Efficient Segmentation and Classification of Hyper Spectral Cubes

Papagrigoriou Stylianos
stelios.papagrigoriou@gmail.com

Thesis submitted to
Technical University of Crete

in partial fulfilment for the award of the degree of

Polytechnical Diploma

in Electronic and Computer Engineering

Advisor : Professor M. Garofalakis
Co-Advisor : Assoc. Professor C. Balas
Co-Advisor : Assist. Professor Y. Papaefstathiou

Electronic and Computer Engineering Department
Akrotiri Campus,
73100 Chania, Crete Greece

July 2012

Acknowledgments

First and foremost, i would like to thank my family and dedicate them this work. If it had not been for their continuous and selfess support, the fulfillment of this thesis would not be possible

I would of course like to thank my teachers and advisors, Professor Minos Garofalakis, Assosiate Professor Costas Balas, and Assosiate Professor Yannis Papaeustathiou for their guidance and support during the implementation of this diploma thesis.

I am really grateful to George Epitropou, PhD Candidate for his non-stop guidance and help for the improvement of this thesis.

Finally, I wish to thank all my good friends at the Technical University of Crete and Chania for the wonderfull experiences we shared all those years.

Last but not least, I would like to thank my lovely Chryssa, for putting up with me all those years, and helping me achieve my goals. I wouldn't make it without you.

abstract

Hyper Spectral Imaging is a powerful analytical tool, which has been used in a wide area of applications, from Satellite Imaging to Biomedical Diagnosis. Spectral imagery of either macroscopic or microscopic origin is usually depicted in a spectral cube, a registered set of images, featuring one spectral and two spatial dimensions as pixel coordinates. From each pixel, associated with a spectrum -instead of an RGB value-, one is able to extract information about the nature of the material, by studying its spectral signature on the Spectral Cube.

This technique offers a non-destructive and non-invasive way (one does not have to extract part of the material and bring it to the lab) of examining materials, suitable for medical purposes.

In the hereby thesis the computational capabilities of spectral imaging methods are examined and attempted to be improved, in order to provide real time pixel classification. Specifically, a successful attempt is made to create a hyper spectral classifier with real-time performance for cubes acquired from a cervix biopsy.

Various techniques are tested for efficient segmentation of the Cube, in order to generate a golden standard for the training process. The classification is performed using Neural Networks while the final result is a GPU implementation, the main reason behind the speed up of the application.

Although this study was based on specific medical data, it is possible to be generalized on any aspect of Hyper Spectral Imaging, and shows that real-time Hyper Spectral Processing for classification purposes is feasible.

Contents

List of Tables

Chapter Information

Chapter 1 presents all the general and necessary concepts about (Hyper) Spectral Imaging, its applications and some foundation on mathematical tools used when adressing Hyper Spectral Imaging problems.

Chapter 2 provides an overview of the problem addressed by the thesis hereby, and presents a simplified view of the solution that will be elaborated further ahead at the following chapters.

Chapter 3 provides an introduction to Artificial Neural Networks and presents the reasons on why Neural Networks can be of use for this particular problem.

Chapter 4 describes the methodologies used in order to conclude to an appropriate Golden Standard and then train the classification system.

Chapter 5 describes the training process of the classifier.

Chapter 6 introduces the concept of GPUs and the reason they can be a great asset to dealing with the illustrated problem.

Chapter 7 summarizes the major findings of this thesis and provides suggested points for future work.

Chapter 1

Introduction

Before venturing into the research areas this thesis is occupied with, some basic introduction is necessary on the aspects of spectrometry and the basic linear algebra tools used. This Chapter, includes all these useful information and prepares the reader for the rest of this thesis. Readers already familiar with similar research, may skip the introduction and continue with the following chapters.

1 Spectroscopy/Spectrometry

Spectroscopy was originally the study of the interaction between radiation and matter as function of wavelength (λ). Historically, spectroscopy refered to the use of visible light dispersed according to its wavelength, e.g. a prism. Later the concept was expanded greatly to comprise any measurement of a quantity as function of either wavelength or frequency. Thus it also can refer to interactions with particle radiation or to a response to an alternating field or varying frequency (v). A further extension of the scope of the definition added energy (E) as a variable, once the very close relationship E=hv for photons was realized. Spectrometry is the spectroscopic technique used to assess the concentration or amount of a given species. In those cases, the instrument that performs such measurements is a spectrometer or spectrograph. Spectroscopy/spectrometry is often used in physical and analytical chemistry for the identification of substances through the spectrum emitted from or absorbed by them. Spectroscopy-/spectrometry is also heavily used in astronomy and remote sensing. Most large telescopes have spectrometers, which are used either to measure the chemical composition and physical properties of astronomical objects or to measure their velocities from the Dopple shift of their spectral lines.

2 Imaging Spectrometry

Imaging spectroscopy (also spectral imaging or chemical imaging) is the application of reflectance spectroscopy to every pixel in a spatial image. It can be considered as the equivalent of color photography, but each pixel needs to acquire many bands of light intensity data from the spectrum, instead of just

the three bands of RGB color model. More precisely, it is the simultaneous acquisition of spatially co-registered images in many spectrally contiguous bands.

Spectroscopy can be used to detect individual absorption features due to specific bonds in a solid, liquid, or gas. Solids can be either crystalline (i.e minerals) or amorphous (like glass). Every material is formed by chemical bonds, and has the potential for detection with spectroscopy. Actual detection is dependent on the spectral coverage, spectral resolution, and signal-to-noise ratio of the spectrometer, the abundance of the material and the strength of the absorption features for that material in the wavelength region measured.

Some spectral images contain only a few image planes of spectral data, while others are better thought of as full spectra at every location in the image. For example, solar physicists use spectroheliograms, images of the Sun built up by scanning the slit of a spectrograph, to study the behavior of surface features on the Sun; such spectroheliogram may have a spectral resolution of over 100,000($\lambda/\Delta\lambda$) and be used to measure local motion (via the Doppler shift) and even the magnetic field at each location in the image plane. The multispectral images collected by the Opportunity rover (NASA robotic rover on mars ongoing exploration missions), in contrast, have only four wavelength bands and hence are only a little more than 3-color images. To be scientifically useful, such measurement should be done using an internationally recognized system of units.

Spectroscopy can be used in laboratories on hand samples, in the field with portable field spectrometers (Spatial resolution in the millimeter to several meter range) from aircraft, or satellites. The aircraft systems now operational can image large areas in short time ($\tilde{2}$ sq. km per second!), producing spectra for each pixel that can be analyzed from specific absorption bands and thus specific materials. These measurements can then be used for the unambiguous direct and indirect identification of surface materials and atmospheric trace gases, the measurement of their relative concentrations, subsequently the assignment of the proportional contribution of mixed pixel signals (e.g., the spectral unmixing problem), the derivation of their spatial distribution (mapping problem), and finally their study over time (multi-temporal analysis).

Figure 1.1: Mantis shrimp possess hyperspectral color vision, allowing up to 12 color channels extending in the ultraviolet. Their eyes are considered to be the most complex eyes in the animal kingdom.

3 Hyper Spectral Imaging

For the last one hundred years detectors have been developed for radiation of almost any region of the electromagnetic spectrum. Recent developments in detector technology incorporate point sensors into integrated detector arrays, which allows setting up imaging radiometers instead of point measuring devices. Quantitive measurements of the spatial distribution of radiometric properties are now available for (remote) sensing at almost any wavelength. Unlike the

human eye, which just sees the visible light, Hyper Spectral imaging is more like the eyes of the mantis shrimp, which not only is able to detect and observe object in the visible light but also bands ranging from ultraviolet to infrared. These hyper spectral capabilities enable the mantis shrimp to recognize different types of coral, prey or predators, all which would appear as the same color to the human eye.

Humans build sensors and processing systems to provide the same type of capabilities. A wide range of tools have been developed so far with some of the most popular applications to be in agriculture, mineralogy, physics, surveillance and other fields of science.

Certain objects leave unique "fingerprints" across the electromagnetic spectrum. These fingerprints are known as spectral signatures and enable identification of the materials that make up a scanned object. For example, having the spectral signature for oil helps mineralogists find new oil fields.

Hyper spectral sensors look at objects using a vast portion of the electromagnetic spectrum and collect information as a set of 'images'.

Figure 1.2: The projection of a hyperspectral cube in two dimensions.

Each image represents a range of the electromagnetic spectrum and is also known as a spectral band. These 'images' are then combined and form a three-dimensional hyper spectral data cube for processing and analysis.

The precision of these sensors is typically measured in spectral resolution, which is the width of each band of the spectrum that is captured. If the scanner picks up on a large number of fairly small wavelengths, it is possible to identify objects even if said objects are only captured in a handful of pixels. However, spatial resolution is a factor in addition to spectral resolution. If the pixels are too large then multiple objects are captured in the same pixel and become difficult to identify. If the pixels are too small, then the energy captured by each sensor-cell is low, and the decreased signal-to-noise ratio reduces the reliability of measured features.

4 Measures of Spectral Similarity

Taking into account the features of a HSI, and excluding the spatial information, one can perceive the rest of the information as a group of n-dimensional vectors. As such, the similarity techniques that can be applied among the pixels are the same as those applied to multidimensional vectors. For the following examples some linear algebra rules will be defined.

Given a Matrix A:

- Transpose of A : $[A^T]_{i,j} = [A]_{j,i}$
- The product of an MxP matrix A with a PxN matrix B is an MxN matrix

denoted AB whose entries are :

$$(AB)_{i,j} = \sum_{k=1}^{p} A_{i,k} B_{k,j}$$

Where $1 \leq i \leq m$ is the row index and $1 \leq j \leq n$ is the column index.

- Inversion of A : A^{-1}, where $A^{-1} * A = I$
  For example a 2x2 matrix inversion is

$$A^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

- Pseudo-inverse matrix D : $D^{+} = (D^{T} D)^{-1} D^{T}$

For the purpose of this study the dimensionality of the vectors (n) equals the number of spectra.

4.1 Euclidean (L2 Norm) Distance

In mathematics the Euclidean Distance is the distance that can be measured with the use of a ruler between two points. The associated norm is the second order or Euclidean norm.

$$EuclideanDistance(p, q) = \|p - q\| = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

4.2 Root Mean Square Error (RMSE)

In statistics the Root Mean Square Error (RMSE) is one of many ways to quantify the difference between an estimator and the true value of the quantity being estimated. RMSE is a risk function, corresponding to the expected value of the squared error loss or quadratic loss, measuring the average of the square of the error. The error is the amount by which, the estimator differs from the quantity to be estimated.

$$RMSE(p, q) = \sqrt{\frac{\sum_{i=1}^{n} (p_i - q_i)^2}{n}}$$

4.3 Spectral Angle Mapper (SAM)

The spectral angle mapper (SAM) [8] is a physically based spectral classifier that determines the spectral similarity between the measured and the reference spectra. The spectra are treated as vectors in a space with dimensionality equal to the number of bands and the angle that is formed between these vectors is used as a metric of the spectral similarity (Figure 2.2).

Smaller angles represent closer matches to the reference spectrum. SAM has also been used as a feature selection method for selecting an optimal subset of spectral bands. [9]

The angle between pixel vectors as a discrimination measure

is given by the following formula:

$$\theta = SAM(p,q) = \arccos(\frac{p*q}{\|p\|\|q\|})$$

Measuring the **direction** and not the **length** of the vector, SAM becomes insensitive to illumination and the possible noise it might add.

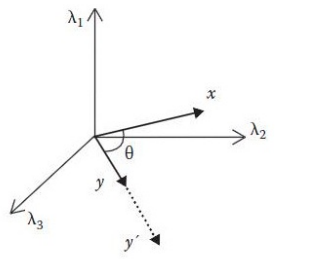


Figure 1.3: Vector representation of x,y pixels in a 3D orthonormal space corresponding to three different wavelengths. SAM essentially calculates the angle between them.

4.4 Spectral Correlation Mapper (SCM)

Another minimum distance classifier is spectral correlation mapper (SCM) or angle (SCA) [10] .SAM cannot distinguish between negative and positive correlations because only the absolute value is considered. SCM has been generated as an improvement on the SAM. The main difference is that SCM standardizes the data, centralizing itself in the mean of x and y.

SCM calculates a statistical measure of independence known as Pearson correlation coefficient. In probability theory and statistics, correlation indicates the strength and direction of a linear relationship between two random variables. SCM similarity metric is calculated using the following formula:

$$SCM(p,q) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

5 Hyper Spectral Imaging Applications

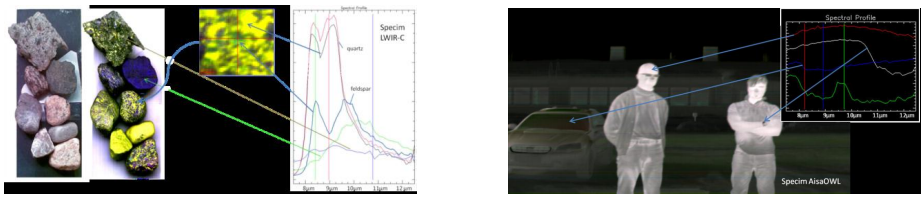
As mentioned already, there are numerous of applications that Hyper Spectral imaging can be used due to the importance of Spectral Information. Apart from the applications mentioned at the previous section, Hyper Spectral Imaging is nowadays commonly used in biomedicine.

A study made by Georgios N. Stamatas [2] in analysis of skin, uses Hyper Spectral Imaging devices in order to produce a reliable quantitative distribution map of chromophores contributing to the color appearance of the skin. A similar work has been made from Daisuke Nakao [3], in order to perform a Real-Time mapping pigmentation in human skin. This is expected to give very useful information about the reproduction of various skin colors as long as various human conditions.

