

ADAPTATION OF HIGH WEIGHTED INTO LOW WEIGHTED CNN FOR DEVELOPMENT OF REAL TIME APPLICATION

A PROJECT REPORT

Submitted by

PREETHI V

312316104129

ROSHNI B

312316104141

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



St. JOSEPH'S COLLEGE OF ENGINEERING, CHENNAI

ANNA UNIVERSITY: CHENNAI 600 025

MARCH 2020

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report “**ADAPTATION OF HIGH WEIGHT INTO LOW WEIGHT CNN DEVELOPMENT FOR REAL TIME APPLICATION**” is the bonafide work of **PREETHI V (312316104129)** and **ROSHNI B (312316104141)** who carried out the project work under my supervision.

SIGNATURE

Dr. A. CHANDRASEKAR M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Professor and Head

Department of

Computer Science and Engineering

St. Joseph’s College of Engineering

Chennai-600119.

SIGNATURE

Mrs. P.N. JEIPRATHA, M.E., {Ph.D}.,

SUPERVISOR

Assistant Professor

Department of

Computer Science and Engineering

St. Joseph’s College of Engineering

Chennai-600119.

Submitted for the Viva Voce held on: _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

CERTIFICATE OF EVALUATION

College Name : St. Joseph's College of Engineering

Branch & Semester : Computer Science and Engineering (VIII)

S.NO	NAME OF THE STUDENTS	TITLE OF THE PROJECT	NAME OF THE SUPERVISOR WITH DESIGNATION
1.	PREETHI V (312316104129)	ADAPTATION OF HIGH WEIGHT INTO LOW WEIGHT CNN DEVELOPMENT FOR REAL TIME APPLICATION	P.N. JEIPRATHA, M.E., {Ph. D}
2.	ROSHNI B (312316104141)		

The report of the project work submitted by the above students in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of Anna University were evaluated and confirmed to be reports of the work done by the above students.

(INTERNAL EXAMINER)

(EXTERNAL EXAMINER)

ACKNOWLEDGMENT

The contentment and elation that accompany the successful completion of any work would be incomplete without mentioning the people who made it possible.

Words are inadequate in offering our sincere thanks and most heartfelt sense of gratitude to **Dr. B. Babu Manoharan, M.A., M.B.A., Ph.D.** Chairman, St. Joseph's Group of Institutions for providing an opportunity to study in his esteemed Institution.

We also express our sincere thanks and gratitude to **Ms. B. Jessie Priya, M.Com,** the Managing Director and **Mr. B. Shashi Sekar, M.Sc.,** Director of St. Joseph's College of Engineering for their constant support throughout the course.

We are extremely happy to express our sincere gratitude to **Dr. Vaddi Seshagiri Rao M.E., M.B.A., Ph.D.,** Principal, St. Joseph's College of Engineering for his encouragement throughout the course.

We also express our sincere thanks and most heartfelt sense of gratitude to **Dr. A. Chandrasekar, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for his dedication, commendable support and encouragement for the completion of project work with perfection.

We convey our sincere thanks to our beloved supervisor **Mrs. P.N.Jeipratha, M.E., {Ph.D}.,** Assistant Professor Department of Computer Science and Engineering, who was always a constant source of inspiration and encouragement to us during the course of the project.

Last but not the least we thank our family members and friends who have been the greatest source of support for us.

ABSTRACT

VGG16 makes an improvement over AlexNet by replacing large-sized filters (11 and 5 in the first and second convolutional layers, respectively) with multiple 3X3 sized filters one after another. With a given receptive field (the effective area size of input image on which output depends), multiple stacked smaller sized kernel is better than the one with a larger sized kernel because multiple non-linear layers increase the depth of the network which enables it to learn more complex features, and that too at a lower cost. While VGG achieves a phenomenal accuracy on ImageNet dataset, its deployment on even the most modest sized GPUs is a problem because of huge computational requirements, both in terms of memory and time. It becomes inefficient due to large width of convolutional layers. For instance, a convolutional layer with 3X3 kernel size which takes 512 channels as input and outputs 512 channels, the order of calculations is $9 \times 512 \times 512$. Compared to VGG16 Net the novel model that will be developed is better for developing application because, novel design of architecture approximates a sparse CNN with a normal dense construction. Since only a small number of neurons are effective as mentioned earlier, the width/number of the convolutional filters of a particular kernel size is kept small. Also, it uses convolutions of different sizes to capture details at various scales (3X3, 1X1).

TABLE OF CONTENTS

CHAPTERS	TITLE	PAGE NO
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST SYMBOLS AND ABBREVIATION	ix
1	INTRODUCTION	1
	1.1 Machine Learning	1
	1.1.1 Deep Learning Process	1
	1.1.2 Convolution Neural Networks	3
	1.2 Problem Statement	4
	1.3 Objective	5
	1.4 Scope of the Project	5
2	LITERATURE REVIEW	6
3	SYSTEM ANALYSIS	13
	3.1 Existing System	13
	3.1.1 Disadvantages of Existing System	13
	3.2 Proposed System	13
	3.2.1 Advantages	14
	3.2.2 Applications	14
	3.3 Software Requirements	14
	3.3.1 Anaconda	14
4	SYSTEM DESIGN	17
	4.1 Architecture Diagram	17

	4.2 Working	17
	4.3 Phase of Implementation	18
	4.3.1 Data Collection	19
	4.3.2 Data Augmentation	20
	4.3.3 Data Pre-processing	21
	4.3.4 Training with the Algorithm	22
	4.3.5 Training with Novel Architecture	25
	4.3.6 Data Optimization	26
	4.3.7 Loss Minimization	28
5	SYSTEM IMPLEMENTATION	30
	5.1 List of Modules	30
	5.2 Module Description	30
	5.3 Datasets	31
6	RESULTS AND DISCUSSION	32
	6.1 Result	32
7	CONCLUSION AND FUTURE WORK	41
	7.1 Conclusion	41
	7.2 Future Work	41
	REFERENCES	42

LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO.
1.1	Deep Learning Process	2
1.2	CNN	3
1.3	Types of CNN	4
4.1	System Architecture	17
4.2	Image Cropping method	22
4.3	VGG-16 Input Output	23
4.4	Architecture of VGG16	24
4.5	Novel architecture	26
4.6	Gradient Descent	27
6.1	Starting the Anaconda Navigator	32
6.2	Moving Inside the coding folder	33
6.3	VGG16 Coding	33
6.4	Executing in the Command Prompt	34
6.5	Code Execution	35
6.6	VGG16 Output	36
6.7	VGG16 Model Size	37
6.8	Prediction for VGG16 Architecture	37
6.9	Light Weight Architecture Output	38
6.10	Light Weight Architecture Model Size	39
6.11	Prediction for Light Weight Architecture	40

LIST OF SYMBOLS AND ABBREVIATIONS

CNN	Convolutional Neural Network
RNNS	Recurrent Neural Network
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
ReLU	Rectified Linear Unit
VGG	Visual Geometry Group
SARSA	State-Action-Reward-State-Action
ADASs	Advanced Driver-Assistance Systems
MFFD	Modular Feature Fusion Detector
PRISM	Privacy -aware Interest Sharing and Matching
CMs	Cable Modems
MAC	Medium Access Control
SDN	Software Defined Networking
R-PHY	Remote-PHY
DOCSIS	Data Over Cable Service Interface Specification
DDPG	Deep Deterministic Policy Gradient
CIN	Converged Interconnect Network
CSA	Carry-Save Adder
CCSA	Configurable Carry-Save Adder
RIT	Robust Information Theoretic
OCM	Organizational Change Management
FC	Fully Connected

SVM	Support Vector Machine
CADx	Computer Aided Diagnosis
PYPI	Python Package Index
CLI	Command Line Interface
CT	Computer Tomography
GUI	Graphical User Interface
LRN	Local Response Normalisation
RGB	Red Green Blue
CVPR	Computer Vision and Pattern Recognition
ADAM	Adaptive Moment Estimation
UAC	User Account Control

CHAPTER 1

INTRODUCTION

1.1 MACHINE LEARNING

In the field of Artificial Intelligence (AI), Deep Learning is a method that falls in the wider family of Machine Learning algorithms. For learning, both Supervised and Unsupervised forms may be used. In deep learning, a computerized model will perform a specific set of classification or pattern analysis tasks based on previously learned data. For that, a model must first be trained with a set of labeled data. Deep learning basically used to classify images, text or sounds. Deep learning models work without human intervention and they are equivalent, and sometimes even, superior to humans. Deep learning models are realized mostly through deep neural networks. It has been applied to several arenas that include Bioinformatics, Image Processing, Industrial Automation, Natural Language Processing, Text and Sound Recognition and many more. Artificial Intelligence consists a set of techniques that can be applied to develop computerized models to make machines to act like a human expert. Machine learning, a part of Artificial Intelligence paradigm, consists of techniques that enable machines to learn by themselves based on experience. It has a scope for machines to improvise automatically over time. Deep learning is a part of machine learning and contains algorithms that make machine “to learn” based on experience to perform the assigned task, mostly text, sound and image recognition. For that, it mainly uses an architecture of artificial neural network with a higher number of hidden layers, called deep neural networks.

1.1.1 DEEP LEARNING PROCESS

A deep neural network provides state-of-the-art accuracy in many tasks, from object detection to speech recognition. They can learn automatically, without predefined knowledge explicitly coded by the programmers.

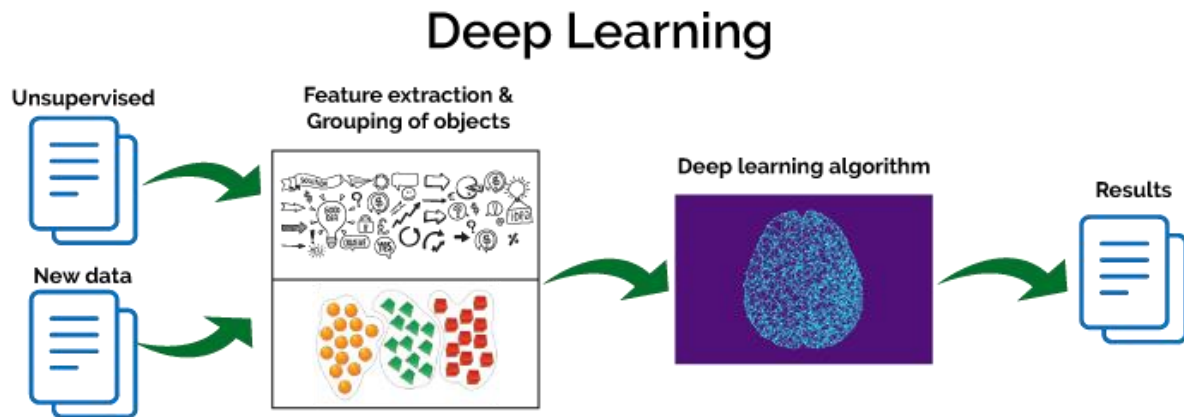


Figure 1.1 Deep Learning Process

To grasp the idea of deep learning, imagine a family, with an infant and parents. The toddler points objects with his little finger and always says the word 'cat.' As its parents are concerned about its education, they keep telling it 'Yes, that is a cat' or 'No, that is not a cat.' The infant persists in pointing at objects but becomes more accurate with 'cats.' The little kid, deep down, does not know why he can say it is a cat or not. He has just learned how to identify complex features coming up with a cat by looking at the pet overall and continue to focus on details such as the tails or the nose before to make up his mind.

