PROJECT REPORT

On

Acute Lymphoblastic Leukemia (ALL) detection using Deep Learning Models

By

Hammad Ul Eaman Mir

CSE-15-118

Rizwan Un Nissa

CSE-15-69

Under the Supervision of

Dr. Ahsan Hussain

Assistant Professor

Computer Sciences and Engineering

Mr. Khalid Pandit

Assistant Professor

Computer Sciences and Engineering

ISLAMIC UNIVERSITY OF SCIENCE & TECHNOLOGY

2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ISLAMIC UNIVERSITY OF SCIENCE AND TECHNOLOGY

1-University Avenue, Awantipora, Pulwama, Jammu & Kashmir 192122

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ISLAMIC UNIVERSITY OF SCIENCE AND TECHNOLOGY
AWANTIPORA 192122 (INDIA)

CERTIFICATE

We hereby certify that the work which has being carried out in this project entitled **Acute Lymphoblastic Leukemia (ALL) detection using Deep Learning**, in partial fulfilment of the requirement for the award of degree of "Bachelors of Technology in Computer Science and Engineering" submitted to Department of Computer Science and Engineering, Islamic University of Science and Technology, Awantipora is an authentic record of our own work carried out under the supervision of **Dr. Ahsan Hussain**.

Hammad Ul Eaman Mir
CSE-15-118
IUST0115007218

Rizwan Un Nissa
CSE-15-69
IUST0115006905

This is to certify that the statements made above by the candidates are correct and true to the best of our knowledge.

Supervisor	**Co-Supervisor**	**Head of Department**	**Project Coordinator**
Dr. Ahsan Hussain	**Mr. Khalid Pandit**	**Dr. Assif Assad**	**Ms. Nailah Afshan**
Assistant Professor	Assistant Professor	Assistant Professor	Assistant Professor
Department of CSE	Department of CSE	Department of CSE	Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ISLAMIC UNIVERSITY OF SCIENCE AND TECHNOLOGY
AWANTIPORA 192122 (INDIA)

ACKNOWLEDGEMENT

We would like to express our deep and sincere gratitude to our supervisor, **Dr. Ahsan Hussain** of Computer Science and Engineering department for his unflagging support and continuous encouragement throughout the project work.

Our Sincere gratitude to **Mr. Khalid Pandit** and **Mr. Rayees Ahmad Dar** for their continuous support and assistance.

We would also like to thank **Mr. Asrar Ul Haq Nehvi** for sharing his work which helped us to overcome multiple hurdles that we faced throughout our project.

We would like to acknowledge the collaborative support of all the faculty and staff members of CSE department.

Hammad Ul Eaman Mir

Rizwan Un Nissa

ABSTRACT

Acute Lymphoblastic Leukemia (ALL) is a cancer of the lymphoid line of blood cells characterized by the development of large numbers of immature lymphocytes. Being an acute cancer, early detection plays a vital role in the successful treatment of the subjects. Extensive research has been done over the application of machine learning algorithms for detection of ALL but use of Deep Learning (DL) models is relatively scarce. For our project we study and compare the applicability of available DL architectures and also developed our own model for detection of ALL. Our model was able to achieve the best accuracy of **93.02%**.

Table of Contents

List of Figures

List of Tables

1. Introduction

Cancer is a group of diseases involving abnormal cell growth with the potential to invade or spread to other parts of the body [1][2]. Cancer has a major effect on societies worldwide as it is one of the leading causes of deaths. In 2015, about 90.5 million people were known to be cancer affected with about 14.1 million new cases occurring each year (not including skin cancer and other melanomas), and led to deaths of 8.8 million people [3][4][5]. Leukemia is one of the most common cases of cancers. Acute Lymphoblastic Leukemia(ALL) is a cancer of the lymphoid line of blood cells characterized by the development of large numbers of immature lymphocytes [6]. As an acute leukemia, ALL progresses rapidly and is typically fatal within weeks or months if left untreated [7]. Table 1 shows types of ALL, along with the type of cells affected as well as characteristics of the affected cells. In 2015 there were 876,000 cases of ALL which resulted in 111,000 deaths [3][5]. In 2019 there have been 5930 cases of ALL and 1500 deaths so far [8]. Diagnosing ALL begins with a thorough medical history, physical examination, complete blood count, and blood smears. Bone Marrow Biopsy. Lumbar Puncture & Medical Imaging(Metastasis). Pathological examination and Cytogenetics [9].

The method through microscopic images to detect and classify various cells (not limited to ALL cells) can be viewed as a kind of well-defined and investigated computer vision (CV) problems. Recently, with developing of deep learning (DL) based techniques, researches have investigated the potential use of applying DL methods in cell image classification problems.

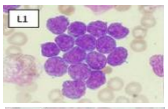
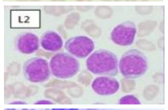
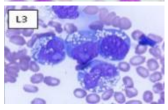
	FAB Subtype	**Cell Type**	**Characteristics**
	ALL - L1	T cell or pre-B cell	Small and homogeneous (uniform) cells
	ALL - L2	T cell or pre-B cell	Large and heterogeneous (varied) cells
	ALL - L3	B cell	Large and varied cells with vacuoles

Table 1: ALL types with affected cell types and cell characteristics.

1.1 Problem Statement

As an acute leukaemia, ALL progresses rapidly and is typically fatal within weeks or months if left untreated [7]. Diagnosing ALL begins with a thorough medical history, physical examination, complete blood count, and blood smears. Bone Marrow Biopsy. Lumbar Puncture & Medical Imaging(Metastasis). Pathological examination and Cytogenetics [9] and are subject to human bias. The two main challenges that we faced are:

- Due to low variance in our dataset classification proves to be a challenging task.
- Transfer learning renders to be ineffective for our specific dataset.

1.2 Aim

Our aim is to study the applicability of different available DL architectures and to devise a new, better model/architecture for ALL detection from microscopic ALL histopathological images. These images are provided as input to the DL models which classify the images as ALL cells or normal cells.

1.3 Main Objectives

Following are the objectives we strived to achieve during our project work:

1. Choosing the appropriate dataset.
2. Research current available architectures for image classification.
3. Applying the various architectures for classification.
4. Comparing the performance of various architectures.
5. Devising a new model for improved classification performance.

2. Background and Literature Survey

Work on Computer Vision dates back to the 1960's with the research in Artificial Intelligence. While understanding the human perception researchers tried to build similar systems for computers. Computer Vision really came into the limelight after the arrival of large scale computing power along with easy access to massive amounts of data. GPUs were instrumental in development of accurate Object Recognition Model's as they worked in parallel and could speed up large complex calculations. Object classification and detection are two fundamental problems in computer vision and pattern recognition. They play fundamental and crucial roles in many applications, e.g., intelligent visual surveillance, image and video retrieval and web content analysis.

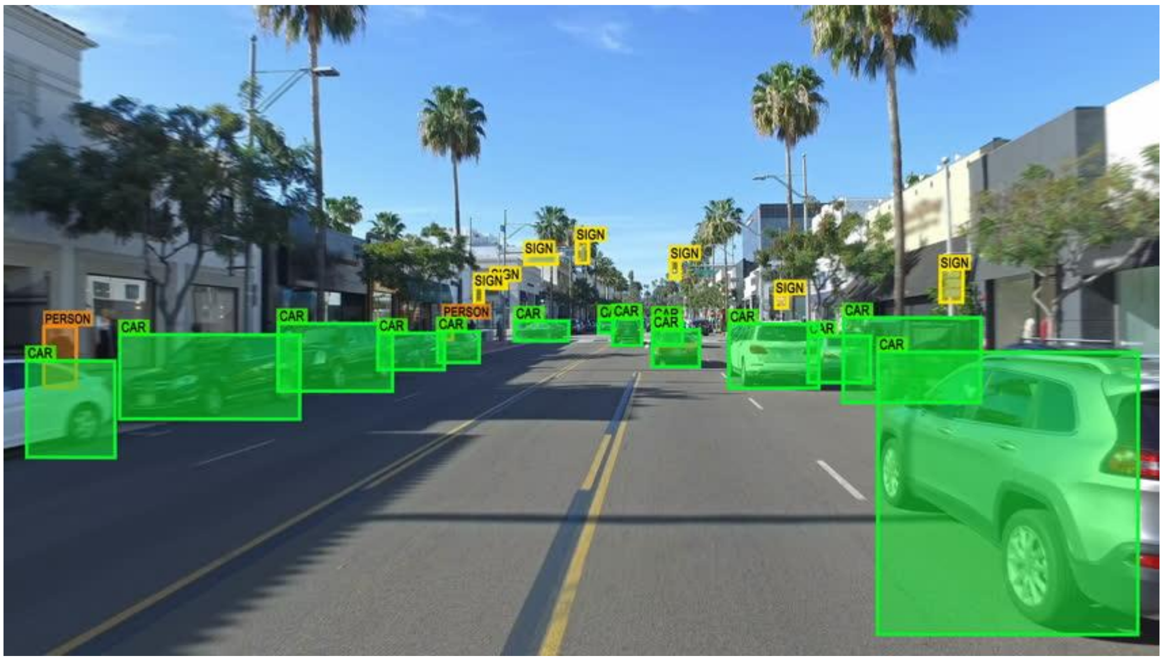


Figure 1: Object Detection, Localization and Classification/Recognition.

In the last few years, the field of machine learning has made tremendous progress on addressing these difficult problems. In particular, we've found that a kind of model called a deep convolutional neural network can achieve reasonable performance on hard visual recognition tasks -- matching or exceeding human performance in some domains.

Researchers have demonstrated steady progress in computer vision by validating their work against ImageNet -- an academic benchmark for computer vision.

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in visual object recognition, object detection and many other domains. Deep learning discovers intricate structure in large datasets. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio [10].