An extremely powerful feature of Hyper Spectral imaging is that it can provide non-destructive analysis. This is really important considering that there are objects which need to be tested without extracting a physical sample as the extraction may damage or destroy the sample or even the object. Konstantinos Rapatzikos have created a non-destructive analysis method [4] in order to acquire images from manuscripts and detect Palimpsests (twice written manuscripts) which can reveal hidden text under the visible one.

Furthermore, recent work has proved that Hyper Spectral Imaging can be used to perform material or color-pigments classification. In the work of Adbelhameed Ibrahim [5], Spectral Information is used in order to classify the materials of printed circuit boards effectively. Another distinguished work is that of Giorgios Epitropou [6] where Hyper Spectral Imaging is used along with classification algorithms in order to perform a non-destructive analysis of El Greco's paintings.

Finally, this thesis is also occupied with the field of classification, trying to provide real time classification on Hyper Spectral Cubes acquired from photographing the cervix.



(a) (b)

Figure 1.4: (a) A set of stones is scanned with a Specim LWIR-C imager in the thermal infrared range from 7.7 um to 12.4 um. The quartz and feldspar spectra are clearly recognizable. (b) Hyperspectral thermal infrared emission measurement, an outdoor scan in winter conditions, ambient temperature -15ÂřC - relative radiance spectra from various targets in the image are shown with arrows. The infrared spectra of the different objects such as the watch glass have clearly distinctive characteristics. The contrast level indicates the temperature of the object. [1]

Concluding. It must be obvious by now, that the importance of Hyper Spectral Imaging is great. Unfortunately, Hyper Spectral Imaging devices are still very heavy and expensive machines and are mostly restricted to static applications. In addition, these devices cannot be used in real time because of their big acquisition and post-acquisition processing time in order to provide the spectral data of an object and then the huge computational cost of processing the data produced.

Bibliography

[1] Holma, H., *"Thermische Hyperspektralbildgebung im langwelligen Infrarot"*, Photonik '11.

[2] Georgios N. Stamatas, Costas J. Balas, Nikiforos Kollias, *" Hyperspectral image acquisition and analysis of skin".*

[3] Georgios N. Stamatas, Costas J. Balas, Nikiforos Kollias, *"Hyperspectral image acquisition and analysis of skin".*

[4] Daisuke Nakao, Norimichi Tsumur, Yoichi Miyake, *"Real-Time multispectral image processing for mapping pigmentation in human skin".*

[5] Konstantinos Rapantzikos, Costas Balas, *"Hyper Spectral Imaging : Potential in non-destructive analysis of palimpsests".*

[6] Abdelhameed Ibrahim, Shoji Tominaga, Takahiko Horiuchi, *" Unsupervised Material Classification of Printed Circuit Boards Using Dimension-Reduced Spectral Information".*

[7] George Epitropou, *"Hyper Spectral Imaging and Spectral Classification algorithms for the non-destructive analysis fo El GrecoâĂŹs paintings".*

[8] Kruse, F. A., Lefkoff, A. B., Boardman, J. W., Heidebrecht, J. W., Shapiro, K. B., Barloon, A. T., and Goetz, P. J. , The spectral image processing system (SIPS)-Interactive visualization and analysis of imaging spectrometer data, 1993, RemoteSensing Environ. 44: 145-163.

[9] Keshava, N., *Distance metrics and band selection in hyperspectral processing with applications to material identification and spectral libraries,*, 2004, IEEE Trans. Geosci. Remote Sensing 42(7): 1552-1565 .

[10] Osmar AbÃŋlio de Carvalho Jr,Paulo Roberto Meneses, *Spectral Correlation Mapper (SCM): An Improvement on the Spectral Angle Mapper (SAM)*, 2000

Chapter 2

The problem at hand.

For the past few years Hyper Spectral Imaging has offered a non-destructive way of analyzing and diagnosing diseases and pathogenecies. A number of methods have been introduced that can take advantage of the electromagnetic "fingerprint" of infected cells in order to provide classification or segmentation services. For example [1], shows succesfully classification of tongue diseases with the use of hypespectral medical tongue images.

Creating such a classifier introduces two great advantages:

- **Biopsy free diagnosis**.

  Solely component of such dignostic methods remains the electromagnetic spectrum that is emmited after the interaction of photons with the tissues. Because of that, a sample can be taken without the need of tissue extraction or any further processing.

- **Removing the ad-hoc human error**.
  Using humans (doctors) as classifiers unfortunatelly and inevitably introduces an ad hoc error based on the doctor's experience, possible fatigue, loss of concentration etc. Computer based classifiers, provide a constant efficiency that is not affected by all those ad-hoc parameters.

Despite the great advantage that might seem motivating for our research, creating such a classifier should be made carefully. A lot of testing is needed in order to guarantee, the performance of the classifier in real and unknown conditions.

1 Overview of the problem.

In the case of this thesis a classifier has to be made in order to provide classification on images taken from cervix tissue. The main idea behind the classifier is to provide effective diagnosis by classifying the pixels into four classes depending on the condition of the cells contained into that pixel. After the classification, a color map shows specifically the location of those four classes on the acquired picture while a "diagnosis" has to be made, by calculating the percentage and intensity of the class over the whole body of pixels.

A series of problems have to be taken into account such as, the preprocessing of images in order to remove noise, the creation of a proper dataset, the training of the classifier and proper testing which will guarantee its performance.

Real-Time Performance. The fact that most of the multi spectral processing methods require off line processing provides a limited approach to a large number of applications with real time needs. For example in the case of colposcopy saving a several-minute-video and a few hundreds or thousands of hyper spectral images for off line processing might not be the most efficient way. In off line processing, the doctor cannot have immediate feedback on the under-examination area which means that he/she is not able to pause to a specific point in order to examine better or even zoom for a better perspective.

A succesful attempt is made in this thesis to provide Real Time classification for the first time on such applications.

2 Acquiring Hyper Spectral Cubes

As mentioned already, Hyper Spectral Imaging (SI) produces Data Cubes - Hyper Spectral Cubes by photographing the object of interest on a wide range of the electromagnetic spectrum. The outcome of this procedure is the production of a 3-D matrix (thus cube) where the first two dimensions hold the spatial details of each pixel and the third dimension holds the electromagnetic fingerprint of the pixel in the form of a vector. The vector is as long as many bands we have scanned during the acquisition of the cube. The number of bands of interest (as mentioned at the introduction) is based on the nature of the problem.

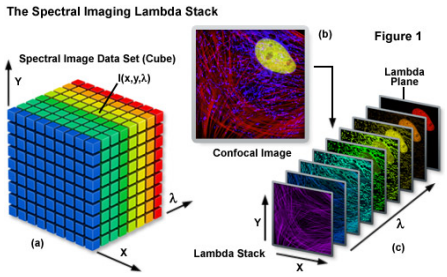


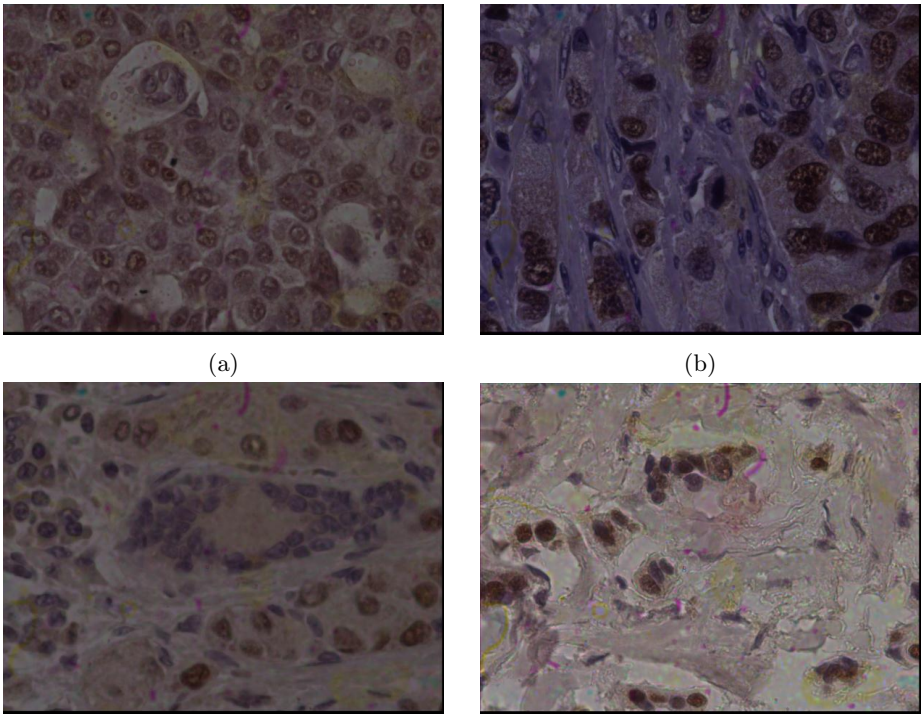
Figure 2.1: Two dimensional spatial projections across the electromagnetic spectrum.

Hyper Spectral Cubes can be acquired from stationary or portable spectrometers and then stored for offline processing. Although theoretically Hyper Spectral Cubes (HSCs) can have information spanning in a wide area of the electromagnetic field and having hundreds of bands scanned, for practical applications smaller cubes are used.

For the purposes of this thesis, HSCs were acquired by biopsies taken from the cervix. After the biopsy, special inks are used that will color cells under not-normal conditions in different colors. The doctors then observe the coloring of the sample and based on their experience and training, produce a diagnosis.

In order to create the needed HSCs, those samples were photographed in 16 bands ranging from 440nms to 720nms. The reason that those bands were selected, is because, wavelengths longer than 800nm, seem to give no significant or useful information. This is due to the nature of the cells and the dominant presence of water.

The samples were given by many Greek hospitals and the acquisition was made using a custom modified Mu SIS HS camera. A total of 24 photos were used, which can be grouped in four categories, depending on the clinical condition of the patient and the amount of cancer cells in the picture. The following pictures give an example of each one of the conditions.



(a) (b)

(c) (d)

Figure 2.2: Examples of the samples for the purposes of this thesis. The samples are divided into four categories depending on the severity of the condition of each tissue. At the first picture (a), a 10% of cancerous cells can be found, while pictures (b), (c) and (d) have 60%, 80% and 98% respectivelly .

3 Classes

In the figure 2.2, 3 colors are mostly notable : Blue, Red and White. The colors occur based on the interaction of special inks with the substances present on the specific part of the tissue.

- **Class A : DAB**
  DAB substance corresponds to the *red color* and is the main class that needs to be tracked.

- **Class B : Hematoxylin**
  Hematoxylin substance corresponds to the *blue color*.

- **Class C : Background**
  The final strictly defined class refers the background and corresponds to the *white color*.

- **Class D : Unspecified**
  Some parts of the image may contain a combination of those two substances (DAB + Hematoxylin). For that reason the final color produced seems to be a combination of blue and red. These parts fall into the Unspecified category.

The final percentage of each class, (technically) is calculated not over the whole body of pixels, but over the pixels that fall under the Class A, Class B and Class D categories. Class C (Background) is excluded from this calculation.

4 Proposed solution

Computational Cost. After examining the features of a hyper spectral cube, it is obvious that even simple correlation acts, have high computational costs, while by the nature of the cube we are affected by the "curse of dimensionality". Having in mind the real time perspective, grouping techniques (like clustering) or other techniques that use comparisons exhaustively are not an option. Despite the fact that a smart solution has to be found, some strong computational machine has to be introduced, in order to deal with the heavy load brought by the said "curse of dimensionality".

Proposed Way of Action. In order to deal with those problems above, this thesis proposes the use of neural networks with the use of GP-GPUs (General Purpose Graphical Processing Units) as means to heavy computational problems. The specific reasons, why those tools were chosen, will be made clear at the following chapters. For now the reader can be sure that neural networks are extensively used for pattern recognition problems while for the last few years GP-GPUs are a powerful asset in solving scientific and medical problems.

For more information on neural networks or GPUs, please refer to Chapter3 or Chapter 5 respectively.

Bibliography

[1] Zhi L, Zhang D, Yan JQ, Li QL, Tang QL, *Classification of hyperspectral medical tongue images for tongue diagnosis*, 2000

Chapter 3

Artificial Neural Networks as a classification tool.

An Artificial Neural Network, usually called Neural Network, is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. An ANN consists of interconnected group of artificial neurons, and it processes information using a connectionist approach to computation.

In more practical terms neural networks are non-linear, statistical data modeling and decision making tools. They can be used to model complex relationships between inputs and outputs or to find patters in data.

However, the paradigm of neural networks - i.e., implicit, not explicit , learning is stressed - seems more to correspond to some kind of natural intelligence than to the traditional symbol-based Artificial Intelligence, which would stress, instead, rule-based learning.

1 Introduction

Artificial neurons were first proposed in 1943 by Warren McCulloch, a neurophysiologist, and Walter Pitts, a logician, who first collaborated at the University of Chicago [1].

In modern software implementations of artificial neural networks the approach inspired by biology has more or less been abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks, or parts of neural networks are used as components in larger systems that combine both adaptive and non-adaptive elements.

The concept of a neural network appears to have first been proposed by Alan Turing in his 1948 paper "Intelligent Machinery".

Applications of natural and artificial neural networks. The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical.

The tasks artificial neural networks are applied to tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction, fitness approximation and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind source separation and compression.
- Robotics, including directing manipulators, Computer numerical control.

Application areas include system identification and control (vehicle control, process control, natural resources management), quantum chemistry,[2] game-playing and decision making (backgammon, chess, poker), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications (automated trading systems), data mining (or knowledge discovery in databases, "KDD"), visualization and mail spam filtering.