A neural network works quite the same. Each layer represents a deeper level of knowledge, i.e., the hierarchy of knowledge. A neural network with four layers will learn more complex features than that with two layers.

The learning occurs in two phases.

- The first phase consists of applying a nonlinear transformation of the input and create a statistical model as output.
- The second phase aims at improving the model with a mathematical method known as derivative.

The neural network repeats these two phases hundreds to thousands of time until it has reached a tolerable level of accuracy. The repetition of this two-phase procedure is called an iteration.

1.1.2 CONVOLUTIONAL NEURAL NETWORKS (CNN)

CNN is a multi-layered neural network with a unique architecture designed to extract increasingly complex features of the data at each layer to determine the output. CNN's are well suited for perceptual tasks.

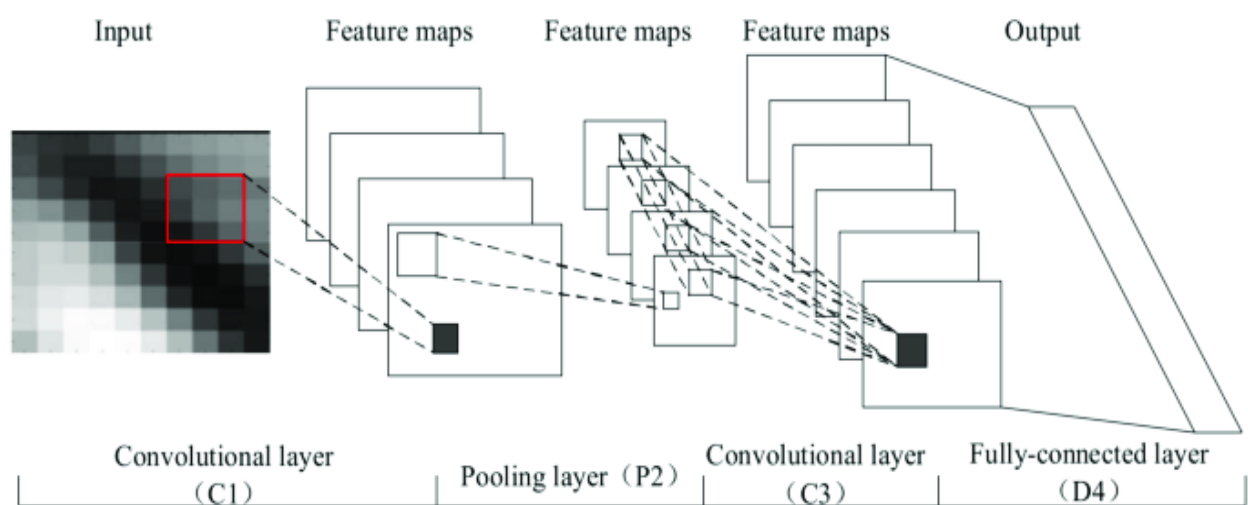


Figure 1.2 CNN

CNN is mostly used when there is an unstructured data set (e.g., images) and the practitioners need to extract information from it

For instance, if the task is to predict an image caption:

- The CNN receives an image of let's say a cat, this image, in computer term, is a collection of the pixel. Generally, one layer for the greyscale picture and three layers for a colour picture.
- During the feature learning (i.e., hidden layers), the network will identify unique features, for instance, the tail of the cat, the ear, etc.

- When the network thoroughly learned how to recognize a picture, it can provide a probability for each image it knows. The label with the highest probability will become the prediction of the network.

A Convolutional Neural Network (CNN, or ConvNet) are a special kind of multi-layer neural networks, designed to recognize visual patterns directly from pixel images with minimal pre-processing. The **ImageNet** project is a large visual database designed for use in visual object recognition software research. The ImageNet project runs an annual software contest, the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**, where software programs compete to correctly classify and detect objects and scenes. Here I will talk about CNN architectures of ILSVRC top competitors.

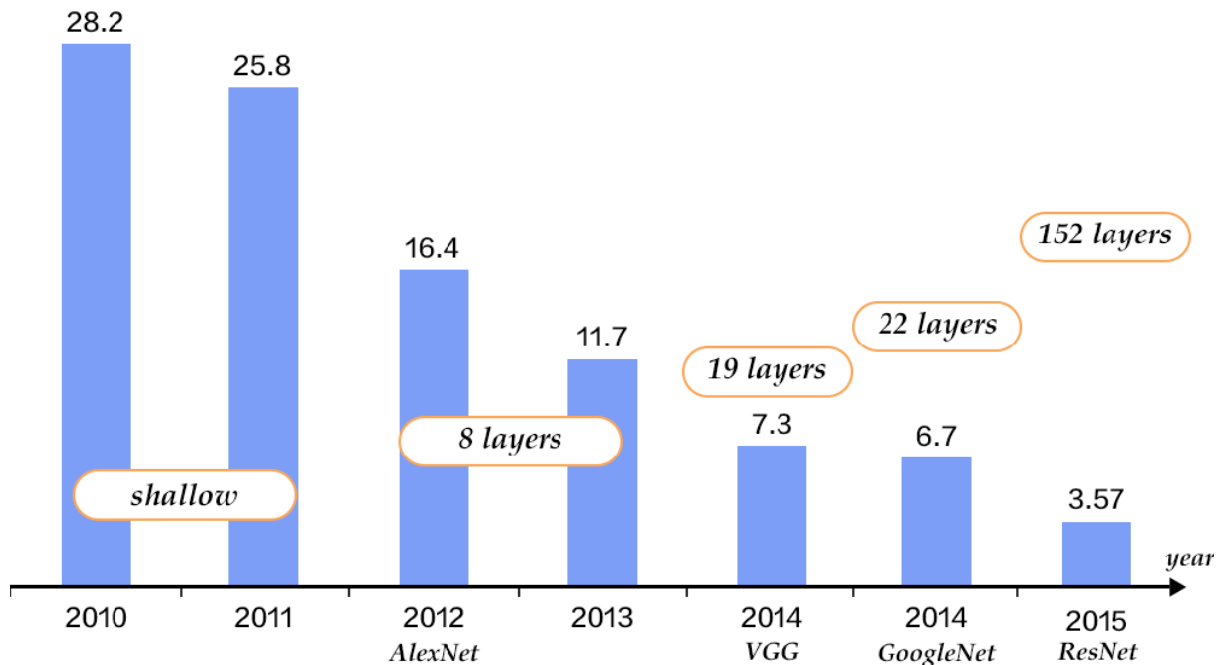


Figure 1.3 Types of CNN

1.2 PROBLEM STATEMENT

VGG achieves a phenomenal accuracy on ImageNet dataset, its deployment on even the most modest sized GPUs is a problem because of huge computational requirements, both in terms of memory and time. It becomes inefficient due to large width of convolutional layers. For instance, a convolutional layer with 3X3 kernel size which takes 512 channels as input and outputs 512 channels, the order of calculations is $9 \times 512 \times 512$. Compared to VGG16 Net the novel model that will be developed is better

for developing application because, novel design of architecture approximates a sparse CNN with a normal dense construction. Since only a small number of neurons are effective as mentioned earlier, the width/number of the convolutional filters of a particular kernel size is kept small. Also, it uses convolutions of different sizes to capture details at various scales (3X3, 1X1).

1.3 OBJECTIVE

- To prove that the novel architecture design is better than the VGG16 architecture.
- To design an application with novel architecture proving the functionality over VGG16.
- To decrease the model size which can be easily used in real time applications.

1.4 SCOPE OF THE PROJECT

The proposed architecture can be used to build applications that should be run on small and modest sized GPUs. This architecture mainly aims on reducing the load of the application reducing the computational needs in terms of both memory and time and thus can be employed at situations where such applications are in demand.

CHAPTER - 2

LITERATURE REVIEW

2.1 INTRODUCTION:

The following is a survey done for Lightweight Network. The most popular of the existing techniques has been discussed as follows.

2.2 LITERATURE SURVEY:

2.2.1 Yazhou Liu, Sen Cao, Pongsak Lasang, Member, IEEE, and Shengmei Shen, Member, IEEE, “Modular Lightweight Network for Road Object Detection Using a Feature Fusion Approach”, IEEE transactions on systems, man, and cybernetics: systems.

A modular lightweight network model for road objects detection, such as car, pedestrian, and cyclist, especially when they are far away from the camera and their sizes are small. Great advances have been made for the deep networks, but small objects detection is still a challenging task. In order to solve this problem, a majority of existing methods utilize complicated network or bigger image size, which generally leads to higher computation cost. The proposed network model is referred to as a modular feature fusion detector (MFFD), using a fast and efficient network architecture for detecting small objects. The contribution lies in the following aspects: 1) two base modules have been designed for efficient computation: a) Front module reduces the information loss from raw input images and b) Tinier module decreases the model size and computation cost, while ensuring the detection accuracy; 2) by stacking the base modules, we design a context features fusion framework for multiscale object detection; and 3) the proposed method is efficient in terms of model size and computation cost, which is applicable for resource-limited devices, such as embedded systems for advanced driver-assistance systems (ADASs). Comparisons with the state-of-the-art on the challenging KITTI dataset reveal the superiority of the proposed method. Especially, 100 ft/s can be achieved on the embedded GPUs such as Jetson TX2.

2.2.2 FIZZA ABBAS, (Student Member, IEEE), UBAIDULLAH RAJPUT, (Student Member, IEEE), AND HEEKUCK OH, (Member, IEEE), “PRISM: Privacy-Aware Interest Sharing and Matching in Mobile Social Networks”, (Student Member, IEEE.

In a profile matchmaking application of mobile social networks, users need to reveal their interests to each other in order to find the common interests. A malicious user may harm a user by knowing his personal information. Therefore, mutual interests need to be found in a privacy preserving manner. In this paper, we propose an efficient privacy protection and interests sharing protocol referred to as Privacy -aware Interest Sharing and Matching (PRISM). PRISM enables users to discover mutual interests without revealing their interests. Unlike existing approaches, PRISM does not require revealing the interests to a trusted server. Moreover, the protocol considers attacking scenarios that have not been addressed previously and provides an efficient solution. The inherent mechanism reveals any cheating attempt by a malicious user. PRISM also proposes the procedure to eliminate Sybil attacks. We analyse the security of PRISM against both passive and active attacks. Through implementation, we also present a detailed analysis of the performance of PRISM and compare it with existing approaches. The results show the effectiveness of PRISM without any significant performance degradation.

2.2.3 Qi Wang, Senior Member, IEEE, Jia Wan, Xuelong Li, Fellow, IEEE, “Comb- Robust Hierarchical Deep Learning for Vehicular Management”, IEEE Transactions on Vehicular Technology.

