Prediction of Acute Lymphoid Leukemia using Privacy Preserving Neural Network

by

Ishfaque Qamar Khilji
15301113
Kamonashish Saha
15341004
Jushan Amin Shonon
19241042
Ragib Israq
19341032

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
December 2019

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Ishfaque Qamar Khilji 15301113	Kamonashish Saha 15341004
Jushan Amin Shonon 19241042	Ragib Israq 19341032

Approval

The thesis/project titled “Prediction of Acute Lymphoid Leukemia using PrivacyP-reserving Neural Network” submitted by

1. Ishfaque Qamar Khilji (15301113)
2. Kamonashish Saha (15341004)
3. Jushan Amin Shonon (19241042)
4. Ragib Israq (19341032)

Of Summer, 2015 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on December, 2019.

Examining Committee:

Supervisor:
(Member)

Muhammad Iqbal Hossain, PhD
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Mahbubul Alam Majumdar, PhD
Professor and Chairperson
Department of Computer Science and Engineering
BRAC University

Abstract

In today's world, machine learning has become a big factor. It not only needs to be helpful, but also accurate and precise prediction is required. Machine learning is now becoming a widely used mechanism and applying it in certain sensitive fields like medical and financial data has only made things easier, but it also brought some difficulty in data privacy and data security which will protect the complete implementation of cloud based machine learning for these aspects due to the law and ethical needs. In this project, to give proper solution, we have come up with the idea using concepts of CryptoNets and Neural Networks, where we will be able to convert the learned neural network with the encrypted data to Cryptonets and the data will be totally encrypted and this will prevent the chances of unencrypted data being available to everyone. In this method, the owner will send the encrypted data to the cloud first and will hold a private key which can be used to decrypt the data later on. The cloud will have no idea about the data there since it will be in encrypted form and any attempts to get data from the cloud will only give the encrypted form. However, applying neural network to the cloud will enable us to store the data and make predictions in encrypted form and also give back the encrypted data to the user. In this way, the cloud will have no idea about the actual data and after the prediction is made, it will give back the predicted data in the encrypted form. We were able to achieve an encrypted prediction of about 78% close to the validation accuracy amount we achieved when training our Neural Network model.

Keywords: Crypto-Nets, Neural Network

Dedication

We would like to dedicate this paper to the Almighty, our family and friends.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.
Secondly, to our co-advisor Muhammad Iqbal Hossain, PhD sir for his kind support and advice in our work. He helped us whenever we needed help.
And finally to our parents without their throughout sup-port it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

ALL	Acute Lymphoid Leukemia
BFV	Brakerski/Fan-Vercauteren
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
KNN	K Nearest Neighbor algorithm
MLP	Multi Layer Perceptron
MNIST	Modified National Institute of Standards and Technology database
NN	Neural Network
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RNN	Recurrent Neural Network
SEAL	Software-Optimized Encryption Algorithm
SVM	Support Vector Machine
SWHE	Somewhat Homomorphic Encryption
YASHE	Yet Another Somewhat Homomorphic Encryption

Chapter 1

Introduction

We are trying to make a system where there will be an assurance about privacy and will also give an initial prediction i.e. whether the patient has cancer or not. This will also decrease the cost of the system because the initial tests are expensive and in our model the price will be less to give an initial prediction. The data will be stored in the cloud and hence the cloud can charge money for the storage and will also be financially beneficial for both the user and the supplier. This system can be used in case of banks, hospitals and other systems.
In our model we included Homomorphic encryption. This system can be used in hospitals, research institutes and others. In this system, it will allow one party to have a public key such as in hospitals where a lot of patients can upload their data through the public key which will be encrypted and stored in the commercial cloud. The owner, in our case the hospital administration, lab techniques, doctors and patients can have policies to decrypt the data when necessary. This will ensure the encryption and decryption in a proper manner and will also ensure proper privacy of the user is they want to store or export their information.
In the encryption process, the owner will only have the private key and will be capable for decrypting the data, on the other hand, the cloud system does not have any key and hence won't decrypt the data and hence it won't know about the data inside or be able to get any data about the predicted data. This will provide a better privacy and will also decrease the overall cost and since there is only one private key, the service provider will not also know about the data.
In our project, the unencrypted data will be first used for training the network. The training data sets can be difficult to find for these type pf projects because they always have a privacy issue and also that its not easily available. The problem of such type is also called Privacy Preserving Data Mining (Ahrawal Srikant, 2000). To come to internal concept of our project, we used CryptoNets where we use Homomorphic Encryption and Neural Network.

1.1 Motivation

In our research regarding the topics, we looked up for many different topics in different sectors. We wanted to work on something that will help our society and our country in the medical sector. Since, cancer detection is quite expensive now and many cannot afford it or is not much available to people, we wanted to find a detection system that would help detection in an easy manner and take less time and also give effective results. We also wanted to make it different and not much research was done on such topic. And since, privacy is a growing issue and not much taken care of in our country, we wanted to make our system secured and ensure the privacy.

1.2 Problem Statement

Many people in mainly the poor or developing countries fail to do the test. It is mainly because it is either very expensive or it is not much available to everyone. In addition to that, there is not much security to the data and does not ensure privacy. This is because anyone can access the diagnosis reports of the patient in the hospitals in many parts of the world. The initial tests to detecting blood Cancer are either expensive or are not much available. Moreover, usually the whole diagnosis process is really time consuming and patients have to wait more days for the diagnosis report to arrive.Due to being more expensive , the people in the developing countries with middle or poor income cannot afford it.Thus ,many people are left untreated and the death rate increases.Also, even if the treatment is available, the diagnosis reports of the patient remains available to anyone who wants to view it. Thus there is loss of privacy. More hospitals and medical agencies will use our model since it is cheap . Its cheap because our model requires only the dataset of the Cancer images needed to for training and the rest of the model is entirely computer based.Moreover, less resources are required compared to the original blood screening. It ensures patients medical records or diagnosis reports being secure as medical predictions or diagnosis is itself encrypted in our model . Hospitals or other medical agencies can provide the securely fast working model then to the mass people at a lower price ,thus many can afford it

1.3 Our proposal

We propose make a Privacy Preserving Neural Network model which can predict blood Cancer as well as maintain the privacy of the patient. In our thesis, at first, we have taken a blood cancer dataset and successfully ran it on various Neural Network and Machine learning models which would accurately predict blood cancer. The results are then compared among them. Moreover, then we made our own Neural Network model which is run on the dataset that we are having which is modified at first in order to run on encryption application .The results are again compared with that of the previous models to prove our NN model is better than the others here. The model is then encrypted to give predictions in secured format. We are on the process of having our own dataset collected from different labs which we kept for future work. We want to provide a system that will not only give the initial result of whether it is cancer or non-cancer but will also will be encrypted and the result will only be known by the patient with the private key which will ensure the privacy.

1.4 Objective and contribution

Our objectives include Detection of Blood Cancer (Leukemia) from imagery test samples after proper modification in order to run on custom the CryptoNets application. A Homomorphic encryption scheme on the whole system which would be used to homomorphically encrypt the images from the Neural Network on which computations and predictions can be done even if the images are encrypted Comparative analysis is done among the first several models run and then between them and our own NN model.The results are then compared. Work on CryptoNets is done currently in mainly 3 datasets : MNIST,CIFAR-10 and Caltech-101. CryptoNets has not been used in practical applications before. Thus our contribution in detecting blood Cancer using imagery in privacy preserving model(CryptoNets) will be the first of its kind.The process that we introduce will pave a way for implementations in various field.This will ensure secure lives and provide customer satisfaction.

1.5 Thesis Structure

Chapter 1: Introduction where motivation, problem statement, objectives and contributions were discussed.

Chapter 2: In literature review, we discussed the previous work and related works. We described about Homomorphic Encryption and how to select its parameters and the encryption scheme algorithms. We also discussed about Neural Network and how it works describing some of the algorithm that are related to our project.

Chapter 3: We described about our proposed model and provided a workflow diagram. There was a description data-set about how we pre-processed data and also performed feature selection. There was a brief description about our model and also about its detail.

Chapter 4: We discussed about the experiments we performed and analyzed the results.

Chapter 5: It includes conclusion and the plan about our future works.

Chapter 2

Literature Review

2.1 Literature review

Early work on automated image-based ALL diagnosis can be loosely grouped into more recent approaches that use convolution neural networks as feature extractors and older approaches that use handmade features.

2.1.1 CNN Features

Shafique and Tehsin [1] used pre-trained AlexNet and fine-tuning for classification of ALL subtypes on ALL-IDB augmented that includes 50 private images. Rehman et al [2] used an AlexNet which was pre trained and fine-tuned for classification of ALL subtypes,worked with a private datatset of 330 images. On the other hand, Vogado et al [3] used a different pre-trained CNNs as fixed feature extraxctors to classify ALL on ALL-IDB.

2.1.2 Handcrafted Features

Mohapatra et al. [4] and Madhloom et al. [5] use private datatset and classify using an ensemble of SVM, KNN, naïve Bayes and a KNN classifier. Putzu and Ruberto [6] classify a number of features such as, compactness area and ratio between cytoplasm and the nucleus with an SVM using ALL-IDB.
In the above case, the dataset used are small compared to others and also tough to compare the results. The datasets which are private are not available and the ALL-IDB datasets which are public are given on their own evaluation procedures. All these factors make comparisons difficult.

Our project is divided into two parts of the programming languages Python and C. The Neural network model building and comparisons of the ML and NN models are done in the python part of the project. The encryption part after that where the ‘CryptoNets’ application is created is done on C sharp.

Grapel et al. [7] shows an implementation of homomorphic encryption of machine learning algorithms where they concentrated on pointing out the algorithms showing encrypted data can be trained and thus were more opted to use learning algorithm where the training algorithm can be shown in a low degree polynomial. Zhan et al.,

[8]; Qi & Atallah, [9] looked up for nearest neighbor divisions but they do not give the same level of accuracy as neural networks. Aslett et al. [10] [11] presented both of the algorithms such as naïve Bayes classifiers and random forests but their model cannot work efficiently in recognizing objects in images.

2.1.3 Updates to SEAL

The SEAL library has been changed and improved and it no longer uses the YASHE scheme and instead uses the Fan-Vercauteren scheme (FV) [12]. The YASHE schemes which was used previously relied on the unusual hardness assumption for security [13] [14] which turned to be easier to break, but on contrast, the recent FV scheme, relies on Ring-Learning with Errors (RLWE) assumptions. The FV when compared with YASHE, has better noise growth properties specifically in homomorphic multiplication and hence many parameters will run with better improvements and improved performance. In addition to that, some new features have been added and a few API changed have been introduced.

2.1.4 Updates to Cryptonets

It was shown in [15] that high throughput can be gained when taken to examine deep convolutional neural networks on encrypted data. That paper gave an accuracy of approximately 99% on detection of hand written digits (MNIST dataset) using neural network and an output of over 50,000 predictions per hour.

2.2 Homomorphic Encryption

Homomorphic encryption is a technique used for encrypting information , such that the information can be evaluated or calculated on by anyone with no access to keys needed for encryption or decryption, and the calculation results are obtained in encrypted form. Homomorphic encryption solutions, that require single operations , such as addition, are there for decades, such as for the ones that has its foundation on the RSA or Elgamal cryptosystems.A homomorphic encryption method that allows an infinite number of simultaneous operations, i.e. both addition and multiplication, allows the calculation of any circuit and thus a complete method of homomorphic (FHE) gained. A leveled homomorphic encryption scheme is an encryption algorithm which makes it possible in selection of parameters that allows homomorphic testing of any given fixed function. Implementing a leveled homomorphic scheme with fixed parameters to enable a definite preset, fixed measure of calculations together with application-specific data encoding and algorithmic optimization points to a certain amount of efficiency and collectively known as Practical Homomorphic Encryption (PHE).
FHE was first presented in Gentry in 2009 [16]. In Gentry, the data encrypted in the bits and for each bit in the message, a separate Ciphertext is produced. It is a sort of addition and multiplication module represented by Boolean circuits with XOR and AND gates.
FHE in ciphertexts contain some inherent noise which grows during homomorphic

encryption and it cannot be decrypted when it gets too large. To solve this problem, Bootstrapping is used where the ciphertexts are constantly refreshed and their noise is reduced [17] [18].
Converting these ideas into practical systems to solve performance and storage challenges are still a serious challenge. However, there are some noticeable improvements which is done by encoding the data in a way which will deduct both the size of Ciphertexts and also test the depth of the circuits. The parameters for Practical Homomorphic Encryption (PHE) should be chosen which would not only increase the efficiency but also preserve privacy and ensure security. In our project, we have used certain tools such as Noise Growth Simulator and Automatic Parameter Selection Module to help the user to achieve maximum performance [19].
Two types of homomorphic encryption includes: Partially and Fully Homomorphic encryption.
Partial Homomorphic encryption is a system in cryptography that is considered partially homomorphic if it demonstrates either additive or multiplicative homomorphic characteristics, not both. Clearly, partial Homomorphic encryption schemes are useful in certain applications.
Somewhat homomorphic encryption approaches can only evaluate a multiple but limited number of addition and multiplication activities. SWHE schemes refer to encryption systems that present certain homomorphic characteristics but lacks full homomorphic capacity.
The fully homomorphic encryption supports a significant measure of multiplications and additions, hence, can calculate functions of any form on encrypted information. For all forms of computations on the information warehoused in cloud, the usage of FHE is thus an important step in developing cloud-computing security.

Encoding

As described previously, a discrepancy exists between the atomic structures in neural networks (real numbers) and the atomic structures in the homomorphic encryption schemes (polynomials in R_t^n) [20].
Each other is mapped by an encoding scheme in a manner that protects the operations of addition and multiplication. Such a scheme of encoding can be constructed in several ways. For example, real numbers can be converted to set precision numbers, and then their binary version can be used to turn them into a polynomial having the binary expansion coefficients. This polynomial will have the property of returning the encoded value when evaluated at 2. Another alternative is to encode as a constant polynomial the fixed number of precision. This encoding is straight forward, but taking into account that only one polynomial coefficient is being applied may seem inefficient. One problem with the scalar encoding is that when homomorphic operations are performed, only the coefficient of the message polynomials grows very quickly.

Encoding Large Numbers

As we have already explained, in this encryption scheme, a major challenge for computation is to prevent the coefficients of the plaintext polynomials from overflowing t [20]. This forces us to pick large values for t, which allows the noise to grow faster in the ciphertexts and reduces the total amount of noise tolerated (with q fixed). Therefore, for security reasons, we need to choose a larger q, and then a larger n. One way to overcome this problem partially is to use the Chinese Remainder Theorem (CRT) (see the supporting material). The concept of using multiple primes is $t_1, ..., t_k$. Given a polynomial $\sum a_i x^i$ we can convert it to k polynomials in such a way that the j-th polynomial is $\sum [a_i (mod t_j)] x^i$. Each polynomial like this is encrypted and controlled in the same way. It is the confirmation of CRT that would enable us to decode back the output, till its coefficient does not grow beyond $\prod t_j$ .Thus, this method lets us to encode exponentially large numbers at the same time incriminating time and space linearly in the number of primes operated.

Plain-text space and homomorphic operations

Plaintext elements(messages encrypted by homomorphic encryption schemes) can be represented as a polynomial ring,R with coefficients minimalized modulo the integer,t. Cipher text elements(encrypted plaintext elements) on the other hand can be similarly represented but instead has coefficients minimalized modulo the integer ,q [19]. Thus , the plaintext space for the ring. Formally, this means that the plain-text space is the ring Rt := R/tR = Zt[X]/(Xn + 1), and the ciphertext space is contained in the ring Rq := R/qR = Zq[X]/(Xn + 1). However, some of elements in Rq are invalid ciphertext. A ciphertext created by the function used for encryption in the scheme that we are using encrypts one plaintext message polynomial m in Rt. If a homomorphic addition(resp. multiplication) is done on ciphertext that encrypts two plaintext messages for example m1,m2 in Rt , the output ciphertext will encrypt the summation of m1+m2(resp.the product m1.m2). Plaintext element computations are done in the ring Rt.Thus,incase of homomorphic addition, the output ciphertext will encrypt the coefficient wise summation m1+m2 , where the coefficients are likewise reduced modulo the plaintext modulus ,t. Incase of homomorphic multiplication, the output ciphertext will encrypt the product m1.m2 in Rt, meaning the polynomial will likewise be reduced modulo Xn+1 where −1 will substitute all powers of Xn and continued till no monomials of n degree or higher than that is remaining. Just like homomorphic addition, the coefficients of polynomial m1.m2 will likewise be deducted modulo integer, t.