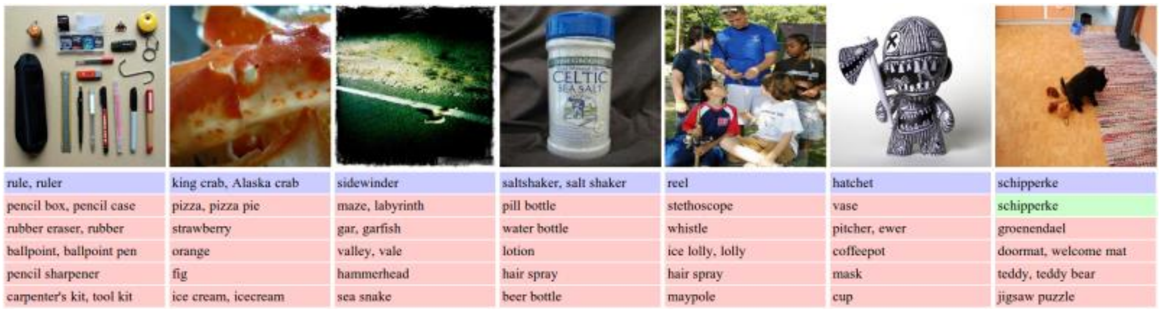


Figure 2: Example of Classification/Recognition on IMAGENET.

2.1 Previous work

Deep Convolutional Neural Networks have been extensively studied and used to classify large scale image data sets like ImageNet [11].

Since the early 2000s, ConvNets have been applied with great success to the detection, segmentation and recognition of objects and regions in images. These were all tasks in which labelled data was relatively abundant [10].

The previous models for automated ALL detection have proposed using machine learning models and have mostly used ALL-IDB dataset.

Luis H. S. Vogado, Rodrigo de M. S. Veras, Alan R. Andrade, Flavio H. D. de Araujo in 2016 proposed a method [12] for Acute lymphoblastic leukemia segmentation using two color channels and unsupervised algorithm k means. Improvement in the technique, they intend to develop a classification step, so we can effectively conclude the creation of an automated detection system.

Jakkrich Laosai and Kosin Chamnongthai [13] proposed a method for classifying leukemia by using cluster of differentiation (CD) method. The classification proposed 3 subtypes of ALL and 8 subtypes of AML that are characterized by unique morphologic, immunologic, and cytogenetic features Immunologic (MIC group) markers cluster of differentiation (CD). They get accuracy 93.89%.

Jakkrich Laosai and Kosin Chamnongthai [14] used SVM and k means algorithm. The method has been evaluate using k means clustering. And they get 92% performance from SVM.

Mashiat Fatma and Jaya Sharma used neural network for leukemia classification [15]. For this the different features are extracted from the input images and based on those features the dataset is created. Then the dataset is utilized as input data to a neural network for training purposes. This neural network is designed and created to categorize the images according to their corresponding leukemia type.

Faisal Asadi et.al. used a combination of back-propagation neural network and blood cells extraction imagery to classify acute leukemia. They used back-propagation neural network algorithm to extract the characteristics of blood cells ALL and AML and digital image processing for the leukemia identification technique. They were able to identify ALL and AML cases about 93.3% and 80%, respectively, and they achieved about 86.66% accuracy in classifying the leukemia acute types on average [16].

Ahmed et.al. used Elephant Herd Optimization with Feed Forward Neural Networks for ALL detection. They used various machine learning and deep learning techniques for classification purpose and achieved an accuracy of 91.8% [17].

3. Work Carried Out

The idea of the project came out of the intention to make a difference in people's lives. Having already delved in the python programming language and machine learning, we already had the resources that we needed to carry out the work. But, the problem to work upon still remained a question. Since our purpose was to make human life better, medicinal domain attracted our attention. Diagnosing a disease at a treatable moment would lead to saving of many lives. With this motivation we researched about different types of cancers for months and at last came to ALL. ALL having unfavorable prognosis if detected late makes it necessary to have early diagnosis. Having the tools, problem and the motivation, we embarked on a journey to reach our set target. We spent 3 weeks at IIT Mandi to learn applied deep learning techniques and it took us nearly 5 months after that to complete our project. Along this journey way we faced many technical hurdles and challenges that while frustrating at times helped us understand and learn new concepts. In this section we will document all our efforts and the work we undertook to go from the beginning to the end of our project.

We divided the project work into six phases:

1. Research and analysis.

2. Finding/Selecting an image dataset.

3. Image pre-processing.

4. Build image classification models.

5. Proposed Model.

6. Network Training.

7. Compare model accuracies on test set.

3.1 Research and analysis

The research work progressed chronologically in the following order:

1. Research in to the various domains of Computer Vision, get a broad perspective of the field and get familiar with the technical language of machine learning specifically deep learning.

2. Perform background survey on object recognition.

3. Understand the state-of-the-art techniques used for image classification.

4. Know the common tools, languages, libraries and frameworks used to build Image Recognition Models.

Since this field is rapidly evolving and active research is being published every other day, we found open published research papers to be our best guide along with helpful blog posts by experts.

The rest of this section gives a brief summary of each topic that was researched for this project.

3.1.1 Machine Learning

Machine Learning is the study of automated learning by machines using data. Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning [10].

3.1.2 Deep learning

Deep learning can be considered as a sub-branch of machine learning, it is growing in popularity because of the promising results it has shown in various areas. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned. For classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations. Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years. It has turned out to be very good at discovering intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government [10].

3.1.3 Gradient Descent

Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks.

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \epsilon R$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla\theta J(\theta)$ w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.

1. Batch gradient descent

Batch gradient descent computes the gradient for the entire training dataset and hence can be very slow and intractable for datasets that do not fit in memory. It also does not allow us to update our model online, i.e. with new examples on-the-fly.

```
for i in range(nb_epochs):
    params_grad = evaluate_gradient(loss_function , data, params)
    params = params - learning_rate * params_grad
```

2. 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.

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(loss_function , example , params)
        params = params - learning_rate * params_grad
```

3. Mini batch gradient descent

Mini-batch gradient descent takes the best of both worlds and performs an update for every mini-batch of n training examples.