Artificial neural networks have also been used to diagnose several cancers. An ANN based hybrid lung cancer detection system named HLND improves the accuracy of diagnosis and the speed of lung cancer radiology [3]. These networks have also been used to diagnose prostate cancer. The diagnoses can be used to make specific models taken from a large group of patients compared to information of one given patient. The models do not depend on assumptions about correlations of different variables. Colorectal cancer has also been predicted using the neural networks. Neural networks could predict the outcome for a patient with colorectal cancer with a lot more accuracy than the current clinical methods. After training, the networks could predict multiple patient outcomes from unrelated institutions [4].

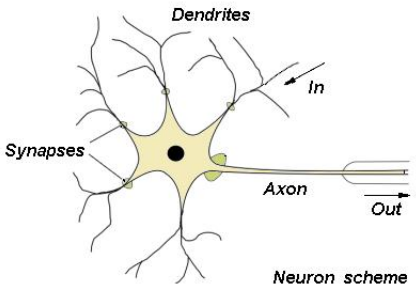
Disadvantages. A common criticism of neural networks, particularly in robotics, is that they require a large diversity of training for real-world operation. This is not surprising, since any learning machine needs sufficient representative examples in order to capture the underlying structure that allows it to generalize to new cases.

Dean Pomerleau, in his research presented in the paper "Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving," uses a neural network to train a robotic vehicle to drive on multiple types of roads (single lane, multi-lane, dirt, etc.). A large amount of his research is devoted to (1) extrapolating multiple training scenarios from a single training experience, and (2) preserving past training diversity so that the system does not become over trained (if, for example, it is presented with a series of right turns - it should not learn to always turn right).

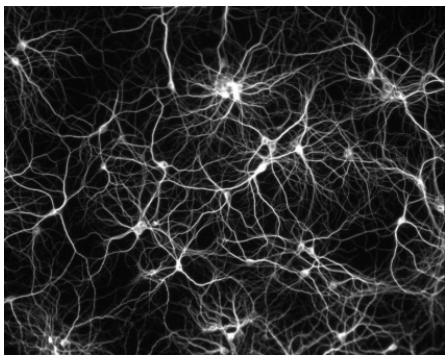
These issues are common in neural networks that must decide from amongst a wide variety of responses, but can be dealt with in several ways, for example by randomly shuffling the training examples, by using a numerical optimization algorithm that does not take too large steps when changing the network connections following an example, or by grouping examples in so-called mini-batches.

2 Architecture and functionality of Neural Networks

2.1 A Simple Neuron



(a)



(b)

Figure 3.1: Figure (a) shows the anatomy of a brain neuron. Input signals come from Dendrites while the output signal comes out of the Axon. Figure (b) represents the complex networks, created by the connections of millions of neurons.

The human brain consists of 10^{10} neurons and 10^{16} synapses (connections) among those cells. Each neuron can have 1 or more input connections, coming from other neurons. Signals coming from its input dendrites force the neuron to be activated (by producing an electrical pulse from its Axon) or not.

Neuron Model: Logistic Unit. In order to represent a single neuron with a model, we have to simplify its functionalities. In that perspective, it is obvious that the neuron does the following processes:

1. Sums up its inputs.
2. Possible some inputs have different weighted value than others.
3. It activates its axon or not, depending on the previous values.

Modern day Neural Networka. In modern day ANNs each artificial neuron is nothing more than a unit that produces an exit according to its input and its activation function.

$$Exit = ActivationFunction(Input).$$

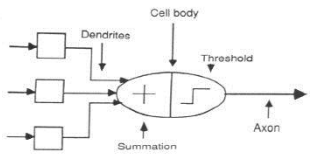


Figure 3.2: A crude equivalent to a natural neuron

Input of the neuron is the summation of the exit of the neurons connected to that one, multiplied by weight of their connection.

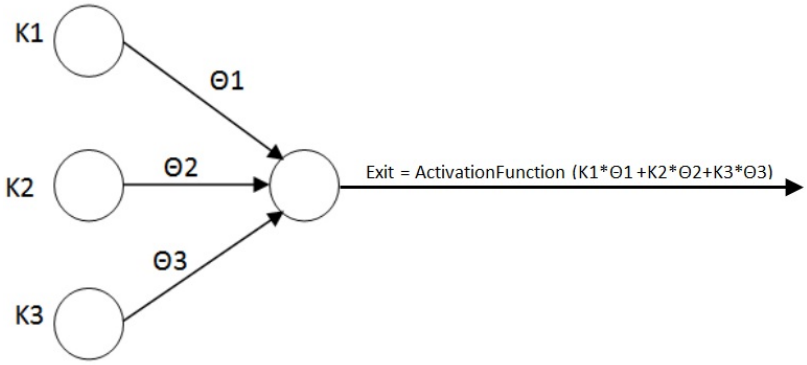
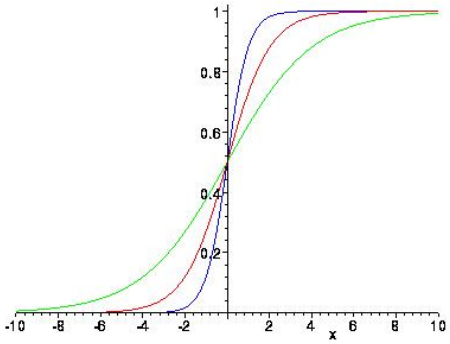


Figure 3.3: Output of a neuron, considering the input streams that end to that neuron.

Figure 3.4: Sigmoid/Logistic Function
It is represented by the following curve:

$$f(x) = \frac{1}{1+e^{-s*x}}$$

The variable s represents the "steepness" of the curve. The larger the s the steeper it is.



Activation Functions. The most commonly used activation function is the Logistic (Sigmoid) Function since it's the closest approximation to a real neuron. Other popular functions that are widely used are the linear function, Symmetric Sigmoid Function, Step Function and Gaussian. Figure 3.5 represents their plot at [-10, 10].

2.2 Feed-Forward networks

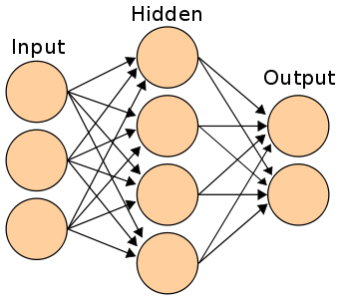
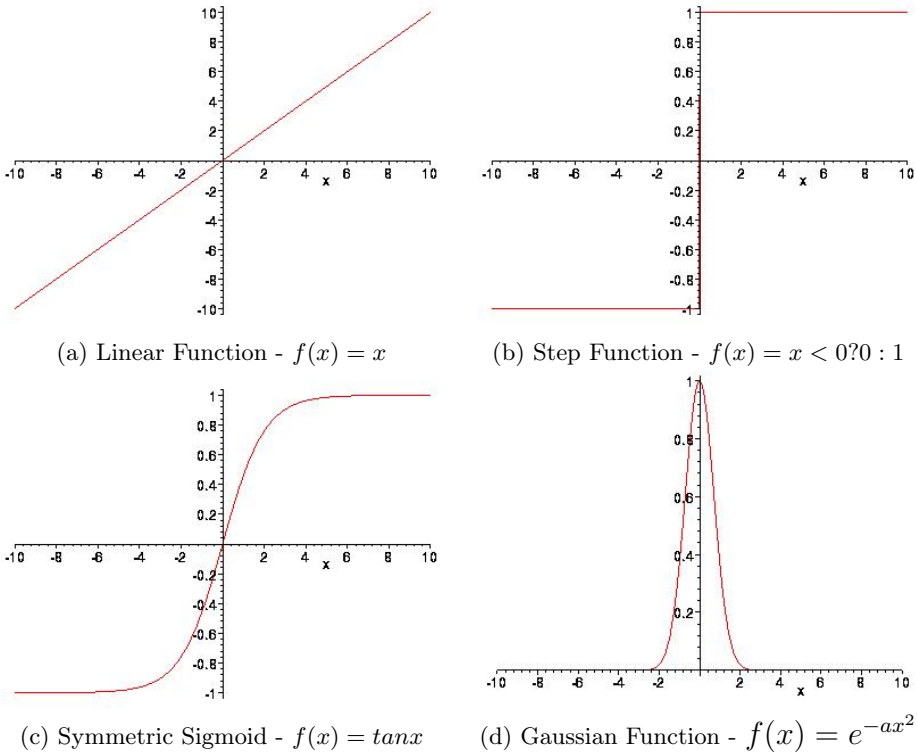


Figure 3.6: A typical ANN with one input layer, two Hidden Layers and one output layer.

In order to have efficiency, neurons alone are not enough. They have to be tailored into networks, connected together. Those can be made by placing neurons into layers and then those layers one after the other. It consists of at least an input and an output layer and in between them one or more Hidden Layers.

The most common layout of neural networks is the feed-forward network. In this layout, neurons of each layer are only connected with neurons from the previous layer and no recursively feedback connection is applied.

The signal starts from the input layer, interacts with the neural network and gives a final output at the exit of the last layer. For classification purposes,



(a) Linear Function - $f(x) = x$

(b) Step Function - $f(x) = x < 0?0 : 1$

(c) Symmetric Sigmoid - $f(x) = tanx$

(d) Gaussian Function - $f(x) = e^{-ax^2}$

Figure 3.5: The most common neuron activation functions.

the final (exit) layer has as many neurons as classes it has to classify. In order to perform the classification, it is trained in a way that only the neuron that corresponds to the class it predicts has output = '1' while the rest of the neurons give output = '0'.

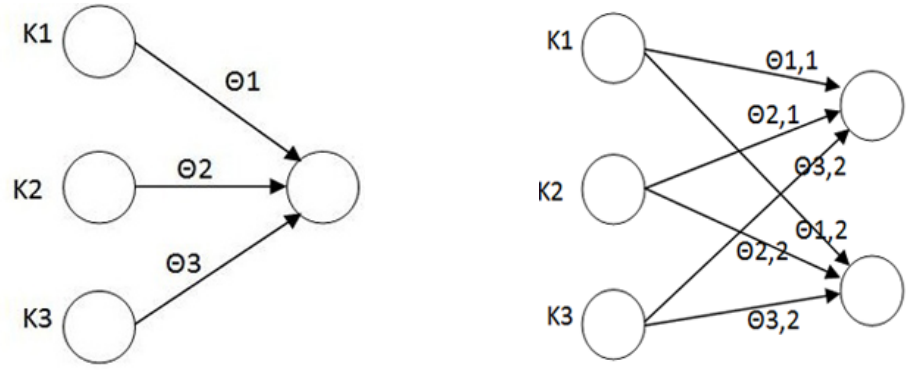
As mentioned before the input signal of a neuron is the summation of the output from the neurons of the previous layer multiplied by the weights of each connection. Assuming a neuron with N inputs from the previous layer and two matrices; a 1xN matrix, K that contains the N neuron outputs of the previous layer and Θ an Nx1 matrix that contains the weights of each connection. The total input to the neuron is the product of those two matrices:

$$Output = ActivationFunction(K * \Theta)$$

Interestingly enough, the Θ matrix can be changed, from Nx1 to NxM, where M equals the number of neurons at the current layer. The new product of $K * \Theta$ is an 1xM vector that will contain the output for the whole layer.

3 Over-fitting / Under-fitting

For every network the number of neurons represents it's capability to recognize and comprehend complex patterns. The more the neurons the more patterns can be recognized but also the more training it requires.



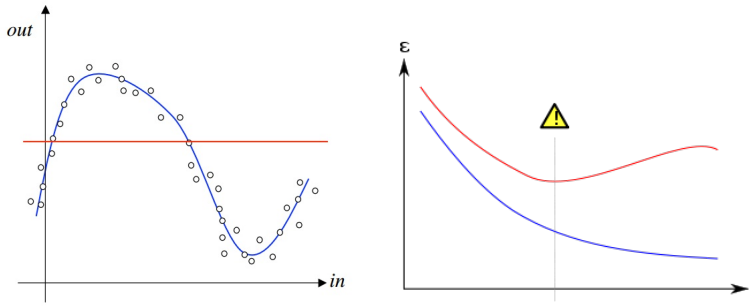
(a) (b)

Figure 3.7: In both cases $Output = ActivationFunction(K * \Theta)$. The only thing that changes is the size of Θ.

By using too few neurons, the neural network might not be able to recognize the patterns present at the dataset while by using too many neurons, the neural network might simply memorize the input output cases.

In the case of too many neurons being present, while not enough training examples are available, the ANN is said to over-fitting the data, while in the case of not enough neurons being present to generate a good approximation of the output, the ANN is said to under-fitting the data.

In both cases, the capability of the ANN to generalize well to new data is crippled.



(a) (b)

Figure 3.8: Examples of two extreme cases of fitting. The first case (a) provides a really poor approach to the required approximation function (displayed with blue). It is an extreme example of under-fitting. The second case (b) provides an over-fitted network to the given dataset. Both those cases, provide poor generalization rules.

Over-fitting or Under-fitting can be prevented by monitoring the performance of the ANN during its training process. More presicelly by splitting the training set into two part. The training set and the cross-validation set. The new training set will be used to train the ANN while the cross-validation set

will be used as an "unknown" set to test the performance of the network on generalizing the data.

The goal is to prevent over-fitting by detecting when the error at the cross validation set starts to rise, while the error at the training set, keeps diminishing. At that point the ANN starts overfitting the data.

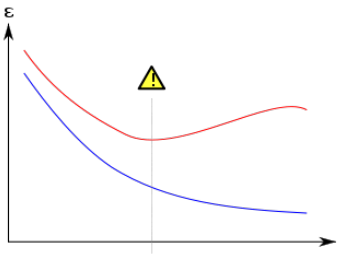


Figure 3.9: Error behavior on cross-validation and training sets.

Akaike information criterion. A good measure of the relative goodness of fit is Akaike's Criterion. It was developed by Hirotsugu Akaike, under the name of "an information criterion" (AIC), and was first published by Akaike in 1974.[9] It is grounded in the concept of information entropy, in effect offering a relative measure of the information lost when a given model is used to describe reality. It can be said to describe the tradeoff between bias and variance in model construction, or loosely speaking between accuracy and complexity of the model.