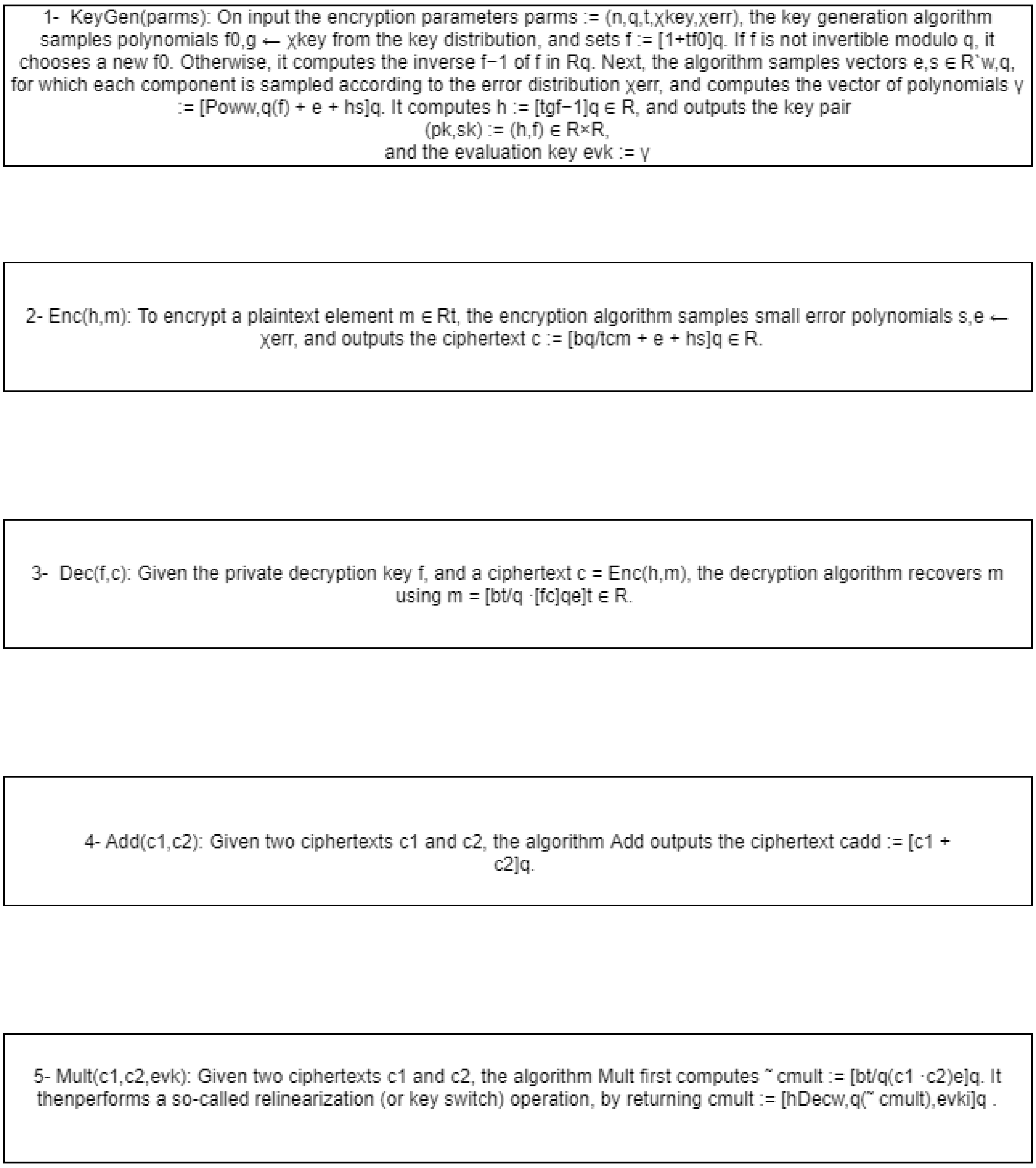
2.3 Selecting encryption parameters

The particular scheme that is used in SEAL is the more practical derivation of the YASHE scheme.Encryption parameters of the scheme are : degree n, the moduli q and t, the decomposition word size w, and distributions Xkey,Xerr. Thus, parameters := (n,q,t,w,Xkey,Xerr). These parameters are explained in more detail below

- n here is used as the maximum number of terms in the polynomials used for showing the plaintext as well as ciphertext elements. SEAL shows n always as a power of 2. Xn + 1 polynomial is the polynomial modulus, shown as polymodulus in SEAL.
- q the coefficient modulus,is an integer modulus operated in reduction the coefficients of ciphertext polynomials. SEAL represents q as coeffmodulus.
- t ,the plaintext modulus,is an integer modulus taken in reductionof the coefficients of plaintext polynomials.SEAL shows t as plain modulus..
- Integer coefficients is decomposed into smaller parts according to the integer base w. The integer calculates the number $_{w,q} := blog_w(q)c + 1$ of parts when decomposing an integer modulo q to the base w. Practically, we take w as a power of two, and take the decomposition bit count as $log_2 w$ .SEAL, shows $log_2 w$ as decomposition bit count.
- Xkey distribution is a probability distribution on polynomials of degree at most n-1 with integer coefficients implemented to sample polynomials with small coefficients that are taken in the key generation procedure. In SEAL, coefficients are sampled uniformly from [1,0,1].
- Likewise, the distribution Xerr on polynomials of degree at most n-1 is used for sampling noise polynomials, essential in time of both key generation and encryption. SEAL has the distribution Xerr as a shortened discontinuous Gaussian centered at zero having standard deviation . SEAL has it called Noise Standard Deviation.

2.4 Homomorphic encryption scheme algorithms

The following chart gives a detailed representation of the key generation, encryption, decryption and homomorphic evaluation algorithms [19]:



1- KeyGen(parms): On input the encryption parameters parms := (n,q,t,χkey,χerr), the key generation algorithm samples polynomials f0,g ← χkey from the key distribution, and sets f := [1+tf0]q. If f is not invertible modulo q, it chooses a new f0. Otherwise, it computes the inverse f−1 of f in Rq. Next, the algorithm samples vectors e,s ∈ R`w,q, for which each component is sampled according to the error distribution χerr, and computes the vector of polynomials γ := [Poww,q(f) + e + hs]q. It computes h := [tgf−1]q ∈ R, and outputs the key pair (pk,sk) := (h,f) ∈ R×R, and the evaluation key evk := γ

2- Enc(h,m): To encrypt a plaintext element m ∈ Rt, the encryption algorithm samples small error polynomials s,e ← χerr, and outputs the ciphertext c := [bq/tcm + e + hs]q ∈ R.

3- Dec(f,c): Given the private decryption key f, and a ciphertext c = Enc(h,m), the decryption algorithm recovers m using m = [bt/q ·[fc]qe]t ∈ R.

4- Add(c1,c2): Given two ciphertexts c1 and c2, the algorithm Add outputs the ciphertext cadd := [c1 + c2]q.

5- Mult(c1,c2,evk): Given two ciphertexts c1 and c2, the algorithm Mult first computes ˜ cmult := [bt/q(c1 ·c2)e]q. It thenperforms a so-called relinearization (or key switch) operation, by returning cmult := [hDecw,q(˜ cmult),evki]q .

2.5 Neural Network

The term Neural Network is an artificial network which is composed of circuits or neurons or artificial nodes. These are leveled circuits and in layers and are usually found in an order where the last layer is the input layer and the first being the output layer. Each layer consists of nodes and they are all incorporated a value of the features of the project. In these layers, the above or previous nodes of the layer compute a function based on the nodes of the layers under it and the first node in the stack becomes the output layer.

2.6 Algorithms used

On pre trained CNN models as well as SVM (Support vector machine) model of our own. The CNN models that we used includes VGG16 and VGG19, AlexNet and ResNet. After running these models with the mentioned dataset, we compared the accuracies (both train and test accuracies). The whole process of running different neural networks mentioned is as follows:

2.6.1 VGG 16 and VGG19

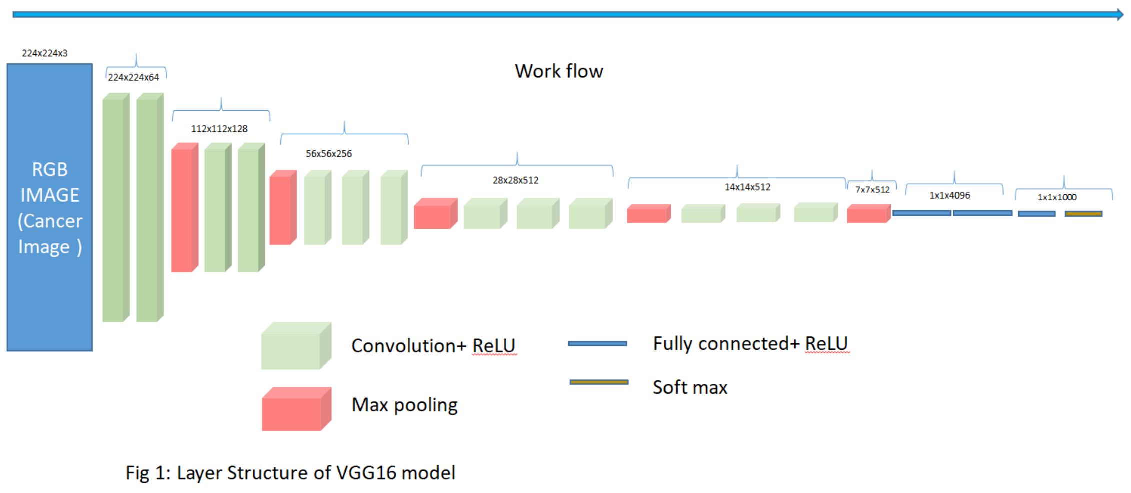


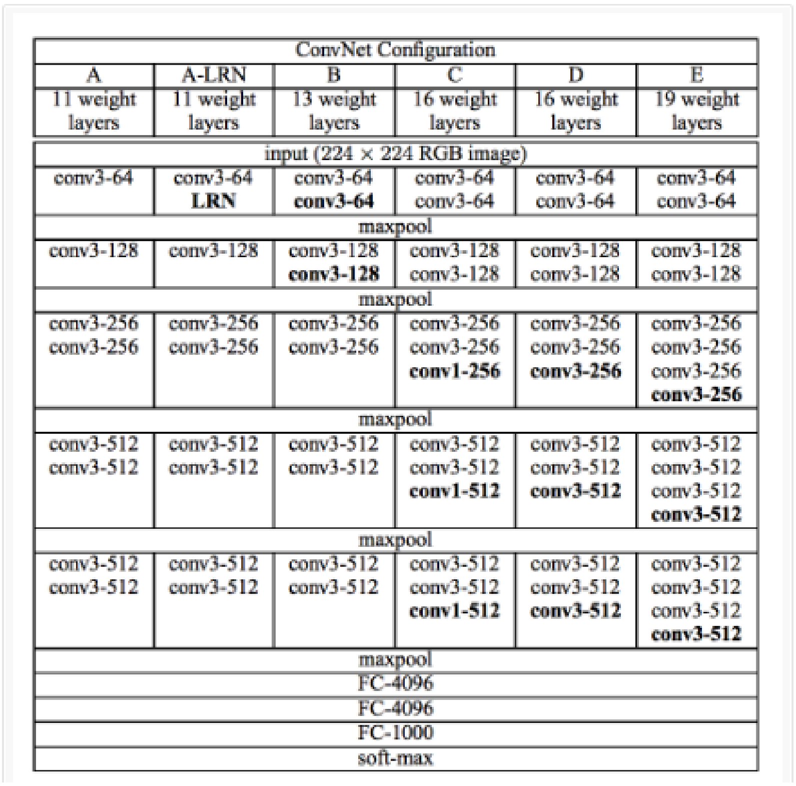
Figure 2.1: Layer Stucture of VGG16 model

From the diagram, the input images are getting filtered in each block of layers .Thus, as the image passes through the Convolutional and Max Pooling layers , their kernel and tensor shape of the input changes .For example in the first Convolutional and Max Pooling layers, the tensor changes to [224,224,64], then to [56,56,256] . In each block, the input image is beeing filtered and features are being extracted. The output layer being a MaxPooling layer [1,1,512].

Over here, we used pre trained VGG 16 and VGG 19 models, using image net Dataset. ImageNet is a project that is used for computer vision search and is where the images are manually labelled into almost 22,000 object categories where the images are used to train the model and later can be used to classify the input images into different categories.

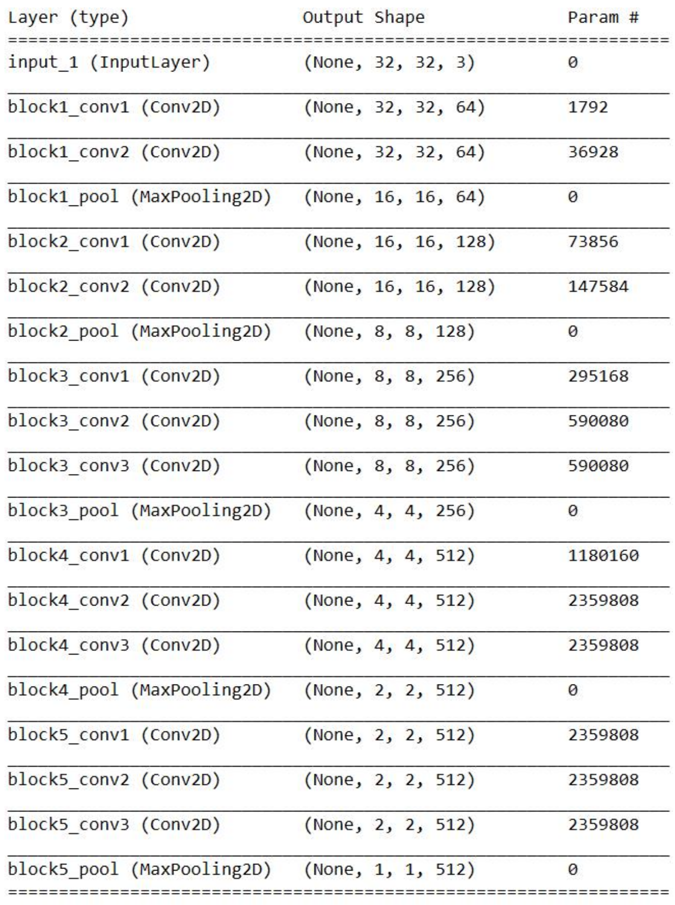
In VGG16 architecture, the images are passed through a sequence of convolutional layer which are of fixed size (224x224 RGB image). Thus, we use the default image size for this model in our dataset. In those layers, the filters that were used had a very short receptive field: 3 by 3 (that is the minimal size in capturing the concept of left/right, up/down, center). In one of the settings, it also uses 1 by 1 convolution filter, that may be considered as a linear change of the input channels. The convolution stride is set to one pixel; the spatial padding of conv. layer input is in a way that the spatial resolution is kept after convolution, i.e. the padding is one pixel for 3 by 3 conv. layers. Spatial pooling is done by five max-pooling layers, which follow some of the convolutional layers . Max-pooling is performed over a two by two pixel window, with stride of two.
There are three fully connected layers which have different dept in different architectures. Amongst them, the first two have 4096 channels, and the third performs 2-way Leukemia dataset classification and contain 2 channels for each individual class and the last layer is soft-max layer. This configuration is the same in all the networks.
The difference between VGG16 and VGG19 is that, VGG19 has 3 more convolutional layers than VGG16. The "16" and "19" shows the number of weight layers in the network (columns D and E in Figure 2.2 below):

