CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The revolution of computer in the past decades, have unfolded the use of computers, be it any field. The Era of maintaining huge records for data storage has evolved. Computers have become most used tool in almost each field. It has made the life of humans easier. Also, Computers have made research very easier. Biomedical has offered modern medical devices for diagnostic and preventive purposes, which include diagnostic test kits, vaccines, antibodies and radiolabelled biological therapeutics used for imaging and investigation purpose. It has played a prominent role in improving the challenges regarding to human health as it has flexibility to reduce global health differences by the provision of promising technologies. Many types of cancers are being detected and has become very common disease with the evolution of the technology.

Skin cancer is the most explored disease. With increased use of cosmetics, pollution & radiations, cancer is becoming a common disease in the modern era. The images of the affected area termed as ―lesion‖ are captured with the help of derma scope and are fed as input to the algorithm. Several algorithms have been proposed which requires input to be fed manually. The aim of this work is to propose an algorithm which require minimal intervention of doctors. In the recent years, due to the increased use of cosmetics, and pollution & radiations, cancer is becoming a common disease in the modern world.

1.2 STAGES OF SKIN CANCER

As soon as the disease is discovered, next task would be determining in which stage the cancer is. The stage in which the cancer is can be determined by various factors such as thickness, the depth of penetration, and the extent to which the melanoma has spread. Based on the stage determined, the patients are treated. The first stage of the skin cancer, that is the early stage of melanoma (Stage 0 and Stage 1) are insular. Stage 0 tumours are in situ, which means they are non-invasive and have not entered beneath the external layer of the skin (the epidermis). Stage I tumours have attacked beneath the epidermis into the skin's next layer (the dermis) yet are little and have no different characteristics, for example, ulceration that put them at high danger of spreading (metastasizing) to close by lymph hubs or beyond. Stage II tumors, however limited, are bigger (for the most part 1 mm. thick or bigger) and additionally may have different characteristics, for example, ulceration that put them at high danger of spreading to the close by lymph hubs or beyond. They are viewed as transitional or "high-chance" melanomas. Further developed melanomas (Stages III and IV) have metastasized to different parts of the body. There are additionally subdivisions inside stages.

1.3 MACHINE LEARNING

Machine learning is not bond to one field it consists of many things, and it is growing rapidly. The whole objective of machine learning is to program, so as to make the computer perform tasks accordingly without explicitly programming each time. A computer learns from the past inputs given by the user, records the patterns and improves with the experience. When a computer is provided with data and then recognize those data with the help of an algorithm, it is called machine learning, the process by which the machine learns is called as training and output processed is the model. The objective of the learner is that to generalize from its experience. Generalization means the ability of the machine to perform accurately on unseen or new data. In the past decade machine learning has amazed us with the invention of automated cars, practical speech recognition, effective web search, and a vast understanding of human genome. Machine learning today, is widely spread that you probably use it dozens of times a day without knowing it.

Machine learning in health care industry are taking the industry by storm. Machine learning in medicine has recently made buzz. With regards to adequacy of machine adapting, more information quite often yields better outcomes—and the human services area is perched on an information goldmine. Calculations can furnish quick advantage to disciplines with procedures that are reproducible. Likewise, those with expansive picture datasets, for example, radiology, cardiology, and pathology, are solid applicants. Machine learning can be prepared to take an insight at pictures, distinguish anomalies, and point to regions that need consideration, consequently enhancing the exactness of every one of these procedures. Long haul, machine learning will profit the family expert or internist at the bedside. Machine learning can offer a target assessment to enhance productivity, dependability, and exactness.

1.4 REGRESSION

Regression examination is a type of predictive displaying method which researches the connection between a reliant (target) and autonomous variable (s) (indicator). This system is utilized for estimating, time arrangement displaying and finding the causal impact connection between the factors. For instance, connection between rash driving and number of street mischances by a driver is best concentrated through relapse. Regression investigation is a critical instrument for displaying and breaking down information. Here, we fit a curve/line to the information focuses, in such a way, to the point that the contrasts between the separations of information focuses from the curve or line is limited. Regression investigation gauges the connection between at least two factors. We should comprehend this with a simple precedent: Suppose, you need to gauge development in offers of an organization dependent on current monetary conditions. You have the ongoing organization information which demonstrates that the development in deals is around more than two times the development in the economy. There are various advantages of utilizing relapse investigation. They are as per the following:

- It shows the noteworthy connections between ward variable and free factor.
- It shows the quality of effect of numerous autonomous factors on a reliant variable.

Regression examination likewise enables us to look at the impacts of factors estimated on various scales, for example, the impact of value changes and the quantity of limited time exercises. These advantages help economic specialists/information experts/information researchers to wipe out and assess the best arrangement of factors to be utilized for building prescient models.

CHAPTER 2

SYSTEM ANALYSIS

2.1 PROBLEM STATEMENT

Skin cancer is the most common due to abnormal growth of skin cells. But this growth always may not be the symptom of deadly cancer. So, distinguishing correctly and proper treatment leads to better cure of the disease. In this project, convolutional neural network (CNN), a machine learning classification technique is used to classify the skin cancer images. As accuracy is the most important factor in this problem, by taking more number of images for training the network and by increasing the number of iterations, the CNN accuracy can be enhanced. Tensor Flow is a large scale machine learning system developed by Google and Inception V3 is Google's CNN architecture here, the CNN algorithm is implemented with Tensor Flow and Inception V3.

2.2 PROBLEM ANALYSIS

Skin is the largest organ of the body. Its importance comes from the way it protects the internal body tissues from the external environment, i.e., skin keeps the body temperature at a constant level, protects our body from undesirable sun radiation such as ultraviolet (UV) light exposure, prevents infections and allows the production of vitamin D, essential for many body functions . In the past few years the number of skin cancer cases has been going up and studies announce that the frequency of melanoma doubles every 20 years. Skin cancer, the most predominant type of cancer, is produced when skin cells begin to grow out of control. There are 3 main types of skin cancers: basal cell skin cancers (basal cell carcinomas), squamous cell skin cancers (squamous cell carcinomas) and melanomas. Generally, skin cancers that are not melanomas are commonly grouped as non-melanoma skin cancers. This project is focused on melanoma detection, which is a fatal form of skin cancer often undiagnosed or misdiagnosed as a benign skin lesion. There are an estimated 76,380 new cases of melanoma and an estimated 6,750 deaths each year in the United States. Melanoma can be detected by simple visual examination since it occurs on the skin surface, but an early detection is imperative the lives of melanoma patients depend on accurate and early diagnosis.

This poses additional challenges to the task of distinguishing among skin lesions, especially between benign or malignant tumours, due to the large imbalance in the number of samples of each class of tumours. These aspects must be kept in mind when designing automatic skin lesions classifiers systems whose performance should be at least comparable to traditional detection methods. That is the reason to distinguish among skin lesions, especially between benign or malignant cancer, has allowed the emergence of automatic skin lesions classifiers systems to compete against traditional detection methods. Most current methods in the field of skin lesion classification rely on hand-crafted features, such as ABCDE rule (the acronym stands for Asymmetry, Border, Colour, Dermoscopic structure and Evolving) , 3-point checklist , 7-point checklist, Menzies method and CASH (Colour, Architecture, Symmetry, and Homogeneity). Physicians often rely on personal experience and evaluate each patient's lesions on a case-by-case basis by taking into account the patient’s local lesion patterns in comparison to that of the entire body. Without any type of computer-based assistance, the clinical diagnosis accuracy for melanoma detection is reported to be between 65 and 80% [16]. Use of dermoscopic images, pictures taken skin by a skin surface microscopy, improves diagnostic accuracy of skin lesions by 49%. However, the visual differences between melanoma and benign skin lesions can be very subtle (Figure 1.1) , making it difficult to distinguish the two cases, even for trained medical experts. For the reasons described above, an intelligent medical imaging-based skin lesion diagnosis system can be a welcome tool to assist a physician in classifying skin lesions.

2.3 RESOURCES REQUIRED

2.3.1 CNN

Convolutional Neural Networks (also known as CNNs or ConvNets) maintain a strong relationship with Artificial Neural Networks: they are also inspired in the behavior of biological systems through artificial neurons with learnable weights and biases. The layered architecture that Neural Networks performs based on matrix multiplications enables its application for image classification tasks. For this reason, ConvNets architectures assume that the input are images that have to 19 be transformed into an output holding the class score predicted. The loss function is used to measure how well the predicted scores agrees with the ground truth labels in the input data. Most common loss functions are the Multiclass SVM and the Softmax.

ConvNets work similarly to Neural Networks: each neuron receive an input, A product (Hadamard product or elementwise multiplication) between each input and its associated weight is performed, followed with a non-linearity. The most common hierarchical distribution of ConvNets layers contains: Input layer, containing the raw pixel values from input images.

• Convolutional layers, the core block of ConvNets, computes a locally dot product (2D in the case of images) between the weights and a certain tiny region of the input volume.

• Non-linear layers, most of the times using a ReLU activation function which applies an elementwise activation by thresholding at zero.

Pooling layers that apply a spatial down sampling along the output volume. Fully Connected layers that compute the class scores, A CNN structure is made up of repetitive patterns (which explains the expression deep learning) of Convolutional, ReLU and Pooling layers (considered hidden layers) and finally the fully-connected layers. The resulting volume structure is called feature map (in the case of images, it has a two dimension volume). The learning process (also referred to network training) where weights are optimized is achieved through backpropagation, a technique to efficiently compute gradients for its weights with respect to the loss function.

2.3.2 TENSORFLOW

Tensor Flow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, Tensor Flow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

Tensor Flow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as *tensors*. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned Tensor Flow, of which only 5 were from Google.

2.3.3 GOOGLE CLOUD

GCP consists of a set of physical assets, such as computers and hard disk drives, and virtual resources, such as VMs, That are contained in Google's data centers around the globe. Each data center location is in a global region. Regions include Central US, Western Europe, and East Asia. Each region is a collection of zone, which are isolated from each other within the region. Each zone is identified by a name that combines a letter identifier with the name of the region. For example, zone a in the East Asia region is named asia-east1-a.This distribution of resources provides several benefits, including redundancy in case of failure and reduced latency by locating resources closer to clients. This distribution also introduces some rules about how resources can be used together.

In cloud computing, what you might be used to thinking of as software and hardware products, become service. These services provide access to the underlying resources. The list of available GCP services is long, and it keeps growing. When you develop your website or application on GCP, you mix and match these services into combinations that provide the infrastructure you need, and then add your code to enable the scenarios you want to build.

Some resources can be accessed by any other resource, across regions and zones. These global server include preconfigured disk images, disk snapshots, and networks. Some resources can be accessed only by resources that are located in the same region. These regional resources include static external IP addresses. Other resources can be accessed only by resources that are located in the same zone. These zonal resources include VM instances, their types, and disks.

The following diagram shows the relationship between global scope, regions and zones, and some of their resources:

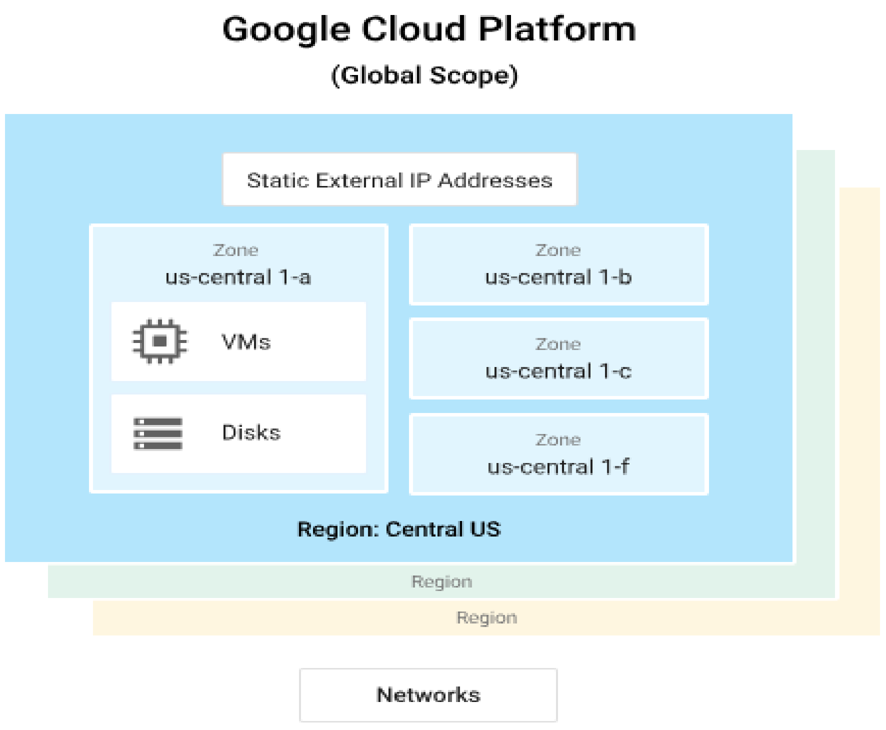


Fig 2.1 Google Cloud Architecture

2.3.4 DOCKER

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines! Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

2.4 FEASIBILITY STUDY

The emergence of a machine learning paradigm known as deep learning and recent advances in computational power have enabled the development of intelligent medical image analysis systems that can display remarkable performance in comparison to hand-crafted features. Artificial neural networks have produced promising results when classifying skin lesions. The purpose of this work is to assess the performance of Convolutional Neural Networks in building models for the diagnosis of skin lesions. To achieve this goal, this work is divided into three parts:

- The development of an algorithm for automated prediction of skin lesion segmentations from dermoscopic images in the form of binary masks.
- The design of a method for automatic classification of skin lesions from dermoscopic images.
- The evaluation of skin lesion segmentation impact on the accuracy of the designed classifier.

The hypothesis of this work states that previous segmentation of an image containing a skin lesion (i.e., isolating the lesion from the background) improves the accuracy and sensitivity of a Deep Learning model approach for a 2-class classifier for early melanoma detection based on skin lesion dermoscopic images. This assumption considers that segmentation removes nonessential information, such as hair or non-target surrounding lesions, and helps in accurately classifying the lesion. However, segmentation may lose information that could be contextually relevant to the CNN. Segmenting images using state of the art techniques provides promising but not perfect results and has yet to be proven to be robust across different datasets. This project was carried out during the fall semester 2016 at the Department of Computer & Electrical Engineering and Computer Science of the Florida Atlantic University. Furthermore, this project contributed to the Medical Imaging Diagnosis using Deep Learning research group from FAU.

Differential diagnosis of melanoma from melanocytic nevi is often not straightforward. Thus, a growing interest has developed in the last decade in the automated analysis of digitized images obtained by epiluminescence microscopy techniques to assist clinicians in differentiating early melanoma from benign skin lesions. The aim of this study was to evaluate diagnostic accuracy provided by different statistical classifiers on a large set of pigmented skin lesions grabbed by four digital analyzers located in two different dermatological units. Images of 391 melanomas and 449 melanocytic nevi were included in the study. A linear classifier was built by using the method of receiver operating characteristic curves to identify a threshold value for a fixed sensitivity of 95%. A K-nearest-neighbour classifier, a nonparametric method of pattern recognition, was constructed using all available image features and trained for a sensitivity of 98% on a large exemplar set of lesions. On independent test sets of lesions, the linear classifier and the K-nearest-neighbour classifier produced a mean sensitivity of 95% and 98% and a mean specificity of 78% and of 79%, respectively. In conclusion, our study suggests that computer-aided differentiation of melanoma from benign pigmented lesions obtained with DB-Mips is feasible and, above all, reliable. In fact, the same instrumentations used in different units provided similar diagnostic accuracy. Whether this would improve early diagnosis of melanoma and/or reducing unnecessary surgery needs to be demonstrated by a randomized clinical trial.

2.5 EXISTING SYSTEM

Existing systems are capable of detecting skin cancer cells efficiently.The current systems are using tensor flow in order to understand the given image and then detect the cancer cells. Many researchers have been working on the Computer vision approach for skin cancer detection. For segmentation of skin lesion in the input image, existing systems either use manual, semi-automatic or fully automatic border detection methods. The features to perform skin lesion segmentation used in various papers are:

- Shape
- Colour
- Texture
- Luminance

The ABCD rule of dermoscopy, suggest that asymmetry is given the most prominent among the four features of asymmetry, border irregularity, colour and diameter. A number of studies have been carried out on quantifying skin cancer. In Some techniques, the symmetry feature is calculated based on geometrical measurements on the whole lesion, e.g. symmetric distance and circularity Other studies, propose the circularity index, as a measure of irregularity of borders in dermoscopy images. The project gives the overview of the most important implementations in the literature and compares the performance of several classifiers on the specific skin lesion diagnostic problem.

Convolution Neural Networks (CNN) as future scope, since CNN models can be used for classification of the affected skin images without the need for performing segmentation and feature extraction independently. Have made use of custom made automated segmentation and have used a novel approach for implementation of the CNN methodology, where CNN has been used for feature extraction and ANN was used to classify those extracted features. The advantage of this proposed system is that CNN does not require any additional classifier like SVM, KNN since 3 fully-connected layers were used for training the classification model. This type of classification brings its own unique benefits, like it is possible to apply back-propagation algorithm, which adjusts the parameters of neurons in all layers to obtain better classification model.

2.5.1 LIMITATIONS

Lack of proper classification of cancer lesions. They lack the effectiveness because of not using proper neural network training methodology. Based on very less input datasets.We recognize that there are limitations to this review. It was restricted to the descriptions available from the online stores; more information may have been available after purchasing or using the apps, but this was not undertaken in order to ensure equity across the review. Moreover, we felt that users should be able to make an informed choice about the apps by reading their description in the online stores.

We reviewed apps only available in English; therefore we may have failed to identify any apps published in other languages. In reality this is unlikely as the U.S.A. and U.K. were the main countries publishing these apps, with a few also coming from Western Europe and South America. Finally, we only reviewed apps applicable to Apple and Android phones as we knew that more than 90% of apps are available via these two platforms. It is likely that small platforms would use the same apps – as some were available via both of the major platforms – and we are therefore confident that few are missing.