AIC values provide a means for model selection. AIC does not provide a test of a model in the sense of testing a null hypothesis; i.e. AIC can tell nothing about how well a model fits the data in an absolute sense. If all the candidate models fit poorly, AIC will not give any warning of that.

4 Why using Neural Networks.

The bibliography gives a lot of classification options each one better tailored for different situations. The scientist or engineer has to brainstorm and experiment in order to get the best results.

For the problem at hand, hyper spectral images/cubes have to be classified based on a 16 spectra input for each pixel. The classification has to be **accurate and in real-time**, something that requires not only a smart and elegant solution but also an implementation that will exploit the way the modern hardware works. For that reason neural networks were chosen.

Accuracy. The reader has to realize that Neural Networks have already been tested, used and been proven to work for this particular problem for decades. Every doctor that is diagnosing, at this moment incidents by examining and "classifying" pictures at the microscope, is using a form of Neural Networks in his mind. With the use of computers an attempt is made to use this human intuition, simulate the procedure, train it with millions of examples and remove the "human error".

Although this gives an intuition on why to choose neural networks, a question arises on whether the performance of neural networks can be improved. After all, the human approach of neural networks has a lot of errors and misclassifications. Can neural networks develop a classification model that will be able to perform accurately for this particular problem?

According to the cybenko theorem [5] the standard multilayer feed-forward network with a single hidden layer, which contains finite number of hidden neurons, is a universal approximator among continuous functions on compact

subsets of Rn, under mild assumptions on the activation function. The only serious constrains would be the size of the dataset size.

Need for Speed. As showed at paragraph 3.2 the output of a layer$L_i$can be found by the product $K * \Theta$, where K is a vector containing the outputs of the layer$L_(i-1)$and Θ is an N x M matrix containing the weights of the connections among the N neurons of $L_(i-1)$ layer and M neurons of L layer.

It is possible to increase the dimensionality of K from 1 x N to X x N where X is the number of pixels that need to be classified. The product $K*\Theta$ now gives an X x M matrix, which contains the output of a layer for the whole amount of pixels to be classified. Each row i will contain the output for the i-th pixel and each column j will contain the information of the j-th neuron output of the layer.

Thus we can get a classification result simple by doing as many products as the number of layers in the network.

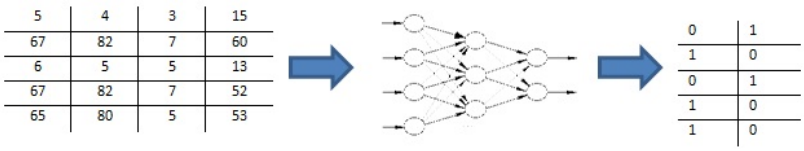


Figure 3.10: Passing the whole dataset through the network, by doing products.

In the problem at hand, we need to classify a 2MPixel Hyper Spectral picture. The picture is represented by a 1200x1600x16, 3-D matrix, which can easily be transformed into a $(1200 * 1600)x16$, 2-D matrix, which can then be treated as the matrix K at our examples.

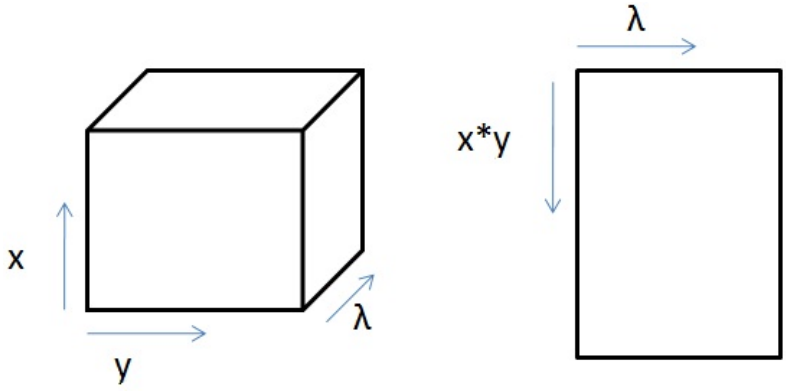


Figure 3.11: Transforming the cube into a two dimensional table.

This feature of the neural network output computation with vectorisation, allows the exploitation of powerful hardware tools and architectures, libraries and techniques which were developed and continue to develop for years.

5 Training Algorithms

A lot of training algorithms have been developed for training with the most popular being the back propagation algorithm.

5.1 Backpropagation

Backpropagation [6] is an abbreviation for "backward propagation of errors" and is a generalization of the delta rule. Backpropagation requires that the activation function used by the artificial neurons be differentiable.

The algorithm is as follows and requires two phases :

Phase 1: Propagation Each propagation involves the following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

2. Backward propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

Phase 2: Weight update For each weight-synapse follow the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.

2. . Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.

Repeat phase 1 and 2 until the performance of the network is satisfactory.

There are two modes of learning to choose from: One is on-line(incremental) learning and the other is batch learning. In on-line(incremental) learning, each propagation is followed immediately by a weight update. In batch learning, many propagations occur before weight updating occurs. Batch learning requires more memory capacity, but on-line learning requires more updates.

Although backpropagation seems easy to implement and gives good training results to a lot of problems, it provides a slow and not-guaranteed convergence of the network. Furthermore, with the batch learning technique the result may generally converge to any local minimum on the error surface, since stochastic gradient descent exists on a surface which is not flat.

5.2 Other Training Algorithms

Despite the fact that backpropagation provided a renaissance for the neural networks community and was reigning as a dominant training algorithm for a long time the need for better algorithms and faster convergence leaded to the creation of new algorithms.

One of the best training algorithms is the **Resilient Backpropagation** (or Rprop) [7], a first-order optimization algorithm. Rprop is considered to be the best choice in a lot of problems and was created by Martin Riedmiller and Heinrich Braun in 1992.

Another one of the better training algorithms is the **Quick Propagation** (or QuickProp) [8], which is loosely based on Newton's method for finding the root of a quadratic function.

Bibliography

[1] McCulloch, Warren; Pitts, Walter, A Logical Calculus of Ideas Immanent in Nervous Activity, 1943, Bulletin of Mathematical Biophysics 5:115-133.

[2] Roman M. Balabin, Ekaterina I. Lomakina , *Neural network approach to quantum-chemistry data: Accurate prediction of density functional theory energies*, 2009,J. Chem. Phys. 131 (7): 074104.

[3] Ganesan, N., *Application of Neural Networks in Diagnosing Cancer Disease Using Demographic Data*, International Journal of Computer Applications

[4] Bottaci, Leonardo, *Artificial Neural Networks Applied to Outcome Prediction for Colorectal Cancer Patients in Separate Institutions*, The Lancet

[5] Cybenko., G., *Approximations by superpositions of sigmoidal functions,* , 1989, Mathematics of Control, Signals, and Systems2 (4), 303-314

[6] RaÃžl Rojas, *The backpropagation algorithm of Neural Networks - A Systematic Introduction* , Chapter 7

[7] Martin Riedmiller , *Rprop âĂŞ Description and Implementation Details*, 1994,Technical report.

[8] Scott E. Fahlman, *An Empirical Study of Learning Speed in Back-Propagation Networks*, 1988.

[9] Akaike, Hirotugu, *A new look at the statistical model identification*, 1974, IEEE Transactions on Automatic Control 19 (6): 716âĂŞ723.

Chapter 4

Obtaining the proper Dataset

One of the most challenging problems addressed during this thesis, was to highlight the right dataset that will act as a solid base for the class creation.Despite of the algorithm used, the backbone of a proper classifier remains the extraction of the features on which the classifier will rely in order to identify the patterns and the classes on which the classifier will learn to operate with.

As analyzed in section 2.2, the selected features to participate are 16 bands of the electromagnetic spectrum, ranging from 440 nms to 720 nms. The problem though, of creating a solid and clear dataset that will serve as a training/testing set remains.For that reason a Golden Standard had to be created, which would dictate the proper shape of the classes and train the classifier. Part of this Golden Standard will be used as a training set and part of it as a testing set.

1 Creating a Golden Standard

Golden Standard in Medical Data. As mentioned already, during the prediction and diagnosis of the doctor, a series of errors are included based on a series of reasons. Statistical tests showed that two diagnoses on a sample by the same doctor can have up to 60% deviation. This makes obvious the fact that an objective solution cannot be found, which makes the creation of a Golden Standard extremelly difficult.

Despite that dificulty, an attempt was made to explore some ways of creating a usable and satisfactory Golden Standard by using the Hyper Spectral Cubes (HSC) mentioned at Chapter 2. Before the use of those HSCs, some preprocessing is required in order to clear the "noisy" camera output.

1.1 Preprocessing the data

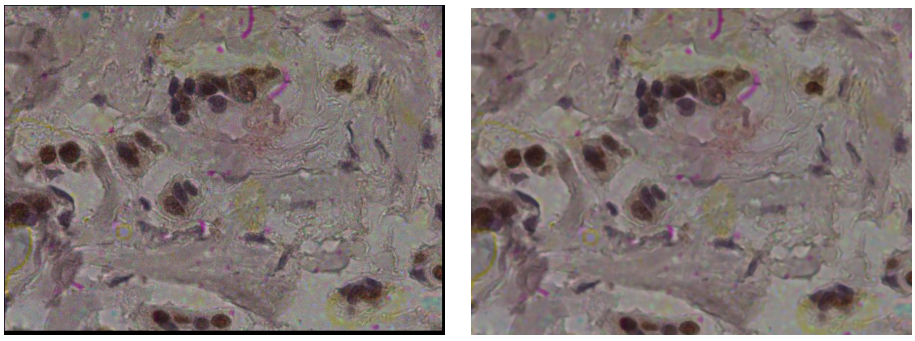
Before continuing, the reader has to understand that it is not possible for the humans to observe the whole HSC in order to recognise possible "noise" or other misunfortunate conditions. Considering though that the noise is mostly from technical reasons, such as the error introduced by the camera (which affects the whole range of the spectrum) and the misuse of the paint used for the creation of the samples (which is something really obvious at the visible spectrum), RGB pictures (cubes with only 3 out of the total 16 bands) can be used in order to

give a "visible" insight on the conditions of cube and thus, an insight on what kind of preprocessing is necessary.

Noise. The HSCs generated have a light "Salt and Pepper" noise, something inevitable due to the process of generating those samples from the biopsy. The ink used to paint the samples, might accidentally paint some cells in the wrong way based on its concentration, the texture of the glass etc. Median filter was used in order to remove this noise without altering the nature of the signal on the picture, and different window sizes were tested, in order to conclude to the best one. It appears that the intensity of noise varies among different Hyper Spectral Cubes, but a window size of 3 up to 5 pixel was always enough to remove it.

Damaged Borders. Apart from the above, it seems that the outer pixels which reside at the borders of the picture suffer from noise of great intensity. Because of that, 25 pixels were cropped out from the borders of all the HSCs.

Figure 4.1 shows the original picture and the outcome of the preprocessing stage.



(a) (b)

Figure 4.1: Figure (a) displays a representation of the original picture (and HSC) while (b) displays the result of the filters and the cropping required in order to remove the noise. It is obvious that (b) provides a clearer and smoother environment to extract the training dataset.

1.2 Extracting the Classes

In order to obtain the classes, 3 methods were used :

- **Hand - Picking Areas.**

  The first attempt was a try to imitate the doctors' behavior, by selecting areas that would be more than obvious to contain pixels of a single particular class. The RGB pictures can be used, the same way doctors use it, in order to pinpoint and pick areas of the image - and data cube - with the same color and thus same class.Figure BLABLA shows a manually selected area.

- **Clustering - Segmenting the image and manually associating classes.**

  Using clustering techniques, the picture can be segmented in areas with similar vector values. For the clustering part, Spectral Clustering and K-Means with four different kernels were used, in order to find the best possible result. In every case, the clustering algorithm was asked to partition/cluster the data, in eight classes (not four). After the clustering process, those eight classes were grouped together to form the four main ones, or ignored as too noisy or redundant. The grouping/ignoring process was done manually, and using the RGB pictures for reference.

- **Clustering - Segmenting the image and associate classes comparing with reference vectors.** Similar to the above notion, with the only difference being the way of grouping/ignoring the classes. A small database of 120 "reference" vectors was created, by manually extracting vectors that resided at places of the image where the classes could be distinguished clearly. The centroids of each of the ten classes created during the clustering, were compared to those 100 reference vectors. Each of the ten classes would then be assigned to the class of the reference vector that its centroid is closest to.

  The Reference Vectors Database is composed by 120 vectors - hand picked from all the picture categories- and refer to the most representative cases of each class.

  In order to create Class D ("unspecified") and ignore possible noisy or redundant classes two similarity thresholds were set. If a centroid's similarity comparing to every reference vector was lower than the first threshold (but not the second one), its class would be set as Class D. If a centroid's similarity comparing to every reference vector was lower than the second threshold, then its class would be ignored and not counted as an addition to the primary classes.

2 Segmentation Algorithms and their Efficiency

In order to provide efficient segmentation the following clustering techniques were tested:

K-Means with L1 and L2 norms. Typical K-Means [1] implementation using a norm as a metric. In this case, both the Manhattan (L1) and Euclidean (L2) distances provided the same segmentation, partitioning the images the same way. Unfortunately, looking at figure 4.2.a the performance of L1/2 norms as a metric, is not as satisfying as it should be.

K-Means with SAM. For a second approach, SAM (Spectral Angle Mapper) was used as a metric. The motivation was that, considering SAM's ability to ignore possible illumination or other tensity-related noise, it would be really fitted for such applications. Looking at figure 4.2.b, it is obvious that SAM

outperforms Euclidean and Manhattan distance and provides clear segmentation and partitioning of the image at parts with single classes.

