrained deep networks. For example, you can train a support vector machine (SVM) using fitcecoc (Statistics and Machine Learning Toolbox™) on the extracted features.

Load Data

Unzip and load the sample images as an image datastore. imageDatastore automatically labels the images based on folder names and stores the data as an ImageDatastoreobject. An image datastore lets you store large image data, including data that does not fit in memory. Split the data into 70% training and 30% test data.

```
unzip('MerchData.zip');
images = imageDatastore('MerchData',...
    'IncludeSubfolders',true,...
    'LabelSource','foldernames');

[trainingImages,testImages] = splitEachLabel(images,0.7,'randomized');
```

There are now 55 training images and 20 validation images in this very small data set. Display some sample images.

```
numTrainImages = numel(trainingImages.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(trainingImages,idx(i));
    imshow(I)
end
```

FIGURE 4(): USE OF ALEXNET IN MATLAB

Load Pretrained Network

Load a pretrained AlexNet network. If Neural Network Toolbox Model *for AlexNet Network* is not installed, then the software provides a download link. AlexNet is trained on more than a million images and can classify images into 1000 object categories. For example, keyboard, mouse, pencil, and many animals. As a result, the model has learned rich feature representations for a wide range of images.

```
net = alexnet;
```

Display the network architecture. The network has five convolutional layers and three fully connected layers.

```
net.Layers

ans =

  25x1 Layer array with layers:

     1   'data'     Image Input                 227x227x3 images with 'zerocenter' normalization
     2   'conv1'    Convolution                 96 11x11x3 convolutions with stride [4  4] and padding [0  0]
     3   'relu1'    ReLU                        ReLU
     4   'norm1'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'    Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0]
     6   'conv2'    Convolution                 256 5x5x48 convolutions with stride [1  1] and padding [2  2]
     7   'relu2'    ReLU                        ReLU
     8   'norm2'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'    Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0]
    10   'conv3'    Convolution                 384 3x3x256 convolutions with stride [1  1] and padding [1  1]
    11   'relu3'    ReLU                        ReLU
    12   'conv4'    Convolution                 384 3x3x192 convolutions with stride [1  1] and padding [1  1]
    13   'relu4'    ReLU                        ReLU
    14   'conv5'    Convolution                 256 3x3x192 convolutions with stride [1  1] and padding [1  1]
    15   'relu5'    ReLU                        ReLU
    16   'pool5'    Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0]
    17   'fc6'      Fully Connected             4096 fully connected layer
    18   'relu6'    ReLU                        ReLU
    19   'drop6'    Dropout                     50% dropout
    20   'fc7'      Fully Connected             4096 fully connected layer
    21   'relu7'    ReLU                        ReLU
    22   'drop7'    Dropout                     50% dropout
    23   'fc8'      Fully Connected             1000 fully connected layer
    24   'prob'     Softmax                     softmax
```

25 'output' Classification Output crossentropyex with 'tench', 'goldfish', and 998 other classes

If the training images differ in size from the image input layer, then you must resize or crop the image data. In this example, the images are the same size as the input size of AlexNet, so you do not need to resize or crop the images.

Extract Image Features

The network constructs a hierarchical representation of input images. Deeper layers contain higher-level features, constructed using the lower-level features of earlier layers. To get the feature representations of the training and test images, use activations on the fully connected layer 'fc7'. To get a lower-level representation of the images, use an earlier layer in the network.

```
layer = 'fc7';
trainingFeatures = activations(net,trainingImages,layer);
testFeatures = activations(net,testImages,layer);
```

Extract the class labels from the training and test data.

```
trainingLabels = trainingImages.Labels;
testLabels = testImages.Labels;
```

Fit Image Classifier

Use the features extracted from the training images as predictor variables and fit a multiclass support vector machine (SVM) usingfitcecoc (Statistics and Machine Learning Toolbox).

```
classifier = fitcecoc(trainingFeatures,trainingLabels);
```

Classify Test Images

Classify the test images using the trained SVM model the features extracted from the test images.

```
predictedLabels = predict(classifier,testFeatures);
```

Display four sample test images with their predicted labels.

```
idx = [1 5 10 15];
figure
for i = 1:numel(idx)
   subplot(2,2,i)
   I = readimage(testImages,idx(i));
   label = predictedLabels(idx(i));
   imshow(I)
   title(char(label))
end
```

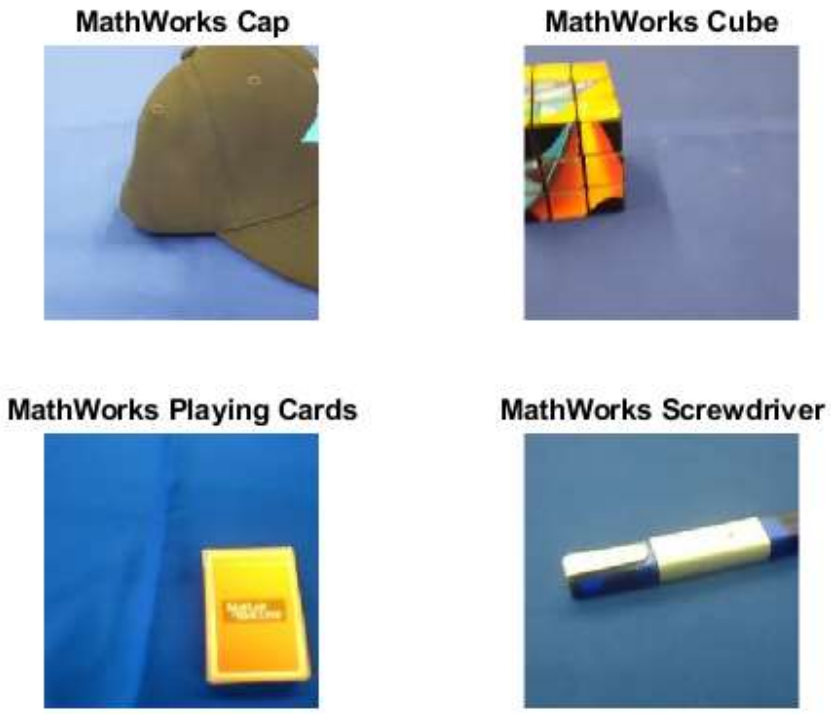


FIGURE 4(B): USE OF ALEXNET IN MATLAB

Calculate the classification accuracy on the test set. Accuracy is the fraction of labels that the network predicts correctly.

accuracy = mean(predictedLabels == testLabels)

accuracy =

1

This SVM has high accuracy. If the accuracy is not high enough using feature extraction, then try transfer learning instead.

Transfer Learning uaing ALEXNET:

This example shows how to fine-tune a pretrained AlexNet convolutional neural network to perform classification on a new collection of images.

Transfer learning is commonly used in deep learning applications. You can take a pretrained network and use it as a starting point to learn a new task. Fine-tuning a network with transfer learning is usually much faster and easier than training a network with randomly initialized weights from scratch. You can quickly transfer learned features to a new task using a smaller number of training images.