2.6 PROPOSED SYSTEM

2.6.1 DATA

The International Skin Imaging Collaboration: Melanoma Project is an academia and industry partnership designed to facilitate the application of digital skin imaging to help reduce melanoma mortality. When recognized and treated in its earliest stages, melanoma is readily curable. Digital images of skin lesions can be used to educate professionals and the public in melanoma recognition as well as directly aid in the diagnosis of melanoma through teledermatology, clinical decision support, and automated diagnosis. Currently, a lack of standards for dermatologic imaging undermines the quality and usefulness of skin lesion imaging. ISIC is developing proposed standards to address the technologies, techniques, and terminology used in skin imaging with special attention to the issues of privacy and interoperability (i.e., the ability to share images across technology and clinical platforms). In addition, ISIC has developed and is expanding an open source public access archive of skin images to test and validate the proposed standards. This archive serves as a public resource of images for teaching and for the development and testing of automated diagnostic systems. Datasets from ISIC that we are going to user are follows:

- ISICUDA-21
- ISICMSK-21
- ISICUDA-11
- ISICMSK-11
- ISICMSK-12

2.6.2 TENSORFLOW

Tensor Flow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, Tensor Flow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Tensor Flow computations are expressed as stateful dataflow graphs. The name Tensor Flow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as *tensors*. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned Tensor Flow, of which only 5 were from Google.

2.6.3 GOOGLE CLOUD

GCP consists of a set of physical assets, such as computers and hard disk drives, and virtual resources, such as VMs, That are contained in Google's data centers around the globe. Each data center location is in a global region. Regions include Central US, Western Europe, and East Asia. Each region is a collection of zone, which are isolated from each other within the region. Each zone is identified by a name that combines a letter identifier with the name of the region.

For example, zone a in the East Asia region is named asia-east1-a.This distribution of resources provides several benefits, including redundancy in case of failure and reduced latency by locating resources closer to clients. This distribution also introduces some rules about how resources can be used together.In cloud computing, what you might be used to thinking of as software and hardware products, become service. These services provide access to the underlying resources. The list of available GCP services is long, and it keeps growing.

When you develop your website or application on GCP, you mix and match these services into combinations that provide the infrastructure you need, and then add your code to enable the scenarios you want to build.Some resources can be accessed by any other resource, across regions and zones. These global server include preconfigured disk images, disk snapshots, and networks. Some resources can be accessed only by resources that are located in the same region. These regional resources include static external IP addresses. Other resources can be accessed only by resources that are located in the same zone. These zonal resources include VM instances, their types, and disks.

CHAPTER 3

LITERATURE SURVEY

The Economic Value of Ipilimumab as a Second-Line Treatment in Patients With Advanced Melanoma In early 2011, ipilimumab, an anti-CTLA- 4 monoclonal antibody with antitumor activity, was approved by the US Food and Drug Administration for the treatment of advanced melanoma. Ipilimumab was compared with the gp100 vaccine in a clinical trial and patients survived a median of 3.7 months longer than those receiving the vaccine. In another trial, ipilimumab with dacarbazine was compared with dacarbazine in previously untreated patients with advanced melanoma. The investigators found that combination therapy yielded higher survivals at 1, 2, and 3 years. Ipilimumab is an expensive treatment that is associated with positive clinical outcomes. Researchers from the United States and United Kingdom (and from Bristol-Myers Squibb, the manufacturer of ipilimumab) sought to evaluate the cost-effectiveness of ipilimumab compared with best supportive care in previously treated patients with advanced (unresectable or metastatic) melanoma.

Progression-free and overall survival data were used from a phase III trial of ipilimumab to model stable disease, progression, and death. Clinical outcomes, quality of life, and healthcare utilization were included in the analyses. The Markov model considered only the direct costs in patients with an average age of 55 years having advanced melanoma. A discount rate of 3% was utilized for both costs and outcomes. Because no previous clinical trial with another therapy demonstrated a prolongation of survival, best supportive care was defined as disease management without active chemotherapy. The researchers acknowledged that this may provide a conservative cost benefit for ipilimumab, as most patients usually receive some type of chemotherapy that increases treatment costs of advanced melanoma. Both the incremental costeffectiveness ratio and incremental cost utility ratio were estimated in the model. They included direct costs associated with pharmaceuticals, clinical management, and costs associated with toxicity-related events The model used ipilimumab clinical data from a trial comparing the agent with gp100, dacarbazine, interleukin-2, and temozolomide.

3.1 ABCD RULE BASED DETECTION

In this paper [9], Hardian et.al uses ABCD rule of dermoscopy for extracting the skin lesion. A smartphone camera is used to capture the lesion image and this image is processed by using the ABCD rule. Feature extraction is performed on the pre-processed image where the four features - Asymmetry (A), Border (B), Color(C), and D (Diameter) are extracted in the following way.

1) Asymmetry – Melanoma lesions are asymmetric in nature. Asymmetry index is used for determining the level of symmetry of the object. This is done by dividing the image horizontally or vertically.
2) Border - In case of melanoma the border is irregular, ragged, and blurred. The compactness index is used to determine the border irregularity.
3) Colour – Melanoma are not uniform in color unlike the benign mole. Normalized Euclidean distance between each pixel is used to determine the color uniformity.
4) Diameter - The melanoma lesion has larger than 6 mm.

The diameter in the image is found out and compared to 6mm measurement. The major drawbacks of the proposed method are:

1) According to, in order to analyse the ABCD score, the criteria are assigned semi-quantitatively. Each of the criteria is then multiplied by a given weight factor to calculate a total dermoscopy score. The ABCD rule works appropriately for thin melanocytic wounds. The ABCD rule has about 59% to 88% accuracy in diagnosing melanoma, but biopsy is needed for more precise diagnosis.
2) According to, the ABCD method, in some cases, isn't helpful to classify malignant and benign skin moles. In addition, the method may not recognize some malignant moles at early stages, for example malignant melanoma with a small diameter than 6mm. This poses the challenge of differentiating malignant melanoma and birthmarks

3.2 BACKPROPAGATION NEURAL NETWORKS

Based diagnosis In this paper [10], Pratik propose the use of the ABCD rule for extracting features and a backpropagation neural network to classify the lesions as melonama or benign. The criteria that combine to create the ABCD rule of dermoscopy are asymmetry, border, color, and diameter. In this paper, they have extracted the features based on the ABCD rule after successful segmentation. As the number of classes increase, it becomes difficult to classify lesions into their appropriate classes accurately. Neural Networks inherently possess better capability of handling complex relationships between different parameters. Thus, the proposed model makes use of Backpropagation neural networks. However, the proposed method as certain drawbacks. The major drawbacks of this method are slow convergence rates and trapping in local minima. According to the backpropagation algorithm is known as a local search algorithm which uses gradient descent to iteratively develop the weights and biases in the neural network

3.3 HYBRID ANN AND WORLD CUP OPTIMIZATION

In this paper [11] Navid Razmjooy have proposed a new hybrid algorithm between the artificial neural network and world cup optimization for enhancing the back-propagation algorithm efficiency and for escaping from trapping in the local minima. World cup optimization algorithm has been used to help ANN to find the initial optimal weights in the back-propagation algorithm, to speed up the convergence speed and to minimize the root mean square error between the actual output and the target output. In WCO algorithm, the best teams from each group arise to the next level and the rest are eliminated. They have used the same concept in Backpropagation algorithm to search around the best cost for some epochs and if the search result is better than the best cost, the output will be the achieved; otherwise, previous output will be selected. The biggest advantage of the proposed method is the use of world cup optimization to resolve the drawbacks of backpropagation algorithm.

3.4 HYBRID GENETIC ALGORITHM- ANN

In this paper[12], Ashwin presents a computer aided approach for skin cancer detection. The steps involved are

1) image processing

2) segmentation

3) feature extraction and

4) classification.

The Image processing step includes image resizing and image hair removal using dull razor software, which is free medical imaging software. Even though the proposed model makes use of an efficient software tool, the paper neglects the negative effects that the tool may have. The software makes use of a mean filter which effectively smoothens the hair in the images; however, in this process it may smoothen the edges and thus compromise the quality of the image. The next step involves segmentation using the open source ImageJ software. However, the author of the paper ignores the fact that the ImageJ software has certain drawbacks. The major drawback is that if you're going to deal with large stacks of images, you may bump up against problems in the default memory configuration for the program. The Otsu color thresholding which is a simple yet powerful thresholding technique is employed in this paper.

Threshold level is set such that the background skin pixels are removed and only foreground lesion remains. The major drawback of thresholding methods, according to is that they can achieve good results only if there is a high contrast between the lesion area and the surrounding skin region, which may not always be the case. They have then used GLCM for feature extraction and to obtain the features, namely Contrast, Correlation, Homogeneity and Energy. The selected features are given as input to classifier which classifies the images as cancerous or non-cancerous .The proposed system for feature extraction makes use of the fact that Malignant melanoma has a mix of red, green, and dark coloration whereas benign lesion has a uniform color pattern. Finally, the Classification is done using a hybrid Genetic algorithm – Artificial neural network classifier.

A conventional ANN classifier uses backpropagation algorithm for training. However, the major drawback of this conventional technique is that the solution may get trapped in the local minima instead of global minima. In order to eliminate this problem, a hybrid approach has been used where the weights of ANN classifier are optimized by the genetic algorithm to improve accuracy. This Paper shows that the results are better using the hybrid approach than the conventional approach.

3.5 CONVOLUTION NEURAL NETWORK BASED DIAGNOSIS

In this paper[10] , It encourage the use of Convolution Neural Networks (CNN) as future scope, since CNN models can be used for classification of the affected skin images without the need for performing segmentation and feature extraction independently. Have made use of custom made automated segmentation and have used a novel approach for implementation of the CNN methodology, where CNN has been used for feature extraction and ANN was used to classify those extracted features. The advantage of this proposed system is that CNN does not require any additional classifier like SVM, KNN since 3 fully-connected layers were used for training the classification model. This type of classification brings its own unique benefits, like it is possible to apply back-propagation algorithm, which adjusts the parameters of neurons in all layers to obtain better classification model.

In this paper , Aya Abu Ali and Hasan Al-Marzouqi have employed transfer learning by using and modifying the CNN architecture in the LightNetpretrained model to achieve higher accuracy. They classified the images without applying lesion segmentation or without using complex image pre-processing techniques. The images were pre-processed and resized to create uniformity in the sizes of all the images. Each block of the 5 blocks had a convolution, relu, pooling and a dropout layer except for the last block where the Last block has fully connected layer and softmax layer. They calculated the error by comparing the predicted output with the actual output.

The advantage of the proposed model presented in this paper is that it is simpler as it does not use data augmentation and uses lesser no of parameters which is crucial in mobile applications where constraints on the size of the network and energy consumption affect the utility of developed tools. This model attains comparable results with fewer parameters. However, improved results can be obtained by using better feature extraction techniques like the ABCD rule and better image pre-processing. The proposed system in addresses 2 parts: lesion segmentation and lesion classification. The proposed solution makes use of a fully convolution-deconvolution network to segment the skin tumour from the surrounding skin. In order to address lesion classification, 2 solutions have been proposed. The first trains the dataset from scratch and second makes use of pre-trained VGG-16. They found that the results obtained from VGG-16 model were better than those obtained from the simple convNet architecture

CHAPTER 4

SYSTEM REQUIRMENTS SPECIFICATION

4.1 HARDWARE REQUIRMENT

- 4gb ram
- 320gb HDD or SDD
- Mobile or web cam
- NVIDIA CUDA 4.0
- Virtualization support

4.2 SOFTWARE REQUIRMENTS

- Tensor Flow
- Docker
- Web browser
- Windows 10
- Google cloud

CHAPTER 5
SYSTEM ARCHITECTURE

5.1 OBJECTIVE

This work develops a study in the Deep Learning field about the impact and tradeoffs of removing skin image background by applying a semantic segmentation for a subsequent classification of the disease. By isolating and comparing results from both unaltered and segmented skin lesion images, this work aims to better understand the impact on performance results when segmenting an image. Specifically, it hopes to better understand whether the values outside the lesion are detrimental to lesion classification, or are instead beneficial to lesion classification by providing contextual information relevant to each lesion. In order to achieve the project goal, two successful and well-known Convolutional Neural Networks architectures in the image semantic segmentation and image classification tasks have been adopted. The order of execution will be:

- Skin lesion segmentation. The first task will perform an automatic prediction of lesion segmentation from dermoscopic images taking the form of binary masks.
- Skin lesion classification.

The second task will perform the automatic classification as either melanoma or non-melanoma. In order to find the best classification model, this task will be divided into three subtasks according to different type of input skin images:

- Unaltered lesion classification. The basic model will perform the classification over the original skin RGB images contained in the ISIC dataset.
- Perfectly segmented lesion classification. The perfectly segmented image will be generated by performing a bit-wise and operation on the original images and its corresponding original binary mask contained in the ISIC dataset. The second model will perform the classification over the perfectly segmented images.
- Segmented lesion classification. The automatically segmented image will be generated by performing a bit-wise and operation on the original images and its corresponding binary mask generated automatically during the first task 1. The third, and most complex model will perform the classification over the automatically segmented images. A summary representation of input images to be classified is shown in Figure 5.1. A properly evaluation under the same conditions will be performed over the three classification models and will confirm or refuse the hypothesis of this work.

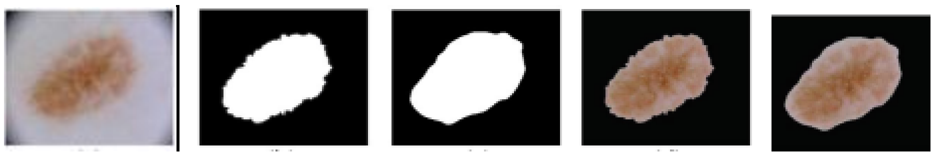


Fig 5.1 Skin cancer lesions

5.2 DATASETS

The ISIC 2016 Challenge dataset for Skin Lesion Analysis towards melanoma detection was used for this work. The dataset is publicly available and contains 12279 RGB images that are pre-partitioned into 900 training images and 379 validation images. All images are labeled as either benign or malignant and include a binary image mask to label the lesion within the image.

In order to solve the different problems addressed by this project, four variations of the ISIC dataset were used:

- Segmentation dataset. The original ISIC dataset split into training and validation subsets (note that a division between classes is not needed for the segmentation problem).
- Unaltered lesion classification dataset. The original ISIC dataset excluding the binary masks.
- Perfectly segmented lesion classification. This dataset contains the perfectly segmented images.
- Automatically segmented lesion classification dataset. This dataset contains the automatically segmented images.

5.3 NEURAL NETWORK ARCHITECTURE

Nowadays, there are several convolutional neural networks architectures that achieved successful results on benchmark challenges such as the Imagenet Large Scale Visual Recognition Challenge, and as a consequence became very popular in the deep learning community. In particular, the U-Net was in charge of the semantic segmentation task, while the VGG-16 was chosen for the classification task among popular architectures such as the AlexNet and the GoogLeNet.

5.3.1 U-NET

The U-Net is a ConvNet architecture specifically designed by O. Ronneberger et al. from the University of Freiburg to solve Biomedical Image Segmentation problems. It was successfully rated for winning the cell tracking challenge 2015. The network architecture is illustrated in Figure. As explained in, the network merges a convolutional network architecture (contracting path on the left side) with a deconvolutional architecture (expansive path on the right side) to obtain the semantic segmentation. The convolutional network is composed of a repetitive pattern of two 3×3 convolutions operations, followed by a ReLU layer and a downsampling process through a 2×2 maxpooling operation with stride2. On the other hand, the deconvolutional architecture includes a upsampling operation of the feature map obtained during the contracting path, followed by a 2×2 deconvolution that fractions the feature map channels into 2. A posteriori concatenation of the resulting feature map and the obtained during the contracting path is needed, followed by a 3×3 convolutions and a ReLU layer. The entire network is 23 convolutional layers deep, where the last layer is used to map each component feature vector related to the number of classes

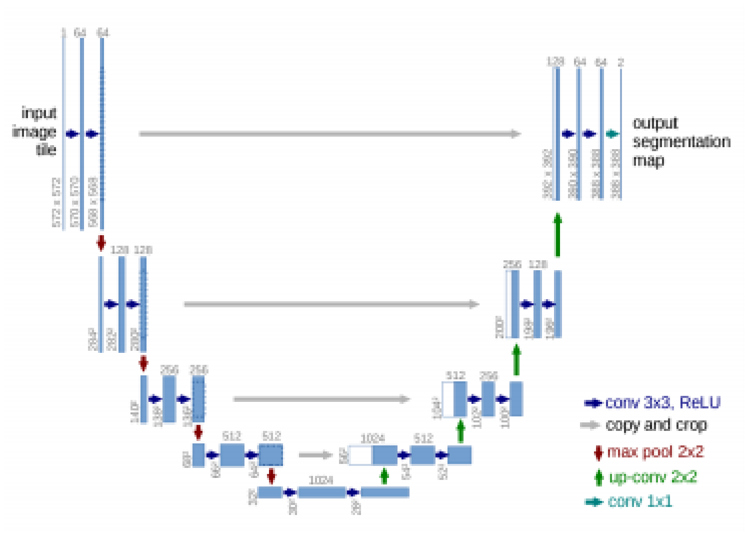


Fig 5.2 lesion segmentation

The architecture accepts RGB input images and their corresponding binary masks. The training is done by learning weights with the stochastic gradient descent implemented in Keras.