```
for i in range(nb_epochs):
    np.random.shuffle(data)
for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function , batch , params)
    params = params - learning_rate * params_grad
```

3.1.4 Backpropagation

Multilayer Neural Network architectures can be trained by simple stochastic gradient descent. As long as the modules are relatively smooth functions of their inputs and of their internal weights, one can compute gradients using the backpropagation procedure. The idea that this could be done, and that it worked, was discovered independently by several different groups during the 1970s and 1980s.

The backpropagation procedure to compute the gradient of an objective function with respect to the weights of a multilayer stack of modules is nothing more than a practical application of the chain rule for derivatives. The key insight is that the derivative (or gradient) of the objective with respect to the input of a module can be computed by working backwards from the gradient with respect to the output of that module (or the input of the subsequent module) The backpropagation equation can be applied repeatedly to propagate gradients through all modules, starting from the output at the top (where the network produces its prediction) all the way to the bottom (where the external input is fed). Once these gradients have been computed, it is straightforward to compute the gradients with respect to the weights of each module [10].

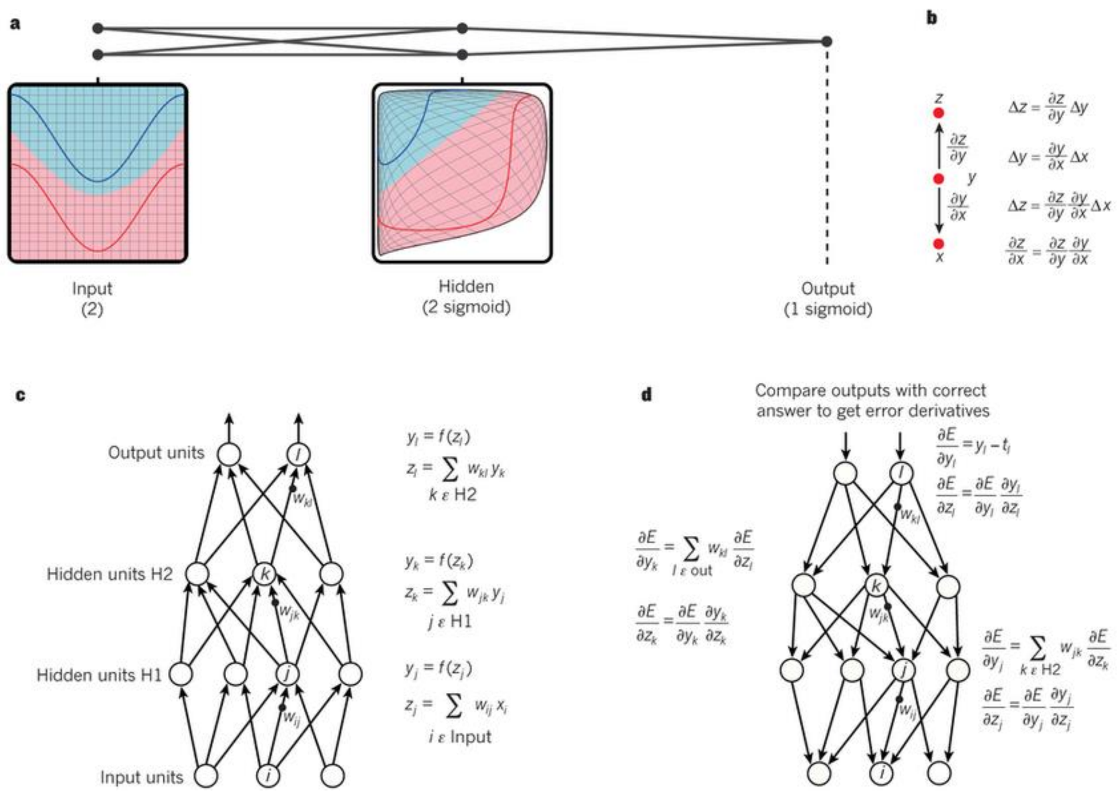

a
Input (2)
Hidden (2 sigmoid)
Output (1 sigmoid)
b
$\Delta z = \frac{\partial z}{\partial y}\Delta y$
$\Delta y = \frac{\partial y}{\partial x}\Delta x$
$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$
c
Output units
Hidden units H2
Hidden units H1
Input units
$y_l = f(z_l)$
$z_l = \sum_{k \epsilon H2} w_{kl} y_k$
$y_k = f(z_k)$
$z_k = \sum_{j \epsilon H1} w_{jk} y_j$
$y_j = f(z_j)$
$z_j = \sum_{i \epsilon Input} w_{ij} x_i$
d
Compare outputs with correct answer to get error derivatives
$\frac{\partial E}{\partial y_l} = y_l - t_l$
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$
$\frac{\partial E}{\partial y_k} = \sum_{l \epsilon out} w_{kl}\frac{\partial E}{\partial z_l}$
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_j} = \sum_{k \epsilon H2} w_{jk}\frac{\partial E}{\partial z_k}$
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$

Figure 3: Backpropagation

3.1.5 Activation Function

Activation functions are really important for an Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable. They introduce non-linear properties to our Network. Their main purpose is to convert a input signal of a node in a NN to an output signal. That output signal now is used as a input in the next layer in the stack.

Specifically in a NN we do the sum of products of inputs (X) and their corresponding Weights (W) and apply a Activation function $f(X)$ to it to get the output of that layer and feed it as an input to the next layer.

ReLU

ReLU has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergence from Tanh function. It avoids and rectifies the vanishing gradient problem in Sigmoid and Tanh but can suffer from dead neurons during training. To fix the problem of dead neurons Leaky ReLU was introduced.

Mathematical form of ReLU is simple and efficient:

$$R(x) \ = \ max(0, x)$$

ReLU avoids and rectifies the vanishing gradient problem. Almost all deep learning Models use ReLU nowadays. It's used in hidden layers only and not in the output layer.

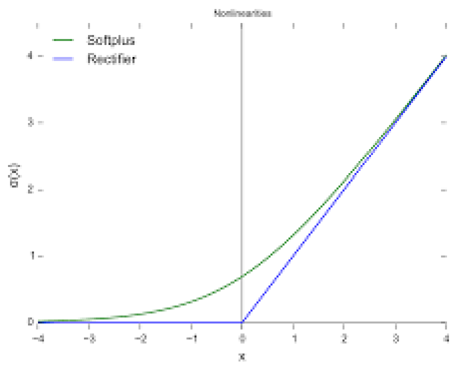


Figure 4: ReLU.

Softmax

For output layers use a Softmax function for a classification problem to compute the probabilities for the classes. Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0.

$$Softmax\ equation: p(y = j|x) = \frac{e^{W_j^T x + b_j}}{\sum_{k=K} e^{W_k^T x + b_k}}$$

This formula basically extends the formula for logistic regression into multiple classes.

3.1.6 Loss functions

In a supervised learning problem, we have to find the error between the actual values and the predicted value. There can be different metrics which can be used to evaluate this error. This metric is often called loss function or cost function or objective function. Loss functions are helpful to train a neural network. Given an input and a target, they calculate the loss, i.e. difference between output and target variable.

Loss functions fall under three major categories:

1. Regressive loss functions - used in case of regression problems (e.g. mean square error)

2. Classification loss functions - used in case of classification problems (e.g. binary cross-entropy)

3. Embedding loss functions - used to measure whether two inputs are similar or dissimilar (e.g. cosine similarity)

Cross-Entropy

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

For binary classification cross-entropy can be calculated as:

$$L = -(y\,Log(p) + (1-y)Log(1-p))$$

3.1.7 Convolutional Neural Network

A convolutional neural network (CNN) is used to progressively extract higher- and higher-level representations of the image content. Instead of pre-processing the data to derive features like textures and shapes, a CNN takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they constitute.

To start, the CNN receives an input feature map: a three-dimensional matrix where the size of the first two dimensions corresponds to the length and width of the images in pixels. The size of the third dimension is 3 (corresponding to the 3 channels of a color image: red, green, and blue). The CNN comprises a stack of modules, each of which performs three operations.

1. Convolution

2. Activation (usually ReLU)

3. Pooling

Convolution

A convolution extracts tiles of the input feature map, and applies filters to them to compute new features, producing an output feature map, or convolved feature (which may have a different size and depth than the input feature map). Convolutions are defined by two parameters:

Size of the tiles that are extracted (typically 3x3 or 5x5 pixels). The depth of the output feature map, which corresponds to the number of filters that are applied.

During a convolution, the filters (matrices the same size as the tile size) effectively slide over the input feature maps grid horizontally and vertically, one pixel at a time, extracting each corresponding tile.

For each filter-tile pair, the CNN performs element-wise multiplication of the filter matrix and the tile matrix, and then sums all the elements of the resulting matrix to get a single value. Each of these resulting values for every filter-tile pair is then output in the convolved feature matrix.

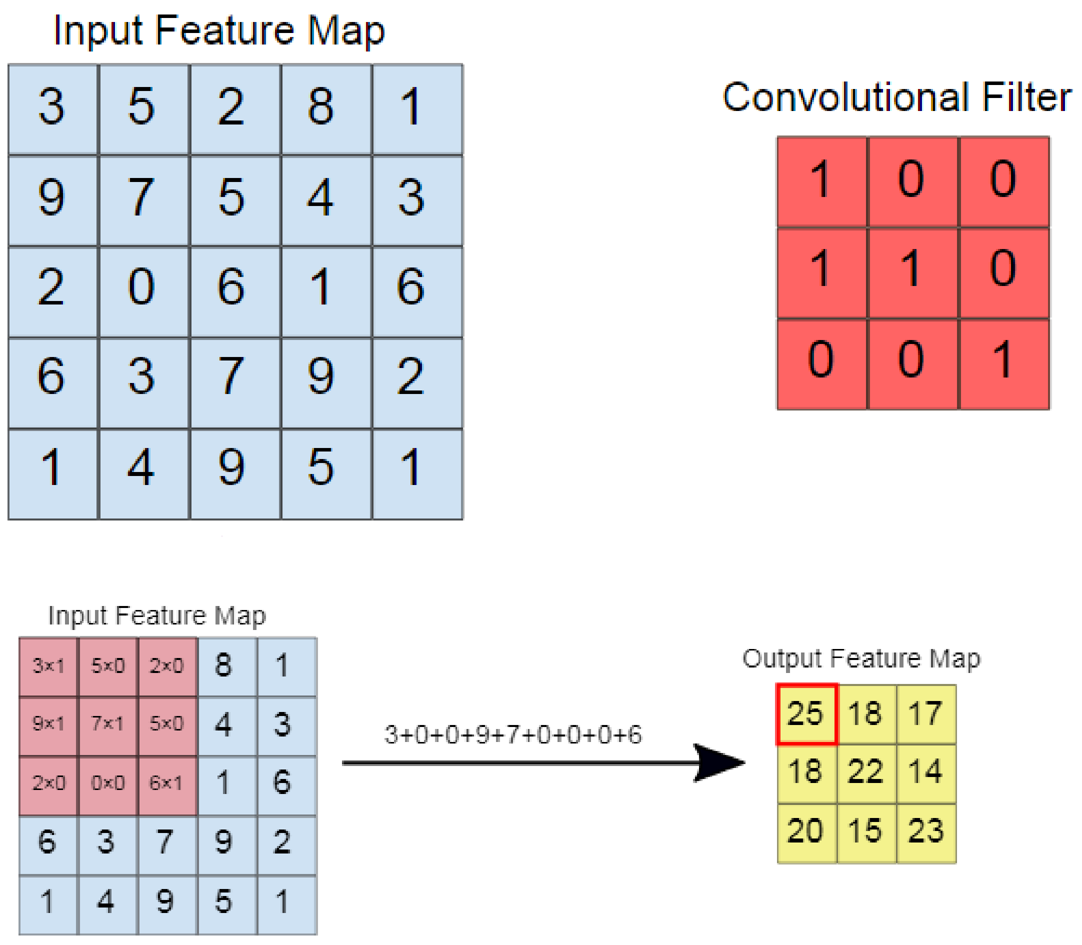


Figure 5: Convolution operation.

During training, the CNN "learns" the optimal values for the filter matrices that enable it to extract meaningful features (textures, edges, shapes) from the input feature map. As the number of filters (output feature map depth) applied to the input increases, so does the number of features the CNN can extract. However, the tradeoff is that filters compose the majority of resources expended by the CNN, so training time also increases as more filters are added. Additionally, each filter added to the network provides less incremental value than the previous one, so engineers aim to construct networks that use the minimum number of filters needed to extract the features necessary for accurate image classification.

Activation – ReLU

Following each convolution operation, the CNN applies a Rectified Linear Unit (ReLU) transformation to the convolved feature, in order to introduce nonlinearity into the model.

Pooling

After ReLU comes a pooling step, in which the CNN down samples the convolved feature (to save on processing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature information. A common algorithm used for this process is called max pooling.

Max pooling operates in a similar fashion to convolution. We slide over the feature map and extract tiles of a specified size. For each tile, the maximum value is output to a new feature map, and all other values are discarded. Max pooling operations take two parameters:

- Size of the max-pooling filter (typically 2x2 pixels).
- Stride: the distance, in pixels, separating each extracted tile. Unlike with convolution, where filters slide over the feature map pixel by pixel, in max pooling, the stride determines the locations where each tile is extracted. For a 2x2 filter, a stride of 2 specifies that the max pooling operation will extract all non-overlapping 2x2 tiles from the feature map.

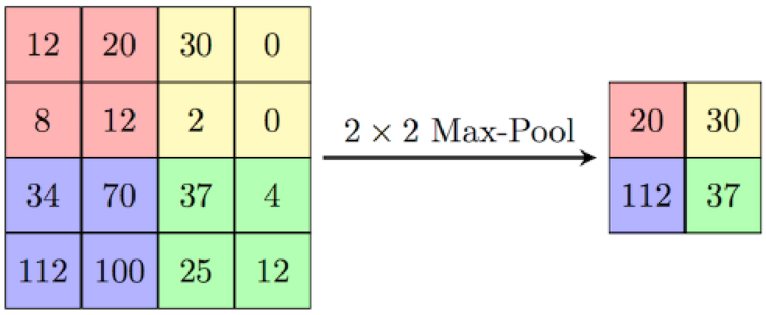


Figure 6: MaxPooling

Fully Connected

At the end of a convolutional neural network are one or more fully connected layers (when two layers are "fully connected," every node in the first layer is connected to every node in the second layer). Their job is to perform classification based on the features extracted by the convolutions. Typically, the final fully connected layer contains a Softmax activation function, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict.

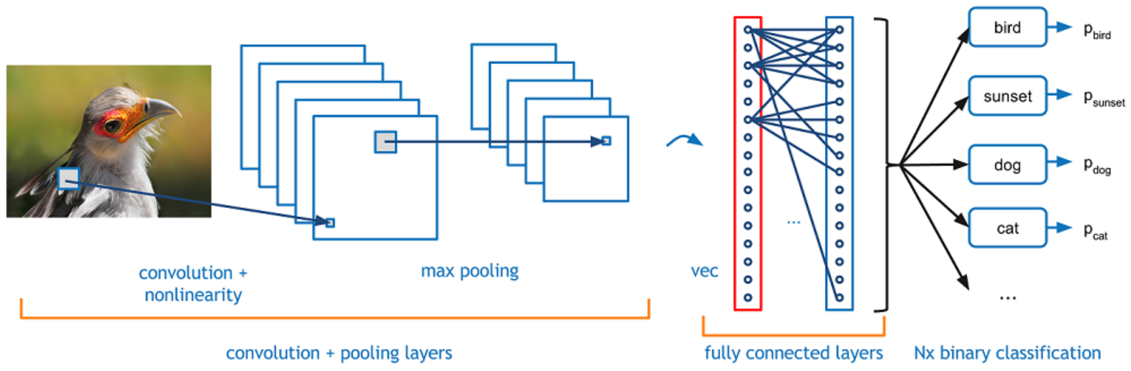


Figure 7: CNN followed by a Fully Connected Network.

3.1.8 Large scale visual recognition models

In the last few years, the field of machine learning has made tremendous progress on addressing Image Classification problems. In particular, a kind of model called a deep convolutional neural network can achieve reasonable performance on hard visual recognition tasks -- matching or exceeding human performance in some domains.

Researchers have demonstrated steady progress in computer vision by validating their work against ImageNet -- an academic benchmark for computer vision. This is a standard task in computer vision, where models try to classify entire images into 1000 classes, like "Zebra", "Dalmatian", and "Dishwasher". Successive models continue to show improvements, each time achieving a new state-of-the-art result:

1. AlexNet
2. VGG
3. Inception (GoogLeNet)
4. ResNet

To compare models, examine how often the model fails to predict the correct answer as one of their top 5 guesses -- termed "top-5 error rate".

- AlexNet achieved a top-5 error rate of 15.3%
- VGG achieved 7.3%
- Inception (GoogLeNet) achieved 6.67%
- ResNet reaches 3.6%

3.1.9 Building Image Recognition Models: Tools, Languages and Libraries

A lot of effort has been put in by the community of researchers and software companies to create useful libraries making it easy to work in the emerging field of Machine Learning.

There are two major languages dominating the field:

1. R
2. Python

While R used to be in the lead it has now been surpassed by python and most of the latest work is being done in it.

Some popular deep learning frameworks available for python at present are:

1. Tensorflow [18]
2. Keras [19]
3. Pytorch [20] etc.

These frameworks are augmented by the various data science libraries for python like:

1. Scikit Learn [21]
2. NumPy [22]
3. Pandas [23]
4. Matplotlib [24] etc.

3.2 Finding/Selecting an image dataset

We decided not to work on the All-IDB dataset since it's a fairly older dataset and consists of only a few 100 data points. Instead we decided to work on the ALL dataset that was released as part of an ISBI challenge on codalab's website. The dataset consists of 10,600 WBC nuclei microscopic images with an image size of $450 * 450$. The main challenge faced with the classification of normal cells from malignant cells in this dataset in the close visual similarity between the images from the two classes.

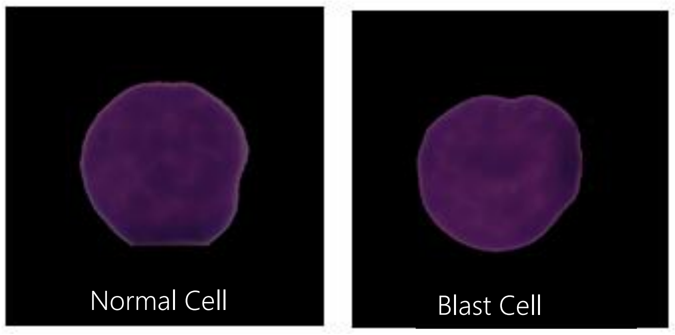


Figure 8: Dataset images with close visual similarities.

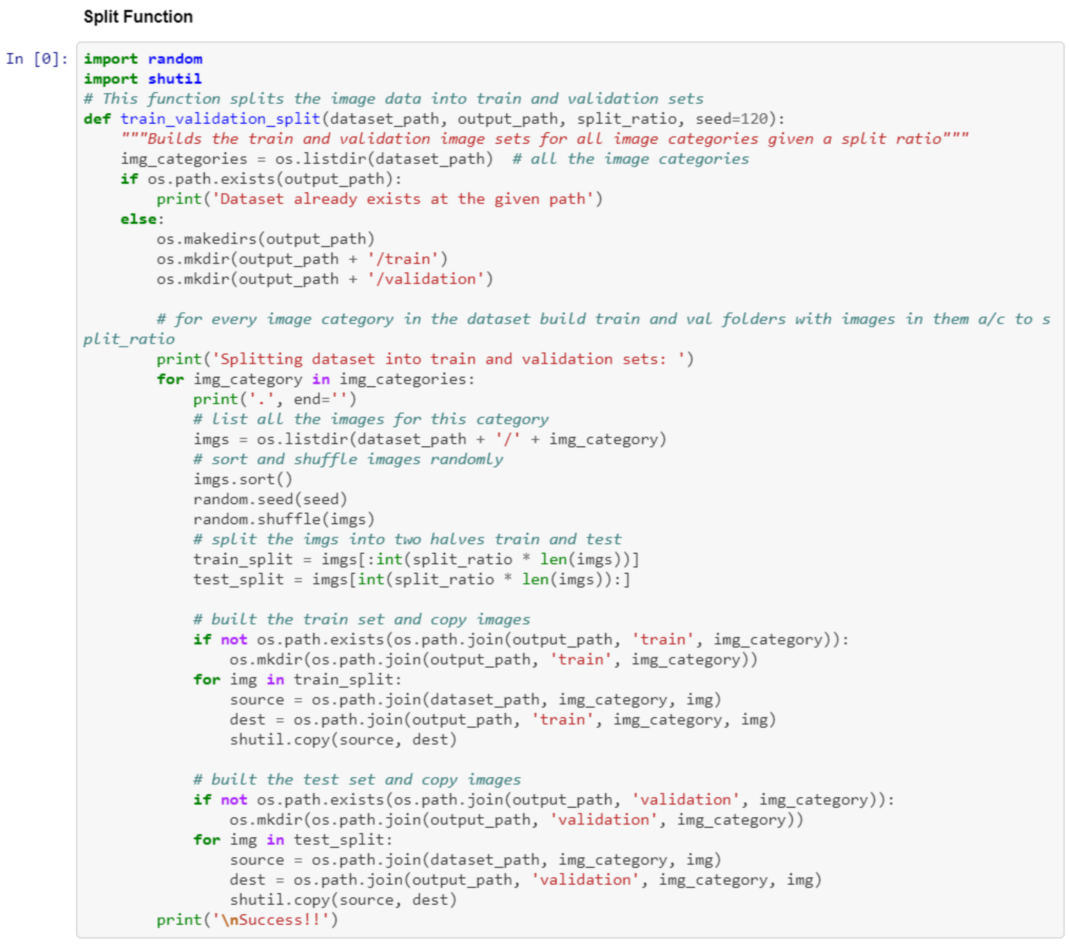
3.3 Data preparation and pre-processing

We organized our data in two folders with the folder names, **all** for blast cells and **hem** for normal cells, serving as the label for the images in them.

We split our data into two parts, for training and testing respectively, with a split ratio of 8:2

- Training set - 8,480 images
- Testing set - 2,120 images

To get a good split, it is important to pick images randomly without any bias. Instead of doing this manually we used a custom python script which takes as input the dataset path, the output path, split ratio and optionally a random seed value [25]. The script builds the train and test image datasets for all image categories given a split ratio randomly by first shuffling all the images randomly then splitting them into two halves.



Split Function