Congestion detection is an important aspect of Vehicular Management. However, most of the existing algorithms are insufficient for real applications. Traditional features are not discriminative which results in rather poor performance under complex scenarios. The deep features can better represent high-level information, but the training of deep networks for regression is difficult. To promote congestion detection, a robust hierarchical deep learning is proposed for the task. In this method, a deep network is designed for hierarchical semantic feature extraction. Different from traditional deep regression networks which usually directly utilize mean squared error as loss function, a robust metric learning is employed to effectively train the network. Based on this,

multiple networks are combined together to further improve the generalization ability. Extensive experiments are conducted and the proposed model is confirmed to be effective.

2.2.4 Tucker James Marion and Marc H. Meyer, “Organizing to Achieve Modular Architecture Across Different Products”, IEEE transactions on engineering management.

The relationship between organization and the development of architecture-enabled modularity within and across product lines. We examine organization structure, formal and informal team communication, and budgetary resources within a larger framework of the matrix organization found necessary for achieving a modular product line architecture spanning different products. Results were controlled for firm size and product complexity. Outcomes were assessed at the individual product level for meeting pre-development performance objectives. We found that the practice of assigning separate, dedicated architecture teams was not necessary to achieve modular architecture across different products or for achieving positive research and development (R&D) outcomes. Nor was a dedicated budget for defining modular product line architecture required for this type of R&D success. Rather, it was found that intensive collaboration and communication between individuals assigned to different product development teams, who nonetheless were working toward a common modular architecture, was essential. These findings, when combined with longitudinal qualitative insights, showed that including a product or systems architect on specific product development teams, where such architects meet and comprise a community of practice within the organization, were strongly associated with modular architecture development and positive R&D outcomes.

2.2.5 Thomas Kohler, Frank Dürr, and Kurt Rothermel, “ZeroSDN: A Highly Flexible and Modular Architecture for Full-Range Distribution of Event-Based Network Control”, IEEE transactions on network and service management, vol. 15, no. 4, December 2018.

Recent years have seen an evolution of software defined networking (SDN) control plane architectures, starting from simple monolithic controllers, over modular

monolithic controllers, to distributed controllers. We observe, however, that today’s distributed controllers still exhibit inflexibility with respect to the distribution of control logic. Therefore, we propose a novel architecture of a distributed SDN controller, providing maximum flexibility with respect to distribution and improved manageability. Our architecture splits control logic into lightweight control modules, called controllets, based on a micro-kernel approach, reducing common controllet functionality to a bare minimum and factoring out all higher-level functionality. Lightweight controllets also allow for pushing control logic onto switches and enabling local processing of data plane events to minimize control latency and communication overhead while leveraging SDN’s global view to maximize control decision quality. Controllets are interconnected through a message bus, supporting the publish/subscribe communication paradigm with specific extensions for content-based message filtering. Publish/subscribe allows for complete decoupling of controllets to further facilitate control plane distribution. Furthermore, we identify crucial requirements for practical on-switch deployments, where we employ lightweight virtualization techniques to ensure a safe control plane operation. We evaluate both the scalability and performance properties of our architecture, including its deployment on a white-box networking hardware switch.

2.2.6 Ziyad Alharbi, Akhilesh S. Thyagaturu, Martin Reisslein, Fellow, IEEE, Hesham ElBakoury, and Ruobin Zheng, “Performance Comparison of R-PHY and R-MACPHY Modular Cable Access Network Architectures”, IEEE transactions on broadcasting.

Emerging modular cable network architectures distribute some cable headend functions to remote nodes that are located close to the broadcast cable links reaching the cable modems (CMs) in the subscriber homes and businesses. In the remote-PHY (R-PHY) architecture, an R-PHY device conducts the physical layer processing for the analog cable transmissions, while the headend runs the data over cable service interface specification (DOCSIS) medium access control (MAC) for the upstream transmissions of the distributed CMs over the shared cable link. In contrast, in the remote MACPHY (R-MACPHY) architecture, an R-MACPHY device (RMD) conducts both the physical and MAC layer processing. In this paper, we conduct a comprehensive performance

comparison of the R-PHY and R-MACPHY architectures. We first develop analytical delay models for the polling-based MAC with gated bandwidth allocation of Poisson traffic in the R-PHY and R-MACPHY architectures. We then conduct extensive simulations to assess the accuracy of the analytical model and to evaluate the delay-throughput performance of the R-PHY and R-MACPHY architectures for a wide range of deployment and operating scenarios. Our evaluations include long converged interconnect network (CIN) distances between remote nodes and headend, bursty self-similar traffic, and double-phase polling to mask long CIN propagation distances. We find that for long CIN distances above 100 miles, the R-MACPHY architecture achieves significantly shorter mean upstream packet delays than the R-PHY architecture, especially for bursty traffic. Our extensive comparative R-PHY and R-MACPHY evaluation can serve as a basis for the planning of modular broadcast cable-based access networks.

2.2.7 Shiann-Rong Kuang, Member, IEEE, Kun-Yi Wu, and Ren-Yao Lu, “Low-Cost High-Performance VLSI Architecture for Montgomery Modular Multiplication”, IEEE transactions on very large-scale integration (vlsi) systems.

A simple and efficient Montgomery multiplication algorithm such that the low-cost and high-performance Montgomery modular multiplier can be implemented accordingly. The proposed multiplier receives and outputs the data with binary representation and uses only one-level carry-save adder (CSA) to avoid the carry propagation at each addition operation. This CSA is also used to perform operand precomputation and format conversion from the carry save format to the binary representation, leading to a low hardware cost and short critical path delay at the expense of extra clock cycles for completing one modular multiplication. To overcome the weakness, a configurable CSA (CCSA), which could be one full-adder or two serial half-adders, is proposed to reduce the extra clock cycles for operand precomputation and format conversion by half. In addition, a mechanism that can detect and skip the unnecessary carry-save addition operations in the one-level CCSA architecture while maintaining the short critical path delay is developed. As a result, the extra clock cycles for operand precomputation and format conversion can be hidden and high throughput can be obtained. Experimental

results show that the proposed Montgomery modular multiplier can achieve higher performance and significant area–time product improvement when compared to previous designs.

2.2.8 Yue Deng, Feng Bao, Xuesong Deng, Ruiping Wang, Member, IEEE, Youyong Kong, and Qionghai Dai, Senior Member, IEEE, “Deep and Structured Robust Information Theoretic Learning for Image Analysis”, IEEE Transactions on Image Processing.

A robust information theoretic (RIT) model to reduce the uncertainties, i.e. missing and noisy labels, in general discriminative data representation tasks. The fundamental pursuit of our model is to simultaneously learn a transformation function and a discriminative classifier that maximize the mutual information of data and their labels in the latent space. In this general paradigm, we respectively discuss three types of the RIT implementations with linear subspace embedding, deep transformation and structured sparse learning. In practice, the RIT and deep RIT are exploited to solve the image categorization task whose performances will be verified on various benchmark datasets. The structured sparse RIT is further applied to a medical image analysis task for brain MRI segmentation that allows group-level feature selections on the brain tissues.

2.2.9 Shiann-Rong Kuang, Member, IEEE, Chih-Yuan Liang, and Chun-Chi Chen, “An Efficient Radix-4 Scalable Architecture for Montgomery Modular Multiplication”, IEEE Transactions on Circuits and Systems II: Express Briefs.

To achieve low latency, existing radix-4 scalable architectures for word-based Montgomery modular multiplication usually suffer from high design and hardware complexities. This brief presents a simple compression scheme and circuit to remove the data dependency in the accumulation process and accomplish one-cycle latency without quotient pipeline. The complex computation and encoding of quotient digits are thus avoided, leading to up to 10.6% and 17.7% reductions in area and power than previous work while maintaining very high performance. Consequently, the proposed radix-4 scalable architecture appears to be very suited for low-complexity and low-power cryptographic applications.

2.2.10 Wen-Ching Lin, Jheng-Hao Ye, and Ming-Der Shieh, Member, IEEE, “Scalable Montgomery Modular Multiplication Architecture with Low Latency and Low Memory Bandwidth Requirement”, IEEE transactions on computers.

A novel scheduling scheme to alleviate the number of memory access in the developed scalable architecture. Analytical results show that the memory bandwidth requirement of the proposed scalable architecture is almost $1/(w-1)$ times that of conventional scalable architectures, where w denotes word size. The proposed one also retains a latency of exactly one cycle between the operations of the same words in two consecutive iterations of the Montgomery modular multiplication algorithm when employing enough processing elements. Compared to the design in the related work, experimental results demonstrate that the proposed one achieves an almost 54% reduction in power consumption with no degradation in throughput. The reduced number of memory access not only leads to lower power consumption, but also facilitates the design of scalable architectures for any precision of operands.

CHAPTER - 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

We use VGG-16 to train an image feature extractor for ultrahigh-resolution OCM images. In particular, we append a new fully-connected layer (FC3) to the second fully-connected (FC2) layer. The FC3 layer outputs a 5-D feature vector representing the input image. Meanwhile, we build a text feature extractor to process patient information in medical records. Note that only age and HPV test results are available for this study. Second, we take advantage of a 7-D feature vector that concatenates the image and text features obtained to train an SVM-based classifier. Third, for a given OCM image, the SVM-based classifier outputs a predicted label of the five fine-grained classes. Besides, we evaluate the CADx method over two general classes, “low risk” (normal, ectropion, and LSIL) and “high risk” (HSIL and cancer), by inferring the probabilities of the corresponding fine-grained classes. The CADx method makes a decision of classification for each 3-D OCM image volume according to the mechanism of voting based on the majority rule.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM:

- Accuracy is less.
- No real time implementation.

3.2. PROPOSED SYSTEM:

In this project, we prove that the novel architecture designed by us is better than VGG16 for real time application. We can understand the problem that led to the creation of novel architecture, we will more easily understand the inner workings of it. Novel architecture approximates a sparse CNN with a normal dense construction. Since only a small number of neurons are effective as mentioned earlier, the width/number of the convolutional filters of a particular kernel size is kept small. Also, it uses convolutions of different sizes to capture details at varied scales (3X3, 1X1). Different loss minimization such as cross entropy and optimization techniques such as Adam will be

applied to increase the accuracy of the model. Reduces the size of the model generated to prove it applicable in real time. We will be developing an application over prediction of cervical cancer presence with three different types.

3.2.1 ADVANTAGES:

- Novel model trains faster than VGG.
- Size of a pre-trained novel architecture is comparatively smaller than VGG. A VGG model can have > 100 MBs, whereas novel architecture designed by us has a size of only 16 MB.

3.2.2 APPLICATIONS:

- It is used for creating light-weight applications.
- Novel architecture model also represents an improvement over normal mobile development in two other areas: the developer experience and cross-platform development potential.
- Predicts three different types of cervical cancer.

3.3 SOFTWARE REQUIREMENTS

Anaconda (Python Distribution Platform)

3.3.1 ANACONDA (Python Distribution Platform)

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Open Source

Anaconda Individual Edition is the world's most popular Python distribution platform with over 20 million users worldwide. You can trust in our long-term commitment to supporting the Anaconda open-source ecosystem, the platform of choice for Python data science.

