A MAJOR PROJECT REPORT

ON

DIABETIC RETINOPATHY DETECTION USING BINARIZED INCEPTION

A major project submitted in partial fulfillment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

By

S.SRISAI	**16JJ1A0546**
G. VINEETH	**16JJ1A0521**
CH. HARI KRISHNA	**16JJ1A0511**
E. SAI CHARAN K.NETHA	**16JJ1A0517**

Under the guidance of

DR. P.SAMMULAL

Professor of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

JAWAHARLALNEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD COLLEGE OF ENGINEERING JAGTIAL

NACHUPALLY (KONDAGATTU),JAGTIAL DIST.,TELANGANA STATE.

2019-2020

JAWAHARLALNEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

COLLEGE OF ENGINEERING JAGTIAL

NACHUPALLY(KONDAGATTU),JAGTIAL DIST.,TELANGANA STATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the mini project entitled "**DIABETIC RETINOPATHY DETECTION USING BINARIZED INCEPTION"** is being submitted by

S.SRISAI	**16JJ1A0546**
G.VINEETH	**16JJ1A0521**
CH.HARIKRISHNA	**16JJ1A0511**
E.SAI CHARAN KUMAR NETHA	**16JJ1A0517**

in partial fulfillment of the requirements for the award of **Bachelor of Technology in Computer Science and Engineering** by the Jawaharlal Nehru Technological University Hyderabad during the Academic Year 2019-2020, is record is a bonafide work carried out by them under my guidance and supervision.

The results embodied in this project work have not been submitted to any other university or Institute for the award of any degree or Diploma.

Project Guide	**Head of the Department**
Dr. P. SAMMULAL	**Dr. T. VENUGOPAL**
Professor of CSE	Professor of CSE

External Examiner

ACKNOWLEDGEMENT

“Task successful” makes everyone happy. But the happiness will be gold without glitter if we didn’t state the persons who have supported us to make it a success.

Success will be crowned to people who made it a reality but the people whose constant guidance and encouragement made it possible will be crowned first on the eve of success.

This acknowledgement transcends the reality of formality when we would like to express deep gratitude and respect to all those people behind the screen who guided, inspired and helped me for the completion of our project work.

I consider myself lucky enough to get such a good project. This project would add as an asset to my academic profile.

I would like to express my thankfulness to my project guide, **Dr. P.SAMMULAL,** Professor of CSE for his constant motivation and valuable help through the project work, and I express my gratitude to **Dr. T.VENUGOPAL**, Professor & Head, Department of Computer Science and Engineering for giving us the opportunity to carry out his project work.

We are very much grateful to our project coordinator **Dr. T. VENUGOPAL**, Professor of CSE for his constant mentoring and cooperation during the project.

We show gratitude to our beloved principal **Dr. N.V. RAMANA** and Vice Principal **Dr. S.VISWANADHA RAJU** for providing necessary infrastructure and resources for the accomplishment of our project report at JNTUHCEJ.

I also extend my thanks to my Team Members for their co-operation during my course.

Finally, I would like to thanks my friends for their co-operation to complete this project.

ABSTRACT

Diabetic Retinopathy is one of the leading causes of blindness and eye disease in working age population of developed world. However the state-of-the-art techniques of deep learning models' computation cost is expensive and consumes large amount of space. To apply these models to low computing devices is challenging task. To address this problem Binary Neural Networks (BNNs) could be best possible solution under research. BNNs have the ability to reduce the memory utilization and computational complexity of the neural network.

CONTENTS

1.INTRODUCTION

1.1 UNDERSTAND THE PROBLEM STATEMENT:

Diabetes mellitus (DM) commonly known as diabetes, as of 2019, an estimated 463 million people had diabetes worldwide. Diabetic retinopathy is upshot of long-standing diabetes which effects eyes and causes blindness. Current solutions to identify DR is time-consuming and highly relies on the expertise of well-trained practitioners. To straighten out the issue, in the past few few years considerable efforts have been put on developing an automated solution for DR detection.

In the early stage, numerous structural methods and traditional machine learning methods have been used in testing DR. Recently, the deep learning methods, particularly convolutional neural networks (CNNs), have contributed substantially to DR screening. With abundant of data and resources everyone contributed to enhance the DR screening system.

Most of previous automated solutions consists of two parts: feature extraction and detection/prediction algorithm . Feature extraction is the main focus as standard machine learning algorithms can be directly used as the detection/prediction algorithm. This type of approaches are effective to some extent but also suffer from several shortcomings.

To address this problem Binary Neural Networks (BNNs) could be best possible solution under research. BNNs have the ability to reduce the memory utilization and computational complexity of the neural network.

Normal eye **DR Effected eye**

1.2 ARTIFICIAL INTELLIGENCE:

In computer science, Artificial Intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans. Artificial Intelligence is a branch of Computer Science that aims to create intelligent machines. It has become an essential part of the technology industry. Knowledge engineering is a core part of Artificial Intelligence Research. Machines can often act and react like humans only if they have abundant information relating to the world.

Artificial Intelligence must have access to objects, categories, properties and relations between all of them to implement Knowledge engineering.

1.3 MACHINE LEARNING:

Machine Learning is a core part of Artificial Intelligence. It is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. It provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Machine Learning focuses on the development of computer programs that can access data and use it to learn for themselves. The process of learning begins with observations or data. The primary aim is to allow the computers to learn automatically without human intervention.

1.4 DEEP LEARNING:

Deep Learning is a subset of machine learning where artificial neural networks, algorithms inspired by the human brain, learn from large amounts of data. Deep learning allows machines to solve complex problems even when using a data set that is very diverse, unstructured and inter-connected.

Deep Learning is the most effective, supervised, time and cost-efficient Machine Learning approach. Deep Learning is not a restricted learning approach, but it abides various procedures and topographies which can be applied to complicated problems. It is considered to be the best choice for discovering complex architecture in high dimensional data by employing backpropagation algorithm.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM:

i)Conventional approach is when a DR occur, manual work had to be done to detect the retinopathy it is time-consuming and highly relies on the expertise of well-trained practitioners. There were no sophisticated tools for timely and error rate is high.

ii)Today, DNNs are almost exclusively trained on one or many very fast and power-hungry Graphic Processing Units (GPUs) . As a result, it is often a challenge to run DNNs on target low-power devices, and substantial research efforts are invested in speeding up DNNs at run-time on both general-purpose and specialized computer hardware. To solve this problem automated solution are presented but they are not feasible to use It also requires high computational power.

2.2 PROPOSED SYSTEM:

We introduce a method to train Binarized Neural Networks (BNNs) - neural networks with binary weights and activations at run-time. At training-time the binary weights and activations are used for computing the parameters gradients. During the forward pass, BNNs drastically reduce memory size and accesses, and replace most arithmetic operations with bit-wise operations, which is expected to substantially improve power-efficiency. To validate the effectiveness of BNNs we conduct experiments on the Kaggle DR dataset. BNNs achieved nearly state-of-the-art results over the kaggle DR dataset.

We show that during forward pass (runtime and train-time), BNNs drastically reduce memory consumption (size and number of accesses), and replace most arithmetic operations with bit-wise operations, which potentially lead to a substantial increase in power-efficiency (see Section 3). Moreover, a binarized CNN can lead to binary convolution kernel repetitions; We argue that dedicated hardware could reduce the time complexity by 60% .

3.SOFTWARE/HARDWARE REQUIREMENTS

SRS (Software Requirement Specification) is a document which defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platform,etc.

3.1 SOFTWARE REQUIREMENTS:

- Scikit-learn
- Tensorflow and keras libraries(Deep learning).
- Numpy and pandas libraries(Python).
- Images data set for training and testing.

3.1.1 WEBBROWSER:

A web browser (commonly referred to as a browser) is a software application for accessing information on the World Wide Web. When a user requests a particular website, the web browser retrieves the necessary content from a web server and then displays the resulting web page on the user's device.

A web browser is not the same thing as a search engine, though the two are often confused. For a user, a search engine is just a website, such as Google, Search, Bing, or Duck Duck Go, that stores searchable data about other websites. However, to connect to a website's server and display its web pages, a user must have a web browser installed.

3.1.2 GOOGLECOLABORATORY:

Google Colaboratory is a Google research project created to help disseminate machine learning education and research. It’s a Jupiter notebook environment that requires no setup to use and runs entirely in the cloud. This means that as long as you have a google account, you can freely train your models on a K80 GPU.

When you log in and connect to the colaboratory run time, you will get your own virtual machine with a K80 GPU and a Jupiter notebook environment. You can use it to your heart’s content for up to 12 hours or you close your browser.

3.1.3 INTERNET:

Internet, sometimes called simply "the Net," is a worldwide system of computer networks -- a network of networks in which users at any one computer can, if they have permission, get information from any other computer (and sometimes talk directly to users at other computers). It was conceived by the Advanced Research Projects Agency (ARPA) of the U.S. government in 1969 and was first known as the **ARPANet**. The original aim was to create a network that would allow users of a research computer at one university to "talk to" research computers at other universities.

A side benefit of ARPANet's design was that, because messages could be routed or re-routed in more than one direction, the network could continue to function even if parts of it were destroyed in the event of a military attack or other disaster.

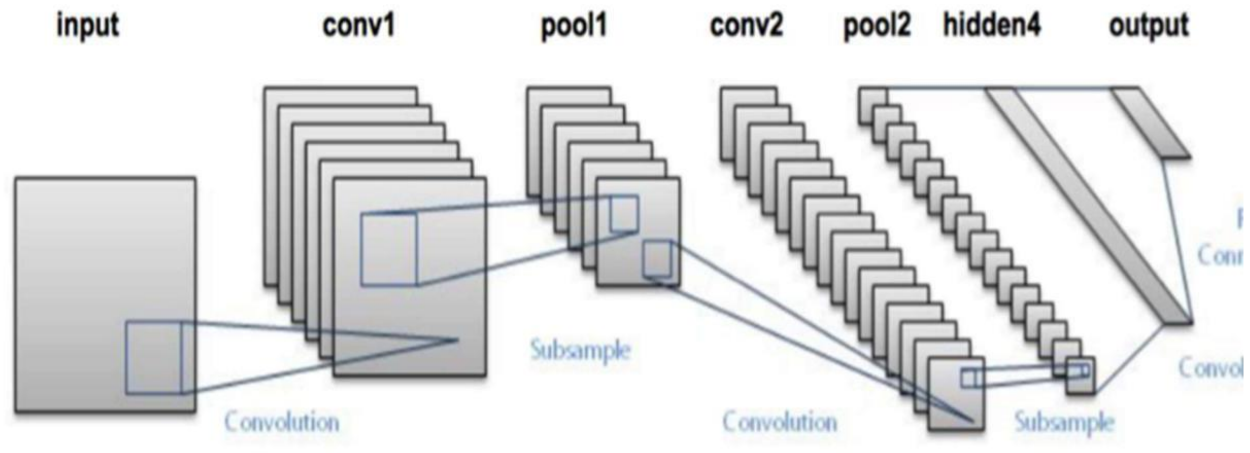


Fig .1 System Architecture

3.2 HARDWARE REQUIREMENTS:

- A computer with a minimum configuration of Intel i5 processor along with Nvidia 960 MX GPU is preferred for better processing.
- A GPU with CUDA support for parallel processing.
- Minimum 4GB of RAM.Minimum 4GB of graphic RAM.

3.2.1 PROCESSOR:

Intel **Core** is a line of mid-to-high end consumer, workstation, and enthusiast **central processing units** (CPU) marketed by **Intel Corporation**. These processors displaced the existing mid-to-high end **Pentium** processors of the time, moving the Pentium to the entry level, and bumping the **Celeron** series of processors to low end. Identical or more capable versions of Core processors are also sold as **Xeon** processors for the server and workstation markets.

As of June 2017, the lineup of Core processors included the **Intel Core i9**, **Intel Core i7**, **Intel Core i5**, and **Intel Core i3**, along with the Y - Series Intel Core CPUs.

In early 2018, news reports indicated that security flaws, referred to as "**Meltdown**" and "**Spectre**", were found "in virtually all Intel processors [made in the past two decades] that will require fixes within Windows, mac, OS and Linux". The flaw also affected cloud servers. At the time, Intel was not commenting on this issue. According to a ***New York Times*** report, "There is no easy fix for Spectre as for Meltdown, the software patch needed to fix the issue could slow down computers by as much as 30percent.

3.2.2 MEMORY RAM:

Random-access memory (RAM) is a form of computer data storage thatstores data and machine code currently being used. A random-access memory deviceallows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, cd-rws, dvd-rwsand the older magnetic tapes and drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement.

Ram contains multiplexing and demultiplexing circuitry, to connect the data lines to the addressed storage for reading or writing the entry. Usually more than one bit of storage is accessed by the same address, and ram devices often have multiple data lines and are said to be “8-bit” or “16-bit”, etc.

In today’s technology, random-access memory takes the form of integrated circuits. Ram is normally associated with volatile types of memory (such as dram modules), where stored information is lost if power is removed, although non-volatile ram has also been developed. Other types of non-volatile memories exist that allow random access for read operations, but either do not allow write operations or have other kinds of limitations on them. These include most types of rom and a type of flash memory called *nor-flash*.

Integrated-circuit ram chips came into the market in the early 1970s, with the

first commercially available dram chip, the intel 1103, introduced in October 1970.

3.2.3 OPERATINGSYSTEM:

An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it. Operating systems are found on many devices that contain a computer – from cellular phones and video game consoles to web servers and supercomputers.

4. MODULES AND ITS DESCRIPTION

4.1 TENSORFLOW:

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

HOW TENSOR FLOW WORKS:

Tensor Flow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation and each connection or edge between nodes is multidimensional data array, or a tensor. Tensor Flow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and Tensor Flow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through Tensor Flow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

TENSOR FLOW BENEFITS:

The single biggest benefit Tensor Flow provides for machine learning development is abstraction. Instead of dealing with the nitty-gritty details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application. Tensor Flow takes care of the details behind the scenes.

Tensor Flow offers additional conveniences for developers who need to debug and gain introspection into Tensor Flow apps. The eager execution mode lets you evaluate and modify each graph operation separately and transparently, instead of constructing the entire graph as a single opaque object and evaluating it all at once. The TensorBoard visualization suite lets you inspect and profile the way graphs run by way of an interactive, web-based dashboard.

.

Some details of TensorFlow's implementation make it hard to obtain totally deterministic model-training results for some training jobs. Sometimes a model trained on one system will vary slightly from a model trained on another, even when they are fed the exact same data. The reasons for this are slippery—e.g., how random numbers are seeded and where, or certain non-deterministic behaviors when using GPUs). That said, it is possible to work around those issues, and TensorFlow's team is considering more controls to affect determinism in a workflow.

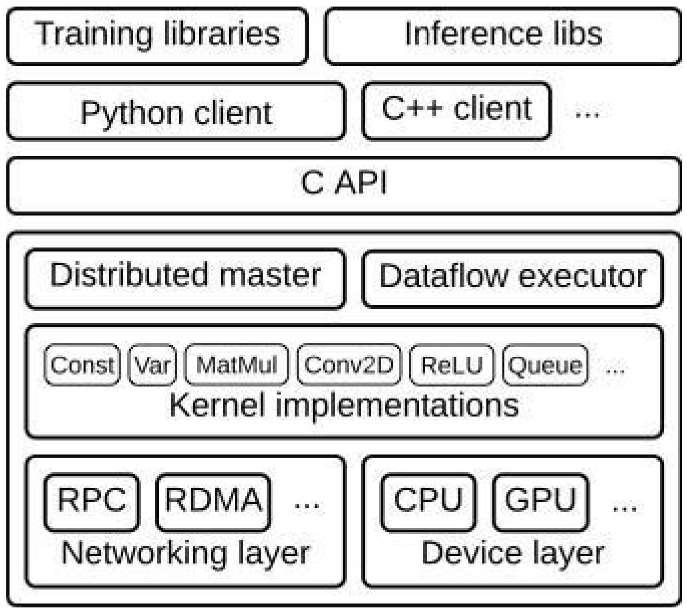


Fig 4.1 Tensorflow architecture.

4.2 **KERAS:**

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

GUIDING PRINCIPLES:

User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
Modularity. A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.
Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

4.3 NUMPY AND PANDAS:

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Num array into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. 16

PANDAS:

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term " panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

4.4 MATPLOTLIB:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Matplotlib was originally written by John D. Hunter, has an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, and further joined by Thomas Caswell.

As of 23 June 2017, matplotlib 2.0.x supports Python versions 2.7 through 3.6. Matplotlib 1.2 is the first version of matplotlib to support Python 3.x. Matplotlib 1.4 is the last version of Matplotlib to support Python 2.6.

Matplotlib has pledged to not support Python 2 past 2020 by signing the Python 3 Statement.

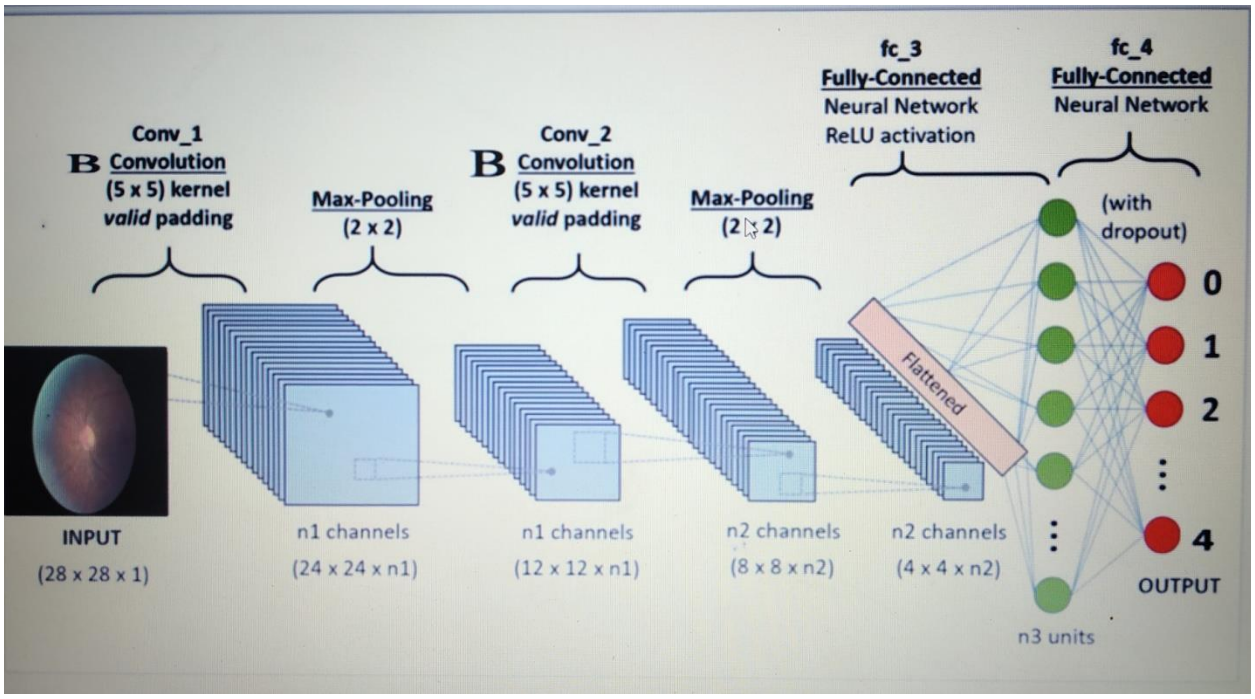
5.BINARIZED NEURAL NETWORK

5.1 UNDERSTANDING BCNN :

In this section, we detail our binarization function, show how we use it to compute the parameters gradients, and how we backpropagate through it.