```
In [0]: import random
        import shutil
        # This function splits the image data into train and validation sets
        def train_validation_split(dataset_path, output_path, split_ratio, seed=120):
            """Builds the train and validation image sets for all image categories given a split ratio"""
            img_categories = os.listdir(dataset_path)  # all the image categories
            if os.path.exists(output_path):
                print('Dataset already exists at the given path')
            else:
                os.makedirs(output_path)
                os.mkdir(output_path + '/train')
                os.mkdir(output_path + '/validation')

                # for every image category in the dataset build train and val folders with images in them a/c to s
        plit_ratio
                print('Splitting dataset into train and validation sets: ')
                for img_category in img_categories:
                    print('.', end='')
                    # list all the images for this category
                    imgs = os.listdir(dataset_path + '/' + img_category)
                    # sort and shuffle images randomly
                    imgs.sort()
                    random.seed(seed)
                    random.shuffle(imgs)
                    # split the imgs into two halves train and test
                    train_split = imgs[:int(split_ratio * len(imgs))]
                    test_split = imgs[int(split_ratio * len(imgs)):]

                    # built the train set and copy images
                    if not os.path.exists(os.path.join(output_path, 'train', img_category)):
                        os.mkdir(os.path.join(output_path, 'train', img_category))
                    for img in train_split:
                        source = os.path.join(dataset_path, img_category, img)
                        dest = os.path.join(output_path, 'train', img_category, img)
                        shutil.copy(source, dest)