Like we mentioned earlier we are using pre trained VGG16 and VGG19 models of ImageNet dataset. Thus, in building our own VGG16 and VGG 19 model we use the 'Weights' of ImageNet. We then extract features of our dataset that is used through VGG16 and VGG19 convolutional base. After the feature extraction the data then passes through the layers described above (VGG 16 and VGG 19). The models are then fitted and trained for 100 epochs. The results are shown in section 4. The layers the VGG16 model involving our dataset is mentioned below:



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.2: Difference in layers between VGG 16 AND 19



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
```

Figure 2.3: Layer shapes and Parameters of the layers after trained with our dataset

2.6.2 SVM

Supervised Vector Machine (SVM) is a supervised machine learning algorithm which divides the dataset into two classes and are mostly used for classification and regression purposes.

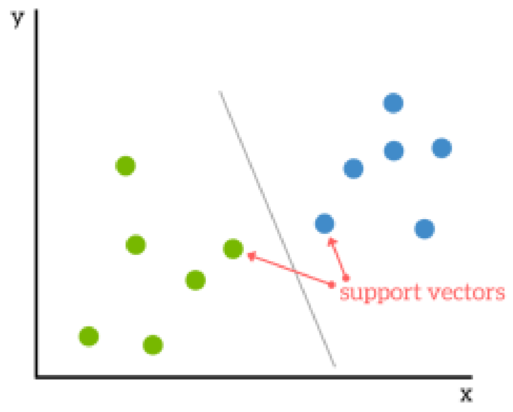


Figure 2.4: Support Vector Machine Seperated by a hyperplane

The support vectors are one of the important components which determine the position of the hyperplane because these are the points what are the closest to the hyperplane.
In order to train a linear support vector machine, the machine learning approach is used. We can use K-fold cross-validation where we can estimate error of our mode. Since this will be used, we can enlarge our training data by concatenating the train and the validation sets.
After the feature extraction using the convolutional base of VGG16 the output tensor [,2] is used in the model fitting if the SVM model. Thus, no separate feature extractions of the pre-processed images that are used are required. The model is run for 100 epochs. The result analysis is given later in this paper.
Lastly, we ensure that the SVM classifier has one hyperparameter which is a penalty parameter C of the error term.
A sample of the code for building the model using 'feature extracted output tensor' as input data is given below:

```
#



Build
model
     import sklearn
     from sklearn.cross_validation import train_test_split
     from sklearn.grid_search import GridSearchCV
     from sklearn.svm import LinearSVC

     X_train, y_train = svm_features.reshape(1*1*512), svm_labels

     param = [{
               "C": [0.01, 0.1, 1, 10, 100]
              }]
     svm = LinearSVC(penalty='l2', loss='squared_hinge')
     clf = GridSearchCV(svm, param, cv=10)
     clf.fit(X_train, y_train)
```

2.6.3 AlexNet

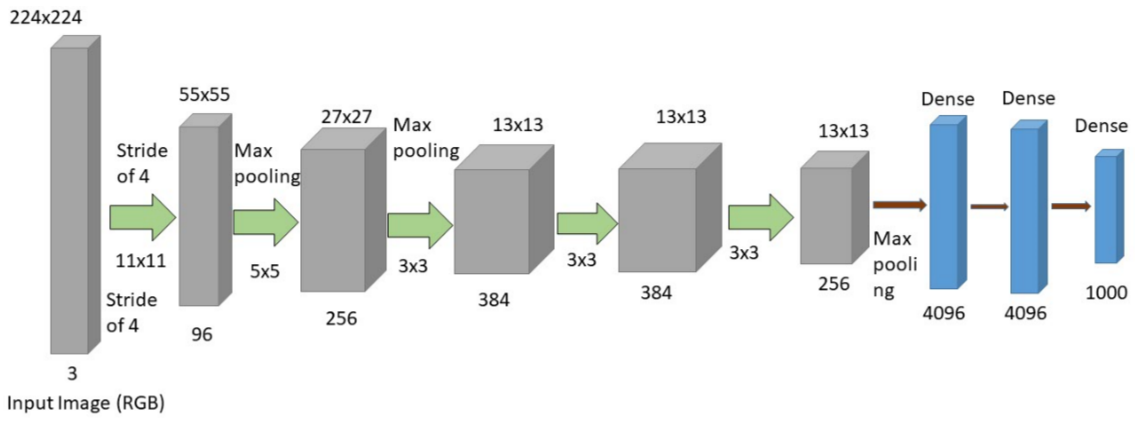


Figure 2.5: Sample AlexNet layer structure

The AlexNet model was first introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (ImageNet Classification with Deep Convolutional Neural Networks by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, 2012)
Classifying the image is a big issue and AlexNet solves it by taking the input image of one of 1000 different classes and gives an output of a vector of 1000 numbers in general use. Here, there are 2 classes instead of 1000 thus there will be an output vector of just 2. The sum of all the elements of the output vector is 1. AlexNet takes an input image of RGB image size 100x100 from the pre-processed dataset. All the images are in RGB, however, is the image is not in RGB or is in grayscale, it is converted to RGB by replicating the single channel to obtain 3 channel RGB image. AlexNet has 5 Convolutional Layers and 3 Fully Connected Layers

Multiple Convolutional Kernels
Multiple convolutional kernels are also called filters which extract the necessary features from an image where the single convolutional layers consist of more then one kernels of the similar size.

The first two convolutional layers
The third, fourth and fifth convolutional layers are linked directly. An Overlapping Max Pooling layer follows the fifth convolutional layer, whose output goes into a series of two fully integrated layers. The second fully integrated layer feeds into 2 class labelled SoftMax heuristic.

Max Pooling Layer
The depth is kept unchanged by down sampling the height and with of the samples. Overlapping Max Pool layers are comparable to Max Pool layers, other than for the neighboring windows over which the max gets calculated to overlap. The model's authors used to pool 3x3 size windows with 2 step between adjacent windows. This overlapping nature of pooling helped lower the top-1 error rate by 0.4 percent, top-5 error rate by 0.3 percent, compared to using 2x2 sized non-overlapping pooling windows with a step of 2 giving similar output dimensions.

An important feature of AlexNet is the use of the non-linearity function in the layers. Activation functions of sigmoid or Tanh functions used to be the standard way to train a neural network model. AlexNet showed that deep CNNs could be trained much faster using the non-linearity function of ReLU other than using saturated activation functions such as tanh or sigmoid.

A network is dropped from the network with a probability of 0.5 in the dropout layer. It doesn't lead to either forward or backward propagation when a neuron is dropped. Each input passes through different architectural design of the network. The parameters of the learned weight are more rigorous and are not easily overfitted. There is no dropout during testing and the entire network is used, but output is scaled by a factor of 0.5 to account for the neurons that have been missed during training. Dropout raises the amount of simulations required to converge by a factor of 2, but AlexNet will be significantly over-fitting despite dropout.

The pre-processed images from our dataset are given as inputs through the first layer of the AlexNet model. The feature extraction part is taken care of in the convolutional layers as the input images are segmented and pixelated in the layers with each pixel having particular weights and bias. So, each layer of the CNN model has different values of weights and biases. After the model is fitted with the data and the layers and the functions are defined, the model is put to training and the validation accuracy of the model is recorded.

2.6.4 ResNet50

Residual Networks or ResNet creates network though models known as residual models known as the degradation problem. Although increasing depth increases the accuracy of the network, but the problem increases when vanishing gradient arises. Another problem which arises with training the deeper network is greater training error as when performing optimization on huge parameter space adds the layers to a greater training error. The ResNet architecture is similar to VGGNet which has 3x3 filters. An image is given to give a brief description:

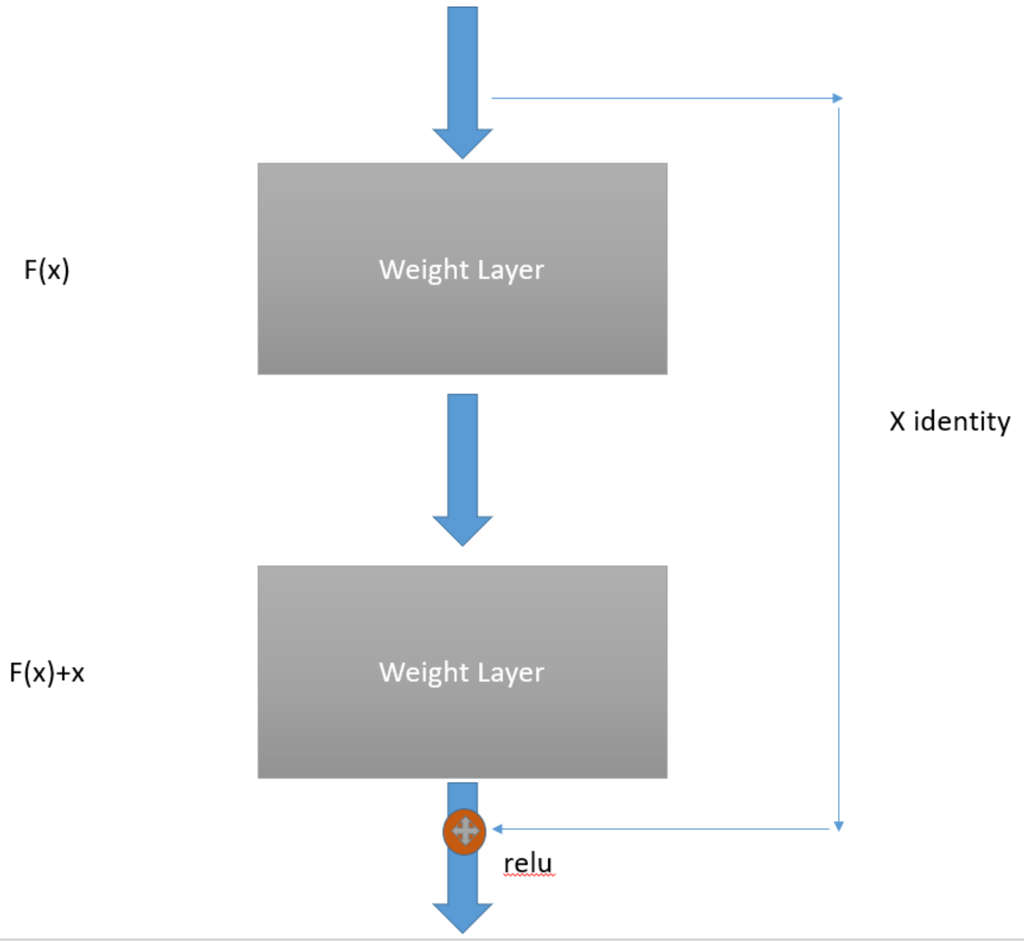


Figure 2.6: Residual Learning : a building block

The ResNet50 model that we will be using is a pre-trained model trained on ImageNet dataset. Below is the architecture of a pre-trained ResNet50 model. The model has '50' weight layers.

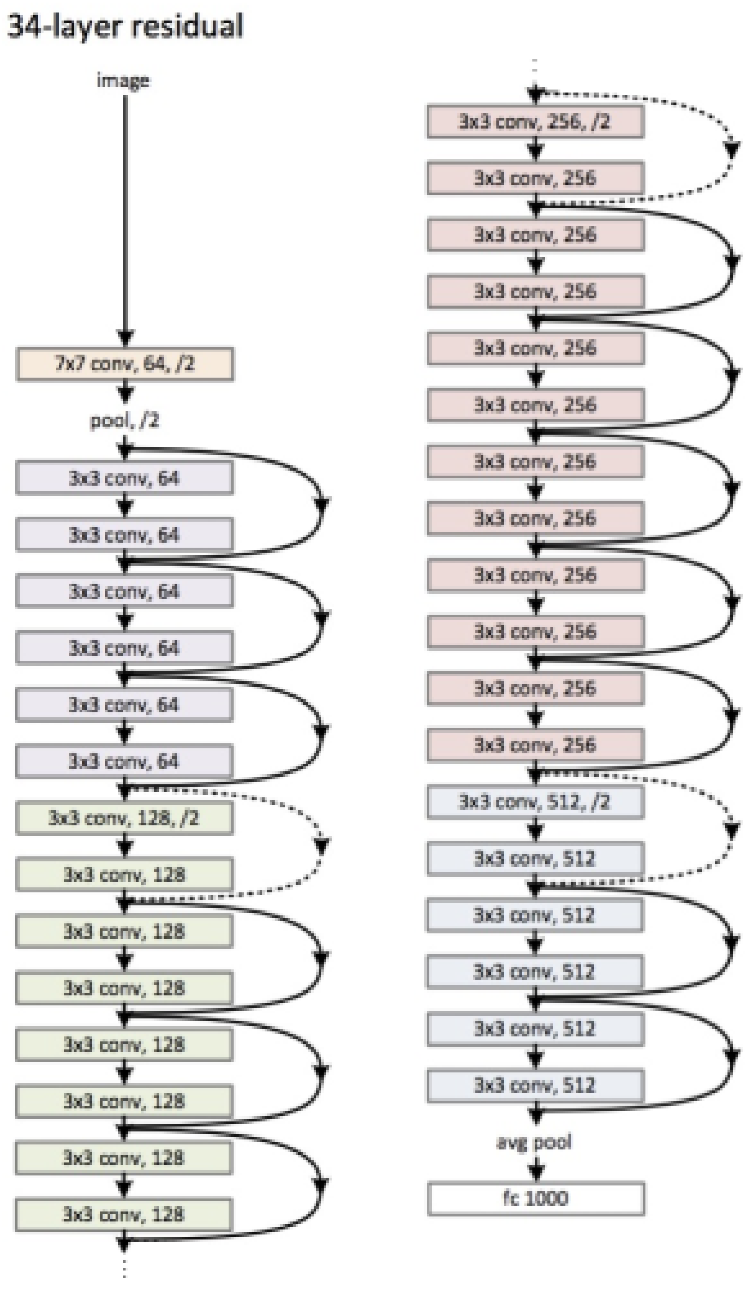
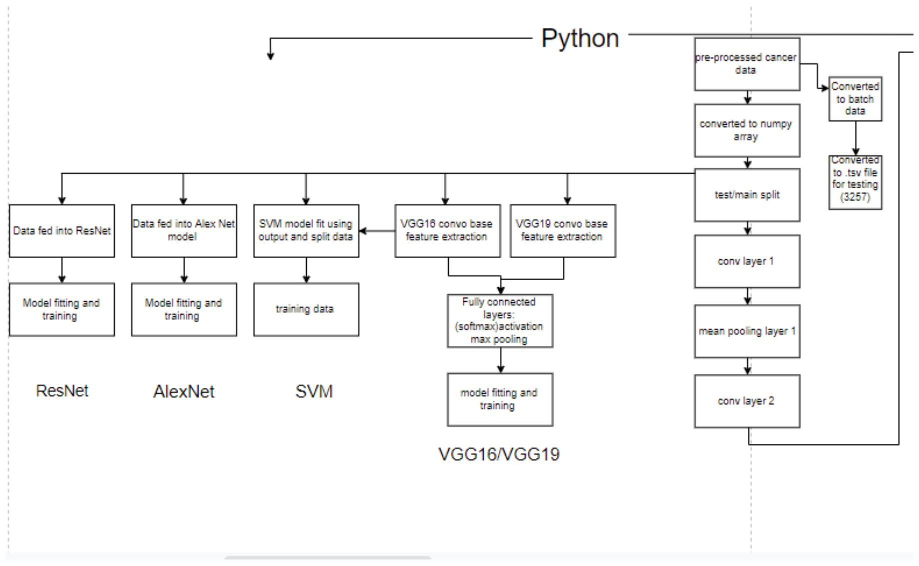


Figure 2.7: ResNet 50 architecture

Chapter 3

Proposed Model

3.1 Workflow Graph



Python
pre-processed cancer data
Converted to batch data
converted to numpy array
Converted to .tsv file for testing (3257)
test/main split
conv layer 1
mean pooling layer 1
conv layer 2
Data fed into ResNet
Model fitting and training
ResNet
Data fed into Alex Net model
Model fitting and training
AlexNet
SVM model fit using output and split data
training data
SVM
VGG16 convo base feature extraction
VGG19 convo base feature extraction
Fully connected layers: (softmax)activation max pooling
model fitting and training
VGG16/VGG19

Figure 3.1: Workflow Diagram

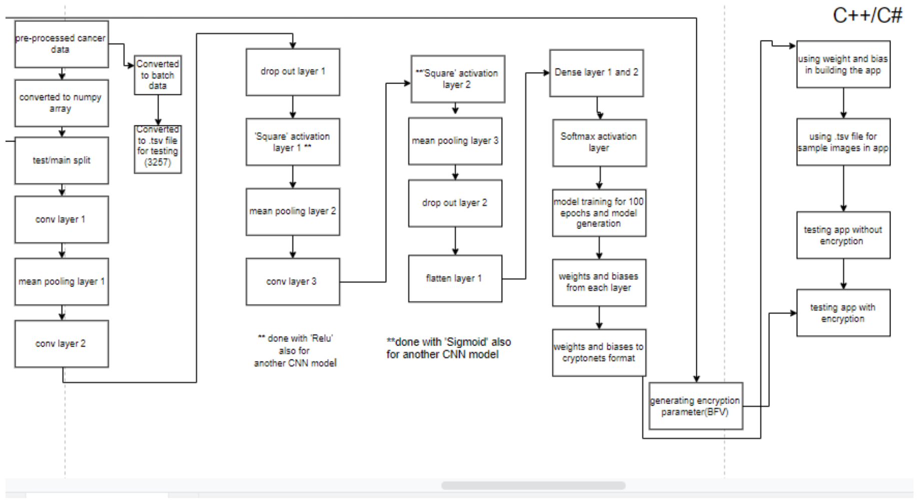

C++/C#
pre-processed cancer data
converted to numpy array
test/main split
conv layer 1
mean pooling layer 1
conv layer 2
Converted to batch data
Converted to .tsv file for testing (3257)
drop out layer 1
'Square' activation layer 1 **
mean pooling layer 2
conv layer 3
** done with 'Relu' also for another CNN model
**'Square' activation layer 2
mean pooling layer 3
drop out layer 2
flatten layer 1
**done with 'Sigmoid' also for another CNN model
Dense layer 1 and 2
Softmax activation layer
model training for 100 epochs and model generation
weights and biases from each layer
weights and biases to cryptonets format
generating encryption parameter(BFV)