K-Means with SCM. SCM (Spectral Correlation Mapper) was introduced by bibliography as an improvement of SAM, so another approach using it was made. Its performance is illustrated at figure 4.2.c

Spectral Clustering. Finally, Spectral Clustering [2] was tried and its outcome can be seen at figure 4.2.d . A k-nearest graph was used, with Gaussian function. Different numbers of k (15 to 150) were tried, but no big difference was noted. Although, Spectral Clustering, gives a very distinguished shape for regions affected by both types of pathogenesis (Class A, B), it fails to distinguish those two.

It is possible that with heavy tuning and testing Spectral Clustering can be guided to provide the performance of the previously illustrated segmentations, but its heavy computational and memory cost in doing so, are really unnecessary. For that reason, further tuning and testing of Spectral Clustering was abandoned.

For the choice of the best - most efficient segmentation technique-, no absolute metrics can be invoked but rather a choice based on what -literally- looks best has to be made. Out of the comparing four cases (some examples are displayed at Figure 4.2), SAM is the best, providing really accurate partitioning for the most cases. Following SAM, the Euclidean distance, also shows some potential when used along with the grouping technique described at 1.2. For that reason, K-Means with SAM and L2 were selected for further testing.

3 Creating the Dataset

Out of the 3 partitioning techniques described at 1.2 of this chapter, the first technique (manually selecting areas of the cubes) was abandoned for being too biased and for introducing no proper way of automatically producing datasets from a big number of pictures/HSCs. The other two were tested seperatelly. (Segmentation and then manually selecting an grouping the classes)

3.1 Tests

K-Means with SAM and Euclidean Distance was used to produce extra partitioned HSCs. The partitioning was extended to 8 classes instead of 4 to provide further accuracy, while for the second category of pictures (60% of the cells with pathologic condition), the partitioning was extended to twelve classes.

During the manual grouping, it was noticed that even on partitioning with a lot more classes that originally needed, SAM greatly outperforms Euclidean Distance as a metric something that came in conflict with the original idea the author had, that Euclidean Distance as a metric could be better with higher level clustering. Because of that Euclidean Distance was abandoned.

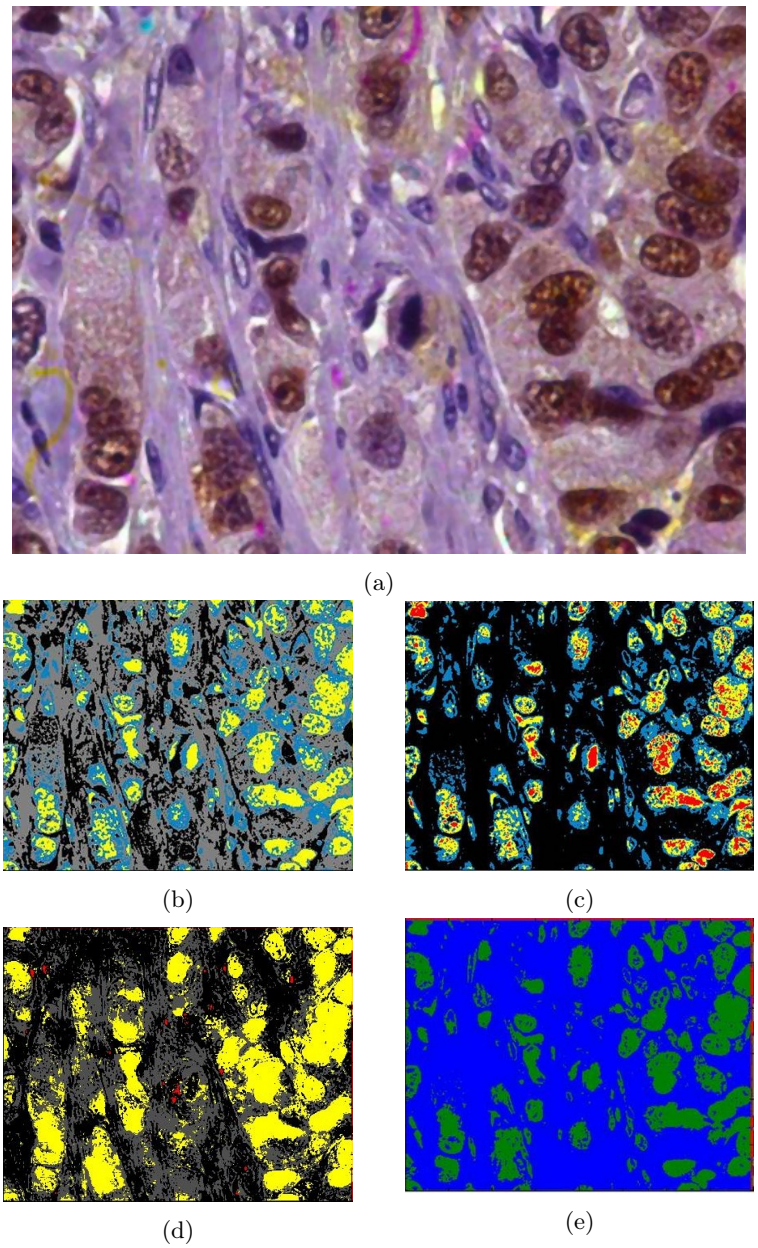


Figure 4.2: (a) Original picture, as observed from the microscope, (b) Product of K-Means with L1 or L2 as a metric,(c) Product of K-Means with SAM as a metric, (d) Product of K-Means with SCM as a metric ,(e)Product of Spectral Clustering

Segmentation and manually grouping. The first approach was to manually select which classes to group together by examining the RGB picture. Partitions extended on Red, Blue or White areas of the pictures were grouped together while partitions that seemed too damaged or noisy were ignored. Fi-

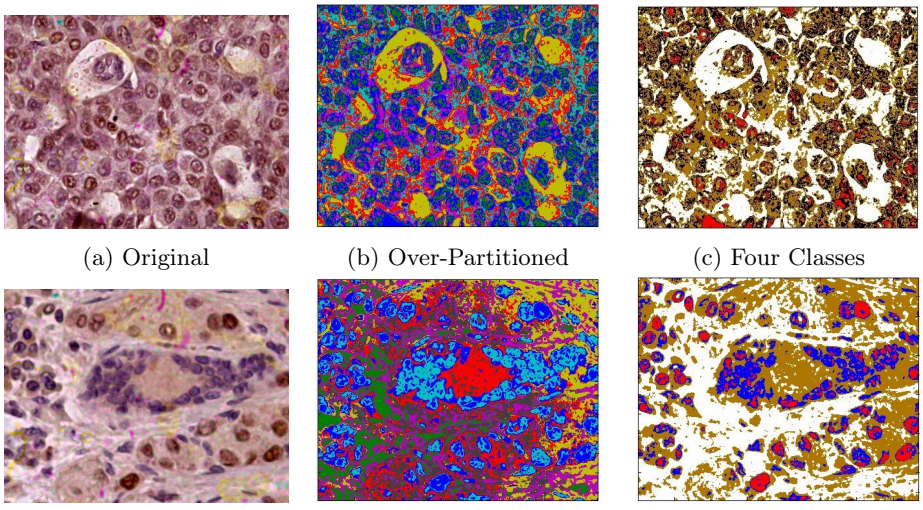
nally, partitions that had too few ink to be of importance or had mixed colors were added to the "Unspecified" category.

Excluding the category of photos with 60% pathogenesis for the rest of the pictures, the procedure of grouping their classes together was quit straightforward. At the 60%pathogenesis category, a large number of partitions and colors produced were a result of a lot of mixing between the two dominant substances (DAB and Hematoxylin). Because of that a second round of partitioning was initiated that segmented the pictures in twelve clusters (rather than eight) making things easier.

The manual procedure took place only at one out of all the pictures/HSC for each category. For the rest of them, a script was coded that assigned each cluster to one of the clusters of the first picture/HSC by comparing their centroids.

Segmentation and automatically grouping. Despite the great hopes for this technique, it didn't turn out to be as effective as originally thought. By introducing two thresholds to provide extra precision and create the Unspecified class, more variables are added up to the problem, making the solution even more biased and complex. After a lot of tests pursuing the best combination of thresholds, it appeared that thresholds may vary greatly depending on the category of the picture and the condition of the cells.

Because of that the thresholds have to be manually adjusted for every case, thus, simulating the manual selection of grouping used at the previous technique. Not only that, but finding the perfect combination that will produce the "unspecified" class and remove the noisy classes seems close to impossible.



(a) Original (b) Over-Partitioned (c) Four Classes

(d) Original (e) Over-Partitioned (f) Four Classes

Figure 4.3: Some examples on the partitioning finally chosen. Red represents DAB, blue represents Hematoxylin, golden/brown represnts "unspecified" and white represents the background. The black areas were abandoned as too noisy or reduntant.

3.2 Results and Conclusion

After extensive testing, the second technique proved to be the most accurate one. Some examples on this particular partitioning are displayed at Figure 4.3. The first column of figures display the original picture, while the second and third display the partitioned and manually grouped conditions respectively.

The final dataset was produced, by sampling the pictures/HSCs which were reserved for training. Each one provided an equal amount of samples, based on the size of the required dataset.

Class shapes. The resulted "class shapes" were produced by calculating the average value for each one of the band values in every class. Their patterns can be displayed at Figure 4.4. DAB is represented by the red line, Hematoxylin by the blue and Unspecified and Background by the golden and grey lines respectively.

It is obvious (and expected) that DAB which is colored red, shows high values at longer wavelengths, close to 700nms, (were the RED color resides) while Hematoxylin which is colored dark blue/purple shows high values also at shorter wavelengths and close to 400nms. Finally, the background classtends to have a uniform intensity on all the visible bands, something that produces its white color.

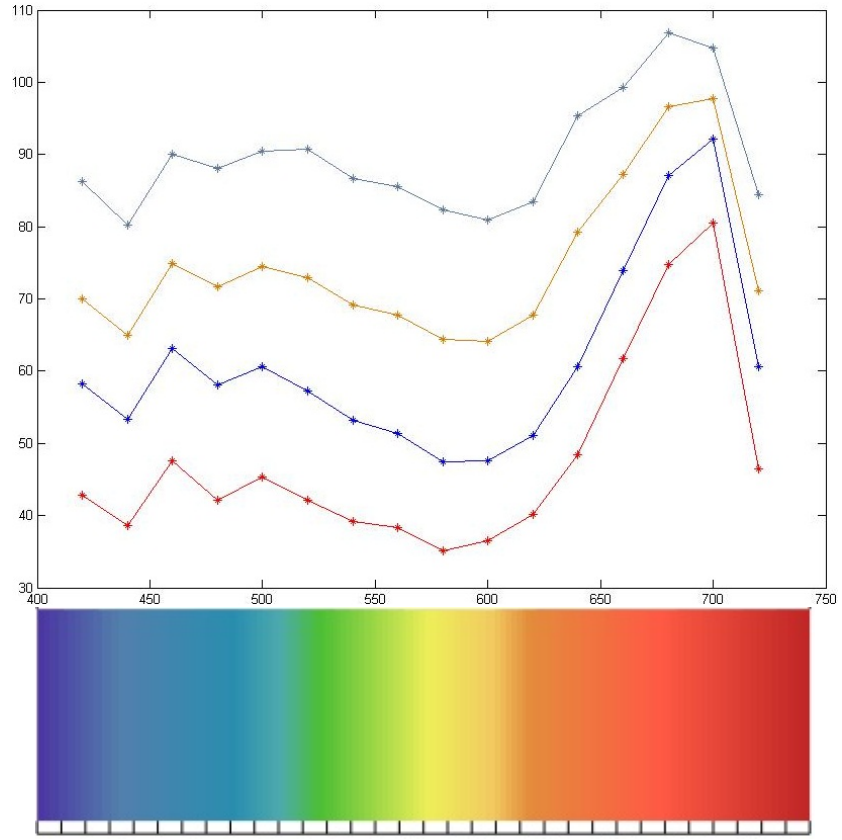


Figure 4.4: Class shapes over the electromagnetic spectrum.

Chapter 5

Training the Neural Network

After acquiring a proper Dataset and defining a Golden standard, it is time for the training. The total Dataset of 25 pictures were split in two halves, one of which would be used for training and the other one for testing. The reason for this split was that the classifier had to be tested on unknown data to guarantee its accuracy.

1 Training Process

1.1 Tools

For the training process many tools were tested, including Matlab's framework, the open source Library FANN, Alyuda NeuroIntelligence software and Peltarion Synapse.

Matlab's framework, provides a decent environment for training, but it's lack of sophisticated algorithms and automatic over-fitting/under-fitting prevention mechanisms, doesn't make it ideal for serial work.

Alyuda NeuroIntelligence, is one of the top ANN software of the market providing a really good interface and a wide area of tools for training neural networks. Alyuda NeuroIntelligence was used to give a quick overview of the best fitting form of the ANN for the data provided.

FANN library, provides a really fast and powerful tool for training neural networks with many different algorithms and automatic mechanisms for identifying the best activation function and training algorithm. FANN library was used exactly for these two purposes (finding the best activation function and training algorithm) and train the Network.

Finally, Peltarion Synapse was used to train the neural network as well, in addition to FANN.

1.2 Training

The training took place using 120000 samples across 12 images that were used for the training process and it was split into actual training (80% of the whole set) and 20% cross validation set. Out of all the different dataset sizes (from 1000 to 600000) which were tried out, 120000 seemed ideal since it provides not only enough samples for the training but also a small enough dataset for an

easy and quick training, thus making the testing of a lot of different networks, more comfortable.

The main idea was to try to fit the data in as few neurons and as few layers as possible, to provide faster passing through the network, during the classification process.

Thirteen networks with a single hidden layer were tested, with neuron number ranging from 2 to 34. For the initial phase of the training, those networks were trained for 1000 iterations, just to give an insight on the capabilities of each architecture. The results displayed at Figure 5.1 show that after 24 neurons the performance of the networks is stabilized at around 72%. Architectures with 16, 20 and 26 neurons were trained again, for 150.000 iterations this time.