5.1.1 DETERMINISTIC VS STOCHASTIC BINARIZATION:

When training a BNN, we constrain both the weights and the activations to either +1 or 1. Those two values are very advantageous from a hardware perspective, as we explain in Section 4. In order to transform the real-valued variables into those two values, we use two different binarization functions. Our first binarization function is deterministic:



$$x^b = \mathrm{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise,} \end{cases}$$

where xb is the binarized variable (weight or activation) and x the real-valued variable. It is very straightforward to implement and works quite well in practice. Our second binarization function is stochastic:

$$x^b = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases}$$

where _ is the “hard sigmoid” function:

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2})).$$

The stochastic binarization is more appealing than the sign function, but harder to implement as it requires the hardware to generate random bits when quantizing. As a result,we mostly use the deterministic binarization function(i.e, the sign function), with the exception of activations at train-time in some of our experiments.

5.1.2. GRADIENT COMPUTATION AND ACCUMULATION:

Although our BNN training method uses binary weights and activation to compute the parameters gradients, the real-valued gradients of the weights are accumulated in real-valued variables, as per Algorithm 1. Real-valued weights are likely required for Stochastic Gradient Descent (SGD) to work at all. SGD explores the space of parameters in small and noisy steps, and that noise is averaged out by the stochastic gradient contributions accumulated in each weight. Therefore, it is important to keep sufficient resolution for these accumulators, which at first glance suggests that high precision is absolutely required.

Moreover, adding noise to weights and activations when computing the parameters gradients provide a form of regularization that can help to generalize better, as previously shown with variational weight noise

Dropout and DropConnect (Wan et al., 2013). Our method of training BNNs can be seen as a variant of Dropout, in which instead of randomly setting half of the activations to zero when computing the prameters gradients, we binarize both the activations and the weights.

5.1.3. PROPAGATING GRADIENTS THROUGH DISCRETIZATION:

The derivative of the sign function is zero almost everywhere, making it apparently incompatible with backpropagation, since the exact gradient of the cost with respect to the quantities before the discretization (pre-activations or weights) would be zero. Note that this remains true even
if stochastic quantization is used. Bengio (2013) studied the question of estimating or propagating gradients through stochastic discrete neurons. They found in their experiments that the fastest training was obtained when using the "straight-through estimator," previously introduced in Hinton
(2012)'s lectures. We follow a similar approach but use the version of the straight-through estimator that takes into account the saturation effect, and does use deterministic rather than stochastic sampling of the bit. Consider the sign function quantization

q = Sign(r);

and assume that an estimator gq of the gradient dC/dq has been obtained (with the straight-through estimator when needed). Then, our straight-through estimator of dC/dq is simply

gr = gq1jrj_1

Note that this preserves the gradient's information and cancels the gradient when r is too large. Not cancelling the gradient when r is too large significantly worsens the performance.
Training a BNN. C is the cost function for minibatch, _ - the learning rate decay factor and L the number
of layers. _ indicates element-wise multiplication. The function Binarize() specifies how to (stochastically or deterministically) binarize the activations and weights, and Clip(), how to clip the weights. BatchNorm() specifies how to batch-normalize the activations, using either batch normalization or its shift-based variant we describe in Algorithm. BackBatchNorm() specifies how to backpropagate through the normalization. Update() specifies how to update the parameters when their gradients are known, using either ADAM or the shift-based AdaMax seen as propagating the gradient through hard tanh, which is the following piece-wise linear activation function:

Htanh(x) = Clip(x;□1; 1) = max(□1; min(1; x))

For hidden units, we use the sign function non-linearity to obtain binary activations, and for weights we combine two ingredients:

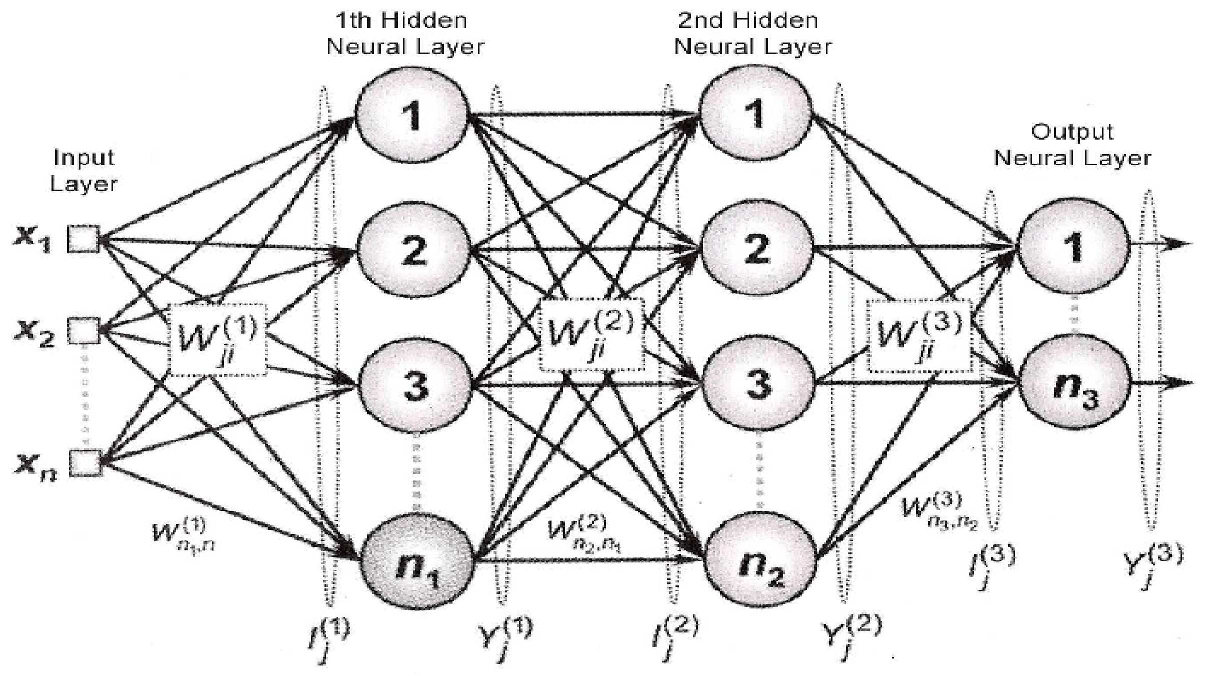
- Constrain each real-valued weight between -1 and 1,by projecting wr to -1 or 1 when the weight update brings wr outside of [□1; 1], i.e., clipping the weights during training. The real-valued weights would otherwise grow very large without any impact on the binary weights.
- When using a weight wr, quantize it using wb = Sign(wr).

This is consistent with the gradient canceling when |wr| > 1

5.1.4. SHIFT BASED BATCH NORMALIZATION:

Batch Normalization (BN) ,accelerates the training and also seems to reduces the overall impact of the weights' scale. The normalization noise may also help to regularize the model. However, at train-time, BN requires many multiplications (calculating the standard deviation and dividing by it), namely, dividing by the running variance (the weighted mean of the training set activation variance). Although the number of scaling calculations is the same as the number of neurons, in the case of ConvNets this number is quite large. For example, in the CIFAR-10 dataset (using our architecture), the first convolution layer, consisting of only 128x3x3 filter masks, converts an image of size 3x32x32 to size 3x128x28x28,which is two orders of magnitude larger than the number of weights. To achieve the results that BN would obtain, we use a shift-based batch normalization (SBN) technique. Detailed. SBN approximates BN almost without multiplications. In the experiment we conducted we did not observe accuracy loss when using the shift based BN algorithm instead of the vanilla BN algorithm.

Backward pass←



→Forward pass

5.1.5. SHIFT BASED ADAMAX:

The ADAM learning rule also seems to reduce the impact of the weight scale. Since ADAM requires many multiplications, we suggest using instead the shift-based AdaMax we detail in Algorithm 4. In the experiment we conducted we did not observe accuracy loss when using the shift-based AdaMax algorithm instead of the vanilla ADAM algorithm.

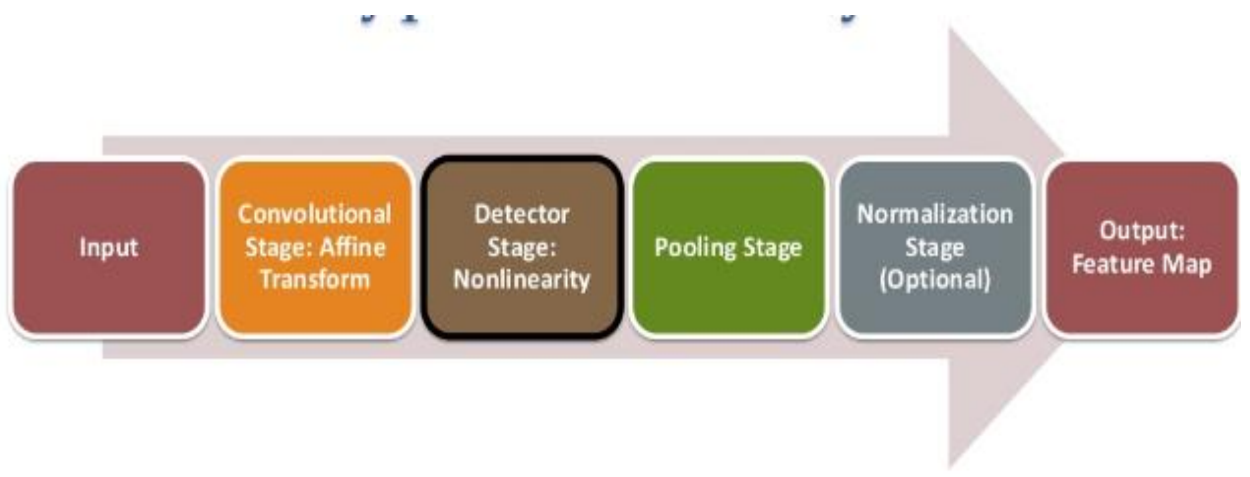
5.1.6. FIRST LAYER:

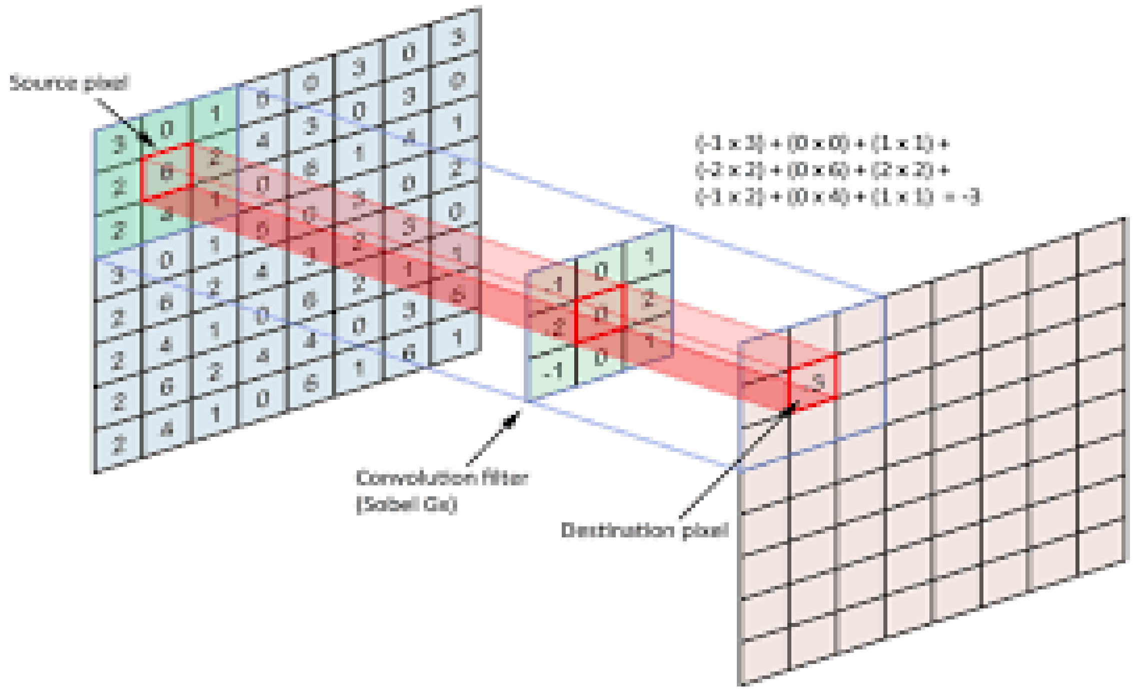
In a BNN, only the binarized values of the weights and activations are used in all calculations. As the output of one layer is the input of the next, all the layers inputs are binary, with the exception of the first layer. However, we do not believe this to be a major issue. First, in computer vision, the input representation typically has much fewer channels (e.g, Red, Green and Blue) than internal representations (e.g, 512). As a result, the first layer of a ConvNet is often the smallest convolution layer, both in terms of parameters and computations. Second, it is relatively easy to handle continuous-valued
inputs as fixed point numbers, withmbits of precision. For example, in the common case of 8-bit fixed point inputs:

$$s = x \cdot w^b$$

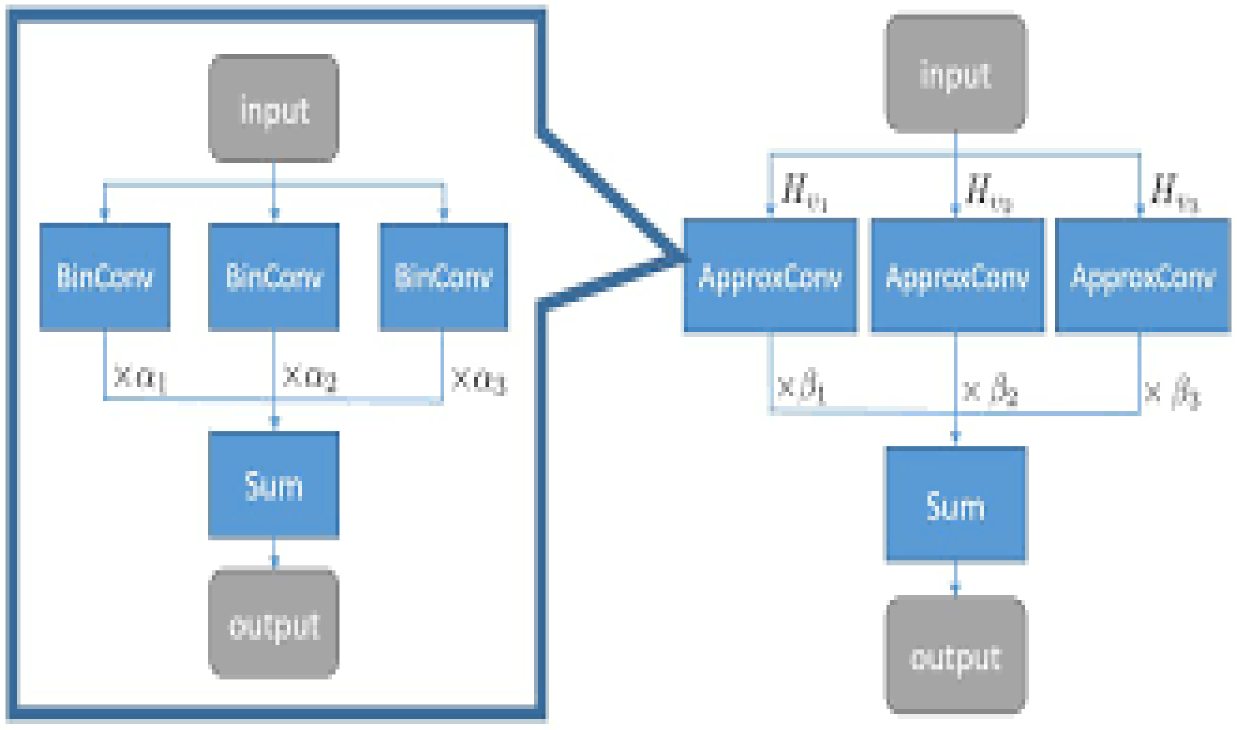
$$s = \sum_{n=1}^{8} 2^{n-1}(x^n \cdot w^b),$$

where x is a vector of 1024 8-bit inputs, x81 is the most significant bit of the first input, wb is a vector of 1024 1-bit weights, and s is the resulting weighted sum.


Input
Convolutional Stage: Affine Transform
Detector Stage: Nonlinearity
Pooling Stage
Normalization Stage (Optional)
Output: Feature Map


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter (Sobel Gx)
Destination pixel

BCNNs:



CRUX IN BCCN:

In BNN, we constrain both the weights and the activations to either +1 or -1.

In order to transform the real-valued variables into those two values, we use deterministic binarization function:

+1 if x >= 0;

Sign(x) =

-1 otherwise; x : real-valued variable

Computer hardware, be it general-purpose or specialized, is composed of memories, arithmetic operators and control logic.

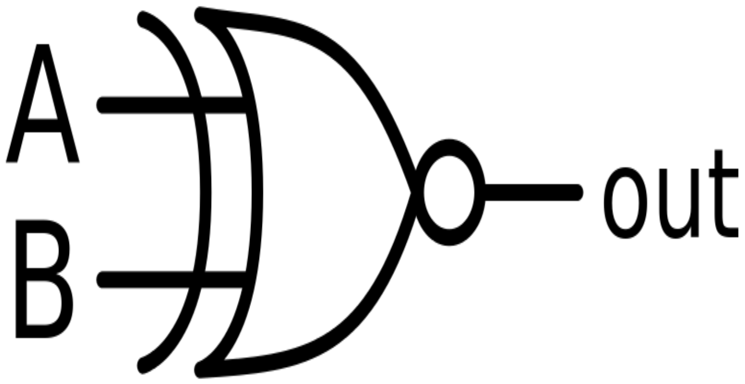
During the forward pass (both at run-time and train time), BNNs drastically reduce memory size and accesses, and replace most arithmetic operations with bit-wise operations, which might lead to a great increase in power efficiency.