using weight and bias in building the app
using .tsv file for sample images in app
testing app without encryption
testing app with encryption

Figure 3.2: Workflow Diagram

3.2 Dataset Description

3.2.1 Dataset and pre-processing and feature selection

Intense lymphoid(lymphoblastic) leukemia (ALL) , a blood cancer/malignancy that is portrayed by the multiplication of unusual lymphoblast cells, in the end prompting the gathering of a deadly number of leukemia cells [21]. In the event that ALL is analyzed in a beginning time, treatment is conceivable. Conclusion is commonly performed using a full blood check and morphological testing of cells using a magnifying instrument by a medicinal master. Stream cytometry can support this work manually, yet needs costly gear that isn't accessible all over the place. Along these lines, robotized frameworks that can perform finding utilizing relatively minimal effort minute pictures give an incredible preferred position. To additionally inquire about toward this path, open datasets are important to analyze various methodologies and track the best in class. A famous model for single cell ALL grouping is ALL-IDB2 [22], however with just 260 white blood cell pictures, the dataset is excessively little to appropriately exploit late profound learning approaches. In 2018, another dataset with in excess of 10,000 preparing pictures and a different test set of ordinary B-lymphoid forerunners and threatening B-lymphoblasts has been discharged as an online test (*) open to the general population. The enormous size of this new dataset permits to make better classifiers dependent upon profound neural systems and furthermore gives an increasingly dependable correlation of contending approaches. Here, we showcase our way in dealing with the arrangement of sound and dangerous cells on the referenced dataset utilizing a convolutional neural system.

*https://competitions.codalab.org/rivalries/20429

The test dataset [23] [24] [25] [26] [27], from now on alluded to as C-NMC dataset, has pictures of platelets(white).This is volunteered by 154 individual subjects, of which 84 display ALL. Table 1 gives a nitty gritty explanation of the quantity of subjects and cells in preparing and test sets. The dataset is imbalanced with about twice the same number of ALL cells as ordinary cells. Each picture has a goal of 450 by 450 pixels and contains just a solitary cell as a result of preprocessing steps applied by the dataset creators: A mechanized division calculation has been utilized to isolate the cells from the foundation. Every pixel that was resolved not to be a piece of the cell is hued totally dark. In any case, since the division calculation isn't great, there are examples where parts of the cell are coincidentally shaded dark or pointless foundation is incorporated. Moreover, the sum total of what pictures have been preprocessed with a stain-standardization system that performs white-adjusting and fixes blunders acquainted due with varieties in the recoloring compound [24]. See Figure 1 for instance pictures from the dataset. Table 1: Composition of the dataset. At the time of writing the ground truth for the final test set is not yet released, so some information is missing.

Dataset part	ALL subjects	Normal Subjects	ALL cells	Normal cells
Train	47	26	7272	3389
Prelim.test	13	15	1219	648
Final test	9	8	?	?

Figure 3.3: Train and test subjects and the corresponding number of samples

Despite the fact that the dataset contains in excess of 10,000 pictures, a few information enlargement strategies can be used to build the measure for preparing greater information and enhance the preparation of our convolutional neural system. Since tiny pictures are invariant to flips and turns, we demonstrate level and vertical flips which has 50 % likelihood and pivots with edge from (0, 360) degrees picked consistently at irregular. Since convolutional neural systems with pooling tasks or walks bigger than one is not flawlessly interpretation invariant, we additionally demonstrate arbitrary interpretations of up to 20 % of each side-length in flat and vertical ways. We don't haphazardly scale the pictures since cell size might be a symptomatic factor to separate among ALL and typical cells [28]. Moreover, we don't have any significant bearing any splendor or shading enlargement because of this current dataset's stain-standardization preprocessing. The two information expansion techniques are usually utilized however would prompt a superfluous dispersion move among preparing and test set on this particular dataset.

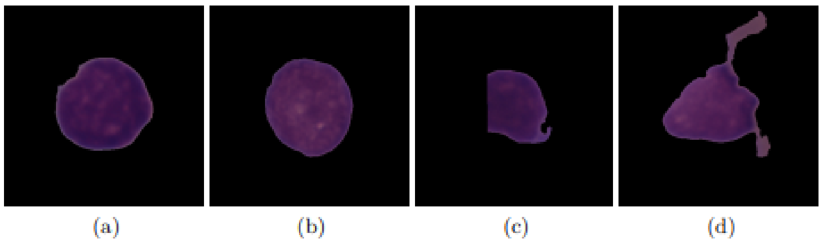


Figure 3.4: Images in the training set. (a) ALL cell (b) Normal cell (c) ALL cell with part of the cell cut off due to an imperfect segmentation (d) Normal cell with superfluous background due to an imperfect segmentation

Also, the pictures are focus trimmed to 100x100 pixels to diminish the dimensionality of the information. This will for the most part make learning a classifier quicker and simpler. Despite the fact that the editing disposes of huge pieces of the picture, it has no impact on the arrangement exactness in light of the fact that without a doubt, not very many cells are really bigger than this harvest. Much of the time, pictures that are not totally dark outside of the harvest are division disappointments that incorporate pieces of the foundation. The dataset is further trimmed, labeled and pre-processed into CIFAR-10 format so that we can run our Crypto Nets model with ease. This part is explained in detail in the next section.

3.3 Model Description

Our own proposed privacy-preserving neural network model consists mainly of 3 components:

1. A Neural-Network model trained and tested with the CNM_C dataset modified to Cifar-10 format in order to run efficiently in Crypto-Nets.
2. A wrapper for Homomorphic Encryption that allow working with matrices and vectors while hiding much of the underlying crypto.
3. Implementation of main Neural-Network layers using the wrapper.

According to the workflow diagram given previously, firstly the CNM_C 2019 dataset is modified, pre-processed to CIFAR-10 format, split into training and test and taken in numpy array accordingly. The conversion of the dataset to CIFAR-10 format is essential because previously Crypto-Nets model has been run on mainly 3 datasets namely Cifar-10,MNIST and Caltech-101 as mentioned earlier of which Cifar-10 deals is much more convenient in dealing with real-life image classification and has an organized 'labeling' along with 'classes' of images in binary format, all of which are convenient in running the Crypto-Nets application using the SEAL version 3.2 HE-wrapper in C and .NET framework version 4.6.2 [20]. The conversion of the dataset to numpy array and using it to train our own cancer predicting Convolutional Neural Network, generating encryption parameters and conversion of test samples to binary version of CIFAR-10 are done prior to building the Crypto-Nets wrapper around it is done using python version 3 code.

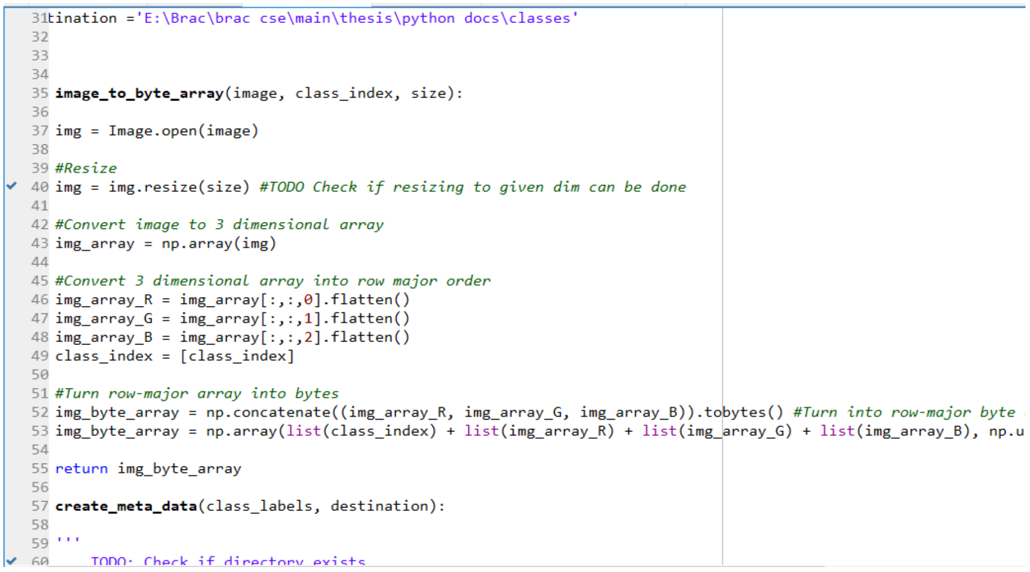
3.3.1 Dataset Conversion and taking into array

For our own Neural Network model:

- After the pre-processing has been done; our 10,000 training images are at first separated equally and placed into 2 different folders with names: 'Cancer' and 'Normal'
- From each class sub folder, we are taking 80% of the images for training and 20% of the images testing. After placing the images, the class subfolders and the images inside the folder are iterated accordingly. An array is first created with dimensions of 32x32 images and an RGB value of '3'.Thus the shape of array would be (32,32,3).For each class subfolder, each image in the subfolder are sliced to obtain the 'R','G' and 'B' values which are then into that array that are concatenated as iteration is done over each image. The array is then appended.

- For the ‘index’ value, a separate array is declared. Each class folders in the input directory would correspond to an image label. The ‘index’ value thus is assigned to each class folder namely ‘0’ for ‘Cancer’ and ‘1’ for ‘Normal’. Each class folder is iterated for images inside and the assigned ‘index’ value is appended in to an array for each iterated image in the subfolder.

- The above steps are repeated for another class subfolder. Below is the code snipped for the training image