It seems that the accuracy with 26 neurons can only go so far. After a long training session of 150.000 iteration, the 16-neuron-architecture, outperformed the 26-neuron one showing that 26 neurons might underfit the given data.

Chosen Neural Network Architecture : *16 - 16 - 4*

By exhaustively searching through the activation functions and training algorithms with FANN, it appears that **Symmetric Sigmoid Fuction** (tah(h)) and **RProp algorithm** provide the best solutions for this particular case.

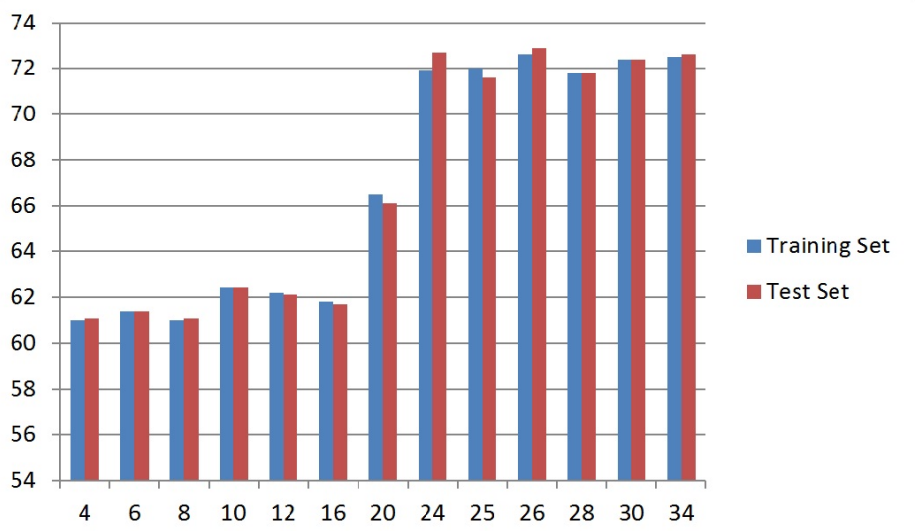


Figure 5.1: Fitness of the ANN over different architectures.

2 Results and Conclusion

After retraining the chosen architecture for 200.000 iterations, it was obvious that the performance couldn't go above $\tilde{7}6\%$. This is mainly due to the fact that a golden standard doesn't exist and because of the way class discrimination was performed (it doesn't give an absolute way of partitioning). Experienced doctors have to be used and a lot of opinions need to be advised for the grouping of the classes produced during the segmentation phase, in order to create an optimal

solution.

Furthermore, looking at the shape of the classes (Figure 4.4), it is obvious that a better feature selection is needed. Some of the 16 features-spectra do not have much information (like the last band - 720 nms) while others prove to be vital for the discrimination among the classes.

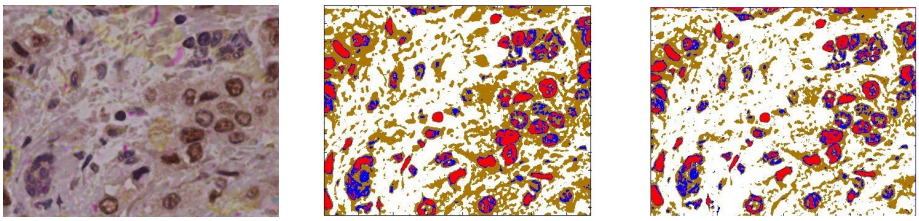
Future Work : Better feature selection and class discrimination during the Database creation phase, will help the classifier provide a lot better results.

	Target Class			
Predicted	*A*	*B*	*C*	*D*
A	**81.34**	10.16	2.13	2.61
B	10.37	**70.46**	6.28	17.14
C	3.47	6.97	**79.9**	10.07
D	4.84	12.41	11.68	**70.31**

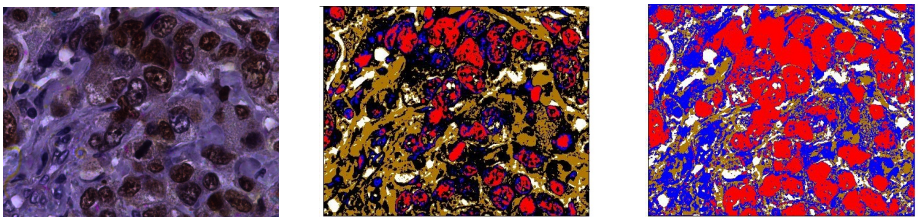
Table 5.1: Confusion matrix for the Global Created Set - Training + Unknown.

Observing the confusion table (Table 5.1), it is obvious that the main confusion exists between Class B (Hematoxylin) and the background, while only 10% is classified wrong between classes A and B. The confusion between Class B and D is probably caused, because some similar intensity blue regions were chosen to be grouped with the Class B while others with the Unspecified class, thus producing this confusion to the neural network. Again more experienced doctors could provide better grouping for those regions.

Despite the accuracy level, the visual result is satisfactory - Figure 5.2. Most of the confusion exists at the category of pictures with 60% of the cells under some condition, where the substances start to mix and their regions border blur.



(a) Original (80%) (b) Partitioned (c) Predicted



(d) Original (60%) (e) Partitioned (f) Predicted

Figure 5.2: Visual Accuracy on Images that weren't used during the training.

At other conditions the visual representation look most accurate and really

close to the way the images are partitioned. Another reason the classifier operates poorly at the 60% category is the fact that, at this category, large regions weren't included during the grouping phase. Those region inevitably have to be classified by the classifier, and the classifier gives its best guess.

Chapter 6

Exploiting the muscle of GP-GPUs

The last decade it is becoming increasing common to use a general purpose graphics processing (GP-GPU) unit or plain GPU as a modified form of stream processor. This concept turns the massive computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power, as opposed to being hard wired solely to do graphical operations.

In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU.

The two largest discrete (see "Dedicated graphics cards" above) GPU designers, ATI and Nvidia, are beginning to pursue this approach with an array of applications. Both Nvidia and ATI have teamed with Stanford University to create a GPU-based client for he Folding@home distributed computing project, for protein folding calculations. In certain circumstances the GPU calculates forty times faster than the conventional CPUs traditionally used by such applications.[1][2]

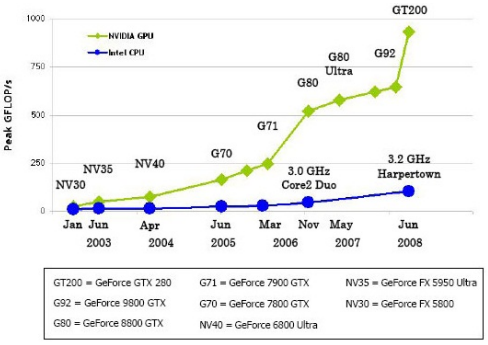


Figure 6.1: Increase in GPU performance (green) in addition to CPU performance (blue).

GPGPU can be used for many types of parallel task including image processing, ray tracing, computational fluid dynamics and weather modelling. They are generally suited to high-throughput type computations that exhibit data-parallelism to exploit the wide vector width SIMD architecture of the GPU.

Furthermore, GPU-based high performance computers are starting to play

a significant role in large-scale modelling. Three of the 5 most powerful supercomputers in the world take advantage of GPU acceleration. This includes the current leader as of October 2010, Tianhe-1A, which uses the Nvidia Tesla platform.

1 Introduction to GPUs

GPUs are usually perceived as multi-processor units with great computational strength. Although, this might be true, it is not entirely accurate. GPUs are mostly specialized units with massive numbers of integer and floating point ALUs and a very wide DRAM.

They are mostly designed to support brute computational needs and does not support more sophisticated services such as, shared cache, data pre-fetching etc similar to a CPUs (even multi-core ones).

(a) (b)

Figure 6.2: (a) Abstract view of a typical CPU architecture, (b) Abstract view of a typical GPU architecture, with 8 multiprocessors, each one having a lot of ALUs

CUDA Programming Model. For the purposes of programming GP-GPUs, the CUDA (Compute Unified Device Architecture) programming model and environment was developed by nVidia.

This model is based on the Single Instruction Multiple Data (SIMD) notion something that can be translated in *single instruction multiple threads* since CUDA provides a thread abstraction to assign work to SIMD units. CUDA code is written in C/C++ plus some necessary extensions.

Main purpose of the developer is the creation of *kernels*. A kernel is nothing more than a typical function which is executed on the GPU rather than on the CPU. Furthermore, following the SIMD model, this kernel is executed independently by every thread assigned on that task with the only difference being the data on which this kernel is applied to.

Other than CUDA, OpenCL a new programming model for GP-GPU has emerged, promising an even more Unified Device Architecture abstraction. OpenCL

code can be translated on any multi-processing unit code, with the use of smart compilers and a -slightly different- programming model.

Threads. GPUs can support tens of thousands of parallel threads,by providing hardware light-weight thread management (creating, deleting and managing threads on CPU is an extremely costly procedure).

The thread handling is also quite different than that of a CPU. Threads are grouped in blocks. Each thread has its own private local memory, organized in 32-bit registers, while each block has a shared memory that can be accessed by all the threads residing in that block, but not the ones outside of it. Blocks are then organized on a grid and have access to a global shared memory.

The developer has to define the Grid and Block parameters as well as the number of threads that will be active at the under-development application.

Architecture. The GPU architecture is composed by a set of processing units called *streaming multi-processors* (SM). Each SM contains a number of scalar processors (SP) or cores. The grouping of the SMs conforts the GPU and is called a device.

Each SM contains :

- A set of 32-bit-registers per SP.
- A read/write memory area, shared for all SPs called *shared memory*.

In addition to that, all SMs can access a global memory area called device memory.

Finally, the Thread Execution Manager is the unit responsible for coordinating the execution and organization of the requested threads within the SPs and SMs of the device.

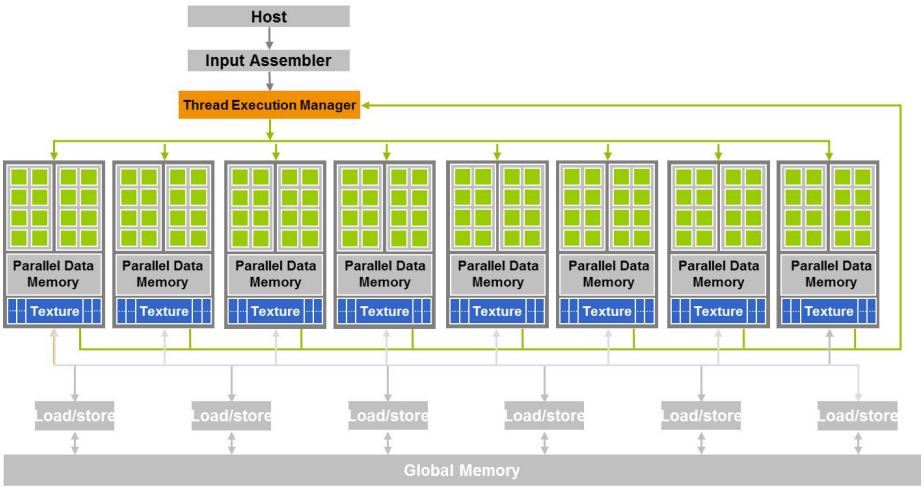


Figure 6.3: A typical CUDA enabled GP-GPU, with 8 streaming multiprocessors and 16 scalar processors within each SM.

Limitations and Disadvantages. Although, GPU usage provides great advantages and computational speed ups, it also suffers from a few limitations.

- **They are hard to program and debug.**
  Despite the fact that CUDA provides an interface and programming model really close to C/C++, writing code in CUDA means coding on lower level than usual.
- **No shared cache, no data prefetch.**
  Sure GPUs can provide an impressive horsepower, but what about brains? Unfortunately, smart solutions implemented on multi-core CPUs are not feasible on the GPU architecture.
- **No thread locking.**
  No thread locking is supported by GPUs. The developer has to make sure, that no thread locking or thread blocking will be required.
- **Host-Device-Host memory copy.**
  The fact that GPUs and CPUs don't have a shared memory, introduces the overhead of copying the data from one memory to the other.
- **Limited support of Floating-Point operation compared to CPUs.**
  By definition, ALU units in CPU can provide calculations of higher precision than those in the GPU.

2 Using GPUs for our problem

As mentioned before GPUs are extremely capable with vector operations. Because of their origin and reason of creation, their architecture, API and developed libraries support extremely well Linear Algebra operations.

More Data - Better performance. Matrices Product operation is one of the most common and typical uses for a GP-Graphical Processing Unit. Field tests have shown that certain CUDA libraries like cuBLAS can perform 7 or more times faster while the bigger the size of the given matrices the more GFLOPS (Giga-Floating Point Operations per Second) can be achieved.

The reader is reminded that the architecture of a neural network allows us to do the classification of the whole set of samples by performing product operations between the sample matrix and the weights that connect each layer to the next one.

$$Output = ActivationFunction(K * \Theta),$$

Output = A matrix that contains the output of all the neurons of a$layer L_l$. Row i contains the output of the layer for the i-th sample.

K = XxN matrix that contains X samples at its X rows, every sample is a 1xN vector.

Θ= NxM matrix that contains the weights of each connection between layers L_l andL_{l-1}.