Load Data

Unzip and load the new images as an image datastore. imageDatastore automatically labels the images based on folder names and stores the data as an ImageDatastore object. An image datastore enables you to store large image data, including data that does not fit in memory, and efficiently read batches of images during training of a convolutional neural network.

```
unzip('MerchData.zip');
images = imageDatastore('MerchData',...
   'IncludeSubfolders',true,...
   'LabelSource','foldernames');
```

Divide the data into training and validation data sets. Use 70% of the images for training and 30% for validation. splitEachLabel splits the images datastore into two new datastores.

```
[trainingImages,validationImages] = splitEachLabel(images,0.7,'randomized');
```

This very small data set now contains 55 training images and 20 validation images. Display some sample images.

```
numTrainImages = numel(trainingImages.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
   subplot(4,4,i)
   I = readimage(trainingImages,idx(i));
   imshow(I)
end
```

FIGURE 4c(): USE OF ALEXNET IN MATLAB

Load Pretrained Network

Load the pretrained AlexNet neural network. If Neural Network Toolbox™ Model *for AlexNet Network* is not installed, then the software provides a download link. AlexNet is trained on more than one million images and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the model has learned rich feature representations for a wide range of images.

net = alexnet;

Display the network architecture. The network has five convolutional layers and three fully connected layers.

net.Layers

ans =

25x1 Layer array with layers:

1 'data' Image Input 227x227x3 images with 'zerocenter' normalization
2 'conv1' Convolution 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3 'relu1' ReLU ReLU
4 'norm1' Cross Channel Normalization cross channel normalization with 5 channels per element
5 'pool1' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6 'conv2' Convolution 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7 'relu2' ReLU ReLU
8 'norm2' Cross Channel Normalization cross channel normalization with 5 channels per element
9 'pool2' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10 'conv3' Convolution 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11 'relu3' ReLU ReLU
12 'conv4' Convolution 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13 'relu4' ReLU ReLU
14 'conv5' Convolution 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15 'relu5' ReLU ReLU
16 'pool5' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17 'fc6' Fully Connected 4096 fully connected layer
18 'relu6' ReLU ReLU
19 'drop6' Dropout 50% dropout
20 'fc7' Fully Connected 4096 fully connected layer
21 'relu7' ReLU ReLU
22 'drop7' Dropout 50% dropout
23 'fc8' Fully Connected 1000 fully connected layer
24 'prob' Softmax softmax
25 'output' Classification Output crossentropyex with 'tench', 'goldfish', and 998 other classes

Transfer Layers to New Network

The last three layers of the pretrained network net are configured for 1000 classes. These three layers must be fine-tuned for the new classification problem. Extract all layers, except the last three, from the pretrained network.

```
layersTransfer = net.Layers(1:end-3);
```

Transfer the layers to the new classification task by replacing the last three layers with a fully connected layer, a softmax layer, and a classification output layer. Specify the options of the new fully connected layer according to the new data. Set the fully connected layer to have the same size as the number of classes in the new data. To learn faster in the new layers than in the transferred layers, increase the WeightLearnRateFactor and BiasLearnRateFactor values of the fully connected layer.

```
numClasses = numel(categories(trainingImages.Labels))
layers = [
    layersTransfer
    fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',20)
    softmaxLayer
    classificationLayer];
```

```
numClasses =

     5
```

If the training images differ in size from the image input layer, then you must resize or crop the image data. In this example, the images are the same size as the input size of AlexNet, so you do not need to resize or crop the images.

Train Network

Specify the training options. For transfer learning, keep the features from the early layers of the pretrained network (the transferred layer weights). Set InitialLearnRate to a small value to slow down learning in the transferred layers. In the previous step, you increased the learning rate factors for the fully connected layer to speed up learning in the new final layers. This combination of learning rate settings results in fast learning only in the new layers and slower learning in the other layers. When performing transfer learning, you do not need to train for as many epochs. An epoch is a full training cycle on the entire training data set. Specify the mini-batch size and validation data. The software validates the network every ValidationFrequency iterations during training, and automatically stops training if the validation loss stops improving. Validate the network once per epoch.

```
miniBatchSize = 10;
numIterationsPerEpoch = floor(numel(trainingImages.Labels)/miniBatchSize);
```

```
options = trainingOptions('sgdm',...
    'MiniBatchSize',miniBatchSize,...
    'MaxEpochs',4,...
    'InitialLearnRate',1e-4,...
    'Verbose',false,...
    'Plots','training-progress',...
    'ValidationData',validationImages,...
    'ValidationFrequency',numIterationsPerEpoch);
```

Train the network that consists of the transferred and new layers. By default, trainNetwork uses a GPU if one is available (requires Parallel Computing Toolbox™ and a CUDA-enabled GPU with compute capability 3.0 or higher). Otherwise, it uses a CPU. You can also specify the execution environment by using the 'ExecutionEnvironment' name-value pair argument of trainingOptions.

```
netTransfer = trainNetwork(trainingImages,layers,options);
```

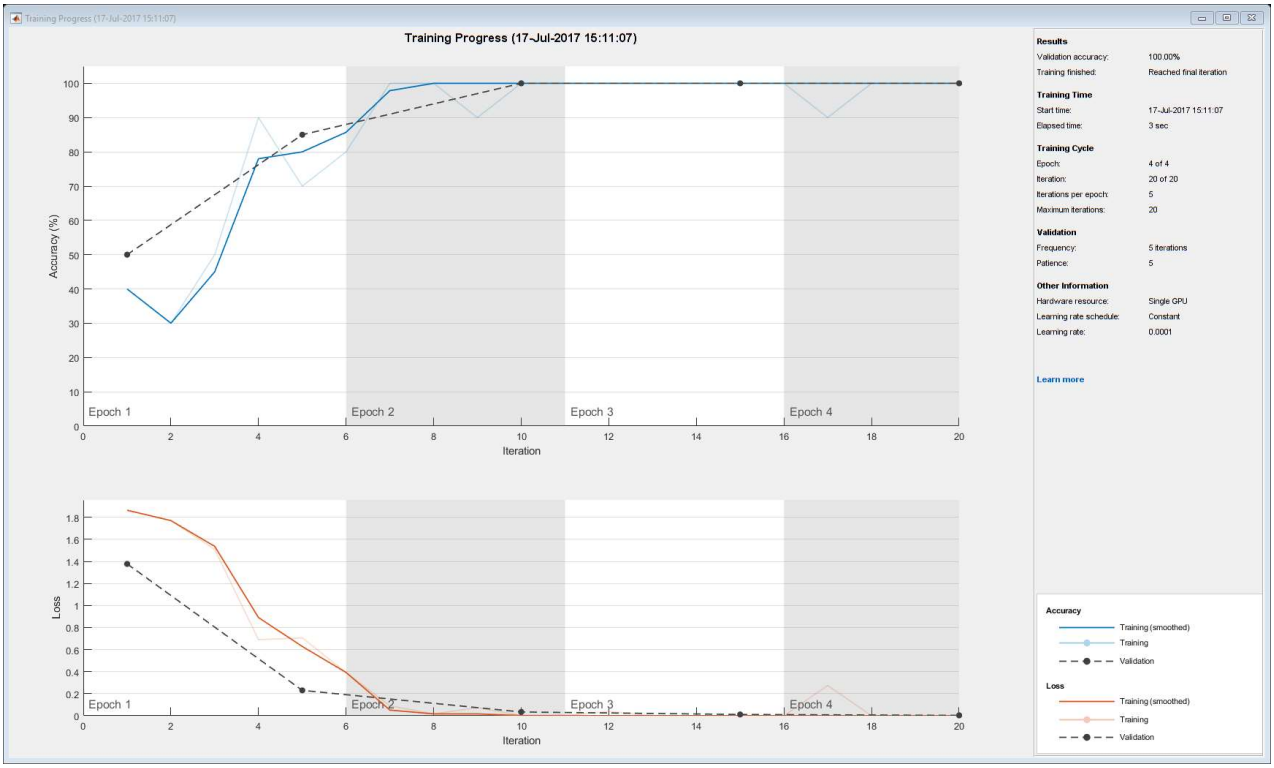


FIGURE 5:

Classify Validation Images

Classify the validation images using the fine-tuned network.

```
predictedLabels = classify(netTransfer,validationImages);
```

Display four sample validation images with their predicted labels.

```
idx = [1 5 10 15];
figure
for i = 1:numel(idx)
    subplot(2,2,i)
    I = readimage(validationImages,idx(i));
    label = predictedLabels(idx(i));
    imshow(I)
    title(char(label))
end
```