5.3.2 VGG-16

VGGNet is a very well-documented and commonly used architecture for convolutional neural networks. This ConvNet became popular by achieving excellent performance on the ImageNet dataset. It comes in several variations of which the two best-performing (with 16 and 19 weight layers) have been made publicly available. In this work, the VGG16 architecture was selected, since it has been shown to generalize well to other datasets. The input layer of the network expects a 224×224 pixel RGB image. The input image is passed through five convolutional blocks. Small convolutional filters with a receptive field of 3×3 are used. Each convolutional block includes a 2D convolution layer operation (the number of filters changes between blocks). All hidden layers are equipped as the activation function layer (nonlinearity operation) and include spatial pooling through use of a max-pooling layer. The network is concluded with a classifier block consisting of three Fully-Connected (FC) layers. The original VGG16 must be modified to suit our needs, as follows

- The final fully-connected output layer performs a binary classification (benign-malignant), not 1000 classes.
- The activation function in the modified layer has been changed from Softmax to Sigmoidal.

5.4 PREPROCESSING

This project takes advantage of ConvNets properties regarding input preprocessing few previous processing techniques are needed. Although some basic preprocessing forms are performed:

- Mean subtraction. In order to center the cloud of RGB values from input data around zero along every dimension of the image, a mean subtraction is applied across the image features.
- Image normalization. By dividing each RGB dimension of input images by its standard deviation, a normalization is obtained from its original 0 and 255 pixel values to 1 an 0 normalized values. This preprocessing technique will avoid further issues caused by poor contrast images.
- Image cropping & resizing. Input images are preprocessed to be accepted by the architecture though cropping the image to the same aspect ratio as needed and resizing the original image to 64×80 pixels for the U-Net and 224×224 pixels for the VGG-16.

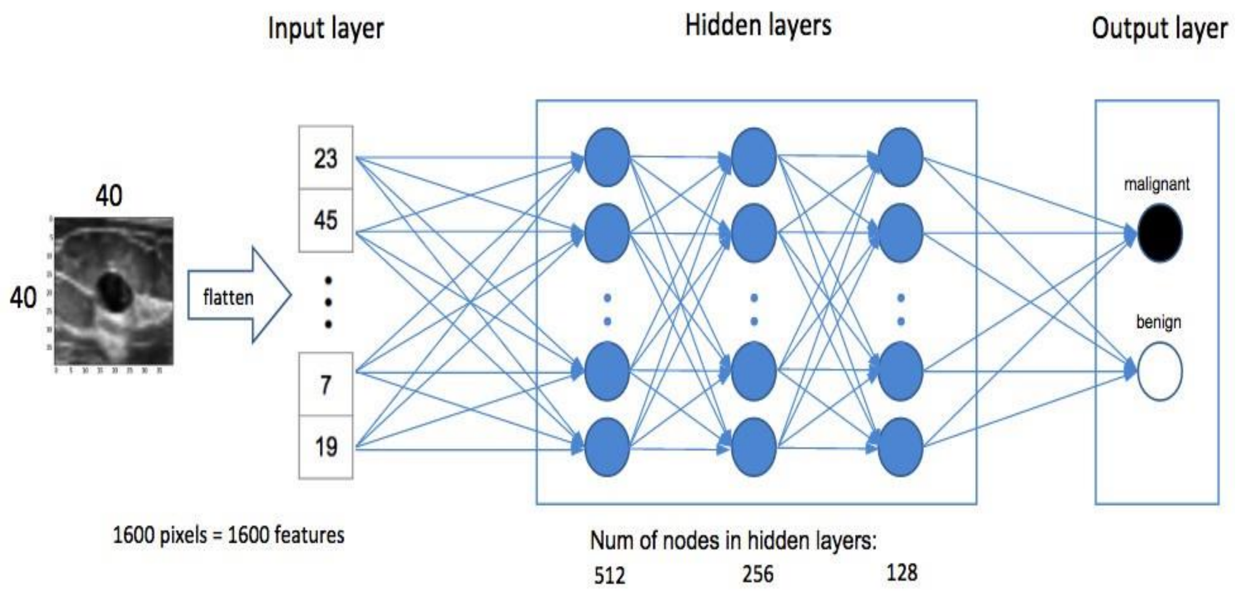


Fig 5.3 NN Layers

5.5 DATA AUGMENTATION

In order to make the most of our few training examples and increase the accuracy of the model, the ISIC dataset was augmented via a number of random transformations. The selected data augmentation techniques were: size re-scaling, rotations of 40 degrees, horizontal shift, image zooming, and horizontal flipping. Furthermore, it is expected that data augmentation should also help prevent overfitting (a common problem in machine learning related to small datasets, when the model, exposed to too few examples, learns patterns that do not generalize to new data) and, for this reason, improving the model's ability to generalize.

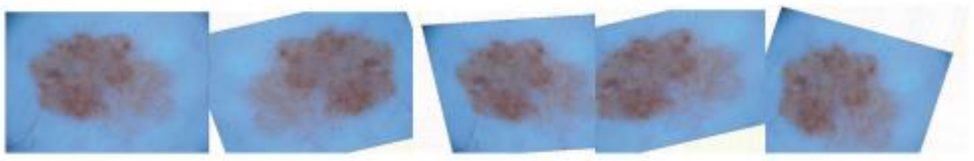


Fig 5.4 Skin lesions

5.6 TRAINING METHODS

The U-NET and VGG training methodology is proposed to face the two tasks of this project. The only difference between the segmentation and classification tasks are the models used. As previously commented in Section, the U-Net architecture is the architecture proposed for the segmentation task, while the VGG-16 is associated with a classification model. Following the diagram of Figure the training data is trained through the learning algorithm defined by each model, which applies the stochastic gradient descent . Once the model has learned the weights, a prediction algorithm classifies the validation data according to the training. A final model evaluation is performed by comparing the predictions with the ground truth data.

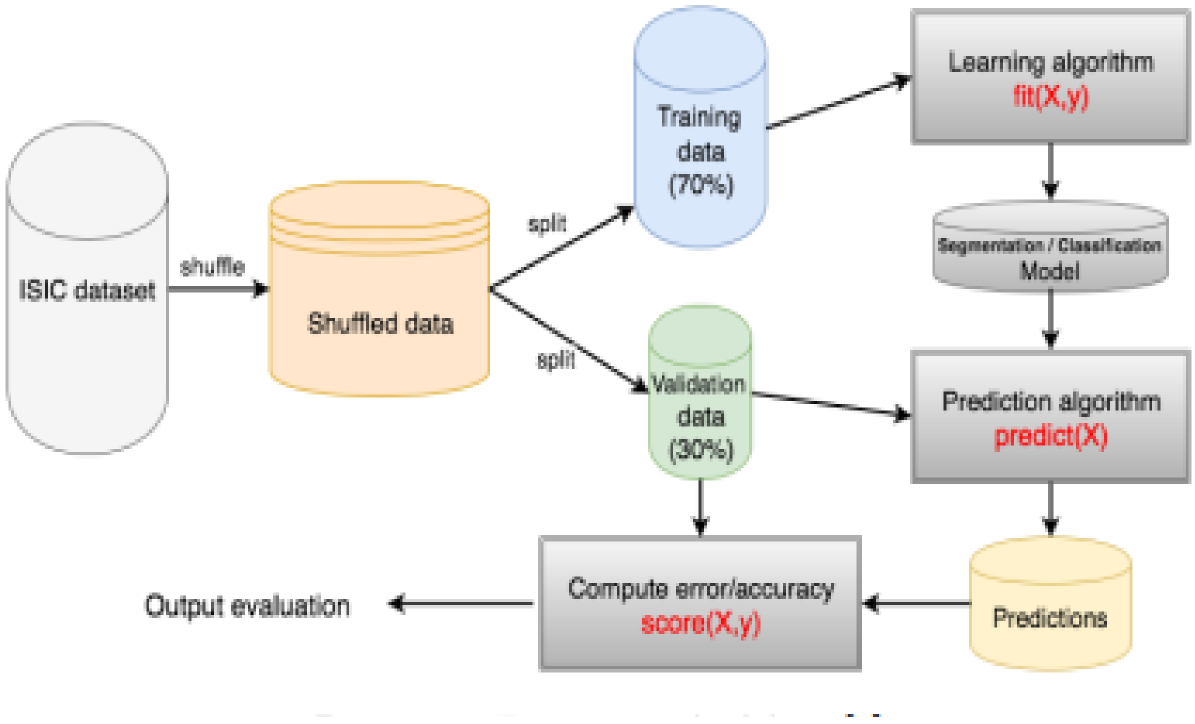


Fig 5.5 Training methodology

5.6.1 DEFINITION OF HYPER-PARAMETERS

During training, some parameters must be considered to be altered in order to get the best performance of the network proposed regarding the problem to be solved. Typical ConvNets parameters are the following:

- Batch size. The batch size is attributed to the number of training images in one forward or backward pass. It is important to highlight that the higher the batch size, the more memory will be needed.
- Iterations. The number of iterations is the number forward or backward of passes: each pass using a batch size number of images.
- Epoch. The number of epochs measures how many times every image has been seen during training (i.e. one epoch means that every image has been seen once). It can be

also understood as a one forward pass and one backward pass of all the training example.

- Loss function. Loss function (also called cost function) evaluates the penalty between the prediction and the ground truth label in every batch.
- Learning rate. The learning rate parameter defines the step size for which the weights of a model are updated regarding the stochastic gradient descent.
- Decay. The weight decay is an additional weight update parameter that induces the weights to exponentially decay to zero once the update process is over.
- Optimizer. Keras framework provides optimizers in order to find the most optimal set of hyperparameters for the model. Some optimizer examples are the SGD, RMSprop and Adam.

5.6.2 SEGMENTATION

The overview of the segmentation task follows the main structure described in Fig5.2. Training and Validation data from the Segmentation dataset is processed with a Python script in order to load the images and masks and convert them into NumPy binary format files (.npy) that will allow a faster loading during the learning and prediction algorithm. Data is previously organized into train, train masks and test folders. The proposed model is the U-Net network trained from scratch, which means that weights are randomly initialized and optimized by backpropagation. The network output is a Numpy array containing the 379 test binary masks, which can be converted into JPG images of 64×80 pixels dimension. This means a posterior post-processing technique will be needed to enlarge the images.

5.6.3 CLASSIFICATION

The main issue of the classification task is to avoiding overfitting caused by the small number of images of skin lesion in most dermatology datasets. In order to solve this problem, the objective of the proposed model is to firstly extract features from images with the original VGG-16 and secondly load those extracted representations on a fine-tuned VGG-16 architecture. Due to the reduced size of the ISIC dataset, the suggested approach initializes the model with weights from the VGG-16 trained on a larger dataset (such as ImageNet), a process known as transfer learning.

The underlying assumption behind transfer learning is that the pre-trained model has already learned features that might be useful for the classification task at hand. This corresponds, in practice, to using selected layer(s) of the pre-trained ConvNet as a fixed feature 29 extractor, which can be achieved by freezing all the convolutional blocks and only training the fully connected layers with the new dataset. Another common transfer learning technique consists of not only retraining the classifier on the top of the network with the new dataset, but also applying a fine-tuning of the network by training only the higher-level portion of the convolutional layers and continuing the backpropagation. In this work, it was proposed to freeze the lower level layers of the network because they contain more generic features of the dataset. The interest in training only the top layers of the network comes from the ability to perform extraction of more specific features. For this reason, the first four convolutional layers in the original VGG-16 architecture are initialized with weights from the

ImageNet dataset. The fifth, and final, convolutional block is initialized with weights saved during the features extraction.

Due to the reduced nature of the ISIC dataset, it was observed that the VGG presented overfitting caused by the few number of images. In order to prevent it, the VGG-16 was chosen in front of the upgraded VGG-19 architecture by reducing the network capacity. Note that the number of parameters to be computed is reduced from 144M parameters (VGG-19) to 138M parameters (VGG-16). This overfitting issue can be also cut down by adding dropout, a network regularization technique that reduces co-adaptation between neurons by increasing iterations for convergence. The last prevention that was applied is data augmentation, previously commented. Since the nature of the ISIC dataset is not balanced (approximately 80% corresponds to the benign class, while the rest is referred to the malignant), the network has to be sure that the same number of images for each class are being trained. Instead of manually balancing the dataset through downsampling, that it would further reduce the small dimension of the datset, a weighted loss function equation was proposed in favor of balancing the minor class.

$$H(p, q) = X\ x\ \alpha(x)\ p(x)\log(q)$$

Major class 1 otherwise where p and q are the probability distributions of the ground truth the ρ value weights the loss regarding the major class instances during training. ρ is defined as 1−frequency apparence(minor class). Finally, the process described in this section were repeated for each classification task considered in this project:

(1) Unaltered Classification dataset

(2) Perfectly Segmented Classification dataset

(3) Automatically Segmented Classification dataset.

CHAPTER 6

IMPLEMENTATION AND CODING

6.1 INSTALLING DOCKER

1. Double-click **Docker Desktop for Windows Installer.exe** to run the installer.

   If you haven't already downloaded the installer (Docker Desktop Installer.exe), you can get it from **download.docker.com**. It typically downloads to your Downloads folder, or you can run it from the recent downloads bar at the bottom of your web browser.

2. Follow the install wizard to accept the license, authorize the installer, and proceed with the install.

   You are asked to authorize Docker.app with your system password during the install process. Privileged access is needed to install networking components, links to the Docker apps, and manage the Hyper-V VMs.

3. Click **Finish** on the setup complete dialog to launch Docker.

6.2 INSTALLING TENSORFLOW

6.2.1 DOWNLOAD TENSORFLOW PACKAGE

```
# Current stable release for CPU-only
pip install tensorflow

# Preview nightly build for CPU-only (unstable)
pip install tf-nightly

# Install TensorFlow 2.0 Alpha
pip install tensorflow==2.0.0-alpha0
```

Fig 6.1 Tensorflow installation

6.2.2 RUN TENSORFLOW CONTAINER

```
docker pull tensorflow/tensorflow                    # Download latest image
docker run -it -p 8888:8888 tensorflow/tensorflow   # Start a Jupyter notebook server
```

Fig 6.2 Tensorflow container

6.2.3 PULLING TENSORFLOW DATA ON DOCKER

```
docker pull tensorflow/tensorflow                     # 
latest stable release
docker pull tensorflow/tensorflow:devel-gpu          # 
nightly dev release w/ GPU support
docker pull tensorflow/tensorflow:latest-gpu-
jupyter  # latest release w/ GPU support and Jupyter
```

Fig 6.3 Pulling data on docker

6.2.4 STARTING TENSORFLOW DOCKER CONTAINER

```
docker run -it --rm tensorflow/tensorflow \
   python -c "import tensorflow as tf; 
tf.enable_eager_execution(); 
print(tf.reduce_sum(tf.random_normal([1000, 1000])))"
```