                    # built the test set and copy images
                    if not os.path.exists(os.path.join(output_path, 'validation', img_category)):
                        os.mkdir(os.path.join(output_path, 'validation', img_category))
                    for img in test_split:
                        source = os.path.join(dataset_path, img_category, img)
                        dest = os.path.join(output_path, 'validation', img_category, img)
                        shutil.copy(source, dest)
                print('\nSuccess!!')
```

Figure 9: Splitting Function

After splitting the dataset, we proceeded to pre-process the images for better and improved training all the while trying to avoid model over-fitting.

3.3.1 Crop

The borders from the images were cropped out to the maximum possible amount while making sure not to remove any part from the nucleus image from any of the images. The images were cropped from their original size of $450 * 450$ to a new size of $400 * 400$. This was done so as to retain as much information as possible during the image resize, which was necessary step owing to the resource constraints of our system.

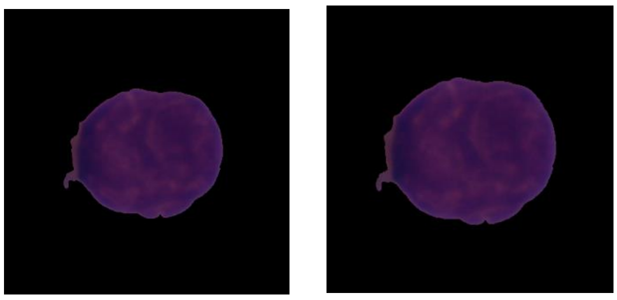
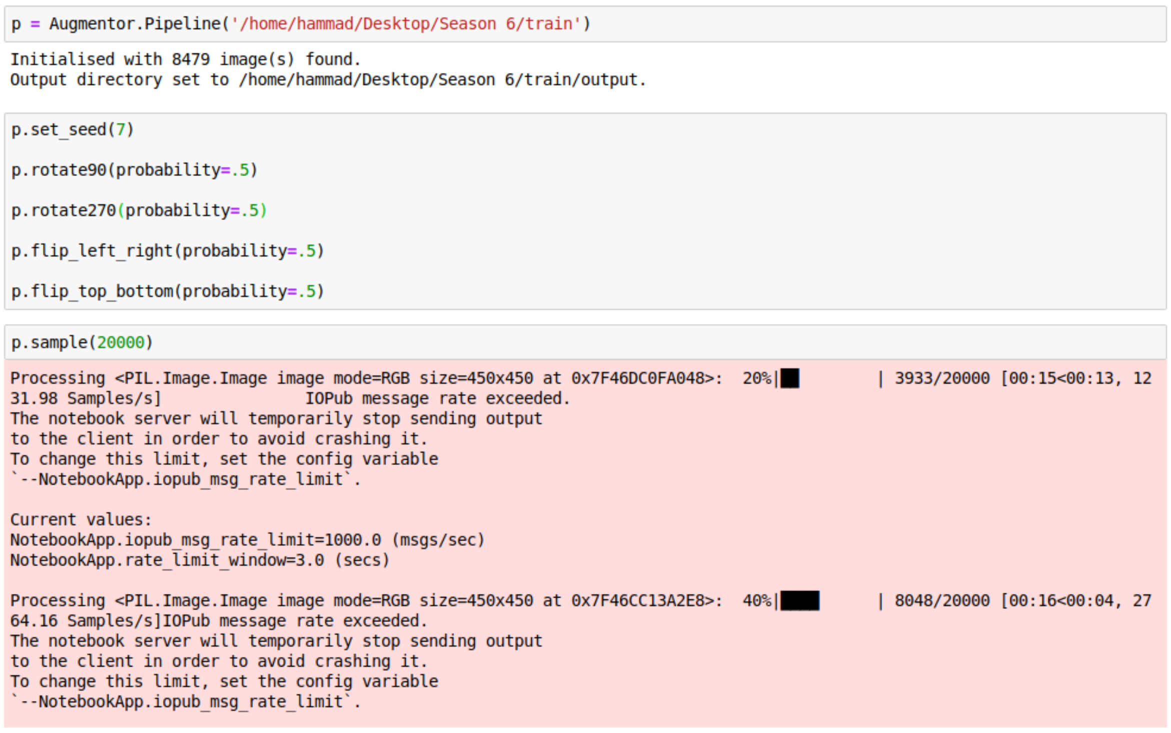


Figure 10: Normal Image(left) and Cropped Image(Right)

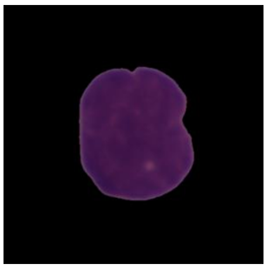
3.3.2 Augmentation

To increase the amount of training data for better classification and also to avoid overfitting, so that at training time, model will never see the exact same picture twice. We performed augmentation on our training dataset by applying random rotations and flips to them. This was done by using the "**Augmentor**" python library available on GitHub [26]. Using augmentor we increased the number of training images from 8,480 to 20,000.

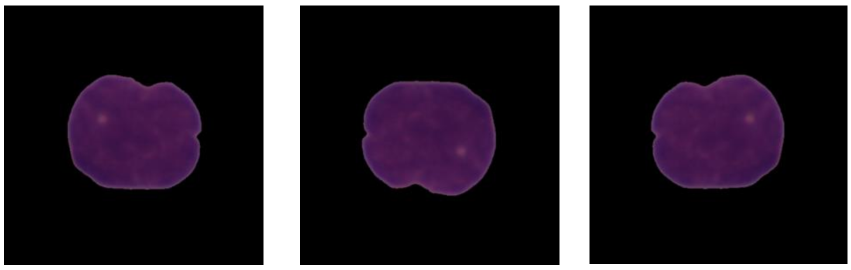


```
p = Augmentor.Pipeline('/home/hammad/Desktop/Season 6/train')

Initialised with 8479 image(s) found.
Output directory set to /home/hammad/Desktop/Season 6/train/output.

p.set_seed(7)

p.rotate90(probability=.5)

p.rotate270(probability=.5)

p.flip_left_right(probability=.5)

p.flip_top_bottom(probability=.5)

p.sample(20000)

Processing <PIL.Image.Image image mode=RGB size=450x450 at 0x7F46DC0FA048>:  20%|█         | 3933/20000 [00:15<00:13, 12
31.98 Samples/s]              IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)