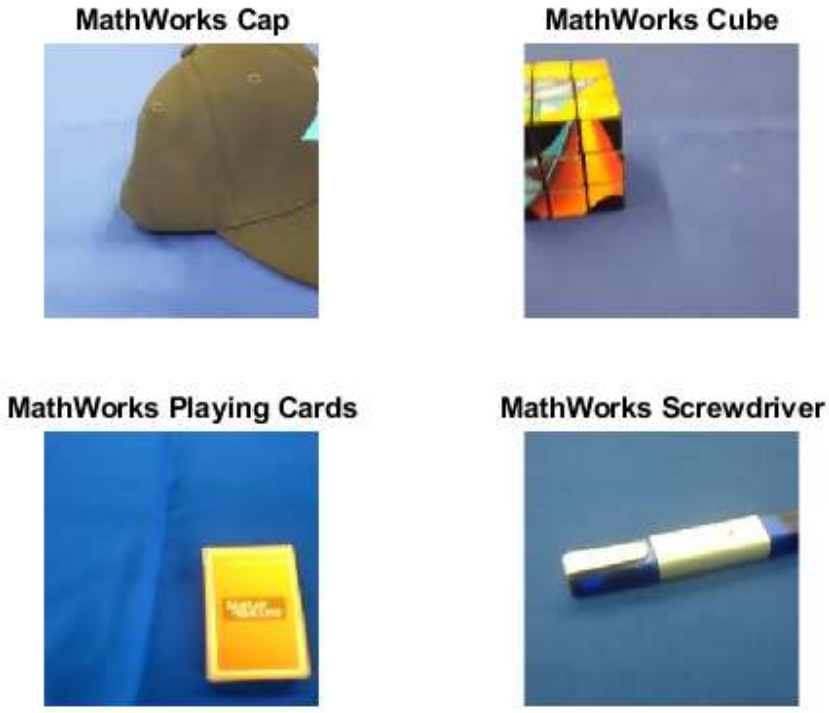


FIGURE 6:

Calculate the classification accuracy on the validation set. Accuracy is the fraction of labels that the network predicts correctly.

```
valLabels = validationImages.Labels;
accuracy = mean(predictedLabels == valLabels)
```

accuracy =

1

This trained network has high accuracy. If the accuracy is not high enough using transfer learning, then try feature extraction instead.

Output Arguments

net — Pretrained AlexNet convolutional neural network SeriesNetwork object

Pretrained AlexNet convolutional neural network returned as a SeriesNetwork object.

CHAPTER 8

SYSTEM IMPLEMENTATION

TOOLS:

MATLAB:

A proprietary programming language developed by MathWorks, **MATLAB** allows matrix manipulations, plotting functions and data, implementation of algorithms,

MATLAB PROFILER:

The Profiler is a user interface based on the results returned by the profile function. If you are profiling code that runs in parallel, for best results use the Parallel Computing Toolbox parallel profiler. The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

CONCLUSION

What we now see from our research is that using alexnet which is written in cuda and training it on a GPU by nvidia increases both the efficiency speed and accuracy of image detection.

FUTURE ENHANCEMENT

AI workloads are different from the calculations most of our current computers are built to perform. AI implies prediction, inference, intuition. But the most creative machine learning algorithms are hamstrung by machines that can't harness their power. Hence, if we're to make great strides in AI, our hardware must change, too. Starting with GPUs, and then evolving to analog devices, and then fault-tolerant quantum computers.

Let's start in the present, with applying massively distributed deep learning algorithms to Graphics processing units (GPU) for high speed data movement, to ultimately understand images and sound. The DDL algorithms "train" on visual and audio data, and the more GPUs should mean faster learning. To date, IBM's record-setting 95 percent scaling efficiency (meaning improved training as more GPUs are added) can recognize 33.8 percent of 7.5 million images, using 256 GPUs on 64 "Minsky" Power systems.

Distributed deep learning has progressed at a rate of about 2.5 times per year since 2009, when GPUs went from video game graphics accelerators to deep learning model trainers. So a question I addressed at Applied Materials' Semiconductor Futurescapes: New Technologies, New Solutions event during the 2017 IEEE International Electron Devices Meeting:

APPENDIX

Transfer learning :

```
%% Load a deep, convolutional neural network
alex = alexnet;
layers = alex.Layers

%% Modify the network to use prefered number of categories
layers(23) = fullyConnectedLayer(2);
layers(25) = classificationLayer

%% Load training data as per requirement
allImages = imageDatastore('E:\myImages', 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
[trainingImages, testImages] = splitEachLabel(allImages, 0.7, 'randomize');

%% Re-train the Deep Neural Network
opts = trainingOptions('sgdm', 'InitialLearnRate', 0.001, 'MaxEpochs', 20, 'MiniBatchSize', 64);
myNet = trainNetwork(trainingImages, layers, opts);

%% Measure network accuracy after retraining
predictedLabels = classify(myNet, testImages);
accuracy = mean(predictedLabels == testImages.Labels)
```

This transfer learning code enables us to retrain the ALEXNET framework on our local GPU using the given image dataset.