Moreover, a binarized CNN can lead to binary convolution kernel repetitions, and we argue that dedicated hardware could reduce the time complexity by 60% .

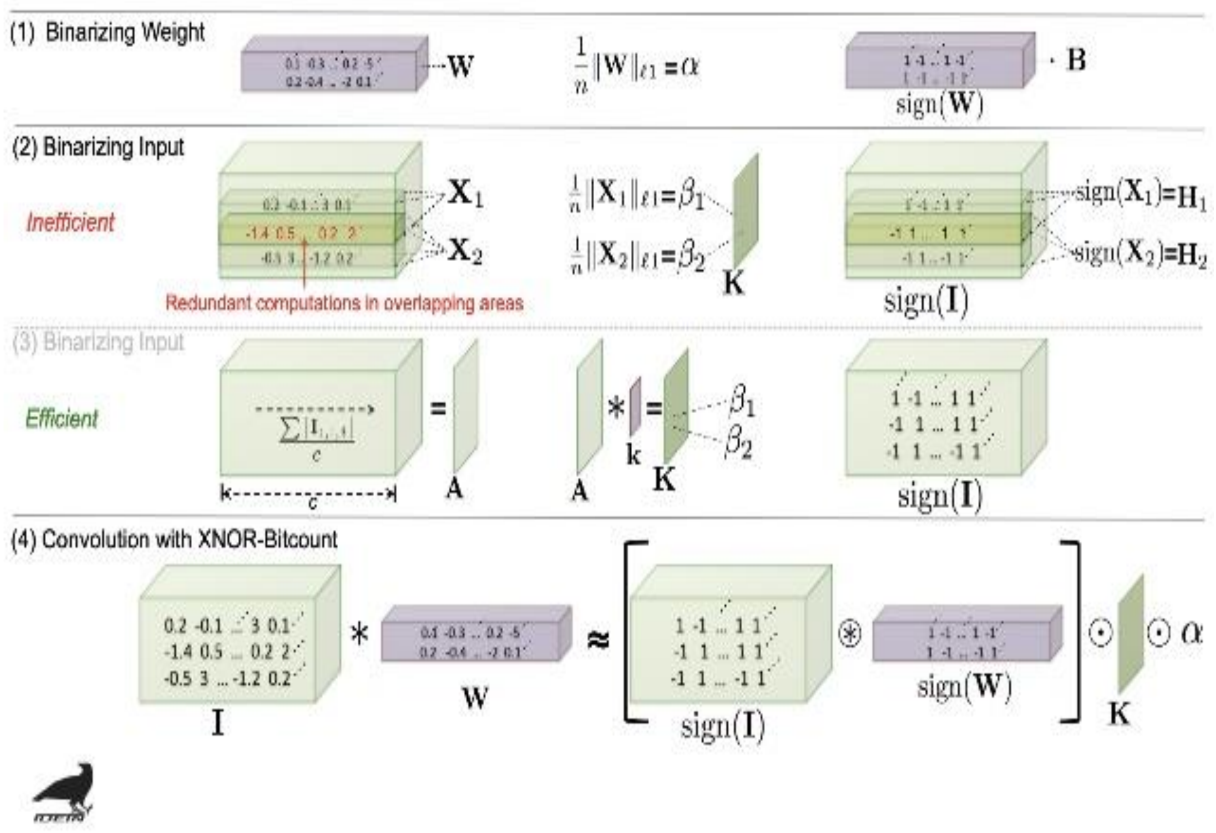
XNOR-COUNT:

Applying a DNN mainly consists of convolutions and matrix multiplications. The key arithmetic operation of deep learning is thus the multiply-accumulate operation. Artificial neurons are basically multiply-accumulators computing weighted sums of their inputs. In BNNs, both the activations and the weights are constrained to either -1 or +1. As a result, most of the 32-bit floating point multiply accumulations are replaced by 1-bit XNOR-count operations. This could have a big impact on deep learning dedicated hardware. For instance, a 32-bit floating point multiplier costs about 200 Xilinx FPGA slices only costs a single slice.

Applying a DNN mainly consists of convolutions and matrix multiplications. The key arithmetic operation of deep learning is thus the multiply-accumulate operation. Artificial neurons are basically multiply-accumulators computing weighted sums of their inputs.
In BNNs, both the activations and the weights are constrained to either -1 or +1. As a result most of the 32-bit floating point multiply accumulations are replaced by1-bit XNOR-count operations.



XNOR-Net: Convolution



STOCHASTIC GRADIENT DESCENT:

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization. A 2018 article implicitly credits Herbert Robbins and Sutton Monro for developing SGD in their 1951 article titled "A Stochastic Approximation Method"; see Stochastic approximation for more information. It is called stochastic because samples are selected randomly (or shuffled) instead of as a single group (as in standard gradient descent) or in the order they appear in the training set.

EXTENSIONS AND VARIANTS:

Many improvements on the basic stochastic gradient descent algorithm have been proposed and used. In particular, in machine learning, the need to set a learning rate (step size) has been recognized as problematic. Setting this parameter too high can cause the algorithm to diverge; setting it too low makes it slow to converge. A conceptually simple extension of stochastic gradient descent makes the learning rate a decreasing function ηt of the iteration number t, giving a learning rate schedule, so that the first iterations cause large changes in the parameters,.

IMPLICIT UPDATES:

As mentioned earlier, classical stochastic gradient descent is generally sensitive to learning rate η. Fast convergence requires large learning rates but this may induce numerical instability. The problem can be largely solved by considering implicit updates whereby the stochastic gradient is evaluated at the next iterate rather than the current one:

MOMENTUM:

Stochastic gradient descent with momentum remembers the update w at each iteration, and determines the next update as a linear combination of the gradient and the previous update**:**

AVERAGING:

Averaged stochastic gradient descent, invented independently by Ruppert and Polyak in the late 1980s, is ordinary stochastic gradient descent that records an average of its parameter vector over time. That is, the update is the same as for ordinary stochastic gradient descent, but the algorithm also keeps track of updates.

ADAGRAD:

AdaGrad (for adaptive gradient algorithm) is a modified stochastic gradient descent with per-parameter learning rate, first published in 2011. Informally, this increases the learning rate for more sparse parameters and decreases the learning rate for fewer sparse ones. This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative. Examples of such applications include natural language processing and image recognition. It still has a base learning rate η, but this is multiplied with the elements of a vector {Gj,j} which is the diagonal of the outer product matrix.

RMSPROP:

RMSProp (for Root Mean Square Propagation) is also a method in which the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight. So, first the running average is calculated in terms of means square.

NATURAL GRADIENT DESCENT AND KSGD:

Kalman-based Stochastic Gradient Descent (kSGD) is an online and offline algorithm for learning parameters from statistical problems from quasi-likelihood models, which include linear models, non-linear models, generalized linear models, and neural networks with squared error loss as special cases. For online learning problems, kSGD is a special case of the Kalman Filter for linear regression problems, a special case of the Extended Kalman Filter for non-linear regression problems, and can be viewed as an incremental Gauss-Newton method. Moreover, because of kSGD's relationship to the Kalman Filter and natural gradient descent's relationship to the Kalman Filter, kSGD is a rigorous improvement over the popular natural gradient descent method.

The benefits of kSGD, in comparison to other methods, are (1) it is not sensitive to the condition number of the problem , (2) it has a robust choice of hyperparameters, and (3) it has a stopping condition. The drawbacks of kSGD is that the algorithm requires storing a dense covariance matrix between iterations, and requires a matrix-vector product at each iteration.

Backpropagation:

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network’s layers. It is commonly used to train deep neural networks.

Backpropagation is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. It is closely related to the Gauss–Newton algorithm and is part of continuing research in neural backpropagation.

LOSS FUNCTION:

The loss function is a function that maps values of one or more variables onto a real number intuitively representing some "cost" associated with those values. For backpropagation, the loss function calculates the difference between the network output and its expected output, after a training example has propagated through the network.

BENEFITS OF BCNN:

- Implement both training and inference using only very simple arithmetic operations(addition, XNOR) .
- Faster Than conventional models.
- Consumes less energy.
- Less space on chip.

6. DATA SET

DATA SET:

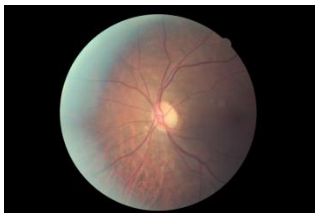
Most public datasets were constructed particularly for automatic feature extraction of DR. Specifically, we used Kaggle datasets that contain labels of lesion areas and grades in this section.

KAGGLE:

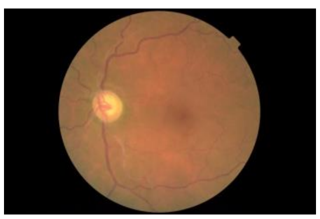
Kaggle19 is an open competition dataset provided by EyePACS which is acquired using multiple fundus cameras at different FOVs[field of view].. The kaggle dataset consists of 35,126 training images with five DR grade labels and 53,576 test images without grade labels. The image quality in this dataset is not uniform, which can be used to improve the generalization of a classifying algorithm.

The colorizing imagesaredownloadedfromtheKaggle website3. The training dataset contains 35126 high resolution images under a variety of imaging conditions. These retina images were obtained from a group of subjects, and foreachsubjecttwoimageswereobtainedforleftandright eyes, respectively. The labels were provided by clinicians who rated the presence of diabetic retinopathy in each image by a scale of “0, 1, 2, 3, 4”, which represent “no DR”, “mild”, “moderate”, “severe”, “proliferative DR” respectively. As mentioned in the description of the dataset, the imagesinthedatasetcomefromdifferentmodelsandtypes of camera, which can affect the visual appearance of leftvs. right. The samples images are shown in Fig 2. Also, thedatasetdoesn’thavetheequaldistributionsamongthe5 scales. As one can expect, normal data with label “0” is the biggest class in the whole dataset, while “proliferative DR” data is the smallest classshows counts of images for different scales in the training dataset.

Class 0 :Normal eye

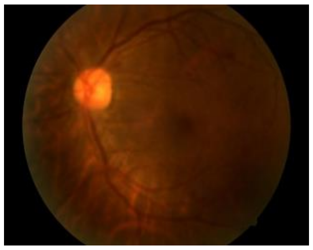


Class 1 :Mild DR eye

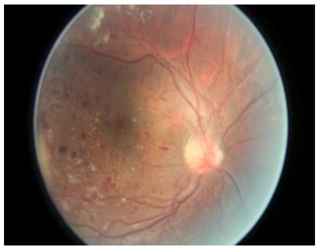


Class 2 :Moderate DR eye

Class 3: Severe DR eye

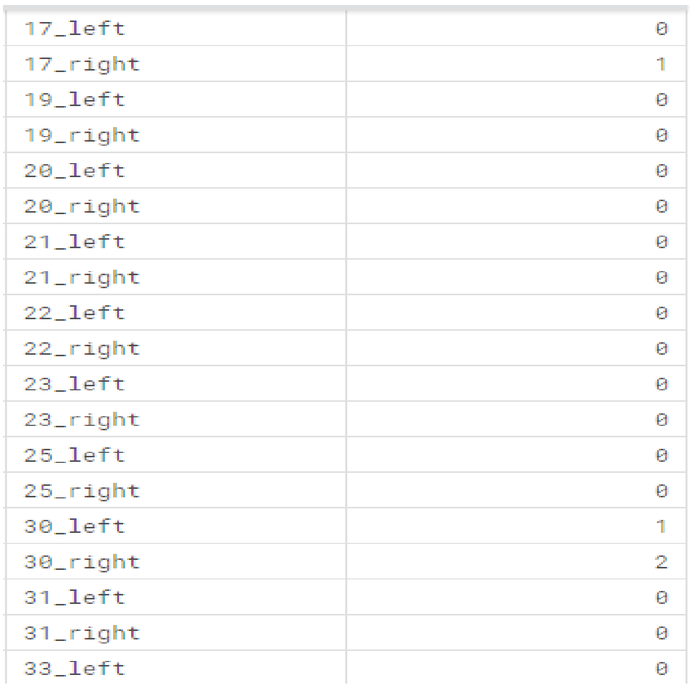


Class 4 :Proliferative eye

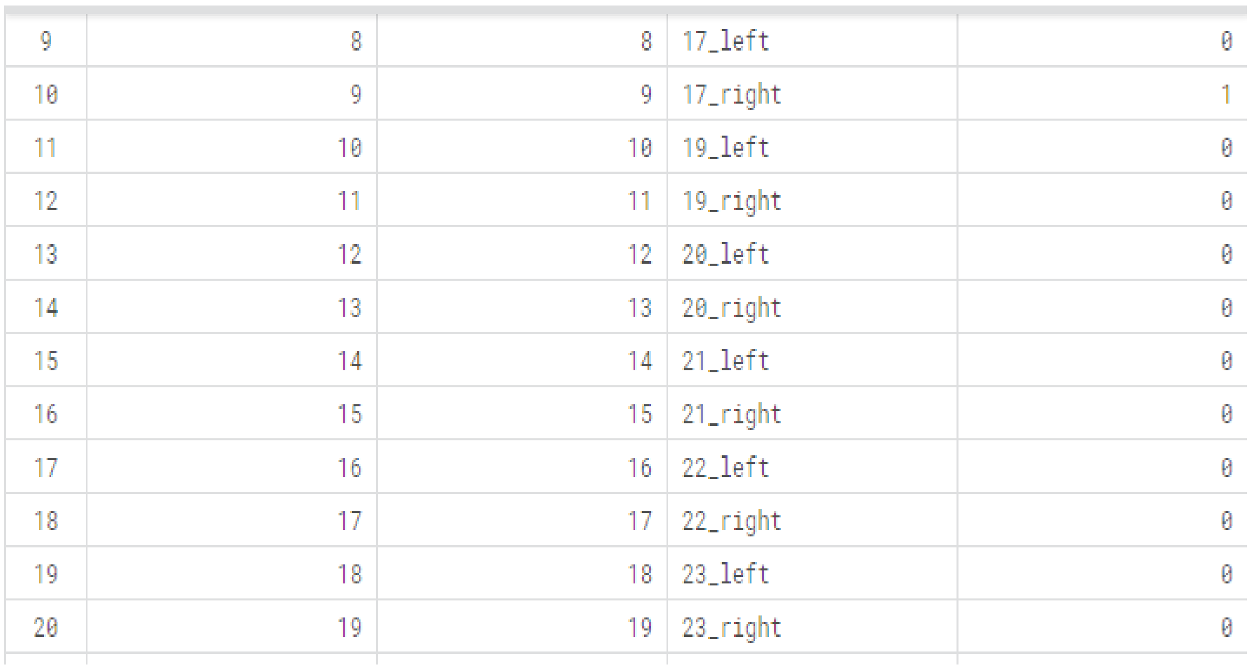


FILES DESCRIPTIONS:

- train.zip.* - the training set (5 files total)
- test.zip.* - the test set (7 files total)
- sample.zip - a small set of images to preview the full dataset
- sampleSubmission.csv - a sample submission file in the correct format
- trainLabels.csv - contains the scores for the training set



17_left	0
17_right	1
19_left	0
19_right	0
20_left	0
20_right	0
21_left	0
21_right	0
22_left	0
22_right	0
23_left	0
23_right	0
25_left	0
25_right	0
30_left	1
30_right	2
31_left	0
31_right	0
33_left	0



9	8	8	17_left	0
10	9	9	17_right	1
11	10	10	19_left	0
12	11	11	19_right	0
13	12	12	20_left	0
14	13	13	20_right	0
15	14	14	21_left	0
16	15	15	21_right	0
17	16	16	22_left	0
18	17	17	22_right	0
19	18	18	23_left	0
20	19	19	23_right	0

7.IMPLEMENTATION .

7.1 SYSTEM DEVELOPMENT ENVIRONMENT:

What is Google Colab?

Google Colab is a free cloud service and now it supports free GPU!

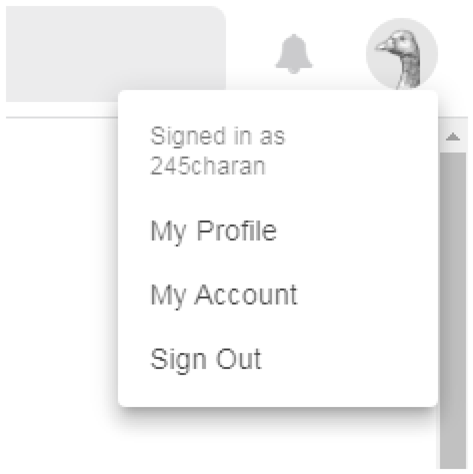
You can:

- improve your **Python** programming language coding skills.
- develop deep learning applications using popular libraries such as **Keras**, **TensorFlow**, **PyTorch,** and **OpenCV**.

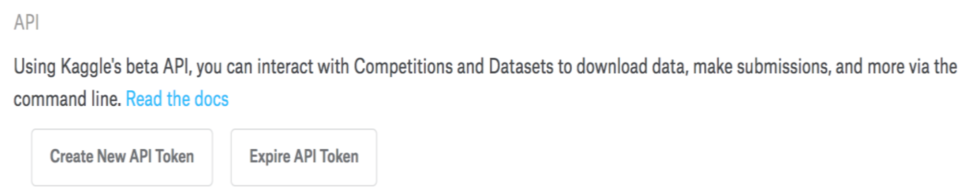
The most important feature that distinguishes Colab from other free cloud services is; **Colab** provides GPU and is totally free.

Step 1: Get Kaggle API Token

Login to your Kaggle account and under “My Account”, navigate to “Create New API Token”. Click the button to download your API token as a json file.



Navigate to your account, using the profile pic on the top right.


API

Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. Read the docs

Click the "Create New API Token" button to download API Token as a json file.

Step 2: Install Kaggle library and Import Google Collab File Library

Use the following code in your Google Collab Notebook.