Processing <PIL.Image.Image image mode=RGB size=450x450 at 0x7F46CC13A2E8>:  40%|████      | 8048/20000 [00:16<00:04, 27
64.16 Samples/s]IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.
```

Figure 11: Augmentation Function



Normal Image



Augmented Images

Figure 12: Augmented data

3.4 Build image classification models

We decided to use 4 of the most popular available CNN models due to their popularity and good results for image classification tasks. The models we used are:

1. AlexNet.

2. VGG16.

3. InceptionV3.

4. ResNet50.

We also decided to design our own model due to unsatisfactory results of the aforementioned architectures.

3.4.1 AlexNet

AlexNet is the name of a convolutional neural network, designed by Alex Krizhevsky and published with Ilya Sutskever and Krizhevsky’s PhD advisor Geoffrey Hinton, who was originally resistant to the idea of his student. AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved a top-5 error of 15.3\%, more than 10.8 percentage points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training.

AlexNet consists of eight layers; the first five are convolutional layers, some of them followed by max-pooling layers and the last three are fully connected layers. It uses the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid [11].

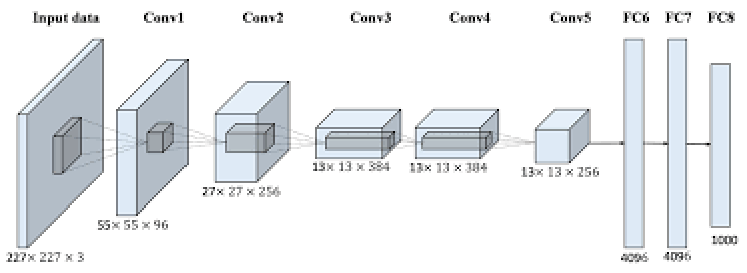


Figure 13: AlexNet block diagram

3.4.2 VGG16

VGG-16 was designed by Karen Simonyan and Andrew Zisserman (Visual Geometry Group, Department of Engineering Science, University of Oxford). The basic aim of the experiment was to understand the relationship of a neural network with regards to its depth.

The basic architecture of the VGG-16 is as follows [27]:

The input image is a fixed $224 * 224$ RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center) are used. In one of the configurations, 1×1 convolution filters are used, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2-pixel window, with stride 2. A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two layers have 4096 channels each, the third performs 1000- way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU) non-linearity.

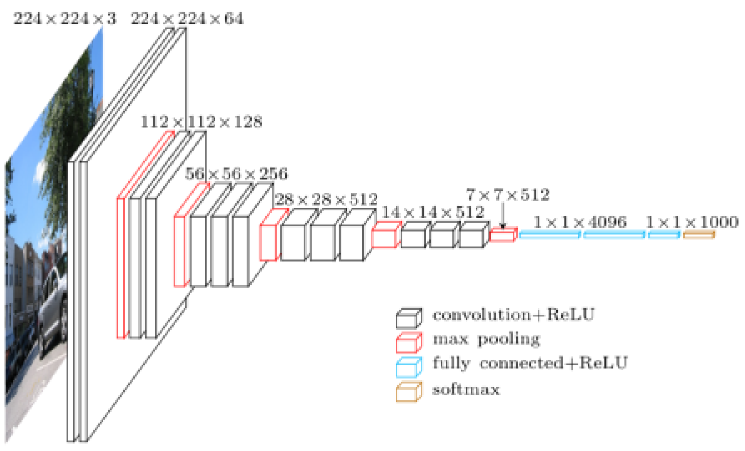


Figure 14: VGG16 block diagram

3.4.3 ResNet50.

A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex[citation needed]. Residual neural networks do this by utilizing skip connections or short-cuts to jump over some layers. In its limit as ResNets it will only skip over a single layer [28]. With an additional weight matrix to learn the skip weights it is referred to as HighwayNets [29]. With several parallel skips it is referred to as DenseNets [30]. In comparison, a non-residual neural network is described as a plain network in the context of residual neural networks.

One motivation for skipping over layers is to avoid the problem of vanishing gradients by reusing activations from a previous layer until the layer next to the current one learns its weights. During training the weights adapt to mute the previous layer and amplify the layer next to the current.

Skipping initially compresses the network into fewer layers, which speeds learning. The network gradually restores the skipped layers as it learns the feature space. During later learning, when all layers are expanded, it stays closer to the manifold and thus learns faster. A neural network without residual parts explores more of the feature space. This makes it more vulnerable to perturbations that cause it to leave the manifold, and necessitates extra training data to recover.

There are many variants of ResNet such as ResNet50, ResNet101, ResNet152, etc. For our purposes we decided to use ResNet50. It is a 50-layer Residual Network and its architecture block diagram is as shown below.

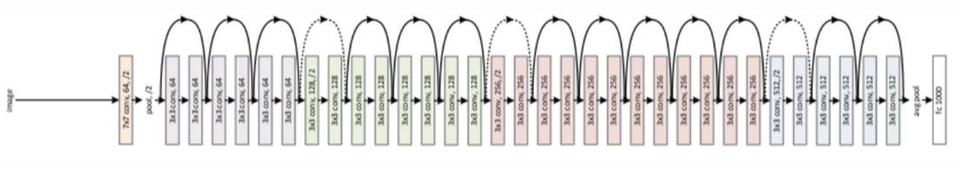


Figure 15: ResNet50 block diagram

Its detailed architecture block diagram can also be found at:

http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006.

3.4.4 InceptionV3.

The most straightforward way of improving the performance of deep neural networks is by increasing their size. This includes both increasing the depth – the number of levels – of the network and its width: the number of units at each level but the drawback being that bigger size typically means a larger number of parameters, which makes the enlarged network more prone to overfitting, especially if the number of labelled examples in the training set is limited [31].

The fundamental way of solving both issues would be by ultimately moving from fully connected to sparsely connected architectures, even inside the convolutions. The main idea of the Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components.

The exact structure of the extra network on the side, including the auxiliary classifier, is as follows:

• An average pooling layer with 5×5 filter size and stride 3, resulting in a 4×4×512 output.

• A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.

• A fully connected layer with 1024 units and rectified linear activation.

• A dropout layer with 70% ratio of dropped outputs.

• A linear layer with softmax loss as the classifier (predicting the same 1000 classes as the main classifier, but removed at inference time).

The side layers added with softmax activation for additional prediction helps in regularization of the model, hence avoiding overfitting. The architecture of the inception network and of a single layer of inception network are as shown below.

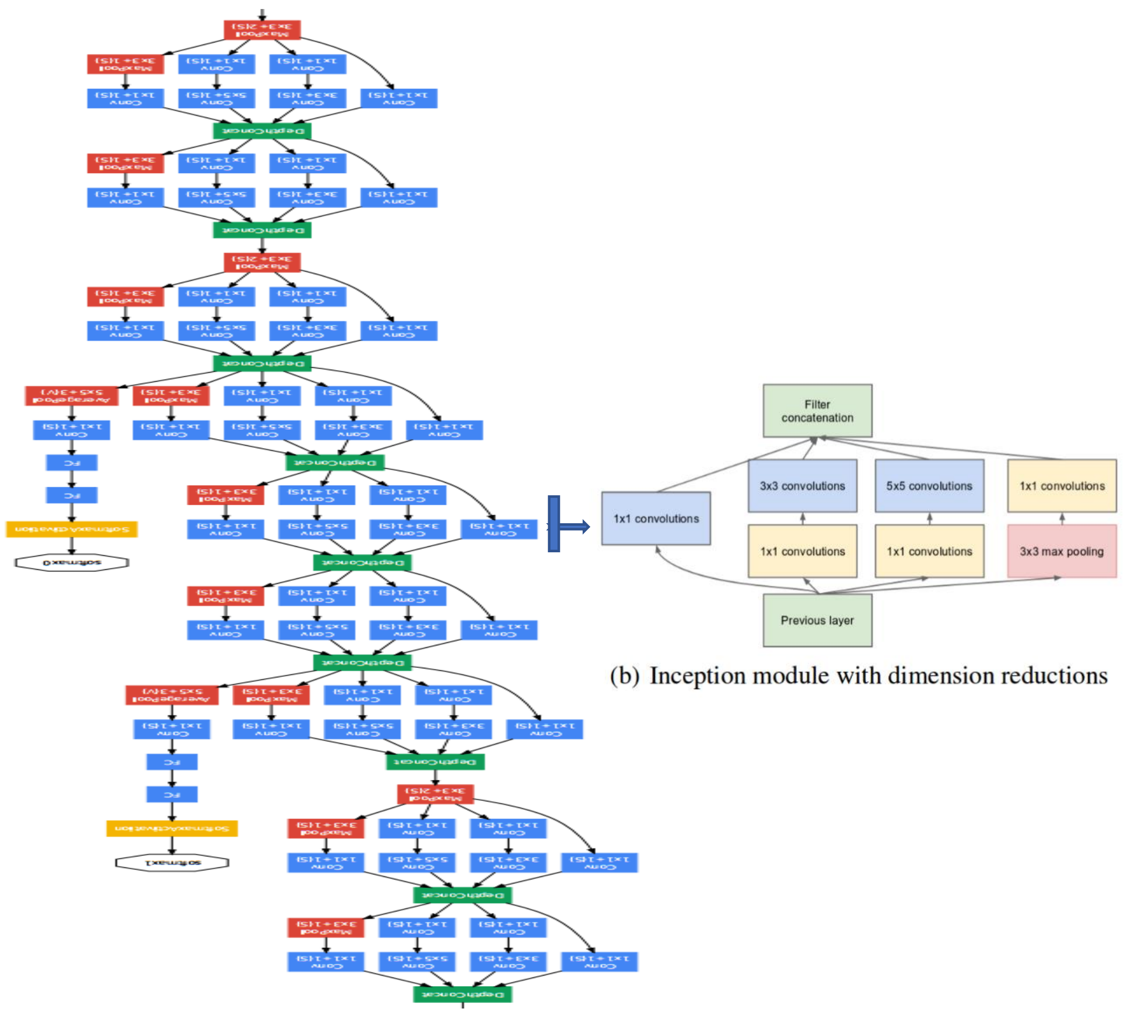


Figure 16: InceptionNet block diagram

3.5 Proposed Model

Owing to the unsatisfactory results from the available DL models, we decided to develop our own model for the classification task. Our model consists of 17 layers, of which 14 ar e convolutional layers - divided into 6 blocks.

Architecture

• 14 {convolution + relu} modules

• 5 {Max pooling} modules

• 3 {fully connected + fully connected} modules

• 2 {dropout} module

• 1 {softmax} module

Convolutions operate on 3x3 windows and max pooling layers operate on 2x2 windows.

First convolution block extracts 32 filters, second convolution block extracts 64 filters, third convolution block extracts 128 filters, fourth convolution block extracts 256 filters, fifth convolution block extracts 384 filters and the last block extracts 512 filters.

The model is built using the following layers:

• Conv2D - 2D Convolutional Network

• Activation - Relu

• Batch Normalization

• Max pooling

• Dense - Fully Connected Neural Network

• Softmax Layer as output

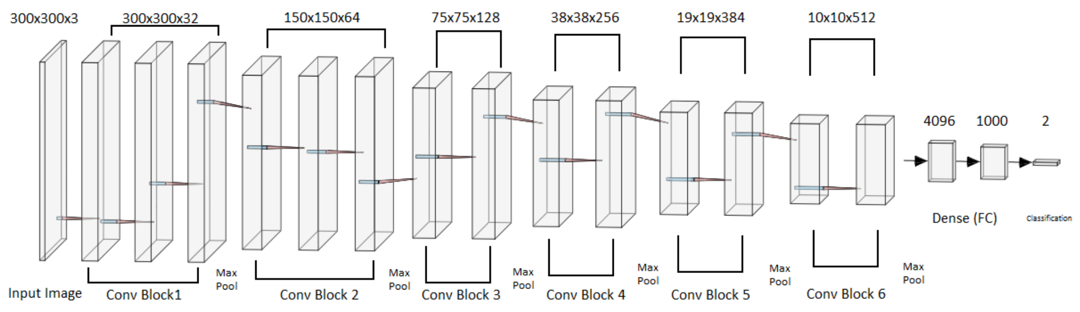
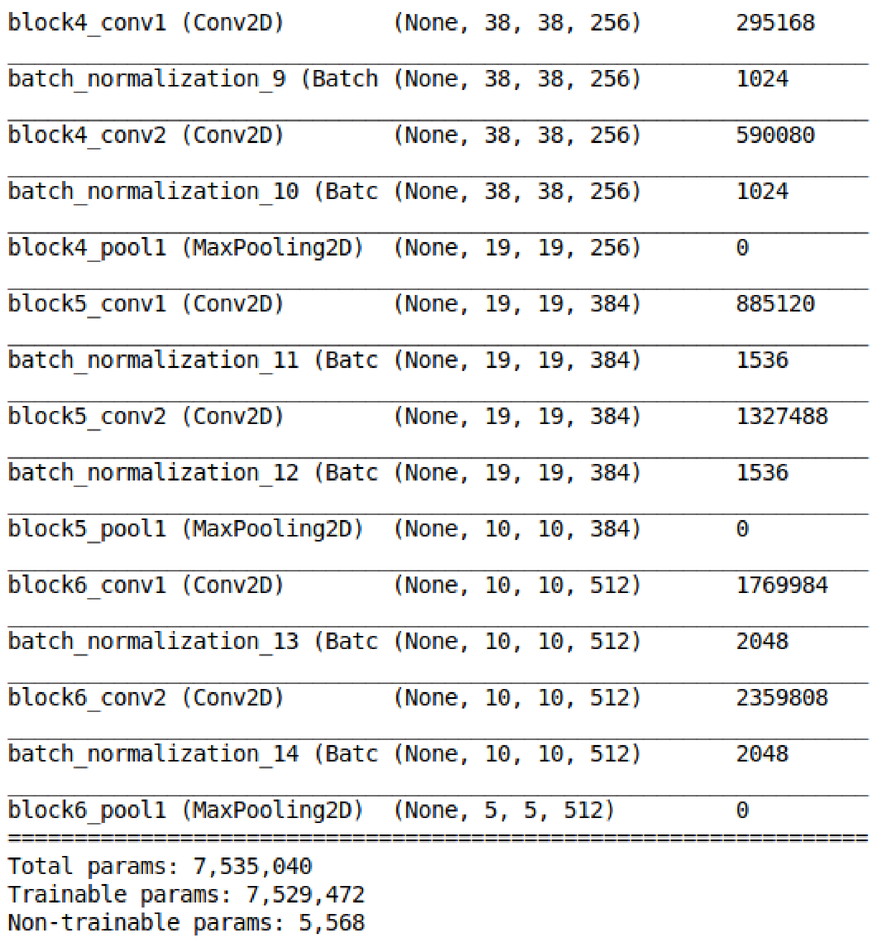
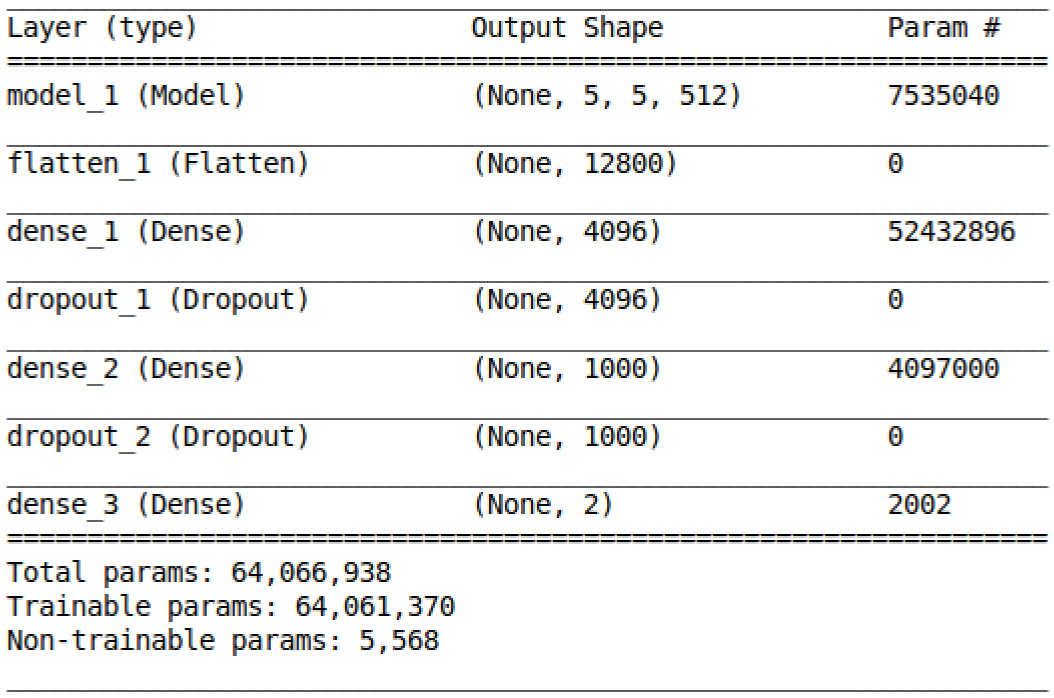


Figure 17: Proposed Architecture