Conda-Install Packages

Search our cloud-based repository to find and install over 7,500 data science and machine learning packages. With the conda-install command, you can start using thousands of open-source Conda, R, Python and many other packages.

Manage Environments

Individual Edition is an open source, flexible solution that provides the utilities to build, distribute, install, update, and manage software in a cross-platform manner. Conda makes it easy to manage multiple data environments that can be maintained and run separately without interference from each other.

Build machine learning models

Build and train machine learning models using the best Python packages built by the open-source community, including scikit-learn, TensorFlow, and PyTorch.

The open-source community at your fingertips

With Anaconda Individual Edition, the open-source world is your oyster. From robotics to data visualization, you can access the open-source software you need for projects in any field.

User interface makes learning easier

Anaconda Navigator is a desktop GUI that comes with Anaconda Individual Edition. It makes it easy to launch applications and manage packages and environments without using command-line commands.

Expedite your data science journey with easy access to training materials, documentation, and community resources including [Anaconda.org](https://anaconda.org).

Anaconda for the enterprise

With Anaconda's Team and Enterprise Editions, our stack can handle the most advanced enterprise data science requirements.

Overview

Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

The big difference between anaconda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason anaconda exists.

When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example, Google Tensorflow, can find that it stops working having used pip to install a different package that requires a different version of the dependent numpy library than the one used by Tensorflow. In some cases, the package may appear to work but produce different results in detail.

In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g. the user may wish to have Tensorflow version 2.0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the CONDA INSTALL command. Anaconda Inc compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed.

Custom packages can be made using the CONDA BUILD command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.

CHAPTER: 4

SYSTEM DESIGN

DETAILED DESIGN OF THE PROJECT:

This chapter describes the overall design. It also describes each module that is to be implemented.

4.1 SYSTEM ARCHITECTURE

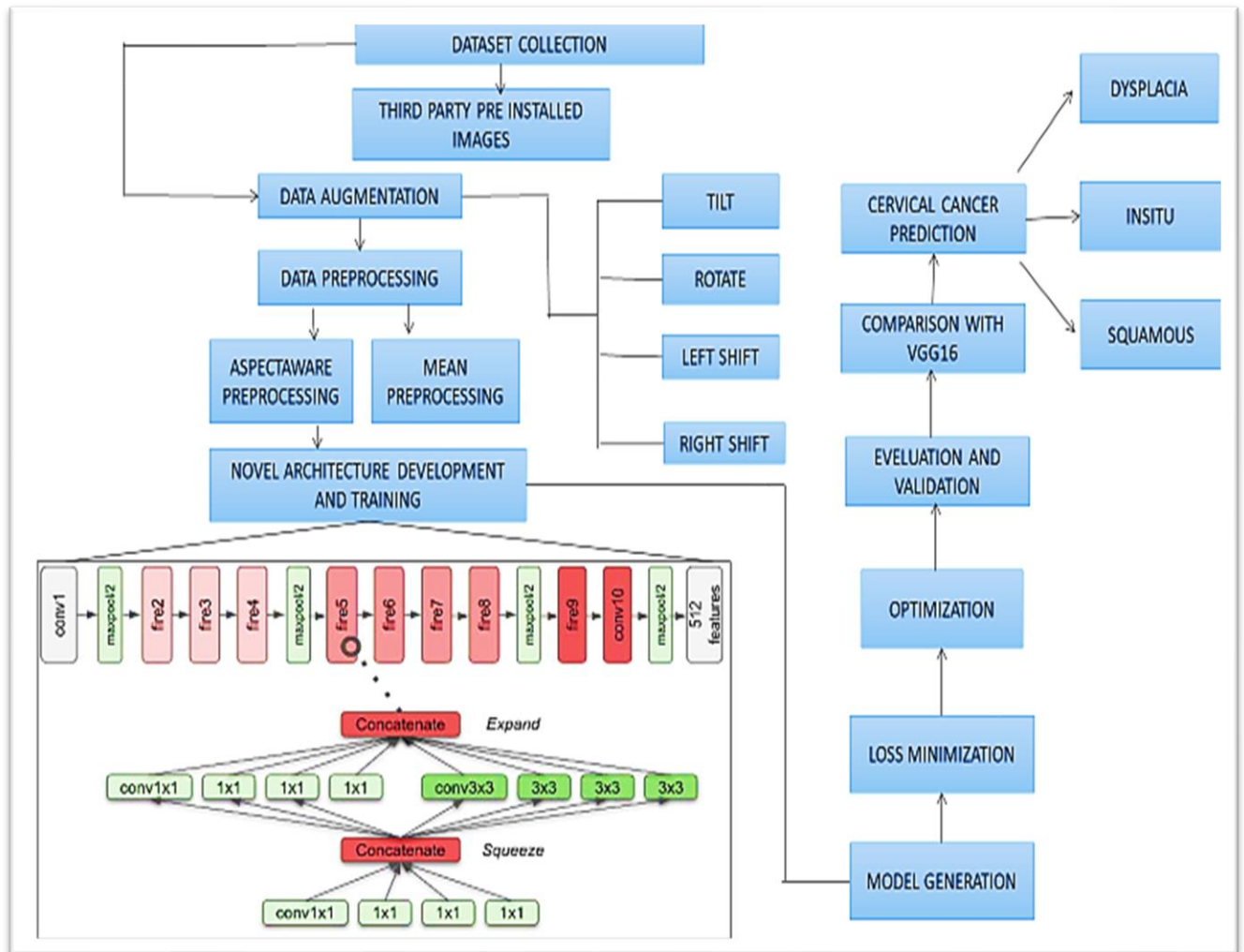


Figure: 4.1: System Architecture

4.2 WORKING

In this project, we are going to design a novel architecture for building a light-weight application and determine presence of three different types of cervical cancer. By this

project, we can determine the presence of cervical cancer in the CT images or x-rays. So, the first step will be dataset collection where we will be collecting dataset such as CT images or x-rays which are used by the laboratories. After that, we will be splitting the dataset into different categories, that is, we will be splitting the dataset into training and testing datasets. In the training dataset, we will be using the dataset for training the module whereas the testing dataset is used to evaluate the model when it is completely ready. So, the training dataset undergoes a process called dataset augmentation, where the dataset is multiplied into many datasets. Then it will undergo a process called pre-processing, where images of different sizes are converted into images with a uniform size. We train that datasets by extracting the features using novel architecture as well as VGG16 architecture. It undergoes a process called optimization which will optimize the model and loss minimization which will reduce the noises generated during training. Finally, it will undergo a process called model serialization, where the model will be evaluated after generating it using the testing dataset and predict the presence of cervical cancer. We determine the difference between both the architectures by its model size. The model size of VGG16 is more which is not optimal for real time usage while the novel architecture's model size appears to be much smaller which is suitable for the same. Thus, this method provides an effective and a cheap real time methodology to determine the presence of cervical cancer than the methodologies used nowadays.

4.3 PHASES OF IMPLEMENTATION

- Data collection
- Data augmentation
- Data pre-processing
- Training with the VGG16 Architecture
- Training with a Novel Architecture
- Data optimization
- Loss minimization

4.3.1 Data Collection

Deep Learning has become the go-to method for solving many challenging real-world problems. It's definitely by far the best performing method for computer vision tasks. The image above showcases the power of deep learning for computer vision. With enough training, a deep network can segment and identify the “key points” of every person in the image.

These deep learning machines that have been working so well need fuel — lots of fuel; that fuel is data. The more labelled data we have, the better our model performs. The idea of more data leading to better performance has even been explored at a large-scale by Google with a dataset of 300 Million images!

Third-party

Since data has become such a valuable commodity in the deep learning era, many start-ups have started to offer their own image annotation services: they'll gather and label the data for you! All you'll have to do is give them a description of what kind of data and annotations you'll need.

Mighty AI is one that has been doing self-driving car image annotation and has become pretty big in the space; they were at CVPR 2018 too. Playment AI is less specialized than Mighty AI, offering image annotation for any domain. They also offer a couple more tools such as video and landmark annotations.

Pre-trained Networks

Many of us already know of the idea of transfer learning: start with a network pre-trained on a large dataset, and then fine tune on our own. Well we can use the same idea for collecting our new dataset. The datasets that these pre-trained networks were trained on are huge; just check out the Open Images dataset with over 15 Million images labelled with bounding boxes from 600 categories! A network trained on this dataset is already going to be pretty darn good at detecting objects. So, we can use it to draw some bounding boxes around the objects in our images. This cuts our work in half since all we then have to do is classify the objects in the boxes! Plus, with 600 categories, some of objects you desire to detect and classify may already be picked up at high accuracy

with this pre-trained network. The TensorFlow Object Detection API actually already has a network pre-trained on Open Images if you'd like to try it out.

4.3.2 DATA AUGMENTATION

The performance of deep learning neural networks often improves with the amount of data available.

Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image.

Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more.

The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set image that are likely to be seen by the model. For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right. A vertical flip of the photo of a cat does not make sense and would probably not be appropriate given that the model is very unlikely to see a photo of an upside-down cat.

As such, it is clear that the choice of the specific data augmentation techniques used for a training dataset must be chosen carefully and within the context of the training dataset and knowledge of the problem domain. In addition, it can be used to experiment with data augmentation methods in isolation and to see if they result in a measurable improvement in model performance, perhaps with a small prototype dataset, model, and training run.

Modern deep learning algorithms, such as the convolutional neural network, or CNN, can learn features that are invariant to their location in the image. Nevertheless, augmentation can further aid in this transform invariant approach to learning and can

aid the model in learning features that are also invariant to transforms such as left-to-right to top-to-bottom ordering, light levels in photographs, and more.

Image data augmentation is typically only applied to the training dataset, and not to the validation or test dataset. This is different from data preparation such as image resizing and pixel scaling; they must be performed consistently across all datasets that interact with the model.

4.3.3 DATA PRE-PROCESSING

Deep learning has truly come into the mainstream in the past few years. Deep learning uses neural nets with a lot of hidden layers (dozens in today's state of the art) and requires large amounts of training data. These models have been particularly effective in gaining insight and approaching human-level accuracy in perceptual tasks like vision, speech, language processing. The theory and mathematical foundations were laid several decades ago. Primarily two phenomena have contributed to the rise of machine learning a) Availability of huge data-sets/training examples in multiple domains and b) Advances in raw computer power and the rise of efficient parallel hardware.

Building an effective neural network model requires careful consideration of the network architecture as well as the input data format. The most common image data input parameters are the number of images, image height, image width, number of channels, and the number of levels per pixel. Typically, we have 3 channels of data corresponding to the colors Red, Green, Blue (RGB) Pixel levels are usually [0,255]. For this exercise let's choose the following values

- number of images = 100
- image width, image height = 100
- 3 channels, pixel levels in the range [0–255]

Let's look at one popular data-set. Labelled Faces in the Wild is a database of facial images, originally designed for studying the problem of face recognition. The data-set contains more than 13,000 images of faces collected from the web, and each face has been labelled with the name of the person pictured. Data-set images need to be converted into the described format. After downloading the image data, notice that the

images are arranged in separate sub-folders, by name of the person. We'll need to get all the photos into a common directory for this exercise. Let's take the first 100 images and copy them into a working directory. The data contains faces of people 'in the wild', taken with different light settings and rotation. They appear to have been centred in this data set, though this need not be the case. There are a number of pre-processing steps we might wish to carry out before using this in any Deep Learning project.