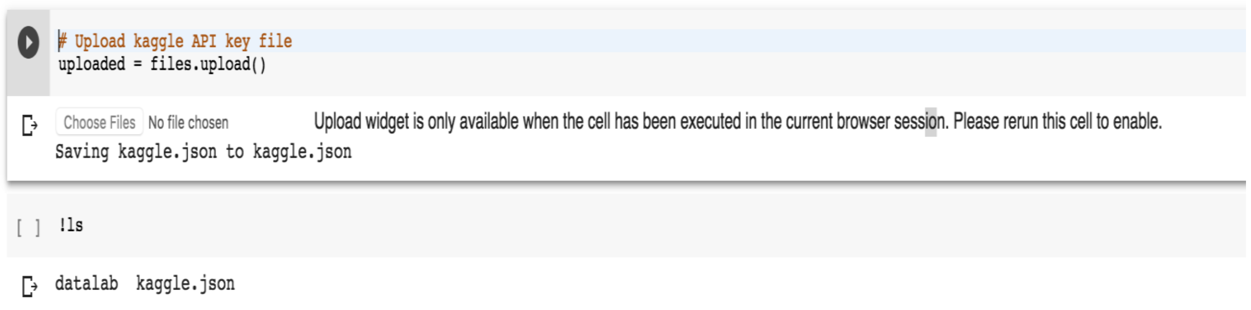
```
# Colab library to upload files to notebook

From goolge.colab import files

#Install kaggle library

!pip install –q kaggle
```

Step 3: Upload Kaggle API json file to Google Colab

The code below will prompt you with a button to upload files to Google Colab. Use this to navigate to the location of the downloaded Kaggle API Token json file and upload it.

```
# Upload kaggleAPI key file
uploaded =files.upload()
```

PS: You could use this to upload files directly from your local machine to the notebook!



```
# Upload kaggle API key file
uploaded = files.upload()
```

```
Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
```

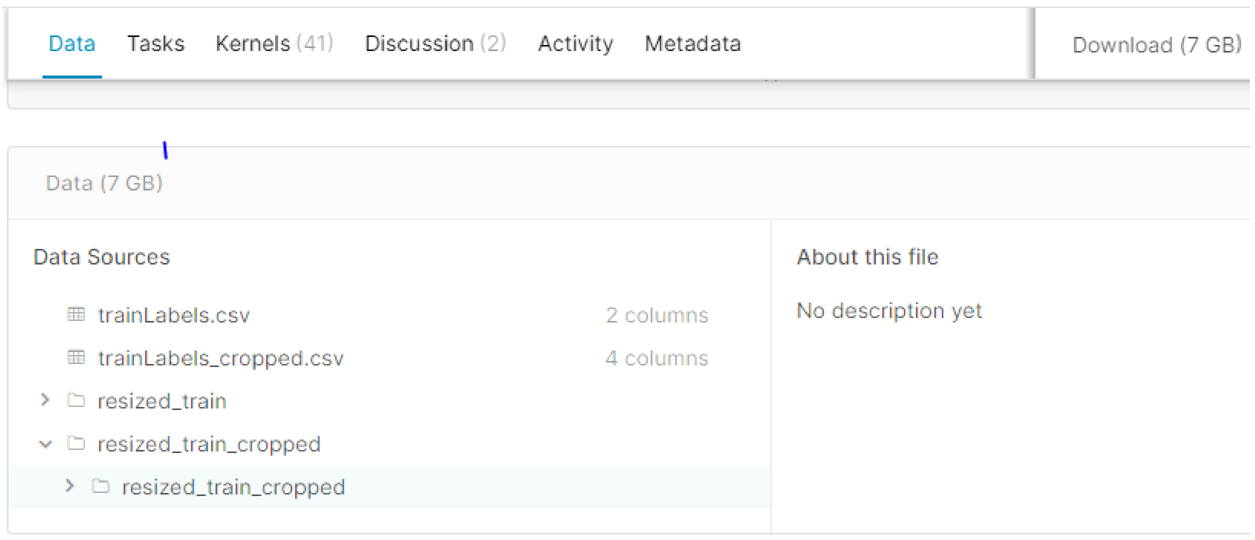
```
!ls
```

```
datalab  kaggle.json
```

Execute the above code, select the json file and upload it. Use "!ls" to see all the files in the directory.

Step 4: Download dataset from Kaggle

Now, go to the kaggle competition dataset you are interested in, navigate to the Data tab, and copy the API link and paste in Colab to download the dataset.NYC Taxi Trip Duration Competion on Kaggle. Copy the command in the API box and execute in Colab.

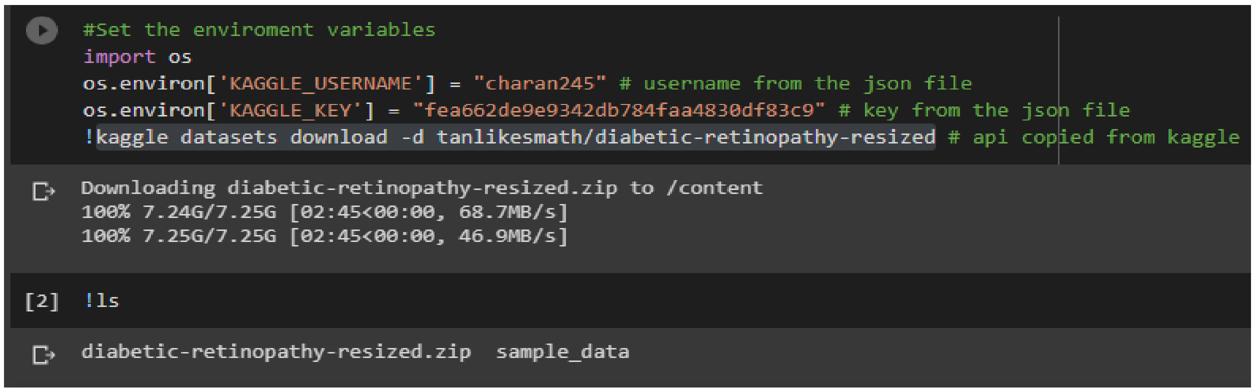


# Downlaod data for the DR detection

!kaggle datasets download -d tanlikesmath/diabetic-retinopathy-resized

***Note**: As you can tell, just replace the competition name in the code (after download -d) with the dataset you are interested in.*

You should see something like this on your Colab notebook:



```
#Set the enviroment variables
import os
os.environ['KAGGLE_USERNAME'] = "charan245" # username from the json file
os.environ['KAGGLE_KEY'] = "fea662de9e9342db784faa4830df83c9" # key from the json file
!kaggle datasets download -d tanlikesmath/diabetic-retinopathy-resized # api copied from kaggle
```

```
Downloading diabetic-retinopathy-resized.zip to /content
100% 7.24G/7.25G [02:45<00:00, 68.7MB/s]
100% 7.25G/7.25G [02:45<00:00, 46.9MB/s]
```

```
[2] !ls
```

```
diabetic-retinopathy-resized.zip  sample_data
```

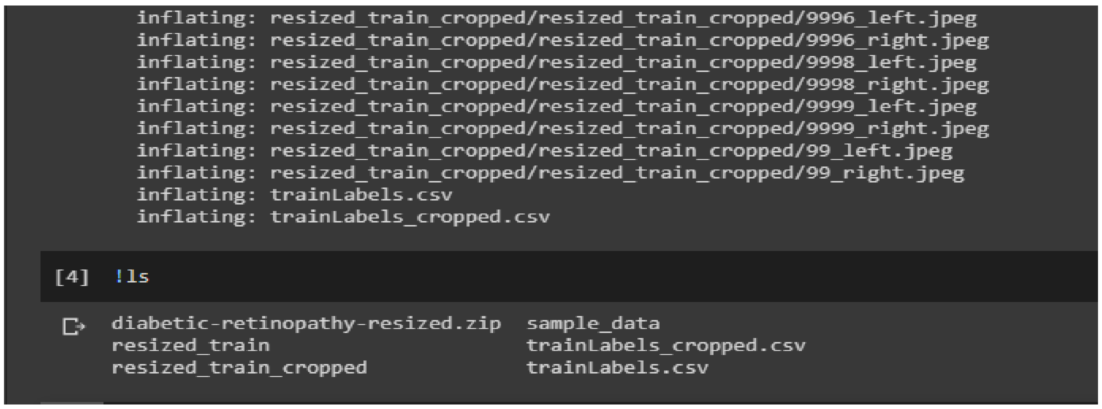
Diabetes Retinopathy fundus images dataset downloaded from Kaggle. Notice how I use "!ls" to list all the files in my noteboook.

7.2 UNZIP DATA:

Unzip datasets and load to Pandas dataframe

Finally, let's load the the datasets into pandas. Since the train and test datasets are in .zip format, we will need to unzip them before reading from the .csv files. Luckily for us, Pandas does it all!

!unzip /content/diabetic-retinopathy-resized.zip



```
inflating: resized_train_cropped/resized_train_cropped/9996_left.jpeg
inflating: resized_train_cropped/resized_train_cropped/9996_right.jpeg
inflating: resized_train_cropped/resized_train_cropped/9998_left.jpeg
inflating: resized_train_cropped/resized_train_cropped/9998_right.jpeg
inflating: resized_train_cropped/resized_train_cropped/9999_left.jpeg
inflating: resized_train_cropped/resized_train_cropped/9999_right.jpeg
inflating: resized_train_cropped/resized_train_cropped/99_left.jpeg
inflating: resized_train_cropped/resized_train_cropped/99_right.jpeg
inflating: trainLabels.csv
inflating: trainLabels_cropped.csv

[4]  !ls

diabetic-retinopathy-resized.zip  sample_data
resized_train                     trainLabels_cropped.csv
resized_train_cropped             trainLabels.csv
```

```
#Set the enviroment variables
import os
os.environ['KAGGLE_USERNAME'] = "charan245" # username from the json file
os.environ['KAGGLE_KEY'] = "0a55b6aaf99c98b7c675644c28b72965" # key from the
json file
!kaggle datasets download -d tanlikesmath/diabetic-retinopathy-
resized # api copied from kaggle
```

```
Downloading diabetic-retinopathy-resized.zip to /content
100% 7.25G/7.25G [02:44<00:00, 68.8MB/s]
100% 7.25G/7.25G [02:44<00:00, 47.4MB/s]
```

```
!unzip /content/diabetic-retinopathy-resized.zip
!ls
```

```
inflating:           resized_train_cropped/resized_train_cropped/998_left.jpeg
inflating:          resized_train_cropped/resized_train_cropped/998_right.jpeg
inflating:          resized_train_cropped/resized_train_cropped/9992_left.jpeg
inflating:         resized_train_cropped/resized_train_cropped/9992_right.jpeg
inflating:          resized_train_cropped/resized_train_cropped/9993_left.jpeg
inflating:         resized_train_cropped/resized_train_cropped/9993_right.jpeg
inflating:          resized_train_cropped/resized_train_cropped/9996_left.jpeg
inflating:         resized_train_cropped/resized_train_cropped/9996_right.jpeg
inflating:          resized_train_cropped/resized_train_cropped/9998_left.jpeg
inflating:         resized_train_cropped/resized_train_cropped/9998_right.jpeg
inflating:          resized_train_cropped/resized_train_cropped/9999_left.jpeg
inflating:         resized_train_cropped/resized_train_cropped/9999_right.jpeg
inflating:            resized_train_cropped/resized_train_cropped/99_left.jpeg
inflating: resized_train_cropped/resized_train_cropped/99_right.jpeg
inflating: trainLabels.csv
```

```
inflating: trainLabels_cropped.csv
diabetic-retinopathy-resized.zip           sample_data           resized_train
trainLabels_cropped.csv resized_train_cropped trainLabels.csv
```

```
import tensorflow as tf
tf.__version__
```

```
'1.15.0'
```

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 13846401428320994031, name: "/device:XLA_CPU:0"
 device_type: "XLA_CPU"
 memory_limit: 17179869184
 locality {
 }
 incarnation: 2984170486963799663
 physical_device_desc: "device: XLA_CPU device", name: "/device:XLA_GPU:0"
 device_type: "XLA_GPU"
 memory_limit: 17179869184
 locality {
 }
 incarnation: 7228925496580143504
 physical_device_desc: "device: XLA_GPU device", name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 11330115994
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 14698679184640823024
 physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:00:04.0,
compute capability: 3.7"]
```

```
import numpy as np # linear algebra
import pandas as pd # data processing,CSV file I/O(e.g.pd.read_csv)
import matplotlib.pyplot as plt # showing and rendering figures
# io related
from skimage.io import imread
import os
from glob import glob
# not needed in Kaggle, but required in Jupyter
%matplotlib inline
```

7.3 DATA INSPECTION:

To ensure that you're dealing with the right information you need a clear view of your data at every stage of the transformation process.

Data inspection meets this need: it is the act of viewing data for verification and debugging purposes, before, during, or after a translation.

```
import os
files = os.listdir('resized_train/resized_train')
print('Total images in resized dataset: ',len(files))
```

```
Total images in resized dataset: 35126
```

```
35108 images found of 35126 total
Using TensorFlow backend.
```

7.4COMBINE IMAGE AND CSV:

```
retina_df = pd.read_csv('trainLabels.csv')
retina_df['PatientId'] = retina_df['image'].map(lambda x: x.split('_')[0])
retina_df['path'] = retina_df['image'].map(lambda x: os.path.join('/content/r
esized_train_cropped/resized_train_cropped','{}.jpeg'.format(x)))
retina_df['exists'] = retina_df['path'].map(os.path.exists)
print(retina_df['exists'].sum(), 'images found of', retina_df.shape[0], 'tota
l')
# left eye = 1, right eye = 0
retina_df['eye'] = retina_df['image'].map(lambda x: 1 if x.split('_')[-
1]=='left' else 0)
# convert to categorical
from keras.utils.np_utils import to_categorical
retina_df['level_cat'] = retina_df['level'].map(lambda x: to_categorical(x, 1
+retina_df['level'].max()))

retina_df.dropna(inplace = True)
retina_df = retina_df[retina_df['exists']]
retina_df.head()
```

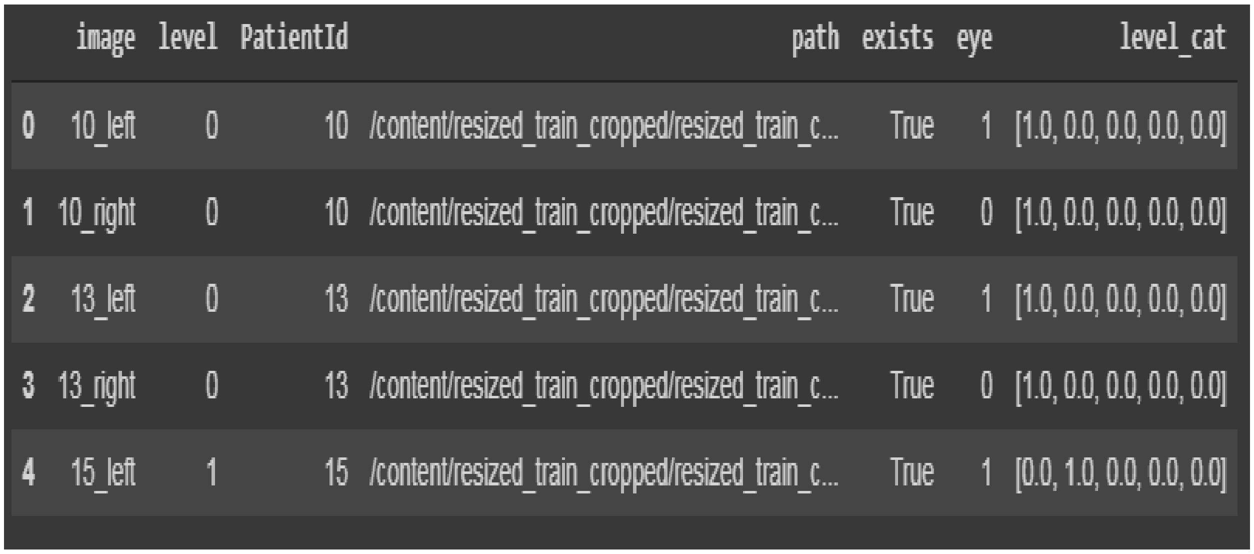
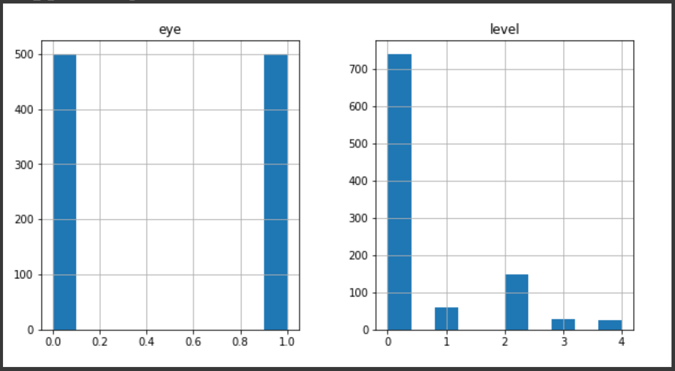


	image	level	PatientId	path	exists	eye	level_cat
0	10_left	0	10	/content/resized_train_cropped/resized_train_c...	True	1	[1.0, 0.0, 0.0, 0.0, 0.0]
1	10_right	0	10	/content/resized_train_cropped/resized_train_c...	True	0	[1.0, 0.0, 0.0, 0.0, 0.0]
2	13_left	0	13	/content/resized_train_cropped/resized_train_c...	True	1	[1.0, 0.0, 0.0, 0.0, 0.0]
3	13_right	0	13	/content/resized_train_cropped/resized_train_c...	True	0	[1.0, 0.0, 0.0, 0.0, 0.0]
4	15_left	1	15	/content/resized_train_cropped/resized_train_c...	True	1	[0.0, 1.0, 0.0, 0.0, 0.0]

7.5 DATA DISTRIBUTION:

```
retina_df[['level', 'eye']].hist(figsize = (10, 5))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7feed72bcc88>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7feec0565978>]],
dtype=object)
```

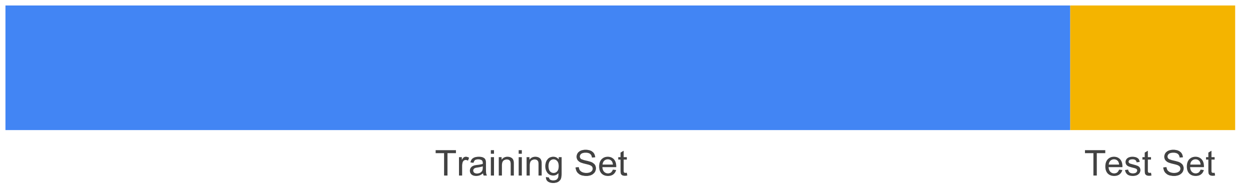


7.6TRAINING AND VALIDATION SPLIT:

The module introduces the idea of dividing your data set into two subsets:

- **training set**—a subset to train a model.
- **test set**—a subset to test the trained model.

You could imagine slicing the single data set as follows:

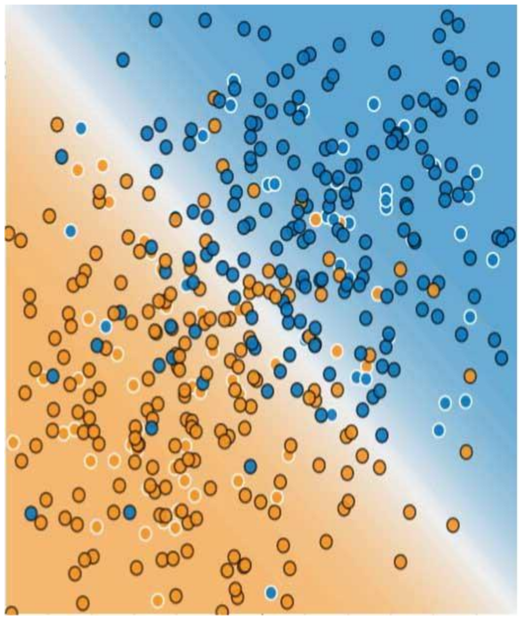


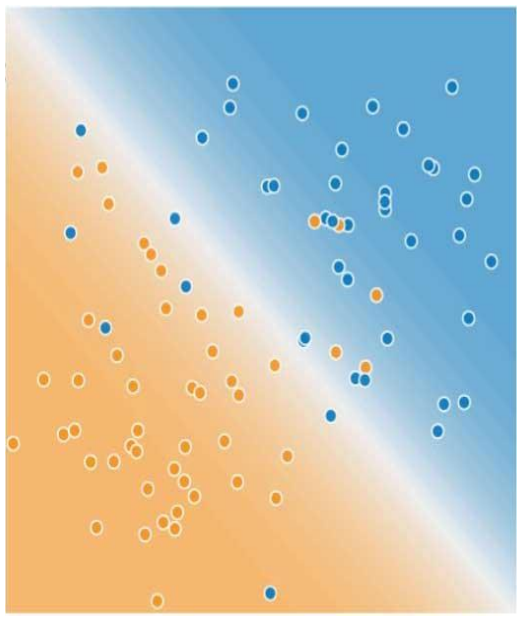
1. Slicing a single data set into a training set and test set.

Make sure that your test set meets the following two conditions:

- Is large enough to yield statistically meaningful results.
- Is representative of the data set as a whole. In other words, don't pick a test set with different characteristics than the training set.

Assuming that your test set meets the preceding two conditions, your goal is to create a model that generalizes well to new data. Our test set serves as a proxy for new data. For example, consider the following figure. Notice that the model learned for the training data is very simple. This model doesn't do a perfect job—a few predictions are wrong. However, this model does about as well on the test data as it does on the training data. In other words, this simple model does not overfit the training data.


Training Data


Test Data

2. Validating the trained model against test data.

Never train on test data: If you are seeing surprisingly good results on your evaluation metrics, it might be a sign that you are accidentally training on the test set. For example, high accuracy might indicate that test data has leaked into the training set.We apportion the data into training and test sets, with an 75-25 split

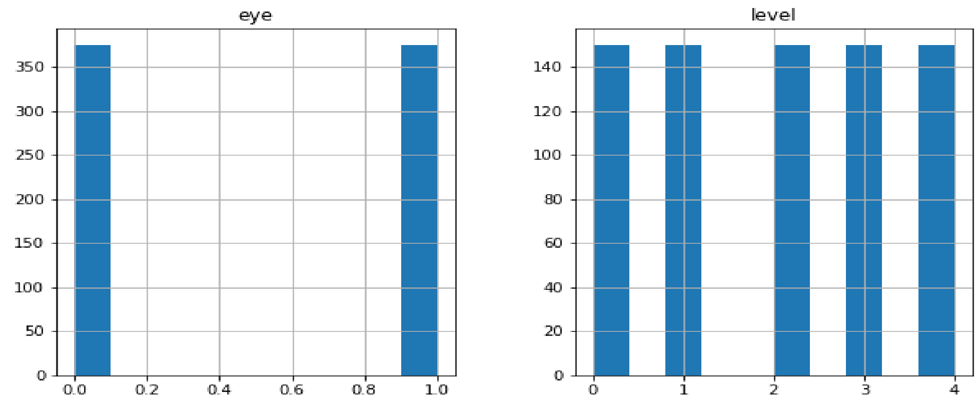
```
from sklearn.model_selection import train_test_split
rr_df = retina_df[['PatientId', 'level']].drop_duplicates()
train_ids, valid_ids = train_test_split(rr_df['PatientId'],
                                        test_size = 0.25,
                                        random_state = 2018,
                                        stratify = rr_df['level'])
raw_train_df = retina_df[retina_df['PatientId'].isin(train_ids)]
valid_df = retina_df[retina_df['PatientId'].isin(valid_ids)]
print('train', raw_train_df.shape[0], 'validation', valid_df.shape[0])
```

```
train 27138 validation 9582
```

7.6.1 BALANCE DISTRIBUTION:

```
train_df = raw_train_df.groupby(['level', 'eye']).apply(lambda x: x.sample(75
, replace = True)).reset_index(drop = True)
print('New Data Size:', train_df.shape[0],'Old Size:', raw_train_df.shape[0])
train_df[['level', 'eye']].hist(figsize = (10, 5))
```

```
New Data Size: 750 Old Size: 27138
```



7.7 DATA AGUMENTATION:

In this section, we present some basic but powerful augmentation techniques that are popularly used. Before we explore these techniques, **for simplicity**, let us make **one assumption**. The assumption is that, **we don't need to consider what lies beyond the image's boundary**. We'll use the below techniques such that our assumption is valid.

What would happen if we use a technique that forces us to guess what lies beyond an image's boundary? In this case, we need to **interpolate** some information. We'll discuss this in detail after we cover the types of augmentation.

For each of these techniques, we also specify the factor by which the size of your dataset would get increased (aka. Data Augmentation Factor).

1. Flip

You can flip images horizontally and vertically. Some frameworks do not provide function for vertical flips. But, a vertical flip is equivalent to rotating an image by 180 degrees and then performing a horizontal flip. Below are examples for images that are flipped.

From the left, we have the original image, followed by the image flipped horizontally, and then the image flipped vertically.

You can perform flips by using any of the following commands, from your favorite packages. **Data Augmentation Factor = 2 to 4x**

You can perform flips by using any of the following commands, from your favorite packages. Data Augmentation Factor = 2 to 4x

```
# NumPy.'img' = A single image.
flip_1 = np.fliplr(img)
# TensorFlow. 'x' = A placeholder for an image.
shape = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
flip_2 = tf.image.flip_up_down(x)
flip_3 = tf.image.flip_left_right(x)
flip_4 = tf.image.random_flip_up_down(x)
flip_5 = tf.image.random_flip_left_right(x)
```

2. Rotation

One key thing to note about this operation is that image dimensions may not be preserved after rotation. If your image is a square, rotating it at right angles will preserve the image size. If it’s a rectangle, rotating it by 180 degrees would preserve the size. Rotating the image by finer angles will also change the final image size. We’ll see how we can deal with this issue in the next section. Below are examples of square images rotated at right angles.

The images are rotated by 90 degrees clockwise with respect to the previous one, as we move from left to right.

You can perform rotations by using any of the following commands, from your favorite packages. **Data Augmentation Factor = 2 to 4x**

```
# Placeholders: 'x' = A single image, 'y' = A batch of images
# 'k' denotes the number of 90 degree anticlockwise rotations
shape = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
rot_90 = tf.image.rot90(img, k=1)
rot_180 = tf.image.rot90(img, k=2)
# To rotate in any angle. In the example below, 'angles' is in radians
shape = [batch, height, width, 3]
y = tf.placeholder(dtype = tf.float32, shape = shape)
rot_tf_180 = tf.contrib.image.rotate(y, angles=3.1415)
```

```
# Scikit-Image. 'angle' = Degrees. 'img' = Input Image
# For details about 'mode', checkout the interpolation section below.
rot = skimage.transform.rotate(img, angle=45, mode='reflect')
```

3. Scale

The image can be scaled outward or inward. While scaling outward, the final image size will be larger than the original image size. Most image frameworks cut out a section from the new image, with size equal to the original image. We'll deal with scaling inward in the next section, as it reduces the image size, forcing us to make assumptions about what lies beyond the boundary. Below are examples or images being scaled.

From the left, we have the original image, the image scaled outward by 10%, and the image scaled outward by 20%

You can perform scaling by using the following commands, using scikit-image. **Data Augmentation Factor = Arbitrary.**

```
# Scikit Image. 'img' = Input Image, 'scale' = Scale factor
# For details about 'mode', checkout the interpolation section below.
scale_out = skimage.transform.rescale(img, scale=2.0, mode='constant')
scale_in = skimage.transform.rescale(img, scale=0.5, mode='constant')
# Don't forget to crop the images back to the original size (for
# scale_out)
```

4. Crop

Unlike scaling, we just randomly sample a section from the original image. We then resize this section to the original image size. This method is popularly known as random cropping. Below are examples of random cropping. If you look closely, you can notice the difference between this method and scaling.

From the left, we have the original image, a square section cropped from the top-left, and then a square section cropped from the bottom-right. The cropped sections were resized to the original image size.

You can perform random crops by using any the following command for TensorFlow. **Data Augmentation Factor = Arbitrary.**

```
# TensorFlow. 'x' = A placeholder for an image.
original_size = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = original_size)
# Use the following commands to perform random crops
```

```
crop_size = [new_height, new_width, channels]
seed = np.random.randint(1234)
x = tf.random_crop(x, size = crop_size, seed = seed)
output = tf.images.resize_images(x, size = original_size)
```

5. Translation

Translation just involves moving the image along the X or Y direction (or both). In the following example, we assume that the image has a black background beyond its boundary, and are translated appropriately. This method of augmentation is very useful as most **objects** can be located at **almost anywhere** in the image. This **forces** your **convolutional neural network to look everywhere**.

From the left, we have the original image, the image translated to the right, and the image translated upwards.

You can perform translations in TensorFlow by using the following commands. **Data Augmentation Factor = Arbitrary.**

```
# pad_left, pad_right, pad_top, pad_bottom denote the pixel
# displacement. Set one of them to the desired value and rest to 0
shape = [batch, height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
# We use two functions to get our desired augmentation
x = tf.image.pad_to_bounding_box(x, pad_top, pad_left, height + pad_bottom + pad_top, width
+ pad_right + pad_left)
output = tf.image.crop_to_bounding_box(x, pad_bottom, pad_right, height, width)
```

6. Gaussian Noise

Over-fitting usually happens when your neural network tries to learn high frequency features (patterns that occur a lot) that may not be useful. Gaussian noise, which has zero mean, essentially has data points in all frequencies, effectively distorting the high frequency features. This also means that lower frequency components (usually, your intended data) are also distorted, but your neural network can learn to look past that. Adding just the right amount of noise can enhance the learning capability.

A toned down version of this is the salt and pepper noise, which presents itself as random black and white pixels spread through the image. This is similar to the effect produced by adding Gaussian noise to an image, but may have a lower information distortion level.

From the left, we have the original image, image with added Gaussian noise, image with added salt and pepper noise

You can add Gaussian noise to your image by using the following command, on TensorFlow. **Data Augmentation Factor = 2x.**

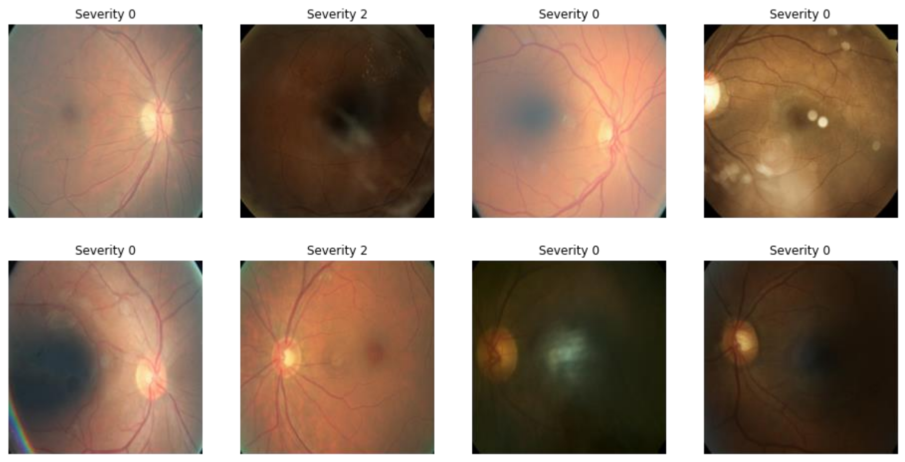
```
#TensorFlow. 'x' = A placeholder for an image.
shape = [height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
# Adding Gaussian noise
noise = tf.random_normal(shape=tf.shape(x), mean=0.0, stddev=1.0,
dtype=tf.float32)
output = tf.add(x, noise)
```

```
from keras import backend as K
from keras.applications.inception_v3 import preprocess_input
import numpy as np
IMG_SIZE = (512, 512) # slightly smaller than vgg16 normally expects
def tf_image_loader(out_size,
                      horizontal_flip = True,
                      vertical_flip = False,
                     random_brightness = True,
                     random_contrast = True,
                    random_saturation = True,
                    random_hue = True,
                      color_mode = 'rgb',
                       preproc_func = preprocess_input,
                       on_batch = False):
    def _func(X):
        with tf.name_scope('image_augmentation'):
            with tf.name_scope('input'):
                X = tf.image.decode_png(tf.read_file(X), channels = 3 if colo
r_mode == 'rgb' else 0)
                X = tf.image.resize_images(X, out_size)
            with tf.name_scope('augmentation'):
                if horizontal_flip:
                    X = tf.image.random_flip_left_right(X)
                if vertical_flip:
                    X = tf.image.random_flip_up_down(X)
                if random_brightness:
                    X = tf.image.random_brightness(X, max_delta = 0.1)
                if random_saturation:
                    X = tf.image.random_saturation(X, lower = 0.75, upper = 1
.5)
                if random_hue:
                    X = tf.image.random_hue(X, max_delta = 0.15)
                if random_contrast:
                    X = tf.image.random_contrast(X, lower = 0.75, upper = 1.5
)
                return preproc_func(X)
    if on_batch:
        # we are meant to use it on a batch
        def _batch_func(X, y):
            return tf.map_fn(_func, X), y
        return _batch_func
    else:
        # we apply it to everything
        def _all_func(X, y):
            return _func(X), y
        return _all_func
def tf_augmentor(out_size,
                intermediate_size = (640, 640),
                 intermediate_trans = 'crop',
                 batch_size = 16,
                   horizontal_flip = True,
                  vertical_flip = False,
                 random_brightness = True,
                 random_contrast = True,
                 random_saturation = True,
                    random_hue = True,
                  color_mode = 'rgb',
                   preproc_func = preprocess_input,
                   min_crop_percent = 0.001,
                   max_crop_percent = 0.005,
                   crop_probability = 0.5,
                   rotation_range = 10):

    load_ops = tf_image_loader(out_size = intermediate_size,
                               horizontal_flip=horizontal_flip,
                               vertical_flip=vertical_flip,
                               random_brightness = random_brightness,
                               random_contrast = random_contrast,
                               random_saturation = random_saturation,
                               random_hue = random_hue,
                               color_mode = color_mode,
                               preproc_func = preproc_func,
                               on_batch=False)
    def batch_ops(X, y):
        batch_size = tf.shape(X)[0]
        with tf.name_scope('transformation'):
            # code borrowed from https://becominghuman.ai/data-augmentation-
on-gpu-in-tensorflow-13d14ecf2b19
            # The list of affine transformations that our image will go under
.
            # Every element is Nx8 tensor, where N is a batch size.
            transforms = []
            identity = tf.constant([1, 0, 0, 0, 1, 0, 0, 0], dtype=tf.float32
)
            if rotation_range > 0:
                angle_rad = rotation_range / 180 * np.pi
                angles = tf.random_uniform([batch_size], -
angle_rad, angle_rad)
                transforms += [tf.contrib.image.angles_to_projective_transfor
ms(angles, intermediate_size[0], intermediate_size[1])]

            if crop_probability > 0:
                crop_pct = tf.random_uniform([batch_size], min_crop_percent,
max_crop_percent)
                left = tf.random_uniform([batch_size], 0, intermediate_size[0
] * (1.0 - crop_pct))
                top = tf.random_uniform([batch_size], 0, intermediate_size[1]
 * (1.0 - crop_pct))
                crop_transform = tf.stack([
                      crop_pct,
                        tf.zeros([batch_size]), top,
                        tf.zeros([batch_size]), crop_pct, left,
                        tf.zeros([batch_size]),
                        tf.zeros([batch_size])
                    ], 1)
                coin = tf.less(tf.random_uniform([batch_size], 0, 1.0), crop_
probability)
                transforms += [tf.where(coin, crop_transform, tf.tile(tf.expa
nd_dims(identity, 0), [batch_size, 1]))]
            if len(transforms)>0:
                X = tf.contrib.image.transform(X,
                      tf.contrib.image.compose_transforms(*transforms),
                      interpolation='BILINEAR') # or 'NEAREST'
            if intermediate_trans=='scale':
                X = tf.image.resize_images(X, out_size)
            elif intermediate_trans=='crop':
                X = tf.image.resize_image_with_crop_or_pad(X, out_size[0], ou
t_size[1])
            else:
                raise ValueError('Invalid Operation {}'.format(intermediate_t
rans))
            return X, y

    def _create_pipeline(in_ds):
        batch_ds = in_ds.map(load_ops, num_parallel_calls=4).batch(batch_size
)
        return batch_ds.map(batch_ops)
    return _create_pipeline


def flow_from_dataframe(idg,
                        in_df,
                        path_col,
                        y_col,
                        shuffle = True,
                        color_mode = 'rgb'):
    files_ds = tf.data.Dataset.from_tensor_slices((in_df[path_col].values,
                                                   np.stack(in_df[y_col].valu
es,0)))
    in_len = in_df[path_col].values.shape[0]
    while True:
        if shuffle:
            files_ds = files_ds.shuffle(in_len) # shuffle the whole dataset

        next_batch = idg(files_ds).repeat().make_one_shot_iterator().get_next
()
        for i in range(max(in_len//32,1)):
            # NOTE: if we loop here it is 'thread-safe-
ish' if we loop on the outside it is completely unsafe
            yield K.get_session().run(next_batch)
batch_size = 48
core_idg = tf_augmentor(out_size = IMG_SIZE,
                        color_mode = 'rgb',
                        vertical_flip = True,
                        crop_probability=0.0, # crop doesn't work yet
                        batch_size = batch_size)
valid_idg = tf_augmentor(out_size = IMG_SIZE, color_mode = 'rgb',
                         crop_probability=0.0,
                         horizontal_flip = False,
                         vertical_flip = False,
                                        random_brightness = False,
                                        random_contrast = False,
                                        random_saturation = False,
                                        random_hue = False,
                                        rotation_range = 0,
                                      batch_size = batch_size)

train_gen = flow_from_dataframe(core_idg, train_df,
                                             path_col = 'path',
                                            y_col = 'level_cat')

valid_gen = flow_from_dataframe(valid_idg, valid_df,
                                             path_col = 'path',
                                            y_col = 'level_cat') # we can use much larger bat
ches for evaluation
```

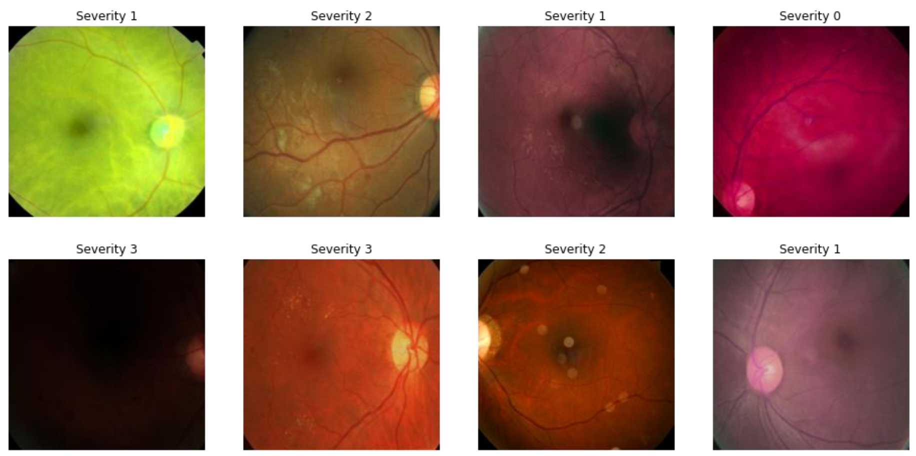
7.8 VALIDATION SET:

We do not perform augmentation at all on these images

```
t_x, t_y = next(valid_gen)
fig, m_axs = plt.subplots(2, 4, figsize = (16, 8))
for (c_x, c_y, c_ax) in zip(t_x, t_y, m_axs.flatten()):
    c_ax.imshow(np.clip(c_x*127+127, 0, 255).astype(np.uint8))
    c_ax.set_title('Severity {}'.format(np.argmax(c_y, -1)))
    c_ax.axis('off')
```



```
t_x, t_y = next(train_gen)
fig, m_axs = plt.subplots(2, 4, figsize = (16, 8))
for (c_x, c_y, c_ax) in zip(t_x, t_y, m_axs.flatten()):
    c_ax.imshow(np.clip(c_x*127+127, 0, 255).astype(np.uint8))
    c_ax.set_title('Severity {}'.format(np.argmax(c_y, -1)))
    c_ax.axis('off')
```



7.9 ATTENTION MODEL:

The basic idea is that a Global Average Pooling is too simplistic since some of the regions are more relevant than others. So we build an attention mechanism to turn pixels in the GAP on an off before the pooling and then rescale (Lambda layer) the results based on the number of pixels. The model could be seen as a sort of 'global weighted average' pooling. There is probably something published about it and it is very similar to the kind of attention models used in NLP. It is largely based on the insight that the winning solution annotated and trained a UNET model to segmenting the hand and transforming it. This seems very tedious if we could just learn attention.