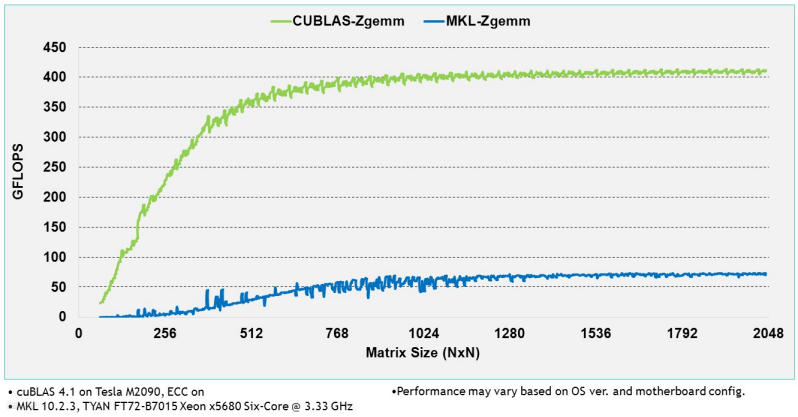


Figure 6.4: Scaling of GPU and CPU speed (in Giga- Floating Operations per Secons)for matrix products over the size of the matrices. The green curve represents the GPU performance for product operations and using the cuBLAS linear algebra library while the blue line represents the CPU performance for product operations using the MKL linear algebra library.

This is perfect for our situation since we can feed the whole Hyper Spectral cube to the process in order to get immediate results and force the GPU to perform at the peak of its capabilities.

Memory Copy overhead. Operations done in the GPU should be in such a way, that the time needed for the copy (from the CPU memory to the GPU memory) to be significantly smaller than the operation itself. In any different case the overhead counter balances the computational gain that can be gained by the GPU.

In our case, copying the data to the GPU has absolutely no difference, considering that, the end result is the visualization of the picture in the form of a color map, which will have to be rendered from the GPU of the computer anyway. Thus, copying the matrix, a few operations earlier doesnâĂŹt add any extra overhead to the whole processing time.

Portability. Finally a great advantage of GPUs over other systems that provide raw computational muscle is their portability. In contrast to other multi-processor units such as clusters, grids or super computers, a GPU can be "cheaply" placed as a component to an everyday machine or custom architecture.

3 Solution Specifications

Inputs - Outputs. The program created should be treated as a black box, which requires a 3-D matrix as input and outputs a 4 class thematic map on the screen. The program should also compute the percentage of pixels with infected cells over the total amount of pixel on the picture. At the current

implementation, the program takes as input .csv files that contain 3-D matrices stored in column-major format.

Compression. In order to reduce the memory copy time of our implementation a compression would be preferred. Taking into account the fact that the values are within the range [0-255], storage as a byte format is doable and ideal. This way we can reduce the memory load that has to be transfer by 75% of the initial cost.

Classification. Before the classification process, the data are normalized at the range of [-1, 1] and then processed through the neural network. Finally, the output Xx4 matrix (where X=number of pixels and 4 = number of classes) is given to a GUI which draws the color map on the screen.

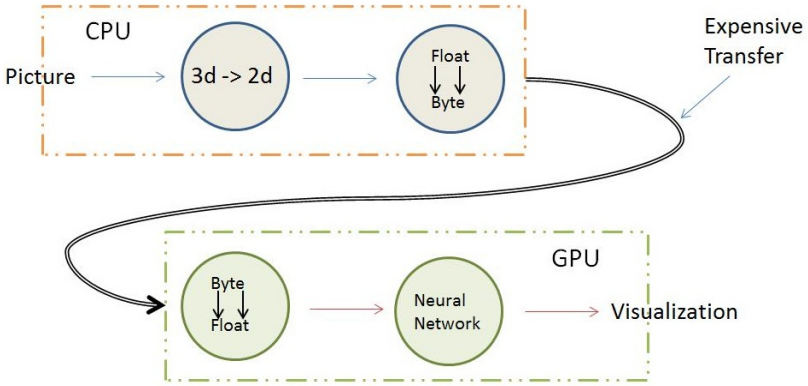


Figure 6.5: Complete overview of the proposed solution. After the acquisition of the picture/data cube, 3-D to 2-D transformation and compression is performed on the CPU. The cube is then moved to GPU were the processing is taking place after the decompression of the data.

4 Implementation and code analysis.

The implementation took part in two phases. The developement of the Back-End functionalities, that will be rensponsible for the classification processing and the Graphical User Interface development.

4.1 Back - end coding

For the Back- End implementation, 3 kernels were created, while cuBLAS library was used for the linear algebra calculations.

Initialization Phase. Before the actual operation of the application takes place, some initializations are necessery. The ANN weights are inputed using .csv input files and the appropriate memory space is allocated on the CPU and gpu. The weights are then transfered into the GPU and the application is ready

to start. For the purposes of the development, during the initialization phase, a HSC, is also inputed into the system, in the form of a .csv.

All the appropriate initialization actions are taking place automatically, following the start of the application.

Main Operation. The reader is reminded that the data are copied from the main memory to the GPU in the form of bytes. So, the first kernel receives a matrix of bytes and outputs a matrix of floats. The outputed floats are a mapping of all the values to the [-1,1] interval.

```
__global__
void byteToFloat
(float *out, unsigned char* in, int n)
{
  int i =  threadIdx.x + blockIdx.x * blockDim.x;

  for (; i < n; i += gridDim.x * blockDim.x)
    out[i] = __fdividef(in[i],255.0)*2 - 1;
}
```

Following that kernel, the data are ready to be inputed to the neural network. First step, through the ANN is the matrix multiplication of the input matrix with the weight matrix. For the multiplication, cublasSgemm() is used, which is available by the cuBLAS library.

Finally, the last step in order to get the the output of each layer is to pass the matrix-multiplication outcome through the activation function. The second kernel, takes care of this operation by accessing the whole body of the matrix and getting the activation values.

```
__global__
void sigmoid
(float *out, float *in, int n)
{
  int i = threadIdx.x + blockIdx.x*blockDim.x;

  for (; i < n; i += gridDim.x * blockDim.x)
    out[i] = __tan(in[1]) ;
}
```

Classification. For classification purposes, two goals have to be achieved:

- Classify each pixel with the appropriate class.
- Count the percentage of each class over the whole body of pixels.

By using the kernels mentioned above, it is possible to guide the data through the ANN, but those two alone, cannot produce a classification result. In order to get a classification result, a way has to be found, that will compare all 4 outputs of the final layer, recognize which one is the biggest and reward that node as the winner one, thus classifying the particular pixel with the class that this node represents.

In order to do so, a new kernel similar to the sigmoid one, was created. The difference in this one is that, apart from calculating the output of the activation function for each node, it also multiplies it by 10 and then does an integer division with 5. That guarantees that any node with an activation output $<$ 0.5 will get 0 as output, while any node with activation output $>$ 0.5 will get 1 as output.

```
__global__
void output_sigmoid
(float *uout, float *out, float *in, int n)
{
  int i = threadIdx.x + blockIdx.x*blockDim.x;
  float tmp;

  for (; i < n; i += gridDim.x * blockDim.x)
  {
    tmp = __tan(in[i]);
    uout[i] =(int) __fdividef(tmp*10,5);
    out[i] = tmp;
  }
}
```

Following this step, the output matrix becomes a series of 0/1. There are M rows (where M = number of pixels) and each row, contains 1 at the column of the class that this pixel is classified to.

Final Step. The reader is reminded that the matrix is stored in column-major format, which means that the first M elements of the matrix contain the first column, the next M elements contain the second column etc. That means that the first M elements represent the "projection" of the image only for the first class (where pixels that are classified with that class have 1 and the rest have 0), the next M elements represent the "projection" of the image only for the second class etc.

So, by taking the first M elements and summarizing their values, we can get the number of pixels that are classified with that class. Same happens with the following M elements etc. The summarizing is performed using the reducing function of cuBLAS, cublasSasum().

Reduction. Reduction is a popular divide and conquer technique, exercised on many problems in GPU application developing. For the case of summarizing the elements of a vector, each thread is performing an operation on only 2 elements of the vector producing a single element result. Doing this with enough

threads, it is possible to parallel "reduce" the vector to half its size. Doing that on loop, the vector is finally reduced to a single element, which is the result of the summarizing procedure.

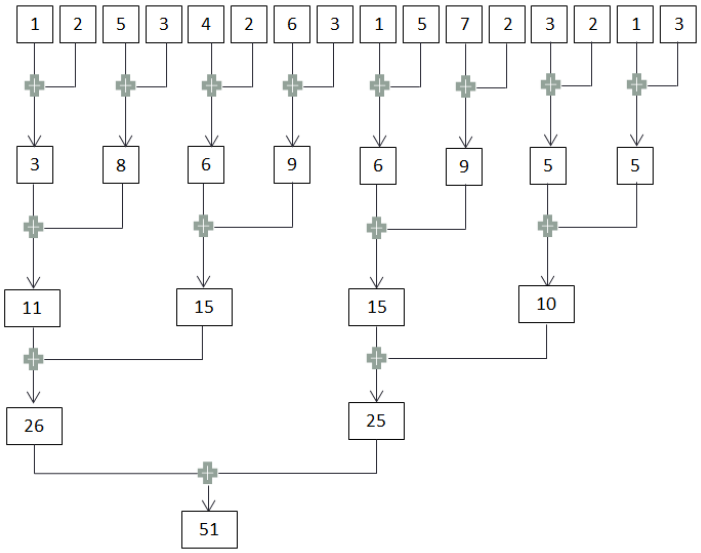


Figure 6.6: Example of a reduction procedure for summarizing the elements of a vector.

4.2 Graphical User Interface

The greatest disadvantage on video and image editing applications is the memory copy overhead applied. The typical approach requires the data to be send for processing to the GUI, then back to the CPU, where video and image display libraries will display them - by sending them back to the GPU. For that reason, GPUs are not favored for such applications.

Aiming for speed, rather than comfort, during this thesis, a GUI (Graphical User Interface) was developed, that displays the images directly from the GPU. It was developed with OpenGL, the open source language that was originally used to program GPUs (back at the day when a GPU was only responsible for Graphics rendering) and it is based on the simpleGL example that nVidia provides at its SDK for CUDA + OpenGL applications.

Main Concept. The GUI needs to satisfy three purposes :

- Show a colormap, that will display each class with different color.
- Display the percentage of each class.
- Display a "diagnostic percentage" and an opinion.

In order to provide a more clear display of the classified classes on the picture, the application generates not one but four colormaps, each one dedicated to one class. At each colormap the class displayed has a custom color and the

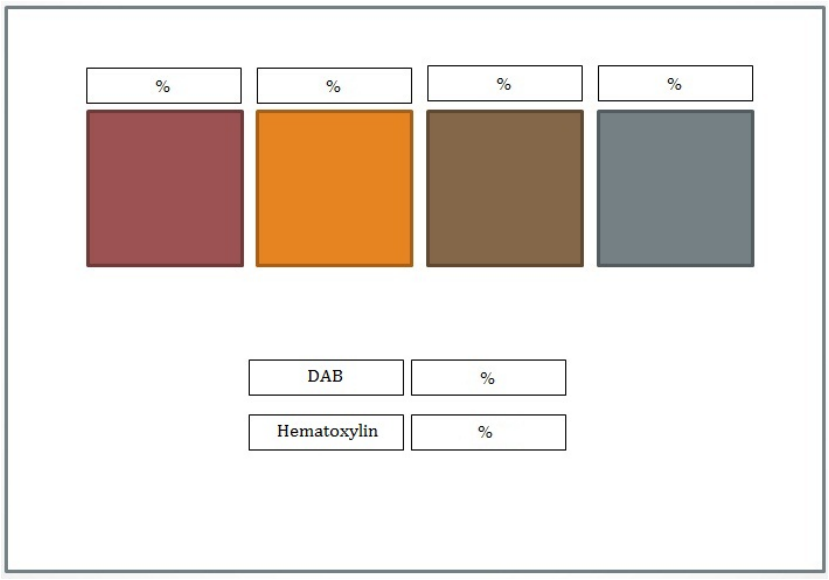


Figure 6.7: A draft of a simple GUI used for displaying the classification outcome.

rest of the picture is painted black. Also each pixel's color density changes according to the classification confidence of the neural network.

The reader is reminded that the ANN output at the final layer some values. The neuron with the highest value wins and this class is assigned to the input pixel. The outputted values can be perceived as a confidence factor of the ANN for the particular class, thus the neuron that has the highest confidence factor wins.

Above each one of the color maps, a percentage shows the ratio of this class over the total amount of pixels at the picture, while two more units below show the percentage of Class A (DAB) and Class B(Hematoxylin) over the total amount of pixels minus the pixels classified as background. Tthe intensity of (+1 to +3) can be diagnosed by the visual result on the colormaps.

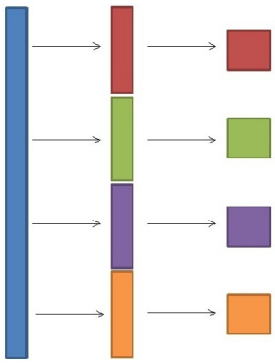


Figure 6.8: Easily extracting the four colormaps from the cube.

The result of the classification process, described at the previous section, is a huge column-major matrix, that contains the classification outcome for the whole picture. Following the same concept used during the reduction phase, the first M elements of the matrix (where M = the number of row = the number of pixels in the picture) contain the first column which represents the "projection" of the image only for the first class (where pixels that are classified with that class have 1 and the rest have 0), the next M elements represent the "projection" of the image

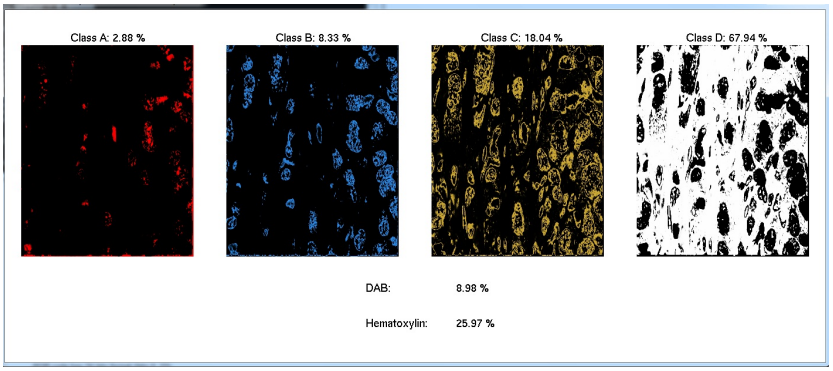


Figure 6.9: Example of the final application using a random image.

only for the second class etc. This way, it is quite straight forward to produce a colormap for each one of the classes, by simply extracting the elements that represent it.