Uniform aspect ratio: One of the first steps is to ensure that the images have the same size and aspect ratio. Most of the neural network models assume a square shape input image, which means that each image needs to be checked if it is a square or not, and cropped appropriately. Cropping can be done to select a square part of the image, as shown. While cropping, we usually care about the part in the centre.

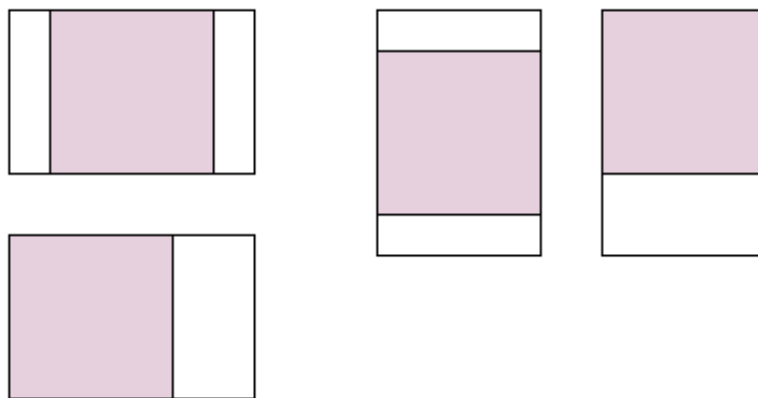


Figure 4.2 Image Cropping method

4.3.4 TRAINING WITH THE VGG16 ARCHITECTURE

In order to train the model, we will be using VGG16 algorithm. VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition ". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous models submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second

convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPUs.

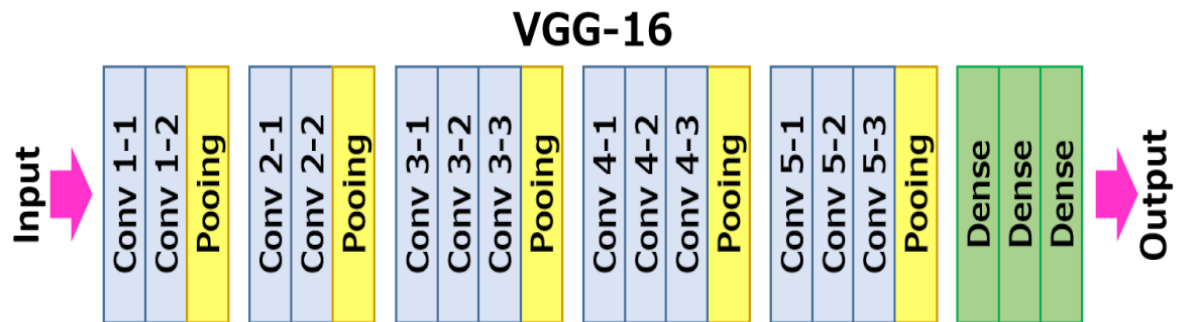


Figure 4.3: VGG-16 Input Output

ImageNet is a dataset of over 15 million labelled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labelled by human labellers using Amazon's Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images. ImageNet consists of variable-resolution images. Therefore, the images have been down-sampled to a fixed resolution of 256×256 . Given a rectangular image, the image is resized and cropped out the central 256×256 patch from the resulting image.

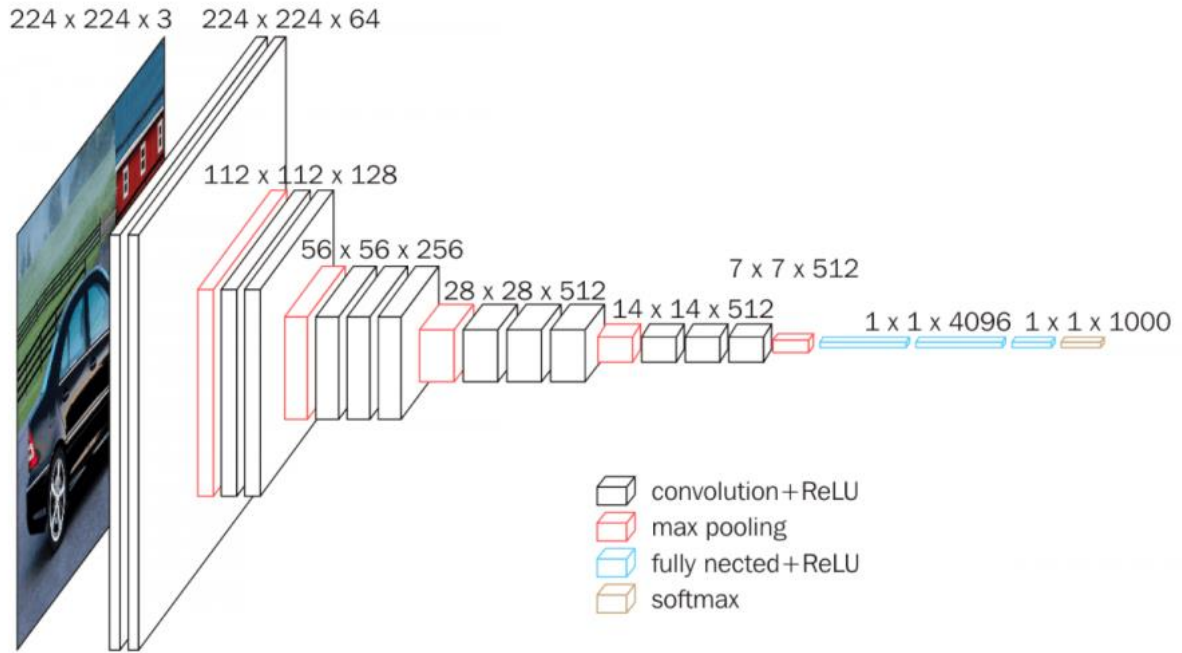


Figure 4.4: Architecture of VGG16

The input to conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filter, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2-pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to an increased memory consumption and computation time.

4.3.5 TRAINING WITH A NOVEL ARCHITECTURE

In order to overcome the disadvantage of the existing VGG16 architecture, we will be developing a novel architecture.

The main ideas of the novel architecture are:

1. Using 1x1(point-wise) filters to replace 3x3 filters, as the former only 1/9 of computation.
2. Using 1x1 filters as a bottleneck layer to reduce depth to reduce computation of the following 3x3 filters.
3. Down sample late to keep a big feature map.

The building brick of Novel architecture is called fire module, which contains two layers: a squeeze layer and an expand layer. A Novel architecture stacks a bunch of fire modules and a few pooling layers. The squeeze layer and expand layer keep the same feature map size, while the former reduces the depth to a smaller number, the later increase it. The squeezing (bottleneck layer) and expansion behaviour is common in neural architectures. Another common pattern is increasing depth while reducing feature map size to get high level abstract.

As shown in the below chart, the squeeze module only contains 1x1 filters, which means it works like a fully-connected layer working on feature points in the same position. In other words, it doesn't have the ability of spatial abstract. As its name says, one of its benefits is to reduce the depth of feature maps. Reducing depth means the following 3x3 filters in the expand layer has fewer computations to do. It boosts the speed as a 3x3 filter needs 9 times computation as a 1x1 filter. By intuition, too much squeezing limits information flow; too few 3x3 filters limits space resolution.

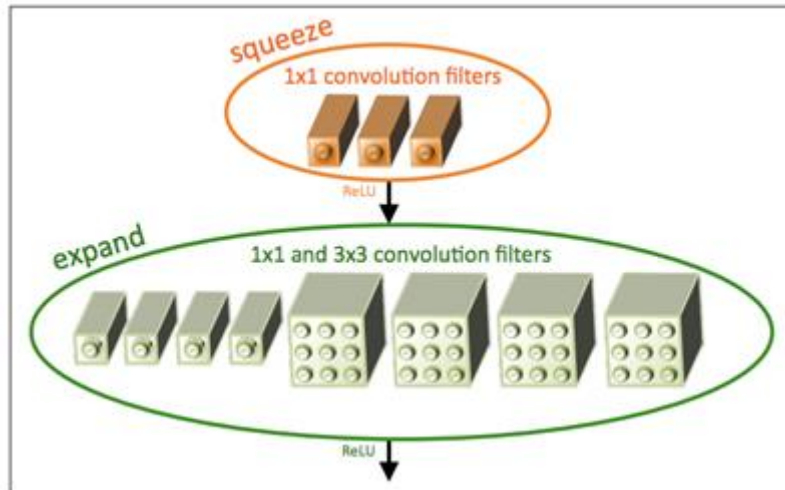


Figure 4.5: Novel architecture

4.3.6 DATA OPTIMIZATION

Optimization algorithms help us to minimize (or maximize) an Objective function (another name for Error function) $E(x)$ which is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values (Y) from the set of predictors (X) used in the model. For example — we call the Weights (W) and the Bias (b) values of the neural network as its internal learnable parameters which are used in computing the output values and are learned and updated in the direction of optimal solution i.e. minimizing the Loss by the network's training process and also play a major role in the training process of the Neural Network Model.

The internal parameters of a model play a very important role in efficiently and effectively training a model and producing accurate results. This is why we use various optimization strategies and algorithms to update and calculate the appropriate and optimum values of the model's parameters which influence its learning process and output.

Gradient Descent is the most important technique and the foundation of how we train and optimize Intelligent Systems. Find the Minima, control the variance and then update the Model's parameters and finally lead us to Convergence. $\theta = \theta - \eta \cdot \nabla J(\theta)$ — is

the formula of the parameter updates, where ‘ η ’ is the learning rate, ‘ $\nabla J(\theta)$ ’ is the Gradient of Loss function- $J(\theta)$ w.r.t parameters- ‘ θ ’.

It is the most popular Optimization algorithms used in optimizing a Neural Network. Now gradient descent is majorly used to do Weights updates in a Neural Network Model, i.e. update and tune the Model’s parameters in a direction so that we can minimize the Loss function. Now we all know a Neural Network trains via a famous technique called Backpropagation, in which we first propagate forward calculating the dot product of Inputs signals and their corresponding Weights and then apply a activation function to those sum of products, which transforms the input signal to an output signal and also is important to model complex Non-linear functions and introduces Non-linearities to the Model which enables the Model to learn almost any arbitrary functional mappings.

After this we propagate backwards in the Network carrying Error terms and updating Weights values using Gradient Descent, in which we calculate the gradient of Error(E) function with respect to the Weights (W) or the parameters, and update the parameters (here Weights) in the opposite direction of the Gradient of the Loss function w.r.t to the Model’s parameters.