Fig 6.4 Starting Docker Container

6.3 RETRAIN.PY

```
# pylint: enable=line-too-long

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import collections
from datetime import datetime
import hashlib
import os.path
import random
import re
import sys

import numpy as np
import tensorflow as tf
import tensorflow_hub as hub

FLAGS = None

MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1  # ~134M

# The location where variable checkpoints will be stored.
CHECKPOINT_NAME = '/tmp/_retrain_checkpoint'

# A module is understood as instrumented for quantization with TF-Lite
# if it contains any of these ops.
FAKE_QUANT_OPS = ('FakeQuantWithMinMaxVars',
                  'FakeQuantWithMinMaxVarsPerChannel')


def create_image_lists(image_dir, testing_percentage, 
validation_percentage):

  if not tf.gfile.Exists(image_dir):
    tf.logging.error("Image directory '" + image_dir + "' not found.")
    return None
  result = collections.OrderedDict()
  sub_dirs = sorted(x[0] for x in tf.gfile.Walk(image_dir))
  # The root directory comes first, so skip it.
  is_root_dir = True
  for sub_dir in sub_dirs:
    if is_root_dir:
      is_root_dir = False
      continue
    extensions = sorted(set(os.path.normcase(ext)  # Smash case on Windows.
                            for ext in ['JPEG', 'JPG', 'jpeg', 'jpg',
'png']))
    file_list = []
    dir_name = os.path.basename(sub_dir)
    if dir_name == image_dir:
      continue
    tf.logging.info("Looking for images in '" + dir_name + "'")
    for extension in extensions:
      file_glob = os.path.join(image_dir, dir_name, '*.' + extension)
      file_list.extend(tf.gfile.Glob(file_glob))
    if not file_list:
      tf.logging.warning('No files found')
      continue
    if len(file_list) < 20:
      tf.logging.warning(
          'WARNING: Folder has less than 20 images, which may cause
issues.')
    elif len(file_list) > MAX_NUM_IMAGES_PER_CLASS:
      tf.logging.warning(
          'WARNING: Folder {} has more than {} images. Some images will '
          'never be selected.'.format(dir_name, MAX_NUM_IMAGES_PER_CLASS))
    label_name = re.sub(r'[^a-z0-9]+', ' ', dir_name.lower())
    training_images = []
    testing_images = []
    validation_images = []
    for file_name in file_list:
      base_name = os.path.basename(file_name)
      # We want to ignore anything after '_nohash_' in the file name when
      # deciding which set to put an image in, the data set creator has a
way of
      # grouping photos that are close variations of each other. For
example
      # this is used in the plant disease data set to group multiple
pictures of
      # the same leaf.
      hash_name = re.sub(r'_nohash_.*$', '', file_name)
      # This looks a bit magical, but we need to decide whether this file
should
      # go into the training, testing, or validation sets, and we want to
keep
      # existing files in the same set even if more files are subsequently
      # added.
      # To do that, we need a stable way of deciding based on just the file
name
      # itself, so we do a hash of that and then use that to generate a
      # probability value that we use to assign it.
      hash_name_hashed =
hashlib.sha1(tf.compat.as_bytes(hash_name)).hexdigest()
      percentage_hash = ((int(hash_name_hashed, 16) %
                          (MAX_NUM_IMAGES_PER_CLASS + 1)) *
```

```
                        (100.0 / MAX_NUM_IMAGES_PER_CLASS))
      if percentage_hash < validation_percentage:
        validation_images.append(base_name)
      elif percentage_hash < (testing_percentage + validation_percentage):
        testing_images.append(base_name)
      else:
        training_images.append(base_name)
    result[label_name] = {
        'dir': dir_name,
        'training': training_images,
        'testing': testing_images,
        'validation': validation_images,
    }
  return result


def get_image_path(image_lists, label_name, index, image_dir, category):

  if label_name not in image_lists:
    tf.logging.fatal('Label does not exist %s.', label_name)
  label_lists = image_lists[label_name]
  if category not in label_lists:
    tf.logging.fatal('Category does not exist %s.', category)
  category_list = label_lists[category]
  if not category_list:
    tf.logging.fatal('Label %s has no images in the category %s.',
                     label_name, category)
  mod_index = index % len(category_list)
  base_name = category_list[mod_index]
  sub_dir = label_lists['dir']
  full_path = os.path.join(image_dir, sub_dir, base_name)
  return full_path


def get_bottleneck_path(image_lists, label_name, index, bottleneck_dir,
                        category, module_name):

  module_name = (module_name.replace('://', '~')  # URL scheme.
                 .replace('/', '~')  # URL and Unix paths.
                 .replace(':', '~').replace('\\', '~'))  # Windows paths.
  return get_image_path(image_lists, label_name, index, bottleneck_dir,
                        category) + '_' + module_name + '.txt'


def create_module_graph(module_spec):
  """Creates a graph and loads Hub Module into it.

  Args:
    module_spec: the hub.ModuleSpec for the image module being used.

  Returns:
    graph: the tf.Graph that was created.
    bottleneck_tensor: the bottleneck values output by the module.
    resized_input_tensor: the input images, resized as expected by the
module.
    wants_quantization: a boolean, whether the module has been instrumented
      with fake quantization ops.
  """
  height, width = hub.get_expected_image_size(module_spec)
  with tf.Graph().as_default() as graph:
    resized_input_tensor = tf.placeholder(tf.float32, [None, height, width,
3])
    m = hub.Module(module_spec)
    bottleneck_tensor = m(resized_input_tensor)
    wants_quantization = any(node.op in FAKE_QUANT_OPS
                             for node in graph.as_graph_def().node)
  return graph, bottleneck_tensor, resized_input_tensor, wants_quantization


def run_bottleneck_on_image(sess, image_data, image_data_tensor,
                            decoded_image_tensor, resized_input_tensor,
                            bottleneck_tensor):
  """Runs inference on an image to extract the 'bottleneck' summary layer.

  Args:
    sess: Current active TensorFlow Session.
    image_data: String of raw JPEG data.
    image_data_tensor: Input data layer in the graph.
    decoded_image_tensor: Output of initial image resizing and
preprocessing.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: Layer before the final softmax.

  Returns:
    Numpy array of bottleneck values.
  """
  # First decode the JPEG image, resize it, and rescale the pixel values.
  resized_input_values = sess.run(decoded_image_tensor,
                                  {image_data_tensor: image_data})
  # Then run it through the recognition network.
  bottleneck_values = sess.run(bottleneck_tensor,
                               {resized_input_tensor:
resized_input_values})
  bottleneck_values = np.squeeze(bottleneck_values)
  return bottleneck_values


def ensure_dir_exists(dir_name):
  """Makes sure the folder exists on disk.

  Args:
    dir_name: Path string to the folder we want to create.
  """
  if not os.path.exists(dir_name):
    os.makedirs(dir_name)


def create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           decoded_image_tensor, resized_input_tensor,
                           bottleneck_tensor):
  """Create a single bottleneck file."""
  tf.logging.info('Creating bottleneck at ' + bottleneck_path)
  image_path = get_image_path(image_lists, label_name, index,
                              image_dir, category)
  if not tf.gfile.Exists(image_path):
    tf.logging.fatal('File does not exist %s', image_path)
  image_data = tf.gfile.FastGFile(image_path, 'rb').read()
  try:
    bottleneck_values = run_bottleneck_on_image(
        sess, image_data, jpeg_data_tensor, decoded_image_tensor,
        resized_input_tensor, bottleneck_tensor)
  except Exception as e:
    raise RuntimeError('Error during processing file %s (%s)' %
(image_path,
                                                                 str(e)))
  bottleneck_string = ','.join(str(x) for x in bottleneck_values)
  with open(bottleneck_path, 'w') as bottleneck_file:
    bottleneck_file.write(bottleneck_string)


def get_or_create_bottleneck(sess, image_lists, label_name, index,
image_dir,
                             category, bottleneck_dir, jpeg_data_tensor,
                             decoded_image_tensor, resized_input_tensor,
                             bottleneck_tensor, module_name):
  """Retrieves or calculates bottleneck values for an image.

  If a cached version of the bottleneck data exists on-disk, return that,
  otherwise calculate the data and save it to disk for future use.

  Args:
    sess: The current active TensorFlow Session.
    image_lists: OrderedDict of training images for each label.
    label_name: Label string we want to get an image for.
    index: Integer offset of the image we want. This will be modulo-ed by
the
    available number of images for the label, so it can be arbitrarily
large.
    image_dir: Root folder string of the subfolders containing the training
    images.
    category: Name string of which set to pull images from - training,
testing,
    or validation.
    bottleneck_dir: Folder string holding cached files of bottleneck
values.
    jpeg_data_tensor: The tensor to feed loaded jpeg data into.
    decoded_image_tensor: The output of decoding and resizing the image.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The output tensor for the bottleneck values.
    module_name: The name of the image module being used.

  Returns:
    Numpy array of values produced by the bottleneck layer for the image.
  """
  label_lists = image_lists[label_name]
  sub_dir = label_lists['dir']
  sub_dir_path = os.path.join(bottleneck_dir, sub_dir)
  ensure_dir_exists(sub_dir_path)
  bottleneck_path = get_bottleneck_path(image_lists, label_name, index,
                                        bottleneck_dir, category,
module_name)
  if not os.path.exists(bottleneck_path):
    create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           decoded_image_tensor, resized_input_tensor,
                           bottleneck_tensor)
  with open(bottleneck_path, 'r') as bottleneck_file:
    bottleneck_string = bottleneck_file.read()
  did_hit_error = False
  try:
    bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
  except ValueError:
    tf.logging.warning('Invalid float found, recreating bottleneck')
    did_hit_error = True
  if did_hit_error:
    create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           decoded_image_tensor, resized_input_tensor,
                           bottleneck_tensor)
    with open(bottleneck_path, 'r') as bottleneck_file:
      bottleneck_string = bottleneck_file.read()
    # Allow exceptions to propagate here, since they shouldn't happen after
a
    # fresh creation
    bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
  return bottleneck_values


def cache_bottlenecks(sess, image_lists, image_dir, bottleneck_dir,
                      jpeg_data_tensor, decoded_image_tensor,
                      resized_input_tensor, bottleneck_tensor,
module_name):
  """Ensures all the training, testing, and validation bottlenecks are
cached.

  Because we're likely to read the same image multiple times (if there are
no
  distortions applied during training) it can speed things up a lot if we
  calculate the bottleneck layer values once for each image during
  preprocessing, and then just read those cached values repeatedly during
  training. Here we go through all the images we've found, calculate those
  values, and save them off.

  Args:
    sess: The current active TensorFlow Session.
    image_lists: OrderedDict of training images for each label.
    image_dir: Root folder string of the subfolders containing the training
    images.
    bottleneck_dir: Folder string holding cached files of bottleneck
values.
    jpeg_data_tensor: Input tensor for jpeg data from file.
    decoded_image_tensor: The output of decoding and resizing the image.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The penultimate output layer of the graph.
    module_name: The name of the image module being used.

  Returns:
    Nothing.
  """
  how_many_bottlenecks = 0
  ensure_dir_exists(bottleneck_dir)
  for label_name, label_lists in image_lists.items():
    for category in ['training', 'testing', 'validation']:
      category_list = label_lists[category]
      for index, unused_base_name in enumerate(category_list):
        get_or_create_bottleneck(
            sess, image_lists, label_name, index, image_dir, category,
            bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
            resized_input_tensor, bottleneck_tensor, module_name)

        how_many_bottlenecks += 1
        if how_many_bottlenecks % 100 == 0:
```

```
        tf.logging.info(
            str(how_many_bottlenecks) + ' bottleneck files created.')


def get_random_cached_bottlenecks(sess, image_lists, how_many,
category,bottleneck_dir, image_dir, jpeg_data_tensor,decoded_image_tensor,
resized_input_tensor,bottleneck_tensor, module_name):

  class_count = len(image_lists.keys())
  bottlenecks = []
  ground_truths = []
  filenames = []
  if how_many >= 0:
    # Retrieve a random sample of bottlenecks.
    for unused_i in range(how_many):
      label_index = random.randrange(class_count)
      label_name = list(image_lists.keys())[label_index]
      image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
      image_name = get_image_path(image_lists, label_name, image_index,
                                  image_dir, category)
      bottleneck = get_or_create_bottleneck(
          sess, image_lists, label_name, image_index, image_dir, category,
          bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
          resized_input_tensor, bottleneck_tensor, module_name)
      bottlenecks.append(bottleneck)
      ground_truths.append(label_index)
      filenames.append(image_name)
  else:
    # Retrieve all bottlenecks.
    for label_index, label_name in enumerate(image_lists.keys()):
      for image_index, image_name in enumerate(
          image_lists[label_name][category]):
        image_name = get_image_path(image_lists, label_name, image_index,
                                    image_dir, category)
        bottleneck = get_or_create_bottleneck(
            sess, image_lists, label_name, image_index, image_dir,
category,
            bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
            resized_input_tensor, bottleneck_tensor, module_name)
        bottlenecks.append(bottleneck)
        ground_truths.append(label_index)
        filenames.append(image_name)
  return bottlenecks, ground_truths, filenames


def get_random_distorted_bottlenecks(
    sess, image_lists, how_many, category, image_dir, input_jpeg_tensor,
    distorted_image, resized_input_tensor, bottleneck_tensor):
  class_count = len(image_lists.keys())
  bottlenecks = []
  ground_truths = []
  for unused_i in range(how_many):
    label_index = random.randrange(class_count)
    label_name = list(image_lists.keys())[label_index]
    image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
    image_path = get_image_path(image_lists, label_name, image_index,
image_dir,
                                category)
    if not tf.gfile.Exists(image_path):
      tf.logging.fatal('File does not exist %s', image_path)
    jpeg_data = tf.gfile.FastGFile(image_path, 'rb').read()
    # Note that we materialize the distorted_image_data as a numpy array
before
    # sending running inference on the image. This involves 2 memory copies
and
    # might be optimized in other implementations.
    distorted_image_data = sess.run(distorted_image,
                                    {input_jpeg_tensor: jpeg_data})
    bottleneck_values = sess.run(bottleneck_tensor,
                                 {resized_input_tensor:
distorted_image_data})
    bottleneck_values = np.squeeze(bottleneck_values)
    bottlenecks.append(bottleneck_values)
    ground_truths.append(label_index)
  return bottlenecks, ground_truths


def should_distort_images(flip_left_right, random_crop, random_scale,
                          random_brightness):
  #for image scaling and cropping
  input_height, input_width = hub.get_expected_image_size(module_spec)
  input_depth = hub.get_num_image_channels(module_spec)
  jpeg_data = tf.placeholder(tf.string, name='DistortJPGInput')
  decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_depth)
  # Convert from full range of uint8 to range [0,1] of float32.
  decoded_image_as_float = tf.image.convert_image_dtype(decoded_image,
                                                        tf.float32)
  decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
  margin_scale = 1.0 + (random_crop / 100.0)
  resize_scale = 1.0 + (random_scale / 100.0)
  margin_scale_value = tf.constant(margin_scale)
  resize_scale_value = tf.random_uniform(shape=[],
                                         minval=1.0,
                                         maxval=resize_scale)
  scale_value = tf.multiply(margin_scale_value, resize_scale_value)
  precrop_width = tf.multiply(scale_value, input_width)
  precrop_height = tf.multiply(scale_value, input_height)
  precrop_shape = tf.stack([precrop_height, precrop_width])
  precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32)
  precropped_image = tf.image.resize_bilinear(decoded_image_4d,
                                              precrop_shape_as_int)
  precropped_image_3d = tf.squeeze(precropped_image, axis=[0])
  cropped_image = tf.random_crop(precropped_image_3d,
                                 [input_height, input_width, input_depth])
  if flip_left_right:
    flipped_image = tf.image.random_flip_left_right(cropped_image)
  else:
    flipped_image = cropped_image
  brightness_min = 1.0 - (random_brightness / 100.0)
  brightness_max = 1.0 + (random_brightness / 100.0)
  brightness_value = tf.random_uniform(shape=[],
                                       minval=brightness_min,
                                       maxval=brightness_max)
  brightened_image = tf.multiply(flipped_image, brightness_value)
  distort_result = tf.expand_dims(brightened_image, 0,
name='DistortResult')
  return jpeg_data, distort_result


def variable_summaries(var):
  """Attach a lot of summaries to a Tensor (for TensorBoard
visualization)."""
```

```
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean', mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    tf.summary.scalar('stddev', stddev)
    tf.summary.scalar('max', tf.reduce_max(var))
    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)


def add_final_retrain_ops(class_count, final_tensor_name,
bottleneck_tensor,
                          quantize_layer, is_training):
  batch_size, bottleneck_tensor_size =
bottleneck_tensor.get_shape().as_list()
  assert batch_size is None, 'We want to work with arbitrary batch size.'
  with tf.name_scope('input'):
    bottleneck_input = tf.placeholder_with_default(
        bottleneck_tensor,
        shape=[batch_size, bottleneck_tensor_size],
        name='BottleneckInputPlaceholder')

    ground_truth_input = tf.placeholder(
        tf.int64, [batch_size], name='GroundTruthInput')

  # Organizing the following ops so they are easier to see in TensorBoard.
  layer_name = 'final_retrain_ops'
  with tf.name_scope(layer_name):
    with tf.name_scope('weights'):
      initial_value = tf.truncated_normal(
          [bottleneck_tensor_size, class_count], stddev=0.001)
      layer_weights = tf.Variable(initial_value, name='final_weights')
      variable_summaries(layer_weights)

    with tf.name_scope('biases'):
      layer_biases = tf.Variable(tf.zeros([class_count]),
name='final_biases')
      variable_summaries(layer_biases)

    with tf.name_scope('Wx_plus_b'):
      logits = tf.matmul(bottleneck_input, layer_weights) + layer_biases
      tf.summary.histogram('pre_activations', logits)

  final_tensor = tf.nn.softmax(logits, name=final_tensor_name)

  if quantize_layer:
    if is_training:
      tf.contrib.quantize.create_training_graph()
    else:
      tf.contrib.quantize.create_eval_graph()

  tf.summary.histogram('activations', final_tensor)

  # If this is an eval graph, we don't need to add loss ops or an
optimizer.
  if not is_training:
    return None, None, bottleneck_input, ground_truth_input, final_tensor

  with tf.name_scope('cross_entropy'):
    cross_entropy_mean = tf.losses.sparse_softmax_cross_entropy(
        labels=ground_truth_input, logits=logits)

  tf.summary.scalar('cross_entropy', cross_entropy_mean)

  with tf.name_scope('train'):
    optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate)
    train_step = optimizer.minimize(cross_entropy_mean)

  return (train_step, cross_entropy_mean, bottleneck_input,
ground_truth_input,
          final_tensor)


def add_evaluation_step(result_tensor, ground_truth_tensor):
  with tf.name_scope('accuracy'):
    with tf.name_scope('correct_prediction'):
      prediction = tf.argmax(result_tensor, 1)
      correct_prediction = tf.equal(prediction, ground_truth_tensor)
    with tf.name_scope('accuracy'):
      evaluation_step = tf.reduce_mean(tf.cast(correct_prediction,
tf.float32))
  tf.summary.scalar('accuracy', evaluation_step)
  return evaluation_step, prediction


def run_final_eval(train_session, module_spec, class_count, image_lists,
                   jpeg_data_tensor, decoded_image_tensor,
                   resized_image_tensor, bottleneck_tensor):
  test_bottlenecks, test_ground_truth, test_filenames = (
      get_random_cached_bottlenecks(train_session, image_lists,
                                    FLAGS.test_batch_size,
                                    'testing', FLAGS.bottleneck_dir,
                                    FLAGS.image_dir, jpeg_data_tensor,
                                    decoded_image_tensor,
resized_image_tensor,
                                    bottleneck_tensor, FLAGS.tfhub_module))

  (eval_session, _, bottleneck_input, ground_truth_input, evaluation_step,
   prediction) = build_eval_session(module_spec, class_count)
  test_accuracy, predictions = eval_session.run(
      [evaluation_step, prediction],
      feed_dict={
          bottleneck_input: test_bottlenecks,
          ground_truth_input: test_ground_truth
      })
  tf.logging.info('Final test accuracy = %.1f%% (N=%d)' %
                  (test_accuracy * 100, len(test_bottlenecks)))

  if FLAGS.print_misclassified_test_images:
    tf.logging.info('=== MISCLASSIFIED TEST IMAGES ===')
    for i, test_filename in enumerate(test_filenames):
      if predictions[i] != test_ground_truth[i]:
        tf.logging.info('%70s  %s' % (test_filename,

list(image_lists.keys())[predictions[i]]))


def build_eval_session(module_spec, class_count):
    # If quantized, we need to create the correct eval graph for exporting.
  eval_graph, bottleneck_tensor, resized_input_tensor, wants_quantization =
(
      create_module_graph(module_spec))

  eval_sess = tf.Session(graph=eval_graph)
  with eval_graph.as_default():
    # Add the new layer for exporting.
    (_, _, bottleneck_input,
     ground_truth_input, final_tensor) = add_final_retrain_ops(
         class_count, FLAGS.final_tensor_name, bottleneck_tensor,
         wants_quantization, is_training=False)

    # Now we need to restore the values from the training graph to the eval
    # graph.
    tf.train.Saver().restore(eval_sess, CHECKPOINT_NAME)

    evaluation_step, prediction = add_evaluation_step(final_tensor,
                                                      ground_truth_input)

  return (eval_sess, resized_input_tensor, bottleneck_input,
ground_truth_input,
          evaluation_step, prediction)


def save_graph_to_file(graph_file_name, module_spec, class_count):
  """Saves an graph to file, creating a valid quantized one if
necessary."""
  sess, _, _, _, _, _ = build_eval_session(module_spec, class_count)
  graph = sess.graph

  output_graph_def = tf.graph_util.convert_variables_to_constants(
      sess, graph.as_graph_def(), [FLAGS.final_tensor_name])

  with tf.gfile.FastGFile(graph_file_name, 'wb') as f:
    f.write(output_graph_def.SerializeToString())


def prepare_file_system():
  # Set up the directory we'll write summaries to for TensorBoard
  if tf.gfile.Exists(FLAGS.summaries_dir):
    tf.gfile.DeleteRecursively(FLAGS.summaries_dir)
  tf.gfile.MakeDirs(FLAGS.summaries_dir)
  if FLAGS.intermediate_store_frequency > 0:
    ensure_dir_exists(FLAGS.intermediate_output_graphs_dir)
  return


def add_jpeg_decoding(module_spec):
   input_height, input_width = hub.get_expected_image_size(module_spec)
  input_depth = hub.get_num_image_channels(module_spec)
  jpeg_data = tf.placeholder(tf.string, name='DecodeJPGInput')
  decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_depth)
  # Convert from full range of uint8 to range [0,1] of float32.
  decoded_image_as_float = tf.image.convert_image_dtype(decoded_image,
                                                        tf.float32)
  decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
  resize_shape = tf.stack([input_height, input_width])
  resize_shape_as_int = tf.cast(resize_shape, dtype=tf.int32)
  resized_image = tf.image.resize_bilinear(decoded_image_4d,
                                           resize_shape_as_int)
  return jpeg_data, resized_image
```

```
def export_model(module_spec, class_count, saved_model_dir):
  """Exports model for serving.

  Args:
    module_spec: The hub.ModuleSpec for the image module being used.
    class_count: The number of classes.
    saved_model_dir: Directory in which to save exported model and
variables.
  """
  # The SavedModel should hold the eval graph.
  sess, in_image, _, _, _, _ = build_eval_session(module_spec, class_count)
  with sess.graph.as_default() as graph:
    tf.saved_model.simple_save(
        sess,
        saved_model_dir,
        inputs={'image': in_image},
        outputs={'prediction': graph.get_tensor_by_name('final_result:0')},
        legacy_init_op=tf.group(tf.tables_initializer(),
name='legacy_init_op')
    )


def main(_):
  # Needed to make sure the logging output is visible.
  # See https://github.com/tensorflow/tensorflow/issues/3047
  tf.logging.set_verbosity(tf.logging.INFO)

  if not FLAGS.image_dir:
    tf.logging.error('Must set flag --image_dir.')
    return -1

  # Prepare necessary directories that can be used during training
  prepare_file_system()

  # Look at the folder structure, and create lists of all the images.
  image_lists = create_image_lists(FLAGS.image_dir,
FLAGS.testing_percentage,
                                   FLAGS.validation_percentage)
  class_count = len(image_lists.keys())
  if class_count == 0:
    tf.logging.error('No valid folders of images found at ' +
FLAGS.image_dir)
    return -1
  if class_count == 1:
    tf.logging.error('Only one valid folder of images found at ' +
                     FLAGS.image_dir +
                     ' - multiple classes are needed for classification.')
    return -1

  # See if the command-line flags mean we're applying any distortions.
  do_distort_images = should_distort_images(
      FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale,
      FLAGS.random_brightness)

  # Set up the pre-trained graph.
  module_spec = hub.load_module_spec(FLAGS.tfhub_module)
  graph, bottleneck_tensor, resized_image_tensor, wants_quantization = (
      create_module_graph(module_spec))

  # Add the new layer that we'll be training.
  with graph.as_default():
    (train_step, cross_entropy, bottleneck_input,
     ground_truth_input, final_tensor) = add_final_retrain_ops(
         class_count, FLAGS.final_tensor_name, bottleneck_tensor,
         wants_quantization, is_training=True)

  with tf.Session(graph=graph) as sess:
    # Initialize all weights: for the module to their pretrained values,
    # and for the newly added retraining layer to random initial values.
    init = tf.global_variables_initializer()
    sess.run(init)

    # Set up the image decoding sub-graph.
    jpeg_data_tensor, decoded_image_tensor = add_jpeg_decoding(module_spec)

    if do_distort_images:
      # We will be applying distortions, so set up the operations we'll
need.
      (distorted_jpeg_data_tensor,
       distorted_image_tensor) = add_input_distortions(
           FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale,
           FLAGS.random_brightness, module_spec)
    else:
      # We'll make sure we've calculated the 'bottleneck' image summaries
and
      # cached them on disk.
      cache_bottlenecks(sess, image_lists, FLAGS.image_dir,
                        FLAGS.bottleneck_dir, jpeg_data_tensor,
                        decoded_image_tensor, resized_image_tensor,
                        bottleneck_tensor, FLAGS.tfhub_module)

    # Create the operations we need to evaluate the accuracy of our new
layer.
    evaluation_step, _ = add_evaluation_step(final_tensor,
ground_truth_input)

    # Merge all the summaries and write them out to the summaries_dir
    merged = tf.summary.merge_all()
    train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train',
                                         sess.graph)

    validation_writer = tf.summary.FileWriter(
        FLAGS.summaries_dir + '/validation')

    # Create a train saver that is used to restore values into an eval
graph
    # when exporting models.
    train_saver = tf.train.Saver()

    # Run the training for as many cycles as requested on the command line.
    for i in range(FLAGS.how_many_training_steps):
      # Get a batch of input bottleneck values, either calculated fresh
every
      # time with distortions applied, or from the cache stored on disk.
      if do_distort_images:
        (train_bottlenecks,
         train_ground_truth) = get_random_distorted_bottlenecks(
             sess, image_lists, FLAGS.train_batch_size, 'training',
             FLAGS.image_dir, distorted_jpeg_data_tensor,
             distorted_image_tensor, resized_image_tensor,
bottleneck_tensor)
      else:
        (train_bottlenecks,
         train_ground_truth, _) = get_random_cached_bottlenecks(
             sess, image_lists, FLAGS.train_batch_size, 'training',
             FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
             decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
             FLAGS.tfhub_module)
      # Feed the bottlenecks and ground truth into the graph, and run a
training
      # step. Capture training summaries for TensorBoard with the `merged`
op.
      train_summary, _ = sess.run(
          [merged, train_step],
          feed_dict={bottleneck_input: train_bottlenecks,
                     ground_truth_input: train_ground_truth})
      train_writer.add_summary(train_summary, i)

      # Every so often, print out how well the graph is training.
      is_last_step = (i + 1 == FLAGS.how_many_training_steps)
      if (i % FLAGS.eval_step_interval) == 0 or is_last_step:
        train_accuracy, cross_entropy_value = sess.run(
            [evaluation_step, cross_entropy],
            feed_dict={bottleneck_input: train_bottlenecks,
                       ground_truth_input: train_ground_truth})
        tf.logging.info('%s: Step %d: Train accuracy = %.1f%%' %
                        (datetime.now(), i, train_accuracy * 100))
        tf.logging.info('%s: Step %d: Cross entropy = %f' %
                        (datetime.now(), i, cross_entropy_value))
        # TODO: Make this use an eval graph, to avoid quantization
        # moving averages being updated by the validation set, though in
        # practice this makes a negligable difference.
        validation_bottlenecks, validation_ground_truth, _ = (
            get_random_cached_bottlenecks(
                sess, image_lists, FLAGS.validation_batch_size,
'validation',
                FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
                decoded_image_tensor, resized_image_tensor,
bottleneck_tensor,
                FLAGS.tfhub_module))
        # Run a validation step and capture training summaries for
TensorBoard
        # with the `merged` op.
        validation_summary, validation_accuracy = sess.run(
            [merged, evaluation_step],
            feed_dict={bottleneck_input: validation_bottlenecks,
                       ground_truth_input: validation_ground_truth})
        validation_writer.add_summary(validation_summary, i)
        tf.logging.info('%s: Step %d: Validation accuracy = %.1f%% (N=%d)'
%
                        (datetime.now(), i, validation_accuracy * 100,
                         len(validation_bottlenecks)))

      # Store intermediate results
      intermediate_frequency = FLAGS.intermediate_store_frequency

      if (intermediate_frequency > 0 and (i % intermediate_frequency == 0)
          and i > 0):
        # If we want to do an intermediate save, save a checkpoint of the
train
        # graph, to restore into the eval graph.
        train_saver.save(sess, CHECKPOINT_NAME)
        intermediate_file_name = (FLAGS.intermediate_output_graphs_dir +
                                   'intermediate_' + str(i) + '.pb')
        tf.logging.info('Save intermediate result to : ' +
                        intermediate_file_name)
        save_graph_to_file(intermediate_file_name, module_spec,
                           class_count)

    # After training is complete, force one last save of the train
checkpoint.
    train_saver.save(sess, CHECKPOINT_NAME)

    # We've completed all our training, so run a final test evaluation on
    # some new images we haven't used before.
    run_final_eval(sess, module_spec, class_count, image_lists,
                   jpeg_data_tensor, decoded_image_tensor,
resized_image_tensor,
                   bottleneck_tensor)

    # Write out the trained graph and labels with the weights stored as
    # constants.
    tf.logging.info('Save final result to : ' + FLAGS.output_graph)
    if wants_quantization:
      tf.logging.info('The model is instrumented for quantization with TF-
Lite')
    save_graph_to_file(FLAGS.output_graph, module_spec, class_count)
    with tf.gfile.FastGFile(FLAGS.output_labels, 'w') as f:
      f.write('\n'.join(image_lists.keys()) + '\n')

    if FLAGS.saved_model_dir:
      export_model(module_spec, class_count, FLAGS.saved_model_dir)


if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--image_dir',
      type=str,
      default='',
      help='Path to folders of labeled images.'
  )
  parser.add_argument(
      '--output_graph',
      type=str,
      default='/tmp/output_graph.pb',
      help='Where to save the trained graph.'
  )
  parser.add_argument(
      '--intermediate_output_graphs_dir',
      type=str,
      default='/tmp/intermediate_graph/',
      help='Where to save the intermediate graphs.'
  )
  parser.add_argument(
      '--intermediate_store_frequency',
      type=int,
      default=0,
      help="""\
         How many steps to store intermediate graph. If "0" then will not
         store.\
      """
  )
  parser.add_argument(
      '--output_labels',
      type=str,
      default='/tmp/output_labels.txt',
      help='Where to save the trained graph\'s labels.'
  )
  parser.add_argument(
      '--summaries_dir',
      type=str,
      default='/tmp/retrain_logs',
      help='Where to save summary logs for TensorBoard.'
  )
  parser.add_argument(
      '--how_many_training_steps',
      type=int,
      default=4000,
      help='How many training steps to run before ending.'
  )
  parser.add_argument(
      '--learning_rate',
      type=float,
      default=0.01,
      help='How large a learning rate to use when training.'
  )
  parser.add_argument(
      '--testing_percentage',
      type=int,
      default=10,
      help='What percentage of images to use as a test set.'
  )
  parser.add_argument(
      '--validation_percentage',
      type=int,
      default=10,
      help='What percentage of images to use as a validation set.'
  )
  parser.add_argument(
      '--eval_step_interval',
      type=int,
      default=10,
      help='How often to evaluate the training results.'
  )
  parser.add_argument(
      '--train_batch_size',
      type=int,
      default=100,
      help='How many images to train on at a time.'
  )
  parser.add_argument(
      '--test_batch_size',
      type=int,
      default=-1,
      help="""\
      How many images to test on. This test set is only used once, to
evaluate
      the final accuracy of the model after training completes.
      A value of -1 causes the entire test set to be used, which leads to
more
      stable results across runs.\
      """
  )
  parser.add_argument(
      '--validation_batch_size',
      type=int,
      default=100,
      help="""\
      How many images to use in an evaluation batch. This validation set is
      used much more often than the test set, and is an early indicator of
how
      accurate the model is during training.
      A value of -1 causes the entire validation set to be used, which
leads to
      more stable results across training iterations, but may be slower on
large
      training sets.\
      """
  )
  parser.add_argument(
      '--print_misclassified_test_images',
      default=False,
      help="""\
      Whether to print out a list of all misclassified test images.\
      """,
      action='store_true'
  )
  parser.add_argument(
      '--bottleneck_dir',
      type=str,
      default='/tmp/bottleneck',
      help='Path to cache bottleneck layer values as files.'
  )
  parser.add_argument(
      '--final_tensor_name',
      type=str,
      default='final_result',
      help="""\
      The name of the output classification layer in the retrained graph.\
      """
  )
  parser.add_argument(
      '--flip_left_right',
      default=False,
      help="""\
      Whether to randomly flip half of the training images horizontally.\
      """,
      action='store_true'
  )
  parser.add_argument(
      '--random_crop',
      type=int,
      default=0,
      help="""\
      A percentage determining how much of a margin to randomly crop off
the
      training images.\
      """
  )
  parser.add_argument(
      '--random_scale',
      type=int,
      default=0,
      help="""\
      A percentage determining how much to randomly scale up the size of
the
      training images by.\
      """
  )
  parser.add_argument(
      '--random_brightness',
      type=int,
      default=0,
      help=
  parser.add_argument(
      '--tfhub_module',
      type=str,
      default=(

  parser.add_argument(
      '--saved_model_dir',
      type=str,
      default='',
      help='Where to save the exported graph.')
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

6.4 IMAGE-CLASSIFIER.PY