Implementation. Because of the way with which GPUs and OpenGL render and then show the graphics, the whole matrix had to be copied to a 4 byte format that would contain the RGB values plus one extra value Alpha [3]. For this purpose a new kernel was created, where x,y represent the rows and columns, spanning from 0 to 1600 and 0 to 1200 respectively:

```
__global__
void fillbuf
(float* data, ulong3 size, uchar3 *buf, int imagenumber, uchar3 color )
{
  unsigned long x = blockIdx.x * blockDim.x + threadIdx.x;
  unsigned long y = blockIdx.y * blockDim.y + threadIdx.y;

  unsigned char r = color.x * data[(size.x*size.y)*imagenumber+ y*size.y+x];
  unsigned char g = color.y * data[(size.x*size.y)*imagenumber+ y*size.y+x];
  unsigned char b = color.z * data[(size.x*size.y)*imagenumber+ y*size.y+x];
  buf[ y * 1200 + x] = make_uchar3(r,g,b);
}
```

The matrix is copied to an openGL vertex buffer, which is mapped as a graphic resource using cudaGraphicsMapResources() function. After the matrix transfer openGL is rensponsible for unmapping the resources found in the vertex buffer, and display the pictures.

After the creation of the GUI, the custom made display() function is called iteratively. This function is responsible to call the back-end program, copy the newly created matrix to the vertex buffer and then display it. The interval with which the GUI calls the display() function, is called REFRESH_DELAY and can be defined manually. By default it is set at 100 ms.

5 Optimization and Results.

5.1 Kernel Tweaking

The final form of the kernels illustrated at the previous section, is a result of an optimization process. The first form of the kernels was the following:

```
__global__
void byteToFloat
(float *out, unsigned char* in, int n)
{
  int i =  threadIdx.x + blockIdx.x * blockDim.x;

  for (; i < n; i += gridDim.x * blockDim.x)
    out[i] = (float) (((float)in[i])/255.0)*2 - 1;
}

__global__
void sigmoid
(float *out, float *in, int n)
{
  int i = threadIdx.x + blockIdx.x*blockDim.x;

  for (; i < n; i += gridDim.x * blockDim.x)
    out[i] = tan(in[i]);
}
```

During the optimization process a lot of tests took place in order to find the best Grid and Block sizes. Those sizes will help the application "hide" the memory access delays, without forcing serialization. Furthermore, for best results the memory batch that will be processed by the block, should fit the SM as good as possible. In order to get those size, different sizes and Grid-Block combinations were tried. Each combination ran for 50 times on 2MP Hyper Spectral Images. Figure 6.10 shows the average time for each one of the tests, while the following tables summarize the results.

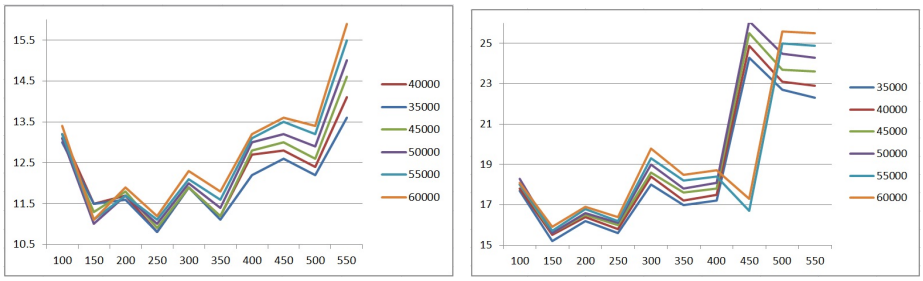
	Block Size									
Grid Size	*100*	*150*	*200*	*250*	*300*	*350*	*400*	*450*	*500*	*550*
35000	13	11.5	11.6	**10.8**	11.9	11.1	12.2	12.6	12.2	13.6
40000	13.2	11.5	11.7	10.8	11.9	11.2	12.7	12.8	12.4	14.1
45000	13.1	11.3	11.8	10.9	11.9	11.2	12.8	13	12.6	14.6
50000	13.1	11	11.7	11	12	11.4	13	13.2	12.9	15
55000	13.2	11.1	11.7	11.1	12.1	11.6	13.1	13.5	13.2	15.5
60000	13.4	11.1	11.9	11.2	12.3	11.8	13.2	13.6	13.4	15.9

Table 6.1: Average execution time for the byteToFloat kernel, based on different Grid and Block sizes.

	Block Size									
Grid Size	*100*	*150*	*200*	*250*	*300*	*350*	*400*	*450*	*500*	*550*
35000	17.7	**15.2**	16.2	15.6	18	17	17.2	24.3	22.7	22.3
40000	17.8	15.5	16.4	15.8	18.4	17.2	17.5	24.9	23.1	22.9
45000	18	15.6	16.5	16	18.6	17.6	17.8	25.5	23.7	23.6
50000	18.3	15.6	16.6	16.1	19	17.8	18.1	26.1	24.5	24.3
55000	18	15.7	16.8	16.2	19.3	18.2	18.4	16.7	25	24.9
60000	18.1	15.9	16.9	16.4	19.8	18.5	18.7	17.3	25.6	25.5

Table 6.2: Average execution time for the Sigmoid kernel, based on different Grid and Block sizes.

Looking at the tables, it is obvious that the best case for the byteToFloat kernel requires 250 threads per Block and 35 000 Blocks on the Grid while the Sigmoid kernel, requires 150 threads per Block and 35 000 Blocks on the Grid.



(a) (b)

Figure 6.10: (a) The performance of the first kernel *byteToFloat* (b) The performance of the second kernel *sigmoid*

Favoring Speed over Precision. At the original kernel, the floating point operations for division followed with the standard procedure. There is a way, with which, one can use functions optimized for speed rather than floating point accuracy.

In this case by using the optimized for speed function __fdividef(f1, f2) (instead of the standard division), and replacing expf() with __expf(), the average speed of the kernels dropped down to **3ms for byteToFloat** and **2ms for Sigmoid**. Tests showed that the precision required for this particular application is not affected, since the results are the same.

The final form of the kernels is illustrated below :

```
__global__
void byteToFloat
(float *out, unsigned char* in, int n)
{
  int i =  threadIdx.x + blockIdx.x * blockDim.x;
```

```
  for (; i < n; i += gridDim.x * blockDim.x)
    out[i] = __fdividef(in[i],255.0)*2 - 1;
}

__global__
void sigmoid
(float *out, float *in, int n)
{
  int i = threadIdx.x + blockIdx.x*blockDim.x;

  for (; i < n; i += gridDim.x * blockDim.x)
    out[i] = __tan(in[i]);
}
```

5.2 Results and Conclusion

Concluding this Chapter, the final time measurements for the whole application along with the maximum FPS (frames per second) that can be achieved, are summarized at Table 6.3. The tests took place on a GeForce GTX 500ti NVidia gpu and on an Intel QuadCore @ 2.6GHz CPU.

Different picture sizes were tested - 1 MegaPixel, 2 MegaPixel and a third option of a sub-sampled 2 MegaPixel picture (800x600). The sub sampled picture can be created, by taking one pixel for every two pixels of the original picture.

For the Summary, the Transfer value is not taken into account, since one can process a picture while a second one is being transfered, thus "hiding" the cost of transferring with a pipelining scheme.

HSC Size	*2 MP*	*1 MP*	*2 MP Sub-Sampled*
Transfer	22 ms	10 ms	6 ms
ByteToFloat	3.5 ms	1 ms	1 ms
Layer1	16 ms	6 ms	3 ms
Layer 2	10 ms	5 ms	3 ms
Reduction	9 ms	7 ms	7 ms
Visualization	2 ms	2 ms	2 ms
Summary	40.5 ms	21 ms	16 ms
Frames Per Second	24.7 fps	47.6 fps	62.5 fps

Table 6.3: Summarizing the execution time of the application.

Considering the high quality of display of a 2 Mega Pixel picture and the size of the final color map displayed, a sub-sampled picture will not affect the perception of the observer. Because of that, sub-sampled 2 Mega Pixel images are proposed as the golden mean between high resolution pictures and high quality fps rates.

Comparing to the CPU. A similar application was developed for the CPU in Matlab. Matlab depends upon highly optimized c++ and fortran libraries on linear algebra such as BLAS and Lapack and a powerfull display system, thus providing a worthy opponent.

HSC Size	*2 MP*	*1 MP*	*2 MP Sub-Sampled*
CPU	3426 ms	1386 ms	964 ms
GPU	40.5 ms	21 ms	16 ms
Speedup	84.6x	66x	60.3x

Table 6.4: CPU vs GPU.

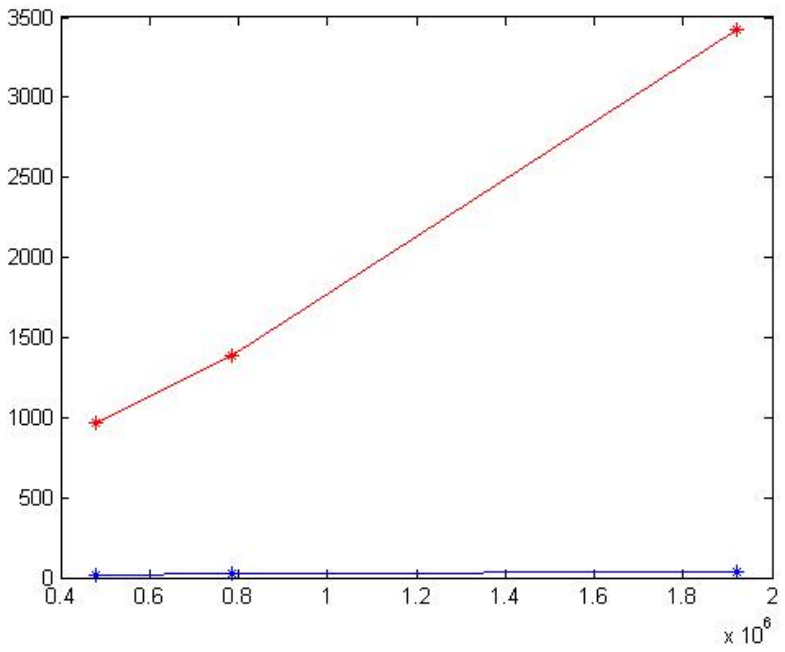


Figure 6.11: Graphical representation of the gigantic difference between the GPU (blue) and CPU (red).

Figure 6.11 displays the gigantic difference between the CPU and GPU per-

formance. The difference grows bigger as the amount of pixels to be classified increases. The Speedup starts at 60.3x for a sub-sampled 2MP picture and goes up to 84.6x for a 2MP picture.

Bibliography

[1] Darren Murph, Stanford University tailors Folding@home to GPUs.

[2] Mike Houston, *Folding@Home - GPGPU*.

[3] Porter, Thomas, Tom Duff , *Compositing Digital Images*, 1984, Computer Graphics 18 (3): 253âĂŞ259.

Chapter 7

Discussion

1 Conclusions

Throughout this thesis we have examined and developed an efficient way to process and segment hyper spectral pictures in order to create datasets, capable of training hyper spectral classifiers accurately. Spectral clustering and K-Means with different kernels were used in order to test the best way of clustering the pixels. We have concluded that clustering the picture in more classes than needed using K-Means and SAM as a measure of spectral similarity, can be really effective. Those extra classes were then grouped together via visual feedback to form the four goal classes.

We have concluded that Neural Networks can not only be very effective in classifying such images but also that combined with the computational powers of a GPU, they can provide real time performance, something that gives an edge to our classifier over others that have been introduced so far. We have trained the Neural Network with different architectures and activation functions and we have concluded that the best activation function is the Assymetric Sigmoid and the 16-26-4 architecture.

Finally, a simple GUI was developed in order to provide feedback to our work and a simple way of testing and using the implemented classifier.

The reader has to realize that despite our efforts to provide the best classes for training, due to the lack of pathologoanatomy experience, the classes provided might not be the best matches. This doesn't affect the results of this thesis though, which shows that the classifier considering a number of classes strictly set, can then be trained and learn to recognize them. The better the shape and nature of these classes the better the classification outcome.

2 Future work

The problem addressed and elaborated by this thesis is widely exploratory and interdisciplinary. For that reason, there is a number of expansions that

could be made upon this thesis on different parts of the implementation. Some proposals and suggestions are:

- **Better feature selection and class discrimination.**
  Due to the confliction of the classes with each other the performance of the classifier is stuck at lower levels than anticipated. Some better feature selection (by selecting the bands most vital for the classification) and better class discrimination(by using trained and experienced doctor) can help the classifier achieve higher levels of prediction.

- **Improve the functionality and interaction capabilities of the GUI.**
  The GUI developed, provides simple functionalities, and was build mainly to prove the capability of building a video streaming application by displaying the processing result, straight from the GPU. With some OpenGL experience, the GUI can be more interactive and support the functionality needs of more complex applications.

- **Directly connect the classifier with the hyper spectral camera in order to provide on line Hyper Spectral Classification.**
  The final connection of the camera with the classifier, provides the last step for the creation of a full functional on-line Hyper Spectral Classifier.

- **Train and test the classifier for in-vivo applications.**
  Although the tests for this thesis, have been performed on biopsies, the results can give quite an optimistic approach for in-vivo applications, thus using Hyper Spectral Imaging most useful tool : non-destructive analysis.