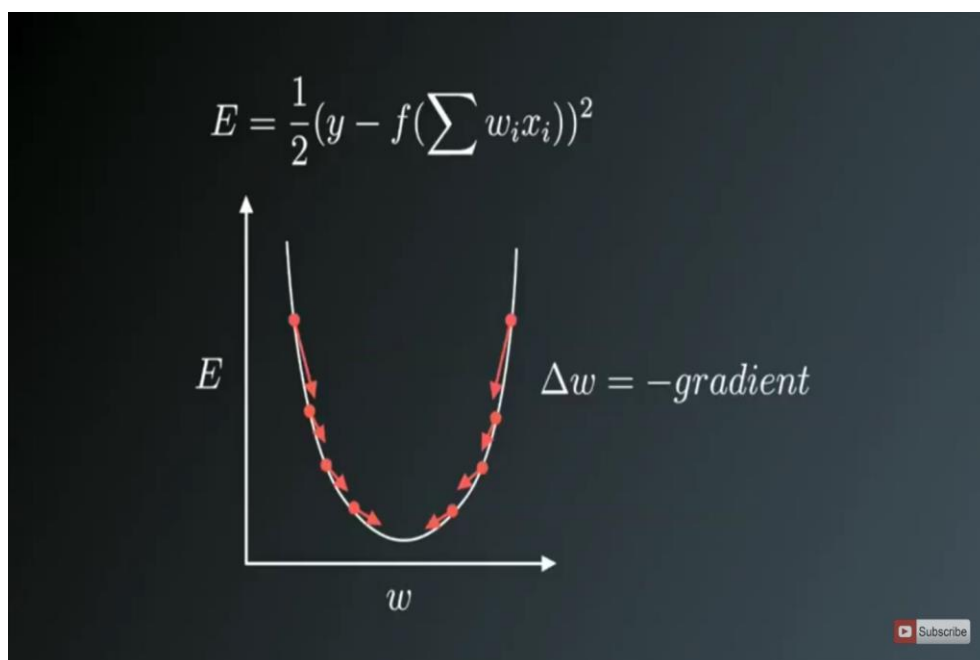


Figure 4.6: Gradient Descent

Weight updates in the opposite direction of the Gradient. The image on above shows the process of Weight updates in the opposite direction of the Gradient Vector of Error w.r.t to the Weights of the Network. The U-Shaped curve is the Gradient (slope). As one can notice if the Weight(W) values are too small or too large then we have large Errors, so want to update and optimize the weights such that it is neither too small nor too large, so we descent downwards opposite to the Gradients until we find a local minimum.

4.3.7 LOSS MINIMIZATION

Training a model simply means learning (determining) good values for all the weights and the bias from labelled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called loss minimization and cross entropy is used for loss minimization.

Cross-Entropy

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So, predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

The CE Loss is defined as:

$$CE = - \sum_i^C t_i \log(s_i)$$

Where t_i and s_i are the ground truth and the CNN score for each class in C . As usually an activation function (Sigmoid / SoftMax) is applied to the scores before the CE Loss computation, we write $f(s_i)$ to refer to the activations.

In a binary classification problem, where $C'=2$, the Cross-Entropy Loss can be defined also as

$$CE = - \sum_{i=1}^{C'=2} t_i \log(s_i) = -t_1 \log(s_1) - (1 - t_1) \log(1 - s_1)$$

Where it's assumed that there are two classes: C_1 and C_2 . $t_1 \in [0,1]$ and s_1 are the groundtruth and the score for C_1 , and $t_2=1-t_1$ and $s_2=1-s_1$ are the groundtruth and the score for C_2 . That is the case when we split a Multi-Label classification problem in CC binary classification problems. Logistic Loss and Multinomial Logistic Loss are other names for Cross-Entropy loss.

CHAPTER – 5

SYSTEM IMPLEMENTATION

5.1 LIST OF MODULES

- **MODULE-1**

Model generation and training using Existing algorithm, VGG-16.

- **MODULE-2**

Model generation and training using Novel algorithm.

- **MODULE-3**

Model Testing and prediction using Existing algorithm

- **MODULE-4**

Model Testing and prediction using Novel algorithm

5.2 MODULE DESCRIPTION

Module 1: The dataset is collected from Kaggle and UAC repositories. The images are then augmented and pre-processed. Then the data set is then split into training and testing datasets. Then the process of training takes place with VGG-16 architecture.

Module 2: The dataset is collected from Kaggle and UAC repositories. The images are then augmented and pre-processed. Then the data set is then split into training and testing datasets. Then the process of training takes place with novel architecture.

Module 3: The data is then tested with testing dataset. Later the model is then optimized with ADAM algorithm and loss minimized with Cross-Entropy. The prediction then takes place. Also, the training accuracy and loss with VGG-16 are plotted using matplotlib.

Module 4: The data is then tested with testing dataset. Later the model is then optimized with ADAM algorithm and loss minimized with Cross-Entropy. The prediction then takes place. Also, the training accuracy and loss with novel architecture are plotted using matplotlib.

5.3 DATASETS

These deep learning machines that have been working so well need fuel — lots of fuel; that fuel is data. The more labelled data we have, the better our model performs. In this project, we are able to determine the presence of cervical cancer using the CT images or x-rays. So, dataset consisting of CT images or x-rays which are used by the laboratories to analyze the presence of cervical cancer will be collected from various resources through internet. The idea of more data leading to better performance has even been explored at a large-scale by Google with a dataset of 300 Million images!

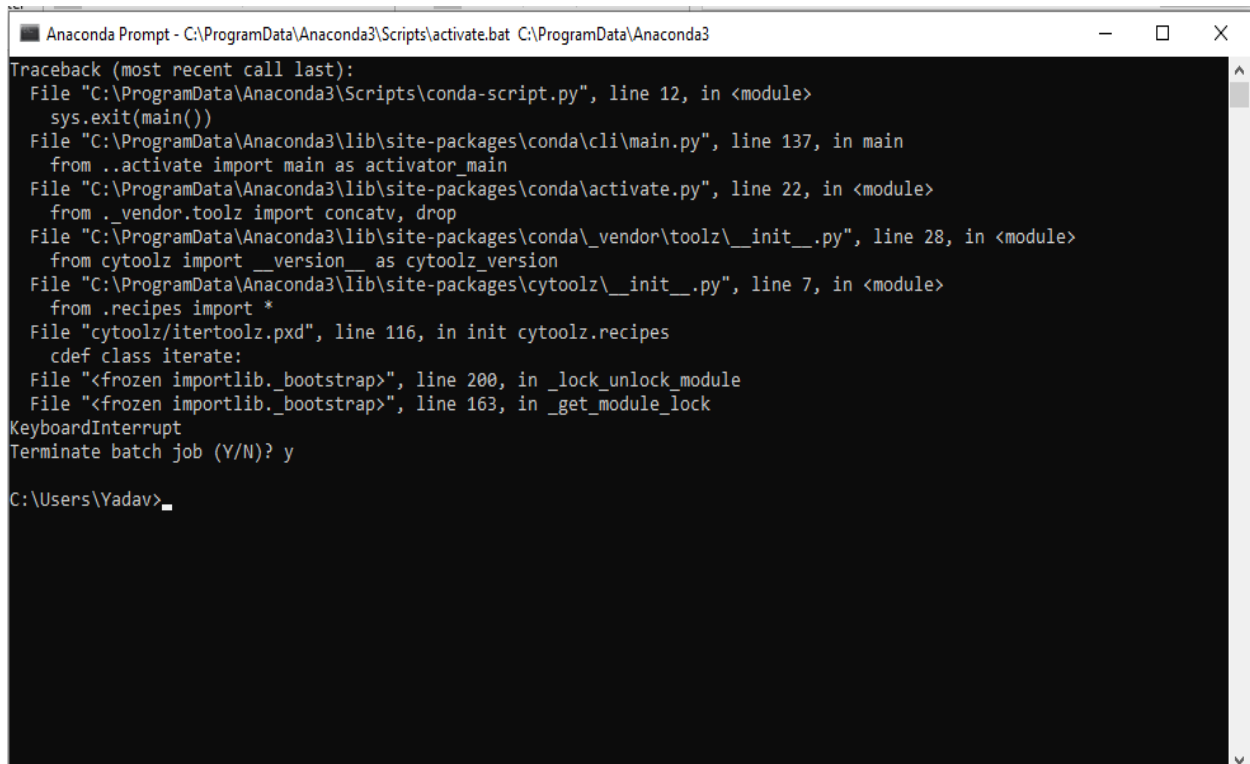
The images have been gathered from Kaggle and UAC repositories. There are four classes of image datasets namely, Squamous, Insitu, Dysplasia, Normal.

CHAPTER – 6

RESULTS AND DISCUSSION

6.1 RESULT

Initially the anaconda navigator is started which is used to perform all the required execution of the project. This can be seen in the following figure



```
Anaconda Prompt - C:\ProgramData\Anaconda3\Scripts\activate.bat C:\ProgramData\Anaconda3
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\Scripts\conda-script.py", line 12, in <module>
    sys.exit(main())
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\cli\main.py", line 137, in main
    from ..activate import main as activator_main
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\activate.py", line 22, in <module>
    from ._vendor.toolz import concatv, drop
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\_vendor\toolz\_init_.py", line 28, in <module>
    from cytoolz import __version__ as cytoolz_version
  File "C:\ProgramData\Anaconda3\lib\site-packages\cytoolz\_init_.py", line 7, in <module>
    from .recipes import *
  File "cytoolz\itertoolz.pxd", line 116, in init cytoolz.recipes
    cdef class iterate:
  File "<frozen importlib._bootstrap>", line 200, in _lock_unlock_module
  File "<frozen importlib._bootstrap>", line 163, in _get_module_lock
KeyboardInterrupt
Terminate batch job (Y/N)? y
C:\Users\Yadav>_
```

Figure 6.1 Starting the Anaconda Navigator

Then the path is moved to the folder where the code is saved to run it and it can be seen using the following figure.

```
Anaconda Prompt - C:\ProgramData\Anaconda3\Scripts\activate.bat C:\ProgramData\Anaconda3

Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\Scripts\conda-script.py", line 12, in <module>
    sys.exit(main())
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\cli\main.py", line 137, in main
    from ..activate import main as activator_main
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\activate.py", line 22, in <module>
    from .vendor.toolz import concatv, drop
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\_vendor\toolz\_init_.py", line 28, in <module>
    from cytoolz import __version__ as cytoolz_version
  File "C:\ProgramData\Anaconda3\lib\site-packages\cytoolz\_init_.py", line 7, in <module>
    from .recipes import *
  File "cytoolz\iterutils.pxd", line 116, in init cytoolz.recipes
  cdef class iterate:
  File "<frozen importlib._bootstrap>", line 200, in _lock_unlock_module
  File "<frozen importlib._bootstrap>", line 163, in _get_module_lock
KeyboardInterrupt
Terminate batch job (Y/N)? y

C:\Users\Yadav>cd projects
C:\Users\Yadav\projects>
```