SAMPLE CODE

input stream.m:

```
classdef InputStream < asyncio.Stream
% A stream that asynchronously reads from a device and buffers incoming data.
%
%   If the device supports input, then isSupported() will return true and
%   data can be read from the stream.
%
%   See also asyncio.OutputStream and asyncio.Channel.

% Authors: DTL
% Copyright 2007-2015 The MathWorks, Inc.

    properties(GetAccess='public',SetAccess='private',Dependent=true)
        % The number of items that can be read without blocking.
        DataAvailable;
    end

    events(NotifyAccess='public')
        % Device has written data to the input stream and data is available
```

```
        % to read. The data associated with this event is an
        % asyncio.DataEventInfo where CurrentCount is the amount of data
        % available to read.
        DataWritten
    end

    methods(Access='public')
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
        % Lifetime
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%

        function obj = InputStream(channelImpl)
        % INPUTSTREAM Create a wrapper for a channel's input stream.

        % OBJ = INPUTSTREAM(CHANNELIMPL) creates an object that wraps the
        % actual input stream of CHANNELIMPL.
        %
        % Notes:
        % If the channel has no input stream, then isSupported will return
        % false and no other methods will succeed.
            assert(nargin == 1, 'The parent channel was not specified');

            % Construct super class.
            obj@asyncio.Stream( channelImpl );

            % Initialize an empty partial packet used by read.
            obj.clearPartialPacket();
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
        % Commands
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%

        function [data, countRead, err] = read(obj, countRequested)
        %READ Read data from the input stream.
        %
        % [DATA, COUNTREAD] = READ(OBJ, COUNTREQUESTED)
        % reads the requested number of items from the input stream. If the
        % count requested is less than DataAvailable then this method will
        % block. If blocking is needed, read will wait until the requested
        % number of items are read or the channel is closed or the device
        % is done reading data or an error occurs.
        %
        % Inputs:
        % COUNTREQUESTED - Indicates the number of items to read. This
        %     parameter is optional and defaults to all the data currently
        %     available on the input stream.
        %
        % Outputs:
        % DATA - An N-dimensional matrix of data. The dimension that
        %     indicates the count is specified by the CountDimension property.
        %     If no data was returned this will be an empty array.
        %
        % COUNTREAD - The actual number of items read. If there was no error,
```

```
%     this value will be equal to the count requested unless the
%     channel was closed or the device was done. If there was an
%     error, this value will be zero.
%
% ERR - A string that indicates if any error occurred while
%     waiting for data to arrive. Possible values are:
%       'timeout' -   The timeout elapsed.
%       'invalid' - The channel or stream was deleted.
%       '' - No error occurred.

    % If countRequested not specified...
    if nargin < 2
        % Read what is available.
        countRequested = obj.DataAvailable;
    % Otherwise, validate it.
    elseif ~(isnumeric(countRequested) && isscalar(countRequested) && ...
            countRequested >= 0)
        error(message('asyncio:InputStream:invalidCountRequested'));
    end

    % Initialize return values.
    err = '';

    % Initialize internal values.
    countDimension = obj.CountDimension;

    % First get data left over from the previous read.
    [data, countRead] = obj.readPartialPacket(countRequested, countDimension);

    % If that fully satisfied our request, we're done.
    if countRead == countRequested
        return;
    end

    % Otherwise initialize the packets read so far.
    if countRead == 0
        packetsRead = {};
    else
        packetsRead = {data};
    end

    % While we need more data to satisfy the request.
    while countRead < countRequested

        % Optimization: Don't call wait unless needed and use
        % underlying stream directly - 15% speedup.
        if obj.StreamImpl.getDataAvailable() == 0

            % Wait for any data to be available on the stream.
            status = obj.wait(@(obj) obj.getDataAvailable() > 0);

            % If no data was available, break.
            if ~strcmpi(status, 'completed')
                % Set error value.
                err = status;

                % If object became invalid, return immediately.
                if strcmpi(status, 'invalid')
```

```
            return;
        end

        % Don't consider done an error.
        if strcmpi(status, 'done')
            err = '';
        end
        break;
      end
    end

    % Try to read what we need from the stream.
    countToRead = countRequested - countRead;

    % Get a cell array of data packets that satisfies
    % as much of our request that is available.
    [packets, count] = obj.readPackets(countToRead);

    % Accumulate the packets and the count.
    packetsRead = [packetsRead packets]; %#ok<AGROW>
    countRead = countRead + count;
  end

  % Now check if we've gotton too much.
  if countRead > countRequested

    % Save the last packet as the partial packet.
    obj.setPartialPacket(packetsRead{end}, countDimension);

    % Determine how much of the last packet is needed.
    countNeeded = obj.PartialPacketCount - (countRead - countRequested);

    % Read what we need and use that as the last packet.
    packetsRead{end} = obj.readPartialPacket(countNeeded, countDimension);
  end

  % Concatentate cell array into a single matrix along the
  % count dimension.
  data = cat(countDimension, packetsRead{:});
  if ~isempty(data)
    countRead = size(data, countDimension);
  else
    countRead = 0;
  end

  % If there was an error, don't return any data,
  % but save it for any subseqent reads.
  if ~isempty(err)
    obj.setPartialPacket(data, countDimension);
    data = [];
    countRead = 0;
  end
end

function [packets, countRead] = readPackets(obj, countRequested)
%READPACKETS Read a cell array of data packets from the input stream.
%
% [PACKETS, COUNTREAD] = READPACKETS(OBJ, COUNTREQUESTED)
```

```
        % reads a cell array of data packets from the input stream that
        % satisfies as much of our request as is available. This method does
        % not block.
        %
        % Inputs:
        % COUNTREQUESTED - Indicates the desired number of items to read.
        %
        % Outputs:
        % PACKETS - A 1xN cell array of data packets.
        %
        % COUNTREAD - The actual number of items read. This may be less than
        %     the number requested if enough data is not available in the
        %     input stream. It also may be more than the number requested
        %     if the count requested is not an even multiple of the packet
        %     size.
            [packets, countRead] = obj.StreamImpl.read(countRequested);
        end

        function flush(obj)
        % FLUSH Flush all data in the stream.
        %
        % FLUSH(OBJ) immediately discards all data in the stream.

            % Clear any partial packet left over from the last read.
            obj.clearPartialPacket();

            % Call superclass to discard anything in the stream.
            flush@asyncio.Stream(obj);
        end
    end

    methods
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
        % Property Access Methods
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%

        function count = get.DataAvailable(obj)
            % Start with any partial packet left over from the last read.
            count = obj.PartialPacketCount;

            % Add data available in the stream.
            count = count + obj.getDataAvailable();
        end
    end

    methods(Static, Hidden)
        function lock()
            mlock;
        end
    end

    methods(Access='private')

        function [dataRead, countRead] = readPartialPacket(obj, countRequested, countDimension)
            ppc = obj.PartialPacketCount;
            % If there is a partial packet, start with that.
```

```
            if ppc > 0
                pps = obj.PartialPacketStart;
                % If the left over partial packet fully satifies the read.
                if countRequested < ppc
                    dataRead = asyncio.Stream.extractFromPacket(obj.PartialPacket, countDimension, pps,
pps+countRequested-1);
                    countRead = countRequested;
                    obj.PartialPacketStart = pps + countRead;
                    obj.PartialPacketCount = ppc - countRead;
                else
                    % Use the entire remaining partial packet.
                    dataRead = asyncio.Stream.extractFromPacket(obj.PartialPacket, countDimension, pps, pps+ppc-
1);
                    countRead = ppc;
                    % Clear partial packet.
                    obj.clearPartialPacket();
                end
            else
                dataRead = [];
                countRead = 0;
            end
        end

        function setPartialPacket(obj, data, countDimension)
            obj.PartialPacket = data;
            obj.PartialPacketStart = 1;
            obj.PartialPacketCount = size(data, countDimension);
        end

        function clearPartialPacket(obj)
            obj.PartialPacket = [];
            obj.PartialPacketStart = 0;
            obj.PartialPacketCount = 0;
        end
    end

    properties(GetAccess='private',SetAccess='private')
        % Holds any extra items from the last read.
        PartialPacket;
        PartialPacketStart;
        PartialPacketCount;
    end
end
```