```
block4_conv1 (Conv2D)        (None, 38, 38, 256)       295168
_________________________________________________________________
batch_normalization_9 (Batch (None, 38, 38, 256)       1024
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 38, 38, 256)       590080
_________________________________________________________________
batch_normalization_10 (Batc (None, 38, 38, 256)       1024
_________________________________________________________________
block4_pool1 (MaxPooling2D)  (None, 19, 19, 256)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 19, 19, 384)       885120
_________________________________________________________________
batch_normalization_11 (Batc (None, 19, 19, 384)       1536
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 19, 19, 384)       1327488
_________________________________________________________________
batch_normalization_12 (Batc (None, 19, 19, 384)       1536
_________________________________________________________________
block5_pool1 (MaxPooling2D)  (None, 10, 10, 384)       0
_________________________________________________________________
block6_conv1 (Conv2D)        (None, 10, 10, 512)       1769984
_________________________________________________________________
batch_normalization_13 (Batc (None, 10, 10, 512)       2048
_________________________________________________________________
block6_conv2 (Conv2D)        (None, 10, 10, 512)       2359808
_________________________________________________________________
batch_normalization_14 (Batc (None, 10, 10, 512)       2048
_________________________________________________________________
block6_pool1 (MaxPooling2D)  (None, 5, 5, 512)         0
=================================================================
Total params: 7,535,040
Trainable params: 7,529,472
Non-trainable params: 5,568
```

Figure 18: Proposed Convolutional Model summary



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
model_1 (Model)              (None, 5, 5, 512)         7535040   
_________________________________________________________________
flatten_1 (Flatten)          (None, 12800)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              52432896  
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000   
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2002      
=================================================================
Total params: 64,066,938
Trainable params: 64,061,370
Non-trainable params: 5,568
_________________________________________________________________
```

Figure 19: Complete Model summary

3.5.1 Key Issue in Building ML Models - Overfitting:

Overfitting happens when a model is exposed to too few examples, it learns patterns that do not generalize to new data, i.e. when the model starts using irrelevant features for making predictions. Overfitting is the central problem in machine learning: given that we are fitting the parameters of our model to a given dataset, how can we make sure that the representations learned by the model will be applicable to data never seen before? How do we avoid learning things that are specific to the training data?

3.5.2 Regularizers

Regularization is one of the techniques to overcome over-fitting. It applies to objective functions in ill-posed optimization problems [32]. There are two main types of regularizers L1 and L2. These update the general cost function by adding another term known as the regularization term [33].

Cost function = Loss (say, binary cross entropy) + Regularization term

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent. However, this regularization term differs in L1 and L2.

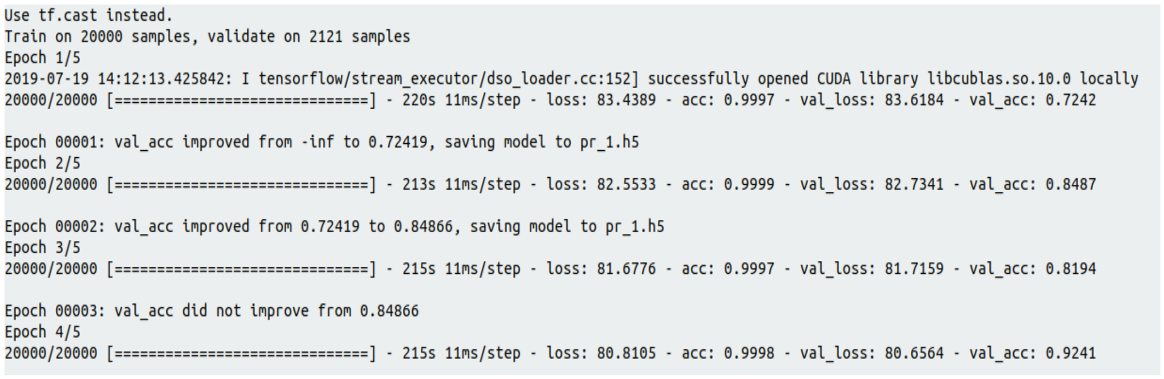
$$L1 = Loss + \frac{\lambda}{2N} * \sum ||W||$$

$$L2 = Loss + \frac{\lambda}{2N} * \sum ||W||^2$$

3.6 Network Training

The model is compiled with training parameters set. The Activation function used for hidden layers is Non-Linear activation function "Relu". The activation function used for the output layer is Probabilistic activation function "Softmax".

During training, SGD optimizer performed the best at a learning rate of 0. 0002.The model ran for 100 epochs with the batch size of 30.

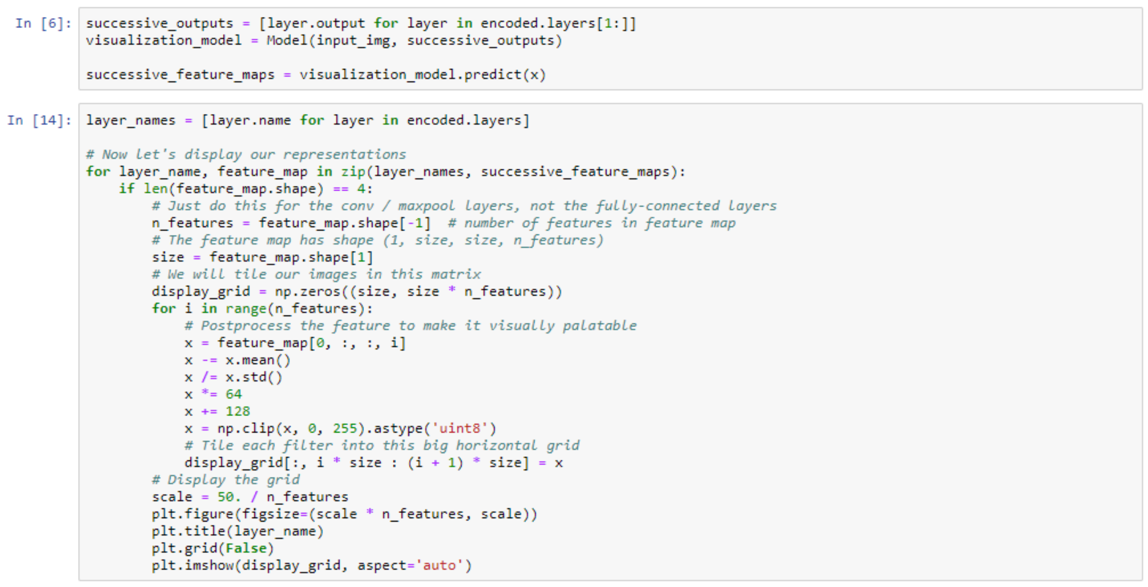


```
Use tf.cast instead.
Train on 20000 samples, validate on 2121 samples
Epoch 1/5
2019-07-19 14:12:13.425842: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.so.10.0 locally
20000/20000 [==============================] - 220s 11ms/step - loss: 83.4389 - acc: 0.9997 - val_loss: 83.6184 - val_acc: 0.7242

Epoch 00001: val_acc improved from -inf to 0.72419, saving model to pr_1.h5
Epoch 2/5
20000/20000 [==============================] - 213s 11ms/step - loss: 82.5533 - acc: 0.9999 - val_loss: 82.7341 - val_acc: 0.8487

Epoch 00002: val_acc improved from 0.72419 to 0.84866, saving model to pr_1.h5
Epoch 3/5
20000/20000 [==============================] - 215s 11ms/step - loss: 81.6776 - acc: 0.9997 - val_loss: 81.7159 - val_acc: 0.8194

Epoch 00003: val_acc did not improve from 0.84866
Epoch 4/5
20000/20000 [==============================] - 215s 11ms/step - loss: 80.8105 - acc: 0.9998 - val_loss: 80.6564 - val_acc: 0.9241
```