Figure 6.2 Moving Inside the coding folder

The code for the VGG16 architecture training process can be seen in the following figure

```
C:\Users\Yadav\Downloads\main\main.py - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

# USAGE
# python main.py --dataset dataset/images --model cervical.model --output plt.png

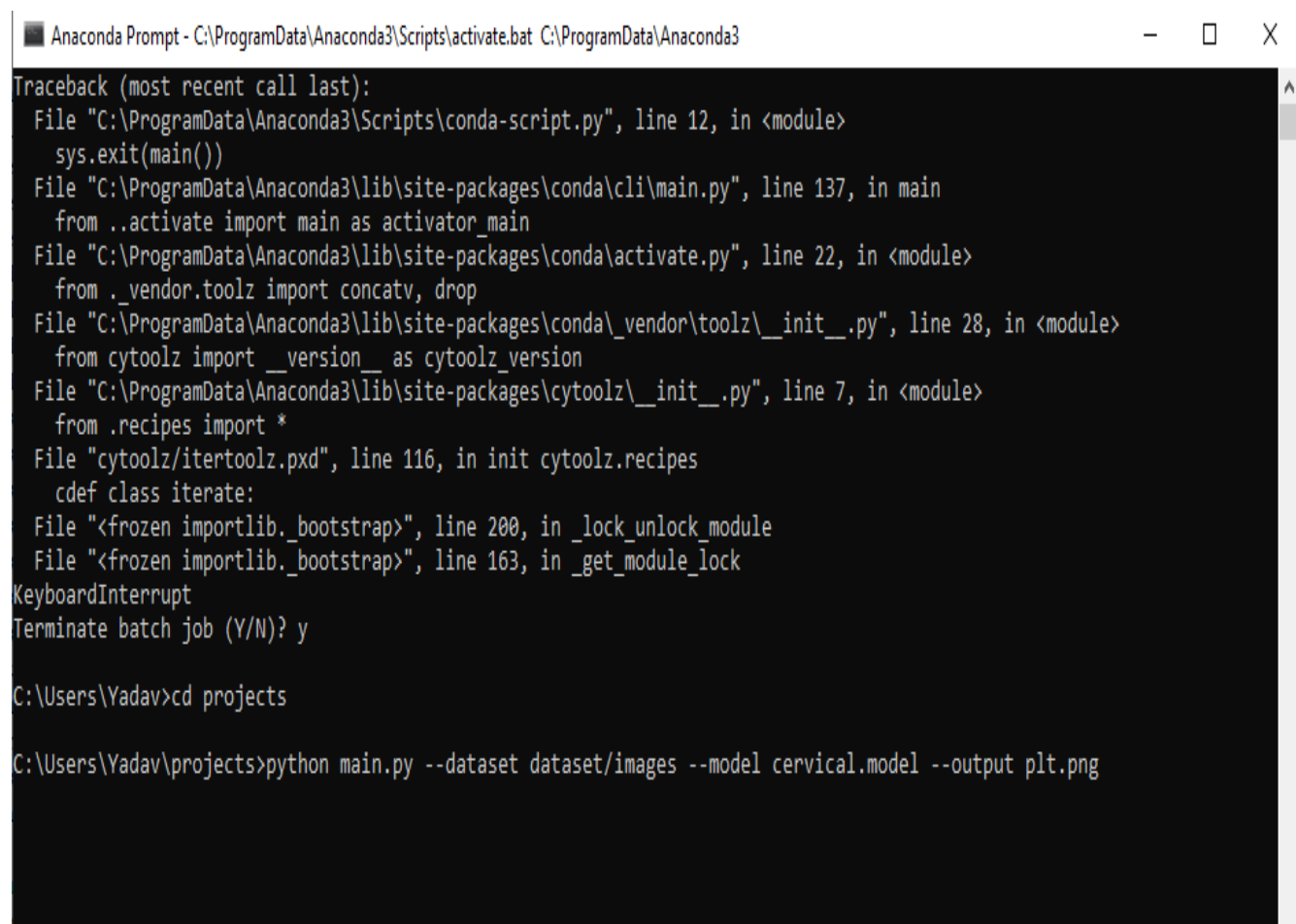
# import the necessary packages
import matplotlib
matplotlib.use("Agg")
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from headers.preprocessing import ImageToArrayPreprocessor
from headers.preprocessing import AspectAwarePreprocessor
from headers.datasets import SimpleDatasetLoader
from headers.nn.conv import FCHearNet
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from imutils import paths
import numpy as np
import argparse
import os
import matplotlib.pyplot as plt

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
    help="path to input dataset")
ap.add_argument("-m", "--model", required=True,
    help="path to output model")
ap.add_argument("-o", "--output", required=True,
    help="path to the output loss/accuracy plot")
args = vars(ap.parse_args())

Python file length: 5,824 lines: 147 Ln: 1 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
01:46 14-03-2020
```

Figure 6.3 VGG16 Coding

Then the code is run using certain commands in the command prompt which can be seen in the following figure

The image shows a screenshot of an Anaconda Prompt window. The title bar reads "Anaconda Prompt - C:\ProgramData\Anaconda3\Scripts\activate.bat C:\ProgramData\Anaconda3". The window has a dark background with white text. It displays a Python traceback error starting with "Traceback (most recent call last):". The error path goes through several files: "C:\ProgramData\Anaconda3\Scripts\conda-script.py", "C:\ProgramData\Anaconda3\lib\site-packages\conda\cli\main.py", "C:\ProgramData\Anaconda3\lib\site-packages\conda\activate.py", "C:\ProgramData\Anaconda3\lib\site-packages\conda_vendor\toolz_init_.py", and "C:\ProgramData\Anaconda3\lib\site-packages\cytoolz_init_.py". The error occurs in the "iterate" class of "cytoolz\itertoolz.pxd". After the error, the user presses a key, resulting in "KeyboardInterrupt". Then, the user enters "Terminate batch job (Y/N)? y". The prompt then shows the user navigating to "C:\Users\Yadav\projects" and running the command "python main.py --dataset dataset/images --model cervical.model --output plt.png".

```
Anaconda Prompt - C:\ProgramData\Anaconda3\Scripts\activate.bat C:\ProgramData\Anaconda3
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\Scripts\conda-script.py", line 12, in <module>
    sys.exit(main())
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\cli\main.py", line 137, in main
    from ..activate import main as activator_main
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\activate.py", line 22, in <module>
    from ._vendor.toolz import concatv, drop
  File "C:\ProgramData\Anaconda3\lib\site-packages\conda\_vendor\toolz\_init_.py", line 28, in <module>
    from cytoolz import __version__ as cytoolz_version
  File "C:\ProgramData\Anaconda3\lib\site-packages\cytoolz\_init_.py", line 7, in <module>
    from .recipes import *
  File "cytoolz\itertoolz.pxd", line 116, in init cytoolz.recipes
    cdef class iterate:
  File "<frozen importlib._bootstrap>", line 200, in _lock_unlock_module
  File "<frozen importlib._bootstrap>", line 163, in _get_module_lock
KeyboardInterrupt
Terminate batch job (Y/N)? y

C:\Users\Yadav>cd projects

C:\Users\Yadav\projects>python main.py --dataset dataset/images --model cervical.model --output plt.png
```

Figure 6.4 Executing in the Command Prompt

As the execution starts the datasets are fetched and processed and the whole execution takes place which can be seen in the below figure.

```
Anaconda Prompt (Anaconda3)

(base) C:\Users\dell>cd C:\Users\dell\project\Modified_Inception

(base) C:\Users\dell\project\Modified_Inception>python train.py --dataset dataset/images --output output.png --model new.model
[INFO] loading images...
[INFO] processed 500/1253
[INFO] processed 1000/1253
WARNING:tensorflow:From C:\Users\dell\Anaconda3\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from tensorflow.python.framework.ops)
is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\dell\Anaconda3\lib\site-packages\tensorflow\python\keras\layers\core.py:143: calling dropout (from tensorflow.python.ops.nn_ops) with keep
_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
[INFO] compiling model...
[INFO] training head...
WARNING:tensorflow:From C:\Users\dell\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
2020-03-09 19:34:02.796675: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
Epoch 1/70
314/314 [=====] - 8s 25ms/sample - loss: 13.0487 - acc: 0.1529
30/30 [=====] - 219s 7s/step - loss: 5.0698 - acc: 0.3727 - val_loss: 13.0342 - val_acc: 0.1529
Epoch 2/70
314/314 [=====] - 7s 23ms/sample - loss: 14.3227 - acc: 0.1529
30/30 [=====] - 210s 7s/step - loss: 4.1399 - acc: 0.5517 - val_loss: 14.3135 - val_acc: 0.1529
Epoch 3/70
314/314 [=====] - 7s 23ms/sample - loss: 14.1832 - acc: 0.1847
30/30 [=====] - 204s 7s/step - loss: 3.8851 - acc: 0.6347 - val_loss: 14.1809 - val_acc: 0.1847
Epoch 4/70
314/314 [=====] - 7s 22ms/sample - loss: 15.2043 - acc: 0.1529
30/30 [=====] - 202s 7s/step - loss: 3.7196 - acc: 0.6816 - val_loss: 15.1923 - val_acc: 0.1529
Epoch 5/70
314/314 [=====] - 7s 23ms/sample - loss: 14.8806 - acc: 0.1529
30/30 [=====] - 201s 7s/step - loss: 3.6392 - acc: 0.7039 - val_loss: 14.8701 - val_acc: 0.1529
Epoch 6/70
314/314 [=====] - 8s 26ms/sample - loss: 14.8630 - acc: 0.1274
30/30 [=====] - 208s 7s/step - loss: 3.4292 - acc: 0.7678 - val_loss: 14.8713 - val_acc: 0.1274
Epoch 7/70
```

Figure 6.5 Code Execution

As the execution gets completed the accuracy is been obtained and it can be seen in the following figure which is around 70%.

```
Anaconda Prompt (Anaconda3)
Epoch 26/30
222/222 [=====] - 73s 331ms/sample - loss: 0.9320 - acc: 0.7252
Epoch 27/30
222/222 [=====] - 72s 325ms/sample - loss: 0.9237 - acc: 0.7342
Epoch 28/30
222/222 [=====] - 73s 330ms/sample - loss: 0.9131 - acc: 0.7477
Epoch 29/30
222/222 [=====] - 72s 323ms/sample - loss: 0.9063 - acc: 0.7342
Epoch 30/30
222/222 [=====] - 73s 329ms/sample - loss: 0.8972 - acc: 0.7342
loss
acc
val_loss
val_acc
[INFO] evaluating after initialization...
      precision    recall  f1-score   support

   DYSPLACIA      0.51      0.95      0.66        59
     INSITU      1.00      0.67      0.80        66
    NORMAL      1.00      0.35      0.52        49
    SQAMOUS      0.90      0.96      0.93        48

   accuracy                   0.73       222
  macro avg      0.85      0.73      0.73       222
weighted avg      0.85      0.73      0.73       222

[INFO] serializing model...

(base) C:\Users\HP\project\low_weight\single_filter>
```

Figure 6.6 VGG16 Output

The model file generated by this architecture is found to have a size of about 105 MB that can be seen in the following figure.