```
tination ='E:\Brac\brac cse\main\thesis\python docs\classes'



image_to_byte_array(image, class_index, size):

img = Image.open(image)

#Resize
img = img.resize(size) #TODO Check if resizing to given dim can be done

#Convert image to 3 dimensional array
img_array = np.array(img)

#Convert 3 dimensional array into row major order
img_array_R = img_array[:,:,0].flatten()
img_array_G = img_array[:,:,1].flatten()
img_array_B = img_array[:,:,2].flatten()
class_index = [class_index]

#Turn row-major array into bytes
img_byte_array = np.concatenate((img_array_R, img_array_G, img_array_B)).tobytes() #Turn into row-major byte
img_byte_array = np.array(list(class_index) + list(img_array_R) + list(img_array_G) + list(img_array_B), np.u

return img_byte_array

create_meta_data(class_labels, destination):

'''
    TODO: Check if directory exists
```

Figure 3.5: Code for training dataset conversion to Cifar-10 dataset format and then to test and train arrays accordingly

- The above steps are repeated for the rest 20% of the training images. The test and train image arrays and the corresponding test and train image labels are saved in variables ‘X_train,Y_train’ and ‘X_test, Y_test’. The code snippet is given below

  ..

  np.save(’X_train.npy’, out) Saving train image arrays
  np.save(’Y_train.npy’, index_array) Saving train labels

  np.save(’X_test.npy’, out_test) Saving test image arrays
  np.save(’Y_test.npy’, index_array_test) Saving test labels
  ”

Since the label numpy array is being iterated and concatenated within the same loop as the same array, one-hot encoding is not necessary here. But we are doing it any way just to be on the safe side .Thus numpy arrays are then one-hot encoded where input that is list of ground truth table where ‘0’ is Cancer and ‘1’ is Normal. Thus, the image data taken in the test and train arrays are in Cifar -10 format as with each image taken in ‘X’ the corresponding ‘Y’ label is inserted in the arrays accordingly.

3.3.2 Model Details

Our own 'CNN' model is built using Tensorflow and Keras. Hence, we use 'tf.tensorflow' The neural network of Crypto-Nets supports only the following types of layers: Dense layers, Convolution layers, Square activations, and mean pool layers. Incase of 'Depth' , one should prefer to use only few square activations if possible since there will be a loss in terms of inference latency and memory requirements for networks with many nonlinear transformations. Incase of 'Width', to improve performance, it is beneficial to make sure that each hidden layer is not wider than the width of the cipher text which is typically 8192 or 16384. To improve performance, it is beneficial to make sure that the inputs and weights do not require high fidelity to ensure correct predictions. This lessens the number of bits required in each message and allows working with smaller parameters. At training time this can be achieved by quantizing the inputs before training. As an example, if the inputs are numbers in the range 0-255 the following command will normalize them to be in the range 0-1 as well as quantize them to have only 8 levels. According to our workflow diagram:

- The images are at first pre-processed, taken in numpy arrays and split accordingly as mentioned in section 3.2
- The arrays are then passed through the layers sequentially as shown in the workflow diagram.
- The input tensor of [32,32,3] passes through the first 'Convolutional' layer. The tensor shape is changed to [32,32,128] output with 'same' padding in the layer and kernel size of 3x3. Due to 'same' padding, there won't be any shrinking outputs or loosing information on corners of the image. It is the first layer where output parameters are obtained and feature extraction takes place. The job of the 'Convolutional' layer here is to make the product of the vector of values at the layer below it and a weights vector and sum of the outputs. During the inference processes, the weights being kept constant. The function is mainly a dot product of the vector of the weights and the vector of magnitudes of the feeding layer. The convolution layer consists of a set of independent filters. Each filter convolves independently with the image and the result is 128 feature maps of shape 32*32*1. All these filters are initialized randomly and become our parameters which will be learned by the network subsequently. In a definite feature map (the output received on convolving the image with a particular filter is called a feature map), each neuron is connected only to a small part of the input image and all the neurons have the similar link'weights. Hence, explaining the feature extraction. This layer has parameters equal to 3584.
- Next, the data from the output tensor from the previous layers passes through the first 'Mean Pooling Layer'. Its function is to sequentially lessen the spatial size of the output in reducing the number of parameters and calculations in the network. Pooling layer works on each feature map independently. No parameters are there in the layer but it has a pool size of 2x2 the output tensor shape changes to [16,16,128].
- The data from the output tensor of the 'Mean Average Pooling Layer' is passed to the 2nd 'Convolutional' layer which has no 'padding' and has a kernel size

of 3x3 .Further feature extraction takes place in this layer like in step 3 . The shape of the layer is now [14,14,83] with parameters equal to 95699.

- The tensor is then passed to the first ‘Dropout’ layer. ‘Dropout’ basically refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. It is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons. Technically, at each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. It is a layer is used to prevent model ‘over-fitting’ .Here, ‘Dropout’ layer after every pooling layer will be used to reduce ‘over-fitting’ in the model with setting the fraction value to .25 so that .25 fraction of units will be dropped .The tensor shape however remains unchanged with no trainable parameters.

- After some neurons being “dropped-out”, the tensor passes through the first ‘Activation’ layer where an activation function operates on the input data from the previous neurons. In our model, there are 3 ‘Activation layers with this being the first one. Our model is trained at first with functions ‘ Relu’ in the 1st and 2nd and ‘Sigmoid’ in the 3rd Activation layers then again is trained using ‘Square’ and ‘Softmax’ layers later the same way. The tensor shape remains the same and no trainable parameters.

  Sigmoid: Take the value of one of the nodes from the feeding layer and calculates the function
  $$z- > \frac{1}{(1+exp(-z))}$$

Rectified Linear: Takes the value of one of the nodes from the feeding layer and calculates the function
$$z- > \frac{1}{max(0,z)}.$$

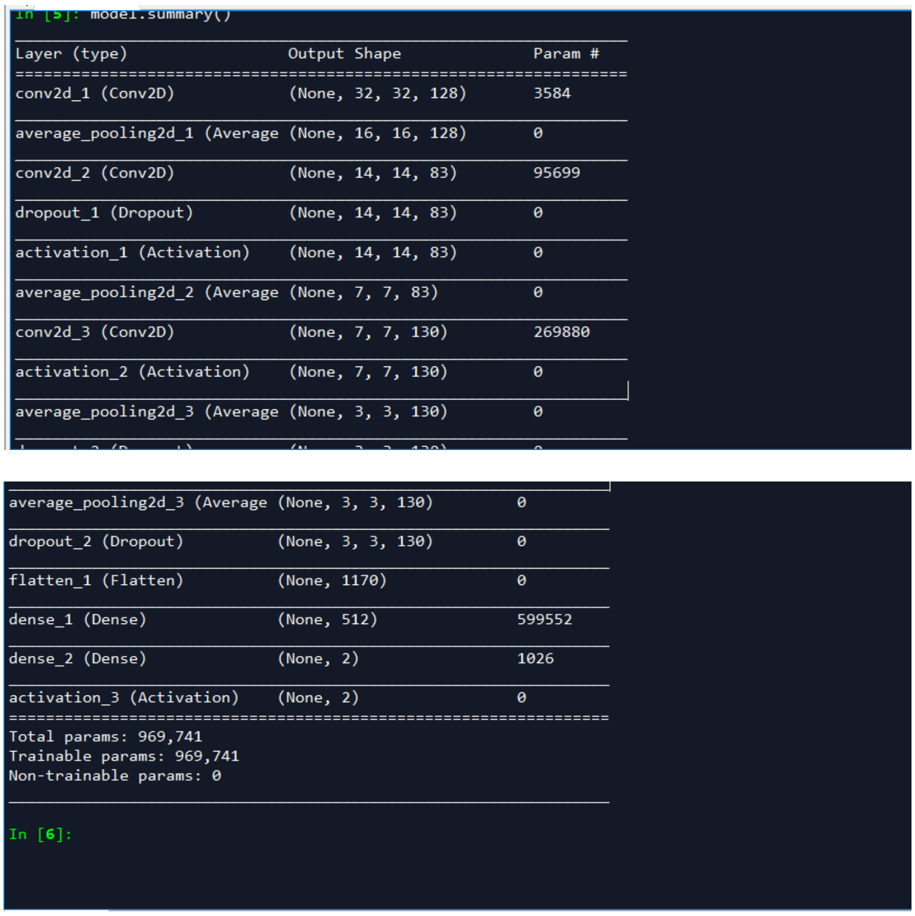
Square Activation Layer: The value at each input node is squared by this layer.

Softmax Layer: This activation function forces the output neurons to take values between zero and one, so that they can represent the probability score.

’ Sigmoid’ and ’Relu’ activation functions are non polynomials. The fix was to estimate these functions with low-degree polynomials but here we will be using a different method. [15] We tried to manipulate the trade-off between possesses a non-linear transformation required by the learning algorithm and also need to maintain the degree of the polynomials minimal to make the parameters of homomorphic encryption realistic. We opted to use the non-linear lowest degree polynomial function, which is the Square function: sqr (z):= z^2. It has been suggested by a theoretical study of a problem regarding neural networks with polynomial activation functions and dedicated majority of their study to the square activation function. [29]

For the training stage, the sigmoid activation function is used to get reasonable terms of error when running the gradient descent algorithm. However in the encrypted world, we don't have a reasonable way to deal with the sigmoid. Fortunately, once we have our weights set and would like to make predictions, we can just take it out. This is because the neural network's prediction is given by the index of its output vector's maximum value, and since the sigmoid function is increasing monotonously, whether we apply it or not will not affect the prediction.

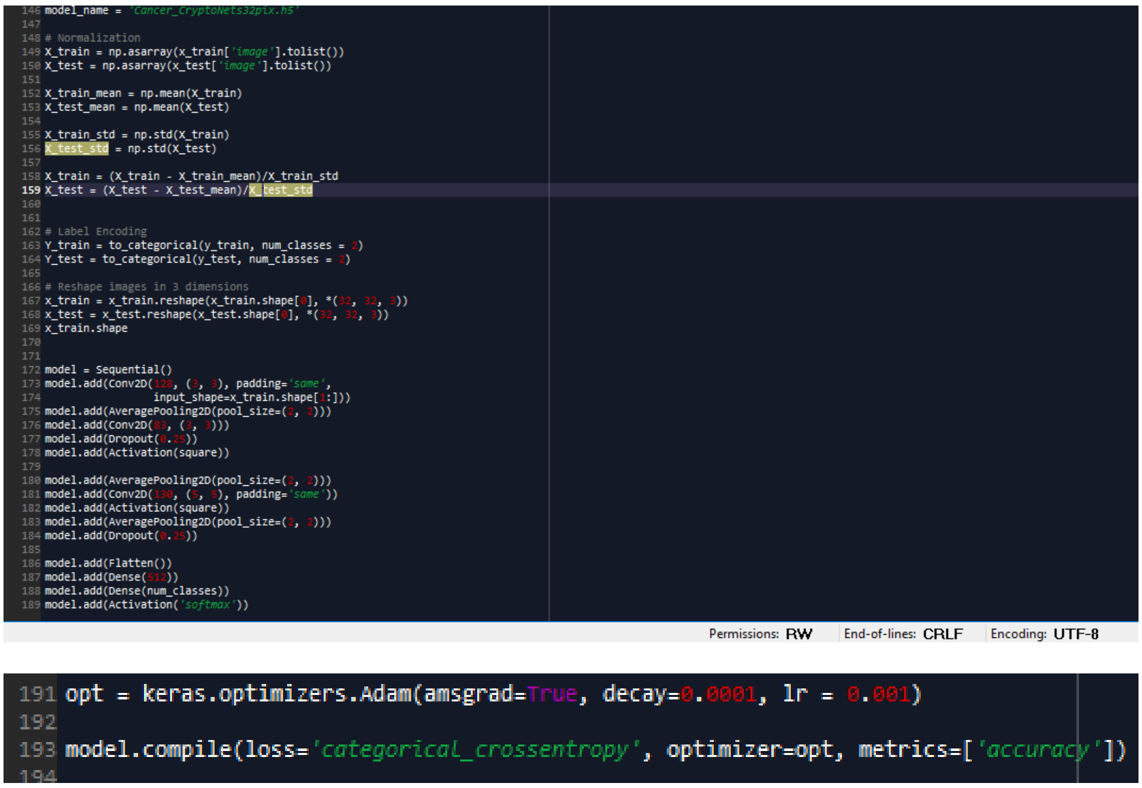
- The output tensor from the 1st activation layer is then fed to 2nd 'Mean Average Pooling Layer' with pool size of 2x2 also where the dimension of the output tensor changes to [7,7,83].
- This tensor is then passed to the last 'Convolutional' layer with 'same' padding and a kernel size of 5x5. The shape of the output tensor changes to [7,7,130] and the parameters are now equal to 256880.
- This tensor is now fed to the 2nd 'Activation Layer' where the activation function is either 'Sigmoid' or 'Softmax'.
- In the last 'Mean Average Pooling' layer, the shape of the output tensor changes [3,3,130] which is then fed to the last 'Dropout' layer.
- The tensor is then passed to the first "Flatten" Layer (a layer which transforms the two-dimensional matrix into a vector to be easily fed into a fully connected neural network) which is found between the Fully connected and Convolutional Layers. Here, the matrices are flattened in a shape of [,1170] finally feeding it to the 2 'Dense' Layers
- In a "Dense" Layer, the neurons are densely connected since all the inputs of the neurons from the previous layers are received by each neuron from this layer. In the layer there is a weight matrix W, a bias vector 'b', and the activations of previous layer 'a'. The first 'Dense' layer gives an output shape of (,512) with parameters equal to 599552 and the 2nd 'Dense' layer being [,2] with parameters equal 1026.
- The last output tensor is from an 'Activation' layer either 'Sigmoid' or 'Softmax' which finally gives an output of [,2] representing output tensor for 2 classes in the Neural Network.
  The picture of the summary of the model is given below:
- The model is them compiled using 'Adam' as optimizer and taking loss function as 'categorical_crossentropy'. The model is then trained 100 epochs. After training the model is loaded. The part of code using all the layers and model compilation is shown below



```
In [5]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 128)       3584
_________________________________________________________________
average_pooling2d_1 (Average (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 83)        95699
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 83)        0
_________________________________________________________________
activation_1 (Activation)    (None, 14, 14, 83)        0
_________________________________________________________________
average_pooling2d_2 (Average (None, 7, 7, 83)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 130)         269880
_________________________________________________________________
activation_2 (Activation)    (None, 7, 7, 130)         0
_________________________________________________________________
average_pooling2d_3 (Average (None, 3, 3, 130)         0
_________________________________________________________________
```

```
_________________________________________________________________
average_pooling2d_3 (Average (None, 3, 3, 130)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 3, 3, 130)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1170)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               599552
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
_________________________________________________________________
activation_3 (Activation)    (None, 2)                 0
=================================================================
Total params: 969,741
Trainable params: 969,741
Non-trainable params: 0
_________________________________________________________________

In [6]:
```

Figure 3.6: Summary of layers in our own model



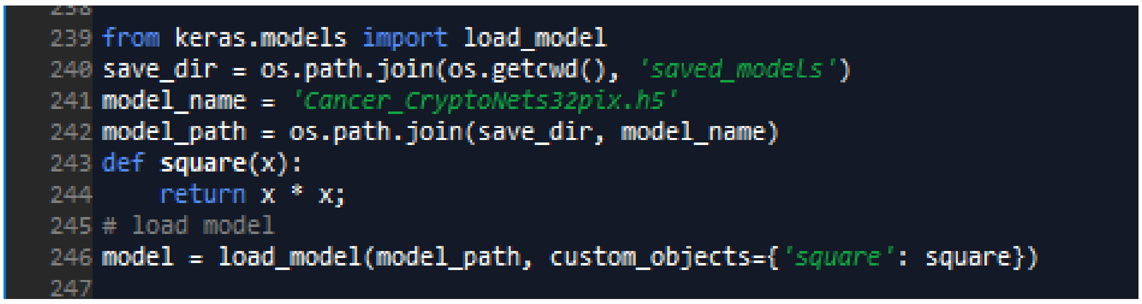
```
model_name = 'Cancer_CryptoNets32pix.h5'

# Normalization
X_train = np.asarray(x_train['image'].tolist())
X_test = np.asarray(x_test['image'].tolist())

X_train_mean = np.mean(X_train)
X_test_mean = np.mean(X_test)

X_train_std = np.std(X_train)
X_test_std = np.std(X_test)

X_train = (X_train - X_train_mean)/X_train_std
X_test = (X_test - X_test_mean)/X_test_std


# Label Encoding
Y_train = to_categorical(y_train, num_classes = 2)
Y_test = to_categorical(y_test, num_classes = 2)

# Reshape images in 3 dimensions
x_train = x_train.reshape(x_train.shape[0], *(32, 32, 3))
x_test = x_test.reshape(x_test.shape[0], *(32, 32, 3))
x_train.shape


model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(AveragePooling2D(pool_size=(2, 2)))
model.add(Conv2D(83, (3, 3)))
model.add(Dropout(0.25))
model.add(Activation(square))

model.add(AveragePooling2D(pool_size=(2, 2)))
model.add(Conv2D(130, (5, 5), padding='same'))
model.add(Activation(square))
model.add(AveragePooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8

```
opt = keras.optimizers.Adam(amsgrad=True, decay=0.0001, lr = 0.001)

model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Figure 3.7: Code for adding layers to custom model

- Since CryptoNets models are using the square activation which does not exist in Keras. We have thus defined the square function before loading a previously saved model. Below is the code snippet.

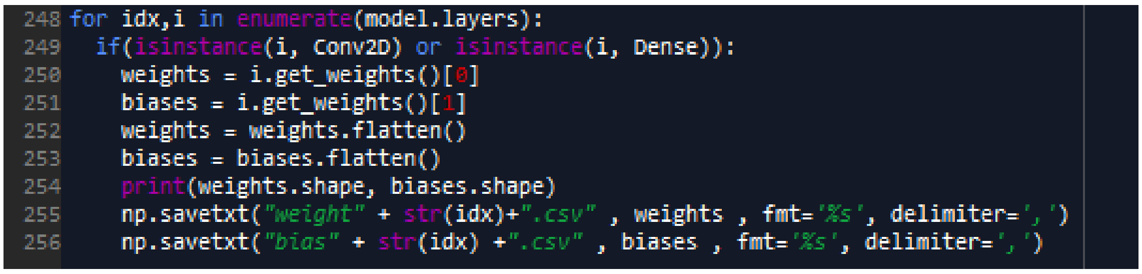