```
"""Inception V3 model for Keras.
Note that the input image format for this model is different than for
the VGG16 and ResNet models (299x299 instead of 224x224),
and that the input preprocessing function is also different (same as Xception).
# Reference
- [Rethinking the Inception Architecture for Computer Vision](
    http://arxiv.org/abs/1512.00567) (CVPR 2016)
"""
```

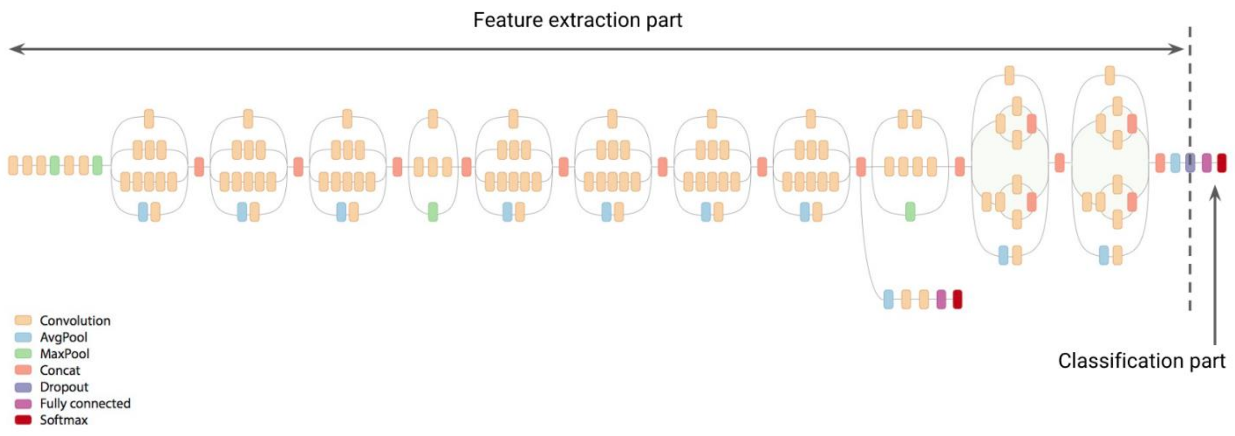
7.9.1 INCEPTION V3 MODEL:

Introduction:

Transfer learning is a machine learning method which utilizes a pre-trained neural network. For example, the image recognition model called Inception-v3 consists of two parts:

- Feature extraction part with a convolutional neural network.
- Classification part with fully-connected and softmax layers.

The pre-trained Inception-v3 model achieves state-of-the-art accuracy for recognizing general objects with 1000 classes, like "Zebra", "Dalmatian", and "Dishwasher". The model extracts general features from input images in the first part and classifies them based on those features in the second part.



In transfer learning, when you build a new model to classify your original dataset, you reuse the feature extraction part and re-train the classification part with your dataset. Since you don't have to train the feature extraction part (which is the most complex part of the model), you can train the model with less computational resources and training time.

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import os
from . import get_submodules_from_kwargs
from . import imagenet_utils
from .imagenet_utils import decode_predictions
from .imagenet_utils import _obtain_input_shape
```

```
WEIGHTS_PATH = (
    'https://github.com/fchollet/deep-learning-models/'
    'releases/download/v0.5/'
    'inception_v3_weights_tf_dim_ordering_tf_kernels.h5')
WEIGHTS_PATH_NO_TOP = (
    'https://github.com/fchollet/deep-learning-models/'
    'releases/download/v0.5/'
    'inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5')

backend = None
layers = None
models = None
keras_utils = None
def conv2d_bn(x,
              filters,
              num_row,
              num_col,
              padding='same',
              strides=(1, 1),
              name=None):
    """Utility function to apply conv + BN.
    # Arguments
        x: input tensor.
        filters: filters in `Conv2D`.
        num_row: height of the convolution kernel.
        num_col: width of the convolution kernel.
        padding: padding mode in `Conv2D`.
        strides: strides in `Conv2D`.
        name: name of the ops; will become `name + '_conv'`
            for the convolution and `name + '_bn'` for the
            batch norm layer.
    # Returns
        Output tensor after applying `Conv2D` and `BatchNormalization`.
    """
    if name is not None:
        bn_name = name + '_bn'
        conv_name = name + '_conv'
    else:
        bn_name = None
        conv_name = None
    if backend.image_data_format() == 'channels_first':
        bn_axis = 1
    else:
        bn_axis = 3
    x = layers.BinaryConv2D(
        filters, (num_row, num_col),
        strides=strides,
        padding=padding,
        use_bias=False,
        name=conv_name)(x)
    x = layers.BatchNormalization(axis=bn_axis, scale=False, name=bn_name)(x)
    x = layers.Activation('relu', name=name)(x)
    return x
def BinarizedInceptionV3(include_top=True,
                weights='imagenet',
                input_tensor=None,
                input_shape=None,
                pooling=None,
                classes=1000,
                **kwargs):
```

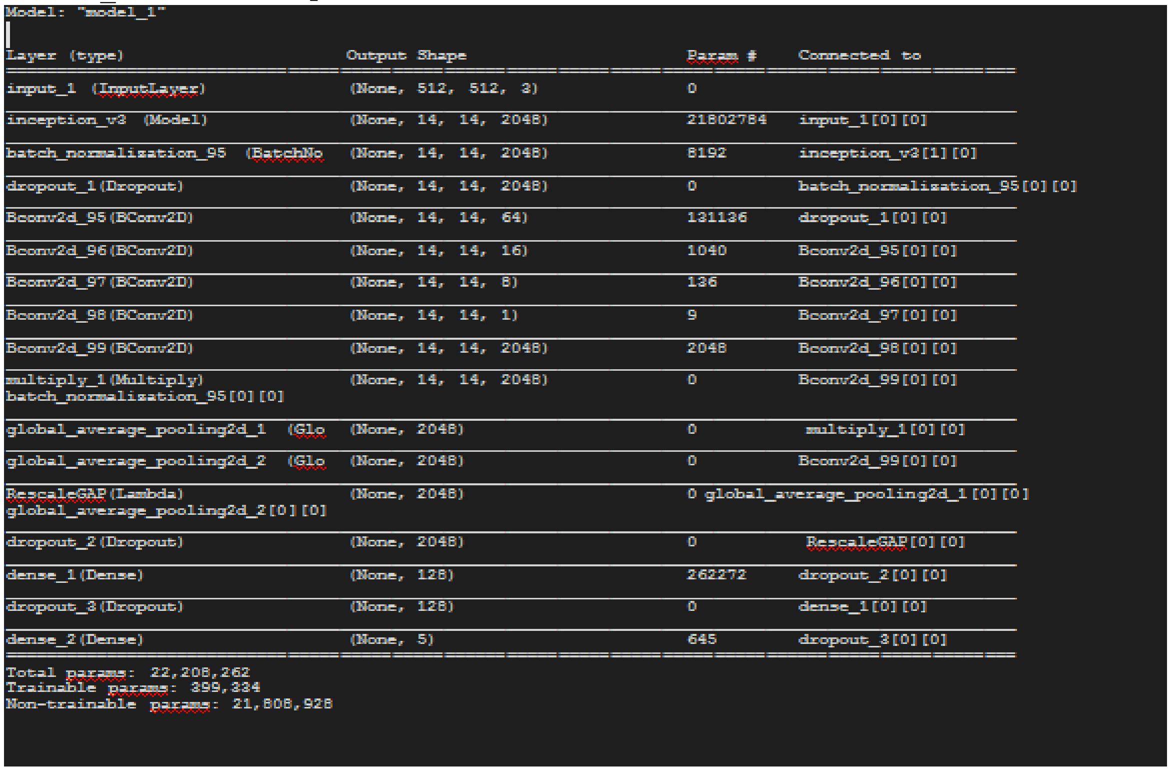
```
    """Instantiates the Inception v3 architecture.
    Optionally loads weights pre-trained on ImageNet.
    Note that the data format convention used by the model is
    the one specified in your Keras config at `~/.keras/keras.json`.
    # Arguments
        include_top: whether to include the fully-connected
            layer at the top of the network.
        weights: one of `None` (random initialization),
              'imagenet' (pre-training on ImageNet),
              or the path to the weights file to be loaded.
        input_tensor: optional Keras tensor (i.e. output of `layers.Input()`)
            to use as image input for the model.
        input_shape: optional shape tuple, only to be specified
            if `include_top` is False (otherwise the input shape
            has to be `(299, 299, 3)` (with `channels_last` data format)
            or `(3, 299, 299)` (with `channels_first` data format).
            It should have exactly 3 inputs channels,
            and width and height should be no smaller than 75.
            E.g. `(150, 150, 3)` would be one valid value.
        pooling: Optional pooling mode for feature extraction
            when `include_top` is `False`.
            - `None` means that the output of the model will be
                the 4D tensor output of the
                last convolutional block.
            - `avg` means that global average pooling
                will be applied to the output of the
                last convolutional block, and thus
                the output of the model will be a 2D tensor.
            - `max` means that global max pooling will
                be applied.
        classes: optional number of classes to classify images
            into, only to be specified if `include_top` is True, and
            if no `weights` argument is specified.
    # Returns
        A Keras model instance.
    # Raises
        ValueError: in case of invalid argument for `weights`,
            or invalid input shape.
    """
    global backend, layers, models, keras_utils
    backend, layers, models, keras_utils = get_submodules_from_kwargs(kwargs)

    if not (weights in {'imagenet', None} or os.path.exists(weights)):
        raise ValueError('The `weights` argument should be either '
                         '`None` (random initialization), `imagenet` '
                         '(pre-training on ImageNet), '
                         'or the path to the weights file to be loaded.')

    if weights == 'imagenet' and include_top and classes != 1000:
        raise ValueError('If using `weights` as `"imagenet"` with `include_to
p`'
                         ' as true, `classes` should be 1000')

 # Determine proper input shape
    input_shape = _obtain_input_shape(input_shape,
                                      default_size=224,
                                      min_size=32,
                                      data_format=backend.image_data_format()
,
                                      require_flatten=include_top,
                                          weights=weights)

    if input_tensor is None:
        img_input = layers.Input(shape=input_shape)
    else:
        if not backend.is_keras_tensor(input_tensor):
            img_input = layers.Input(tensor=input_tensor, shape=input_shape)
        else:
            img_input = input_tensor
    # Block 1
    x = layers.BinaryConv2D(64, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block1_conv1')(img_input)
    x = layers.BinaryConv2D(64, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block1_conv2')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

    # Block 2
    x = layers.BinaryConv2D(128, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block2_conv1')(x)
    x = layers.BinaryConv2D(128, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block2_conv2')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

    # Block 3
    x = layers.BinaryConv2D(256, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block3_conv1')(x)
    x = layers.BinaryConv2D(256, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block3_conv2')(x)
    x = layers.BinaryConv2D(256, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block3_conv3')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

    # Block 4
    x = layers.BinaryConv2D(512, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block4_conv1')(x)
    x = layers.BinaryConv2D(512, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block4_conv2')(x)
    x = layers.BinaryConv2D(512, (3, 3),
                      activation='relu',
                      padding='same',
                      name='block4_conv3')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)
```

```
# Block 5
x = layers.BinaryConv2D(512, (3, 3),
                  activation='relu',
                  padding='same',
                  name='block5_conv1')(x)
x = layers.BinaryConv2D(512, (3, 3),
                  activation='relu',
                  padding='same',
                  name='block5_conv2')(x)
x = layers.BinaryConv2D(512, (3, 3),
                  activation='relu',
                  padding='same',
                  name='block5_conv3')(x)
x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
# Determine proper input shape
input_shape = _obtain_input_shape(
    input_shape,
    default_size=299,
    min_size=75,
    data_format=backend.image_data_format(),
    require_flatten=include_top,
    weights=weights)

if input_tensor is None:
    img_input = layers.Input(shape=input_shape)
else:
    if not backend.is_keras_tensor(input_tensor):
        img_input = layers.Input(tensor=input_tensor, shape=input_shape)
    else:
        img_input = input_tensor

if backend.image_data_format() == 'channels_first':
    channel_axis = 1
else:
    channel_axis = 3

x = conv2d_bn(img_input, 32, 3, 3, strides=(2, 2), padding='valid')
x = conv2d_bn(x, 32, 3, 3, padding='valid')
x = conv2d_bn(x, 64, 3, 3)
x = layers.MaxPooling2D((3, 3), strides=(2, 2))(x)

x = conv2d_bn(x, 80, 1, 1, padding='valid')
x = conv2d_bn(x, 192, 3, 3, padding='valid')
x = layers.MaxPooling2D((3, 3), strides=(2, 2))(x)

# mixed 0: 35 x 35 x 256
branch1x1 = conv2d_bn(x, 64, 1, 1)

branch5x5 = conv2d_bn(x, 48, 1, 1)
branch5x5 = conv2d_bn(branch5x5, 64, 5, 5)

branch3x3dbl = conv2d_bn(x, 64, 1, 1)
branch3x3dbl = conv2d_bn(branch3x3dbl, 96, 3, 3)
branch3x3dbl = conv2d_bn(branch3x3dbl, 96, 3, 3)

branch_pool = layers.AveragePooling2D((3, 3),
                                      strides=(1, 1),
                                      padding='same')(x)
branch_pool = conv2d_bn(branch_pool, 32, 1, 1)
```

```
x = layers.concatenate(
    [branch1x1, branch5x5, branch3x3dbl, branch_pool],
    axis=channel_axis,
    name='mixed0')

# mixed 1: 35 x 35 x 288
branch1x1 = conv2d_bn(x, 64, 1, 1)

branch5x5 = conv2d_bn(x, 48, 1, 1)
branch5x5 = conv2d_bn(branch5x5, 64, 5, 5)

branch3x3dbl = conv2d_bn(x, 64, 1, 1)
branch3x3dbl = conv2d_bn(branch3x3dbl, 96, 3, 3)
branch3x3dbl = conv2d_bn(branch3x3dbl, 96, 3, 3)

branch_pool = layers.AveragePooling2D((3, 3),
                                      strides=(1, 1),
                                      padding='same')(x)
branch_pool = conv2d_bn(branch_pool, 64, 1, 1)
x = layers.concatenate(
    [branch1x1, branch5x5, branch3x3dbl, branch_pool],
    axis=channel_axis,
    name='mixed1')

# mixed 2: 35 x 35 x 288
branch1x1 = conv2d_bn(x, 64, 1, 1)

branch5x5 = conv2d_bn(x, 48, 1, 1)
branch5x5 = conv2d_bn(branch5x5, 64, 5, 5)

branch3x3dbl = conv2d_bn(x, 64, 1, 1)
branch3x3dbl = conv2d_bn(branch3x3dbl, 96, 3, 3)
branch3x3dbl = conv2d_bn(branch3x3dbl, 96, 3, 3)

branch_pool = layers.AveragePooling2D((3, 3),
                                      strides=(1, 1),
                                      padding='same')(x)
branch_pool = conv2d_bn(branch_pool, 64, 1, 1)
x = layers.concatenate(
    [branch1x1, branch5x5, branch3x3dbl, branch_pool],
    axis=channel_axis,
    name='mixed2')

# mixed 3: 17 x 17 x 768
branch3x3 = conv2d_bn(x, 384, 3, 3, strides=(2, 2), padding='valid')

branch3x3dbl = conv2d_bn(x, 64, 1, 1)
branch3x3dbl = conv2d_bn(branch3x3dbl, 96, 3, 3)
branch3x3dbl = conv2d_bn(
    branch3x3dbl, 96, 3, 3, strides=(2, 2), padding='valid')

branch_pool = layers.MaxPooling2D((3, 3), strides=(2, 2))(x)
x = layers.concatenate(
    [branch3x3, branch3x3dbl, branch_pool],
    axis=channel_axis,
    name='mixed3')

# mixed 4: 17 x 17 x 768
branch1x1 = conv2d_bn(x, 192, 1, 1)
```