channel.m:

```
classdef Channel < dynamicprops
% A communications channel to any device that is a data source or sink.
%
%   The device may be a piece of hardware, a file, socket, network, etc.
%   The device may be bidirectional, input-only, or output-only. If the device
%   is a source of incoming data, then the channel will contain a valid
%   InputStream property. If the device is a sink for outgoing data, then the
%   channel will contain a valid OutputStream property.
%
%   The channel works in conjunction with a two C++ plug-ins. The device
```

```
%   plug-in wraps the device-specific software API. The converter plug-in
%   converts data in MATLAB format to a format expected by the device
%   plug-in (and vica-versa).
%
%   See also asyncio.Channel.Channel, asyncio.InputStream, asyncio.OutputStream.

% Authors: DTL
% Copyright 2007-2016 The MathWorks, Inc.

  properties(GetAccess='public',SetAccess='private')
    % An asyncio.InputStream used for reading.
    InputStream;

    % An asyncio.OutputStream used for writing.
    OutputStream;
  end

  events(NotifyAccess='private')
    % The channel has been closed.
    Closed

    % The channel has been opened.
    Opened

    % A device-specific custom event has occurred.
    Custom
  end

  methods(Access='public')
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
    % Lifetime
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%

    function obj = Channel(devicePluginPath, converterPluginPath, ...
                           options, streamLimits)
    % CHANNEL Create an asynchronous communication channel to a device.
    %
    % OBJ = CHANNEL(DEVICEPLUGINPATH, CONVERTERPLUGINPATH,
    %               OPTIONS, STREAMLIMITS)
    % creates a communication channel to the given device and sets up
    % the appropriate input and output streams.
    %
    % Inputs:
    % DEVICEPLUGINPATH - The full path and name of the device plug-in.
    % The file extension of the plug-in should be omitted.
    %
    % CONVERTERPLUGINPATH - The full path and name of the converter
    % plug-in. The file extension of the plug-in should be omitted.
    %
    % OPTIONS - A structure containing information that needs to be
    % passed to the device plug-in during initialization. This parameter
    % is optional unless STREAMLIMITS also needs to be specified.
    % The default value for this parameter is an empty structure.
    %
    % STREAMLIMITS - An array of two doubles that indicate the maximum
    % number of items to buffer in the input and output streams. Valid
```

```
% values for each limit are between 0 and Inf, inclusive. If Inf is
% used, buffering will be limited only by the amount of memory available
% to the application. If zero is used, the stream is unbuffered and all
% reads or writes are synchronous (i.e go directly to the device).
% This parameter is optional. The default value for this parameter
% is [Inf Inf].
%
% Notes:
% During initialization, the device plug-in can specifiy custom
% properties and their initial values. These properties will be
% created as dynamic properties on the channel object and can be
% updated at any time by the device plug-in.

    % If no stream limits specified, provide a default.
    if nargin < 4
        streamLimits = [Inf Inf];
    end

    % If no options specified, provide a default.
    if nargin < 3 || isempty(options)
        options = struct([]);
    end

    if ~isfloat(streamLimits) || length(streamLimits) ~= 2 || ...
        any(isnan(streamLimits))
        error(message('asyncio:Channel:invalidStreamLimits'));
    end

    % NOTE: The destructor will always be called if there is any
    % failure from this point forward. MATLAB object system rules
    % dictate that once we access any object property in the
    % constructor, then the destructor must be called.

    % Create underlying C++ channel implementation.
    obj.ChannelImpl = asyncioimpl.Channel(devicePluginPath,...
                                          converterPluginPath,...
                                          streamLimits(1),...
                                          streamLimits(2));

    % Lock asyncio class definitions when there are any instances
    % of a Channel. See g1306865, g1314266, and g1355310 for info.
    asyncio.Channel.lock();

    % Do pre init functionality (can be overridden by a subclass).
    obj.preInit();

    % Initialize device plug-in and get custom property/value pairs.
    customProps = obj.ChannelImpl.init(options);

    % Add and initialize dynamic properties.
    fields = fieldnames(customProps);
    for i = 1:length(fields)
        prop = addprop(obj, fields{i});
        obj.(fields{i}) = customProps.(fields{i});
        prop.SetObservable = true;
    end

    % Create the input/output streams.
```

```
            obj.InputStream = asyncio.InputStream(obj.ChannelImpl);
            obj.OutputStream = asyncio.OutputStream(obj.ChannelImpl);

            % Do post init functionality (can be overridden by a subclass).
            obj.postInit();
        end

        function delete(obj)
        % DELETE Destroy the communications channel.
        %
        % If the communication channel is still open, it will be closed
        % and all data in the input and output streams will be lost.

            % If the underlying channel implementation never completed,
            % bail out to avoid any errors here.
            if isempty(obj.ChannelImpl)
                return;
            end

            % Make sure we are closed.
            if isOpen(obj)
                warning(message('asyncio:Channel:stillOpenDuringDelete'));
                obj.close();
            end

            % Do pre-term functionality (can be overridden by a subclass).
            obj.preTerm();

            % Terminate device plug-in.
            % NOTE: term is called even if init() fails.
            obj.ChannelImpl.term();

            % Delete streams.
            delete(obj.InputStream);
            delete(obj.OutputStream);

            % Delete underlying channel implementation.
            delete(obj.ChannelImpl);

            % Unlock asyncio class definitions when there are no more instances
            % of a Channel.
            asyncio.Channel.unlock();
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
        % Getters/Setters
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%

        function result = isOpen(obj)
        % ISOPEN Return true is the channel is open, false otherwise.

            assert( isscalar(obj), 'Channel:isOpen:notScalar',...
                              'OBJ must be scalar.');
            result = obj.ChannelImpl.isOpen();
        end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Commands
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function open(obj, options)
% OPEN Connect to the device and begin the streaming of data.
%
% OPEN(OBJ,OPTIONS) opens the communication channel, gains
% exclusive access to any resources, and allows the device
% to begin sending and receiving data. If the streams have any
% filters, they will also be opened.
%
% Inputs:
% OPTIONS - A structure containing information that needs to be passed
% to the device plug-in and filter plug-in(s) prior to opening. This
% parameter is optional and defaults to an empty structure.
%
% Notes:
% - Open does not flush either the input stream or the output
%   stream. To alter this behavior, override the preOpen method.
% - Resources will be opened in the following order:
%     1) Filter plug-in(s) of the input stream, if any, in the
%        order in which they were added to the input stream.
%     2) Filter plug-in(s) of the output stream, if any, in the
%        order in which they were added to the output stream.
%     3) Device plug-in.
%
% See also asyncio.Stream.addFilter
%
  assert( isscalar(obj), 'Channel:open:notScalar',...
                    'OBJ must be scalar.');

  % If no options specified...
  if nargin < 2 || isempty(options)
     options = struct([]);
  end

  if isOpen(obj)
     return;
  end

  % Do pre-open functionality (can be overridden by a subclass).
  obj.preOpen();

  % Gain exclusive access of the device.
  obj.ChannelImpl.open(options);

  % Notify any listeners that we have opened.
  notify(obj, 'Opened');
end

function close(obj)
% CLOSE Disconnect from the device and stop the streaming of data.
%
% CLOSE(OBJ) stops the streaming of data, releases exclusive access
% to any resources, and closes the communication channel. If the
```

```
% streams have any filters, they will also be closed.
%
% Notes:
% - Close does not flush either the input stream or the output
%   stream. To alter this behavior, override the postClose method.
% - Resources will be closed in the following order:
%       1) Device plug-in.
%       2) Filter plug-in(s) of the output stream, if any, in the
%          order in which they were added to the output stream.
%       3) Filter plug-in(s) of the input stream, if any, in the
%          order in which they were added to the input stream.
%
% See also asyncio.Stream.addFilter
%
    assert( isscalar(obj), 'Channel:close:notScalar',...
                        'OBJ must be scalar.');

    if ~isOpen(obj)
        return;
    end

    % Release exclusive access to the device.
    obj.ChannelImpl.close();

    % Do post-close functionality (can be overridden by a subclass).
    obj.postClose();

    % Notify any listeners that we have closed.
    notify(obj, 'Closed');
end

function execute(obj, command, options)
% EXECUTE Execute an arbitrary device-specific command.
%
% EXECUTE(OBJ,COMMAND,OPTIONS) will pass the given command and
% options to the device plug-in.
%
% Inputs:
% COMMAND - A string that represents the command to execute.
% OPTIONS - A structure containing information that needs to be passed
% to the device plug-in in order to execute the command. This
% parameter is optional and defaults to an empty structure.
%
% Notes:
% Execute can be called at any time, not just when the channel is open.
% Errors, warnings, custom events, and custom property updates are
% propagated as usual during execute.

    assert( isscalar(obj), 'Channel:execute:notScalar',...
                        'OBJ must be scalar.');

    % If no options specified...
    if nargin < 3 || isempty(options)
        options = struct([]);
    end

    obj.ChannelImpl.execute(command, options);
end
```

```
    end

    methods(Static, Access='private')
        function lock()
            % Lock this class definition and the class definitions of the
            % children on the first lock.
            if asyncio.Channel.updateAndFetchLockCount(1) > 0
                if ~mislocked % just in case
                    mlock;
                    asyncio.InputStream.lock();
                    asyncio.OutputStream.lock();
                end
            end
        end

        function unlock()
            % Unlock this class definition and the class definitions of the
            % children on the last unlock.
            if asyncio.Channel.updateAndFetchLockCount(-1) < 1
                if mislocked % just in case
                    munlock;
                    munlock('asyncio.InputStream');
                    munlock('asyncio.OutputStream');
                end
            end
        end

        function count = updateAndFetchLockCount(increment)
            persistent lockCount;
            if isempty(lockCount)
                lockCount = 0;
            end
            lockCount = lockCount + increment;
            count = lockCount;
        end
    end

    methods(Access='protected')

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
        % Helpers
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%

        function preInit(obj)
        % Functionality done just before channel is inited.

            obj.WarningListener = event.listener(obj.ChannelImpl,...
                                 'Warning',...
                                 @(source, data) obj.onWarning(data.ID, data.Text));

            obj.TraceListener = event.listener(obj.ChannelImpl,...
                                 'Trace',...
                                 @(source, data) obj.onTrace(data.Text));
        end

        function postInit(obj)
```

```
% Functionality done just after channel is inited.

    % Connect property, message, and custom events to our methods.
    obj.PropertyChangedListener = event.listener(obj.ChannelImpl,...
                        'PropertyChanged',...
                        @(source, data) obj.onPropertyChanged(data.Name, data.Value));
    % Allow property events to recurse since a client's listener
    % may do something that causes another property event to fire
    % synchronously.
    obj.PropertyChangedListener.Recursive = true;

    obj.ErrorListener = event.listener(obj.ChannelImpl,...
                        'PreError',...
                        @(source, data) obj.onPreError(data.ID, data.Text));

    obj.CustomListener = event.listener(obj.ChannelImpl,...
                        'Custom',...
                        @(source, data) obj.onCustomEvent(data.Type, data.Data));
    % Allow custom events to recurse since a client's listener
    % may do something that causes another custom event to fire
    % synchronously.
    obj.CustomListener.Recursive = true;
end

function preOpen(obj)
% Functionality done just prior to device being opened.

    % If data flow events are enabled...
    if ~obj.DataEventsDisabled

        % Connect data flow callbacks to our methods.
        obj.DataReceivedListener = event.listener(obj.ChannelImpl,...
                            'DataReceived',...
                            @(source, data) obj.onDataReceived());

        obj.DataSentListener = event.listener(obj.ChannelImpl,...
                            'DataSent',...
                            @(source, data) obj.onDataSent());
    else
        obj.DataReceivedListener = [];
        obj.DataSentListener = [];
    end
end

function postClose(obj)
% Functionality done just after device is closed.

    % Disconnect data flow events.
    % This has the effect of stopping the transfer of events
    % that may have been queued by the data source or sink but
    % have not yet been processed by MATLAB.
    if ~isempty(obj.DataSentListener)
        delete(obj.DataSentListener);
    end
    if ~isempty(obj.DataReceivedListener)
        delete(obj.DataReceivedListener);
    end
end
```

```
        function preTerm(obj)
        % Functionality done just before channel is destroyed.

            % Disconnect property, message, and custom events.
            % This has the effect of stopping the transfer of events
            % that may have been queued by the device but have not yet
            % processed by MATLAB.
            delete(obj.CustomListener);
            delete(obj.ErrorListener);
            delete(obj.WarningListener);
            delete(obj.TraceListener);
            delete(obj.PropertyChangedListener);
        end
    end

    methods(Access='private')

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
        % Event handlers
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%

        function onPropertyChanged(obj, name, value)
            % Handle property update from device plug-in.
            obj.(name) = value;
        end

        function onPreError(obj, ~, ~)
            % Close the channel before any error is thrown but don't do
            % the error here. It is done by asyncioimpl.Channel.
            obj.close();
        end

        function onWarning(~, id, text)
            prevState = warning('off','backtrace');
            warning(id, text);
            warning(prevState);
        end

        function onTrace(obj, text)
            if obj.TraceEnabled
                disp(text);
            end
        end

        function onCustomEvent(obj, type, data)
            % Handle custom event from device plug-in.
            % Notify any listeners.
            notify(obj, 'Custom', asyncio.CustomEventInfo(type,data));
        end

        function onDataReceived(obj)
            % Handle data received event from engine.

            % Notify any listeners with the amount of data available.
            % If no data is available to read, don't send the event.
```

```
            count = obj.InputStream.DataAvailable;
            if count > 0
                notify(obj.InputStream, 'DataWritten', ...
                    asyncio.DataEventInfo(count));
            end
        end

        function onDataSent(obj)
            % Handle data sent event from engine.

            % Notify any listeners with the amount of space available.
            % If no space is available to write, don't send the event.
            space = obj.OutputStream.SpaceAvailable;
            if space > 0
                notify(obj.OutputStream, 'DataRead', ...
                    asyncio.DataEventInfo(space));
            end
        end
    end

    properties(Hidden=true)
        % Enables/disables trace statements from plug-ins.
        TraceEnabled = false;

        % Enables/disables the data flow events from the underlying
        % C++ channel. This is a performance optimization for clients
        % that don't need to use the DataRead and DataWritten events.
        % NOTE: Must be set before the channel is opened.
        DataEventsDisabled = false;
    end

    properties(GetAccess='private',SetAccess='private')
        % Underlying C++ implementation of channel.
        ChannelImpl;

        % Listeners for ChannelImpl events.
        CustomListener;
        ErrorListener;
        WarningListener;
        TraceListener;
        PropertyChangedListener;
        DataReceivedListener;
        DataSentListener;
    end
end
```