```
from keras.models import load_model
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'Cancer_CryptoNets32pix.h5'
model_path = os.path.join(save_dir, model_name)
def square(x):
    return x * x;
# load model
model = load_model(model_path, custom_objects={'square': square})
```

Figure 3.8: Code for adding layers to custom model

Converting Weights and Biases to Crypto-Nets format

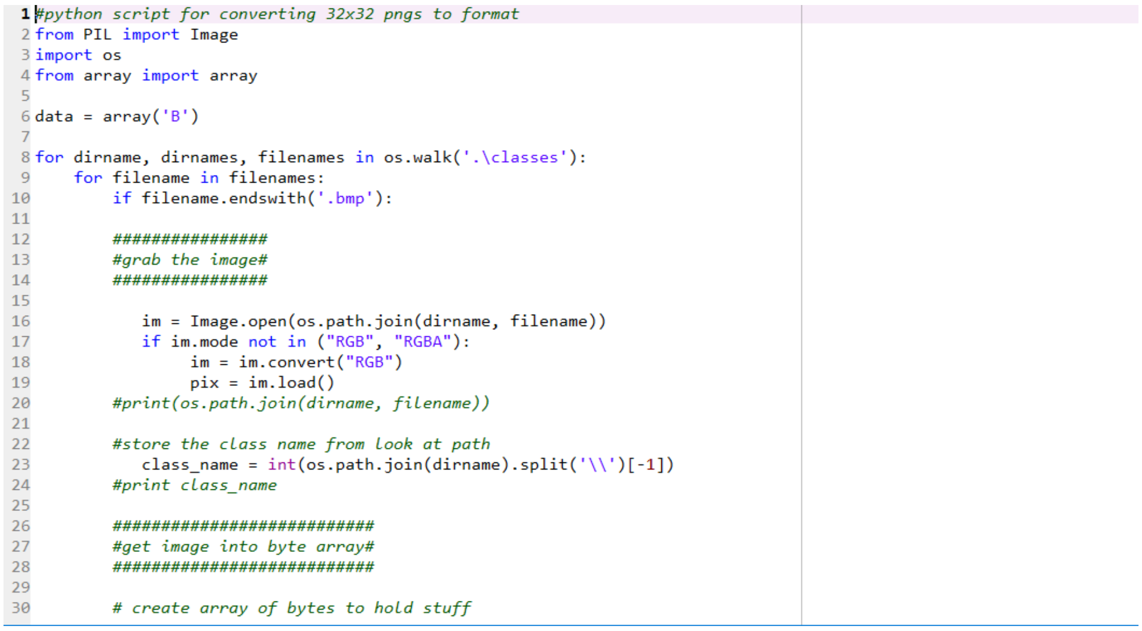
Once the model is training the next step is to convert the weights and bias vectors to a format that Crypto-Nets recognizes. Crypto-Nets expects the weights to be in a CSV file where the weights for each layer are in a separate line. One challenge is to collapse adjected linear layers into a single linear layer. For each layer with trainable weights (a dense layer or a convolution layer) a bias file and a weights file should be generated. Once done that for all the relevant layers, we combine all the weights into a one file and all the biases into a second file. Below is the code snippet of how the 'weights' and 'biases' of the 'Convolutional' and 'Dense' layers are obtained as a separate file. A total of 10 files (5 for weights and 5 for biases) are generated for the 3 ‘Convolutional’ and 2 ‘Dense’ layers.



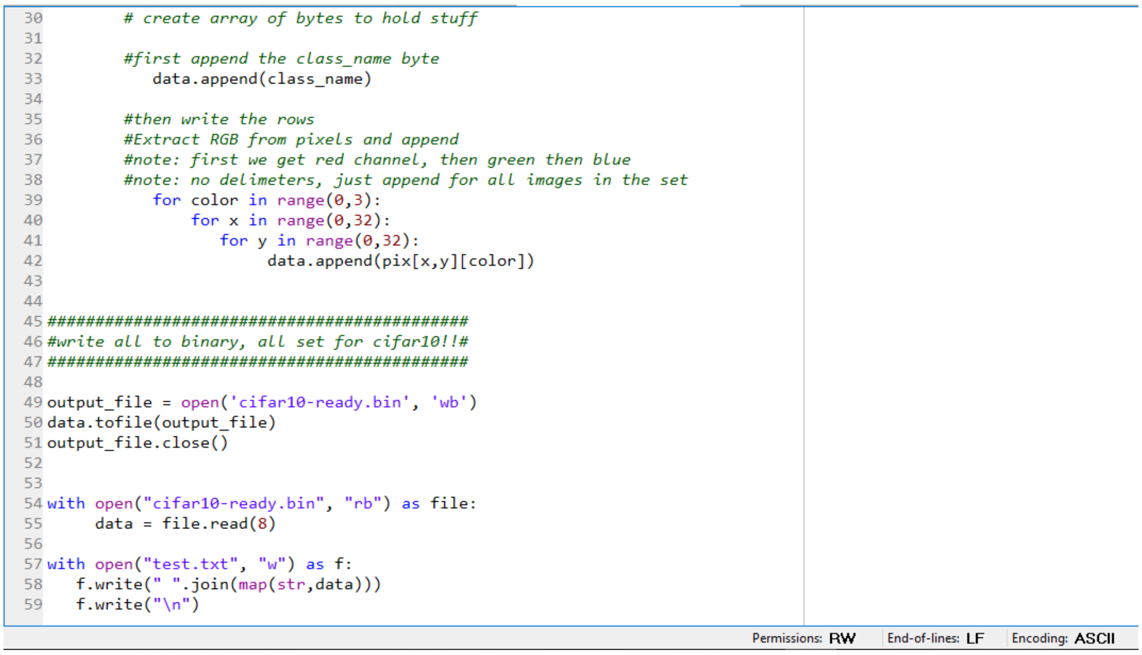
```python
for idx,i in enumerate(model.layers):
  if(isinstance(i, Conv2D) or isinstance(i, Dense)):
    weights = i.get_weights()[0]
    biases = i.get_weights()[1]
    weights = weights.flatten()
    biases = biases.flatten()
    print(weights.shape, biases.shape)
    np.savetxt("weight" + str(idx)+".csv" , weights , fmt='%s', delimiter=',')
    np.savetxt("bias" + str(idx) +".csv" , biases , fmt='%s', delimiter=',')
```

Figure 3.9: Code for adding layers to custom model

Values in the files are now in single columns. Thus, each column in each file of all the weights and biases for each layer in transposed into single rows. All the ‘weights.csv’ and ‘bias.csv’ files are combined to a single ‘all_weights.csv’ and ‘all_bias.csv’ file .This is done using passing a command in the windows command shell after installing the ‘Gnu’ software for windows. Below is the command line for it:
gawk ‘{print $0;}’ weights1.csv weights2.csv weights3.csv... > all_weights.csv
The same is repeated for biases:
gawk ‘{print $0;}’ bias1.csv bias2.csv bias3.csv... > all_bias.csv
Building and Testing the application without Encryption
The model is first tested without any encryption parameters. Prior to that, the ‘test.tsv’ file is created in python. At first we a create ‘.bin’ file similar to the binary version of the CIFAR-10 dataset for our test samples of the cancer dataset which had been trimmed, pre-processed and put into folders with labels ‘0’ and ‘1’ in order to work with Crypto-Nets like the Cifar-10 dataset. The test samples of the cancer dataset are thus arranged accordingly. The ‘.bin’ file hence is a batch file created containing binary version of the 3527 test samples arranged in bytes in the .bin file. The code snippet for the file conversion is given below:



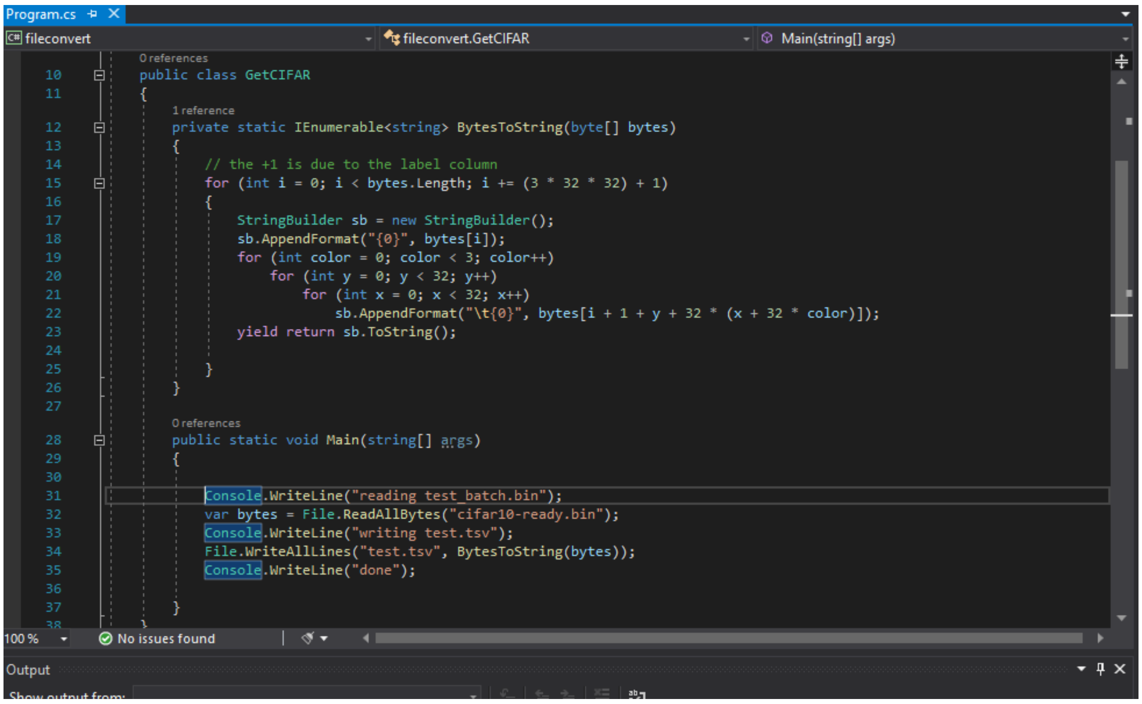
```python
#python script for converting 32x32 pngs to format
from PIL import Image
import os
from array import array

data = array('B')

for dirname, dirnames, filenames in os.walk('.\classes'):
    for filename in filenames:
        if filename.endswith('.bmp'):

        ################
        #grab the image#
        ################

           im = Image.open(os.path.join(dirname, filename))
           if im.mode not in ("RGB", "RGBA"):
                im = im.convert("RGB")
                pix = im.load()
        #print(os.path.join(dirname, filename))

        #store the class name from look at path
           class_name = int(os.path.join(dirname).split('\\')[-1])
        #print class_name

        ###########################
        #get image into byte array#
        ###########################

        # create array of bytes to hold stuff
```



```python
        # create array of bytes to hold stuff

        #first append the class_name byte
          data.append(class_name)

        #then write the rows
        #Extract RGB from pixels and append
        #note: first we get red channel, then green then blue
        #note: no delimeters, just append for all images in the set
          for color in range(0,3):
              for x in range(0,32):
                  for y in range(0,32):
                      data.append(pix[x,y][color])


###############################################
#write all to binary, all set for cifar10!!#
###############################################

output_file = open('cifar10-ready.bin', 'wb')
data.tofile(output_file)
output_file.close()


with open("cifar10-ready.bin", "rb") as file:
    data = file.read(8)

with open("test.txt", "w") as f:
  f.write(" ".join(map(str,data)))
  f.write("\n")
```

Figure 3.10: Code for generating .bin file of our dataset in CIFAR-10 format

The '.bin' is then converted to '.tsv file where should have one line per image where each line contains 1 + 33232 tab separated columns in which the first column is the label and the other column are the RGB values of a 32*32 image. The bytes in the '.bin' file are converted to strings when converting to '.tsv'. This is done using C. The code snippet for this is given below:

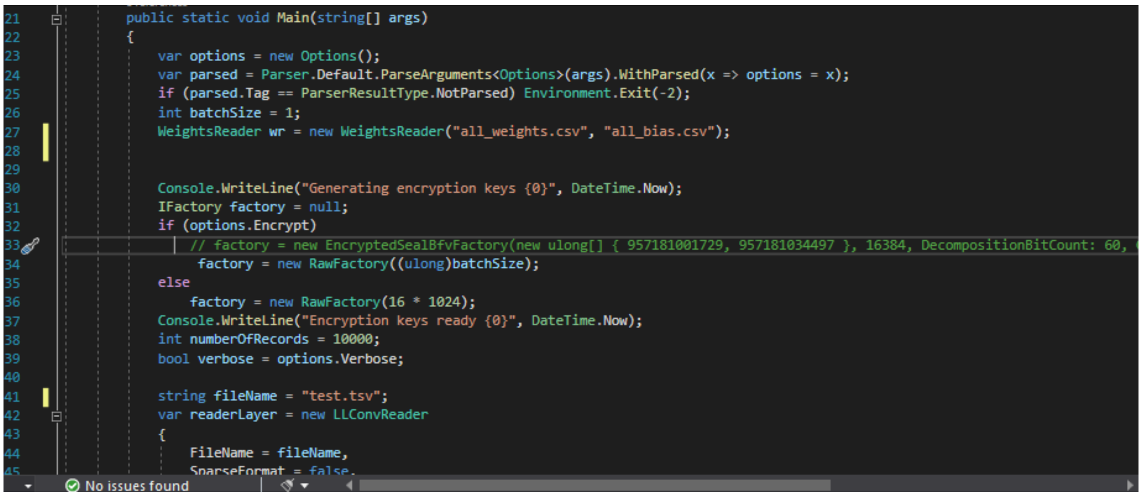


```csharp
public class GetCIFAR
{
    private static IEnumerable<string> BytesToString(byte[] bytes)
    {
        // the +1 is due to the label column
        for (int i = 0; i < bytes.Length; i += (3 * 32 * 32) + 1)
        {
            StringBuilder sb = new StringBuilder();
            sb.AppendFormat("{0}", bytes[i]);
            for (int color = 0; color < 3; color++)
                for (int y = 0; y < 32; y++)
                    for (int x = 0; x < 32; x++)
                        sb.AppendFormat("\t{0}", bytes[i + 1 + y + 32 * (x + 32 * color)]);
            yield return sb.ToString();

        }
    }

    public static void Main(string[] args)
    {

        Console.WriteLine("reading test_batch.bin");
        var bytes = File.ReadAllBytes("cifar10-ready.bin");
        Console.WriteLine("writing test.tsv");
        File.WriteAllLines("test.tsv", BytesToString(bytes));
        Console.WriteLine("done");

    }
}
```

Figure 3.11: Code for .bin to.tsv

The application is coded in C# using 'Visual Studio 2019' and was tested in the windows environment used .Net framework version 4.6.2. This project depends on

SEAL version 3.2 .Thus a Nuget package containing SEAL, is added as reference which is essential.The 'all weights' and 'all biases' are passed in the 'WeightsReader' function and the parameters are loaded. The string file is passed into the application. The project is then built in x64 architecture The code snippet is given below:



```
    public static void Main(string[] args)
    {
        var options = new Options();
        var parsed = Parser.Default.ParseArguments<Options>(args).WithParsed(x => options = x);
        if (parsed.Tag == ParserResultType.NotParsed) Environment.Exit(-2);
        int batchSize = 1;
        WeightsReader wr = new WeightsReader("all_weights.csv", "all_bias.csv");


        Console.WriteLine("Generating encryption keys {0}", DateTime.Now);
        IFactory factory = null;
        if (options.Encrypt)
            // factory = new EncryptedSealBfvFactory(new ulong[] { 957181001729, 957181034497 }, 16384, DecompositionBitCount: 60,
             factory = new RawFactory((ulong)batchSize);
        else
            factory = new RawFactory(16 * 1024);
        Console.WriteLine("Encryption keys ready {0}", DateTime.Now);
        int numberOfRecords = 10000;
        bool verbose = options.Verbose;