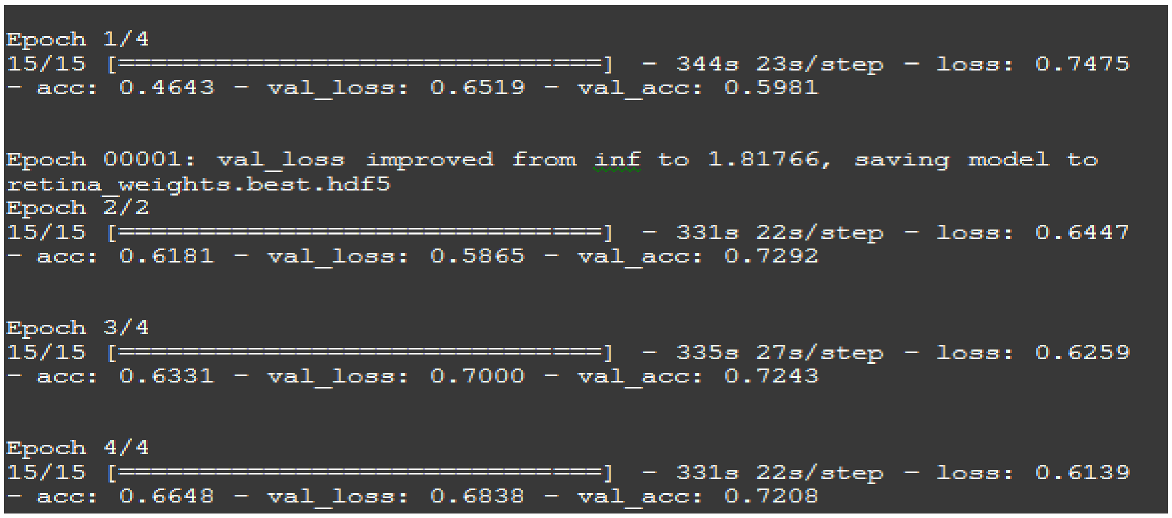
```
branch7x7 = conv2d_bn(x, 128, 1, 1)
branch7x7 = conv2d_bn(branch7x7, 128, 1, 7)
branch7x7 = conv2d_bn(branch7x7, 192, 7, 1)

branch7x7dbl = conv2d_bn(x, 128, 1, 1)
branch7x7dbl = conv2d_bn(branch7x7dbl, 128, 7, 1)
branch7x7dbl = conv2d_bn(branch7x7dbl, 128, 1, 7)
branch7x7dbl = conv2d_bn(branch7x7dbl, 128, 7, 1)
branch7x7dbl = conv2d_bn(branch7x7dbl, 192, 1, 7)

branch_pool = layers.AveragePooling2D((3, 3),
                                      strides=(1, 1),
                                      padding='same')(x)
branch_pool = conv2d_bn(branch_pool, 192, 1, 1)
x = layers.concatenate(
    [branch1x1, branch7x7, branch7x7dbl, branch_pool],
    axis=channel_axis,
    name='mixed4')

# mixed 5, 6: 17 x 17 x 768
for i in range(2):
    branch1x1 = conv2d_bn(x, 192, 1, 1)

    branch7x7 = conv2d_bn(x, 160, 1, 1)
    branch7x7 = conv2d_bn(branch7x7, 160, 1, 7)
    branch7x7 = conv2d_bn(branch7x7, 192, 7, 1)

    branch7x7dbl = conv2d_bn(x, 160, 1, 1)
    branch7x7dbl = conv2d_bn(branch7x7dbl, 160, 7, 1)
    branch7x7dbl = conv2d_bn(branch7x7dbl, 160, 1, 7)
    branch7x7dbl = conv2d_bn(branch7x7dbl, 160, 7, 1)
    branch7x7dbl = conv2d_bn(branch7x7dbl, 192, 1, 7)

    branch_pool = layers.AveragePooling2D(
        (3, 3), strides=(1, 1), padding='same')(x)
    branch_pool = conv2d_bn(branch_pool, 192, 1, 1)
    x = layers.concatenate(
        [branch1x1, branch7x7, branch7x7dbl, branch_pool],
        axis=channel_axis,
        name='mixed' + str(5 + i))

# mixed 7: 17 x 17 x 768
branch1x1 = conv2d_bn(x, 192, 1, 1)

branch7x7 = conv2d_bn(x, 192, 1, 1)
branch7x7 = conv2d_bn(branch7x7, 192, 1, 7)
branch7x7 = conv2d_bn(branch7x7, 192, 7, 1)

branch7x7dbl = conv2d_bn(x, 192, 1, 1)
branch7x7dbl = conv2d_bn(branch7x7dbl, 192, 7, 1)
branch7x7dbl = conv2d_bn(branch7x7dbl, 192, 1, 7)
branch7x7dbl = conv2d_bn(branch7x7dbl, 192, 7, 1)
branch7x7dbl = conv2d_bn(branch7x7dbl, 192, 1, 7)

branch_pool = layers.AveragePooling2D((3, 3),
                                      strides=(1, 1),
                                      padding='same')(x)
branch_pool = conv2d_bn(branch_pool, 192, 1, 1)
x = layers.concatenate(
    [branch1x1, branch7x7, branch7x7dbl, branch_pool],
        axis=channel_axis,
        name='mixed7')

    # mixed 8: 8 x 8 x 1280
    branch3x3 = conv2d_bn(x, 192, 1, 1)
    branch3x3 = conv2d_bn(branch3x3, 320, 3, 3,
                          strides=(2, 2), padding='valid')

    branch7x7x3 = conv2d_bn(x, 192, 1, 1)
    branch7x7x3 = conv2d_bn(branch7x7x3, 192, 1, 7)
    branch7x7x3 = conv2d_bn(branch7x7x3, 192, 7, 1)
    branch7x7x3 = conv2d_bn(
        branch7x7x3, 192, 3, 3, strides=(2, 2), padding='valid')

    branch_pool = layers.MaxPooling2D((3, 3), strides=(2, 2))(x)
    x = layers.concatenate(
        [branch3x3, branch7x7x3, branch_pool],
        axis=channel_axis,
        name='mixed8')

    # mixed 9: 8 x 8 x 2048
    for i in range(2):
        branch1x1 = conv2d_bn(x, 320, 1, 1)

        branch3x3 = conv2d_bn(x, 384, 1, 1)
        branch3x3_1 = conv2d_bn(branch3x3, 384, 1, 3)
        branch3x3_2 = conv2d_bn(branch3x3, 384, 3, 1)
        branch3x3 = layers.concatenate(
            [branch3x3_1, branch3x3_2],
            axis=channel_axis,
            name='mixed9_' + str(i))

        branch3x3dbl = conv2d_bn(x, 448, 1, 1)
        branch3x3dbl = conv2d_bn(branch3x3dbl, 384, 3, 3)
        branch3x3dbl_1 = conv2d_bn(branch3x3dbl, 384, 1, 3)
        branch3x3dbl_2 = conv2d_bn(branch3x3dbl, 384, 3, 1)
        branch3x3dbl = layers.concatenate(
            [branch3x3dbl_1, branch3x3dbl_2], axis=channel_axis)

        branch_pool = layers.AveragePooling2D(
            (3, 3), strides=(1, 1), padding='same')(x)
        branch_pool = conv2d_bn(branch_pool, 192, 1, 1)
        x = layers.concatenate(
            [branch1x1, branch3x3, branch3x3dbl, branch_pool],
            axis=channel_axis,
            name='mixed' + str(9 + i))
    if include_top:
        # Classification block
        x = layers.GlobalAveragePooling2D(name='avg_pool')(x)
        x = layers.BinaryDense(classes, activation='softmax', name='predictio
ns')(x)
    else:
        if pooling == 'avg':
            x = layers.GlobalAveragePooling2D()(x)
        elif pooling == 'max':
            x = layers.GlobalMaxPooling2D()(x)

    # Ensure that the model takes into account
    # any potential predecessors of `input_tensor`.
    if input_tensor is not None:
        inputs = keras_utils.get_source_inputs(input_tensor)
    else:
        inputs = img_input
    # Create model.
    model = models.Model(inputs, x, name='inception_v3')


    # Load weights.
    if weights == 'imagenet':
        if include_top:
            weights_path = keras_utils.get_file(
                'inception_v3_weights_tf_dim_ordering_tf_kernels.h5',
                WEIGHTS_PATH,
                cache_subdir='models',
                file_hash='9a0d58056eeedaa3f26cb7ebd46da564')
        else:
            weights_path = keras_utils.get_file(
                'inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5',
                WEIGHTS_PATH_NO_TOP,
                cache_subdir='models',
                file_hash='bcbd6486424b2319ff4ef7d526e38f63')
        model.load_weights(weights_path)
    elif weights is not None:
        model.load_weights(weights)

    return model


def preprocess_input(x, **kwargs):

    """Preprocesses a numpy array encoding a batch of images.
    # Arguments
        x: a 4D numpy array consists of RGB values within [0, 255].
    # Returns
        Preprocessed array.
    """

    return imagenet_utils.preprocess_input(x, mode='tf', **kwargs)
```

```
from keras.applications.inception_v3 import InceptionV3 as PTModel
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten, Inp
ut, Conv2D, multiply, LocallyConnected2D, Lambda
from keras.models import Model
in_lay = Input(t_x.shape[1:])
base_pretrained_model = PTModel(input_shape =  t_x.shape[1:], include_top = F
alse, weights = 'imagenet')
base_pretrained_model.trainable = False
pt_depth = base_pretrained_model.get_output_shape_at(0)[-1]
pt_features = base_pretrained_model(in_lay)
from keras.layers import BatchNormalization
bn_features = BatchNormalization()(pt_features)

# here we do an attention mechanism to turn pixels in the GAP on an off
```

```
attn_layer = Conv2D(64, kernel_size = (1,1), padding = 'same', activation = '
relu')(Dropout(0.5)(bn_features))
attn_layer = Conv2D(16, kernel_size = (1,1), padding = 'same', activation = '
relu')(attn_layer)
attn_layer = Conv2D(8, kernel_size = (1,1), padding = 'same', activation = 'r
elu')(attn_layer)
attn_layer = Conv2D(1,
                    kernel_size = (1,1),
                    padding = 'valid',
                    activation = 'sigmoid')(attn_layer)
# fan it out to all of the channels
up_c2_w = np.ones((1, 1, 1, pt_depth))
up_c2 = Conv2D(pt_depth, kernel_size = (1,1), padding = 'same',
               activation = 'linear', use_bias = False, weights = [up_c2_w])
up_c2.trainable = False
attn_layer = up_c2(attn_layer)
mask_features = multiply([attn_layer, bn_features])
gap_features = GlobalAveragePooling2D()(mask_features)
gap_mask = GlobalAveragePooling2D()(attn_layer)
# to account for missing values from the attention model
gap = Lambda(lambda x: x[0]/x[1], name = 'RescaleGAP')([gap_features, gap_mas
k])
gap_dr = Dropout(0.25)(gap)
dr_steps = Dropout(0.25)(Dense(128, activation = 'relu')(gap_dr))
out_layer = Dense(t_y.shape[-1], activation = 'softmax')(dr_steps)
retina_model = Model(inputs = [in_lay], outputs = [out_layer])
from keras.metrics import top_k_categorical_accuracy
def top_2_accuracy(in_gt, in_pred):
    return top_k_categorical_accuracy(in_gt, in_pred, k=2)

retina_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
                     metrics = ['categorical_accuracy', top_2_accuracy]
)
retina_model.summary()
```


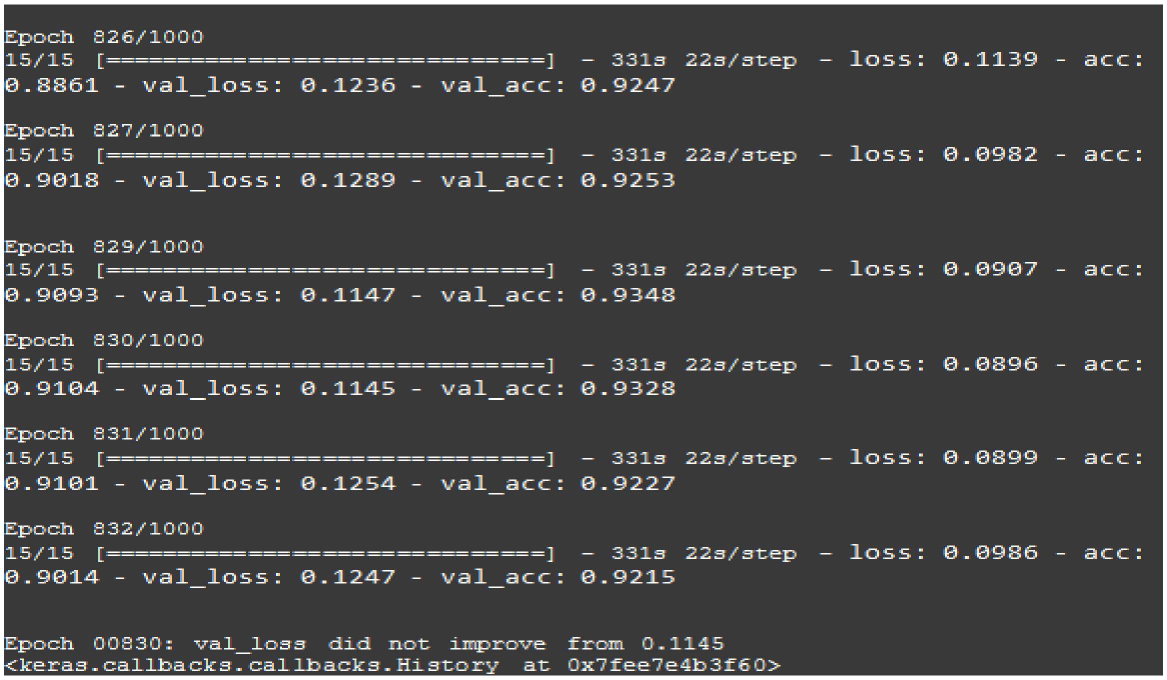
```
Model: "model_1"

Layer (type)                     Output Shape              Param #     Connected to
==========================================================================================
input_1 (InputLayer)             (None, 512, 512, 3)       0
__________________________________________________________________________________________
inception_v3 (Model)             (None, 14, 14, 2048)      21802784    input_1[0][0]
__________________________________________________________________________________________
batch_normalization_95 (BatchNo  (None, 14, 14, 2048)      8192        inception_v3[1][0]
__________________________________________________________________________________________
dropout_1(Dropout)               (None, 14, 14, 2048)      0           batch_normalization_95[0][0]
__________________________________________________________________________________________
Bconv2d_95(BConv2D)              (None, 14, 14, 64)        131136      dropout_1[0][0]
__________________________________________________________________________________________
Bconv2d_96(BConv2D)              (None, 14, 14, 16)        1040        Bconv2d_95[0][0]
__________________________________________________________________________________________
Bconv2d_97(BConv2D)              (None, 14, 14, 8)         136         Bconv2d_96[0][0]
__________________________________________________________________________________________
Bconv2d_98(BConv2D)              (None, 14, 14, 1)         9           Bconv2d_97[0][0]
__________________________________________________________________________________________
Bconv2d_99(BConv2D)              (None, 14, 14, 2048)      2048        Bconv2d_98[0][0]
__________________________________________________________________________________________
multiply_1(Multiply)             (None, 14, 14, 2048)      0           Bconv2d_99[0][0]
batch_normalization_95[0][0]
__________________________________________________________________________________________
global_average_pooling2d_1 (Glo  (None, 2048)              0            multiply_1[0][0]
__________________________________________________________________________________________
global_average_pooling2d_2 (Glo  (None, 2048)              0           Bconv2d_99[0][0]
__________________________________________________________________________________________
RescaleGAP(Lambda)               (None, 2048)              0 global_average_pooling2d_1[0][0]
global_average_pooling2d_2[0][0]
__________________________________________________________________________________________
dropout_2(Dropout)               (None, 2048)              0            RescaleGAP[0][0]
__________________________________________________________________________________________
dense_1(Dense)                   (None, 128)               262272      dropout_2[0][0]
__________________________________________________________________________________________
dropout_3(Dropout)               (None, 128)               0           dense_1[0][0]
__________________________________________________________________________________________
dense_2(Dense)                   (None, 5)                 645         dropout_3[0][0]
==========================================================================================
Total params: 22,208,262
Trainable params: 399,334
Non-trainable params: 21,808,928
```

```
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStop
ping, ReduceLROnPlateau
weight_path="{}_weights.best.hdf5".format('retina')

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min', save_weights_on
ly = True)

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.8, patience=3
, verbose=1, mode='auto', epsilon=0.0001, cooldown=5, min_lr=0.0001)
early = EarlyStopping(monitor="val_loss",
                      mode="min",
                      patience=6) # probably needs to be more patient, but ka
ggle time is limited
callbacks_list = [checkpoint, early, reduceLROnPlat]



!rm -rf ~/.keras
# clean up before starting training
#gc.collect()

retina_model.fit_generator(train_gen,
                           steps_per_epoch = train_df.shape[0]//batch_size,
                           validation_data = valid_gen,
                           validation_steps = valid_df.shape[0]//batch_size,
                              epochs = 2,
                              callbacks = callbacks_list,
                             workers = 0, # tf-generators are not thread-safe
                             use_multiprocessing=False,
                             max_queue_size = 0
                            )



# load the best version of the model
retina_model.load_weights(weight_path)
retina_model.save('full_retina_model.h5')
```



```
Epoch 1/4
15/15 [==============================] - 344s 23s/step - loss: 0.7475
- acc: 0.4643 - val_loss: 0.6519 - val_acc: 0.5981


Epoch 00001: val_loss improved from inf to 1.81766, saving model to
retina_weights.best.hdf5
Epoch 2/2
15/15 [==============================] - 331s 22s/step - loss: 0.6447
- acc: 0.6181 - val_loss: 0.5865 - val_acc: 0.7292


Epoch 3/4
15/15 [==============================] - 335s 27s/step - loss: 0.6259
- acc: 0.6331 - val_loss: 0.7000 - val_acc: 0.7243


Epoch 4/4
15/15 [==============================] - 331s 22s/step - loss: 0.6139
- acc: 0.6648 - val_loss: 0.6838 - val_acc: 0.7208
```

Fig : For 4 Epochs loss and accuracy



```
Epoch 826/1000
15/15 [==============================] - 331s 22s/step - loss: 0.1139 - acc:
0.8861 - val_loss: 0.1236 - val_acc: 0.9247

Epoch 827/1000
15/15 [==============================] - 331s 22s/step - loss: 0.0982 - acc:
0.9018 - val_loss: 0.1289 - val_acc: 0.9253


Epoch 829/1000
15/15 [==============================] - 331s 22s/step - loss: 0.0907 - acc:
0.9093 - val_loss: 0.1147 - val_acc: 0.9348

Epoch 830/1000
15/15 [==============================] - 331s 22s/step - loss: 0.0896 - acc:
0.9104 - val_loss: 0.1145 - val_acc: 0.9328

Epoch 831/1000
15/15 [==============================] - 331s 22s/step - loss: 0.0899 - acc:
0.9101 - val_loss: 0.1254 - val_acc: 0.9227

Epoch 832/1000
15/15 [==============================] - 331s 22s/step - loss: 0.0986 - acc:
0.9014 - val_loss: 0.1247 - val_acc: 0.9215


Epoch 00830: val_loss did not improve from 0.1145
<keras.callbacks.callbacks.History at 0x7fee7e4b3f60>
```