```
# First import tensorflow!
import tensorflow as tf
# The sys module provides access to some variables used/maintained by the
interpreter
# and to functions that interact with the interpreter.
import sys

# sys.argv is a list that contains the command-line arguments passed to the
script.
# Here, we're trying to get the image name, so change this as you see fit!
image_path = sys.argv[1]

# Read in the image_data!
# Here, 'rb' will open the file for reading in binary format.
image_data = tf.gfile.FastGFile(image_path, 'rb').read()

# Loads label file, strips off carriage return.
# Those are basically the classes we retrained our model on!
label_lines = [line.rstrip() for line
                   in tf.gfile.GFile("/tf_files/retrained_labels.txt")]

# Unpersists graph from file..
# Here, get our model from the saved retrained_graph.pb file and save it
# in graph_def variable then parse it!
with tf.gfile.FastGFile("/tf_files/retrained_graph.pb", 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    _ = tf.import_graph_def(graph_def, name='')

# Now,  need the session to get the predictions from the retrained model!
with tf.Session() as sess:
    # Feed the image_data as input to the graph and get first prediction.
    # The Softmax function will be used in the final layer to map the input
data
    # into probabilities of an expected output!
    softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')
    # This will run the Softmax function on our image data, which will
provide the predictions array.
    predictions = sess.run(softmax_tensor, \
             {'DecodeJpeg/contents:0': image_data})

    # Sort predictions with argsort to show labels of first prediction in
order of confidence
    top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]

    # For each prediction we have, we'll print out the label and its score!
    for node_id in top_k:
        human_string = label_lines[node_id]
        score = predictions[0][node_id]
        print('%s (score = %.5f)' % (human_string, score))
```

6.5 TRAINING CODES