        string fileName = "test.tsv";
        var readerLayer = new LLConvReader
        {
            FileName = fileName,
            SparseFormat = false,
```

No issues found

Figure 3.12: Code snippet for the application

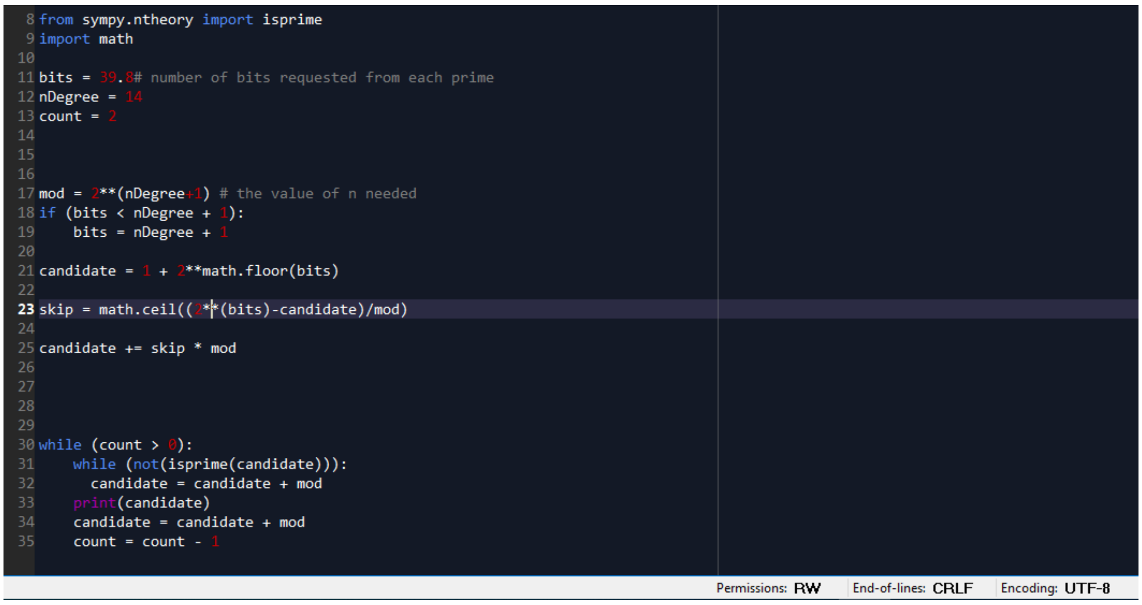
Notice that the line of code: var Factory = new EncryptedSealBfvFactory(new ulong[] 957181001729, 957181034497 , 16384); is commented out. Thus, the variable 'factory' has only one value.

Selecting Encryption Parameters(key generation)

The theoretical process and mathematical formulae to calculate the correct parameters are given in the previous section 2.3 'Parameter Selection' part. We know from our previous section of 2.2 , to allow correctness the parameters should support large enough number to be processed. Much like in traditional programming where a program might fail if number are allocated with insufficient space (short integers vs. long integers or floats vs. doubles), the same thing may happen when using homomorphic encryption. Thus, The first step is to determine the amount of space needed. When running without encryption (using the RawFactory), CryptoNets keys track of the size of number of processes using the SEAL function 'Raw.Matrix.Max' which keeps track of the maximum number used (in absolute value) and the number of bits this number required to encode this number. To determine the number of bits needed, we add 1 to this number since an additional bit is required to hold the sign of the number.

To provide the required number of bits, a number of prime numbers are provided such that the product of these numbers is as at least the required number of bits. For example, if 70 bits are needed, we can use 2 prime numbers with 35 bits each. Working with more prime numbers increases the running time. However, smaller primes allow more computation to be done before the noise budget exceeds.
Noise budget is another important parameter of Homomorphic Encryption. In a nut-shell, a freshly encrypted number has a certain amount of noise budget. Every operation on such number (addition, multiplication, etc) reduces this budget. Once this budget equals zero, the decryption will fail to provide correct results. The amount of noise budget available is determined by several parameters, the most important of them are the dimension used. (N) and the size of the prime numbers used as plaintext-modulus. The dimension N should be a power of two, the larger it is, the greater the noise budget is. However, the larger N is, the slower the program runs. Typical values for 'N' range from 2ˆ2 to 2ˆ5. On the other hand, greater noise budget is available when the plaintext modulus is smaller. However, working with smaller plaintext modulus requires using more plaintext modulus to achieve the required number of bits and therefore slows down the application. Selecting a good set of parameters is currently done manually.

After determining the required number of bits, select a value for N and the number of primes to be used. 3 parameters are specified to generate the encryption parameters that are to be passed in the application. The code in python 3 generates these parameters:



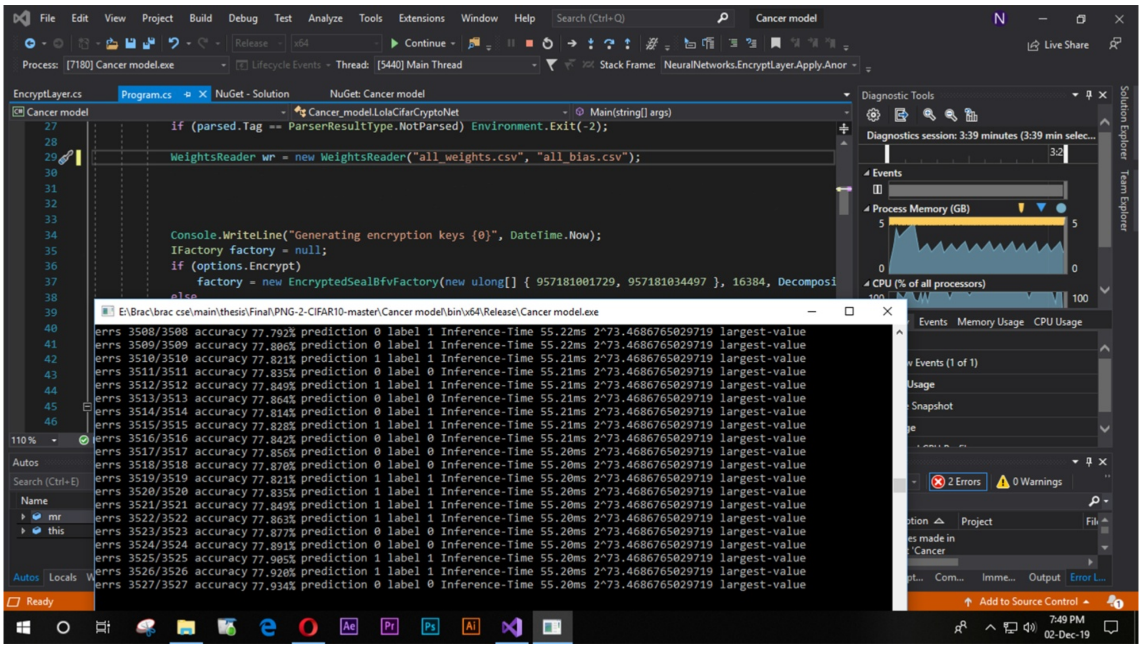
```python
from sympy.ntheory import isprime
import math

bits = 39.8# number of bits requested from each prime
nDegree = 14
count = 2



mod = 2**(nDegree+1) # the value of n needed
if (bits < nDegree + 1):
    bits = nDegree + 1

candidate = 1 + 2**math.floor(bits)

skip = math.ceil((2**(bits)-candidate)/mod)

candidate += skip * mod




while (count > 0):
    while (not(isprime(candidate))):
      candidate = candidate + mod
    print(candidate)
    candidate = candidate + mod
    count = count - 1
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8

Figure 3.13: Python code for generating encryption parameters

In the code above 3 parameters are set where ‘bits’ is the minimal number of bits of each prime, ‘ndegree’ is the number of bits in N and ‘count’ is the number of primes to generate. The code above generates parameters of 957181001729 and 957181034497

These parameters are passed into the application and the line of code: var Factory = new EncryptedSealBfvFactory(new ulong[] 957181001729, 957181034497 , 16384); where 16384 is the value of ‘N’. Since 2 prime numbers were demanded with 39.8 bits each, these parameters can support 79.6 bits. Below are the outputs after the application with parameters has been created:



```csharp
            if (parsed.Tag == ParserResultType.NotParsed) Environment.Exit(-2);

            WeightsReader wr = new WeightsReader("all_weights.csv", "all_bias.csv");




            Console.WriteLine("Generating encryption keys {0}", DateTime.Now);
            IFactory factory = null;
            if (options.Encrypt)
                factory = new EncryptedSealBfvFactory(new ulong[] { 957181001729, 957181034497 }, 16384, Decomposi
```

E:\Brac\brac cse\main\thesis\Final\PNG-2-CIFAR10-master\Cancer model\bin\x64\Release\Cancer model.exe

```
errs 3508/3508 accuracy 77.792% prediction 0 label 1 Inference-Time 55.22ms 2^73.4686765029719 largest-value
errs 3509/3509 accuracy 77.806% prediction 0 label 1 Inference-Time 55.22ms 2^73.4686765029719 largest-value
errs 3510/3510 accuracy 77.821% prediction 1 label 1 Inference-Time 55.21ms 2^73.4686765029719 largest-value
errs 3511/3511 accuracy 77.835% prediction 0 label 0 Inference-Time 55.21ms 2^73.4686765029719 largest-value
errs 3512/3512 accuracy 77.849% prediction 1 label 1 Inference-Time 55.21ms 2^73.4686765029719 largest-value
errs 3513/3513 accuracy 77.864% prediction 0 label 0 Inference-Time 55.21ms 2^73.4686765029719 largest-value
errs 3514/3514 accuracy 77.814% prediction 0 label 1 Inference-Time 55.21ms 2^73.4686765029719 largest-value
errs 3515/3515 accuracy 77.828% prediction 1 label 1 Inference-Time 55.21ms 2^73.4686765029719 largest-value
errs 3516/3516 accuracy 77.842% prediction 0 label 0 Inference-Time 55.21ms 2^73.4686765029719 largest-value
errs 3517/3517 accuracy 77.856% prediction 0 label 0 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3518/3518 accuracy 77.870% prediction 0 label 0 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3519/3519 accuracy 77.821% prediction 1 label 1 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3520/3520 accuracy 77.835% prediction 1 label 1 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3521/3521 accuracy 77.849% prediction 1 label 1 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3522/3522 accuracy 77.863% prediction 1 label 1 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3523/3523 accuracy 77.877% prediction 0 label 0 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3524/3524 accuracy 77.891% prediction 0 label 0 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3525/3525 accuracy 77.905% prediction 1 label 1 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3526/3526 accuracy 77.920% prediction 1 label 1 Inference-Time 55.20ms 2^73.4686765029719 largest-value
errs 3527/3527 accuracy 77.934% prediction 0 label 0 Inference-Time 55.20ms 2^73.4686765029719 largest-value
```

Figure 3.14: Output for the encrypted prediction

Where each sample is checked with the label and the accuracy is measure for each sample along with inference time for each sample. Here, the output accuracy of each sample is 77.9% a bit less than 79% when running an un-encrypted model.

Chapter 4

Experiments and Result Analysis

Each model mentioned earlier in the paper is trained on a PC of GTX 750ti, 8gb Ram and a processor of core i5 4th generation. Each model is trained for100 epochs except for AlexNet and ResNet which are trained for approximately 20 epochs since they are better CNN models with more convolutional layers and training them for more epochs may result in 'over-fitting'.

The validation and training accuracy graphs for experimental models for our modified dataset are mentioned given below. Some of them are generated in code and some from Alexnet:

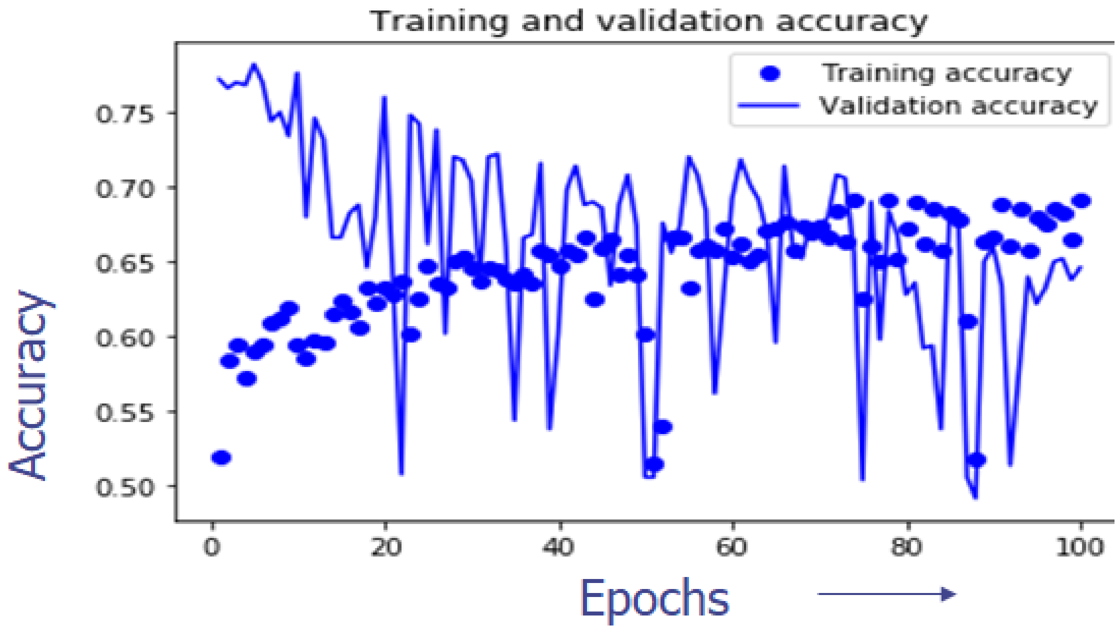


Figure 4.1: Training and Validation accuracy for VGG16

SVM:

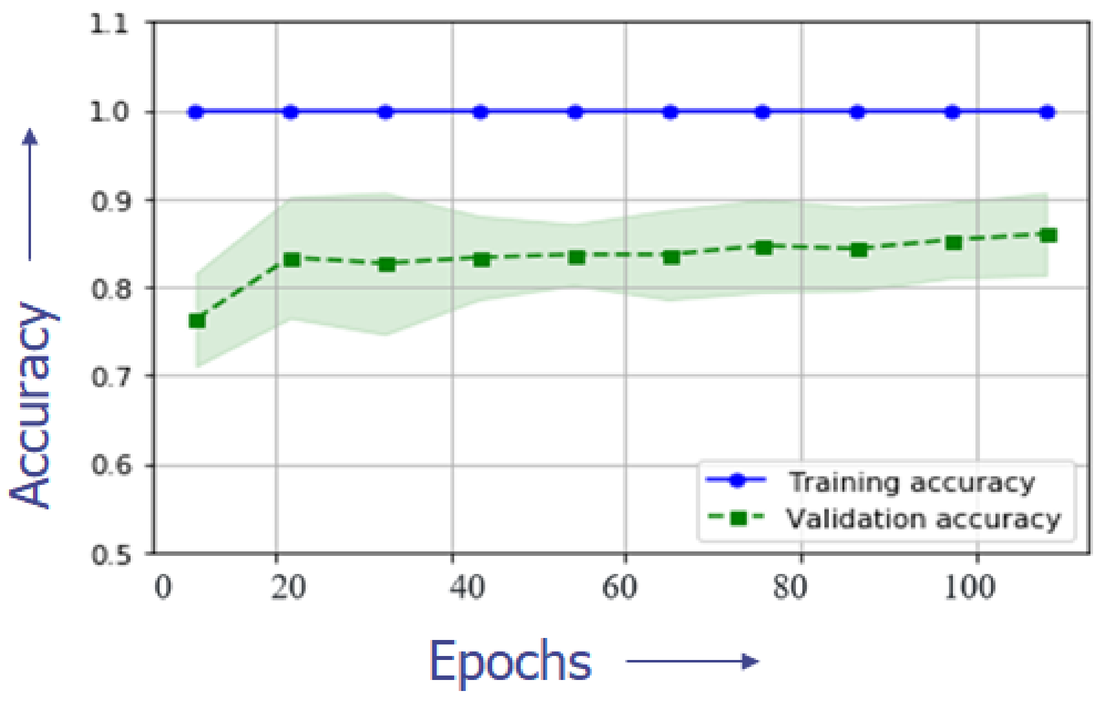


Figure 4.2: Training and Validation accuracies for SVM

AlexNet: Validation accuracy

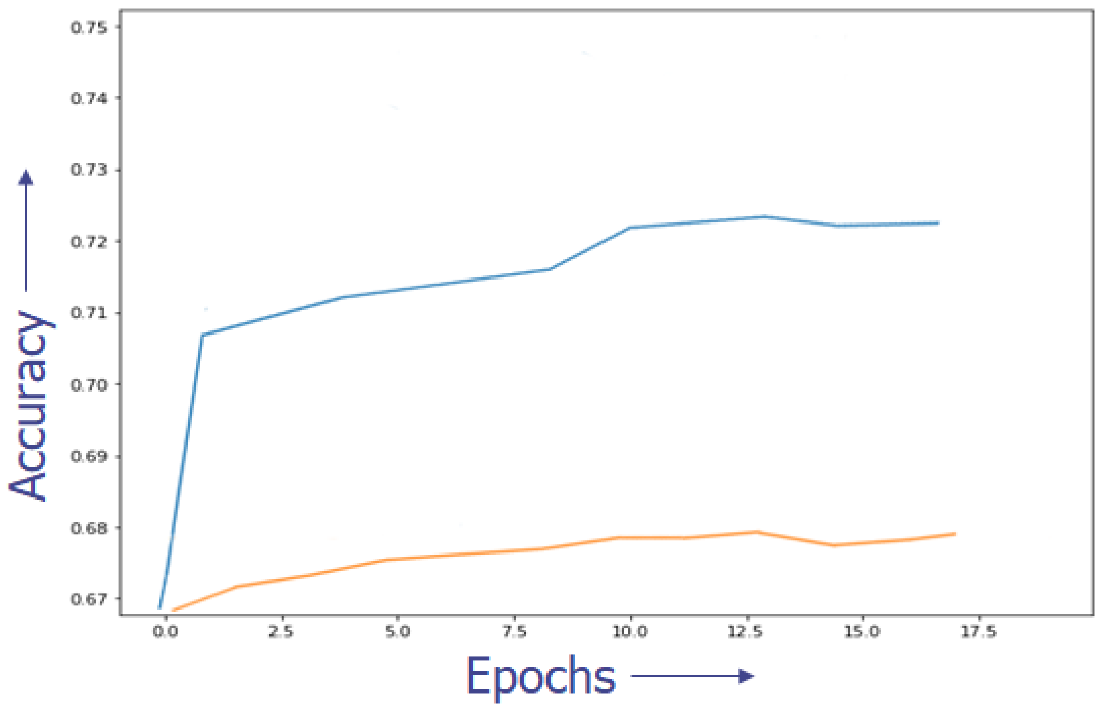


Figure 4.3: AlexNet validation(orange) and train accuracy(blue)

From the graph after training AlexNet model after 100 epochs , we see that the training accuracy is approximately 73% which is higher than the steady increasing validation accuracy of 68%.The model thus is not over-fitting Both the model's training and testing accuracy increases at a decreasing rate .