image datastore.m :

```
function ds = imageDatastore(location, varargin)
%IMAGEDATASTORE Create an ImageDatastore to work with collections of images.
%   IMDS = imageDatastore(LOCATION) creates an ImageDatastore IMDS given the
%   LOCATION of the image files. LOCATION has the following properties:
%      - Can be a filename or a folder name
%      - Can be a cell array of multiple file or folder names
%      - Can contain a relative path (HDFS requires a full path)
%      - Can contain a wildcard (*) character.
```

```
%     - All the files in LOCATION must have extensions supported by IMFORMATS
%
%   IMDS = imageDatastore(__,'IncludeSubfolders',TF) specifies the logical
%   true or false to indicate whether the files in each folder and its
%   subfolders are included recursively or not.
%
%   IMDS = imageDatastore(__,'FileExtensions',EXTENSIONS) specifies the
%   extensions of files to be included. The extensions are not required to
%   be supported by IMFORMATS. Values for EXTENSIONS can be:
%     - A character vector, such as '.jpg' or '.png' (empty quotes '' are
%       allowed for files without extensions)
%     - A cell array of character vector, such as {'.jpg', '.png'}
%
%   IMDS = imageDatastore(__,'ReadSize',READSIZE) specifies the maximum
%   number of image files to read in a call to the read function. By default,
%   READSIZE is 1. The output of read is a cell array of image data when
%   READSIZE > 1.
%
%   IMDS = imageDatastore(__,'ReadFcn',@MYCUSTOMREADER) specifies the user-
%   defined function to read files. The value of 'ReadFcn' must be a
%   function handle with a signature similar to the following:
%     function data = MYCUSTOMREADER(filename)
%     ..
%     end
%
%   IMDS = imageDatastore(__,'LabelSource',SOURCE) specifies the source from
%   which the Labels property obtains labels. By default, the value of
%   SOURCE is 'none'. If SOURCE is 'foldernames', then the values for the
%   Labels property are obtained from the folder names of the image files.
%
%   IMDS = imageDatastore(__,'Labels',LABELS) specifies the datastore labels
%   according to LABELS. LABELS must be a cell array of character vectors or
%   a vector of numeric, logical, or categorical type.
%
%   ImageDatastore Properties:
%
%     Files          - Cell array of file names
%     ReadSize       - Upper limit on the number of images returned by the read method
%     ReadFcn        - Function handle used to read files
%     Labels         - A set of labels for images
%
%   ImageDatastore Methods:
%
%     hasdata        - Returns true if there is more data in the datastore
%     read           - Reads the next consecutive file
%     reset          - Resets the datastore to the start of the data
%     preview        - Reads the first image from the datastore
%     readimage      - Reads a specified image from the datastore
%     readall        - Reads all image files from the datastore
%     partition      - Returns a new datastore that represents a single
%                      partitioned portion of the original datastore
%     numpartitions  - Returns an estimate for a reasonable number of
%                      partitions to use with the partition function,
%                      according to the total data size
%     splitEachLabel - Splits the ImageDatastore labels according to the
%                      specified proportions, which can be represented as
%                      percentages or number of files.
%     countEachLabel - Counts the number of unique labels in the ImageDatastore
```

```
%     shuffle       - Shuffles the files of ImageDatastore using randperm
%
%   Example:
%   --------
%     folders = fullfile(matlabroot,'toolbox','matlab',{'demos','imagesci'});
%     exts = {'.jpg','.png','.tif'};
%     imds = imageDatastore(folders,'FileExtensions',exts);
%     img1 = read(imds);                % Read the first image
%     img2 = read(imds);                % Read the next image
%     readall(imds)                     % Read all of the images
%     imgarr = cell(numel(imds.Files),1);
%     for i = 1:numel(imds.Files)       % Read images using a for loop
%         imgarr{i} = readimage(imds,i);
%     end
%
%   See also datastore, mapreduce, imformats, matlab.io.datastore.ImageDatastore.

%   Copyright 2015 The MathWorks, Inc.
    ds = matlab.io.datastore.ImageDatastore(location, varargin{:});
end
```