Fig: High accuracy at 830 Epochs

```
plt.figure(figsize=(20,10))
plt.subplot(1,2,1)
plt.plot(hist.history["loss"],label="train loss")
plt.plot(hist.history["val_loss"],label="val loss")
plt.legend()
plt.show()
```

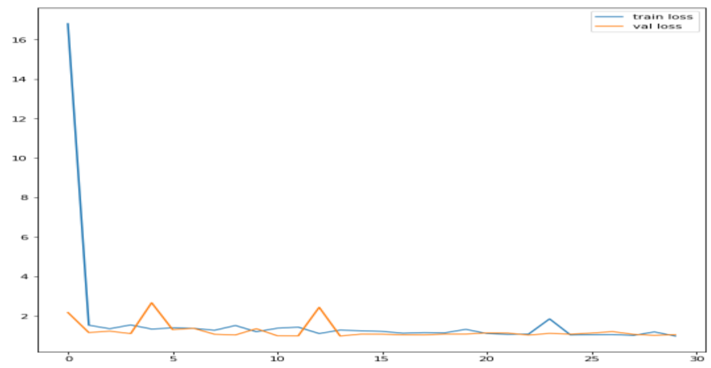
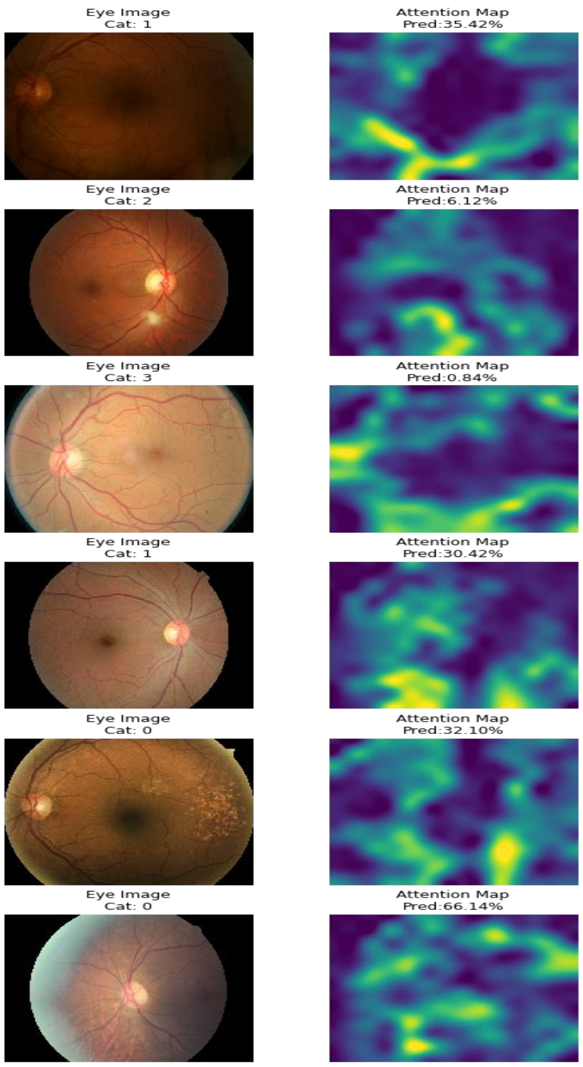


Fig: train_loss vs val_loss

7.9.2 ATTENTION MAP:

```
# get the attention layer since it is the only one with a single output dim
for attn_layer in retina_model.layers:
    c_shape = attn_layer.get_output_shape_at(0)
    if len(c_shape)==4:
        if c_shape[-1]==1:
            print(attn_layer)
            break

import keras.backend as K
rand_idx = np.random.choice(range(len(test_X)), size = 6)
```

```
attn_func = K.function(inputs = [retina_model.get_input_at(0), K.learning_pha
se()],
           outputs = [attn_layer.get_output_at(0)]
          )
fig, m_axs = plt.subplots(len(rand_idx), 2, figsize = (8, 4*len(rand_idx)))
[c_ax.axis('off') for c_ax in m_axs.flatten()]
for c_idx, (img_ax, attn_ax) in zip(rand_idx, m_axs):
    cur_img = test_X[c_idx☹c_idx+1)]
    attn_img = attn_func([cur_img, 0])[0]
    img_ax.imshow(np.clip(cur_img[0,:,:,:]*127+127, 0, 255).astype(np.uint8))
    attn_ax.imshow(attn_img[0, :, :, 0]/attn_img[0, :, :, 0].max(), cmap = 'v
iridis',
                   vmin = 0, vmax = 1,
                   interpolation = 'lanczos')
    real_cat = np.argmax(test_Y[c_idx, :])
    img_ax.set_title('Eye Image\nCat:%2d' % (real_cat))
    pred_cat = retina_model.predict(cur_img)
    attn_ax.set_title('Attention Map\nPred:%2.2f%%' % (100*pred_cat[0,real_ca
t]))
fig.savefig('attention_map.png', dpi = 300)
```



7.10 ROC CURVE FOR HEALTHY VS SICK:

Here we make an ROC curve for healthy (severity == 0) and sick (severity>0) to see how well the model works at just identifying the disease

When making a prediction for a two-class classification problem, the following types of errors can be made by a classifier:

- **False Positive (FP)**: predict an event when there was no event.
- **False Negative (FN)**: predict no event when in fact there was an event.
- **True Positive (TP)**: predict an event when there was an event.
- **True Negative (TN)**: predict no event when in fact there was no event.

  - Accuracy simply measures the number of correct predicted samples over the total number of samples. For instance, if the classifier is 90% correct, it means that out of 100 instance it correctly predicts the class for 90 of them.
  - accuracy=nr. correct predictions/nr. total predictions
    =(TP+TN)/(TP+TN+FP+FN)

While defining the metrics above, I assumed that we are directly given the predictions of each class. But it might be the case that we have the probability for each class instead, which then allows to calibrate the threshold on how to interpret the probabilities. Does it belong to positive class if it's greater than 0.5 or 0.3 ?

The curve is a plot of *false positive rate* (x-axis) versus the *true positive rate* (y-axis) for a number of different candidate threshold values between 0.0 and 1.0. An operator may plot the ROC curve and choose a threshold that gives a desirable balance between the false positives and false negatives.

- **x-axis**: the false positive rate is also referred to as the inverted specificity where specificity is the total number of true negatives divided by the sum of the number of true negatives and false positives.

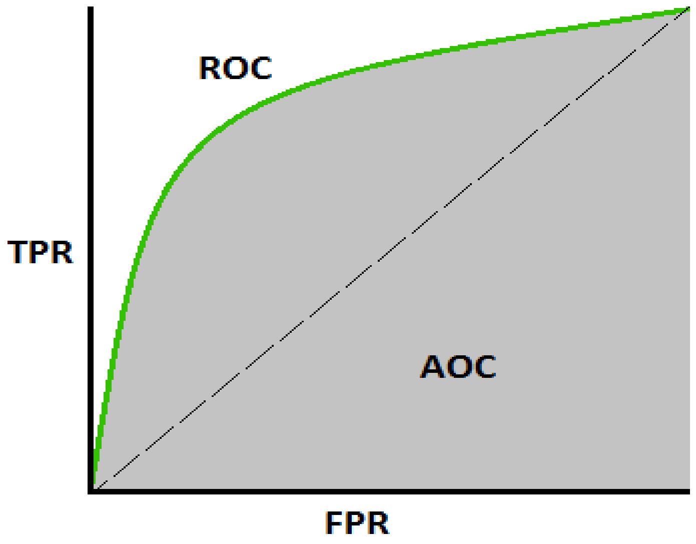
False Positive Rate=FP/FP+TN

- **y-axis**: the true positive rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.

True Positive Rate=TP/TP+FN

NOTE: remember that both the False Positive Rate and the True Positive Rate are calculated for different probability thresholds.

The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.

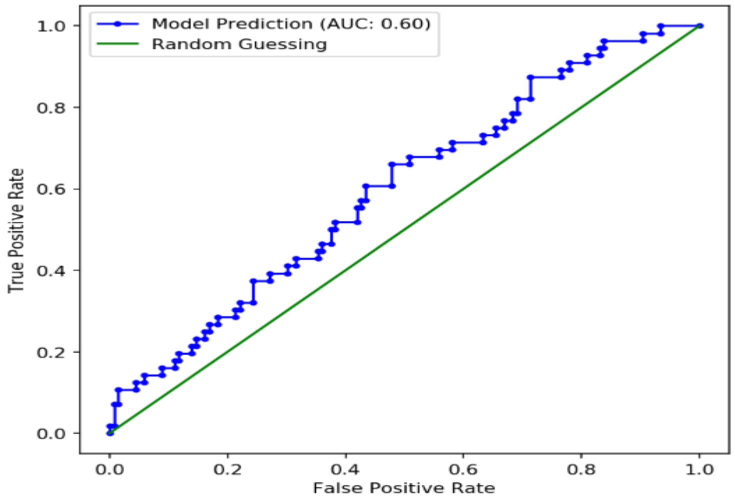


HEALTHY VS SICK:

An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever.

```
from sklearn.metrics import roc_curve, roc_auc_score
sick_vec = test_Y_cat>0
sick_score = np.sum(pred_Y[:,1:],1)
fpr, tpr, _ = roc_curve(sick_vec, sick_score)
```

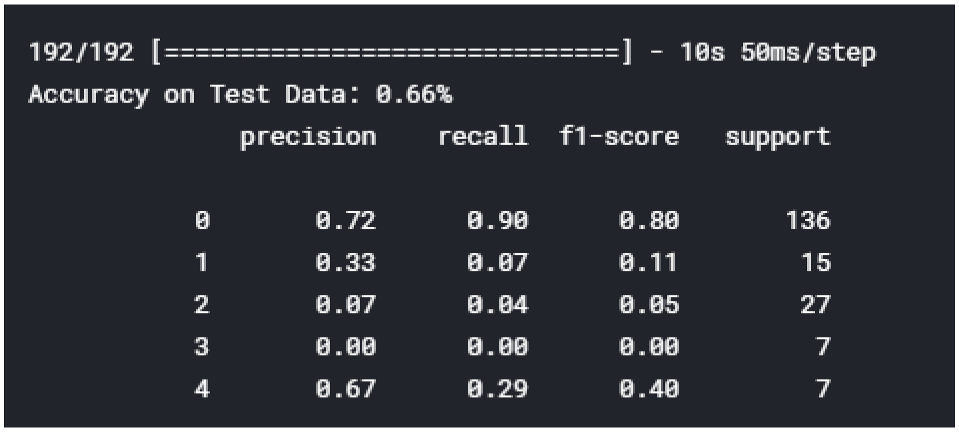
```
fig, ax1 = plt.subplots(1,1, figsize = (6, 6), dpi = 150)
ax1.plot(fpr, tpr, 'b.-
', label = 'Model Prediction (AUC: %2.2f)' % roc_auc_score(sick_vec, sick_sco
re))
ax1.plot(fpr, fpr, 'g-', label = 'Random Guessing')
ax1.legend()
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
```



8.EVALUATATION

Here we evaluate the results by loading the colab version of the model and seeing how the predictions look on the results. We then visualize spec

```
from sklearn.metrics import accuracy_score, classification_report
pred_Y = retina_model.predict(test_X, batch_size = 32, verbose = True)
pred_Y_cat = np.argmax(pred_Y, -1)
test_Y_cat = np.argmax(test_Y, -1)
print('Accuracy on Test Data: %2.2f%%' % (accuracy_score(test_Y_cat, pred_Y_c
at)))
print(classification_report(test_Y_cat, pred_Y_cat))
```

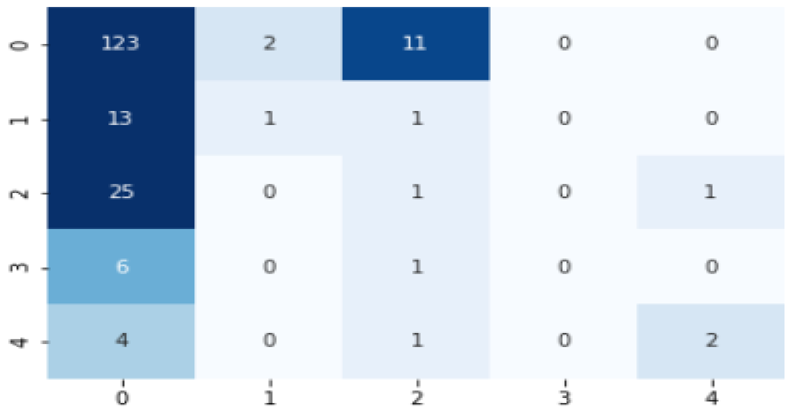


```
192/192 [==============================] - 10s 50ms/step
Accuracy on Test Data: 0.66%
              precision    recall  f1-score   support

           0       0.72      0.90      0.80       136
           1       0.33      0.07      0.11        15
           2       0.07      0.04      0.05        27
           3       0.00      0.00      0.00         7
           4       0.67      0.29      0.40         7
```

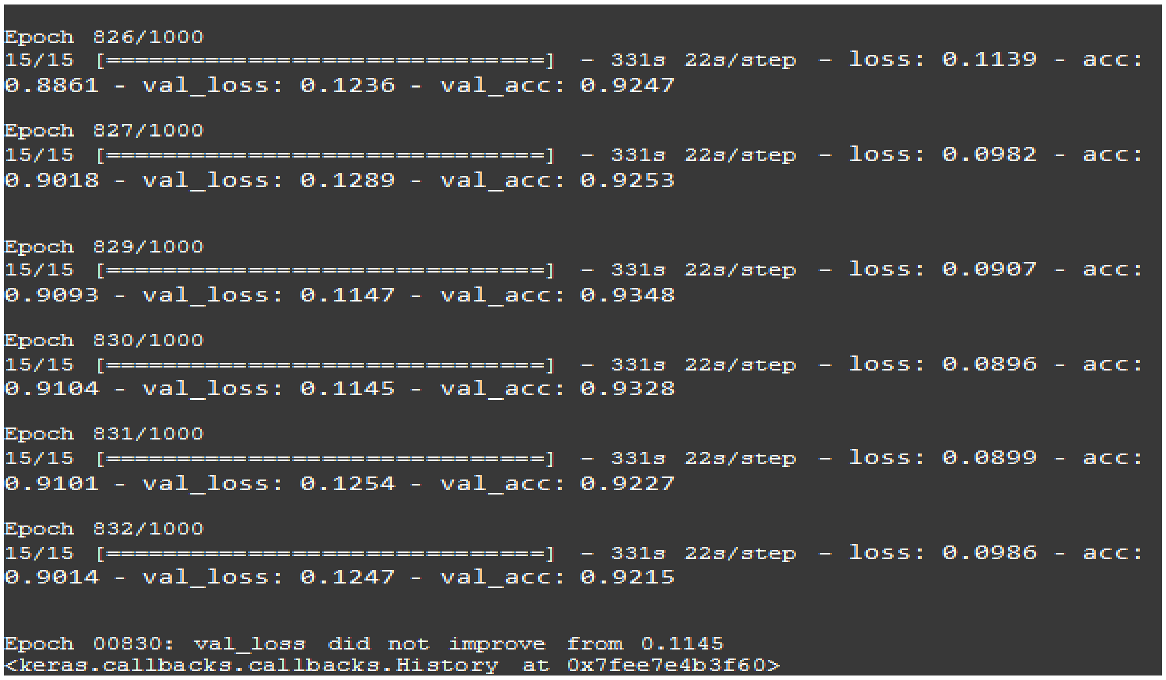
Fig: classification report

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(test_Y_cat, pred_Y_cat),
            annot=True, fmt="d", cbar = False, cmap = plt.cm.Blues, vmax = te
st_X.shape[0]//16)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7a7f0342e8>
```



```
score = model.evaluate(train_ids,valid_ids,verbose=0)
print(score)
0.9104431552887
```



```
Epoch 826/1000
15/15 [==============================]  - 331s 22s/step  - loss: 0.1139 - acc:
0.8861 - val_loss: 0.1236 - val_acc: 0.9247

Epoch 827/1000
15/15 [==============================]  - 331s 22s/step  - loss: 0.0982 - acc:
0.9018 - val_loss: 0.1289 - val_acc: 0.9253


Epoch 829/1000
15/15 [==============================]  - 331s 22s/step  - loss: 0.0907 - acc:
0.9093 - val_loss: 0.1147 - val_acc: 0.9348

Epoch 830/1000
15/15 [==============================]  - 331s 22s/step  - loss: 0.0896 - acc:
0.9104 - val_loss: 0.1145 - val_acc: 0.9328

Epoch 831/1000
15/15 [==============================]  - 331s 22s/step  - loss: 0.0899 - acc:
0.9101 - val_loss: 0.1254 - val_acc: 0.9227

Epoch 832/1000
15/15 [==============================]  - 331s 22s/step  - loss: 0.0986 - acc:
0.9014 - val_loss: 0.1247 - val_acc: 0.9215


Epoch 00830: val_loss did not improve from 0.1145
<keras.callbacks.callbacks.History at 0x7fee7e4b3f60>
```

9.CONCLUSION.

We have implemented a novel binarization scheme for weights and activations during forward and backward propagations called BCNNs.

The binarization scheme proposed in this work is parallelizable and hardware friendly, and the impact of such a method on specialized hardware implementations of CNNs could be major, by replacing most multiplications in convolution with bitwise operations.

The potential to speed-up the test-time inference might be very useful for real-time embedding systems. Our model ran on kaggle dataset which gave the accuracy upto 91.04% by using high end GPUs.

Future work includes the extension of those results to other tasks such as object detection and other models such as RNN.

10.REFERENCES.

[1] **Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to + 1 or –1**

[2] Horowitz, Mark. Computing's Energy Problem (and what we can do about it). IEEE International Solid State Circuits Conference, pp. 10–14, 2014.

[3] https://www.kaggle.com/tanlikesmath/diabetic-retinopathy-resized

[4]https://numpy.org/

[5]https://pandas.pydata.org/

[6]https://keras.io/ https://github.com/keras-team

[7]https://www.tensorflow.org/

[8]https://medium.com/@saedhussain/google-colaboratory-and-kaggle-datasets-b57a83eb6ef8

[9] Deep Learning book with Python by Francois Chollet

[10] Hands on Machine Learning with sklearn,tensorflow,keras:Concepts and Techniques to build Intelligent Systems book by GeronAurelien.

[11] Deep Learning book by IanGoodfellow