```
$ python tensorflow/examples/image_retraining/retrain.py \
--bottleneck_dir=/tf_files/bottlenecks \
--how_many_training_steps 4000 \
--model_dir=/tf_files/inception \
--output_graph=/tf_files/retrained_graph.pb \
--output_labels=/tf_files/retrained_labels.txt \
--image_dir /tf_files/skin_lesions
```

CHAPTER 7

RESULT AND SCREENSHOTS

7.1 SCREENSHOTS

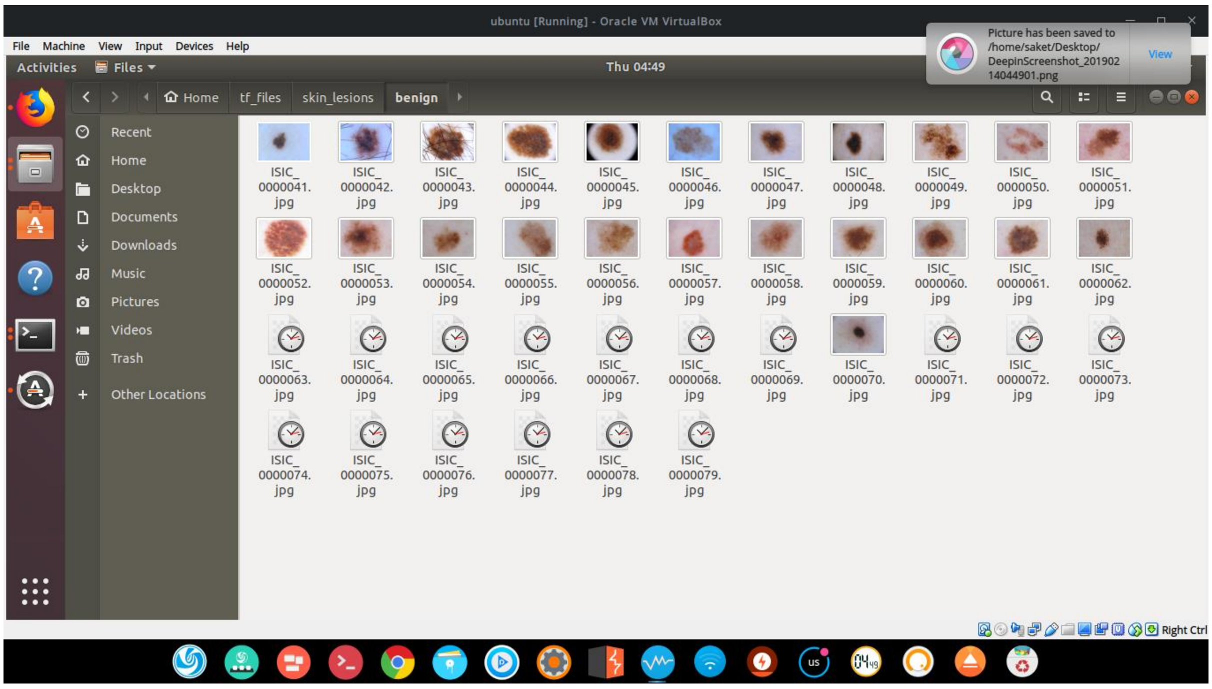

ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Files
Thu 04:49
Picture has been saved to /home/saket/Desktop/DeepinScreenshot_20190214044901.png
View
Home tf_files skin_lesions benign
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
ISIC_0000041.jpg ISIC_0000042.jpg ISIC_0000043.jpg ISIC_0000044.jpg ISIC_0000045.jpg ISIC_0000046.jpg ISIC_0000047.jpg ISIC_0000048.jpg ISIC_0000049.jpg ISIC_0000050.jpg ISIC_0000051.jpg
ISIC_0000052.jpg ISIC_0000053.jpg ISIC_0000054.jpg ISIC_0000055.jpg ISIC_0000056.jpg ISIC_0000057.jpg ISIC_0000058.jpg ISIC_0000059.jpg ISIC_0000060.jpg ISIC_0000061.jpg ISIC_0000062.jpg
ISIC_0000063.jpg ISIC_0000064.jpg ISIC_0000065.jpg ISIC_0000066.jpg ISIC_0000067.jpg ISIC_0000068.jpg ISIC_0000069.jpg ISIC_0000070.jpg ISIC_0000071.jpg ISIC_0000072.jpg ISIC_0000073.jpg
ISIC_0000074.jpg ISIC_0000075.jpg ISIC_0000076.jpg ISIC_0000077.jpg ISIC_0000078.jpg ISIC_0000079.jpg
Right Ctrl

Fig 7.1 Directory Structure of benign

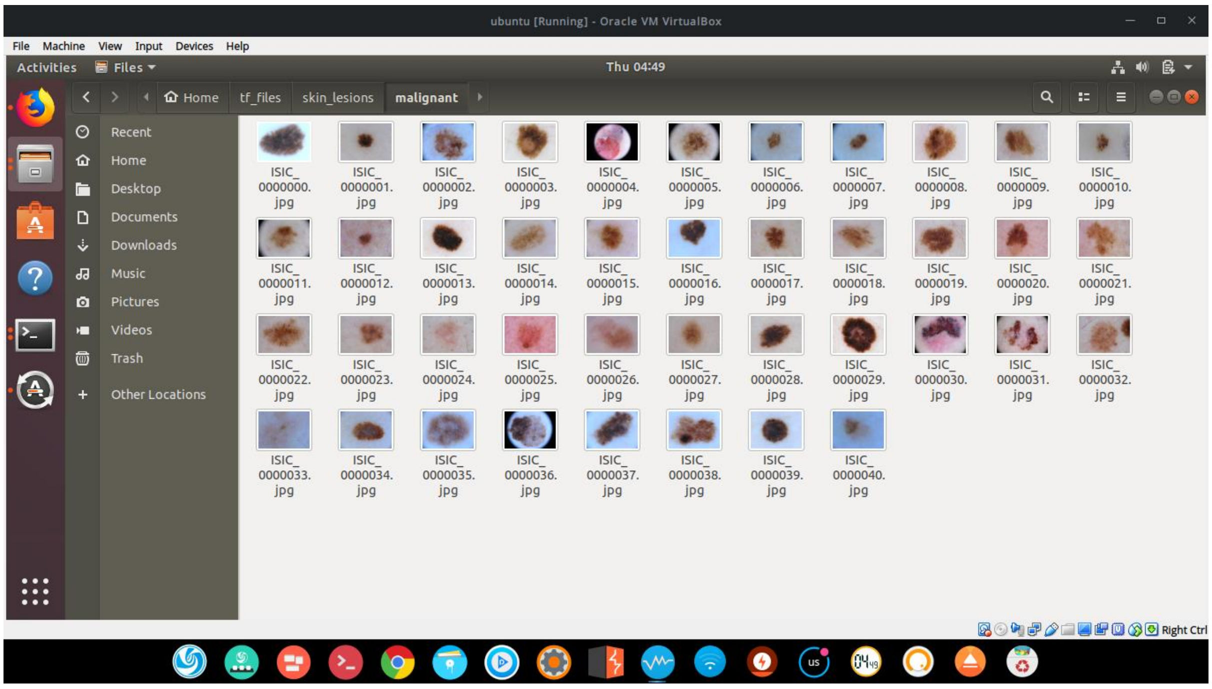
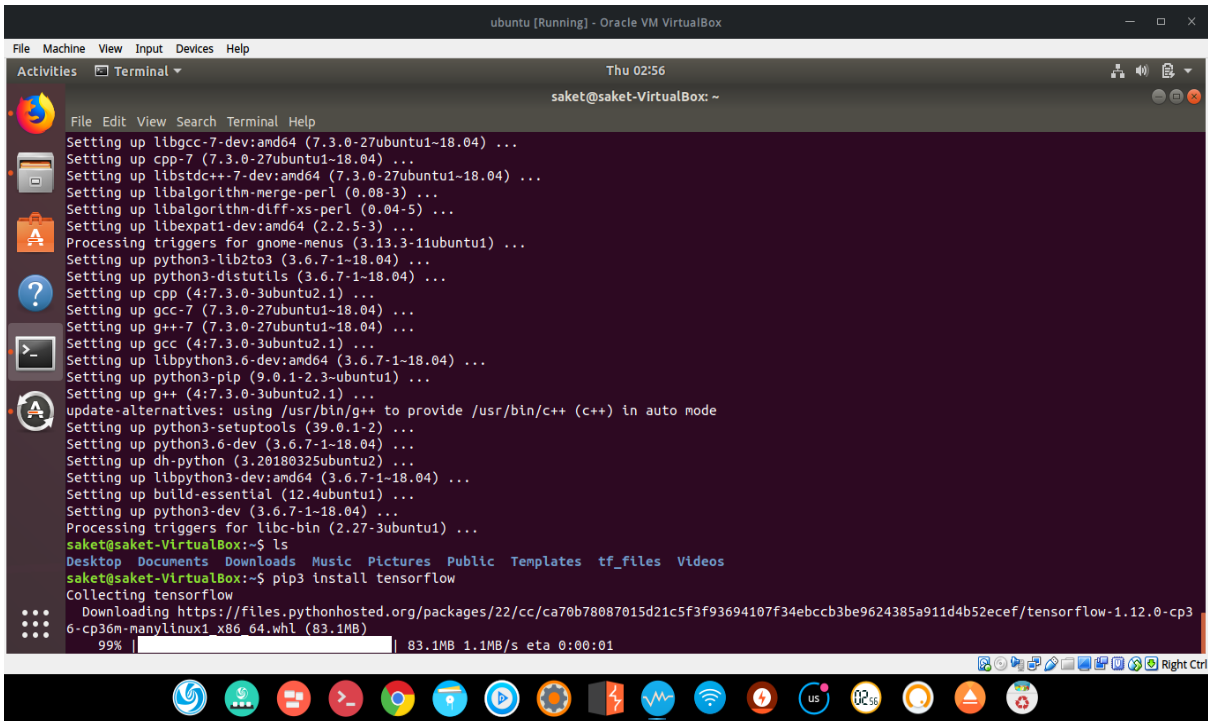
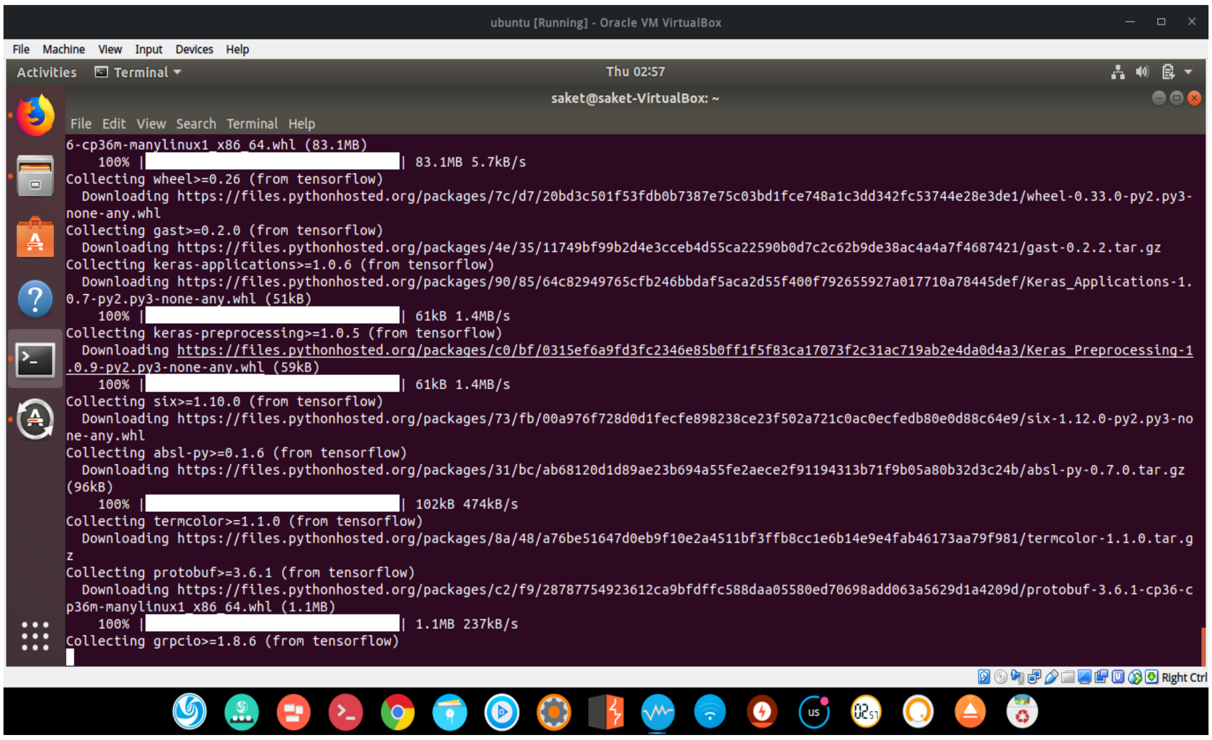

ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Files
Thu 04:49
Home tf_files skin_lesions malignant
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
ISIC_0000000.jpg ISIC_0000001.jpg ISIC_0000002.jpg ISIC_0000003.jpg ISIC_0000004.jpg ISIC_0000005.jpg ISIC_0000006.jpg ISIC_0000007.jpg ISIC_0000008.jpg ISIC_0000009.jpg ISIC_0000010.jpg
ISIC_0000011.jpg ISIC_0000012.jpg ISIC_0000013.jpg ISIC_0000014.jpg ISIC_0000015.jpg ISIC_0000016.jpg ISIC_0000017.jpg ISIC_0000018.jpg ISIC_0000019.jpg ISIC_0000020.jpg ISIC_0000021.jpg
ISIC_0000022.jpg ISIC_0000023.jpg ISIC_0000024.jpg ISIC_0000025.jpg ISIC_0000026.jpg ISIC_0000027.jpg ISIC_0000028.jpg ISIC_0000029.jpg ISIC_0000030.jpg ISIC_0000031.jpg ISIC_0000032.jpg
ISIC_0000033.jpg ISIC_0000034.jpg ISIC_0000035.jpg ISIC_0000036.jpg ISIC_0000037.jpg ISIC_0000038.jpg ISIC_0000039.jpg ISIC_0000040.jpg
Right Ctrl

Fig 7.2 Directory Structure of malignant



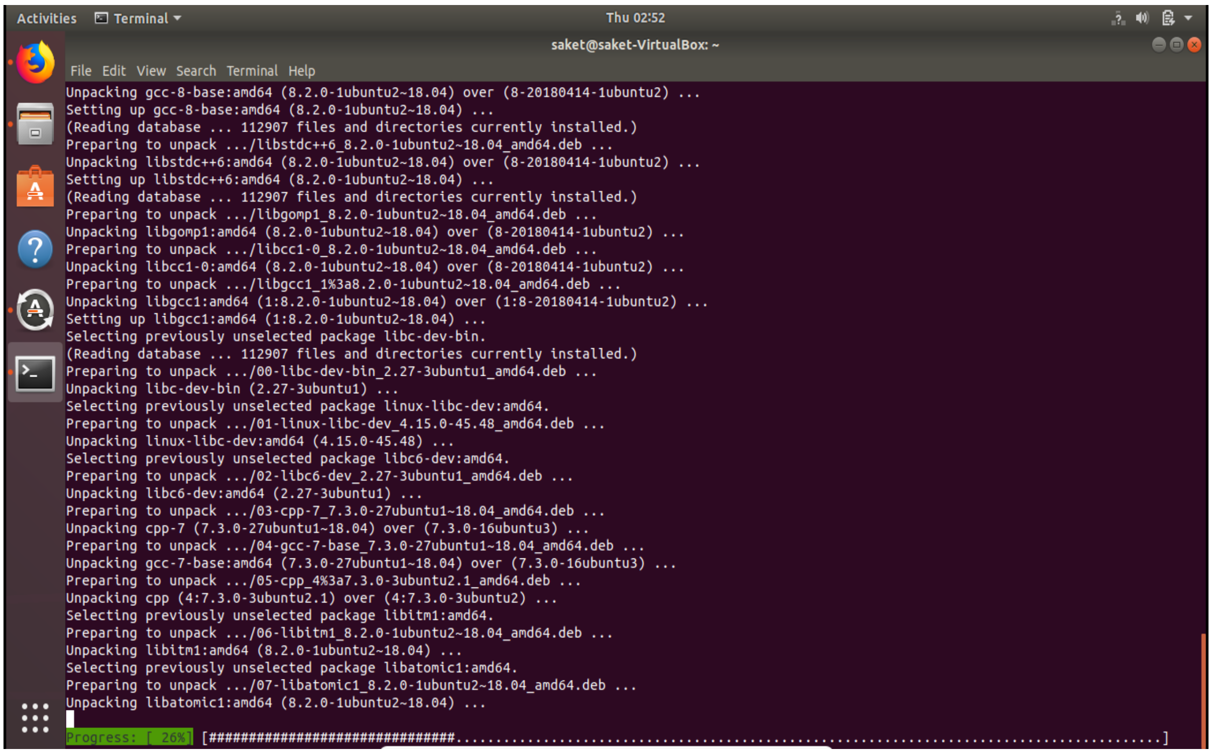
```
Setting up libgcc-7-dev:amd64 (7.3.0-27ubuntu1~18.04) ...
Setting up cpp-7 (7.3.0-27ubuntu1~18.04) ...
Setting up libstdc++-7-dev:amd64 (7.3.0-27ubuntu1~18.04) ...
Setting up libalgorithm-merge-perl (0.08-3) ...
Setting up libalgorithm-diff-xs-perl (0.04-5) ...
Setting up libexpat1-dev:amd64 (2.2.5-3) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1) ...
Setting up python3-lib2to3 (3.6.7-1~18.04) ...
Setting up python3-distutils (3.6.7-1~18.04) ...
Setting up cpp (4:7.3.0-3ubuntu2.1) ...
Setting up gcc-7 (7.3.0-27ubuntu1~18.04) ...
Setting up g++-7 (7.3.0-27ubuntu1~18.04) ...
Setting up gcc (4:7.3.0-3ubuntu2.1) ...
Setting up libpython3.6-dev:amd64 (3.6.7-1~18.04) ...
Setting up python3-pip (9.0.1-2.3~ubuntu1) ...
Setting up g++ (4:7.3.0-3ubuntu2.1) ...
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
Setting up python3-setuptools (39.0.1-2) ...
Setting up python3.6-dev (3.6.7-1~18.04) ...
Setting up dh-python (3.20180325ubuntu2) ...
Setting up libpython3-dev:amd64 (3.6.7-1~18.04) ...
Setting up build-essential (12.4ubuntu1) ...
Setting up python3-dev (3.6.7-1~18.04) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
saket@saket-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  tf_files  Videos
saket@saket-VirtualBox:~$ pip3 install tensorflow
Collecting tensorflow
  Downloading https://files.pythonhosted.org/packages/22/cc/ca70b78087015d21c5f3f93694107f34ebccb3be9624385a911d4b52ecef/tensorflow-1.12.0-cp3
6-cp36m-manylinux1_x86_64.whl (83.1MB)
    99% |████████████████████████████████| 83.1MB 1.1MB/s eta 0:00:01
```

Fig 7.3 Installing tensorflow