Figure 20: Network Training

3.6.1 Visualize Network Layers

To see what kind of features the convnet has learned, we visualized how an input gets transformed as it goes through the convnet. The images go from the raw pixels to increasingly abstract and compact representations. The representations downstream start highlighting what the network pays attention to, and they show fewer and fewer features being "activated"; most are set to zero.



```
In [6]:  successive_outputs = [layer.output for layer in encoded.layers[1:]]
         visualization_model = Model(input_img, successive_outputs)

         successive_feature_maps = visualization_model.predict(x)

In [14]: layer_names = [layer.name for layer in encoded.layers]

         # Now let's display our representations
         for layer_name, feature_map in zip(layer_names, successive_feature_maps):
             if len(feature_map.shape) == 4:
                 # Just do this for the conv / maxpool layers, not the fully-connected layers
                 n_features = feature_map.shape[-1]  # number of features in feature map
                 # The feature map has shape (1, size, size, n_features)
                 size = feature_map.shape[1]
                 # We will tile our images in this matrix
                 display_grid = np.zeros((size, size * n_features))
                 for i in range(n_features):
                     # Postprocess the feature to make it visually palatable
                     x = feature_map[0, :, :, i]
                     x -= x.mean()
                     x /= x.std()
                     x *= 64
                     x += 128
                     x = np.clip(x, 0, 255).astype('uint8')
                     # Tile each filter into this big horizontal grid
                     display_grid[:, i * size : (i + 1) * size] = x
                 # Display the grid
                 scale = 50. / n_features
                 plt.figure(figsize=(scale * n_features, scale))
                 plt.title(layer_name)
                 plt.grid(False)
                 plt.imshow(display_grid, aspect='auto')
```

Figure 21: Visualization function

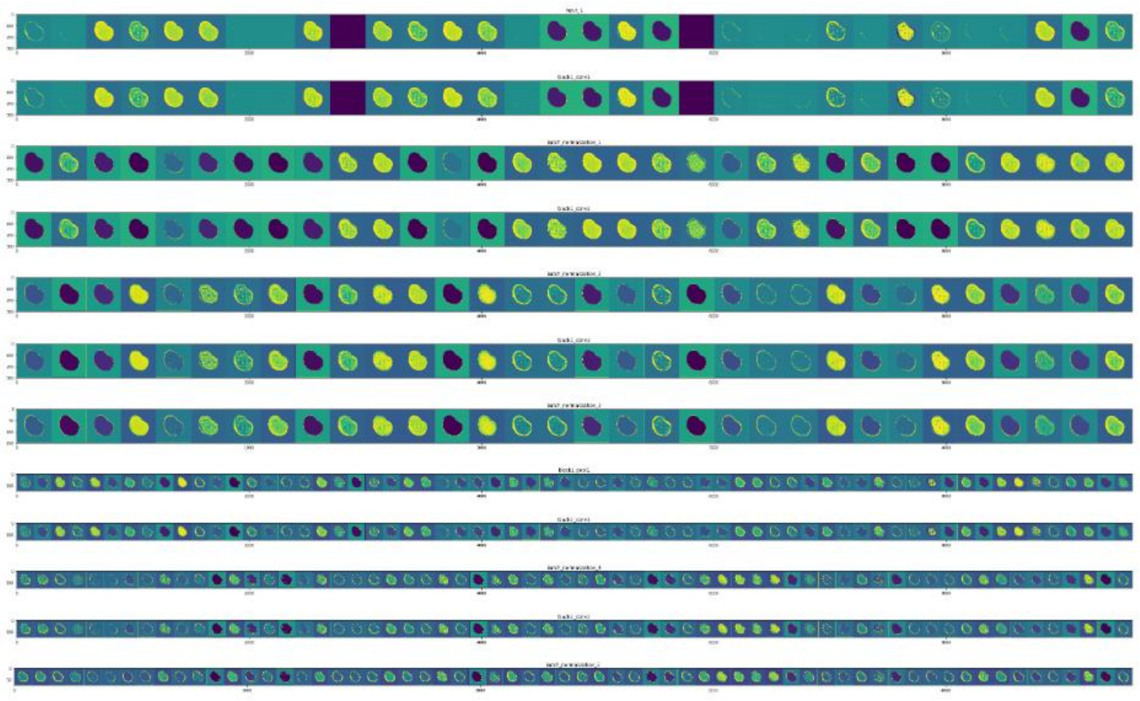


Figure 22: Visualizing network layers

4. Experimental Results and Comparison

4.1 Results

The performance metric selected for model evaluation was accuracy. There are four possible outcomes of a classifier: true positives (when cancerous cells are correctly classified), true negatives (when non-cancerous cells are correctly identified), false positives (when non-cancerous cells are classified as cancerous) and false negatives (when cancerous cells are classified as non-cancerous).

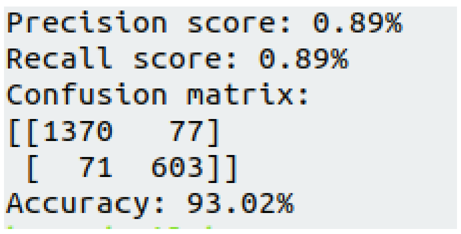
$$Accuracy = \frac{No.of\ correct\ Predictions}{Total\ no.of\ predictions}$$

Callbacks were chosen to monitor the validation accuracy and save the model at best possible parametric weights. Considering the simplicity of the model architecture, an accuracy rate of 93.02% was achieved.

Table 2 shows the performance comparison of the various different models and our model.

Architecture	Accuracy(%)
AlexNet	81.62
VGG_16	84.8
ResNet50	84.77
InceptionV3	83.67
Proposed Model	**93.02**

Table 2: Performance Comparison



```
Precision score: 0.89%
Recall score: 0.89%
Confusion matrix:
[[1370   77]
 [  71  603]]
Accuracy: 93.02%
```

Figure 23: Test Results

5. Conclusion, Summary and Future Scope

5.1 Conclusion

Our results conclude that a simple architecture can be devised which classifies our data with more precision and accuracy. The classical architectures fail mainly due to high invariance in the dataset and thus being incapable of effective feature selection and classification. The main advantage of this model is that it is computationally efficient compared to the much complex architectures. The model also continues to grow effectively as the epochs are increased.

We also found out that the of all the optimizers used, “SGD” gave the best classification result in all the configurations at a learning rate of 0.0002. “ReLU”, “Sigmoid” function was used as the activation functions for the convolution network layers and “Softmax” function was used as the activation function for the last Fully Connected (FC) layer for classification.

Future Work

The results can further be improved by combining various similar models in an ensemble technique. Also, auto-encoders can be applied for weight matrix initialization. Implementing the same model with greater number of epochs in a faster GPU system might also enhance the models functionality. Further, additional image processing techniques may also be explored for data preparation. Standardizing the image acquisition and pre-processing standards can, in effect, make the model universally applicable.

References

[1] "Cancer". World Health Organization. 12 September 2018. Retrieved 19 December 2018.

[2] "Defining Cancer". National Cancer Institute. 17 September 2007. Retrieved 28 March 2018.

[3] Vos, Theo, et al. "Global, regional, and national incidence, prevalence, and years lived with disability for 310 diseases and injuries, 1990–2015: a systematic analysis for the Global Burden of Disease Study 2015." The Lancet 388.10053 (2016): 1545-1602.

[4] Bernard, W. "Stewart and Christopher P. Wild.: World cancer report, ISBN 978-92-832-0429-9." (2014).

[5] Wang, Haidong, et al. "Global, regional, and national life expectancy, all-cause mortality, and cause-specific mortality for 249 causes of death, 1980–2015: a systematic analysis for the Global Burden of Disease Study 2015." The lancet388.10053 (2016): 1459-1544.

[6] "Childhood Acute Lymphoblastic Leukemia Treatment". National Cancer Institute. 8 December 2017. Retrieved 20 December 2017.

[7] Marino, Bradley S.; Fine, Katie S. (2013). Blueprints Pediatrics. Lippincott Williams & Wilkins. p. 205. ISBN 9781451116045

[8] “Key Statistics for Acute Lymphocytic Leukemia (ALL)” American Cancer Society.

[9] Ferri, Fred F. Ferri's Clinical Advisor 2016 E-Book: 5 Books in 1. Elsevier Health Sciences, 2015.

[10] LeCun, Y., Y. Bengio, and G. Hinton. "Deep learning. nature 521." (2015).

[11] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[12] Vogado, Luis HS, et al. "Unsupervised leukemia cells segmentation based on multi-space color channels." 2016 IEEE International Symposium on Multimedia (ISM). IEEE, 2016.

[13] Laosai, Jakkrich, and Kosin Chamnongthai. "Classification of acute leukemia using CD markers." 2016 8th International Conference on Knowledge and Smart Technology (KST). IEEE, 2016.

[14] Laosai, Jakkrich, and Kosin Chamnongthai. "Acute leukemia classification by using SVM and K-Means clustering." 2014 International Electrical Engineering Congress (iEECON). IEEE, 2014.

[15] Fatma, Mashiat, and Jaya Sharma. "Identification and classification of acute leukemia using neural network." 2014 International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom). IEEE, 2014.

[16] Asadi, Faisal, et al. "Implementation of Backpropagation Neural Network and Blood Cells Imagery Extraction for Acute Leukemia Classification." 2017 5th International Conference on Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME). IEEE, 2017.

[17] Sahlol, Ahmed T., et al. "Elephant herd optimization with neural networks: a case study on acute lymphoblastic leukemia diagnosis." 2017 12th International Conference on Computer Engineering and Systems (ICCES). IEEE, 2017.

[18] "Tensorflow — tensorflow," https://www.tensorflow.org/.

[19] "Home - keras documentation," https://keras.io/.

[20] "Pytorch," https://pytorch.org/.

[21] "scikit-learn: machine learning in python — scikit-learn 0.21.2 documentation," https://scikit-learn.org/stable/index.html.

[22] "Numpy — numpy," https://numpy.org/.

[23] "Python data analysis library — pandas: Python data analysis library," https://pandas.pydata.org/.

[24] "Matplotlib: Python plotting — matplotlib 3.1.1 documentation," https://matplotlib.org/.

[25] "Github - aszenz/kashir-cheez-classifier: Simple four category image classifier for some local kashmiri objects," https://github.com/aszenz/kashir-cheez-classifier, (Accessed on 07/22/2019).

[26] "Github - mdbloice/augmentor: Image augmentation library in python for machine learning." https://github.com/mdbloice/Augmentor, (Accessed on 07/22/2019).

[27] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[28] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[29] Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. "Highway networks." arXiv preprint arXiv:1505.00387 (2015).

[30] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[31] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[32] Regularization (mathematics). (2019). En.wikipedia.org. Retrieved 20 July 2019.

[33] https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques Learning, D., & code), A. (2018). An Overview of Regularization Techniques in Deep Learning (with Python code). Analytics Vidhya. Retrieved 20 July 2019.