ResNet50:

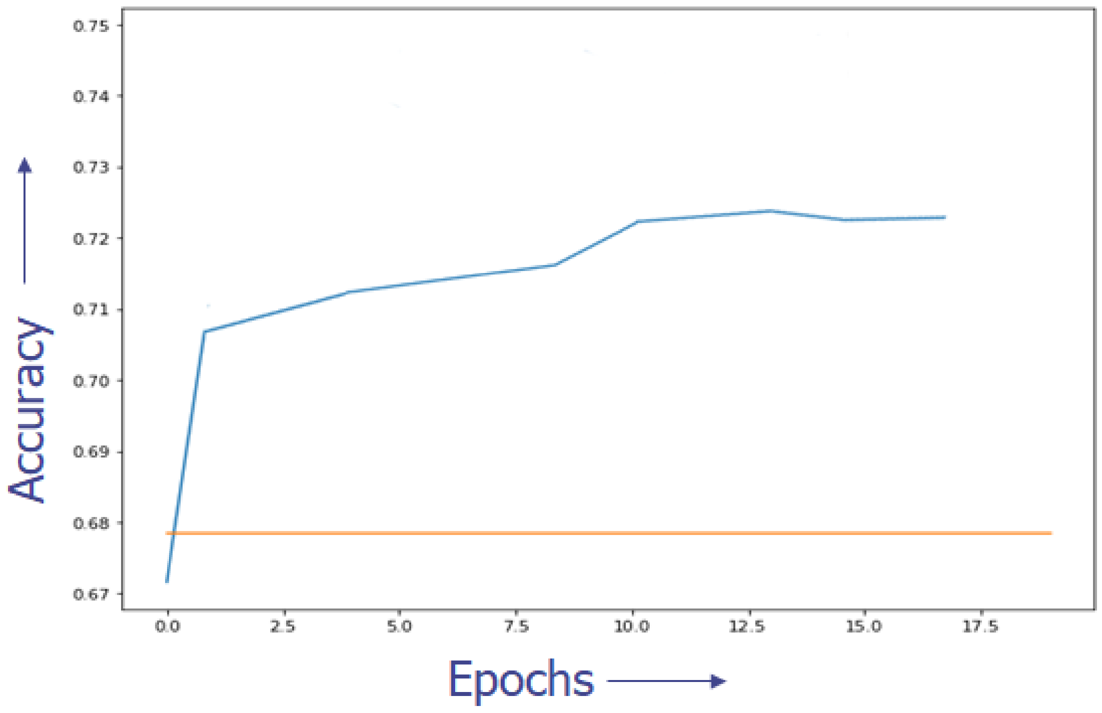


Figure 4.4: Validation and training accuracy of ResNet50 model

From the graph after training AlexNet model after 100 epochs , we see that the training accuracy is approximately 72.50% which is higher than the steady increasing validation accuracy of 67.80%.The model it seems is not over-fitting The model's training accuracy increases at a decreasing rate but the validation accuracy remains constant.

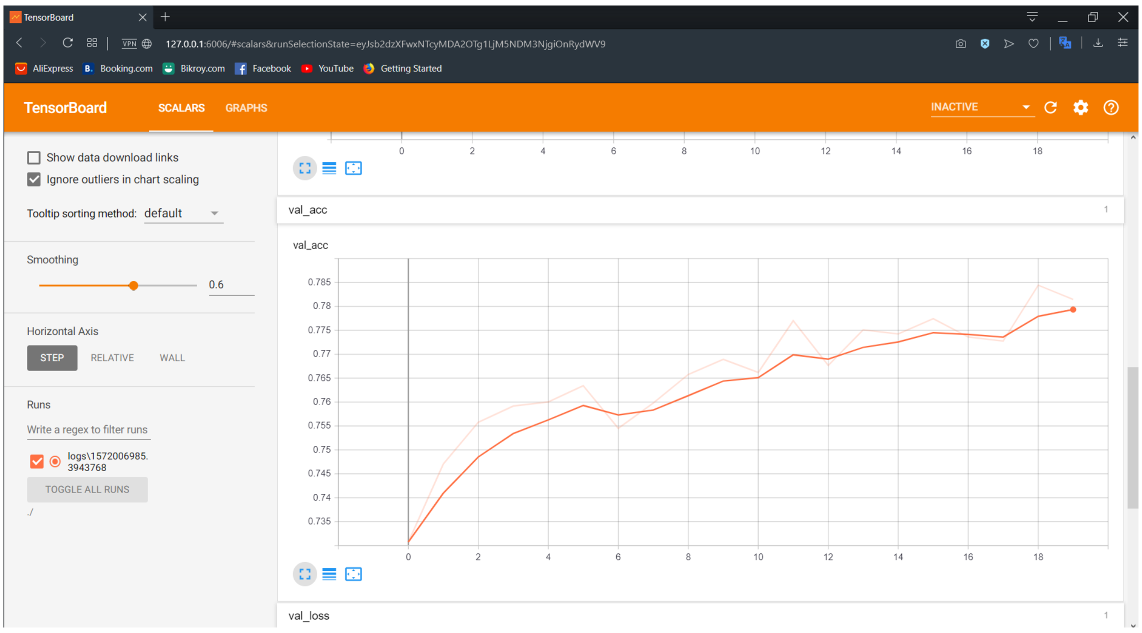
The table below shows the comparison between all the other models:

	AlexNet	ResNet50	VGG16	SVM	CNN(our model)
Training accuracy	72.90%	72.50%	68.20%	100%	82.6%
Validation Accuracy	68%	67.80%	64.80%	86%	80%

Table 4.1: Comparison between models

From the above table we see that SVM has the most validation accuracy. It is surprising how an ML model had performed better than the rest of the Neural Network models. This maybe be due to 'over-fitting' of the model after put into training taking the output tensor of the convolutional base of VGG16 into the model for feature extraction. VGG19 model also works the same way except that there are differences in layers. Since we have included the work of VGG19 in our workflow diagram, our implementation on this will be for future works.

Our own Neural Network model is defined as above and is trained for 100 epochs. Like mentioned earlier our model is first fitted using 'Relu' function in the first two 'Activation' layers and 'Sigmoid' function in the last 'Activation' layer and the Neural Network is trained. The same process is repeated using 'Square' function instead of 'Relu' and 'Softmax' instead of 'Sigmoid'. The graphs of the validation accuracy our own model using different set of functions twice is given below. The graphs were obtained from Tensorflow. Although the models with different functions are trained for different number of epochs, they are trained with the same dataset. Thus, there won't much of a difference in accuracy.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs: ☑ logs\1572006985.3943768

Figure 4.5: Validation Accuracy using 'Relu' and 'Sigmoid':

The model with the 'Square' and 'Softmax' activation functions have higher test or validation accuracy of 80% than the previous AlexNet and Resnet models when compared and also has more validation accuracy than that when the other two functions are used to build or own model.

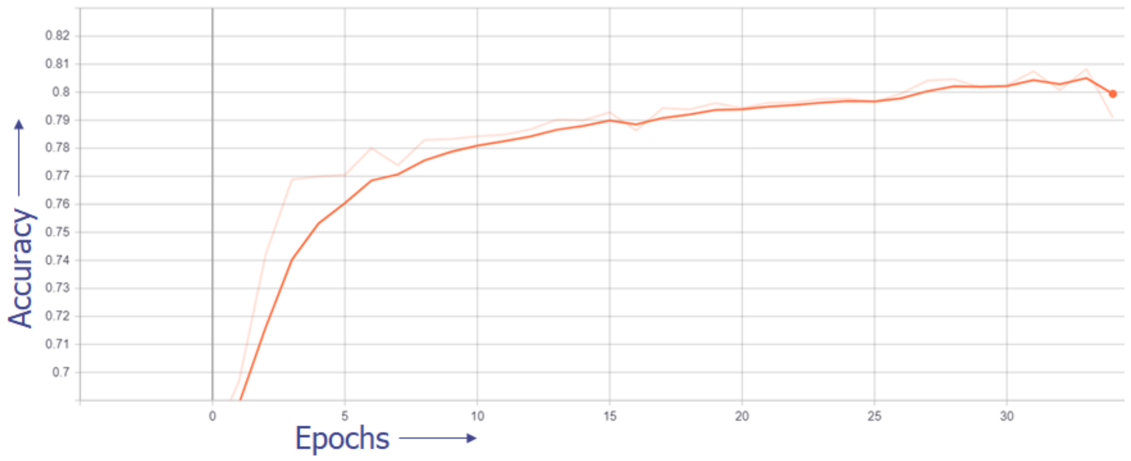


Figure 4.6: Validation Accuracy using ‘Square’ and ‘Softmax’

Chapter 5

Conclusion

In our work, we tried to compare different accuracies of the different neural network models in the same dataset and also making our own CNN model and all being under the light of an encryption technique called homomorphic encryption. We used the techniques from machine learning, cryptography and engineering to come up with our model which will not only give a high throughput of 80% (77.934% with our own encrypted NN model) but will also ensure proper security. For future works, we are collecting ALL- Acute Lymphoid Leukemia images with 'patient id', 'age', and 'gender'. For now, we have 290 images which is more than the ALL-IDB dataset which is frequently used in detection of blood cancer using ML and NN models.Previous works done on ALL detection used ALL-IDB dataset which has about 270 ALL blood cancer images. As of now, we are using the CNM-C dataset of our model which is significantly larger than the ALL-IDB dataset and has about 10000 training images of which we are using 3257 images for testing. Prior to building our own CNN model, we have used this dataset on various NN and ML models and achieved test accuracies of 68% (Alexnet), 67.8% (ResNet), 64.8% (VGG16) and 86% (SVM). We are hopeful to successfully collect about 2000 images, label it and run it on our own Crypto-Nets model for secure prediction of Cancer. Moreover it will provide a comparatively less expensive preliminary screening and will also ensure the proper privacy of the user. Our model can also be used in different sectors such as in other medical fields, finance and banking .

Bibliography

[1] S. Shafique and S. Tehsin, “Acute lymphoblastic leukemia detection and classification of its subtypes using pretrained deep convolutional neural networks”, *Technology in cancer research & treatment*, vol. 17, p. 1 533 033 818 802 789, 2018.

[2] A. Rehman, N. Abbas, T. Saba, S. I. u. Rahman, Z. Mehmood, and H. Kolivand, “Classification of acute lymphoblastic leukemia using deep learning”, *Microscopy Research and Technique*, vol. 81, no. 11, pp. 1310–1317, 2018.

[3] L. H. S. Vogado, R. D. M. S. Veras, A. R. Andrade, F. H. D. De Araujo, R. R. V. e Silva, and K. R. T. Aires, “Diagnosing leukemia in blood smear images using an ensemble of classifiers and pre-trained convolutional neural networks”, in *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, IEEE, 2017, pp. 367–373.

[4] S. Mohapatra, D. Patra, and S. Satpathy, “An ensemble classifier system for early diagnosis of acute lymphoblastic leukemia in blood microscopic images”, *Neural Computing and Applications*, vol. 24, no. 7-8, pp. 1887–1904, 2014.

[5] H. T. Madhloom, S. A. Kareem, and H. Ariffin, “A robust feature extraction and selection method for the recognition of lymphocytes versus acute lymphoblastic leukemia”, in *2012 international conference on advanced computer science applications and technologies (ACSAT)*, IEEE, 2012, pp. 330–335.

[6] L. Putzu and C. Di Ruberto, “White blood cells identification and counting from microscopic blood image”, in *Proceedings of World Academy of Science, Engineering and Technology*, World Academy of Science, Engineering and Technology (WASET), 2013, p. 363.

[7] T. Graepel, K. Lauter, and M. Naehrig, “Ml confidential: Machine learning on encrypted data”, in *International Conference on Information Security and Cryptology*, Springer, 2012, pp. 1–21.

[8] J. Z. Zhan, L. Chang, and S. Matwin, “Privacy preserving k-nearest neighbor classification.”, *IJ Network Security*, vol. 1, no. 1, pp. 46–51, 2005.

[9] Y. Qi and M. J. Atallah, “Efficient privacy-preserving k-nearest neighbor search”, in *2008 The 28th International Conference on Distributed Computing Systems*, IEEE, 2008, pp. 311–319.

[10] L. J. Aslett, P. M. Esperança, and C. C. Holmes, “Encrypted statistical machine learning: New privacy preserving methods”, *arXiv preprint arXiv:1508.06845*, 2015.

[11] ——, “A review of homomorphic encryption and software tools for encrypted statistical machine learning”, *arXiv preprint arXiv:1508.06574*, 2015.

[12] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption.", *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[13] M. Albrecht, S. Bai, and L. Ducas, "A subfield lattice attack on overstretched ntru assumptions", in *Annual International Cryptology Conference*, Springer, 2016, pp. 153–178.

[14] P. Kirchner and P.-A. Fouque, "Comparison between subfield and straightforward attacks on ntru.", *IACR Cryptology ePrint Archive*, vol. 2016, p. 717, 2016.

[15] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, "Crypto-nets: Neural networks over encrypted data", *arXiv preprint arXiv:1412.6181*, 2014.

[16] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices.", in *Stoc*, vol. 9, 2009, pp. 169–178.

[17] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages", in *Annual cryptology conference*, Springer, 2011, pp. 505–524.

[18] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping", *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[19] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Manual for using homomorphic encryption for bioinformatics", 2015.

[20] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy", in *International Conference on Machine Learning*, 2016, pp. 201–210.

[21] C.-H. Pui, D. Pei, S. C. Raimondi, E. Coustan-Smith, S. Jeha, C. Cheng, W. P. Bowman, J. T. Sandlund, R. C. Ribeiro, J. E. Rubnitz, *et al.*, "Clinical impact of minimal residual disease in children with different subtypes of acute lymphoblastic leukemia treated with response-adapted therapy", *Leukemia*, vol. 31, no. 2, p. 333, 2017.

[22] R. D. Labati, V. Piuri, and F. Scotti, "All-idb: The acute lymphoblastic leukemia image database for image processing", in *2011 18th IEEE International Conference on Image Processing*, IEEE, 2011, pp. 2045–2048.

[23] A. Gupta, R. Duggal, R. Gupta, L. Kumar, N. Thakkar, and D. Satpathy, "Gcti-sn: Geometry-inspired chemical and tissue invariant stain normalization of microscopic medical images", *Under review*,

[24] R. Gupta, P. Mallick, R. Duggal, A. Gupta, and O. Sharma, "Stain color normalization and segmentation of plasma cells in microscopic images as a prelude to development of computer assisted automated disease diagnostic tool in multiple myeloma", *Clinical Lymphoma, Myeloma and Leukemia*, vol. 17, no. 1, e99, 2017.

[25] R. Duggal, A. Gupta, R. Gupta, M. Wadhwa, and C. Ahuja, "Overlapping cell nuclei segmentation in microscopic images using deep belief networks", in *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, ACM, 2016, p. 82.

[26] R. Duggal, A. Gupta, and R. Gupta, “Segmentation of overlapping/touching white blood cell nuclei using artificial neural networks”, *CME Series on Hemato-Oncopathology, All India Institute of Medical Sciences (AIIMS). New Delhi, India*, 2016.

[27] R. Duggal, A. Gupta, R. Gupta, and P. Mallick, “Sd-layer: Stain deconvolutional layer for cnns in medical microscopic imaging”, in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2017, pp. 435–443.

[28] S. Chiaretti, G. Zini, and R. Bassan, “Diagnosis and subclassification of acute lymphoblastic leukemia”, *Mediterranean journal of hematology and infectious diseases*, vol. 6, no. 1, 2014.

[29] R. Livni, S. Shalev-Shwartz, and O. Shamir, “On the computational efficiency of training neural networks”, in *Advances in neural information processing systems*, 2014, pp. 855–863.