```
6-cp36m-manylinux1_x86_64.whl (83.1MB)
    100% |████████████████████████████████| 83.1MB 5.7kB/s
Collecting wheel>=0.26 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/7c/d7/20bd3c501f53fdb0b7387e75c03bd1fce748a1c3dd342fc53744e28e3de1/wheel-0.33.0-py2.py3-
none-any.whl
Collecting gast>=0.2.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/4e/35/11749bf99b2d4e3cceb4d55ca22590b0d7c2c62b9de38ac4a4a7f4687421/gast-0.2.2.tar.gz
Collecting keras-applications>=1.0.6 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/90/85/64c82949765cfb246bbdaf5aca2d55f400f792655927a017710a78445def/Keras_Applications-1.
0.7-py2.py3-none-any.whl (51kB)
    100% |████████████████████████████████| 61kB 1.4MB/s
Collecting keras-preprocessing>=1.0.5 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/c0/bf/0315ef6a9fd3fc2346e85b0ff1f5f83ca17073f2c31ac719ab2e4da0d4a3/Keras_Preprocessing-1
.0.9-py2.py3-none-any.whl (59kB)
    100% |████████████████████████████████| 61kB 1.4MB/s
Collecting six>=1.10.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-no
ne-any.whl
Collecting absl-py>=0.1.6 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/31/bc/ab68120d1d89ae23b694a55fe2aece2f91194313b71f9b05a80b32d3c24b/absl-py-0.7.0.tar.gz
(96kB)
    100% |████████████████████████████████| 102kB 474kB/s
Collecting termcolor>=1.1.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/8a/48/a76be51647d0eb9f10e2a4511bf3ffb8cc1e6b14e9e4fab46173aa79f981/termcolor-1.1.0.tar.g
z
Collecting protobuf>=3.6.1 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/c2/f9/28787754923612ca9bfdffc588daa05580ed70698add063a5629d1a4209d/protobuf-3.6.1-cp36-c
p36m-manylinux1_x86_64.whl (1.1MB)
    100% |████████████████████████████████| 1.1MB 237kB/s
Collecting grpcio>=1.8.6 (from tensorflow)
```

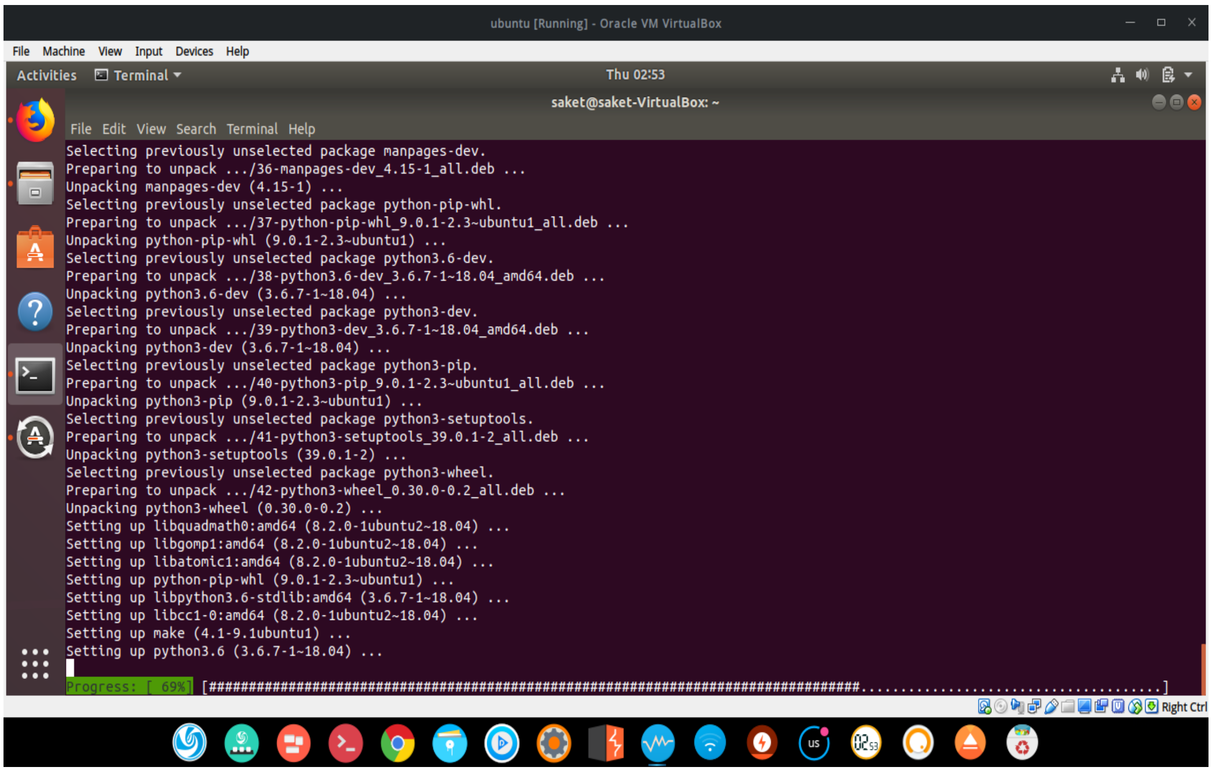
Fig 7.4 Tensorflow pull request



Activities
Terminal
Thu 02:52
saket@saket-VirtualBox: ~
File Edit View Search Terminal Help

```
Unpacking gcc-8-base:amd64 (8.2.0-1ubuntu2~18.04) over (8-20180414-1ubuntu2) ...
Setting up gcc-8-base:amd64 (8.2.0-1ubuntu2~18.04) ...
(Reading database ... 112907 files and directories currently installed.)
Preparing to unpack .../libstdc++6_8.2.0-1ubuntu2~18.04_amd64.deb ...
Unpacking libstdc++6:amd64 (8.2.0-1ubuntu2~18.04) over (8-20180414-1ubuntu2) ...
Setting up libstdc++6:amd64 (8.2.0-1ubuntu2~18.04) ...
(Reading database ... 112907 files and directories currently installed.)
Preparing to unpack .../libgomp1_8.2.0-1ubuntu2~18.04_amd64.deb ...
Unpacking libgomp1:amd64 (8.2.0-1ubuntu2~18.04) over (8-20180414-1ubuntu2) ...
Preparing to unpack .../libcc1-0_8.2.0-1ubuntu2~18.04_amd64.deb ...
Unpacking libcc1-0:amd64 (8.2.0-1ubuntu2~18.04) over (8-20180414-1ubuntu2) ...
Preparing to unpack .../libgcc1_1%3a8.2.0-1ubuntu2~18.04_amd64.deb ...
Unpacking libgcc1:amd64 (1:8.2.0-1ubuntu2~18.04) over (1:8-20180414-1ubuntu2) ...
Setting up libgcc1:amd64 (1:8.2.0-1ubuntu2~18.04) ...
Selecting previously unselected package libc-dev-bin.
(Reading database ... 112907 files and directories currently installed.)
Preparing to unpack .../00-libc-dev-bin_2.27-3ubuntu1_amd64.deb ...
Unpacking libc-dev-bin (2.27-3ubuntu1) ...
Selecting previously unselected package linux-libc-dev:amd64.
Preparing to unpack .../01-linux-libc-dev_4.15.0-45.48_amd64.deb ...
Unpacking linux-libc-dev:amd64 (4.15.0-45.48) ...
Selecting previously unselected package libc6-dev:amd64.
Preparing to unpack .../02-libc6-dev_2.27-3ubuntu1_amd64.deb ...
Unpacking libc6-dev:amd64 (2.27-3ubuntu1) ...
Preparing to unpack .../03-cpp-7_7.3.0-27ubuntu1~18.04_amd64.deb ...
Unpacking cpp-7 (7.3.0-27ubuntu1~18.04) over (7.3.0-16ubuntu3) ...
Preparing to unpack .../04-gcc-7-base_7.3.0-27ubuntu1~18.04_amd64.deb ...
Unpacking gcc-7-base:amd64 (7.3.0-27ubuntu1~18.04) over (7.3.0-16ubuntu3) ...
Preparing to unpack .../05-cpp_4%3a7.3.0-3ubuntu2.1_amd64.deb ...
Unpacking cpp (4:7.3.0-3ubuntu2.1) over (4:7.3.0-3ubuntu2) ...
Selecting previously unselected package libitm1:amd64.
Preparing to unpack .../06-libitm1_8.2.0-1ubuntu2~18.04_amd64.deb ...
Unpacking libitm1:amd64 (8.2.0-1ubuntu2~18.04) ...
Selecting previously unselected package libatomic1:amd64.
Preparing to unpack .../07-libatomic1_8.2.0-1ubuntu2~18.04_amd64.deb ...
Unpacking libatomic1:amd64 (8.2.0-1ubuntu2~18.04) ...

Progress: [ 26%] [###############################.............................................................................]
```

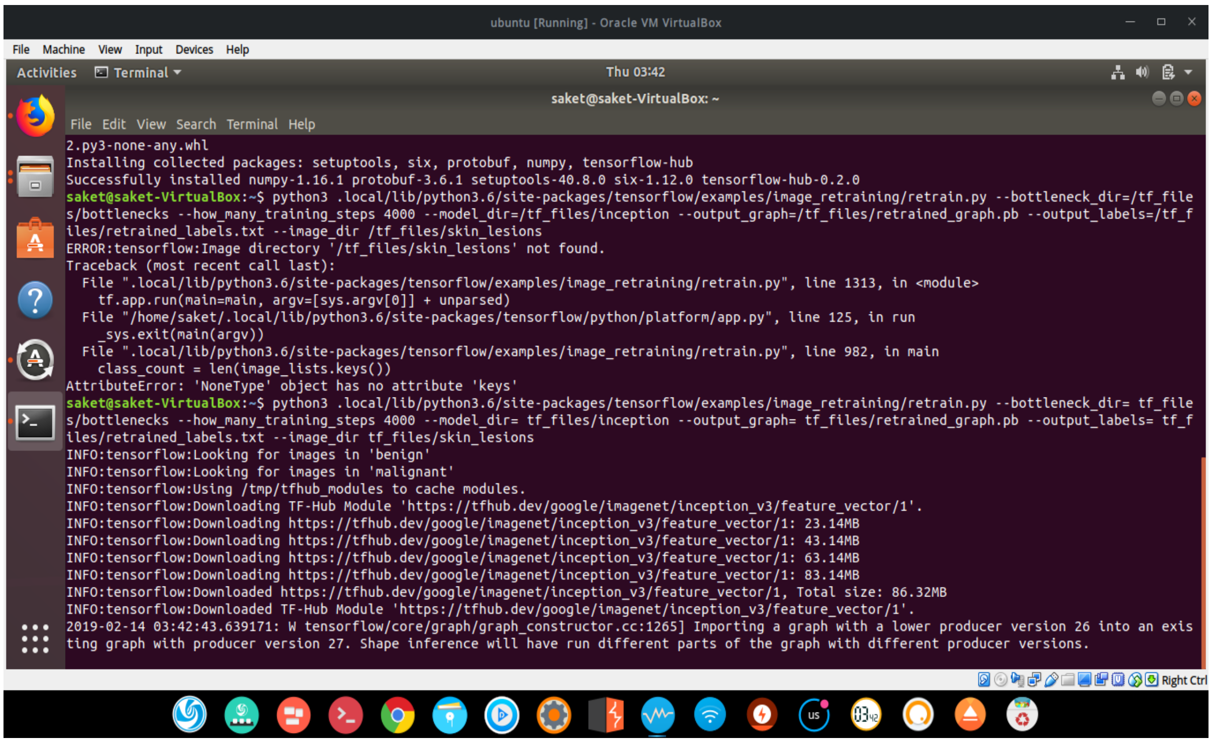
Fig 7.5 Unpacking tensorflow files



ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Thu 02:53
saket@saket-VirtualBox: ~
File Edit View Search Terminal Help
Right Ctrl

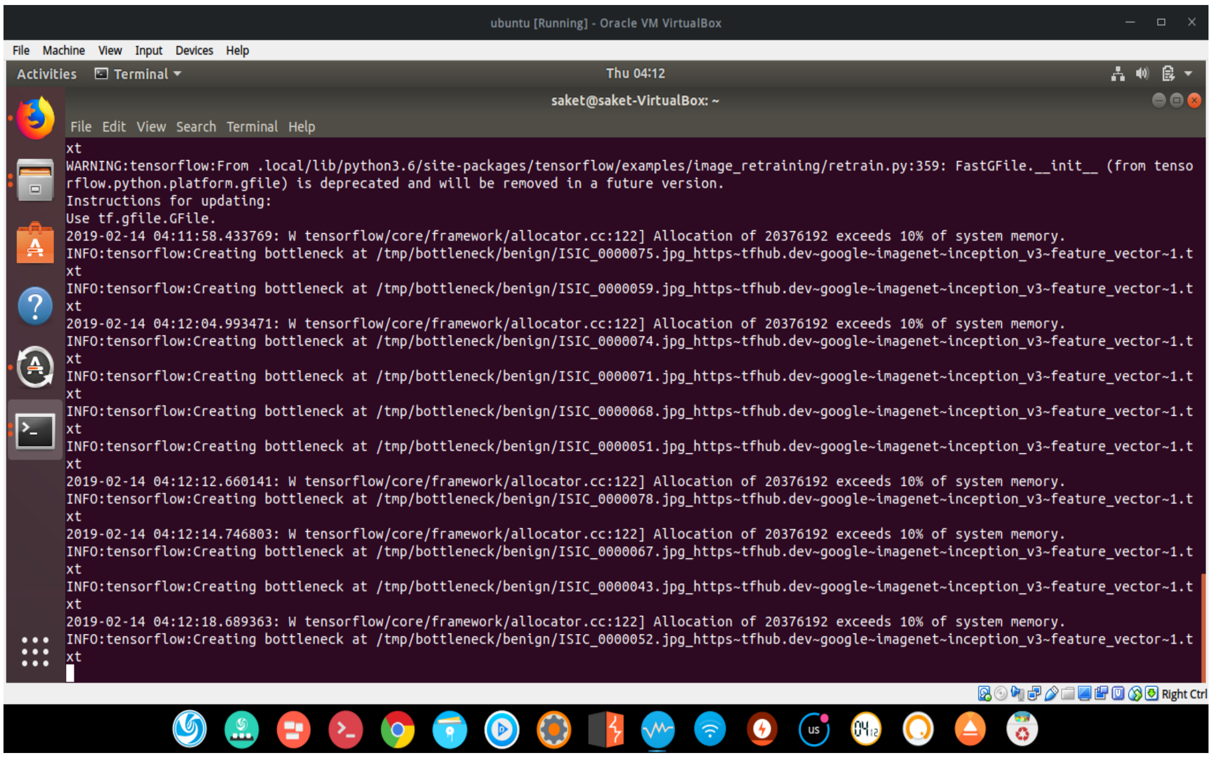
```
Selecting previously unselected package manpages-dev.
Preparing to unpack .../36-manpages-dev_4.15-1_all.deb ...
Unpacking manpages-dev (4.15-1) ...
Selecting previously unselected package python-pip-whl.
Preparing to unpack .../37-python-pip-whl_9.0.1-2.3~ubuntu1_all.deb ...
Unpacking python-pip-whl (9.0.1-2.3~ubuntu1) ...
Selecting previously unselected package python3.6-dev.
Preparing to unpack .../38-python3.6-dev_3.6.7-1~18.04_amd64.deb ...
Unpacking python3.6-dev (3.6.7-1~18.04) ...
Selecting previously unselected package python3-dev.
Preparing to unpack .../39-python3-dev_3.6.7-1~18.04_amd64.deb ...
Unpacking python3-dev (3.6.7-1~18.04) ...
Selecting previously unselected package python3-pip.
Preparing to unpack .../40-python3-pip_9.0.1-2.3~ubuntu1_all.deb ...
Unpacking python3-pip (9.0.1-2.3~ubuntu1) ...
Selecting previously unselected package python3-setuptools.
Preparing to unpack .../41-python3-setuptools_39.0.1-2_all.deb ...
Unpacking python3-setuptools (39.0.1-2) ...
Selecting previously unselected package python3-wheel.
Preparing to unpack .../42-python3-wheel_0.30.0-0.2_all.deb ...
Unpacking python3-wheel (0.30.0-0.2) ...
Setting up libquadmath0:amd64 (8.2.0-1ubuntu2~18.04) ...
Setting up libgomp1:amd64 (8.2.0-1ubuntu2~18.04) ...
Setting up libatomic1:amd64 (8.2.0-1ubuntu2~18.04) ...
Setting up python-pip-whl (9.0.1-2.3~ubuntu1) ...
Setting up libpython3.6-stdlib:amd64 (3.6.7-1~18.04) ...
Setting up libcc1-0:amd64 (8.2.0-1ubuntu2~18.04) ...
Setting up make (4.1-9.1ubuntu1) ...
Setting up python3.6 (3.6.7-1~18.04) ...