SAMPLE SCREENSHOT

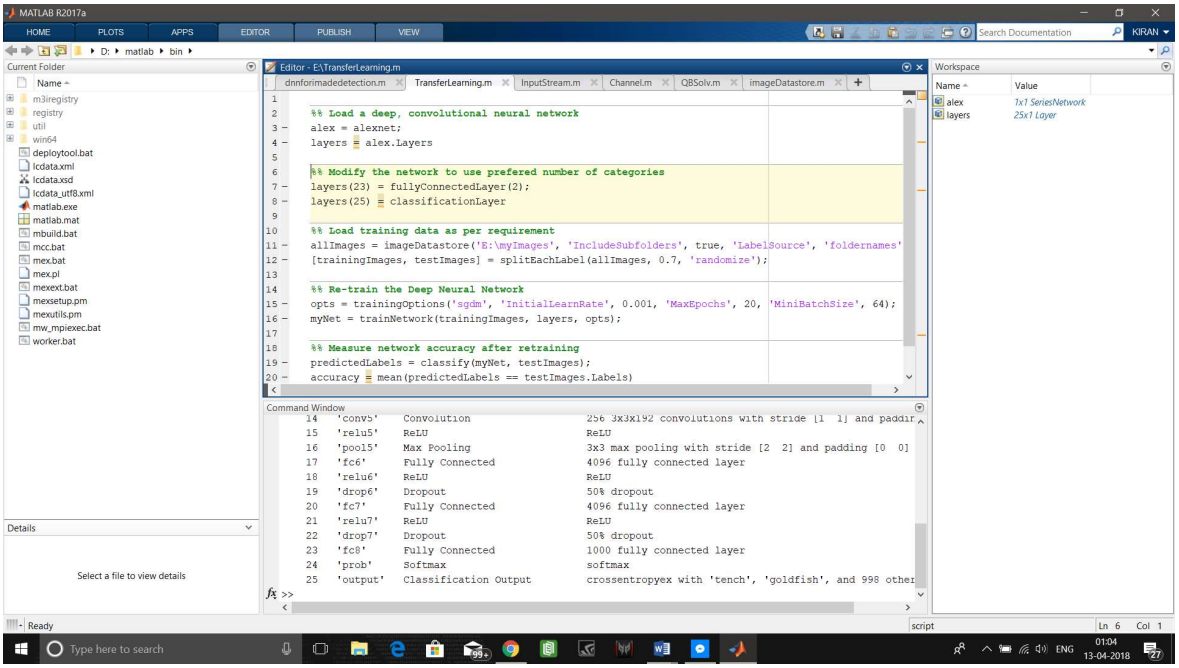


FIGURE 7:

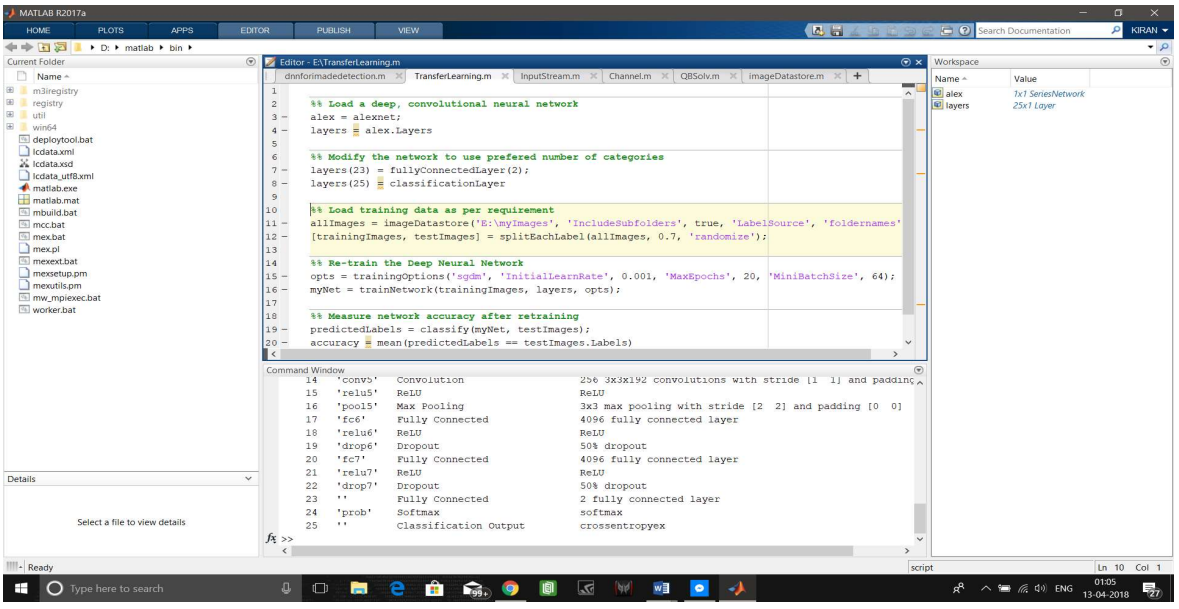


FIGURE 8:

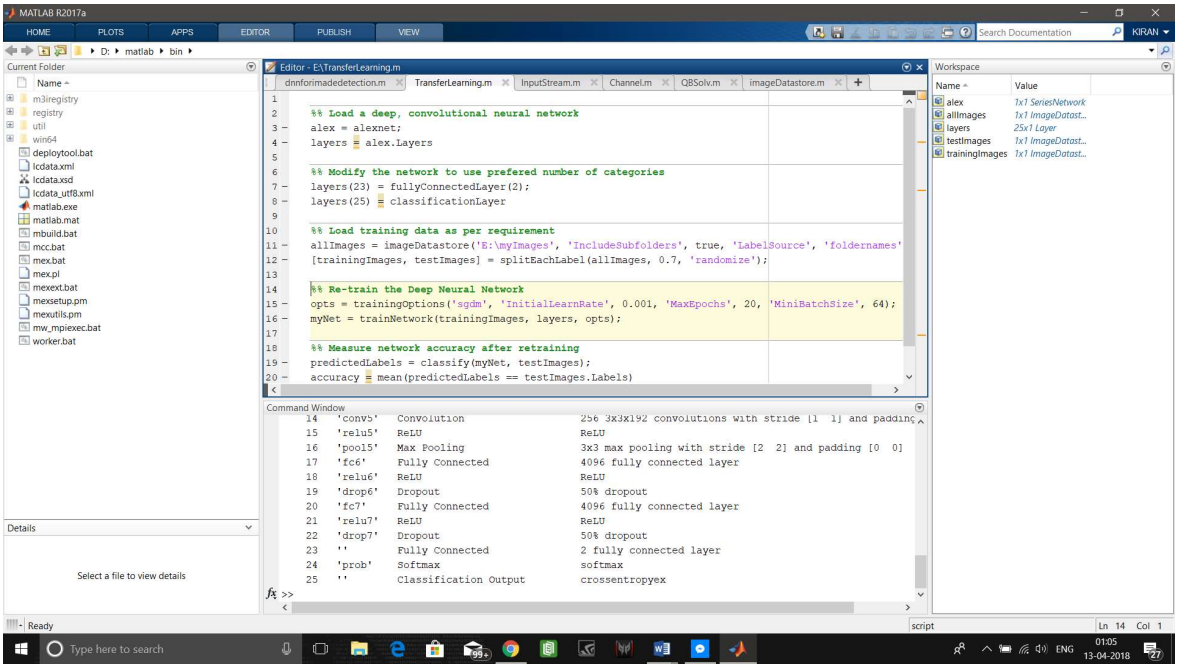


FIGURE 9:

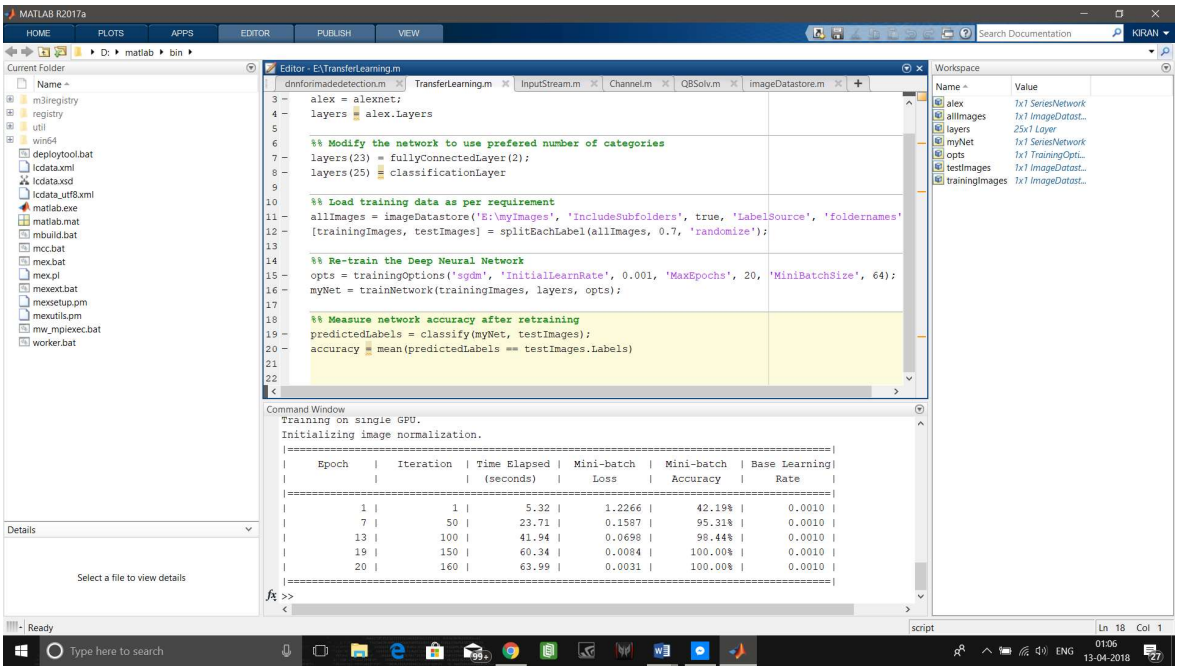


FIGURE 10:

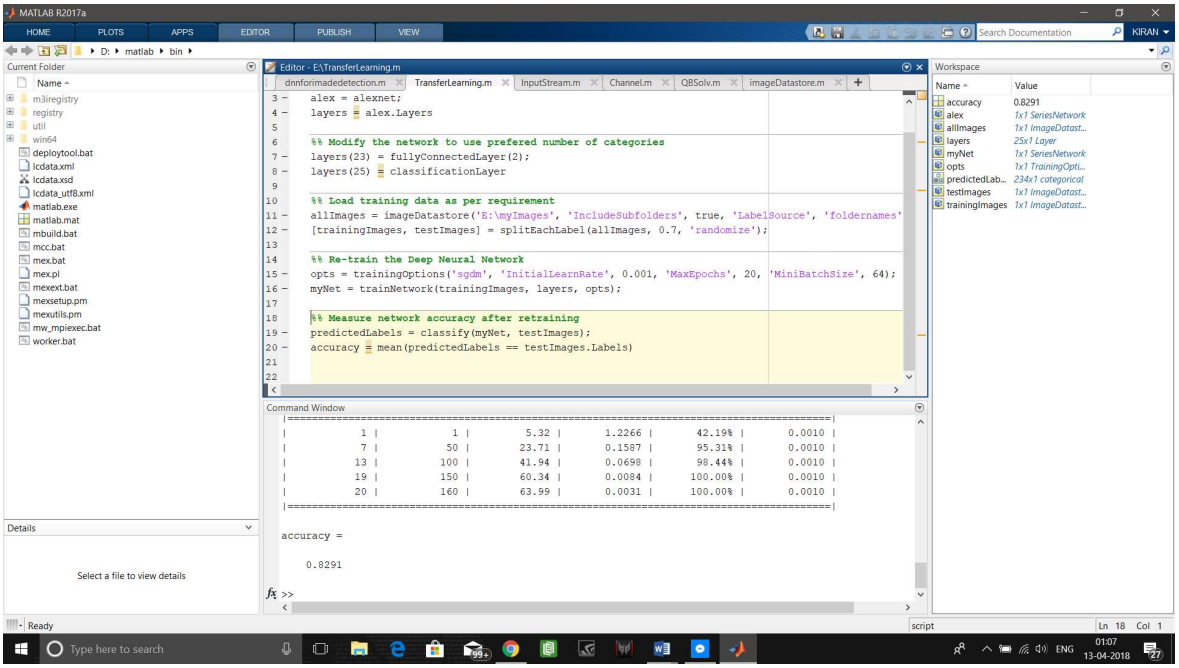


FIGURE 11:

Epoch	Iteration	Time elapsed(seconds)	Mini-batch loss	Mini-batch accuracy(in %)	Base learning rate
1	1	5.32	1.2266	42.19	0.0010
7	50	23.71	0.1587	95.31	0.0010
13	100	41.94	0.0698	98.44	0.0010
19	150	60.34	0.0084	100.00	0.0010
20	160	63.99	0.0031	100.00	0.0010

Final accuracy of diagnosis of tuberculosis= 82.9%

FIGU

REFERENCE

[1] Collobert, R. and Weston, J., 2008, July. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning (pp. 160-167). ACM.

[2] Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256).

[3] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[4] Yosinski, J., Clune, J., Bengio, Y. and Lipson, H., 2014. How transferable are features in deep neural networks?. In Advances in neural information processing systems (pp. 3320-3328).

[5] Simonyan, K., Vedaldi, A. and Zisserman, A., 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034.

[6] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[7] Ciregan, D., Meier, U. and Schmidhuber, J., 2012, June. Multi-column deep neural networks for image classification. In Computer vision and pattern recognition (CVPR), 2012 IEEE conference on (pp. 3642-3649). IEEE.

[8] Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural networks, 61, pp.85-117.

[9] Venkatasubramanian, V. and Chan, K., 1989. A neural network methodology for process fault diagnosis. AIChE Journal, 35(12), pp.1993-2002.

[10] Lakhani, P. and Sundaram, B., 2017. Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks. Radiology, 284(2), pp.574-582.