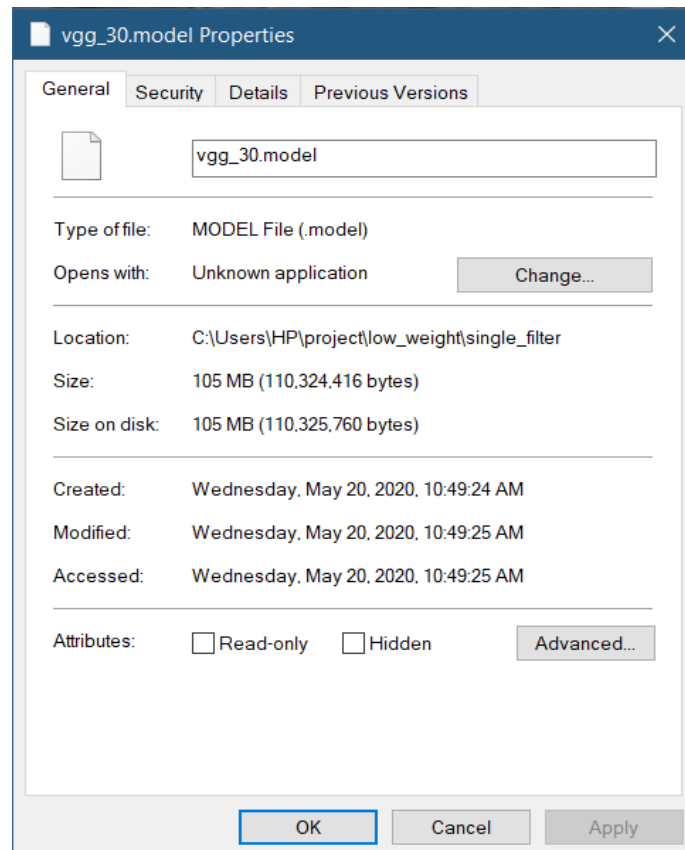


Figure 6.7 VGG16 Model Size

The model for VGG-16 Architecture is obtained with the size of 105 MB. The prediction for cervical cancer is shown in the following figure.

```
C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:535: FutureWarning: Passing (type, 1) or
ood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
[INFO] loading model...
WARNING:tensorflow:From C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:435: co
ure version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\keras\layers\core.py:143: calling dr
a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
2020-05-23 09:11:00.967810: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that t
WARNING:tensorflow:From C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
Instructions for updating:
Use tf.cast instead.
[INFO] starting video stream thread...
SQAMOUS
```

Fig 6.8 Prediction for VGG16 Architecture

The same process is carried out for our novel architecture and we obtained an accuracy of about 97% which can be seen in the following figure.

```

222/222 [=====] - 10s 47ms/sample - loss: 0.6298 - acc: 0.9550
21/21 [=====] - 387s 18s/step - loss: 0.6337 - acc: 0.9639 - val_loss: 0.6293 - val_acc: 0.9550
Epoch 28/30
222/222 [=====] - 10s 47ms/sample - loss: 0.6305 - acc: 0.9550
21/21 [=====] - 387s 18s/step - loss: 0.6178 - acc: 0.9699 - val_loss: 0.6300 - val_acc: 0.9550
Epoch 29/30
222/222 [=====] - 10s 47ms/sample - loss: 0.6189 - acc: 0.9640
21/21 [=====] - 387s 18s/step - loss: 0.6346 - acc: 0.9563 - val_loss: 0.6184 - val_acc: 0.9640
Epoch 30/30
222/222 [=====] - 10s 47ms/sample - loss: 0.6132 - acc: 0.9640
21/21 [=====] - 387s 18s/step - loss: 0.6193 - acc: 0.9699 - val_loss: 0.6128 - val_acc: 0.9640
loss
acc
val_loss
val_acc
[INFO] evaluating after initialization...
      precision    recall  f1-score   support

 DYSPLACIA      0.92      0.98      0.95        59
    INSITU      1.00      0.97      0.98        66
    NORMAL      0.94      0.90      0.92        49
    SQAMOUS      1.00      1.00      1.00        48

 accuracy              0.96        222
  macro avg           0.96      0.96      0.96        222
weighted avg           0.96      0.96      0.96        222

[INFO] serializing model...

```

Figure 6.9 Light Weight Architecture Output

The model file generated for the novel architecture had a size of about 16.8MB which is 5 times smaller than the VGG16 architecture and that can be seen in the following figure.

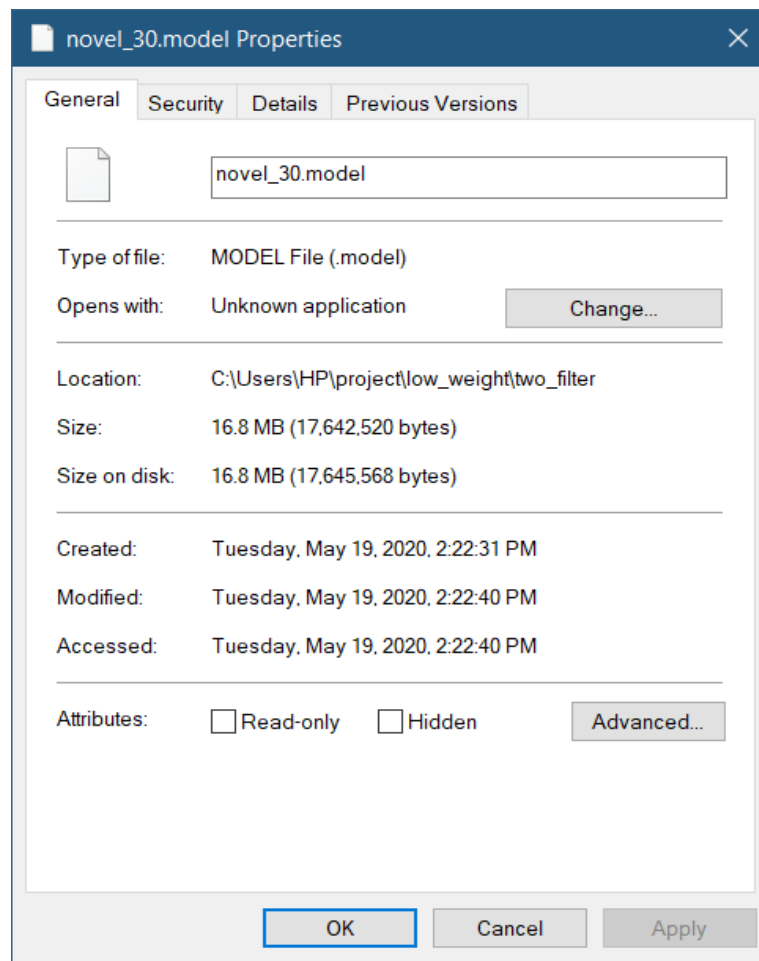


Figure 6.10 Light Weight Architecture Model Size

Thus, from the results obtained it is clear enough that we have successfully designed a novel low weight architecture which generates a model file of very low weight that can be compatible for real time applications. Thus, we achieved the scope of the project.

```

C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:527: FutureWarning
ood as (type, (1,)) / '(1,)type'.
    _np_quint8 = np.dtype(["quint8", np.uint8, 1])
C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:528: FutureWarning
ood as (type, (1,)) / '(1,)type'.
    _np_quint16 = np.dtype(["quint16", np.int16, 1])
C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:529: FutureWarning
ood as (type, (1,)) / '(1,)type'.
    _np_quint16 = np.dtype(["quint16", np.uint16, 1])
C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:530: FutureWarning
ood as (type, (1,)) / '(1,)type'.
    _np_quint32 = np.dtype(["quint32", np.int32, 1])
C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:535: FutureWarning
ood as (type, (1,)) / '(1,)type'.
    np_resource = np.dtype(["resource", np.ubyte, 1])
[INFO] loading model...
WARNING:tensorflow:From C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\ops\resource
ure version.
Instructions for updating:
Colocations handled automatically by placer.
2020-05-23 08:28:08.006048: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU sup
WARNING:tensorflow:From C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\ops\math_op
Instructions for updating:
Use tf.cast instead.
[INFO] starting video stream thread...
INSITU

```

Fig 6.11 Prediction for Light weight Architecture

The model for Light weight Architecture is obtained with the size of 16.8 MB. The prediction for cervical cancer is shown in the above figure.

CHAPTER – 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

This project proves that the novel architecture designed by us is better than VGG16 for real time application. As well as Novel architecture approximates a sparse CNN with a normal dense construction. It can be used to reduce the size of the model generated to prove it applicable in real time. So, reduces the time required for manual classification and eliminates the human error rate by this project.

7.2 FUTURE WORK

In the coming future, we review the application of the novel architecture design technology in the medical field and it can promote better architecture than VGG16 for real time application. In the medical field there are more chances to develop or convert this project in many ways. Thus, this project will be developing an application over prediction of cervical cancer presence with three different types.

REFERENCE

- [1] FIZZA ABBAS, (Student Member, IEEE), UBAIDULLAH RAJPUT, (Student Member, IEEE), AND HEEKUCK OH, (Member, IEEE), “PRISM: PRivacy-Aware Interest Sharing and Matching in Mobile Social Networks”, (Student Member, IEEE.
- [2] Qi Wang, Senior Member, IEEE, Jia Wan, Xuelong Li, Fellow, IEEE, “Comb-Robust Hierarchical Deep Learning for Vehicular Management”, IEEE Transactions on Vehicular Technology.
- [3] Shiann-Rong Kuang, Member, IEEE, Kun-Yi Wu, and Ren-Yao Lu, “Low-Cost High-Performance VLSI Architecture for Montgomery Modular Multiplication”, IEEE transactions on very large-scale integration (vlsi) systems.
- [4] Shiann-Rong Kuang, Member, IEEE, Chih-Yuan Liang, and Chun-Chi Chen, “An Efficient Radix-4 Scalable Architecture for Montgomery Modular Multiplication”, IEEE Transactions on Circuits and Systems II: Express Briefs.
- [5] Tucker James Marion and Marc H. Meyer, “Organizing to Achieve Modular Architecture Across Different Products”, IEEE transactions on engineering management.
- [6] Thomas Kohler, Frank Dürr, and Kurt Rothermel, “ZeroSDN: A Highly Flexible and Modular Architecture for Full-Range Distribution of Event-Based Network Control”, IEEE transactions on network and service management, vol. 15, no. 4, december 2018.
- [7] Wen-Ching Lin, Jheng-Hao Ye, and Ming-Der Shieh, Member, IEEE, “Scalable Montgomery Modular Multiplication Architecture with Low Latency and Low Memory Bandwidth Requirement”, IEEE transactions on computers.
- [8] Yazhou Liu , Sen Cao, Pongsak Lasang, Member, IEEE, and Shengmei Shen, Member, IEEE, “Modular Lightweight Network for Road Object Detection Using a Feature Fusion Approach”, IEEE transactions on systems, man, and cybernetics: systems.

[9] Yue Deng, Feng Bao, Xuesong Deng, Ruiping Wang, Member, IEEE, Youyong Kong, and Qionghai Dai, Senior Member, IEEE, “Deep and Structured Robust Information Theoretic Learning for Image Analysis”, IEEE Transactions on Image Processing.

[10] Ziyad Alharbi, Akhilesh S. Thyagaturu, Martin Reisslein, Fellow, IEEE, Hesham ElBakoury, and Ruobin Zheng, “Performance Comparison of R-PHY and R-MACPHY Modular Cable Access Network Architectures”, IEEE transactions on broadcasting.