Progress: [ 69%] [##################################################################################.................................]
```

Fig 7.6 Setting up tensorflow files



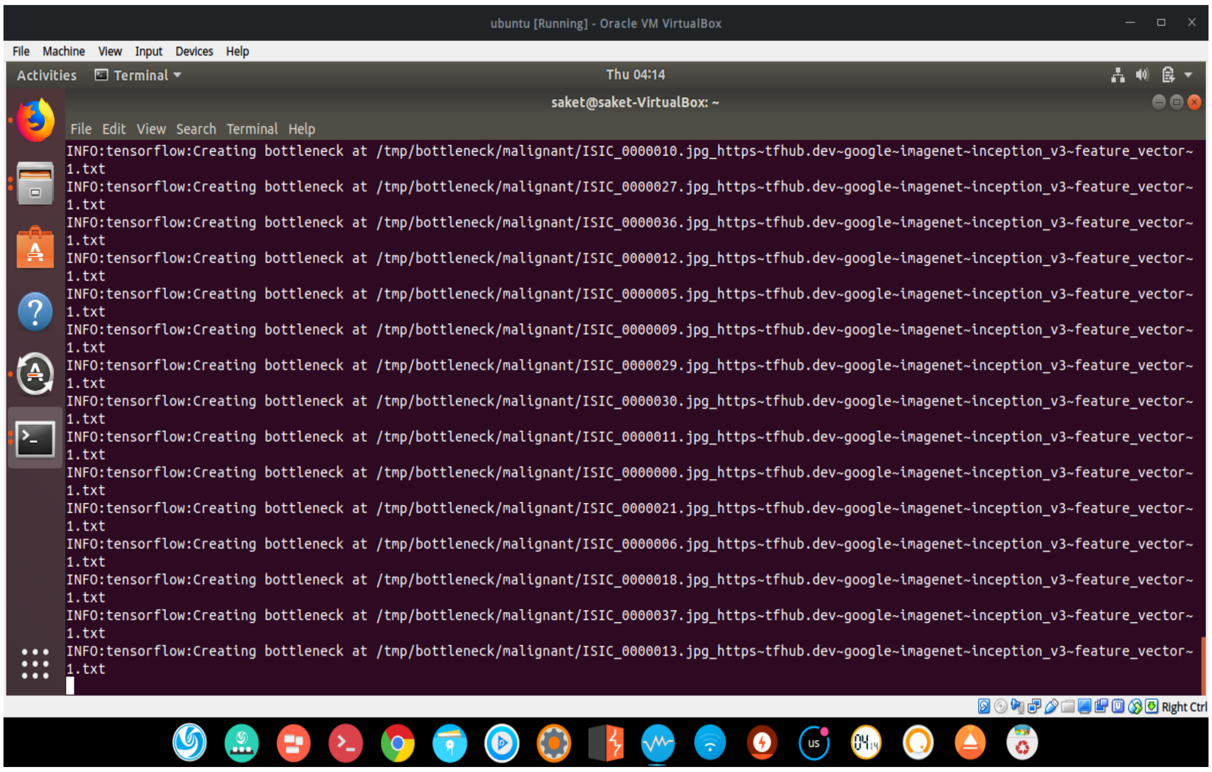
```
2.py3-none-any.whl
Installing collected packages: setuptools, six, protobuf, numpy, tensorflow-hub
Successfully installed numpy-1.16.1 protobuf-3.6.1 setuptools-40.8.0 six-1.12.0 tensorflow-hub-0.2.0
saket@saket-VirtualBox:~$ python3 .local/lib/python3.6/site-packages/tensorflow/examples/image_retraining/retrain.py --bottleneck_dir=/tf_file
s/bottlenecks --how_many_training_steps 4000 --model_dir=/tf_files/inception --output_graph=/tf_files/retrained_graph.pb --output_labels=/tf_f
iles/retrained_labels.txt --image_dir /tf_files/skin_lesions
ERROR:tensorflow:Image directory '/tf_files/skin_lesions' not found.
Traceback (most recent call last):
  File ".local/lib/python3.6/site-packages/tensorflow/examples/image_retraining/retrain.py", line 1313, in <module>
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
  File "/home/saket/.local/lib/python3.6/site-packages/tensorflow/python/platform/app.py", line 125, in run
    _sys.exit(main(argv))
  File ".local/lib/python3.6/site-packages/tensorflow/examples/image_retraining/retrain.py", line 982, in main
    class_count = len(image_lists.keys())
AttributeError: 'NoneType' object has no attribute 'keys'
saket@saket-VirtualBox:~$ python3 .local/lib/python3.6/site-packages/tensorflow/examples/image_retraining/retrain.py --bottleneck_dir= tf_file
s/bottlenecks --how_many_training_steps 4000 --model_dir= tf_files/inception --output_graph= tf_files/retrained_graph.pb --output_labels= tf_f
iles/retrained_labels.txt --image_dir tf_files/skin_lesions
INFO:tensorflow:Looking for images in 'benign'
INFO:tensorflow:Looking for images in 'malignant'
INFO:tensorflow:Using /tmp/tfhub_modules to cache modules.
INFO:tensorflow:Downloading TF-Hub Module 'https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1'.
INFO:tensorflow:Downloading https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1: 23.14MB
INFO:tensorflow:Downloading https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1: 43.14MB
INFO:tensorflow:Downloading https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1: 63.14MB
INFO:tensorflow:Downloading https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1: 83.14MB
INFO:tensorflow:Downloaded https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1, Total size: 86.32MB
INFO:tensorflow:Downloaded TF-Hub Module 'https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1'.
2019-02-14 03:42:43.639171: W tensorflow/core/graph/graph_constructor.cc:1265] Importing a graph with a lower producer version 26 into an exis
ting graph with producer version 27. Shape inference will have run different parts of the graph with different producer versions.
```

Fig 7.7 Training instructions



```
xt
WARNING:tensorflow:From .local/lib/python3.6/site-packages/tensorflow/examples/image_retraining/retrain.py:359: FastGFile.__init__ (from tenso
rflow.python.platform.gfile) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.gfile.GFile.
2019-02-14 04:11:58.433769: W tensorflow/core/framework/allocator.cc:122] Allocation of 20376192 exceeds 10% of system memory.
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000075.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000059.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
2019-02-14 04:12:04.993471: W tensorflow/core/framework/allocator.cc:122] Allocation of 20376192 exceeds 10% of system memory.
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000074.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000071.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000068.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000051.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
2019-02-14 04:12:12.660141: W tensorflow/core/framework/allocator.cc:122] Allocation of 20376192 exceeds 10% of system memory.
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000078.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
2019-02-14 04:12:14.746803: W tensorflow/core/framework/allocator.cc:122] Allocation of 20376192 exceeds 10% of system memory.
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000067.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000043.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
2019-02-14 04:12:18.689363: W tensorflow/core/framework/allocator.cc:122] Allocation of 20376192 exceeds 10% of system memory.
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/benign/ISIC_0000052.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~1.t
xt
```

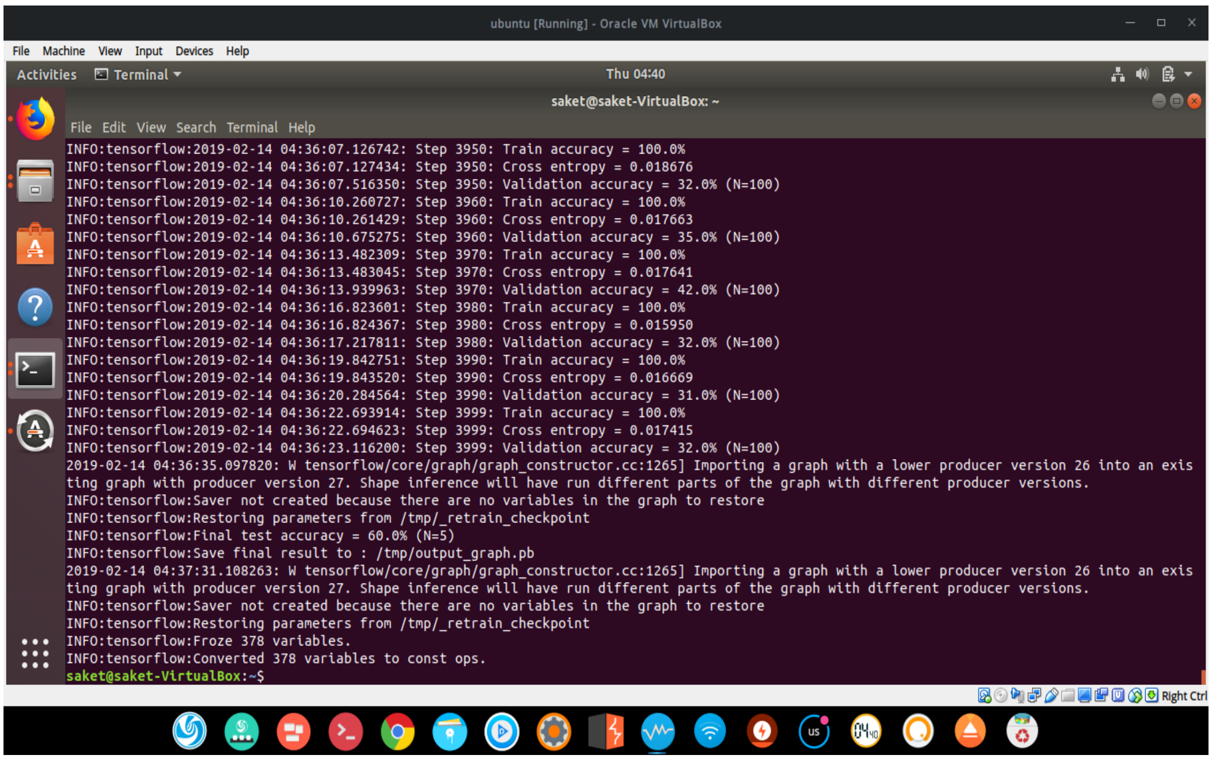
Fig 7.8 Bottleneck creation



```
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000010.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000027.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000036.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000012.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000005.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000009.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000029.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000030.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000011.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000000.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000021.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000006.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000018.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000037.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/malignant/ISIC_0000013.jpg_https~tfhub.dev~google~imagenet~inception_v3~feature_vector~
1.txt
```

ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Thu 04:14
saket@saket-VirtualBox: ~
File Edit View Search Terminal Help
Right Ctrl

Fig 7.9 Training Started



```
INFO:tensorflow:2019-02-14 04:36:07.126742: Step 3950: Train accuracy = 100.0%
INFO:tensorflow:2019-02-14 04:36:07.127434: Step 3950: Cross entropy = 0.018676
INFO:tensorflow:2019-02-14 04:36:07.516350: Step 3950: Validation accuracy = 32.0% (N=100)
INFO:tensorflow:2019-02-14 04:36:10.260727: Step 3960: Train accuracy = 100.0%
INFO:tensorflow:2019-02-14 04:36:10.261429: Step 3960: Cross entropy = 0.017663
INFO:tensorflow:2019-02-14 04:36:10.675275: Step 3960: Validation accuracy = 35.0% (N=100)
INFO:tensorflow:2019-02-14 04:36:13.482309: Step 3970: Train accuracy = 100.0%
INFO:tensorflow:2019-02-14 04:36:13.483045: Step 3970: Cross entropy = 0.017641
INFO:tensorflow:2019-02-14 04:36:13.939963: Step 3970: Validation accuracy = 42.0% (N=100)
INFO:tensorflow:2019-02-14 04:36:16.823601: Step 3980: Train accuracy = 100.0%
INFO:tensorflow:2019-02-14 04:36:16.824367: Step 3980: Cross entropy = 0.015950
INFO:tensorflow:2019-02-14 04:36:17.217811: Step 3980: Validation accuracy = 32.0% (N=100)
INFO:tensorflow:2019-02-14 04:36:19.842751: Step 3990: Train accuracy = 100.0%
INFO:tensorflow:2019-02-14 04:36:19.843520: Step 3990: Cross entropy = 0.016669
INFO:tensorflow:2019-02-14 04:36:20.284564: Step 3990: Validation accuracy = 31.0% (N=100)
INFO:tensorflow:2019-02-14 04:36:22.693914: Step 3999: Train accuracy = 100.0%
INFO:tensorflow:2019-02-14 04:36:22.694623: Step 3999: Cross entropy = 0.017415
INFO:tensorflow:2019-02-14 04:36:23.116200: Step 3999: Validation accuracy = 32.0% (N=100)
2019-02-14 04:36:35.097820: W tensorflow/core/graph/graph_constructor.cc:1265] Importing a graph with a lower producer version 26 into an exis
ting graph with producer version 27. Shape inference will have run different parts of the graph with different producer versions.
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
INFO:tensorflow:Restoring parameters from /tmp/_retrain_checkpoint
INFO:tensorflow:Final test accuracy = 60.0% (N=5)
INFO:tensorflow:Save final result to : /tmp/output_graph.pb
2019-02-14 04:37:31.108263: W tensorflow/core/graph/graph_constructor.cc:1265] Importing a graph with a lower producer version 26 into an exis
ting graph with producer version 27. Shape inference will have run different parts of the graph with different producer versions.
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
INFO:tensorflow:Restoring parameters from /tmp/_retrain_checkpoint
INFO:tensorflow:Froze 378 variables.
INFO:tensorflow:Converted 378 variables to const ops.
saket@saket-VirtualBox:~$
```

ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Thu 04:40
saket@saket-VirtualBox: ~
File Edit View Search Terminal Help
Right Ctrl

Fig 7.10 Training Finished

7.2 RESULTS

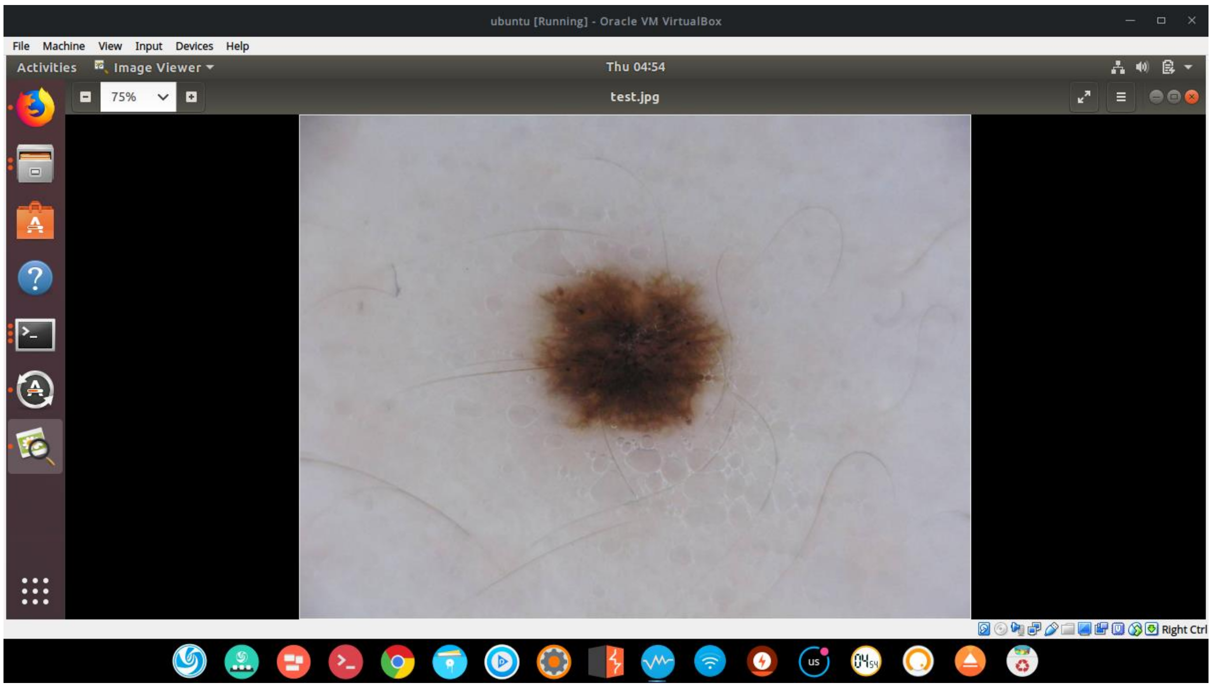

ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Image Viewer
Thu 04:54
75%
test.jpg
Right Ctrl

Fig 7.11 Skin lesion

```
$ python /tf_files/label_image.py
/tf_files/skin_lesions/benign/ISIC_0010786.jpg
```

Fig 7.12 Passing Image as parameter to model

```
benign (score = 0.95702)
malignant (score = 0.04298)
```

Fig 7.13 Subsequent result

CHAPTER 8
CONCLUSION AND FUTUREWORK

8.1 CONCLUSION

The methodology has proposed a Deep Learning solution for assisting dermatologists during the diagnosis of skin lesions. More specifically, it has investigated how a previous semantic segmentation of the skin lesion improves the performance of a fine-tuned convolutional neural network model approach for a 2-class classifier for early melanoma detection.The proposed segmentation model trains from scratch the U-Net, a ConvNet for Biomedical Image Segmentation, in order to obtain binary masks that will produce the automatically segmented skin image. The project achieves promising results, most notably, a Jaccard value of approximately 95.76%.

The classification model accepts input skin lesion images labeled as benign or malignant, and it predicts whether a previously unseen image of a skin lesion is either benign or malignant.When classifying

(i) the original ISIC dataset

(ii) the perfectly segmented ISIC dataset and

(iii)the automatically segmented ISIC dataset by fine-tuning the VGG-16

8.2 FUTUREWORKS

In future work, tweaks in regularization (stronger L2 weight penalty, higher dropout (>0.5)) could also be applied. Moreover, weights could be saved and loaded from the architecture being trained with Dermnet, a larger and skin related dataset, rather than Imagenet, a general dataset, that would also help lessen the risk of overfitting. Further work could also explore the performance of Residual Neural Networks, that have recently performed excellent results on image classification tasks by proposing substantially deeper and easier to optimize networks.

CHAPTER 9

REFERENCES

[1]. http://www.skincancer.org

[2]. http://www.medicinenet.com

[3]. Haykin, Simon. Neural networks: a comprehensive foundation. Prentice Hall PTR, 1994

[4]. Kopf, Alfred W. "Prevention and early detection of skin cancer/melanoma." Cancer 62.S1

[5]. Hoshyar, Azadeh Noori, Adel Al-Jumaily, and Riza Sulaiman. "Review on automatic early skin cancer detection." Computer Science and Service System (CSSS), 2011 International Conference on. IEEE, 2011.

[6]. Sonali Raghunath Jadhav, D.K.Kamat. ”Segmentation based detection of skin cancer” IRF international conference, 20- july-2014

[7]. Lau, Ho Tak, and Adel Al-Jumaily. "Automatically Early Detection of Skin Cancer: Study Based on Nueral Netwok Classification." Soft Computing and Pattern Recognition, 2009. SOCPAR'09. International Conference of. IEEE, 2009.

[8]. Jaleel, Dr J. Abdul, Sibi Salim, and R. B. Aswin. "Artificial Neural Network Based Detection of Skin Cancer." International Journal of Advanced Research in Electronics and Instrumentation Engineering 1.3 (2012).

[9] Firmansyah, Hardian Robby, EntinMartianaKusumaningtyas, and FadilahFahrulHardiansyah. "Detection melanoma cancer using ABCD rule based on mobile device." Knowledge Creation and Intelligent Computing (IES-KCIC), 2017 International Electronics Symposium on IEEE, 2017.

[10]. Dubai, Pratik, et al. "Skin cancer detection and classification." Electrical Engineering and Informatics (ICEEI), 2017 6th International Conference on.IEEE, 2017.

[11]. Razmjooy, Navid, Fatima Rashid Sheykhahmad, and NoradinGhadimi. "A hybrid neural network–world cup optimization algorithm for melanoma detection." Open Medicine 13.1 (2018): 9-16.

[12]. Aswin, R. B., J. Abdul Jaleel, and SibiSalim. "Hybrid genetic algorithm—artificial neural network classifier for skin cancer detection."Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on.IEEE, 2014.
[13]. Elgamal, Mahmoud. "Automatic Skin Cancer Images Classification."International Journal of Advanced Computer Science & Applications 4.3 (2013).